

ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ  
ФАКУЛЬТЕТ МАТЕМАТИКИ, МЕХАНИКИ И КОМПЬЮТЕРНЫХ НАУК  
КАФЕДРА МАТЕМАТИЧЕСКОГО АНАЛИЗА

Ю. А. Кирютенко  
**TikZ & PGF**  
Создание графики в  $\text{\LaTeX}2_{\epsilon}$ -документах

Ростов-на-Дону  
2014

Ю. А. Кирютенко.

TikZ & PGF. Создание графики в  $\text{\LaTeX}2_\epsilon$ -документах. — Ростов-на-Дону, 2014, 277 с.

Пособие к курсу «Информационные технологии: среда  $\text{\LaTeX}2_\epsilon$ ». Пособие основано на руководстве Тилла Тантау (Till Tantau) [1], собравшего разработанные до него части системы pgf и их документацию, и написавшего большой объем своего кода. Из руководства выбраны и описываются только те разделы, в которых объясняется, как использовать в  $\text{\LaTeX}2_\epsilon$  классы TikZ & PGF в среде MikTeX. В этом пособии не рассматриваются в деталях дополнительные библиотеки и утилиты, поставляемые с TikZ & PGF, не рассматривается почти все, что касается изменений и дополнений к классам и библиотекам TikZ & PGF. Последние разделы пособия посвящены поставляемым с PGF математическим классам, в разработку которых внесли большой вклад Марк Виброу (Mark Wibrow) и Христиан Фейерзангер (Christian Feuersänger. Все примеры рисунков (иногда слегка измененные) взяты из руководства Тилла Тантау (Till Tantau) [1]. Многие формы, механизм художественного оформления (декорации) и матрицы были написаны и зарегистрированы Марком Виброу.

Предполагается, что читатель знает среду  $\text{\LaTeX}$  (см. [3], [4], [5]) и умеет работать в ней. Чтобы работать с TikZ & PGF в среде MikTeX, в нее обычным образом нужно установить последние версии пакетов pgf и xcolor (их можно взять на сайте [www.ctan.org](http://www.ctan.org)).

# Глава 1


## Введение

Пакет `pgf` (portable graphics format — переносимый графический формат) является пакетом для создания встраиваемой в  $\TeX$ -документ графики и определяет множество  $\TeX$ -команд для ее создания. Чтобы иметь возможность воспользоваться этими командами, в преамбулу  $\LaTeX$ -документа нужно вставить строку

```
\usepackage{tikz}
```


Теперь можно «рисовать»! Например, нарисовать наклонную линию, позволяет код

```
\tikz \draw (0pt, 0pt) -- (1in, 8pt);
```



а нарисовать оранжевый кружок — код

```
\tikz \fill[orange] (1ex, 1ex) circle (1ex);
```



В некотором смысле, когда используется `pgf`, графика программируется так же, как программируется документ, когда используется  $\TeX$ . В этом случае сохраняются все преимущества  $\TeX$ -подхода к набору графики: быстрое создание простых графиков, точное позиционирование, использование макроопределений, очень высокое качество печати. Но наследуются и все недостатки  $\TeX$ 'а: код не дает представления о конечном результате, необходимо тратить время на изучение используемых пакетов, отсутствует наглядность, требуется полная повторная компиляция даже при минимальных изменениях.

Система `pgf` состоит из нескольких уровней:

**Системный уровень:** обеспечивает полную абстракцию того, что передается драйверу. Драйвер — программа, аналогичная `dvips` или `dvipdfm`, которая принимает `*.dvi` файл и производит `*.ps` или `*.pdf` файл (`pdftex` также рассматривается как драйвер, даже при том, что не требуется `*.dvi`-файла для ввода). В зависимости от используемого драйвера, системные команды преобразуются в различные специальные команды. Пользователь не должен непосредственно использовать системный уровень.

**Основной уровень:** обеспечивает набор основных команд, позволяющих легко строить сложную графику, значительно легче, чем при использовании системного уровня. Основной уровень состоит из *ядра*, которое содержит нескольких взаимозависимых пакетов, загружаемых в одном блоке, и *дополнительных модулей*, расширяющих ядро большим числом команд специального назначения. Например, пакет для создания презентаций `beamer` использует только ядро.

**Уровень внешнего интерфейса:** Внешний интерфейс (их может быть несколько) определяет набор команд и/или специальный синтаксис, которые упрощают использование основного уровня. Например, чтобы нарисовать простой треугольник, нужно провести три линии треугольника, расположенные под определенными углами друг к другу. При использовании внешнего интерфейса, например, *TikZ*, создание рисунка сводится к одной простой команде:

```
\draw (0,0) -- (1,0) -- (1,1) -- circle;
```

Существует несколько разных внешних интерфейсов для pgf:

- *TikZ* — позволяет обращаться ко всем особенностям pgf, и, одновременно, достаточно удобен. Синтаксис *TikZ* — смесь синтаксиса программ *metafont*, *pstricks* и некоторых идей Т. Тантау.
- *pgfpict2e* — переопределяет стандартное окружение `picture` и его команды, используя основной уровень pgf.

Пользователю pgf, основная задача которого — построение нужных ему рисунков в  $\TeX$ -документе, в основном достаточно команд внешнего интерфейса. Далее описывается внешний интерфейс *TikZ*.

Разумеется, pgf не единственный графический пакет для  $\TeX$ 'а. Приведем некоторое сравнение pgf с другими пакетами.

1. Стандартное окружение `picture`, позволяет создавать только простую графику. Это цена за мобильность: окружение `picture` работает со всеми драйверами.

2. Пакет *pstricks* достаточно мощен и позволяет создать любую мыслимую графику. По сравнению с pgf, *pstricks* имеет более широкую базовую поддержку, есть много вещей, которые можно сделать в *pstricks*, и невозможно сделать в pgf. В частности, *pstricks* имеет доступ к мощному языку программирования PostScript. Но *pstricks* не переносим и не работает с *pdftex*. Для *TikZ* создано много хороших дополнительных пакетов, ориентированных на часто встречающиеся конкретные ситуации. Синтаксис *TikZ* более последователен, чем синтаксис *pstricks*, поскольку создавался как единое целое.

3. Пакет *hupic* — один из старых пакетов для создания графики. Однако, он труден для использования и изучения, поскольку имеет достаточно загадочные синтаксис и документацию.

4. Пакет *dratex* — маленький графический пакет. По сравнению с другими пакетами, включая pgf, он очень маленький, и, возможно, это его единственное преимущество.

5. Программа *metapost* — очень мощная альтернатива pgf. Однако, это внешняя программа, что влечет за собой несколько проблем. Время, необходимое на создание маленького графика и его компиляции намного больше, чем в pgf. Главная проблема программы *metapost* — включение меток, что много легче сделать, используя pgf.

6. Программа *xfig* — важная альтернатива *TikZ* для тех пользователей, которые не желают программировать графику в манере *TikZ* и других пакетов, упомянутых выше. Существует программа преобразования *xfig*-графики и для *TikZ*, и для pgf, но программа все еще не закончена.

Пакет pgf поставляется с несколькими сервисными пакетами, которые можно использовать независимо от pgf. Однако, они связаны с pgf, частично ради удобства, частично потому, что их функциональные возможности переплетены с pgf:

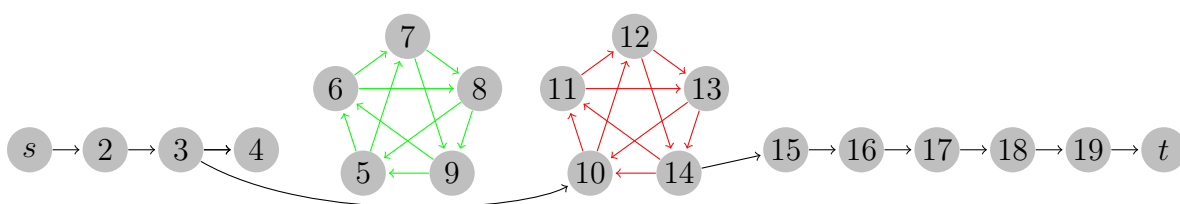
1) пакет `pgfkeys` позволяет управлять ключами. Может использоваться независимо от `pgf`;

2) пакет `pgffor` определяет полезное итерационное выражение `\foreach`;

3) пакет `pgfcalendar` определяет макросы для создания календарей и команды для работы с датами;

4) пакет `pgfpages` позволяет собрать несколько страниц в одну, определяя команды для сборки нескольких виртуальных страниц в единственную физическую страницу. Однако, `pgfpages` может делать много больше: его можно использовать для того, чтобы помещать на страницах эмблемы и водяные знаки, печатать до 16 страниц на одной странице, добавлять границы к страницам, и многое другое.

В пособии в основном описываются команды и окружения из интерфейса `TikZ` и из многочисленных подгружаемых библиотек. В каждом описании команды или окружения главный текст напечатан красным цветом, основной текст — черным, зеленый текст отражает необязательную часть (которая при использовании команды или окружения может отсутствовать).

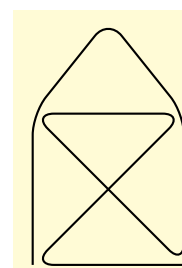


# Часть I

## Примеры и рекомендации

Начнем изучение pgf с уровня внешнего интерфейса TikZ: на примерах рассмотрим некоторые его возможности, пока не углубляясь в детали, дадим некоторые рекомендации по созданию рисунков, используя TikZ.

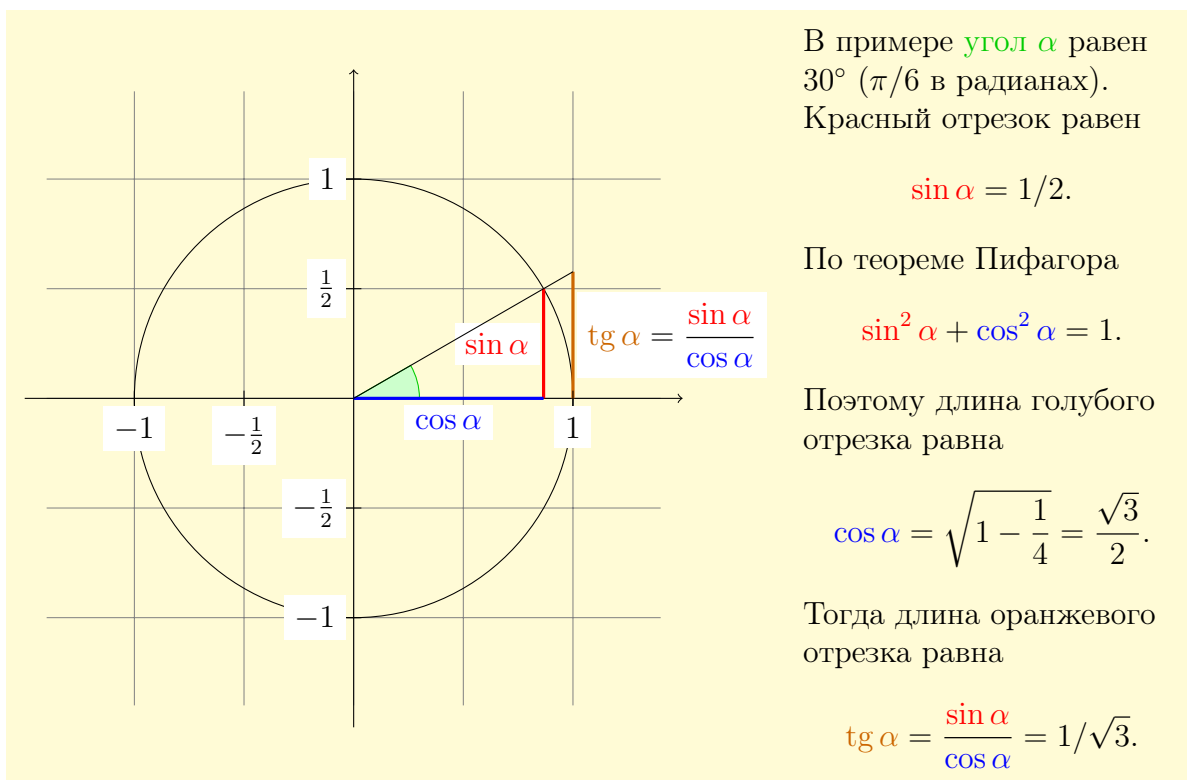
```
\tikz\draw[thick,rounded corners=8pt]
(0,0) -- (0,2) -- (1,3.25) -- (2,2) --
(2,0) -- (0,2) -- (2,2) -- (0,0) -- (2,0);
```



# Глава 2

## Определение синуса и косинуса

Начнем с простой задачи, ориентированной на тех пользователей, которые первый раз работают с пакетами `pgf` и `TikZ`. Эта глава не дает исчерпывающей информации о всех возможностях этих пакетов. Здесь рассмотрим только те особенности, которые, вероятно, будут встречаться достаточно часто. Простая задача этой главы — показать, как создать, используя `TikZ`, следующий простой рисунок.



### 2.1 Постановка задачи

Итак, создадим рисунок, помогающий понять школьнику, что такое синус и косинус угла. Для этого нарисуем систему координат, единичный круг, под некоторым углом проведем радиус и расставим поясняющие надписи.

Чтобы нарисовать картинку в `TikZ`, сначала надо сообщить об этом Л<sup>A</sup>T<sub>E</sub>X'у. Для этого используют окружение `{tikzpicture}`. Но, напомним еще раз, прежде нужно загрузить пакет `TikZ`, прописывая в преамбуле документа строку `\usepackage{tikz}`.

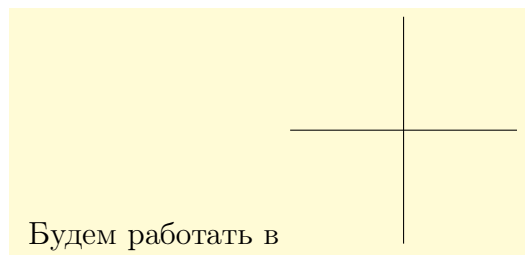
Сначала нарисуем оси координат:



```

Будем работать в
\begin{tikzpicture}
\draw (-1.5,0) -- (1.5,0);
\draw (0,-1.5) -- (0,1.5);
\end{tikzpicture}

```



В этом коде в окружении `{tikzpicture}` используются две команды `\draw`, каждая из которых завершается точкой с запятой. Первая команда рисует прямую от точки с координатами  $(-1.5, 0)$  до точки с координатами  $(1.5, 0)$ . Здесь, позиции определяются в пределах специальной системы координат, в который, изначально, одна единица — 1 см. Окружение автоматически резервирует достаточно места вокруг рисунка.

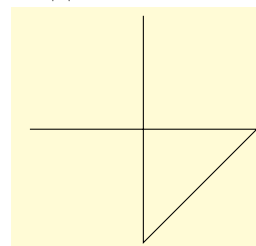
## 2.2 Конструирование прямого пути

Основной стандартный блок всех рисунков в TikZ — путь. Путь — ряд как-то связанных между собой прямых линий и кривых. Можно начать путь, определяя координаты точки начала, указывая координаты точки в круглых скобках, например,  $(0,0)$ . Точки сопровождаются «операциями расширения пути». Самая простая операция: `--` (она уже использовалась!). Эта операция должна сопровождаться другой точкой, формируя путь в виде прямой линии. Например, если нужно превратить две оси в предыдущем примере в один путь, можно было бы написать следующий код:

```

\tikz\draw (-1.5,0) -- (1.5,0)
-- (0,-1.5) -- (0,1.5);

```



Здесь нет окружения `{tikzpicture}`. Вместо него стоит команда `\tikz`, либо принимающая аргумент в фигурных скобках (как в коде `\tikz{\draw (0,0) -- (1.5,0);}`, рисуящем прямолинейный отрезок \_\_\_\_\_), либо собирающая все, что стоит до следующей точки с запятой и помещающая все это в окружение `{tikzpicture}`. На практике, все команды, создающие рисунок в TikZ, должны входить в качестве аргумента в команду `\tikz` или в окружение `{tikzpicture}`.

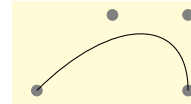
## 2.3 Конструирование кривой

Чтобы нарисовать круг, нужен некоторый способ, позволяющий рисовать кривые. Для этого TikZ определяет специальный синтаксис. Для начала нужны одна или две точки. Математика, используемая в этом случае, не тривиальна (кривые Безье), но основная идея проста: предположим, что отправная точка  $x$  и контрольная точка  $y$ . Пусть кривая строится в направлении точки  $y$  так, что касательная к кривой в точке  $x$  проходит через  $y$ . Пусть кривая должна закончиться в точке  $z$  и есть вторая точка поддержки  $w$ . Тогда кривая должна прийти в  $z$  так, чтобы касательная к кривой в точке  $z$  проходила через  $w$ . Приведем пример, в котором нарисованы (в виде небольших серых кружочков) начальная и конечная точки кривой и обе контрольные точки:

```

\begin{tikzpicture}
\filldraw [gray] (0,0) circle (2pt)
              (1,1) circle (2pt)
              (2,1) circle (2pt)
              (2,0) circle (2pt);
\draw (0,0) .. controls (1,1) and (2,1) .. (2,0);
\end{tikzpicture}

```



Общий синтаксис создания кривой следующий:

```

.. controls < first control point > and < second control point > .. < end point >

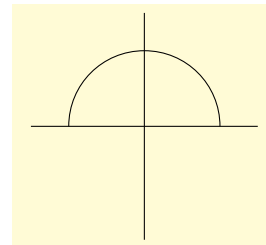
```

Можно опустить часть `and < second control point >`, тогда первая контрольная точка будет использоваться дважды. Теперь можно попробовать нарисовать полукруг:

```

\begin{tikzpicture}
\draw (-1.5,0) -- (1.5,0);
\draw (0,-1.5) -- (0,1.5);
\draw (-1,0) .. controls (-1,0.555)
               and (-0.555,1) .. (0,1)
              .. controls (0.555,1)
               and (1,0.555) .. (1,0);
\end{tikzpicture}

```



Такой способ выглядит чрезвычайно неуклюже. К счастью, есть и более простой.

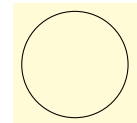
## 2.4 Конструирование окружности и круга

Чтобы нарисовать круг, следует использовать операцию `circle`, сопровождая ее радиусом в круглых скобках. Точка, предыдущая операции `circle` — центр круга.

```

\tikz\draw (0,0) circle (20pt);

```

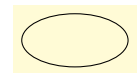


Точно также, используя операцию `ellipse` и, вместо единственного радиуса, определяя длины двух полуосей ( $x$  и  $y$ ), можно нарисовать эллипс:

```

\tikz\draw (0,0) ellipse (20pt and 10pt);

```

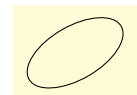


Чтобы нарисовать эллипс, оси которого не являются горизонтальными и вертикальными, можно использовать преобразование поворота (оно рассматривается позже).

```

\tikz\draw [rotate=30] (0,0) ellipse (20pt and 10pt);

```

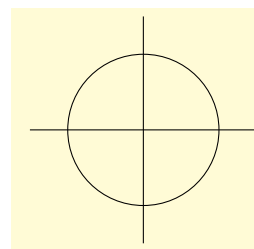


Таким образом, чтобы нарисовать оси и круг радиуса 1cm, следует написать:

```

\begin{tikzpicture}
  \draw (-1.5,0) -- (1.5,0);
  \draw (0,-1.5) -- (0,1.5);
  \draw (0,0) circle (1cm);
\end{tikzpicture}

```



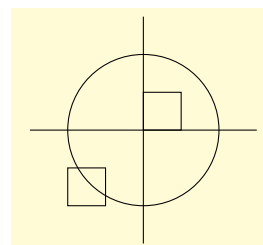
## 2.5 Конструирование прямоугольника

Следующий объект — прямоугольная сетка. Ее можно нарисовать несколькими способами, например, нарисовать много прямоугольников. Их рисовать не сложно, для этого есть операция `rectangle`. Но чтобы нарисовать сетку, следует прямоугольники правильно расположить: угол созданного прямоугольника — начальная точка для следующего. Добавим два прямоугольника в ранее созданный рисунок:

```

\begin{tikzpicture}
\draw (-1.5,0) -- (1.5,0);
\draw (0,-1.5) -- (0,1.5);
\draw (0,0) circle (1cm);
\draw (0,0) rectangle (0.5,0.5);
\draw (-0.5,-0.5) rectangle (-1,-1);
\end{tikzpicture}

```



Но чтобы нарисовать сетку на плоскости, такой подход не годится: нужно нарисовать много прямоугольников, да еще есть прямоугольники с неполной границей.

## 2.6 Конструирование сетки

Сетку добавляет в текущий путь операция `grid`, которая рисует прямые линии, составляющие сетку, заполняя ими прямоугольник, один угол которого — текущая точка, а другой — точка после `grid`.

```

\tikz \draw[step=5pt] (0,0) grid (20pt,20pt);

```

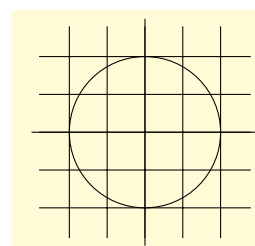


Дополнительный параметр `step` команды `\draw` определяет шаг сетки (есть также параметры `xstep` и `ystep`, которые определяются независимо друг от друга). Таким образом, чтобы нарисовать сетку, можно использовать следующий код:

```

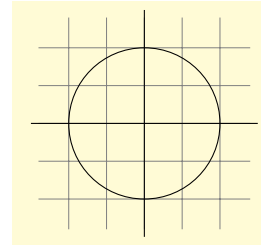
\begin{tikzpicture}
\draw (-1.5,0) -- (1.5,0);
\draw (0,-1.5) -- (0,1.5);
\draw (0,0) circle (1cm);
\draw[step=.5cm] (-1.4,-1.4) grid (1.4,1.4);
\end{tikzpicture}

```



Но сетка должна быть менее заметной. Чтобы добиться этого, добавим к команде `\draw` еще две опции: они сделают сетку серой (`gray`), а линии сетки очень тонкими (`very thin`). Еще изменим порядок команд так, чтобы сетка рисовалась первой.

```
\begin{tikzpicture}
\draw[step=.5cm,gray,very thin]
  (-1.4,-1.4) grid (1.4,1.4);
\draw (-1.5,0) -- (1.5,0);
\draw (0,-1.5) -- (0,1.5);
\draw (0,0) circle (1cm);
\end{tikzpicture}
```



## 2.7 Понятие о стиле

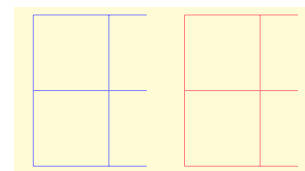
Вместо опций `gray`, `very thin`, можно указать опцию `help lines` (вспомогательные линии). Это предопределенный *стиль*. Стили позволяют упростить операцию создания рисунка. Указывая стиль `help lines`, который был (кем-то ранее) определен, можно сократить код. Например, пусть сетки должны рисоваться в виде тонких синих линий (`blue!50`), тогда сначала можно определить желаемый стиль представления сетки: `help lines/.style={color=blue!50,very thin}`. Эффект от установки такого стиля, тот же, что и от опций `color=blue!50,very thin`.

Используя стили, можно сделать процесс создания графики более гибким. Обычно, стили определяются в начале рисунка. Но можно определить стиль глобально, чтобы его использовали все рисунки документа. Для этого в начале документа следует использовать команду `\tikzset`. Например, `\tikzset{help lines/.style=very thin}`.

Можно строить иерархию стилей, используя один стиль в другом. Так, например, чтобы определить использованный ранее стиль для отображения сетки, можно сказать `\tikzset{My grid/.style={help lines,color=blue!50}}`.

Стили можно сделать еще более мощными через параметризацию. Это означает, что, как и другие опции, стили могут использоваться с параметром. Например, можно параметризовать стиль для отображения сетки так, чтобы, по умолчанию она была синей, но можно было бы использовать и другой цвет.

```
\begin{tikzpicture}
[My grid/.style={help lines,color=#1!50},
My grid/.default=blue]
\draw[My grid] (0,0) grid (1.5,2);
\draw[My grid=red] (2,0) grid (3.5,2);
\end{tikzpicture}
```



## 2.8 Опции рисунка

Линии можно рисовать тонкими (`thin`), очень тонкими (`very thin`), супер тонкими (`ultra thin`), толстыми (`thick`), очень толстыми (`very thick`), супер толстыми (`ultra thick`). Нормальная толщина линий — `thin`. Есть еще толщина посередине между тонкими и толстыми: `semithick` (полутолстый).

Цвет линии можно задать опцией `color=<color>` или, что почти то же самое, опцией `\draw=<color>`. Во-втором случае цвет устанавливается только для линий, и можно задать другой цвет для заполнения областей (например, заполнить цветом круговой сектор, соответствующий углу  $\alpha$  на рисунке).

Другая полезная опция позволяет рисовать линии штриховыми `dashed` или точечными `dotted`. Обе опции существуют в свободной и плотной версии — `loosely dashed`, `densely dashed`, `loosely dotted`, `densely dotted`.

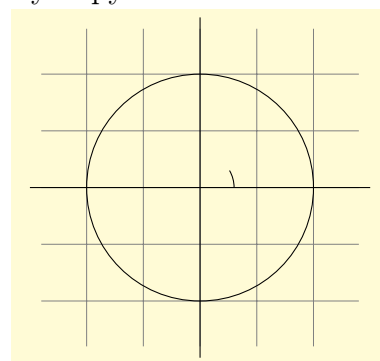
## 2.9 Конструирование дуги

Наша следующая задача — научиться рисовать дугу кривой, в частности, часть круга или эллипса. Для этого используется операция `arc`, которая должна сопровождаться тройкой чисел в круглых скобках, компоненты которой разделяются двоеточиями. Первые два компонента — углы в градусах, последний — радиус. Например, тройка `(10:80:10pt)` означает дугу от  $10^\circ$  до  $80^\circ$  в круге радиуса `10pt`. Центр круга будет размещаться в текущей точке. В нашем случае, чтобы построить дугу от  $0^\circ$  до  $30^\circ$  с радиусом равным приблизительно одной трети от радиуса круга, надо написать: `arc(0:30:3mm)`.

В случае эллипса построение происходит аналогично (только задаются «два радиуса»): `\tikz \draw (0,0) arc (0:315:1.25cm and 0.5cm);`.

Чтобы сделать рисунок больше, его можно масштабировать: добавить опцию вида `[scale=2]` к каждой команде `\draw`, но лучше сразу ко всему окружению.

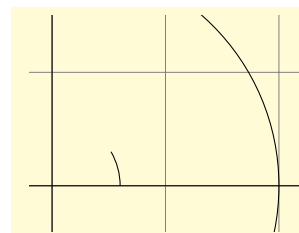
```
\begin{tikzpicture} [scale=1.5]
\draw[step=.5cm,gray,very thin]
(-1.4,-1.4) grid (1.4,1.4);
\draw (-1.5,0) -- (1.5,0);
\draw (0,-1.5) -- (0,1.5);
\draw (0,0) circle (1cm);
\draw (3mm,0mm) arc (0:30:3mm);
\end{tikzpicture}
```



## 2.10 Отсечение

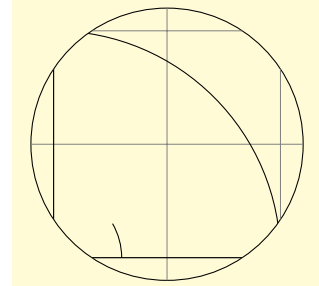
Чтобы сохранить место или выделить главное, иногда следует показать только часть рисунка. Этого можно добиться отсечением, используя команду `\clip`.

```
\begin{tikzpicture}[scale=3]
\clip (-0.1,-0.2) rectangle (1.1,0.75);
\draw[step=.5cm,gray,very thin]
(-1.4,-1.4) grid (1.4,1.4);
\draw (-1.5,0) -- (1.5,0);
\draw (0,-1.5) -- (0,1.5);
\draw (0,0) circle (1cm);
\draw (3mm,0mm) arc (0:30:3mm);
\end{tikzpicture}
```



Можно одновременно и рисовать, и отсекают. Для этого, следует использовать команду `\draw` с опцией `clip` (подобное не применимо ко всему рисунку), а можно использовать команду `\clip` с опцией `draw` (это также не применимо ко всему рисунку). В действительности, `\draw` — сокращение для `\path[draw]`, а `\clip` — сокращение для `\path[clip]`, поэтому можно использовать команду `\path[draw,clip]`.

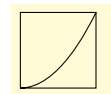
```
\begin{tikzpicture}[scale=3]
\clip[draw] (0.5,0.5) circle (.6cm);
\draw[step=.5cm,gray,very thin]
(-1.4,-1.4) grid (1.4,1.4);
\draw (-1.5,0) -- (1.5,0);
\draw (0,-1.5) -- (0,1.5);
\draw (0,0) circle (1cm);
\draw (3mm,0mm) arc (0:30:3mm);
\end{tikzpicture}
```



## 2.11 Конструирование параболы и синусоиды

Чтобы нарисовать параболу и синусоиду в TikZ есть операции `parabola`, `sin`, `cos`. При выполнении операции `parabola` на параболу будет лежать текущая точка и точка, указанная после операции. Рассмотрим следующий пример.

```
\tikz \draw (0,0) rectangle (1,1) (0,0) parabola (1,1);
```



Можно разместить дугу параболы где-то в другом месте:

```
\tikz \draw[x=1pt,y=1pt] (0,0) parabola bend (5,20) (10,10);
```

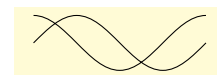


Операции `sin` и `cos` рисуют синусоиду и косинусоиду в интервале  $[0, \pi/2]$  так, что текущая точка — начало кривой, следующая за операцией точка — конец кривой.

```
\tikz \draw[x=1ex,y=1ex] (0,0) sin (1.57,1);
```



```
\tikz \draw [x=1.57ex,y=1ex] (0,0) sin (1,1)
cos (2,0) sin (3,-1) cos (4,0) (0,1) cos
(1,0) sin (2,-1) cos (3,0) sin (4,1);
```



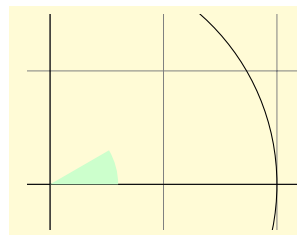
## 2.12 Области и границы

Выделенный ранее на рисунке круговой сектор с углом в  $30^\circ$  заполним цветом, скажем, светло-зеленым. Для этого вместо команды `\draw` используем команду `\fill`.

```

\begin{tikzpicture}[scale=3]
\clip (-0.1,-0.2) rectangle (1.1,0.75);
\draw[step=.5cm,gray,very thin]
(-1.4,-1.4) grid (1.4,1.4);
\draw (-1.5,0) -- (1.5,0);
\draw (0,-1.5) -- (0,1.5);
\draw (0,0) circle (1cm);
\fill[green!20!white](0,0) -- (3mm,0mm) arc (0:30:3mm) -- (0,0);
\end{tikzpicture}

```



Цвет `green!20!white` означает, что смешивается 20%-зеленого цвета и 80% белого (такое возможно, так как `pgf` использует пакет `xcolor`). Что было бы, если бы путь оказался не замкнутым (не было бы `-- (0,0)` в конце)? В этом случае путь замкнулся бы автоматически, поэтому код `-- (0,0)` можно было опустить. Но лучше написать так: `\fill [green!20!white] (0,0) -- (3mm,0mm) arc (0:30:3mm) -- cycle;`. Завершающий код `-- cycle` заставляет текущий путь замкнуться, гладко соединяя первую и последнюю точки. Но при этом меняется характер соединения. Например,

```

\begin{tikzpicture}[line width=5pt]
\draw (0,0) -- (1,0) -- (1,1) -- (0,0);
\draw (2,0) -- (3,0) -- (3,1) -- cycle;
\end{tikzpicture}

```

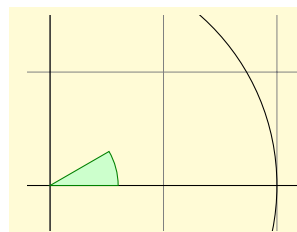


Чтобы и заполнить область цветом, и нарисовать ее границу, нужно использовать команду `\filldraw`, устанавливая разный цвет для границы и внутренности области.

```

\begin{tikzpicture}[scale=3]
\clip (-0.1,-0.2) rectangle (1.1,0.75);
\draw[step=.5cm,gray,very thin]
(-1.4,-1.4) grid (1.4,1.4);
\draw (-1.5,0) -- (1.5,0);
\draw (0,-1.5) -- (0,1.5);
\draw (0,0) circle (1cm);
\filldraw[fill=green!20!white,draw=green!50!black]
(0,0) -- (3mm,0mm) arc (0:30:3mm) -- cycle;
\end{tikzpicture}

```



## 2.13 Растушевывание

Вместо выделения угла светло-зеленым цветом (однородным) цветом, можно создать гладкий переход от одного цвета к другому. Для этого используются команды `\shade` и `\shadedraw`: `\tikz \shade (0,0) rectangle (2,1) (3,0.5) circle (.5cm);`



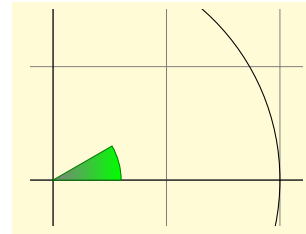
По умолчанию операция растушевывания реализует гладкий переход от серого к белому. Используя опции, можно определить другие цвета:

```
\begin{tikzpicture}[rounded corners,ultra thick]
\shade[top color=yellow,bottom color=black] (0,0) rectangle +(2,1);
\shade[left color=yellow,right color=black] (3,0) rectangle +(2,1);
\shadedraw[inner color=yellow,outer color=black,draw=yellow]
(6,0) rectangle +(2,1);
\shade[ball color=green] (9,.5) circle (.5cm);
\end{tikzpicture}
```



Для создаваемого рисунка выделение угла растушевыванием можно реализовать так:

```
\begin{tikzpicture}[scale=3]
\clip (-0.1,-0.2) rectangle (1.1,0.75);
\draw[step=.5cm,gray,very thin]
(-1.4,-1.4) grid (1.4,1.4);
\draw (-1.5,0) -- (1.5,0);
\draw (0,-1.5) -- (0,1.5);
\draw (0,0) circle (1cm);
\shadedraw[left color=gray,right color=green,draw=green!50!black]
(0,0) -- (3mm,0mm) arc (0:30:3mm) -- cycle;
\end{tikzpicture}
```



Стоит заметить, что растушевывание, обычно, только отвлекает внимание, и потому его следует использовать в рисунках крайне осторожно.

## 2.14 Определение координат

Есть ли способы определять координаты точек на рисунке? Самый простой путь состоит в том, чтобы сказать нечто подобное  $(10pt, 2cm)$ , то есть  $x = 10pt$ ,  $y = 2cm$ . Можно опустить единицы измерения и написать, например,  $(1,2)$ , что означает «единица умножить на текущий  $x$ -вектор плюс 2 умножить на текущий  $y$ -вектор». Значение размерности по умолчанию —  $1cm$ .

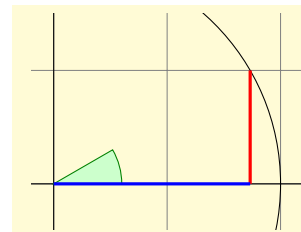
Чтобы определить точку в полярных координатах, используется запись  $(30:1cm)$ , что означает «1 cm в направлении  $30^\circ$ ». Очевидно, так можно определить на круге точку с координатами  $(\cos 30^\circ, \sin 30^\circ)$ .

Можно поставить один или два знака  $+$  перед точкой, например,  $+(1cm, 0cm)$  или  $++(0cm, 2cm)$ . Такие координаты интерпретируются по-разному. Первая форма означает: на 1cm вверх от предыдущей позиции. Вторая форма означает: на 2cm вправо от предыдущей позиции, делая полученную точку новой специальной позицией. Например, код `\draw[red,very thick] (30:1cm) -- +(0,-0.5);` нарисует красный отрезок синуса. Здесь используется тот факт, что  $\sin 30^\circ = 1/2$ . Код  $(30:1cm)$  — в полярной системе координат указывает на точку пересечения окружности радиуса 1cm с радиусом, проведенным под углом в  $30^\circ$  к оси  $OX$ . Код  $+(0,-0.5)$  сдвигает точку вниз от указанной



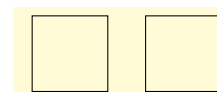
точки до оси  $OX$ . Но можно использовать специальный синтаксис:  $(30:1\text{cm} \mid - 0,0)$ . Запись вида  $(\langle p \rangle \mid - \langle q \rangle)$  означает: пересечение вертикальной линии, проходящей через  $p$ , и горизонтальной линии, проходящей через  $q$ . Далее, чтобы нарисовать отрезок косинуса, можно было написать  $(30:1\text{cm} \mid - 0,0) -- (0,0)$ . Другой способ начертить линию косинуса: начать рисовать отрезок из той точки, где заканчивается линия синуса:

```
\begin{tikzpicture}[scale=3]
\clip (-0.1,-0.2) rectangle (1.1,0.75);
\draw[step=.5cm,gray,very thin]
(-1.4,-1.4) grid (1.4,1.4);
\draw (-1.5,0) -- (1.5,0);
\draw (0,-1.5) -- (0,1.5);
\draw (0,0) circle (1cm);
\filldraw[fill=green!20,draw=green!50!black]
(0,0) -- (3mm,0mm) arc (0:30:3mm) -- cycle;
\draw[red,very thick] (30:1cm) -- +(0,-0.5);
\draw[blue,very thick](30:1cm) ++(0,-0.5) -- (0,0);
\end{tikzpicture}
```

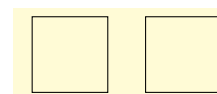


Отметим, что нет  $--$  между  $(30:1\text{cm})$  и  $++(0,-0.5)$ . Такой путь интерпретируется так: сначала код  $(30:1\text{cm})$  перемещает перо в точку  $(\cos 30^\circ, 1/2)$ ; затем, не рисуя, перемещает перо вниз на  $0.5\text{cm}$ ; и, наконец, перемещает перо в точку  $(0,0)$ , на сей раз, рисуя линию (присутствует  $--$ ). Чтобы увидеть различие между  $+$  и  $++$  рассмотрим следующие примеры (когда используется один  $+$ , нужны другие координаты).

```
\begin{tikzpicture}
\def\rectanglepath{-- ++(1cm,0cm) --
++(0cm,1cm) -- ++(-1cm,0cm) -- cycle}
\draw (0,0)\rectanglepath; \draw (1.5,0)\rectanglepath;
\end{tikzpicture}
```

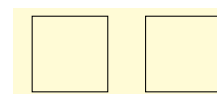


```
\begin{tikzpicture}
\def\rectanglepath{-- +(1cm,0cm) --
+(1cm,1cm) -- +(0cm,1cm) -- cycle}
\draw (0,0)\rectanglepath; \draw (1.5,0)\rectanglepath;
\end{tikzpicture}
```



Естественно, все это можно написать более ясно и более экономно:

```
\tikz \draw (0,0) rectangle +(1,1)
(1.5,0) rectangle +(1,1);
```



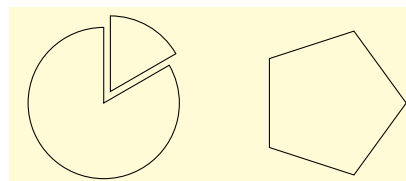
Остается нарисовать линию тангенса. Для этого нужен другой способ определения координат: можно определить точку, как точку пересечения прямых. Линия тангенса начинается в точке  $(1,0)$ , и идет вверх до точки пересечения с прямой, идущей из начала координат через точку  $(30:1\text{cm})$ . Синтаксис для определения такой точки следующий

(чтобы этой возможностью пользоваться надо в преамбуле документа подключить библиотеку `intersections`):

```
\draw[very thick,orange] (1,0) --
      (intersection of 1,0--1,1 and 0,0--30:1cm);
```

Приведем еще два примера того, как использовать относительное позиционирование. Обратим внимание, что опции преобразования (они рассматриваются позже) часто более полезны, чем относительное позиционирование (просто сдвиг части рисунка).

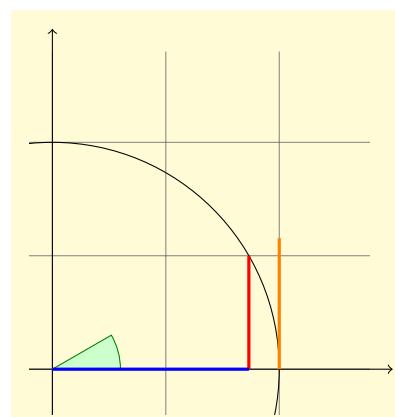
```
\begin{tikzpicture}
\draw (0,0) -- (90:1cm) arc (90:360:1cm)
      arc (0:30:1cm) -- cycle;
\draw (60:5pt) -- +(30:1cm) arc (30:90:1cm)
      -- cycle;
\draw (3,0) +(0:1cm) -- +(72:1cm) --
      +(144:1cm) -- +(216:1cm) --
      +(288:1cm) -- cycle;
\end{tikzpicture}
```



## 2.15 Построение стрелок

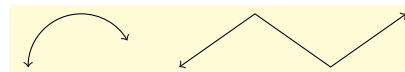
Чтобы добавить небольшие стрелки в конце осей координат, в команде `\draw` нужно использовать специальные опции: `->` (стрелка в конце пути), `<-` (стрелка в начале пути), `<->` (стрелка и в конце, и в начале пути).

```
\begin{tikzpicture}[scale=3]
\clip (-0.1,-0.2) rectangle (1.52,1.55);
\draw[step=.5cm,gray,very thin]
      (-1.4,-1.4) grid (1.4,1.4);
\draw[->] (-1.5,0) -- (1.5,0);
\draw[->] (0,-1.5) -- (0,1.5);
\draw (0,0) circle (1cm);
\filldraw[fill=green!20,draw=green!50!black]
      (0,0) -- (3mm,0mm) arc (0:30:3mm) -- cycle;
\draw[red,very thick] (30:1cm) -- +(0,-0.5);
\draw[blue,very thick]
      (30:1cm) ++(0,-0.5) -- (0,0);
\draw[orange,very thick] (1,0) --
      (intersection of 1,0--1,1 and 0,0--30:1cm);
\end{tikzpicture}
```



Есть ограничения на вид путей, к которым можно добавлять стрелки. Эмпирическое правило таково: можно добавить стрелки только к открытой линии. Например, в настоящее время нельзя добавить стрелки, скажем, к прямоугольнику или кругу (но в будущих версиях все может измениться!). Однако, можно добавлять стрелки и к кривым, и к ломаным, состоящим из нескольких частей, как в следующих примерах:

```
\begin{tikzpicture}[scale=2]
\draw [<->] (0,0) arc (180:30:10pt);
\draw [<->] (1,0) -- (1.5cm,10pt) --
(2cm,0pt) -- (2.5cm,10pt);
\end{tikzpicture}
```



Форма такой стрелки — форма стрелки в  $\TeX$ 'е:  $A \rightarrow B$ . Можно использовать и некоторые другие виды стрелок: ставить несколько знаков  $>$  или  $<$ ; использовать формат  $>=\langle\text{right arrow tip kind}\rangle$ , где  $\langle\text{right arrow tip kind}\rangle$  — спецификация специального типа стрелки. Например,  $>=\text{stealth}$ :

```
\begin{tikzpicture}[scale=2,>=stealth]
\draw[->] (0,0) arc (180:30:10pt);
\draw[<<- ,very thick] (1,0) -- (1.5cm,10pt)
-- (2cm,0pt) -- (2.5cm,10pt);
\end{tikzpicture}
```



Все predefined виды стрелок определены в библиотеке `arrows` и перечислены в оригинальном руководстве [1, глава 23]. Чтобы пользоваться этой библиотекой, в преамбуле документа нужно ее загрузить обычным образом: `\usetikzlibrary{arrows}`. Можно определять и собственные типы стрелок.

## 2.16 Окружение {scope}

Есть много графических опций, связанных с прорисовкой линий. Часто некоторые из них нужно применить к нескольким графическим командам. Например, часть линий нарисовать, используя толстое перо, а все остальные рисовать обычным образом.

Если нужно установить некоторую графическую опцию для целого рисунка, можно просто передать ее команде `\tikz` или окружению `{tikzpicture}`. Однако, если нужно применить опцию к локальной группе, эти команды следует поместить в окружение `{scope}`, передавая ему графические опции в качестве необязательного параметра. Такие опции будут применяться только к командам внутри окружения. Например:

```
\begin{tikzpicture}[ultra thick]
\draw (0,0) -- (0,1);
\begin{scope}[thin]
\draw (1,0) -- (1,1); \draw (2,0) -- (2,1);
\end{scope}
\draw (3,0) -- (3,1);
\end{tikzpicture}
```



Окружение `{scope}` имеет и другой интересный эффект: в нем любые изменения в области отсечения являются локальными. Таким образом, если сказать `\clip` где-нибудь внутри окружения `{scope}`, эффект от команды `\clip` не выйдет за границы окружения. Это бывает полезно, когда нет другого пути увеличения области отсечения.

Важно отметить, что опции к командам, подобным `\draw`, в действительности не являются опциями к таким командам, а являются опциями к рисуемому пути и могут

задаться на протяжении всего пути. Итак, вместо `\draw[thin] (0,0) -- (1,0);` можно написать `\draw (0,0) [thin] -- (1,0);` или `\draw (0,0) -- (1,0) [thin];`. Все они делают одно и то же. Это выглядит несколько странно, так как в последнем случае, казалось бы, опция `[thin]` должна действовать уже только после того, как линия от точки  $(0,0)$  до точки  $(1,0)$  будет нарисована. Однако, **большинство** графических опций применяются ко всему пути в целом после того, как путь создан, но до того, как он нарисован. Когда в пути указываются две однотипные опции, например, `thin` и `thick`, то «побеждает» (реализуется) последняя.

Заметим, что **не все** графические опции обращаются к целому пути. Все опции преобразований воздействуют не на весь путь, а только на ту часть пути, которая располагается после них. Чуть позже это будет рассматриваться более детально. Но все опции, заданные при конструировании пути, применяется только к этому пути.

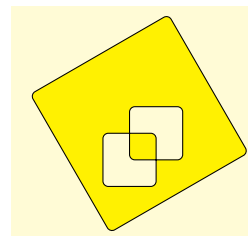
## 2.17 Преобразования

Когда определяется точка, такая как  $(1\text{cm}, 1\text{cm})$ , в какое место страницы она помещается? Чтобы определить это место, TikZ, TeX, PDF или PostScript применяют определенные преобразования к заданной точке. TikZ определяет набор опций, которые позволяют преобразовывать точки в системе координат pgf. Например, опция `xshift` позволяет сдвигать все последующие точки на определенное расстояние:

```
\tikz \draw (0,0) -- (0,0.5) [xshift=2pt] (0,0) -- (0,0.5);
```

Важно отметить, что можно применить преобразование в середине пути. Такую особенность не поддерживают pdf и PostScript. Причина в том, что pgf сохраняет информацию о своей матрице преобразования и может ее изменять «на лету». Вот более сложный пример:

```
\begin{tikzpicture}[even odd rule,
  rounded corners=2pt,x=10pt,y=10pt,scale=2]
\filldraw[fill=yellow] (0,0) rectangle (1,1)
[xshift=5pt,yshift=5pt] (0,0) rectangle (1,1)
[rotate=30] (-1,-1) rectangle (2,2);
\end{tikzpicture}
```



Перечислим самые полезные преобразования.

- `xshift` или `yshift` — производит сдиг в  $x$ -направлении или  $y$ -направлении;
- `shift` — производит сдиг в заданную точку: `shift={(1,0)}` или `shift={+(0,0)}` (фигурные скобки нужны, чтобы TeX не подумал, что запятая разделяет опции);
- `rotate` — производит вращение на определенный угол (есть также `rotate around` для вращения вокруг заданной точки);
- `scale` — производит масштабирование в определенное число раз;
- `xscale` и `yscale` — производит масштабирование только в  $x$ -направлении или  $y$ -направлении (`xscale=-1` — зеркальное отображение);
- `xslant` и `yslant` — производит наклонение.

Если этих преобразований, и даже тех, которые не были упомянуты, не достаточно, опция `cm` позволяет применить произвольную матрицу преобразования (об этом позже).

## 2.18 Итерации и циклы

Следующая задача: добавить небольшие метки на оси координат в точках  $-1$ ,  $-1/2$ ,  $1/2$ ,  $1$ . Хорошо было бы использовать некоторый цикл, делающий одно и то же в каждой из точек. Для решения такой задачи разные пакеты используют разные команды. Pgf вводит команду `\foreach` (автор команды Тилл Тангау), которая, в действительности определяется в пакете `pgffor` и может использоваться независимо от `pgf`. TikZ включает `\foreach` автоматически. В ее основной форме команда весьма удобна:

```
\foreach \x in {1,2,3} {$x =\x$, }
```

$x = 1, x = 2, x = 3,$

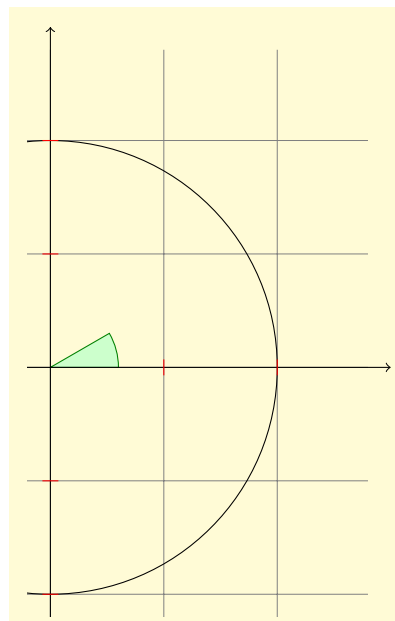
Общий синтаксис команды `\foreach` такой:

```
\foreach <variable> in {<list of values>} <commands>
```

В каждой команде из `<commands>`, переменной `<variable>` последовательно назначаются значения из списка `<list of values>`. Если опция `<commands>` не начинается со скобки, все до следующей точки с запятой используется как `<commands>`.

Чтобы поставить красные метки на оси координат, используем следующий код:

```
\begin{tikzpicture}[scale=3]
\clip (-0.1,-0.2) rectangle (1.52,1.55);
\draw[step=.5cm,gray,very thin]
(-1.4,-1.4) grid (1.4,1.4);
\filldraw[fill=green!20,draw=green!50!black]
(0,0) -- (3mm,0mm) arc (0:30:3mm) -- cycle;
\draw[->] (-1.5,0) -- (1.5,0);
\draw[->] (0,-1.5) -- (0,1.5);
\draw (0,0) circle (1cm);
\foreach \x in {-1cm, -0.5cm, 0.5cm, 1cm}
\draw[red] (\x,-1pt) -- (\x,1pt);
\foreach \y in {-1cm, -0.5cm, 0.5cm, 1cm}
\draw[red] (-1pt,\y) -- (1pt,\y);
\end{tikzpicture}
```



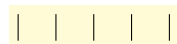
Есть много способов создания меток. Например, можно поместить `\draw...` в фигурные скобки, можно использовать операции сдвига. Если диапазон значений переменной достаточно большой, то не нужно все их перечислять, достаточно поставить многоточие, как следующем примере:

```
\tikz \foreach \x in {1,...,10} \draw (\x,0) circle (0.4cm);
```



Если указать два числа до многоточия `...`, то команда `\foreach` будет использовать их разность как шаг цикла:

```
\tikz \foreach \x in {-1,-0.5,...,1}
      \draw (\x cm,-5pt) -- (\x cm,5pt);
```



Используя вложенные циклы, можно получать интересные результаты:

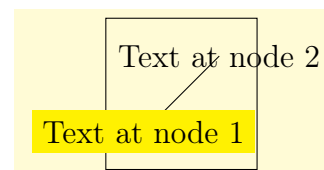
```
\begin{tikzpicture}
\foreach \x in {1,2,...,5,7,8,...,12}
\foreach \y in {1,...,5}{
  \draw (\x,\y) +(-.5,-.5) rectangle ++(.5,.5);
  \draw (\x,\y) node{\x,\y}; }
\end{tikzpicture}
```

1,5	2,5	3,5	4,5	5,5	7,5	8,5	9,5	10,5	11,5	12,5
1,4	2,4	3,4	4,4	5,4	7,4	8,4	9,4	10,4	11,4	12,4
1,3	2,3	3,3	4,3	5,3	7,3	8,3	9,3	10,3	11,3	12,3
1,2	2,2	3,2	4,2	5,2	7,2	8,2	9,2	10,2	11,2	12,2
1,1	2,1	3,1	4,1	5,1	7,1	8,1	9,1	10,1	11,1	12,1

## 2.19 Добавление текста в рисунок

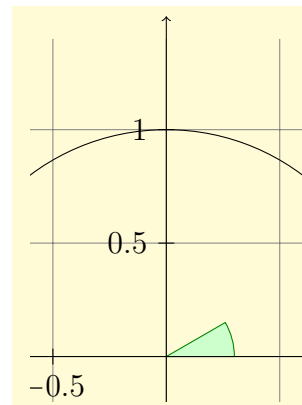
Поставив на осях координат метки, указывающие конкретные точки, нужно поставить и надписи к меткам, указывающие координаты меток на осях. TikZ предлагает удобную в работе и мощную систему добавления текста, и более широко, сложных форм, в конкретные позиции рисунка. Основная идея состоит в следующем: когда TikZ создает путь и сталкивается с ключевым словом `node` (узел) в середине пути, он читает спецификацию узла. Ключевое слово `node` обычно сопровождается некоторыми опциями (в квадратных скобках) и некоторым текстом (в фигурных скобках). Этот текст помещается в обычный бокс TeX'a (если спецификация узла следует непосредственно за координатой, что обычно и имеет место, но TikZ в состоянии выполнить и некоторые “волшебные манипуляции”, что делает иногда возможным использование дословного текста в боксах), а затем текст помещается в текущей позиции, то есть, в последней указанной позиции (возможен небольшой сдвиг, в соответствии с заданными опциями). Однако, все узлы прорисовываются только после того, как путь полностью нарисован/заполнен/растушеван/....

```
\begin{tikzpicture}
\draw (0,0) rectangle (2,2);
\draw (0.5,0.5) node [fill=yellow]
      {Text at node 1}
  -- (1.5,1.5) node {Text at node 2};
\end{tikzpicture}
```



Иногда нужно поместить узел не в указанной позиции, а слева или справа от нее. Для этого каждый объект `node`, вставляемый в рисунок, снабжен несколькими якорями — `anchor`. Например, северный якорь (`north`) находится в середине верхней части формы, южный якорь (`south`) — в нижней части формы, северо-восточный якорь (`north east`) — в верхнем правом углу. Когда задана опция `anchor=north`, текст будет помещаться так, что этот северный якорь ляжет в текущую позицию и, таким образом, текст будет размещен ниже текущей позиции.

```
\begin{tikzpicture}[scale=3]
\clip (-0.6,-0.2) rectangle (0.6,1.51);
\draw[step=.5cm,help lines]
(-1.4,-1.4) grid (1.4,1.4);
\filldraw[fill=green!20,draw=green!50!black]
(0,0) -- (3mm,0mm)
arc (0:30:3mm) -- cycle;
\draw[->] (-1.5,0) -- (1.5,0);
\draw[->] (0,-1.5) -- (0,1.5);
\draw (0,0) circle (1cm);
\foreach \x in {-1,-0.5,1}
\draw (\x cm,1pt) -- (\x cm,-1pt)
node[anchor=north] {$\x$};
\foreach \y in {-1,-0.5,0.5,1}
\draw (1pt,\y cm) -- (-1pt,\y cm)
node[anchor=east] {$\y$};
\end{tikzpicture}
```



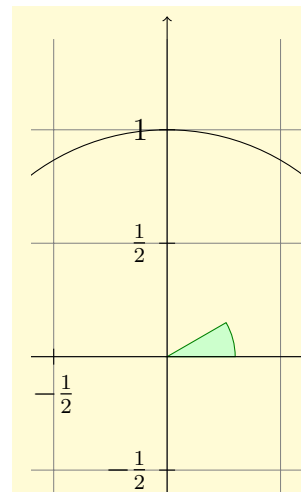
Используя должным образом якоря, можно добавить в рисунок большинство других текстовых элементов. Кроме того, можно поместить текст ниже данной точки, используя опцию `below`, которая делает то же, что и `anchor=north`. Точно так же, есть опция `above right`, которая делает то же, что и `anchor=south east`. Кроме того, опция `below` может иметь необязательный аргумент, который, если задан, позволяет дополнительно сдвинуть вниз форму на заданное в ней число. Так, опция `below=1pt` позволит поместить текст ниже некоторой точки, дополнительно сдвигая его на 1pt вниз.

Еще хотелось бы вместо десятичных дробей (например, 0.5) иногда подставлять рациональную дробь (например,  $\frac{1}{2}$  или  $1/2$ ), а иногда и числа типа  $\pi$ . Другими словами, иногда надо подписать метку не числом, а математическим выражением. TikZ понимает математику, но, в данной ситуации, возникает другая проблема: для команды `\foreach`, точку `\x` нужно задавать как 0.5, так как иначе она не поймет, куда ставить точку `\frac{1}{2}`. С другой стороны, отображаемый на рисунке текст должен иметь вид  $\frac{1}{2}$ . Решить эту проблему, позволяет специальный синтаксис команды `\foreach`: вместо одной переменной `\x` можно определить две (или даже больше) переменных (например, `x` и `xtext`), разделяя их слэшем — `\x/\xtext`. Тогда, элементы в наборе, по которому команда `\foreach` строит цикл, должны также иметь форму `<first>/<second>`. В каждой итерации, переменная `\x` будет принимать значение `<first>`, а переменная `\xtext` — значение `<second>`. Если значение `<second>` не определяется, снова используется значение `<first>`. Этими возможностями можно воспользоваться следующим образом:

```

\begin{tikzpicture}[scale=3]
\clip (-0.6,-0.2) rectangle (0.6,1.51);
\draw[step=.5cm,help lines]
(-1.4,-1.4) grid (1.4,1.4);
\filldraw[fill=green!20,draw=green!50!black]
(0,0) -- (3mm,0mm) arc (0:30:3mm) -- cycle;
\draw[->] (-1.5,0) -- (1.5,0);
\draw[->] (0,-1.5) -- (0,1.5);
\draw (0,0) circle (1cm);
\foreach \x/\xtext in {-1, -0.5/-\frac{1}{2}, 1}
\draw (\x cm,1pt) -- (\x cm,-1pt)
node[anchor=north] {\xtext};
\foreach \y/\ytext in
{-1, -0.5/-\frac{1}{2}, 0.5/\frac{1}{2}, 1}
\draw (1pt,\y cm) -- (-1pt,\y cm)
node[anchor=east] {\ytext};
\end{tikzpicture}

```



Возникла еще одна проблема: сетка сливается с разделителем рациональной дроби и со знаком минус (их не видно). Для этой задачи есть простое решение: добавить в узел опцию `[fill=white]`, делая фон текста метки белым, а весь рисунок поместить в цветной бокс.

Добавим надписи, подобные  $\sin \alpha$ , которые поместим примерно посередине соответствующих линии. Вместо того, чтобы определять узел, например, для  $\sin \alpha$ , можно задать метку непосредственно после `--` и перед координатой соответствующей линии. По умолчанию, она помещается по середине линии. Но есть опция `pos=`, которая позволяет изменить такое размещение. Кроме того, чтобы изменить местоположение метки, могут использоваться опции `near start` и `near end`.

```

\colorbox{yellow!40!white}{
\begin{tikzpicture}[scale=3]
\clip (-2,-0.2) rectangle (2,0.8);
\draw[step=.5cm,gray,very thin] (-1.4,-1.4) grid (1.4,1.4);
\filldraw[fill=green!20,draw=green!50!black]
(0,0) -- (3mm,0mm) arc (0:30:3mm) -- cycle;
\draw[->] (-1.5,0) -- (1.5,0) coordinate (x axis);
\draw[->] (0,-1.5) -- (0,1.5) coordinate (y axis);
\draw (0,0) circle (1cm);
\draw[very thick,red]
(30:1cm) -- node[left=1pt,fill=white] {\sin \alpha} (30:1cm |- x axis);
\draw[very thick,blue]
(30:1cm |- x axis) -- node[below=2pt,fill=white] {\cos \alpha} (0,0);
\draw[very thick,orange] (1,0) -- node [right=1pt,fill=white]
{\displaystyle \tan \alpha \color{black}=
\frac{\color{red}\sin \alpha}{\color{blue}\cos \alpha}}
(intersection of 0,0--30:1cm and 1,0--1,1) coordinate (t);
\draw (0,0) -- (t);
\foreach \x/\xtext in {-1, -0.5/-\frac{1}{2}, 1}

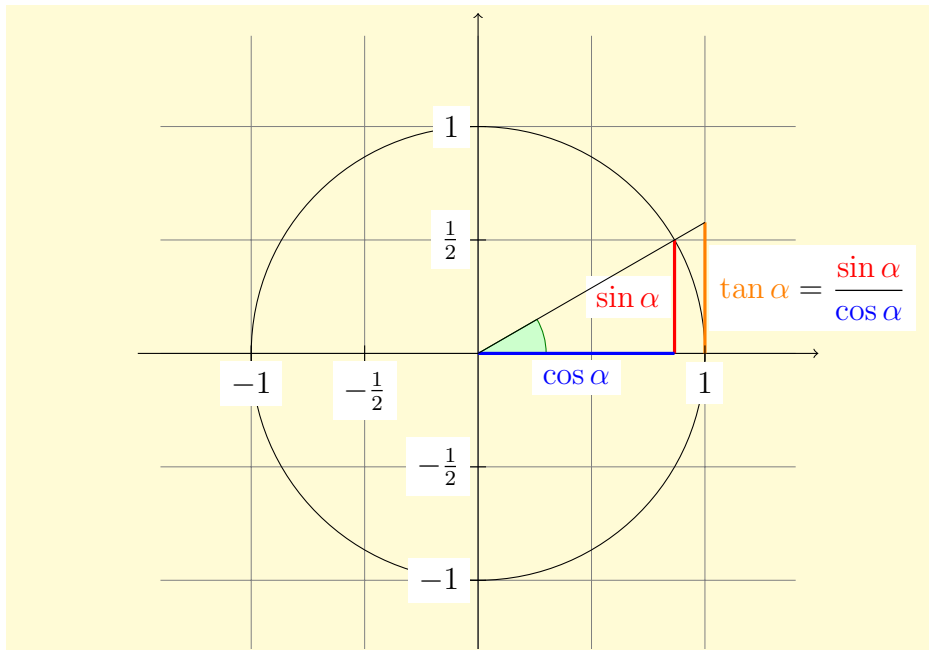
```



```

\draw (\x cm,1pt) -- (\x cm,-1pt) node[anchor=north,fill=white] {\xtext$};
\foreach \y/\ytext in {-1, -0.5/\frac{1}{2}, 0.5/\frac{1}{2}, 1}
\draw (1pt,\y cm) -- (-1pt,\y cm) node[anchor=east,fill=white] {\ytext$};
\end{tikzpicture}

```

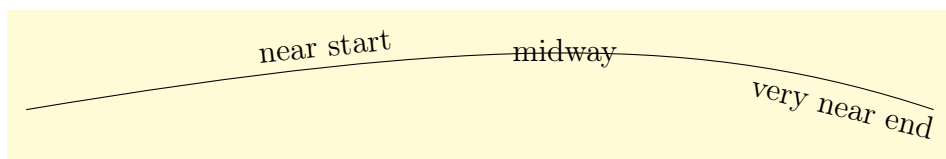


Можно также устанавливать надписи на кривых, добавляя опцию `sloped`, которая размещает их в соответствии с наклоном линии. Все тонкости поясняет следующий пример:

```

\begin{tikzpicture}
\draw (0,0) .. controls (6,1) and (9,1) ..
node[near start,sloped,above] {near start}
node {midway}
node[very near end,sloped,below] {very near end} (12,0);
\end{tikzpicture}

```



## 2.20 Полный код примера

Остается нарисовать поясняющий текст справа от рисунка. Главная трудность состоит в ограничении ширины поясняющего текста. В этой ситуации нужно использовать опцию `text width`. Приведем полный код примера, в который добавлены пока не объясненные, но, возможно, понятные, улучшающие дополнения.

```

\colorbox{yellow!20!white}{
\begin{tikzpicture}[scale=2.9]

```

```

% Определение цвета разных частей рисунка

```

```

\colorlet{anglecolor}{green!80!black}
\colorlet{sincolor}{red}
\colorlet{tancolor}{orange!80!black}
\colorlet{coscolor}{blue}

% Рисунок
\draw[step=.5cm,gray,very thin] (-1.4,-1.4) grid (1.4,1.4);
\filldraw[fill=green!20,draw=anglecolor]
    (0,0) -- (3mm,0mm) arc (0:30:3mm) -- cycle;
\draw[->] (-1.5,0) -- (1.5,0) coordinate (x axis);
\draw[->] (0,-1.5) -- (0,1.5) coordinate (y axis);
\draw (0,0) circle (1cm);
\draw[very thick,sincolor]
    (30:1cm) -- node[left=1pt,fill=white] {\sin \alpha} (30:1cm |- x axis);
\draw[very thick,coscolor]
    (30:1cm |- x axis) -- node[below=2pt,fill=white] {\cos \alpha} (0,0);
\draw[very thick,tancolor](1,0) -- node [right=1pt,fill=white]
    {\displaystyle \tg \alpha \color{black}=
        \frac{{\color{red}\sin \alpha}}{{\color{blue}\cos \alpha}}$}
        (intersection of 0,0--30:1cm and 1,0--1,1) coordinate (t);
\draw (0,0) -- (t);
\foreach \x/\xtext in {-1, -0.5/-\frac{1}{2}, 1}
    \draw (\x cm,1pt) -- (\x cm,-1pt) node[anchor=north,fill=white] {\xtext};
\foreach \y/\ytext in {-1, -0.5/-\frac{1}{2}, 0.5/\frac{1}{2}, 1}
    \draw (1pt,\y cm) -- (-1pt,\y cm) node[anchor=east,fill=white] {\ytext};

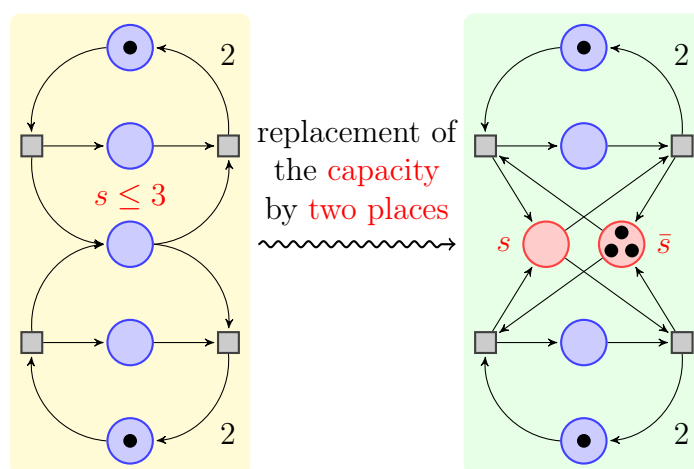
% Поясняющий текст
\draw[xshift=2cm]
node[right,text width=5cm]
{
    В примере {\color{anglecolor} угол $\alpha$ равен $30^\circ$
    ($\pi/6$ в радианах). Красный отрезок равен \linebreak
    ${\color{sincolor} \sin \alpha} = 1/2.$
    \par По теореме Пифагора
    $$ {\color{sincolor}\sin^2\alpha} + {\color{coscolor}\cos^2\alpha} =1.$$
    Поэтому длина голубого отрезка равна
    $$ {\color{coscolor}\cos\alpha} = \sqrt{1-\dfrac{1}{4}}=\dfrac{\sqrt{3}}{2}.$
    Тогда длина оранжевого отрезка равна
    $$ {\color{tancolor} \tg \alpha}=
        \frac{{\color{sincolor}\sin\alpha}}{{\color{coscolor}\cos\alpha}}=
        1/\sqrt{3}.$
};
\end{tikzpicture}
}

```

# Глава 3

## Сеть Петри

Рассмотрим пример, требующий построения узлов и связей между ними.



Создадим рисунок, демонстрирующий, как сеть Петри с местами, имеющими емкость, может быть смоделирована сетью без емкостей. Чтобы решить задачу, надо загрузить не только пакет `tikz`, но и дополнительные пакеты, содержащие библиотеки, расширяющие возможности пакета `tikz`: библиотеку стрелок `arrows`, библиотеку `decoration.pathmorphing` для рисования извилистых линий, библиотеку `backgrounds` для рисования двух прямоугольных областей, которые располагаются позади двух главных частей рисунка, библиотеку `fit`, позволяющую легко вычислять размеры этих прямоугольников, библиотеку `positioning`, чтобы иметь возможность позиционировать одни узлы относительно других, и библиотеку `petri`, ориентированную на создание рисунков сетей Петри. Поэтому в преамбулу  $\LaTeX$ -документа нужно вставить строки

```
\usepackage{tikz}
\usetikzlibrary{arrows,decorations.pathmorphing,
                backgrounds,positioning,fit,petri}
```

### 3.1 Введение в узлы

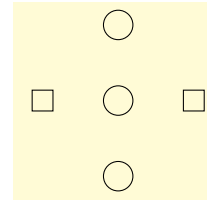
В принципе, чтобы создать нужный рисунок, следует взять большой прямоугольник, добавить в него маленькие круги и прямоугольники, плюс некоторые стрелки, соединяющие их. Однако, такой подход имеет много недостатков: во-первых, в случае необходимости в таком рисунка трудно изменить что-либо. Например, если надо будет

добавить больше мест к сетям Петри (в теории сетей Петри круги называют местами), все координаты изменятся, и придется повторно все вычислять. Во-вторых, трудно читать код для сети Петри, представляющий длинный и сложный список координат и команд рисования — основная структура сети Петри теряется.

TikZ имеет мощный механизм узлов, позволяющий избежать этих проблем. При построении примера в главе 2 узлы уже использовались, чтобы добавить текст к графическим меткам рисунка, но узлы гораздо более мощный механизм. Когда узел создается, нужно указать позицию, в которой он должен быть нарисован, и его форму. Узел формы `circle` рисуется как круг, узел формы `rectangle` — как прямоугольник, и так далее. Узел может содержать текст. Наконец, узлу можно дать *имя* для дальнейшей ссылки на него.

В рисунке сети Петри будем использовать узлы для мест и переходов: места — круги, переходы — прямоугольники. Начнем рисунок с формирования верхней половины левой сети Петри, в которой три места и два перехода. Вместо того, чтобы нарисовать три круга и два прямоугольника, используем понятие узла, три из которых имеют форму `circle`, а два — форму `rectangle`.

```
\begin{tikzpicture}
\path ( 0,2) node [shape=circle,draw] {}
      ( 0,1) node [shape=circle,draw] {}
      ( 0,0) node [shape=circle,draw] {}
      ( 1,1) node [shape=rectangle,draw] {}
      (-1,1) node [shape=rectangle,draw] {};
\end{tikzpicture}
```



Этот рисунок далек от того, что нужно, но полезен как первый шаг. Посмотрим внимательно на код. Весь рисунок состоит из единственного пути: последовательности координат, не связанных между собой. И, даже если что-то случится, и надо будет нарисовать прямую или кривую от узла к узлу, команда `\path` ничего не должна делать с уже созданным путем. Таким образом, все магические действия должны происходить в узлах — в командах `node`. В примере каждый из пяти узлов добавляется в путь в разных точках, а текст узла пуст (на его месте стоит `{}`). Так, почему же мы все же что-то видим? Дело в том, что опция `draw` операции `node` рисует вокруг текста (даже пустого) указанную опцией форму.

Таким образом, код `(0,2) node [shape=circle,draw] {}` имеет следующий смысл: в создаваемом пути переместиться в позицию  $(0,2)$ , создать в этой позиции узел, но пока не рисовать его, а нарисовать после того, как весь путь будет создан. Далее, согласно коду, переместиться в позицию  $(0,1)$ ... , и так далее.

Первый вариант кода несовершенен и, наверное, не совсем понятен. Так как узлы помещаются в последнюю указанную позицию, предыдущий код можно написать в более понятной форме, пользуясь спецификатором `at`:

```
\begin{tikzpicture}
\path node at ( 0,2) [shape=circle,draw] {}
      node at ( 0,1) [shape=circle,draw] {}
      node at ( 0,0) [shape=circle,draw] {}
      node at ( 1,1) [shape=rectangle,draw] {}
      node at (-1,1) [shape=rectangle,draw] {};
\end{tikzpicture}
```

Но и этот код можно сделать лучше с помощью команды `\node`, которая является сокращением для `\path node`. В команде `\node` можно опускать спецификатор `shape=` (как и спецификатор `color=`), если это не приводит к путанице.

```
\begin{tikzpicture}
  \node at ( 0,2) [circle,draw] {};
  \node at ( 0,1) [circle,draw] {};
  \node at ( 0,0) [circle,draw] {};
  \node at ( 1,1) [rectangle,draw] {};
  \node at (-1,1) [rectangle,draw] {};
\end{tikzpicture}
```

## 3.2 Использование стилей

Чтобы заполнить круги и прямоугольники разным цветом, напомним такой код:

```
\begin{tikzpicture}[thick]
  \node at (0,2) [circle,draw=blue!50,fill=blue!20] {};
  \node at (0,1) [circle,draw=blue!50,fill=blue!20] {};
  \node at (0,0) [circle,draw=blue!50,fill=blue!20] {};
  \node at (1,1)
    [rectangle,draw=black!50,fill=black!20] {};
  \node at (-1,1)
    [rectangle,draw=black!50,fill=black!20] {};
\end{tikzpicture}
```

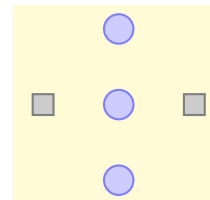


Рисунок выглядит лучше, но код стал громоздким и уродливым. В идеале, хотелось бы код преобразовать в сообщение: нарисовать три места и два перехода, и не указывать цвет. Решить задачу позволяют стили. Определим один стиль для мест, другой стиль для переходов и получим тот же рисунок, но при более коротком коде.

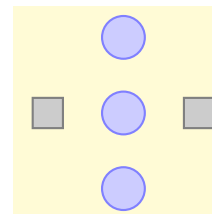
```
\begin{tikzpicture}
[place/.style={circle,draw=blue!50,fill=blue!20,thick},
transition/.style={rectangle,draw=black!50,fill=black!20,thick}]
\node at ( 0,2) [place] {};
\node at ( 0,1) [place] {};
\node at ( 0,0) [place] {};
\node at ( 1,1) [transition] {};
\node at (-1,1) [transition] {};
\end{tikzpicture}
```

## 3.3 Размеры узлов

Прежде, чем именовать и связывать узлы, изменим размеры узлов — увеличим их, они слишком маленькие. Вообще говоря, следует задать вопрос: почему узлы вообще имеют размер, ведь текст пуст? Причина в том, что TikZ автоматически добавляет некоторое пустое пространство вокруг текста. Его объем определяется опцией `inner sep`. Поэтому, чтобы увеличивать размеры узлов, нужно в этой опции определить

другое значение, например, `inner sep=2mm`. Таким образом, предыдущий код нужно переписать следующим образом:

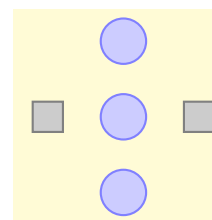
```
\begin{tikzpicture}[inner sep=2mm,
  place/.style={circle,draw=blue!50,fill=blue!20,thick},
  transition/.style={rectangle,draw=black!50,
    fill=black!20,thick}]
\node at ( 0,2) [place] {};
\node at ( 0,1) [place] {};
\node at ( 0,0) [place] {};
\node at ( 1,1) [transition] {};
\node at (-1,1) [transition] {};
\end{tikzpicture}
```



Однако, это не лучший способ добиться желаемого. Лучше использовать опцию `minimum size`, позволяющую определить минимальный размер, который должен иметь узел. Если узел должен стать больше из-за более длинного текста, он станет длиннее, но если текст отсутствует, узел будет иметь минимальный размер. Есть еще опции `minimum height` и `minimum width`, позволяющие независимо определять минимальную высоту и ширину.

Таким образом, следует определить минимальный размер узлов и установить опцию `inner sep=0pt`, гарантируя, что узлы будут действительно иметь минимальный размер, а при наличии текста автоматически добавляемого пространства не будет.

```
\begin{tikzpicture}
[place/.style={circle,draw=blue!50,fill=blue!20,
  thick,inner sep=0pt,minimum size=6mm},
transition/.style={rectangle,draw=black!50,fill=black!20,
  thick,inner sep=0pt,minimum size=4mm}]
\node at ( 0,2) [place] {};
\node at ( 0,1) [place] {};
\node at ( 0,0) [place] {};
\node at ( 1,1) [transition] {};
\node at (-1,1) [transition] {};
\end{tikzpicture}
```



### 3.4 Именованние узлов

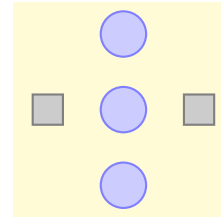
Следующая задача состоит в том, чтобы соединить узлы, используя стрелки. Стрелки должны начинаться не в середине узлов, а в точке, расположенной где-нибудь на границе, но не хотелось бы вычислять эти позиции вручную. К счастью, `pgf` сумеет выполнить все необходимые вычисления сам. Однако, сначала нужно назначить узлам имена так, чтобы на них можно было позже сослаться.

Есть два способа именования узлов. Первый — использовать опцию `name=`. Второй — написать имя в круглых скобках сразу после команды `\node`. Второй метод сначала кажется несколько странным, но он имеет свои преимущества.

```

\begin{tikzpicture} [Определение стилей . . . ]
\node (waiting)      at ( 0,2) [place] {};
\node (critical)     at ( 0,1) [place] {};
\node (semaphore)    at ( 0,0) [place] {};
\node (leave critical) at ( 1,1) [transition] {};
\node (enter critical) at (-1,1) [transition] {};
\end{tikzpicture}

```



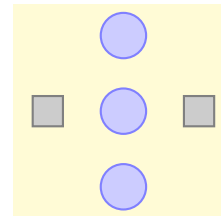
Имена узлов должны помочь в понимании кода. Они могут быть произвольными, но не должны содержать запятых, точек, круглых скобок, двоеточий и некоторых других специальных символов. Но могут содержать символ подчеркивания и дефис.

Синтаксис для команды `\node` весьма либерален относительно порядка, в котором должны появляться имена узлов, спецификатор `at`, различные опции. Между командой `\node` и текстом в фигурных скобках можете стоять даже несколько необязательных блоков. Их можно менять местами и, возможно, следующий порядок предпочтительнее:

```

\begin{tikzpicture} [Определение стилей . . . ]
\node[place]      (waiting)      at ( 0,2) {};
\node[place]      (critical)     at ( 0,1) {};
\node[place]      (semaphore)    at ( 0,0) {};
\node[transition] (leave critical) at ( 1,1) {};
\node[transition] (enter critical) at (-1,1) {};
\end{tikzpicture}

```



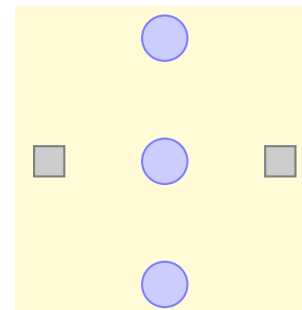
### 3.5 Относительное размещение узлов

Прежде, чем соединять узлы, снова обратимся к проблеме размещения узлов. Хотя `at`-синтаксис хорош, в данном случае предпочтительнее размещать узлы относительно друг друга. В таком случае можно сказать, что узел `(critical)` должен быть ниже узла `(waiting)`, где бы не располагался последний. Есть несколько способов добиться этого, но самый хороший — использовать опции `above`, `below`, `right`, `left`:

```

\begin{tikzpicture} [Определение стилей . . . ]
\node[place] (waiting)  {};
\node[place] (critical) [below=of waiting] {};
\node[place] (semaphore) [below=of critical] {};
\node[transition] (leave critical)
                    [right=of critical] {};
\node[transition] (enter critical)
                    [left=of critical] {};
\end{tikzpicture}

```



С библиотекой `positioning`, когда опция типа `below=` сопровождается опцией `of`, позиция сдвигаемого узла располагается на расстоянии `node distance` от указанного узла в указанном направлении; `node distance` — расстояние между центрами узлов (когда `on grid` есть `true`), или расстояние между границами (когда `on grid` — `false`, это значе-

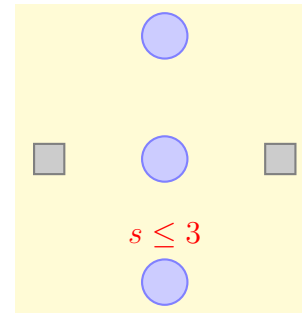
ние по умолчанию). Последний способ размещения узлов предпочтительнее, поскольку код понятнее и дает представление о положении узлов друг относительно друга.

### 3.6 Добавление текста рядом с узлами

Чтобы добавить текст рядом с узлом, существуют два подхода.

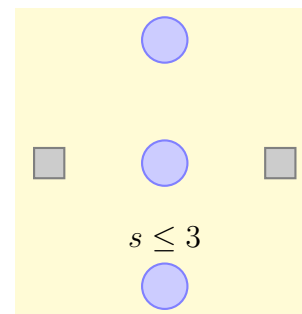
1. Общий подход, который будет работать всегда. Например, можно добавить новый узел выше северного якоря узла `semaphore`.

```
\begin{tikzpicture}[Определение стилей . . . ]
\node[place] (waiting) {};
\node[place] (critical) [below=of waiting] {};
\node[place] (semaphore) [below=of critical] {};
\node[transition] (leave critical)
    [right=of critical]{};
\node[transition] (enter critical)
    [left=of critical] {};
\node [red,above] at (semaphore.north) {$s \le 3$};
\end{tikzpicture}
```



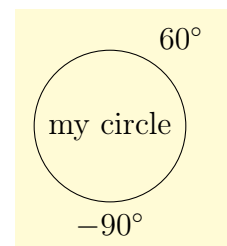
2. Использовать опцию `label` (можно многократно), которая определяется для узла и создает другой узел, добавляемый рядом с тем узлом, в котором задана опция `label`. Чтобы получить результат, аналогичный первому, нужно использовать опцию `label=above:$s \le 3$`.

```
\begin{tikzpicture}[Определение стилей . . . ]
\node[place] (waiting) {};
\node[place] (critical) [below=of waiting] {};
\node[place] (semaphore)
    [below=of critical,label=above:$s \le 3$] {};
\node[transition] (leave critical)
    [right=of critical]{};
\node[transition] (enter critical)
    [left=of critical] {};
\end{tikzpicture}
```



Перед двоеточием Вместо `above` можно использовать нечто подобное `below left` или число, например, `60`.

```
\tikz
\node[shape=circle,draw,
    label=60:$60^\circ$,
    label=below:$-90^\circ$] {my circle};
```

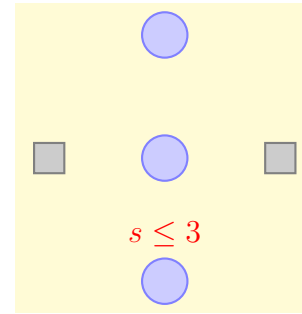


Чтобы метка  $s \leq 3$  была красной, вместо `label=above:$s \le 3$` нужно написать `label={ [red] above:$s \le 3$}`. Здесь нужны фигурные скобки, чтобы  $\LaTeX$  не принял закрывающую скобку `]` после `red` за закрывающую скобку описания опций команды



`\node`. Но лучше определить новый стиль: `every label/.style={red}`.

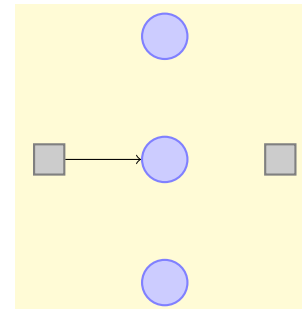
```
\begin{tikzpicture}
[Определение стилей place и transition,
every label/.style={color=red} ]
\node[place] (waiting) {};
\node[place] (critical) [below=of waiting] {};
\node[place] (semaphore)
[below=of critical,label=above:$s\leq 3$] {};
\node[transition](leave critical)
[right=of critical] {};
\node[transition](enter critical)
[left=of critical] {};
\end{tikzpicture}
```



### 3.7 Соединение узлов

Решение задачи соединения узлов начнем с самой простой из них, а именно, с соединения прямой линией. Соединим правую сторону узла `enter critical` с левой стороной узла `critical`. Для этого используем якоря узлов. Каждый узел определяет набор якорей, которые лежат на его границе или в нем самом. Например, якорь `center` располагается в центре узла, якорь `west` на левой границе узла, и так далее. Чтобы обращаться к координатам якорей узла, следует использовать те координаты, которые содержит имя узла, сопровождаемое точкой, после которой располагается имя якоря.

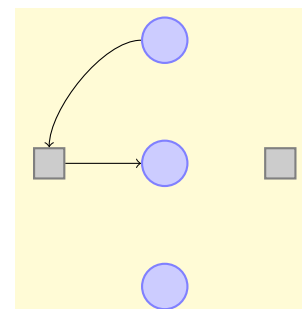
```
\begin{tikzpicture}[Определение стилей . . . ]
\node[place] (waiting) {};
\node[place] (critical) [below=of waiting] {};
\node[place] (semaphore)[below=of critical] {};
\node[transition](leave critical)
[right=of critical]{};
\node[transition] (enter critical)
[left=of critical]{};
\draw[->](enter critical.east) -- (critical.west);
\end{tikzpicture}
```



Проведем кривую от узла `waiting` к узлу `enter critical`. Для этого определим начальную и конечную точки кривой и контрольные точки:

```
\begin{tikzpicture}
[Определение стилей и узлов . . . ]

\draw[->](enter critical.west) -- (critical.east);
\draw[->] (waiting.west) .. controls +(left:5mm)
and +(up:5mm) .. (enter critical.north);
\end{tikzpicture}
```



Теперь можно было бы нарисовать все нужные дуги, но приведенный код выглядит неуклюжим, не гибким, и не отражает в полной мере структуру рисунка. Поэтому постараемся сделать его лучше. Сначала, уберем якоря и изменим контрольные точки:

```
\begin{tikzpicture}
[Определение стилей и узлов . . . ]

\draw[->](enter critical) -- (critical);
\draw[->] (waiting) .. controls +(left:8mm)
and +(up:8mm) .. (enter critical);
\end{tikzpicture}
```

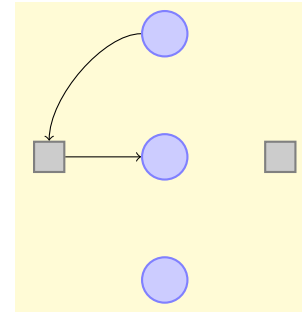
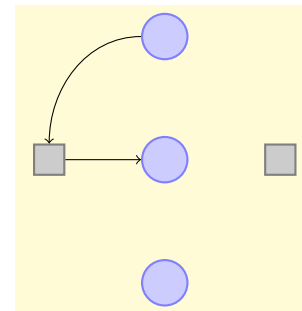


Рисунок практически не изменился. Всякий раз, когда TikZ сталкивается с именем узла при определении его координат, он старается быть интеллектуалом в вопросе выбора якорей для этого узла. В зависимости от ситуации, TikZ выберет якорь, который находится на границе узла на линии к следующей позиции или контрольной точке. Точные правила немного сложнее, но выбранная TikZ точка обычно бывает правильной, и если это не то, что нужно, всегда можно указать желаемый якорь вручную.

Но можно упростить и саму операцию создания кривой, используя специальную операцию построения пути: операцию `to`. Эта операция может использовать много опций (можно самостоятельно определять новые опции). В данной ситуации полезна пара опций: `in` и `out`. Они определяют углы в тех точках, в которых кривая должна начинаться или завершаться. Без задания таких углов будет нарисована прямая линия.

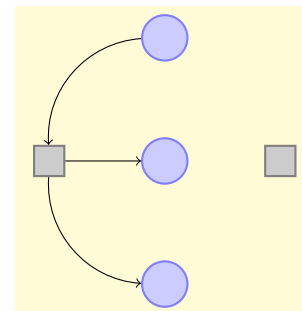
```
\begin{tikzpicture}
[Определение стилей и узлов . . . ]

\draw [->] (enter critical) to (critical);
\draw [->] (waiting) to [out=180,in=90]
(enter critical);
\end{tikzpicture}
```



Есть и другие опции для операции `to`, которые еще лучше подходят для решения данной задачи: опции `bend right` и `bend left`. Эти опции также принимают угол, на который кривая должна изгибаться вправо или влево, соответственно. Пользуясь новой опцией, соединим еще два узла.

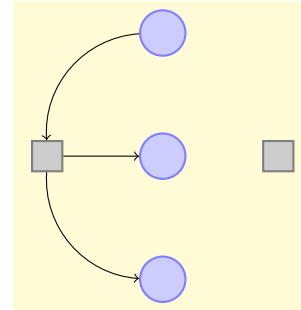
```
\begin{tikzpicture}
[Определение стилей и узлов . . . ]
\draw [->] (enter critical) to (critical);
\draw [->] (waiting) to [bend right=45]
(enter critical);
\draw [->] (enter critical) to [bend right=45]
(semaphore);
\end{tikzpicture}
```



Еще один способ определения дуги — использовать операцию пути `edge`. Она очень похожа на операцию `to`, но есть одно важное отличие: подобно `node`, операция `edge`

не является частью главного пути, а добавляется позже. Это, возможно, не выглядит очень важным моментом, но имеет некоторые важные следствия. Например, каждая дуга может иметь собственные метки, собственный цвет и так далее. И еще, все дуги можно задать в одном и том же пути. Это позволяет написать следующий код:

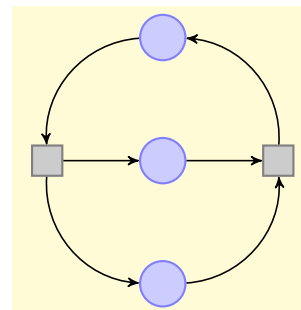
```
\begin{tikzpicture}[Определение стилей . . .]
\node[place] (waiting) {};
\node[place] (critical) [below=of waiting] {};
\node[place] (semaphore)[below=of critical] {};
\node[transition](leave critical)
    [right=of critical]{};
\node[transition] (enter critical)
    [left=of critical]{}
    edge [->] (critical)
    edge [<-,bend left=45] (waiting)
    edge [->,bend right=45] (semaphore);
\end{tikzpicture}
```



Обратите внимание на исчезновение точки с запятой после объявления узла `enter critical` и последующую привязку всех дуг к этому узлу, как к начальной точке. Конечная точка дуги — узел после команды `edge`.

Чтобы завершить создание понятного языка рисунка, введем еще два стиля `pre` и `post` (опция `>=stealth'` устанавливает тип используемой стрелки) и определим опцию `bend angle=45`, чтобы установить один и тот же угол изгиба для всех ситуаций.

```
\begin{tikzpicture}
[Определение стилей place и transition,
bend angle=45,
pre/.style={<-,>=stealth',semithick},
post/.style={->,>=stealth',semithick}]
\node[place] (waiting) {};
\node[place] (critical) [below=of waiting] {};
\node[place] (semaphore)[below=of critical] {};
\node[transition](leave critical)[right=of critical]{}
    edge [pre] (critical)
    edge [post,bend right] (waiting)
    edge [pre, bend left] (semaphore);
\node[transition](enter critical) [left=of critical] {}
    edge [post] (critical)
    edge [pre, bend left] (waiting)
    edge [post,bend right] (semaphore);
\end{tikzpicture}
```

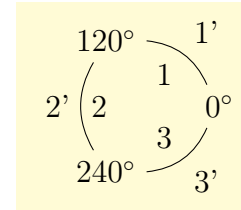


### 3.8 Метки рядом с дугами

Добавим к дугам метки. Чтобы узлы с текстом на кривых размещались автоматически, добавим опцию `auto`, в соответствии с которой TikZ установит узлы с текстом не

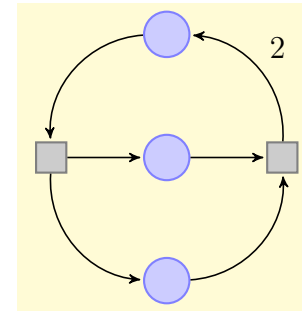
на кривых, а рядом с ними. Добавим опцию `swap`, которая отобразит метки зеркально относительно линии. Сначала приведем поясняющий пример:

```
\begin{tikzpicture}[auto,bend right]
\node (a) at (0:1) {$0^\circ$};
\node (b) at (120:1) {$120^\circ$};
\node (c) at (240:1) {$240^\circ$};
\draw (a) to node [swap] {1'} (b)
      (b) to node [swap] {2'} (c)
      (c) to node [swap] {3'} (a);
\end{tikzpicture}
```



Когда узлы задаются внутри операции `to`, текстовый узел помещается в середину кривой или линии, созданной операцией `to`. Опция `auto` такой узел помещает так, чтобы он располагался рядом с кривой. В примере выше определяются по два текстовых узла в каждой операции `to`.

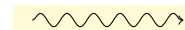
```
\begin{tikzpicture}[Определение стилей . . . ]
\node[place] (waiting) {};
\node[place] (critical)[below=of waiting]{};
\node[place] (semaphore)[below=of critical]{};
\node[transition] (leave critical)
  [right=of critical]{}
  edge [pre] (critical)
  edge [post,bend right]
    node [auto,swap] {2} (waiting)
  edge [pre, bend left] (semaphore);
\node[transition] (enter critical)
  [left=of critical]{}
  edge [post] (critical)
  edge [pre, bend left] (waiting)
  edge [post,bend right]
\end{tikzpicture}
```



### 3.9 Волнистые линии и многострочный текст

Чтобы создать волнистую линию между узлами, нужно использовать библиотеку `decoration`. Для создания волнистой (змеевидной — `snake`) линии нужно установить на кривой две опции: `decorate` и `decoration=snake`. Отметим, что волнистые линии можно использовать только в специальных частях пути (пока это не важно).

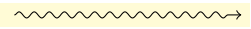
```
\begin{tikzpicture}
\draw [->,decorate,decoration=snake] (0,0) -- (2,0);
\end{tikzpicture}
```



Выглядит плохо, потому что волнистая линия, так иногда случается, заканчивается не в той точке, в которой начинается стрелка. Но с помощью опций можно справиться с

этой проблемой. Кроме того, волнистая линия часто должна иметь меньшую амплитуду, для установки которой также есть нужные опции.

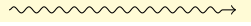
```
\begin{tikzpicture}
\draw [->,decorate,decoration={snake,amplitude=.4mm,
segment length=2mm,post length=1mm}] (0,0) -- (3,0);
\end{tikzpicture}
```



Чтобы расположить многострочный текст выше волнистой линии, следует определить способ выравнивания текста соответствующей опцией, например, `align=center`, и, используя команды `\`, определить точки разрыва текста.

```
\begin{tikzpicture}
\draw [->,decorate,decoration={snake,amplitude=.4mm,
segment length=2mm,post length=1mm}] (0,0) -- (3,0)
node [above,align=center,midway] {replacement of\
the \textcolor{red}{capacity}\
by \textcolor{red}{two places}};
\end{tikzpicture}
```

replacement of  
the **capacity**  
by **two places**



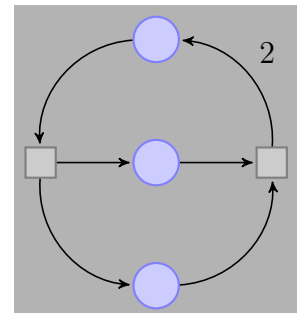
## 3.10 Фоновые прямоугольники

Добавим в рисунок фоновые прямоугольники, которые нарисуем после сети Петри. Нарисовать этот прямоугольник первым не получится — еще не известны точные размеры сети Петри. Чтобы справиться с задачей, воспользуемся понятием **уровня**.

После того, как загружена фоновая библиотека `background`, можно поместить часть рисунка в окружение `{pgfonlayer}`, делая его частью уровня, который задается в виде параметра этого окружения. Когда окружение `{tikzpicture}` завершается, уровни помещаются сверху друг друга, начиная с уровня `background`. Это приводит к тому, что фоновый уровень будет рисоваться самым первым, под основным текстом.

Следующий важный вопрос: каковы размеры прямоугольника, в который поместятся все узлы? Конечно, их можно вычислить вручную, используя данные о  $x$ - и  $y$ -координатах узлов, но лучше заставить сам TikZ вычислить этот прямоугольник. Для этого нужно использовать библиотеку `fit`, определяющую опцию `fit`, которая, когда объявляется в узле, заставляет этот узел изменить размеры и сдвинуться так, чтобы точно покрыть все узлы и координаты, заданные как параметры опции `fit`.

```
\begin{tikzpicture}
[Определение стилей, узлов и дуг . . . ]
\begin{pgfonlayer}{background}
\node [fill=black!30,
fit=(waiting) (critical) (semaphore)
(leave critical) (enter critical)] {};
\end{pgfonlayer}
\end{tikzpicture}
```



### 3.11 Полный код примера

Как было сказано в начале главы, есть библиотека `petri`, предназначенная для создания рисунков сети Петри, которая дает почти те же определения, что были сделаны выше. Корректировка кода, использующая библиотеку, немного его сокращает, поскольку нужно меньше определений стилей, но все еще нужно определять цвета для мест и переходов.

```

\begin {tikzpicture}
%Определение стилей
[node distance=1.3cm,on grid,>=stealth',bend angle=45,auto,
  place/.style= {shape=circle,minimum size=6mm,thick,
                 draw=blue!75,fill=blue!20},
  transition/.style={shape=rectangle,thick,draw=black!75,fill=black!20},
  red place/.style= {place,draw=red!75,fill=red!20},
  every label/.style= {color=red}]

%Определение узлов и дуг левой части рисунка

\node [place,tokens=1] (w1) {};
\node [place]          (c1) [below=of w1] {};
\node [place]          (s)  [below=of c1,label=above:$s\le 3$] {};
\node [place]          (c2) [below=of s] {};
\node [place,tokens=1] (w2) [below=of c2] {};
\node [transition] (e1) [left=of c1] {}
  edge [pre,bend left] (w1)
  edge [post,bend right] (s)
  edge [post] (c1);
\node [transition] (e2) [left=of c2] {}
  edge [pre,bend right] (w2)
  edge [post,bend left] (s)
  edge [post] (c2);
\node [transition] (l1) [right=of c1] {}
  edge [pre] (c1)
  edge [pre,bend left] (s)
  edge [post,bend right] node[swap] {2} (w1);
\node [transition] (l2) [right=of c2] {}
  edge [pre] (c2)
  edge [pre,bend right] (s)
  edge [post,bend left] node {2} (w2);

%Определение узлов и дуг правой части рисунка

\begin{scope}[xshift=6cm]
\node [place,tokens=1] (w1') {};
\node [place]          (c1') [below=of w1'] {};
\node [red place]      (s1') [below=of c1',xshift=-5mm]
  [label=left:$s$] {};
\node [red place,tokens=3] (s2') [below=of c1',xshift=5mm]
  [label=right:$\bar{s}$] {};

```

```

\node [place]          (c2') [below=of s1',xshift=5mm] {};
\node [place,tokens=1] (w2') [below=of c2'] {};
\node [transition] (e1') [left=of c1'] {}
  edge [pre,bend left] (w1')
  edge [post]          (s1')
  edge [pre]           (s2')
  edge [post]          (c1');
\node [transition] (e2') [left=of c2'] {}
  edge [pre,bend right] (w2')
  edge [post]           (s1')
  edge [pre]           (s2')
  edge [post]          (c2');
\node [transition] (l1') [right=of c1'] {}
  edge [pre]           (c1')
  edge [pre]           (s1')
  edge [post]          (s2')
  edge [post,bend right] node[swap] {2} (w1');
\node [transition] (l2') [right=of c2'] {}
  edge [pre]           (c2')
  edge [pre]           (s1')
  edge [post]          (s2')
  edge [post,bend left] node {2} (w2');
\end{scope}

% Код для фона частей и волнистой линии посередине

\begin{pgfonlayer}{background}
  \node (r1) [fill=yellow!20,rounded corners,
    fit=(w1)(w2)(e1)(e2)(l1)(l2)] {};
  \node (r2) [fill=green!10,rounded corners,
    fit=(w1')(w2')(e1')(e2')(l1')(l2')] {};
\end{pgfonlayer}

\draw [shorten >=1mm,-to,thick,decorate,
  decoration={snake,amplitude=.4mm,segment length=2mm,
    pre=moveto,pre length=1mm,post length=2mm}]
  (r1) -- (r2) node [above=1mm,midway,text width=3cm,align=center]
  {replacement of the \textcolor{red}{capacity} by
    \textcolor{red}{two places}};
\end{tikzpicture}

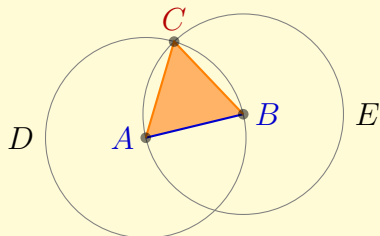
```

## Глава 4

# Элементы геометрии Евклида

Используем TikZ для рисования сложных геометрических конструкций. Для решения задач этой главы нужно подгрузить TikZ-библиотеки `calc`, `intersections`, `through`, `backgrounds`.

### 4.1 Начала. Книга I. Предложение 1



#### Предложение I.

Построить *равносторонний треугольник* на заданном отрезке прямой.

Пусть  $AB$  — заданный *отрезок прямой*. Нужно построить *равносторонний треугольник*, одна из сторон которого — отрезок  $AB$ .

Для этого построим две окружности: окружность  $D$  с центром в точке  $A$  и радиусом, равным длине отрезка  $AB$ , и окружность  $E$  с центром в точке  $B$  и радиусом, равным длине отрезка  $AB$ . Эти окружности пересекаются в двух точках. Обозначим одну из них через  $C$ . Соединим точку  $C$  с точками  $A$  и  $B$  прямолинейными отрезками. Так как  $AB$  и  $AC$  — радиусы в окружности  $D$ , то их длины равны. Аналогично, так как  $AB$  и  $BC$  — радиусы в окружности  $E$ , то их длины равны.

Следовательно, три отрезка  $AB$ ,  $BC$  и  $AC$  имеют одну и ту же длину, и треугольник  $ABC$  — *равносторонний*.

#### 4.1.1 Построение прямой

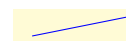
Прежде всего нарисуем прямую  $AB$ . Это достаточно просто сделать, например, написать `\draw (0,0) -- (2,1);`. Однако, нам нужны не точки  $(0,0)$  и  $(2,1)$ , а произвольные точки  $A$  и  $B$ , на которые впоследствии можно сослаться. Поэтому сначала определим две точки, используя команду `\coordinate`:



```

\begin{tikzpicture}
\coordinate (A) at (0,0);
\coordinate (B) at (1.25,0.25);
\draw[blue] (A) -- (B);
\end{tikzpicture}

```

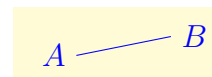


Чтобы указать точки на рисунке, используем опцию `label`:

```

\begin{tikzpicture}
\coordinate [label=left:\textcolor{blue}{\$A\$}]
(A) at (0,0);
\coordinate [label=right:\textcolor{blue}{\$B\$}]
(B) at (1.25,0.25);
\draw[color=blue] (A) -- (B);
\end{tikzpicture}

```



Можно определить точки так, чтобы они в некотором смысле были случайными. В TikZ есть функция `rand`, которая возвращает случайное число между  $-1$  и  $1$ . Так как TikZ немного знает математику, можно задать координаты точек следующим образом:

```

\coordinate [...] (A) at (0+0.1*rand,0+0.1*rand);
\coordinate [...] (B) at (1.25+0.1*rand,0.25+0.1*rand);

```

Можно сделать и по-другому: сохранить основные координаты  $(0,0)$  и  $(1.25,0.25)$ , и прибавлять случайную точку  $0.1(\text{rand}, \text{rand})$ . Это означает, что точка  $A$  — точка  $(0,0)$  плюс одна десятая вектора  $(\text{rand}, \text{rand})$ . Аналогично определяется и точка  $B$ . Такие вычисления делает библиотека `calc`, позволяющая использовать специальные координаты, которые начинаются с  $(\$$  и заканчиваются  $\$)$ , а не  $($  и  $)$ . Внутри таких скобок можно задать и линейную комбинацию координат. Заметим, что скобки  $(\$$  и  $\$)$  предназначены, чтобы только указать на такие специальные координаты, и не имеют отношения к набору математических формул. Новый код для координат может быть таким:

```

\begin{tikzpicture}
\coordinate [label=left:\textcolor{blue}{\$A\$}]
(A) at (\$ (0,0) + 0.1*(rand,rand) \$);
\coordinate [label=right:\textcolor{blue}{\$B\$}]
(B) at (\$ (1.25,0.25) + 0.1*(rand,rand)\$);
\draw[color=blue] (A) -- (B);
\end{tikzpicture}

```



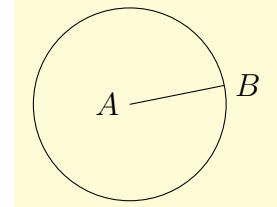
## 4.1.2 Построение окружности

Нарисовать окружность с центром в точке  $A$  и радиусом, равным длине отрезка  $AB$ , не сложно, но об этом мы расскажем позже, а пока пойдем “прямым путем”. Сложность здесь состоит в вычислении длины этого отрезка. Решить задачу можно так: сначала вычислить вектор  $(\$ (A) - (B) \$)$ , затем вычислить длину этого вектора.

Если известны координаты  $x$  и  $y$ , для решения последней задачи можно воспользоваться функцией `veclen(x,y)`, которая возвращает число  $\sqrt{x^2 + y^2}$ .

Единственная проблема состоит в том, как обратиться к  $x$ - и  $y$ -координатам вектора  $(A) - (B)$ ? Для этого, нужно ввести новую операцию: `let`. Ее можно задать где-нибудь в пути, там где обычно определяется путь. Операция `let` вычисляет координаты и передает результат специальному макросу. Этот макрос облегчат доступ к  $x$ - и  $y$ -координатам точки. Таким образом, можно написать следующий код:

```
\begin{tikzpicture}
\coordinate [label=left:$A$] (A) at (0,0);
\coordinate [label=right:$B$] (B) at (1.25,0.25);
\draw (A) -- (B);
\draw (A) let \p1 = ($ (B) - (A) $)
in circle ({veclen(\x1,\y1)});
\end{tikzpicture}
```



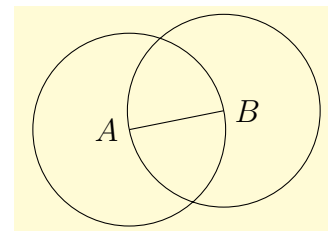
Назначение в `let` начинается с буквы `\p` (от point), обычно сопровождаемой цифрой `<digit>`. Затем пишется знак равенства и координата. Координата вычисляется, а результат сохраняется. Теперь можно использовать следующие выражения:

- 1) `\x<digit>` дает  $x$ -координату получившейся точки;
- 2) `\y<digit>` дает  $y$ -координату получившейся точки;
- 3) `\p<digit>` дает точку  $(\x<digit>, \y<digit>)$ .

В операции `let` можно многократно выполнять допустимые назначения, отделяя их запятыми. В последующих назначениях можно использовать результаты предыдущих. Отметим, что `\p1` не точка в обычном смысле, скорее `\p1` расширяется в строку типа `10pt, 20pt`. Так что нельзя написать, например, `(\p1.center)`, так как такое расширение приводит к `(10pt, 20pt.center)`, что не имеет смысла.

Но надо нарисовать две окружности: вторая с центром в точке  $B$ . Радиус каждой окружности равен `veclen(\x1,\y1)`. Он должен вычисляться один раз, а результат сохраняться, используя операцию `let`: вместо `\p1=`, пишем `\n2=` ( $n$  — от number). При назначении число должно обязательно передаваться в фигурных скобках.

```
\begin{tikzpicture}
\coordinate [label=left:$A$] (A) at (0,0);
\coordinate [label=right:$B$] (B) at (1.25,0.25);
\draw (A) -- (B);
\draw let \p1 = ($ (B) - (A) $),
\n2 = {veclen(\x1,\y1)}
in (A) circle (\n2) (B) circle (\n2);
\end{tikzpicture}
```

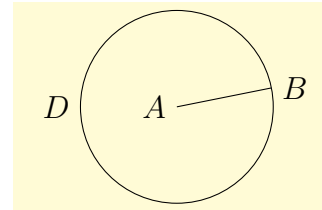


В примере выше, что бы случилось, если бы ввели `\n1`? Ответ: ничего. Макросы `\p`, `\x`, `\y` обращаются к одной и той же логической точке, а макрос `\n` имеет собственное пространство имен. Число после `\n` — обычный Т<sub>Е</sub>X-параметр. Можно было бы указать разумное имя, но тогда нужно использовать фигурные скобки:

```
\draw let \p1 = ($ (B) - (A) $), \n{radius} = {veclen(\x1,\y1)}
in (A) circle (\n{radius}) (B) circle (\n{radius});
```

Но есть более легкий способ нарисовать окружность, если использовать библиотеку `through`. Как подсказывает ее имя, она содержит код для создания форм, проходящих через заданную точку. Опция, нужная в данной ситуации — `circle through=`, которая передается узлу. Узел устанавливает форму в `circle` и, наконец, устанавливает радиус узла таким, чтобы окружность проходила через точку, заданную опцией `circle through`. Этот радиус по существу вычисляется так же, как было сделано выше.

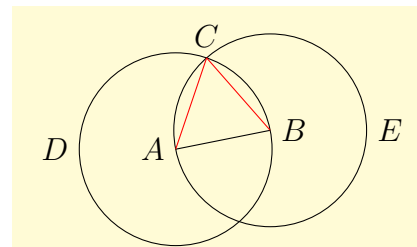
```
\begin{tikzpicture}
\coordinate [label=left:$A$] (A) at (0,0);
\coordinate [label=right:$B$] (B) at (1.25,0.25);
\draw (A) -- (B);
\node [draw,circle through=(B),label=left:$D$]
      at (A) {};
\end{tikzpicture}
```



### 4.1.3 Пересечение окружностей

Найдем точку пересечения двух проведенных окружностей. Вычисление в ручную не тривиально, но библиотека `intersection` позволяет определять точки пересечения произвольных путей: сначала именуются два пути, используя опцию `name path`, затем, в последующем коде используется опция `name intersections=`, которая вычисляет координаты с именами `intersection-1`, `intersection-2`, и так далее, для всех точек пересечения путей. Можно назвать две окружности `D` и `E` и два узла `D` и `E`, при этом ничего не произойдет, так как такие имена находятся в разных пространствах имен.

```
\begin{tikzpicture}
\coordinate [label=left:$A$] (A) at (0,0);
\coordinate [label=right:$B$]
          (B) at (1.25,0.25);
\draw (A) -- (B);
\node(D) [name path=D,draw,
          circle through=(B),
          label=left:$D$] at (A) {};
\node(E) [name path=E,draw,
          circle through=(A), label=right:$E$] at (B) {};
% Именует (но не рисует) точки пересечения
\path [name intersections={of=D and E}];
% Рисует метку и точку пересечения:
\coordinate [label=above:$C$] (C) at (intersection-1);
\draw[red] (A) -- (C); \draw[red] (B) -- (C);
\end{tikzpicture}
```



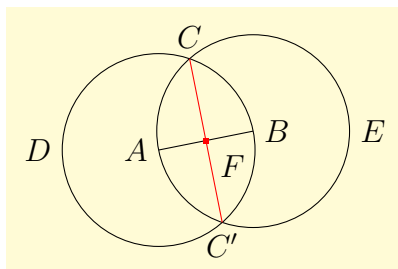
Этот код можно сократить: опция `name intersections` имеет необязательный параметр, позволяющий определять имена для координат и опции для них. Как пример, приведем код, демонстрирующий вычисление точки, делящей отрезка `AB` пополам.

```
\begin{tikzpicture}
\coordinate [label=left:$A$] (A) at (0,0);
```

```

\coordinate [label=right:$B$] (B) at (1.25,0.25);
\draw [name path=A--B] (A) -- (B);
\node (D) [name path=D,draw,circle through=(B),label=left:$D$] at (A) {};
\node (E) [name path=E,draw,circle through=(A),label=right:$E$] at (B) {};
\path [name intersections={of=D and E, by={ [label=above:$C$]C,
                                           [label=below:$C'$]C' }}];
\draw [name path=C--C',color=red] (C) -- (C');
\path [name intersections={of=A--B and C--C',by=F}];
\node [fill=red,inner sep=1pt,label=-45:$F$] at (F) {};
\end{tikzpicture}

```



#### 4.1.4 Полный код примера

Вернемся к поставленной задаче. Введем несколько макроопределений, позволяющих упростить код (макрос `\A`, чтобы рисовать синее  $A$  и так далее), вновь используем уровень `background`, чтобы в самом конце нарисовать оранжевый треугольник.

```

\begin{tikzpicture}[thick,help lines/.style={thin,draw=black!50}]
\def\A{\textcolor{input}{$A$}} \def\B{\textcolor{input}{$B$}}
\def\C{\textcolor{output}{$C$}} \def\D{$D$} \def\E{$E$}
\colorlet{input}{blue!80!black} \colorlet{output}{red!70!black}
\colorlet{triangle}{orange}

\coordinate [label=left:\A] (A) at ($ (0,0) + .1*(rand,rand) $);
\coordinate [label=right:\B] (B) at ($ (1.25,0.25) + .1*(rand,rand) $);

\draw [input] (A) -- (B);

\node [name path=D,help lines,draw,label=left:\D]
      (D) at (A) [circle through=(B)] {};
\node [name path=E,help lines,draw,label=right:\E]
      (E) at (B) [circle through=(A)] {};

\path [name intersections={of=D and E,by={ [label=above:\C]C }}];

\draw [color=orange] (A) -- (C) -- (B);

\foreach \point in {A,B,C} \fill [black,opacity=.5] (\point) circle (2pt);

\begin{pgfonlayer}{background}
  \fill[color=orange!60] (A) -- (C) -- (B) -- cycle;
\end{pgfonlayer}
\end{tikzpicture}

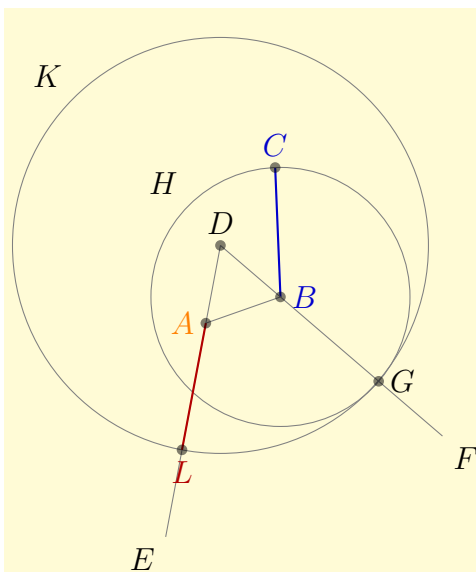
```

```

\mode [below right,text width=10cm,align=justify] at (3.5,3) {
\small\textbf{Предложение I.}\par
\emph{Построить \textcolor{output}{равносторонний треугольник}
на заданном \textcolor{input}{отрезке прямой}.} \par\vskip1em
Пусть  $AB$  --- заданный \textcolor{input}{отрезок прямой}.
Нужно построить \textcolor{output}{равносторонний треугольник},
одна из сторон которого --- отрезок  $AB$ . \par
Для этого построим две окружности: окружность  $D$  с центром в точке
\textcolor{input}{ $A$ } и радиусом, равным длине отрезка  $AB$  и
окружность  $E$  с центром в точке \textcolor{input}{ $B$ } и тем же
радиусом. Эти окружности пересекаются в двух точках. Обозначим одну из
них через \textcolor{output}{ $C$ }. Соединим точку \textcolor{output}{ $C$ }
с точками \textcolor{input}{ $A$ } и \textcolor{input}{ $B$ } прямолинейными
отрезками. Так как  $AB$  и  $AC$  --- радиусы в окружности  $D$ , то их длины
равны. Аналогично, так как  $AB$  и  $BC$  --- радиусы в окружности  $E$ , то
их длины равны. \par
Следовательно, три отрезка  $AB$ ,  $BC$  и  $AC$  имеют одну и ту же длину, и
треугольник  $ABC$  --- \textcolor{output}{равносторонний}.};
\end{tikzpicture}

```

## 4.2 Начала. Книга I. Предложение 2



### Предложение II.

Построить *прямолинейный отрезок*, равный по длине заданному *прямолинейному отрезку*  $BC$ , так, чтобы один из его концов совпал с *заданной точкой*  $A$ .

Пусть  $BC$  — заданный *отрезок прямой* и  $A$  — *заданная точка*. Проведем отрезок  $AB$  и на нем как на основании построим *равносторонний треугольник*  $ABD$ . Проведем окружность  $H$  с центром в точке  $B$  и радиусом  $BC$ .

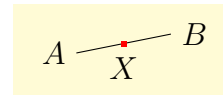
Проведем две прямые: (1) прямую  $F$  через точки  $D$  и  $B$ , (2) прямую  $E$  через точки  $D$  и  $A$ . Проведем окружность  $H$  с центром в точке  $B$  и радиусом  $BC$ . Пусть  $G$  — точка пересечения окружности  $H$  с прямой  $DB$ . Проведем окружность  $K$  с центром в точке  $D$  и радиусом  $DG$ . Пусть  $L$  — точка пересечения окружности  $K$  с прямой  $DA$ .

Длины отрезков  $BC$  и  $BG$  равны как радиусы окружности  $H$ . Длины отрезков  $DL$  и  $DG$  равны как радиусы окружности  $K$ . Треугольник  $ABD$  — *равносторонний*, поэтому длины сторон  $DA$  и  $DB$  равны. Длины отрезков  $AL$  и  $BG$  равны как разности отрезков одной длины. Но тогда равны и длины отрезков  $AL$  и  $BC$ .

### 4.2.1 Другое построение равностороннего треугольника

Чтобы решить задачу, познакомимся с еще одной синтаксической конструкцией TikZ, которая похожа на синтаксис пакета `xcolor`, используемого для смешивания цветов:  $(\$ \langle p \rangle ! n ! \langle q \rangle \$)$ . Это выражение вычисляет точку, расположенную на прямой, проходящей через точки  $p$  и  $q$  по правилу  $(1 - n)\langle p \rangle + n\langle q \rangle$ ; здесь  $n = 0$  — первая точка,  $n = 1$  — вторая точка, любое другое значение  $n \in \mathbb{R}$  строит точку на прямой от  $p$  до  $q$  (если  $n \in (0, 1)$ ), на этой прямой за точкой  $q$  (если  $n > 1$ ) или на прямой перед точкой  $p$  (если  $n < 0$ ). Например, построим точку  $X$ , расположенную посередине отрезка  $AB$ :

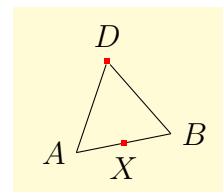
```
\begin{tikzpicture}
\coordinate [label=left:$A$] (A) at (0,0);
\coordinate [label=right:$B$] (B) at (1.25,0.25);
\draw (A) -- (B);
\node [fill=red,inner sep=1pt,label=below:$X$]
      (X) at ($ (A)!0.5!(B) $) {};
\end{tikzpicture}
```



Чтобы построить на  $AB$ , как на основании, равносторонний треугольник, применим метод построения, отличный от метода раздела 4.1 и опирающийся на идею предыдущего абзаца. Рассмотрим прямолинейный отрезок  $XB$ , половину отрезка  $AB$ , и предположим, что мы вращаем его вокруг точки  $X$  на  $90^\circ$ , а затем растягиваем его на  $2 \sin 60^\circ$ , тем самым, получая точку  $D$ . Тогда длина отрезка  $XD$  равна  $|AB| \sin 60^\circ$  (будем длину отрезка обозначать  $|\cdot|$ ), то есть равна высоте равностороннего треугольника с основанием  $AB$ . Чтобы получить равносторонний треугольник, достаточно соединить прямолинейными отрезками точку  $D$  с точками  $A$  и  $B$ .

Для реализации такого вычисления точки  $D$ , нужно указать угол вращения точки. Для этого нужно написать код  $(\$ (X) ! \{\sin(60)*2\} ! 90: (B) \backslash \$)$ . Здесь второй координате предшествует угол поворота. Но точка между  $X$  и  $B$  вычисляется как обычно (как будто угол и не задавался), а затем получившаяся точка вращается на указанный угол вокруг первой точки, то есть  $X$ . Полный код построения равностороннего треугольника, основанный на этой идее выглядит так:

```
\begin{tikzpicture}
\coordinate [label=left:$A$] (A) at (0,0);
\coordinate [label=right:$B$] (B) at (1.25,0.25);
\draw (A) -- (B);
\node [fill=red,inner sep=1pt,label=below:$X$]
      (X) at ($ (A)!0.5!(B) $) {};
\node [fill=red,inner sep=1pt,label=above:$D$]
      (D) at ($ (X) ! \{\sin(60)*2\} ! 90:(B) $) {};
\draw (A) -- (D) -- (B);
\end{tikzpicture}
```

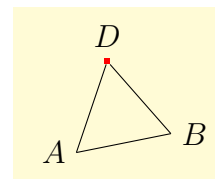


Наконец, уберем точку  $X$ , она для дальнейшего построения не нужна. Это можно сделать, немного изменяя вычисление пути после построения отрезка  $AB$ :

```

\begin{tikzpicture} . . . . .
\node [fill=red,inner sep=1pt,label=above:$D$]
  (D) at ($ (A) ! .5 ! (B) ! {sin(60)*2} ! 90:(B) $) {};
\draw (A) -- (D) -- (B);
\end{tikzpicture}

```



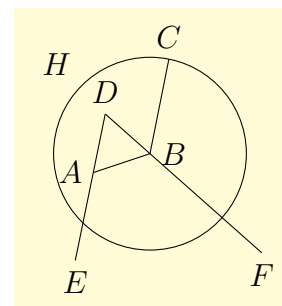
## 4.2.2 Пересечение прямой и окружности

Провести окружность с центром в  $B$  через  $C$  достаточно просто, используя опцию `circle through`. Продолжение отрезков  $DA$  и  $DB$  можно сделать, используя вычисление точки на прямой, но коэффициент оказывается вне отрезка  $[0, 1]$ :

```

\begin{tikzpicture}
\coordinate [label=left:$A$] (A) at (0,0);
\coordinate [label=right:$B$] (B) at (0.75,0.25);
\coordinate [label=above:$C$] (C) at (1,1.5);
\draw (A) -- (B) -- (C);
\coordinate [label=above:$D$] (D) at
($ (A) ! .5 ! (B) ! {sin(60)*2} ! 90:(B) $) {};
\draw (D) -- ($ (D) ! 3.5 ! (B) $) coordinate
  [label=below:$F$] (F);
\draw (D) -- ($ (D) ! 2.5 ! (A) $) coordinate
  [label=below:$E$] (E);
\node (H) [label=135:$H$,draw,circle through=(C)] at (B) {};
\end{tikzpicture}

```

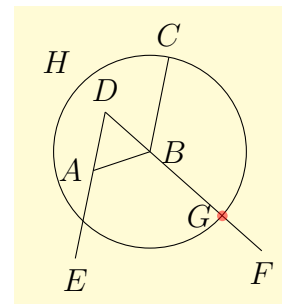


Теперь требуется найти точку  $G$  — точку пересечения прямой  $DF$  с окружностью  $H$ . Один из способов поиска состоит в том, чтобы использовать еще один вариант вычисления точки на прямой. Обычно формула  $(\langle p \rangle ! n ! \langle q \rangle)$  вычисляет точку, расположенную на прямой, проходящей через точки  $p$  и  $q$ , по правилу  $(1 - n)\langle p \rangle + n\langle q \rangle$ . Как альтернатива, вместо множителя  $n$  можно использовать расстояние между точками  $\langle \text{dimension} \rangle$ . В этом случае будет вычислена точка, удаленная на расстояние  $\langle \text{dimension} \rangle$  от точки  $\langle p \rangle$  на прямой к  $\langle q \rangle$ . Мы знаем, что точка  $G$  находится на прямой от  $B$  к  $F$ . Расстояние задается радиусом круга  $H$ . Вот код, вычисляющий точку  $G$ :

```

\begin{tikzpicture} . . . . .
\node (H) [label=135:$H$,draw,circle through=(C)]
  at (B) {};
\path let \p1 = ($ (B) - (C) $) in
  coordinate [label=left:$G$] (G) at
    ($ (B) ! veclen(\x1,\y1) ! (F) $);
\fill[color=red,opacity=.5] (G) circle (2pt);
\end{tikzpicture}

```

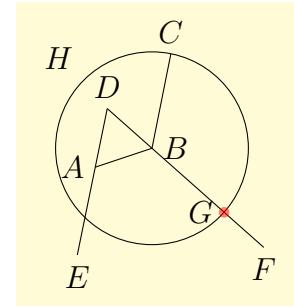


Есть другой способ найти точку  $G$ : присвоить имена окружности и отрезку  $BF$ ; используя опцию `name intersections=`, найти точку пересечения прямой и окружности:

```

\begin{tikzpicture} . . . . .
\node (H) [name path=H,label=135:$H$,
          draw,circle through=(C)] at (B) {};
\path [name path=B--F] (B) -- (F);
\path [name intersections={of=H and B--F,
                          by={label=left:$G$}G}}];
\fill[color=red,opacity=.5] (G) circle (2pt);
\end{tikzpicture}

```



### 4.2.3 Полный код примера

Остается написать код для построения окружности  $K$  с центром в точке  $D$  и радиусом, равным длине отрезка  $DG$ , вычислить точку  $L$  — точку пересечения окружности  $H$  и прямой, проходящей через точки  $D$  и  $A$ , написать сопровождающий рисунок текст.

```

\begin{tikzpicture}[thick,help lines/.style={thin,draw=black!50}]
\def\A{\textcolor{orange}{A}} \def\B{\textcolor{input}{B}}
\def\C{\textcolor{input}{C}} \def\D{D}
\def\E{E} \def\F{F}
\def\G{G} \def\H{H}
\def\K{K} \def\L{\textcolor{output}{L}}

\colorlet{input}{blue!80!black} \colorlet{output}{red!70!black}

\coordinate [label=left:\A] (A) at ($ (0,0) + .1*(rand,rand) $);
\coordinate [label=right:\B] (B) at ($ (1,0.2) + .1*(rand,rand) $);
\coordinate [label=above:\C] (C) at ($ (1,2) + .1*(rand,rand) $);

\draw [input] (B) -- (C);
\draw [help lines] (A) -- (B);

\coordinate [label=above:\D] (D) at ($ (A)!.5!(B) ! {\sin(60)*2} ! 90:(B) $);

\draw [help lines] (D) -- ($ (D)!3.75!(A) $) coordinate [label=-135:\E] (E);
\draw [help lines] (D) -- ($ (D)!3.70!(B) $) coordinate [label=-45:\F] (F);

\node (H) at (B) [name path=H,help lines,circle through=(C),
                 draw,label=135:\H] {};

\path [name path=B--F] (B) -- (F);
\path [name intersections={of=H and B--F,by={label=right:\G}G}}];

\node (K) at (D) [name path=K,help lines,circle through=(G),
                 draw,label=135:\K] {};

\path [name path=A--E] (A) -- (E);
\path [name intersections={of=K and A--E,by={label=below:\L}L}}];

\draw [output] (A) -- (L);

```



```

\foreach \point in {A,B,C,D,G,L} \fill [black,opacity=.5]
                                     (\point) circle (2pt);

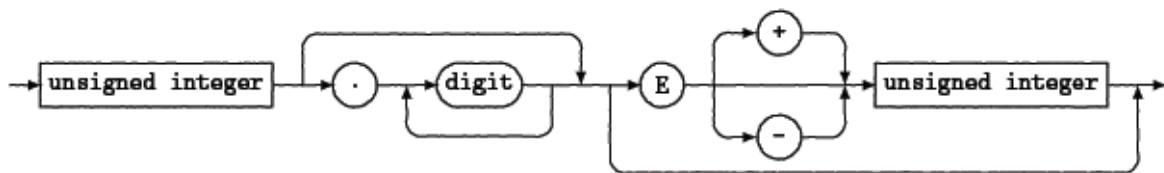
\node [below right, text width=9cm,align=justify] at (4,4)
{\small\textbf{Предложение II.}\par
\emph{Построить \textcolor{input}{прямолинейный отрезок}, равный по длине
данному прямолинейному отрезку  $BC$ , так, чтобы один из его концов
совпадал с \textcolor{orange}{заданной точкой  $A$ }.} \par\vskip1em
Пусть  $BC$ --- заданный \textcolor{input}{отрезок прямой} и
 $A$ --- \textcolor{orange}{заданная точка}. Проведем отрезок  $AB$  и на нем
как на основании построим \textcolor{output}{равносторонний треугольник}
 $ABD$ . Проведем окружность  $HN$  с центром в точке \textcolor{input}{ $B$ } и
радиусом  $BC$ .\par
Проведем две прямые: (1) прямую  $FD$  через точки  $D$  и  $B$ , (2) прямую  $FE$ 
через точки  $D$  и  $A$ . Проведем окружность  $HN$  с центром в точке
\textcolor{input}{ $B$ } и радиусом  $BC$ . Пусть  $G$ --- точка пересечения
окружности  $HN$  с прямой  $DB$ . Проведем окружность  $KN$  с центром в
точке \textcolor{input}{ $D$ } и радиусом  $DG$ . Пусть  $L$ --- точка
пересечения окружности  $KN$  с прямой  $DA$ .\par
Длины отрезков  $BC$  и  $BG$  равны как радиусы окружности  $HN$ . Длины
отрезков  $DL$  и  $DG$  равны как радиусы окружности  $KN$ . Треугольник
 $ABD$ --- \textcolor{output}{равносторонний}, поэтому длины сторон
 $DA$  и  $DB$  равны. Длины отрезков  $AL$  и  $BG$  равны как разности
отрезков одной длины. Но тогда равны и длины отрезков  $AL$  и  $BC$ . };
\end{tikzpicture}

```

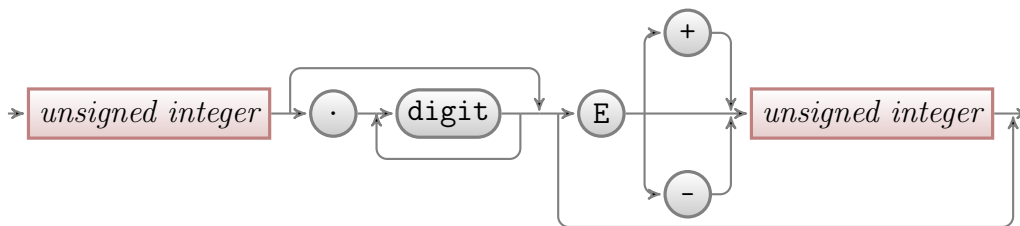
# Глава 5

## Цепочки и матрицы

Рассмотрим задачу построения цепочек и матриц и их последующего использования для построения всевозможных диаграмм. В добрые старые времена, в начале эпохи программирования, диаграммы Бекуса-Наура, например, для языка программирования Паскаль, рисовали, используя перо и линейку, а результат был похож на цепь, представленную на следующем рисунке:



Используя TikZ, можно создать аналогичную, но более привлекательную, цепь:



Чтобы такое нарисовать, нужны будут некоторые библиотеки. Но отложим этот вопрос до тех пор, пока они реально не потребуются.

### 5.1 Стиль узлов

Для построения диаграмм требуются узлы и связи, создающие цепочки. Начнем с определения стилей для узлов. В приведенных выше цепочках есть два вида узлов, которые теоретики любят называть терминалами и нетерминалами. Для терминалов, будем использовать серый цвет, для нетерминалов добавим немного красного. Сначала определим стиль для более простых нетерминалов, границами которых будут прямоугольники.

```

\begin{tikzpicture}
[nonterminal/.style={
  rectangle,                % форма
  minimum size=6mm,         % размер
  very thick,               % граница и ...
  draw=red!50!black!50,     % ... ее цвет
  top color=white,          % Заполнение от белого сверху,
  bottom color=red!50!black!20, % до цветного снизу
  font=\itshape             % Шрифт }]
\node [nonterminal] {unsigned integer};
\end{tikzpicture}

```

Модель терминалов немного сложнее из-за использования скругленных углов. Есть несколько опций, которые позволяют сделать это. Одна из таких опций `rounded corners`. Она получает размеры в качестве параметров и делает скругленными все углы, заменяя их дугами заданного радиуса. Устанавливая в модели радиус круга в 3mm, получим круг, когда форма пуста, а в противном случае — по половине круга на коротких сторонах прямоугольника:

```

\begin{tikzpicture}[node distance=5mm,
terminal/.style={
  rectangle,minimum size=6mm, % форма
  rounded corners=3mm, % скругление углов ...
  very thick,             %... и все остальное
  draw=black!50,
  top color=white,
  bottom color=black!20,
  font=\ttfamily}]
\node (dot) [terminal] {.};
\node (digit) [terminal,right=of dot] {digit};
\node (E) [terminal,right=of digit]{E};
\end{tikzpicture}

```

Другая возможность состоит в том, чтобы использовать форму, которая специально создана для того, чтобы рисовать прямоугольники с дугами на сторонах (для этого нужна библиотека `shapes.misc`). Эта форма предоставляет больше возможностей (например, дугу можно нарисовать только на левой стороне). Пока этой возможностью пользоваться не будем и оставим для терминалов стиль, созданный выше.

Договоримся, далее, для сокращения записи кода примеров не повторять в коде однотипные определения стилей `nonterminal` и `terminal`. Отметим, что когда несколько рисунков используют одни и те же стили, их определения можно указать в преамбуле, используя макрос `\tikzset{...}`, например, `\tikzset {terminal/.style=...}`. Потому будем считать, что для терминальных и нетерминальных узлов сделано именно так.

Вернемся к предыдущему примеру, в котором есть небольшая проблема: базовые линии текстов, расположенные в разных узлах, не совпадают. Чтобы увидеть это, нарисуем три базовые линии:

```

\begin{tikzpicture}[node distance=5mm]
\node (dot) [terminal]          {.};
\node (digit) [terminal,right=of dot] {digit};
\node (E) [terminal,right=of digit]{E};
\draw [help lines] let \p1 = (dot.base),
                \p2 = (digit.base), \p3 = (E.base)
                in (-.5,\y1) -- (3.5,\y1) (-.5,\y2) -- (3.5,\y2)
                (-.5,\y3) -- (3.5,\y3);
\end{tikzpicture}

```



Для `digit` и `E` различие в базовых линиях почти незаметно, но для точки проблема весьма серьезна: точка здесь больше похожа на операцию умножения, чем на точку. Чтобы поправить дело, можно попытаться использовать опцию `base right=of...`, а не опцию `right=of...`

```

\begin{tikzpicture}[node distance=5mm]
\node (dot) [terminal]          {.};
\node (digit) [terminal,base right=of dot] {digit};
\node (E) [terminal,base right=of digit]{E};
\end{tikzpicture}

```



С базовыми линиями текстов в узлах теперь все в порядке, но заплясали сами узлы! Проблему не совпадения базовых линий можно решить, если воспользоваться одной уловкой. Проблема вызвана тем фактом, что тексты `digit` и `E` имеют разную высоту и глубину. Если их уровнять, все станет на свои места. Для этого следует использовать опции `text height` и `text depth`, явно определяя высоту и глубину текста в узлах.

```

\begin{tikzpicture}[node distance=5mm,
                text height=1.5ex,text depth=.25ex]
\node (dot) [terminal]          {.};
\node (digit) [terminal,right=of dot] {digit};
\node (E) [terminal,right=of digit]{E};
\end{tikzpicture}

```



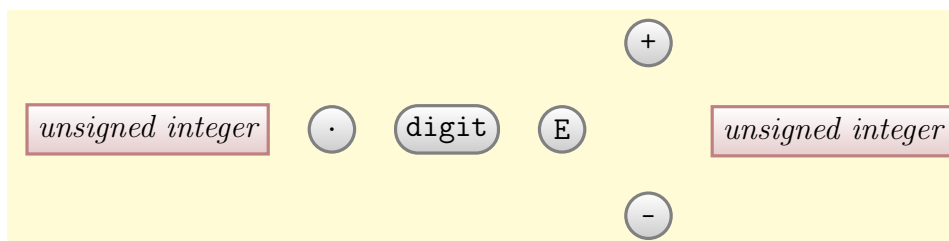
## 5.2 Выравнивание узлов позиционированием

Когда узлы созданы, следующая задача — правильно разместить их друг относительно друга. Для этого есть несколько способов. Самое простое: вручную поместить узлы в определенные точки. Для простых ситуаций такой подход вполне годится, но имеет несколько существенных недостатков:

1. Для большего количества графических объектов, вычисление их местоположения может оказаться непростой задачей.
2. Изменение текста узлов может привести к повторному вычислению их координат.
3. Исходный код рисунка не всегда понятен, если отношения между позициями узлов производятся явно.

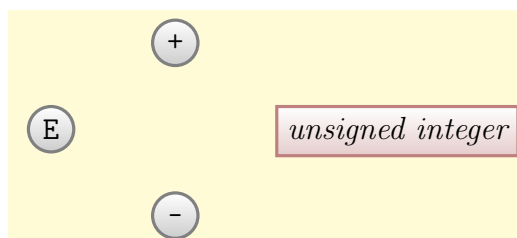
Обсудим другие способы упорядочения узлов на странице. Первый метод — использование опций позиционирования. Их можно использовать, если загрузить библиотеку `positioning`, которая позволяет обращаться к опциям, подобным `above` или `left`, так как теперь можно сказать `above=of <some node>`, чтобы разместить узел выше узла `<some node>`, с границами, удаленными на расстояние `node distance` друг от друга.

```
\begin{tikzpicture}[node distance=5mm and 5mm]
\node (ui1) [nonterminal] {unsigned integer};
\node (dot) [terminal,right=of ui1] {·};
\node (digit) [terminal,right=of dot] {digit};
\node (E) [terminal,right=of digit] {E};
\node (plus) [terminal,above right=of E] {+};
\node (minus) [terminal,below right=of E] {-};
\node (ui2) [nonterminal,below right=of plus] {unsigned integer};
\end{tikzpicture}
```



Горизонтальное и вертикальное расстояния между сторонами прямоугольников, содержащих узлы, здесь равны 5mm, как определено опцией `node distance`. Не нужно ли узлы с + и - сместить вправо? Самый легкий способ сделать это состоит в том, чтобы просто немного увеличить сдвиг по горизонтали вручную:

```
\begin{tikzpicture}[node distance=5mm and 5mm]
\node (E) [terminal] {E};
\node (plus) [terminal,above right=of E,xshift=5mm] {+};
\node (minus) [terminal,below right=of E,xshift=5mm] {-};
\node (ui2) [nonterminal,below right=of plus,xshift=5mm] {unsigned integer};
\end{tikzpicture}
```



Другой способ выравнивания: использовать матрицы. Но об этом позже. Теперь, когда узлы размещены, добавим связи. Одни связи нарисовать просто, другие сложнее. Просто соединить узел с точкой с узлом `digit`: `\path (dot) edge[->] (digit)`.

Сложнее нарисовать ломаную линию вокруг узла `digit`. Чтобы сделать это, следует: начать линию справа от узла `digit`, провести ее вниз, повернуть направо, пройти прямо, повернуть направо, завершить ее слева от узла `digit`. Вот код, реализующий этот алгоритм:

```

\begin{tikzpicture}[node distance=5mm and 5mm]
\node (dot) [terminal] {.};
\node (digit) [terminal,right=of dot] {digit};
\node (E) [terminal,right=of digit]{E};
\path (dot) edge [->] (digit) % простой путь
      (digit) edge[->] (E);
\draw [->] ($(digit.east) + (2mm,0)$)% стать справа от якоря digit.east
  -- ++(0,-.5) % теперь пойти вниз
  -| ($(digit.west) - (2mm,0)$); % прийти слева от якоря digit.west
\end{tikzpicture}

```



Код `\path ... -| <coordinate> ...`; означает: сначала по горизонтали, затем по вертикали, а код `\path ... |- <coordinate> ...`; означает: сначала по вертикали, затем по горизонтали (он ранее использовался в первом рисунке этой главы).

Но похожие пути надо будет рисовать несколько раз, поэтому выглядит заманчиво определить для этого специальный путь `to path` с именем `skip loop`. И всякий раз, когда возникнет команда `edge[skip loop]`, она будет добавлять в путь текущее значение `to path`. Определим стиль, содержащий нужный путь, и получим тот же результат:

```

\begin{tikzpicture}[node distance=5mm and 5mm,
  skip loop/.style={to path={-- ++(0,-.5) -|
    (\tikztotarget)}}]
\node (dot) [terminal] {.};
\node (digit) [terminal,right=of dot] {digit};
\node (E) [terminal,right=of digit]{E};
\path (dot) edge[->] (digit)
      (digit) edge[->] (E) ($ (digit.east) + (2mm,0) $)
      edge[->,skip loop] ($ (digit.west) - (2mm,0)$);
\end{tikzpicture}

```

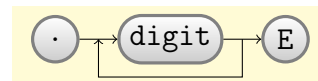


Можно пойти дальше и параметризовать стиль `skip loop`. Для этого, шаг по вертикали передадим в `skip loop` как параметр, используя стандартную для Т<sub>Е</sub>X'a нотацию `#1`. Кроме того, определим начальную и конечную точки пути отдельно, как точки, которые являются «серединой между узлами».

```

\begin{tikzpicture}[node distance=5mm and 5mm,
  skip loop/.style={to path={-- ++(0,#1) -|
    (\tikztotarget)}}]
\node (dot) [terminal] {.};
\node (digit) [terminal,right=of dot] {digit};
\node (E) [terminal,right=of digit]{E};
\path (dot) edge[->] (digit)
      (digit) edge[->] (E)($ (digit.east)!.5!(E.west)$)
      edge[->,skip loop=-5mm] ($ (digit.west)!.5!(dot.east)$);
\end{tikzpicture}

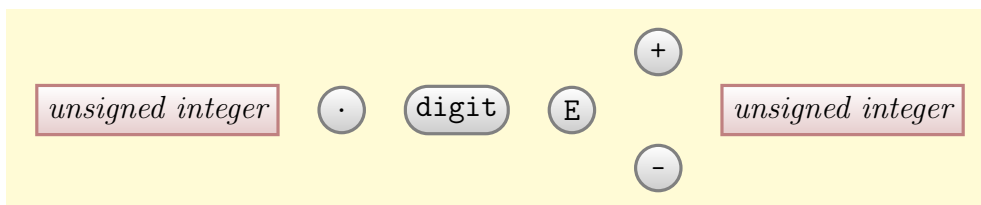
```



## 5.3 Выравнивание узлов, используя матрицы

Возможно лучший способ установить и выровнять узлы состоит в том, чтобы использовать для этого матрицу. В TikZ матрицы можно использовать и для того, чтобы выравнивать графические объекты по строкам и столбцам. Синтаксис матрицы подобен синтаксису массивов и таблиц в Т<sub>Е</sub>X'е (реально используются внутренне таблицы Т<sub>Е</sub>X'а, но дополнительно добавлено много новых возможностей). Для разделения столбцов матрицы будем здесь и далее использовать вместо символа & символы \&. В следующем примере диаграмма будет состоять из трех строк и шести столбцов:

```
\begin{tikzpicture}
\matrix[row sep=1mm,column sep=5mm,ampersand replacement=\&]{
% Первая строка:
\& \& \& \& \node [terminal] {+}; \& \\
% Вторая строка:
\node [nonterminal] {unsigned integer}; \&
\node [terminal] {.}; \&
\node [terminal] {digit}; \&
\node [terminal] {E}; \&
\node [nonterminal] {unsigned integer}; \\
% Третья строка:
\& \& \& \& \node [terminal] {-}; \& \\
};
\end{tikzpicture}
```



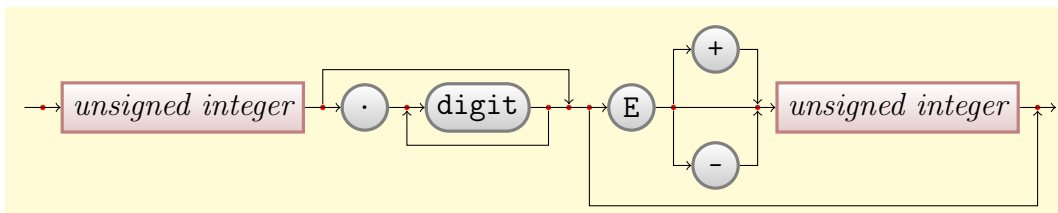
Теперь в матрице нужно связать узлы. На сей раз идея состоит в следующем: добавить небольшие узлы (позже их можно сделать невидимыми) во всех позициях, где будут начинаться и заканчиваться связи. Связи построить так же, как и раньше в разделе 5.2, и получить желаемую диаграмму.

```
\begin{tikzpicture}[
point/.style={circle,inner sep=0pt,minimum size=2pt,fill=red},
skip loop/.style={to path={-- ++(0,#1) -| (\tikztotarget)}}]
\matrix[row sep=1mm,column sep=2mm,ampersand replacement=\&]{
% Первая строка:
\& \& \& \& \& \& \& \& \& \& \& \node [terminal] {+}; \\
% Вторая строка:
\node (p1) [point] {}; \& \node [nonterminal] {unsigned integer}; \&
\node (p2) [point] {}; \& \node [terminal] {.}; \&
\node (p3) [point] {}; \& \node [terminal] {digit}; \&
\node (p4) [point] {}; \& \node (p5) [point] {}; \&
\node (p6) [point] {}; \& \node [terminal] {E}; \&
\node (p7) [point] {}; \&
```

```

\node (p8) [point] {}; \& \node [nonterminal] {unsigned integer}; \&
\node (p9) [point] {}; \&
% Третья строка:
\& \& \& \& \& \& \& \& \& \& \& \& \& \& \node [terminal] {-};\& \& };
\path (p4) edge [->,skip loop=-5mm] (p3)
      (p2) edge [->,skip loop=5mm] (p5)
      (p6) edge [->,skip loop=-13mm] (p9);
\draw [->] (p7) |- (minus.west); \draw [->] (p7) |- (plus.west);
\draw [->] (p8) |- (minus.east); \draw [->] (p8) |- (plus.east);
\path ($ (p1.west) - (2mm,0mm) $) edge[->] (ui1)
      (ui1) edge[->] (dot) (dot) edge[->] (digit)
      (digit) edge[->] (E) (E) edge[->] (ui2)
      (ui2) edge[->] ($ (p9.east) + (2mm,0mm) $);
\end{tikzpicture}

```



## 5.4 Использование библиотеки цепочек

Матрицы позволяют очень просто выравнять узлы, но создание связей требует некоторых усилий. Проблема в том, что код, использующий матрицы, не отражает тот реальный путь, который присутствует на диаграмме. Попробуем решить задачу, используя библиотеку `chains`, предназначенную для построения цепочек. Цепочка — это последовательность связанных узлов. Узлы могут быть уже созданы, или могут создаваться по мере создания цепочки (эти процессы можно смешивать).

### 5.4.1 Создание простой цепочки

Создадим цепочку на пустом месте. Запускает процесс создания цепочки опция `start chain`. Чтобы добавить узел в цепочку, используется опция `on chain` (узлы (+) и (-) добавим позже).

```

\begin{tikzpicture}[start chain,node distance=5mm]
\node [on chain,nonterminal] {unsigned integer};
\node [on chain,terminal] {·};
\node [on chain,terminal] {digit};
\node [on chain,terminal] {E};
\node [on chain,nonterminal] {unsigned integer};
\end{tikzpicture}

```



По умолчанию узлы в цепочке располагаются по горизонтали. Но ситуацию можно изменить, если, например, написать `start chain=going below` и получить каждый следующий узел ниже предыдущего. Чтобы связать дугами все узлы цепочки, следует в каждый узел добавить опцию `join`.



```
\begin{tikzpicture}[start chain,node distance=5mm]
\node [on chain,join,nonterminal] {unsigned integer};
\node [on chain,join,terminal] {.};
\node [on chain,join,terminal] {digit};
\node [on chain,join,terminal] {E};
\node [on chain,join,nonterminal] {unsigned integer};
\end{tikzpicture}
```



Чтобы получить стрелки, нужно переопределить стиль `every join`. Кроме того, чтобы не повторять код, можно переместить опции `join`, `on chain` в стиль `every node`.

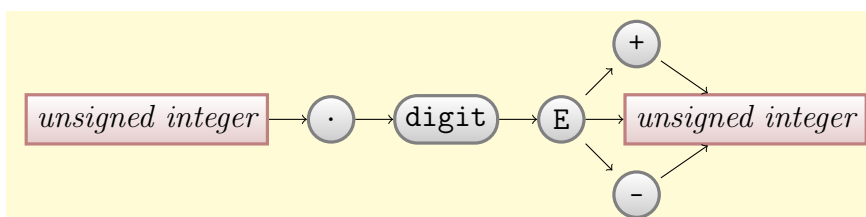
```
\begin{tikzpicture}[start chain,node distance=5mm,
                    every node/.style={on chain,join},
                    every join/.style={->}]
\node [nonterminal] {unsigned integer};
\node [terminal] {.};
\node [terminal] {digit};
\node [terminal] {E};
\node [nonterminal] {unsigned integer};
\end{tikzpicture}
```



### 5.4.2 Ветвление и связывание в цепочке

Чтобы добавить в диаграмму узлы (+) и (-), следует создать ветви от главной цепочки. Ветвление происходит по опции `start branch`.

```
\begin{tikzpicture}[start chain,node distance=5mm,
                    every node/.style={on chain,join}, every join/.style={->}]
\node [nonterminal] {unsigned integer};
\node [terminal] {.};
\node [terminal] {digit};
\node [terminal] {E};
\begin{scope}[start branch=plus]
\node (plus) [terminal,on chain=going above right] {+};
\end{scope}
\begin{scope}[start branch=minus]
\node (minus) [terminal,on chain=going below right] {-};
\end{scope}
\node [nonterminal,join=with plus,join=with minus] {unsigned integer};
\end{tikzpicture}
```



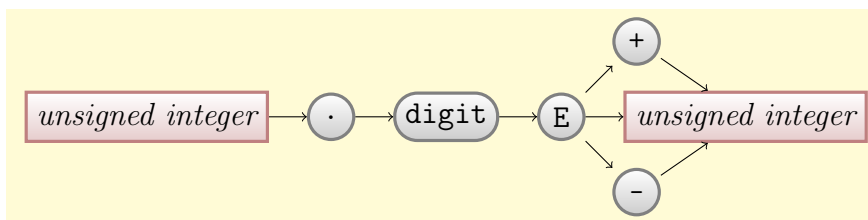
Опция `start branch` начинает ветвь с узла, созданного последним в текущей цепочке, в этом случае с узла `E`. Он будет первым узлом в создаваемой ветви. Ветвь ничем не отличается от цепочки, поэтому узел (в данном случае, узел `plus`) помещается в цепочку, используя опцию `on chain`. В то же время, размещение узла происходит явно, используя опцию `going <direction>`, в данном случае `going above right`, которая помещает узел (+) выше и правее узла `E`. При этом он автоматически связывается с предшествующим узлом неявной опцией `join`.

Когда первая ветвь заканчивается, текущей цепочкой снова становится главная цепочка и мы возвращаемся в узел `E`. Теперь создается новая ветвь для узла (-). После создания этой ветви, мы вновь возвращаемся в узел `E`.

Наконец, в главную цепочку добавляется последний узел, который корректно связывается с узлом `E`. Две дополнительных опции `join=with...`, за которыми следуют имена узлов, позволяют соединить его с узлами, отличными от предшествующего ему узла цепочки.

Если загрузить библиотеку `scopes`, в коде выше можно заменить `\begin{scope}` на `{` и `\end{scope}` на `}`. Кроме того, на узлы (+) и (-) можно ссылаться, используя их автоматически созданные имена: узел `<i>` в цепочке получает имя `chain-<i>`. В ветви `<branch>` узел `<i>` получает имя `chain/<branch>-<i>`. Можно заменить `<i>` на `begin` и `end`, чтобы, соответственно, сослаться на первый и последний узел в цепочке.

```
\begin{tikzpicture}[start chain,node distance=5mm,
  every on chain/.style={join}, every join/.style={->}]
\node [on chain,nonterminal] {unsigned integer};
\node [on chain,terminal]    {.};
\node [on chain,terminal]    {digit};
\node [on chain,terminal]    {E};
{ [start branch=plus]
  \node (plus) [terminal,on chain=going above right] {+}; }
{ [start branch=minus]
  \node (minus) [terminal,on chain=going below right] {-}; }
\node [nonterminal,on chain,join=with chain/plus-end,
  join=with chain/minus-end] {unsigned integer};
\end{tikzpicture}
```



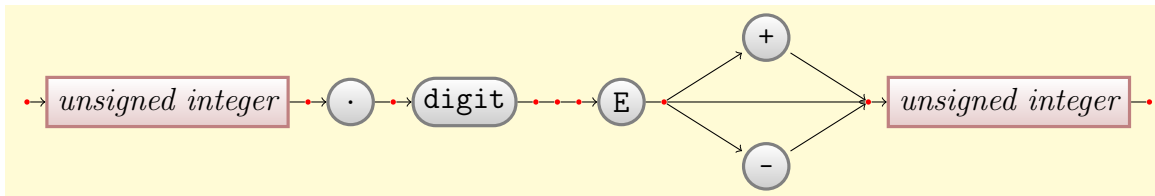
Добавим в цепочку промежуточные координатные узлы (красные точки) так же, как это было сделано в случае использования матрицы. Для этого нужно изменить стиль `join`, поскольку для таких узлов не нужны стрелки. Этого можно добиться или (локально) изменяя стиль `every join` или, что и сделано в примере ниже, задавая желаемый стиль и используя опцию `join=by...`, где `...` — стиль, который следует использовать для соединения.

```
\begin{tikzpicture}[start chain,node distance=5mm and 2mm,
  every node/.style={on chain}, nonterminal/.append style={join=by ->},
```

```

terminal/.append style={join=by ->},
point/.style={join=by -,circle,fill=red, minimum size=2pt,inner sep=0pt}]
\node [point] {}; \node [nonterminal] {unsigned integer};
\node [point] {}; \node [terminal] {.};
\node [point] {}; \node [terminal] {digit};
\node [point] {}; \node [point] {};
\node [point] {}; \node [terminal] {E};
\node [point] {};
{[node distance=5mm and 1cm] % локальное изменение расстояния по горизонтали
  {[start branch=plus]
    \node (plus) [terminal,on chain=going above right] {+}; }
  { [start branch=minus]
    \node (minus) [terminal,on chain=going below right] {-}; }
  \node [point,below right=of plus,join=with chain/plus-end by ->,
    join=with chain/minus-end by ->] {};
}
\node [nonterminal] {unsigned integer};
\node [point] {};
\end{tikzpicture}

```



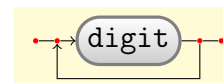
### 5.4.3 Связи в цепочках с уже созданными узлами

Добавим в предыдущую диаграмму недостающие стрелки. Для этого можно использовать ветвление (хорошая практика, отражающая структуру диаграммы в коде). Но поступим по-другому. Начнем с петли вокруг узла `digit`. Такую петлю можно представить как ветвь, которая начинается в точке после узла `digit`, и заканчивается в точке перед узлом `digit`. Однако, нужные точки уже созданы и можно **пристроить цепочку** к уже установленному узлу, используя команду `\chainin`, которая должна сопровождаться координатой, содержащей имя узла и (необязательно) некоторыми опциями. При этом названный узел становится частью текущей цепочки.

```

\begin{tikzpicture}[start chain
  % Определение стилей ... ]
\node [point] {};
\node (before digit) [point] {};
\node [terminal] {digit};
\node [point] {};
{ [start branch=digit loop]
  \chainin (before digit) [join=by {skip loop=-5mm, ->}]; }
\node [point] {};
\end{tikzpicture}

```

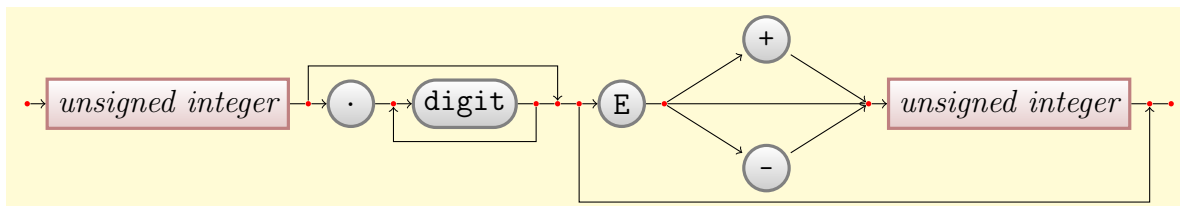


Пользуясь идеей последнего примера, определим все оставшиеся связи и завершим построение диаграммы.

```

\begin{tikzpicture}[start chain,node distance=5mm and 2mm,
  skip loop/.style={to path={ -- ++(0,#1) -| (\tikztotarget)}}},
  every node/.style={on chain}, nonterminal/.append style={join=by ->},
  terminal/.append style={join=by ->},
  point/.style={join=by -,circle,fill=red,minimum size=2pt,inner sep=0pt}]
\node (p1) [point] {}; \node [nonterminal] {unsigned integer};
\node (p2) [point] {}; \node [terminal] {.};
\node (p3) [point] {}; \node [terminal] {digit};
\node (p4) [point] {};
  { [start branch=digit loop]
    \chainin (p3) [join=by {skip loop=-5mm,->}]; }
\node (p5) [point] {};
  { [start branch=dot and digit loop]
    \chainin (p2) [join=by {skip loop=5mm,<-}]; }
\node (p6) [point] {}; \node [terminal] {E};
\node (p7) [point] {};
  { [node distance=5mm and 1cm] % local change in horizontal distance
    { [start branch=plus]
      \node (plus) [terminal,on chain=going above right] {+}; }
    { [start branch=minus]
      \node (minus) [terminal,on chain=going below right] {-}; }
    \node (p8) [point,below right=of plus,join=with chain/plus-end by ->,
      join=with chain/minus-end by ->] {};
  }
\node [nonterminal] {unsigned integer}; \node (p9) [point] {};
  { [start branch=finish loop]
    \chainin (p6) [join=by {skip loop=-13mm,<-}]; }
\node (p10) [point] {};
\end{tikzpicture}

```



#### 5.4.4 Совместное использование матриц и цепочек

Матрицы и цепочки можно объединить следующим образом: использовать матрицу, чтобы установить и выровнять узлы. Однако, чтобы показать в динамике логическую структуру диаграммы, следует создавать цепочки и ветви, показывающие, как происходит движение по диаграмме. Начинать надо с матрицы, применяя ее как и ранее, но следует немного изменить стили, а затем создать главную цепочку и ее ветви. В заключение приведем полный код рисунка, воспроизведенного в начале этой главы, в котором используются и матрицы (с разделителем &), и цепочки.

```

\begin{tikzpicture}[
  skip loop/.style={to path={ -- ++(0,#1) -| (\tikztotarget)}}},
  point/.style={coordinate},>=stealth',thick,draw=black!50,

```

```

tip/.style={->,shorten >=1pt}, every join/.style={rounded corners},
hv path/.style={to path={|-| (\tikztotarget)}},
vh path/.style={to path={|- (\tikztotarget)}}]
\matrix[column sep=2.5mm,row sep=4mm]{
% Первая строка матрицы:
& & & & & & & & & & \node (plus) [terminal] {+};\\
% Вторая строка матрицы:
\node (p1) [point] {}; & \node (ui1) [nonterminal] {unsigned integer}; &
\node (p2) [point] {}; & \node (dot) [terminal] {.}; &
\node (p3) [point] {}; & \node (digit) [terminal] {digit}; &
\node (p4) [point] {}; & \node (p5) [point] {}; &
\node (p6) [point] {}; & \node (e) [terminal] {E}; &
\node (p7) [point] {}; & &
\node (p8) [point] {}; & \node (ui2) [nonterminal] {unsigned integer}; &
\node (p9) [point] {}; & \node (p10) [point] {};\\
% Третья строка матрицы:
& & & & & & & & & & \node (minus)[terminal] {-};\\
}; % Построение матрицы завершено. Строим связи . . .
{ [start chain]
  \chainin (p1);
  \chainin (ui1) [join=by tip];
  \chainin (p2) [join];
  \chainin (dot) [join=by tip];
  \chainin (p3) [join];
  \chainin (digit) [join=by tip];
  \chainin (p4) [join];
  { [start branch=digit loop]
    \chainin (p3) [join=by {skip loop=-6mm,->}];} % 1-ая ветвь
  \chainin (p5) [join,join=with p2 by {skip loop=6mm,tip}];
  \chainin (p6) [join];
  \chainin (e) [join=by tip];
  \chainin (p7) [join];
  { [start branch=plus]
    \chainin (plus) [join=by {vh path,tip}];
    \chainin (p8) [join=by {hv path,tip}];      } % 2-ая ветвь
  { [start branch=minus]
    \chainin (minus) [join=by {vh path,tip}];
    \chainin (p8) [join=by {hv path,tip}];      } % 3-ая ветвь
  \chainin (p8) [join];
  \chainin (ui2) [join=by tip];
  \chainin (p9) [join,join=with p6 by {skip loop=-15mm,tip}];
  \chainin (p10) [join=by tip];
} % Построение связей завершено
\end{tikzpicture}

```

# Глава 6

## Дерево для расписания лекций

### 6.1 Постановка задачи

Предположим, что надо прочитать курс лекций по новой для студентов дисциплине и сразу же дать им краткий обзор того, о чем рассказывается в курсе. Для этого предлагается построить некоторое дерево (или граф), содержащее фундаментальные понятия курса. Такое дерево далее будем называть *картой курса*. Создадим карту курса со следующими характеристиками:

1. Карта должна содержать дерево (или граф), отображающее понятия курса.
2. Карта должна, так или иначе, визуализировать те лекции, которые будут читаться. Отметим, что лекции не обязательно то же самое, что и карта курса: карта может содержать больше понятий, чем будет прочитано в лекциях, а некоторые понятия могут относиться более чем к одной лекции.
3. Карта курса должна содержать календарь, показывающий, когда будет читаться каждая лекция.
4. Карта курса должна иметь визуально хороший и богатый информационный фон.

Как всегда, необходимо подключить нужные библиотеки в среду `tikz`. Потребуется библиотека `mindmap`, определяющая дополнительные стили и опции для создания специальных диаграмм, `calendar`, позволяющая вычислять и строить календари, `backgrounds`, позволяющая рисовать на фоновом уровне, и `shadows`, вводящая новые стили, которые позволяют добавлять в путь или узел частично прозрачные тени.

### 6.2 Понятие дерева

Сначала надо решить, представлять ли все понятия в виде дерева с корнем, ветвями и листьями, или организовать их в виде общего графа. Дерево лучше позволяет представить понятия, в то время как граф более гибок. Поэтому наилучшее решение — компромисс: в основном понятия будут организованы в виде дерева, однако, на различных уровнях или ветвях дерева выборочно могут появляться такие связи между понятиями, которые приводят к графу. Начнем с древовидного списка понятий, которые важны в курсе Computational Complexity (Вычислительная сложность):

- Computational Problems (Вычислительные Проблемы)
  - Problem Measures (Проблема измерений)
  - Problem Aspects (Проблема аспектов)

- Problem Domains (Проблема доменов)
- Key Problems (Ключевые проблемы)
- Computational Models (Вычислительные модели)
  - Turing Machines (Машины Тьюринга)
  - Random-Access Machines (Машины произвольного доступа)
  - Circuits (Схемы)
  - Binary Decision Diagrams (Бинарные схемы принятия решений)
  - Oracle Machines (Oracle-машины)
  - Programming in Logic (Логическое программирование)
- Measuring Complexity (Измерение сложности)
  - Complexity Measures (Меры сложности)
  - Classifying Complexity (Классификация сложности)
  - Comparing Complexity (Сравнение сложности)
  - Describing Complexity (Описание сложности)
- Solving Problems (Решение задач)
  - Exact Algorithms (Точные алгоритмы)
  - Randomization (Рандомизация)
  - Fixed-Parameter Algorithms (Алгоритмы с фиксированным параметром)
  - Parallel Computation (Параллельное вычисление)
  - Partial Solutions (Частичные решения)
  - Approximation (Приближение)

Конечно, это начальный список, который в последующем может уточняться и изменяться. В него надо будет добавить множество подтем (например, в теме **Classifying Complexity** существует достаточно много классов сложности).

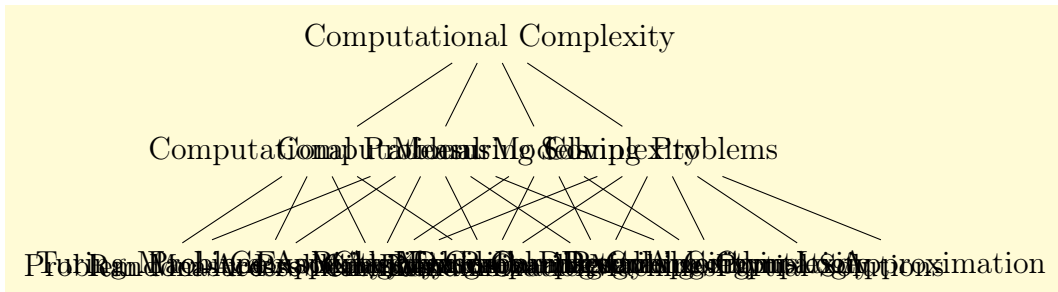
В принципе, превращение списка тем в TikZ-дерево достаточно просто. Основная идея дерева в том, что узел (родитель) может иметь узлы-потомки (или дочерние узлы), которые, в свою очередь, могут иметь узлы-потомки, и так далее. Чтобы добавить потомка в узел, надо просто написать `child {<node>}` правее существующего узла, при этом `<node>`, в свою очередь, должен быть кодом для создания нового узла. Чтобы добавить следующий узел, нужно написать `child {<node>}` еще раз, и так далее. Как первый опыт, напишем такой код:

```
\tikz
\node {Computational Complexity} % корень
  child { node {Computational Problems}
    child { node {Problem Measures} }
    child { node {Problem Aspects} }
    child { node {Problem Domains} }
    child { node {Key Problems} }
  }
  child { node {Computational Models}
    child { node {Turing Machines} }
    child { node {Random-Access Machines} }
    child { node {Circuits} }
    child { node {Binary Decision Diagrams} }
    child { node {Oracle Machines} }
```

```

    child { node {Programming in Logic} }
    }
child { node {Measuring Complexity}
  child { node {Complexity Measures} }
  child { node {Classifying Complexity} }
  child { node {Comparing Complexity} }
  child { node {Describing Complexity} }
  }
child { node {Solving Problems}
  child { node {Exact Algorithms} }
  child { node {Randomization} }
  child { node {Fixed-Parameter Algorithms} }
  child { node {Parallel Computation} }
  child { node {Partial Solutions} }
  child { node {Approximation} }
  };

```



Ошибки этого решения очевидны:

1. Произошло перекрытие узлов вследствие того, что TikZ не отслеживает размещение дочерних узлов. Даже притом, что можно конфигурировать TikZ так, чтобы он использовал «умные» методы размещения, в TikZ нет способов, позволяющих принимать во внимание размеры дочерних узлов. Это может показаться странным, но причина в том, что дочерние узлы задаются и размещаются по одному, таким образом, размер последнего узла не известен, когда обрабатывается первый узел. Таким образом, требуется как-то определить соответствующий уровень и расстояние до соседнего узла на том же уровне «вручную».
2. Стандарт построения дерева сверху вниз явно не подходит для визуализации понятий. Было бы лучше или повернуть карту на 90° или, еще лучше, использовать некоторый вид кругового расположения.

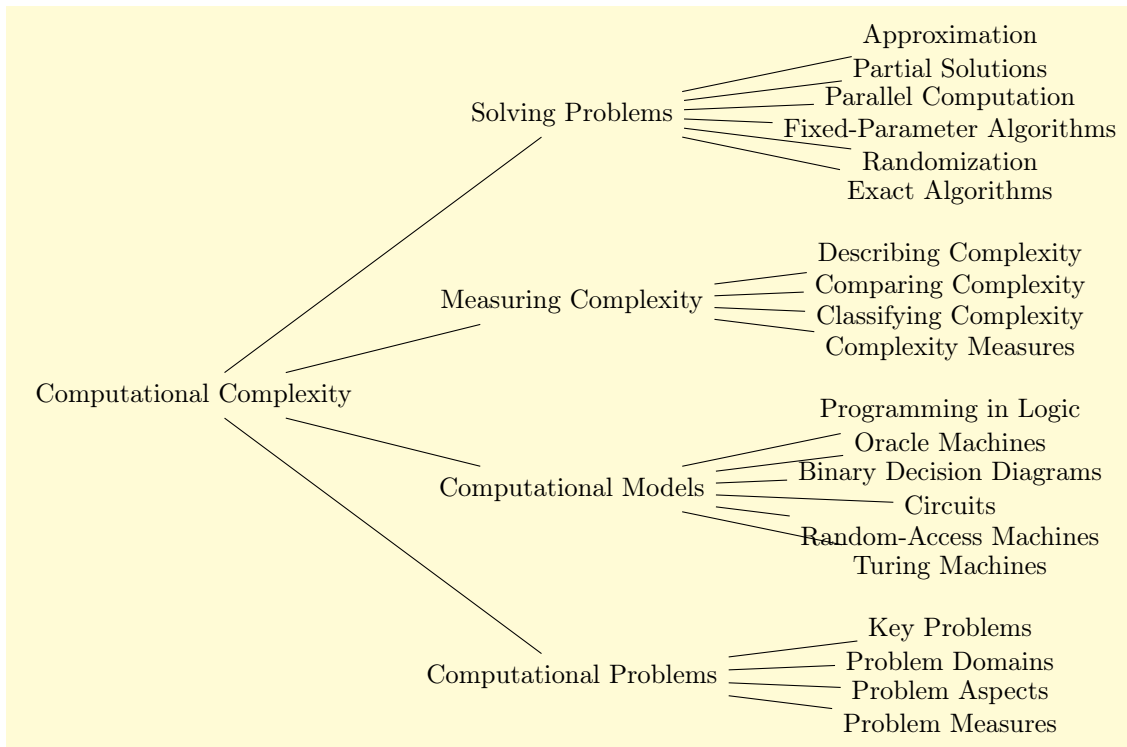
Перерисуем дерево, но на сей раз с некоторым набором опций, смысл которых почти очевиден:

```

\tikz [font=\footnotesize,grow=right,
  level 1/.style={sibling distance=6em},
  level 2/.style={sibling distance=1em},
  level distance=5cm]
\node {Computational Complexity} % корень
  child { node {Computational Problems}
    child { node {Problem Measures} }
    child { node {Problem Aspects} }
  }
% . . . и так далее, как и в коде выше . . .

```





Для построения дерева важны два параметра:

- **level distance** — указывает TikZ расстояние между центрами узлов на смежных уровнях дерева;
- **sibling distance** — указывает TikZ расстояние между центрами элементов одного уровня дерева.

Можно глобально установить для дерева эти параметры, устанавливая их где-нибудь перед началом построения дерева, но, обычно, эти параметры разные для разных деревьев. В этом случае использован стиль `level 1` для первого уровня дерева и стиль `level 2` для второго уровня дерева. Число стилей для уровней можно продолжить.

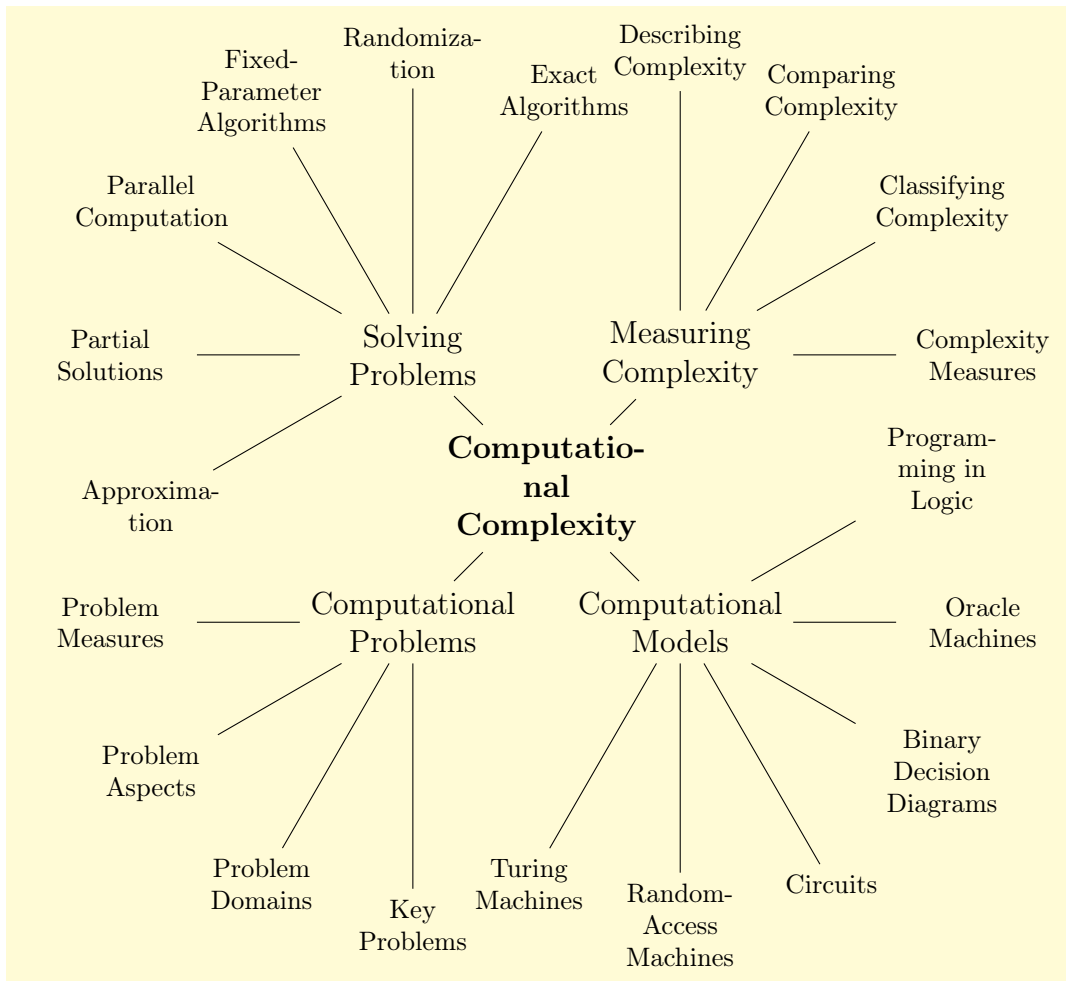
Устанавливать опции расстояния можно только для конкретных узлов, передавая эти параметры команде `child` как опции. (Отметим, что опции команды `node` локальны и не оказывают никакого действия на дочерние узлы. Однако можно определять опции, которые воздействуют на дочерние узлы. Но определение таких опций — тайное искусство, и, если оно действительно нужно, следует изучить раздел 15.4.)

Параметр `grow` позволяет указать направление роста дерева, которое можно изменить в любом месте дерева, просто изменяя значение этого параметра для единственного потомка или для целого уровня. Если подключена библиотека `tree`, появляется доступ к дополнительным стратегиям роста дерева, типа `cyclic` (по кругу):

```

\tikz [text width=2.7cm, align=flush center, grow cyclic,
       level 1/.style={level distance=2.5cm, sibling angle=90},
       level 2/.style={text width=2cm, font=\footnotesize,
                       level distance=4cm, sibling angle=30}]
\node[font=\bfseries] {Computational Complexity} % корень
  child { node {Computational Problems}
    child { node {Problem Measures} }
  }
% . . . и далее, как и выше . . .

```



Можно обращаться к узлу дерева и управлять им как обычным узлом. В частности, можно назвать узел, используя опцию `name=` или нотацию `<name>`, можно также использовать любую доступную форму или стиль для узлов деревьев. Можно связывать узлы дерева, используя синтаксис `\draw (some node) -- (another node);`. В основном, команда `child` только вычисляет соответствующую позицию узла и добавляет линию от потомка к родительскому узлу.

### 6.3 Создание карты курса

Следующий шаг: сделать карту курса более привлекательной. Для этого потребуется библиотека `mindmap`, определяющая множество дополнительных стилей, поддерживающих создание дерева «привлекательного вида».

Добавим в дерево одну из опций: `mindmap`, `big mindmap` или `huge mindmap`. Эти опции дают один и тот же эффект, но различные размеры самих узлов и шрифтов в них (по возрастанию).

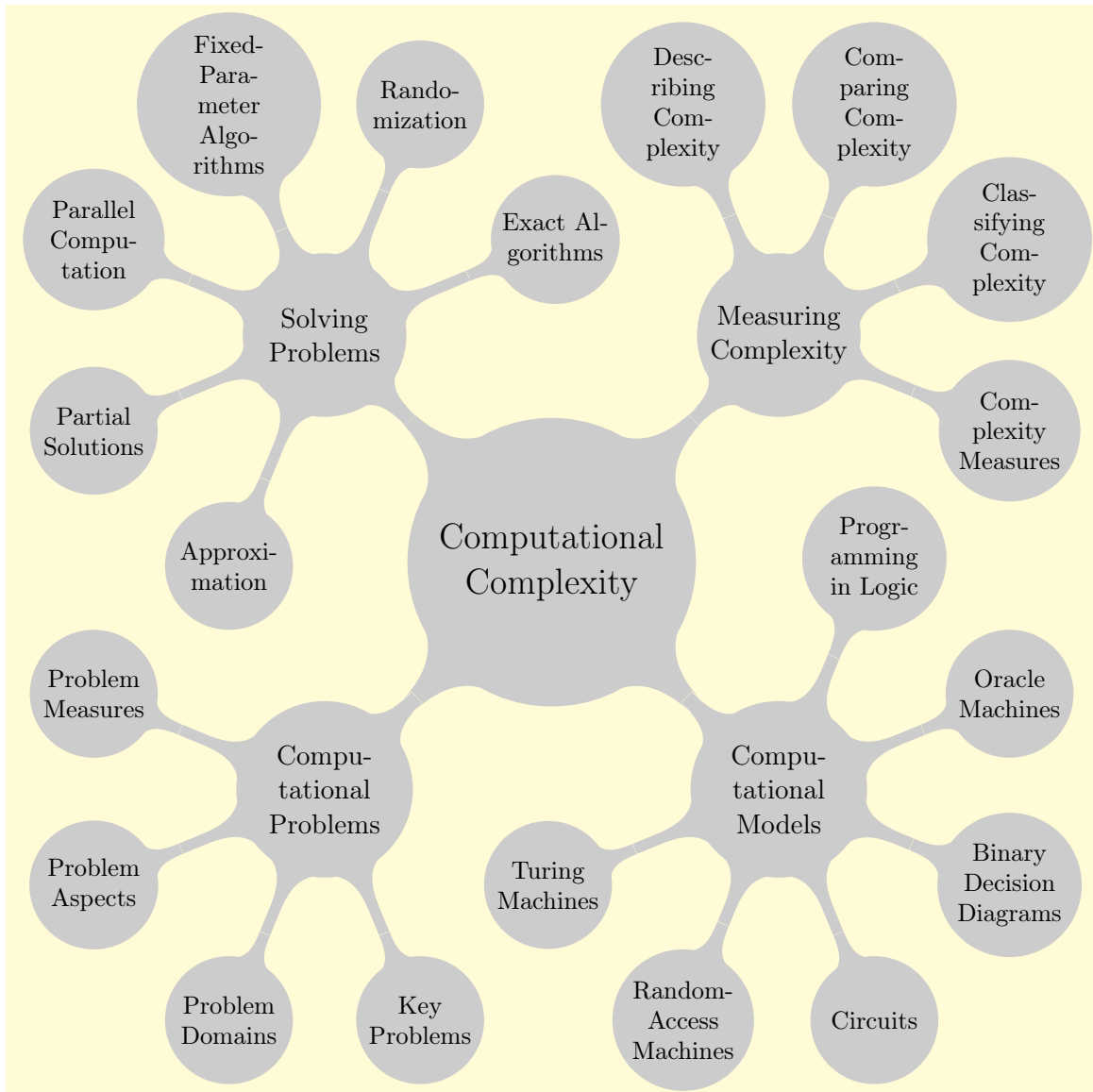
Определим единый стиль узлов `concept`, а затем добавим опцию `concept` к каждому узлу, который реально должен соответствовать этому стилю. Идея состоит в том, что одни узлы дерева будут соответствовать выбранной концепции, а другие узлы могут оставаться простыми потомками родительского узла.

Наконец, определим угол поворота (не расстояние!) для родственных узлов одного уровня, при переходе от одного уровня к другому.

```

\tikz [shape=circle,mindmap, every node/.style=concept,
      concept color=black!20, grow cyclic,
      level 1/.append style={level distance=4.5cm,sibling angle=90},
      level 2/.append style={level distance=3.5cm,sibling angle=45}]
\node [root concept] {Computational Complexity} % корень
  child { node {Com\pu\ta\ti\o\n\al Problems}
    child { node {Problem Measures} }
    % . . . и так далее, как и выше . . .

```

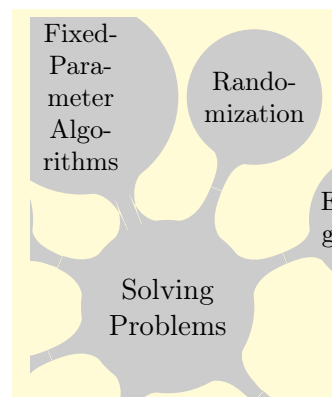


При создании такой карты,  $\text{T}_\text{E}_\text{X}$  (законно) может начать жаловаться на переполнение полей, так как слова типа `Randomization` не помещаются в круг. Чтобы он их переносил можно расставить в словах символ принудительного переноса. Но почему  $\text{T}_\text{E}_\text{X}$  сам не переносит такие слова? Причина в том, что  $\text{T}_\text{E}_\text{X}$  никогда не переносит первое слово абзаца, поскольку он начинает искать `hyphenatable` символы только после так называемого клея. Чтобы заставить  $\text{T}_\text{E}_\text{X}$  переносить первые слова, нужно его обмануть: вставить `\hskip0pt` перед таким словом, вставляя перед ним (невидимый) клей и, таким образом, позволить  $\text{T}_\text{E}_\text{X}$ ’у его переносить. Чтобы не определять одно и то же в каждом узле, можно это сделать один раз с помощью опции узла `execute at begin node=\hskip0pt`.

Для того чтобы выделить и показать только части карты курса, сокращая исполь-

зуемое пространство на странице, следует использовать прямоугольник отсечения. Так и будем поступать в следующих примерах, вплоть для конца этой главы.

```
\begin{tikzpicture}
  [shape=circle,mindmap,
   every node/.style={concept,
                      execute at begin node=\hskip0pt},
   concept color=black!20, grow cyclic,
   level 1/.append style=
     {level distance=4.5cm,sibling angle=90},
   level 2/.append style=
     {level distance=3cm,sibling angle=45}]
\clip (-1,2) rectangle ++ (-4,5);
\node[root concept] {Computational Complexity}
  child { node {Computational Problems}
    child { node {Problem Measures} }
  }
  % . . . и так далее, как и выше . . .
\end{tikzpicture}
```



Теперь раскрасим карту курса, используя разные краски для разных частей дерева. Тогда во время лекции можно вести разговор о зеленых или красных темах. Выберем для узла *Computational Problems* красный цвет, для узла *Solving Problems* — зеленый, для узла *Measuring Complexity* — оранжевый, а для узла *Computational Models* — синий.

Чтобы установить цвет узла, можно использовать опцию `concept color`, а не только, скажем, выражение `node [fill=red]`. Последнее сделает узел красным, но только узел, а не области, связанные с его родителем и его потомками. В то же время, специальная опция `concept color` не только установит цвет узла и его потомков, но также (как по волшебству) создаст соответствующие цветовые переходы так, чтобы цвет родительского узла гладко переходил в цвет потомка.

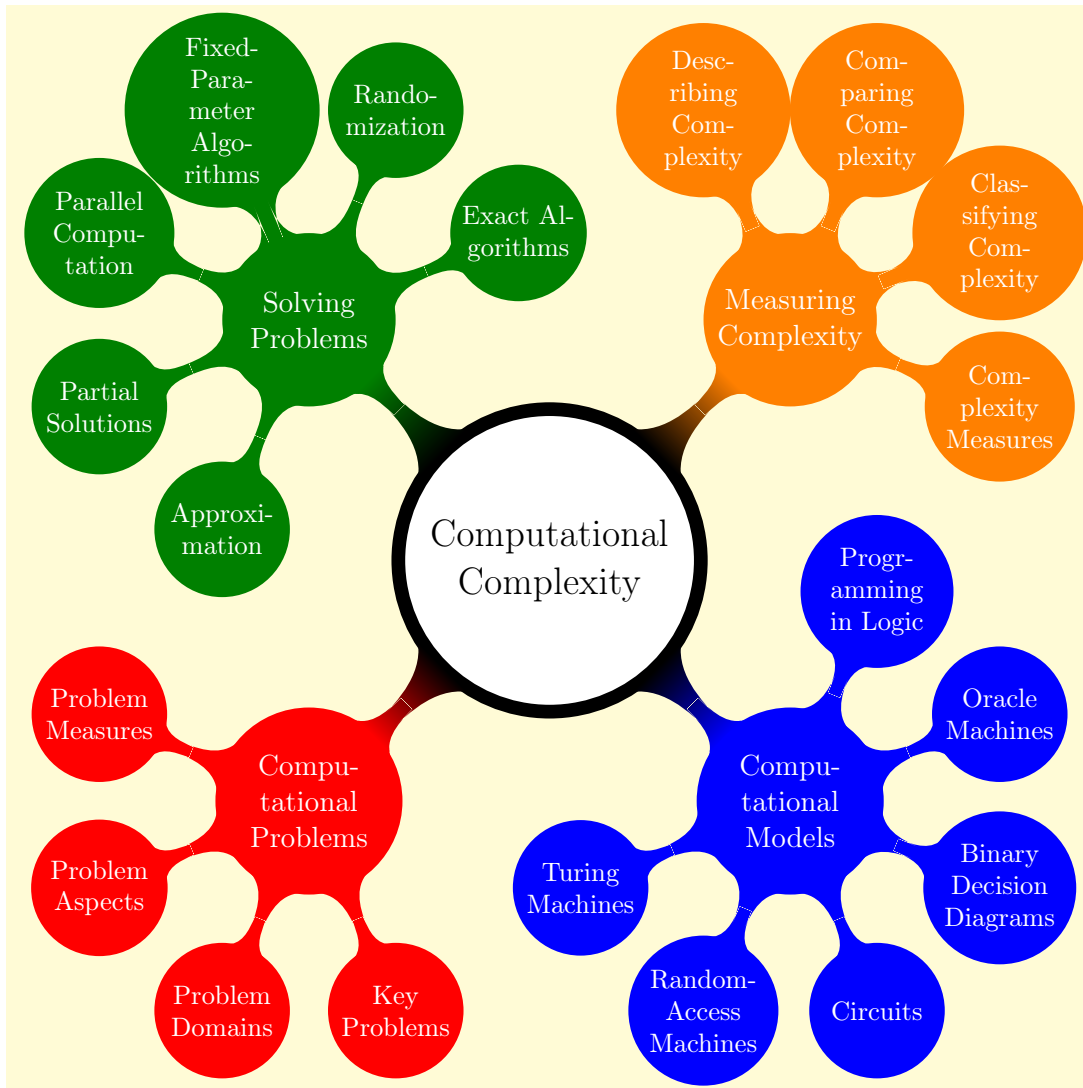
Для корневого понятия сделаем нечто специальное: установим цвет понятия в черный, установим большую ширину линий, а цвет заполнения определим как белый. После этого корневое понятие будет окружено толстой черной линией, а потомки будут связаны с центральным понятием через небольшие области.

```
\begin{tikzpicture} [shape=circle,mindmap,text=white, grow cyclic,
  every node/.style={concept, execute at begin node=\hskip0pt},
  root concept/.append style={concept color=black,fill=white,
                              line width=1ex, text=black},
  level 1/.append style={level distance=4.5cm,sibling angle=90},
  level 2/.append style={level distance=3cm,sibling angle=45}]
%\clip (0,-1) rectangle ++(4,5);
\node [root concept]
  {Computational Complexity}
  child [concept color=red]
    {node {Computational Problems}
      child { node {Problem Measures} }
      % . . . и так далее, как и выше . . . }
  child [concept color=blue]
    {node {Computational Models}}
```

```

    child { node {Turing Machines} }
    % . . . и так далее, как и выше . . . }
child [concept color=orange]
  {node {Measuring Complexity}
    child { node {Complexity Measures} }
    % . . . и так далее, как и выше . . . }
child [concept color=green!50!black]
  {node {Solving Problems}
    child { node {Exact Algorithms} }
    % . . . и так далее, как и выше . . . } };
\end{tikzpicture}

```



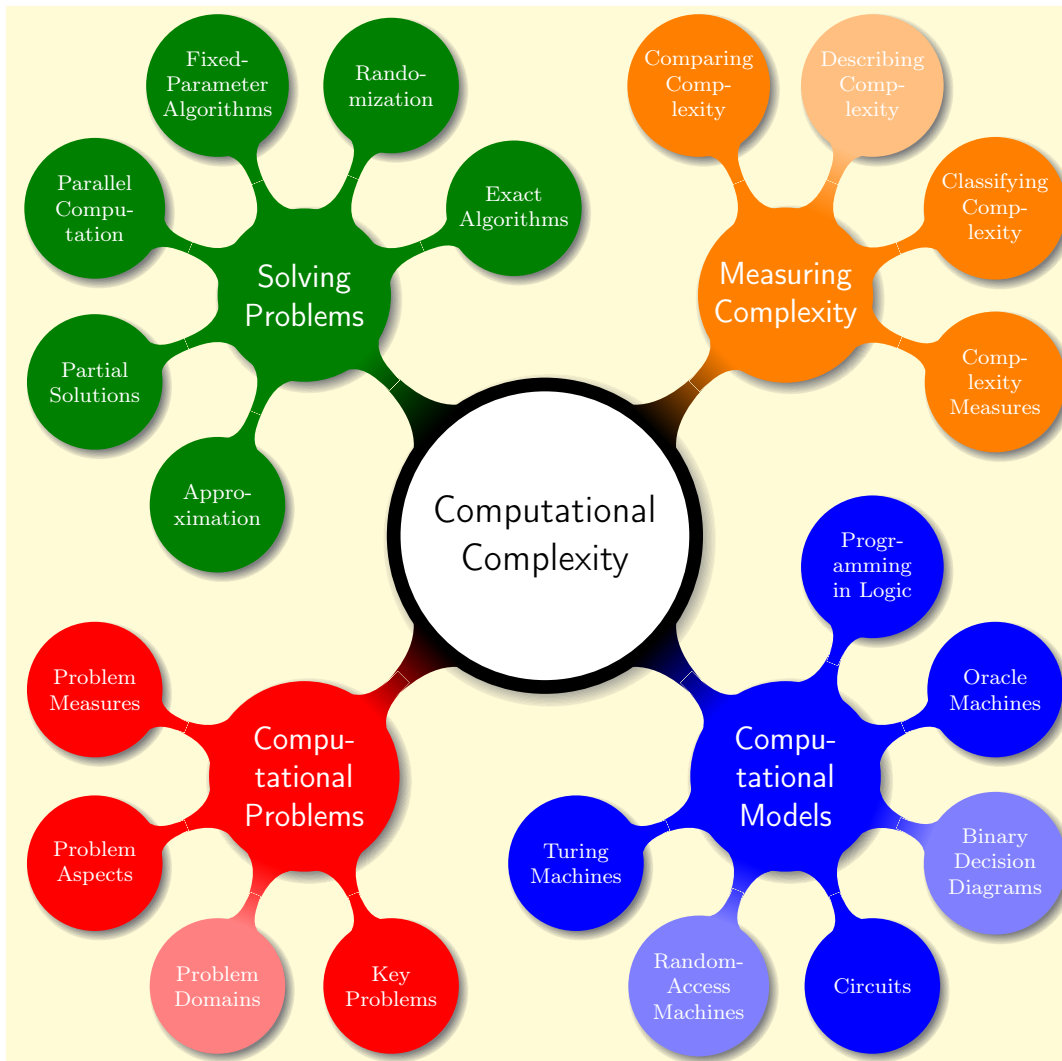
Внесем еще три изменения. Первое: изменим шрифт основных понятий на `sans serif`. Второе: те понятия, которые в принципе важны и принадлежат карте курса, но о которых не будет ничего говориться в лекциях, сделаем менее яркими и, тем самым, выделим их. Для этого определим четыре стиля, по одному для каждого из четырех основных переходов. Эти стили (1) устанавливают правильный цвет понятия для всей ветви, (2) определяют менее яркий цвет для соответствующей ветви. Третье: добавим опцию `a circular drop shadow`, определенную в библиотеке `shadows`, для понятий, чтобы сделать цветовые переходы более причудливыми.

```
\begin{tikzpicture}[shape=circle,mindmap]
```

```

\begin{scope}[text=white,grow cyclic,
  every node/.style={concept, circular drop shadow,
    execute at begin node=\hskip0pt},
  root concept/.append style={concept color=black,fill=white,
    line width=1ex, text=black, font=\large\sffamily},
  computational problems/.style={concept color=red,
    faded/.style={concept color=red!50}},
  computational models/.style={concept color=blue,
    faded/.style={concept color=blue!50}},
  measuring complexity/.style={concept color=orange,
    faded/.style={concept color=orange!50}},
  solving problems/.style={concept color=green!50!black,
    faded/.style={concept color=green!50!black!50}},
  level 1/.append style={level distance=4.5cm,
    sibling angle=90,font=\sffamily},
  level 2/.append style={level distance=3cm,
    sibling angle=45,font=\scriptsize}]
\node [root concept] {Computational Complexity} % root
  child [computational problems] { node {Computational Problems}
    child { node {Problem Measures} }
    child { node {Problem Aspects} }
    child [faded] { node {Problem Domains} }
    child { node {Key Problems} }
  }
  child [computational models] { node {Com\pu\ta\ti\o\n\al Models}
    child { node {Turing Machines} }
    child [faded] { node {Random-Access Machines} }
    child { node {Circuits} }
    child [faded] { node {Binary Decision Diagrams} }
    child { node {Oracle Machines} }
    child { node {Pro\g\r-am\ming in Logic} }
  }
  child [measuring complexity] { node {Measuring Complexity}
    child { node {Com\p\le\x\i\ty Measures} }
    child { node {Clas\si\fy\ing Com\p\le\x\i\ty} }
    child [faded] { node {Des\c\ri\b\ing Com\p\le\x\i\ty} }
    child { node {Com\pa\r\ing Com\p\le\x\i\ty} }
  }
  child [concept color=green!50!black] { node {Solving Problems}
    child { node {Exact Al\go\r\ithms} }
    child { node {Ran\do\mi\za\ti\on} }
    child { node {Fixed-Parameter Al\go\r\ithms} }
    child { node {Parallel Com\pu\ta\ti\on} }
    child { node {Partial Solutions} }
    child { node {Ap\pro\xi\ma\ti\on} }
  };
\end{scope}
\end{tikzpicture}

```



## 6.4 Добавление аннотаций к лекциям

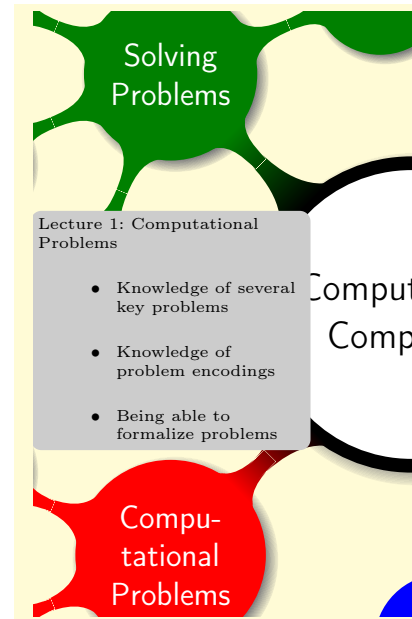
Курс содержит много лекций. Для каждой лекции существует (короткий) список целей обучения, указывающий, какие знания и навыки приобретают студенты, изучая данную лекцию (отметим, что цели обучения не то же самое, что содержание лекции). Поэтому для каждой лекции желательно поместить на карту где-то близко от темы лекции небольшой прямоугольник, содержащий цели и название лекции. Такие «маленькие прямоугольники» на языке библиотеки `mindmap` называются аннотациями (annotations).

Чтобы поместить аннотацию рядом с понятием, нужно указать имя узла с этим понятием. Можно положиться на автоматическое именование узлов дерева, при котором потомки узла `root` получают имена `root-1`, `root-2`, `root-3`, и так далее. Однако, поскольку окончательный порядок понятий в дереве еще не определен, лучше явно назвать все узлы дерева, например, по именам содержащихся в них понятий.

```
\node [root concept] (Computational Complexity) {Computational Complexity}
  child [computational problems] {node (Computational Problems)
    {Computational Problems}
    child { node (Problem Measures) {Problem Measures} }
    child { node (Problem Aspects) {Problem Aspects} }
    % . . . и так далее . . .
```

Стиль `annotation` библиотеки `mindmap` определяет для аннотации прямоугольную форму соответствующего размера. Чтобы определить внешний вид аннотации, определим стиль `every annotation`.

```
\begin{tikzpicture}[shape=circle,mindmap]
\clip (-5,-4) rectangle ++ (5,6);
\begin{scope}[
    every node/.style={concept, ... }]
\node[root concept]Computational Complexity)
    {Computational Complexity} % корень
    % . . . все как и раньше . . .
\end{scope}
\begin{scope}[
    every annotation/.style={fill=black!20}]
\node [annotation, above] at
    (Computational Problems.north)
{
Lecture 1: Computational Problems
    \begin{itemize}
        \item Knowledge of several key problems
        \item Knowledge of problem encodings
        \item Being able to formalize problems
    \end{itemize}
};
\end{scope}
\end{tikzpicture}
```



Окружение `{itemize}` плохо выглядит в создаваемой карте и слишком большое. Можно внести нужные исправление и вручную, но лучше написать макрос, который позаботится о правильных аннотациях. Один из хороших вариантов — определить макрос `\lecture`, принимающий фиксированное число параметров, каждый из которых имеет следующий смысл: первый параметр — номер лекции, второй — название лекции, третий — опции позиционирования, подобные `above`, четвертый — позиция, в которую помещается узел аннотации, пятый — список отображаемых элементов аннотации, шестой — дата, когда лекция будет прочитана (этот параметр пока не нужен, но он понадобится позже).

```
\def\lecture#1#2#3#4#5#6
{
\node [annotation,#3,scale=0.65,text width=4cm,inner sep=2mm] at (#4)
{
Lecture #1: \textcolor{orange}{\textbf{#2}}
\list{--}{\topsep=2pt\itemsep=0pt\parsep=0pt
\parskip=0pt\labelwidth=8pt\leftmargin=8pt
\itemindent=0pt\labelsep=2pt} #5
\endlist
};
} % конец определения макроса
```

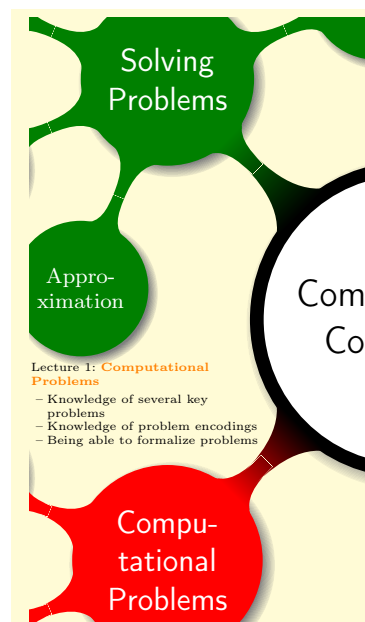
После такого определения предыдущий код можно переписать следующим образом:



```

\begin{tikzpicture}[shape=circle, mindmap]
\clip (-5,-4) rectangle ++ (4.5,8);
\begin{scope}[every node/.style={concept,...}]
\node[root concept]Computational Complexity)
      {Computational Complexity}
% . . . все как и раньше . . .
\end{scope}
\lecture{1}
  { Computational Problems }
  { above,xshift=-3mm }
  {Computational Problems.north}
  {\item Knowledge of several key problems
   \item Knowledge of problem encodings
   \item Being able to formalize problems }
  {2011-04-08}
\end{tikzpicture}

```



Точно также можно добавить аннотации и к другим лекциям. При этом, возможно, возникнут проблемы с размещением аннотаций на карте курса, которая должна помещаться на одной странице листа формата А4. Но подбирая интервалы и проводя некоторое экспериментирование, можно разместить должным образом все аннотации.

## 6.5 Добавление фона

Чтобы подчеркнуть области с темами, можно добавить фон к каждой такой области. Добавление фона дело достаточно хитрое. В данном случае подойдет «цветное колесо», которое имеет синий цвет внизу справа, и затем гладко изменяется до оранжевого цвета в верхнем правом углу карты, затем меняется на зеленый в верхнем левом углу и так далее. К сожалению, нет простого способа создать такое цветное колесо (хотя это и можно сделать в принципе, но очень дорогой ценой (см. [1, p.413]).

Поэтому создадим более простой фон: определим четыре больших прямоугольника, по одному для каждого из четырех квадрантов вокруг центрального понятия, каждый раскрасим более светлой краской, чем краска понятия, расположенного в квадранте. Затем, чтобы сгладить изменения цвета между смежными прямоугольниками, их растушуем. Так как фоновые прямоугольники должны располагаться позади всего остального, поместим весь фон на уровень `background`. Чтобы сэкономить место, отобразим только центральную часть карты:

```

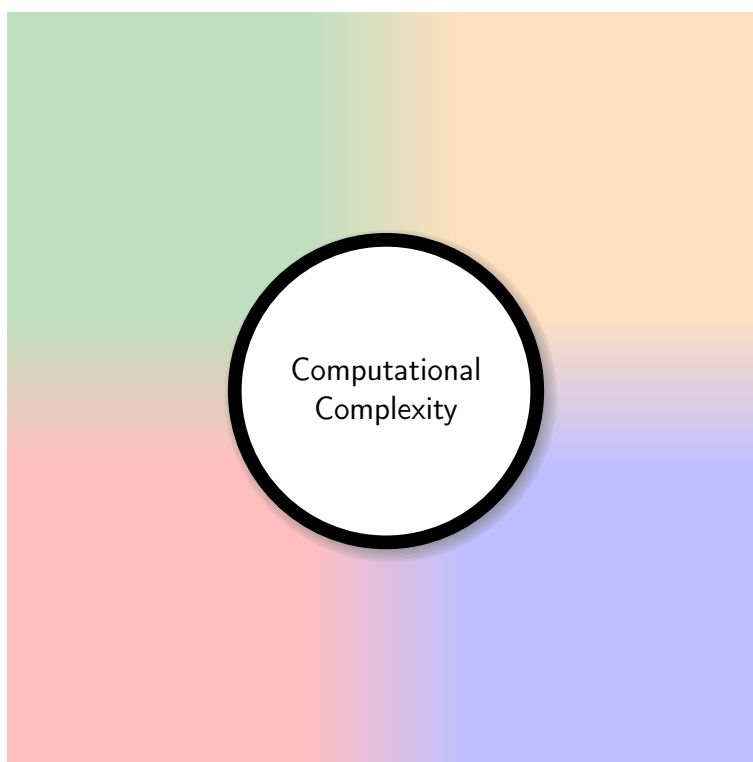
\begin{tikzpicture}[shape=circle, mindmap, concept color=black,
  root concept/.append style={concept, circular drop shadow, fill=white,
  line width=1ex, text=black, font=\sffamily}]
\clip (-5,-5) rectangle ++(10,10);
\node [root concept](Computational Complexity){Computational Complexity};
\begin{pgfonlayer}{background} \clip (-5,-5) rectangle ++(10,10);
% Большие прямоугольники
\fill [color=green!50!black!25] (Computational Complexity)
      rectangle ++(-10,10);
\fill [color=orange!25] (Computational Complexity)

```

```

rectangle ++(10,10);
\fill [color=red!25] (Computational Complexity)
rectangle ++(-10,-10);
\fill [color=blue!25] (Computational Complexity)
rectangle ++(10,-10);
% Обработка полутонов:
\shade [left color=green!50!black!25,right color=orange!25]
([xshift=-1cm]Computational Complexity)rectangle ++(2,10);
\shade [left color=red!25,right color=blue!25]
([xshift=-1cm]Computational Complexity)rectangle ++(2,-10);
\shade [top color=green!50!black!25,bottom color=red!25]
([yshift=-1cm]Computational Complexity)rectangle ++(-10,2);
\shade [top color=orange!25,bottom color=blue!25]
([yshift=-1cm]Computational Complexity)rectangle ++(10,2);
\end{pgfonlayer}
\end{tikzpicture}

```



## 6.6 Создание календаря

Поскольку есть расписание занятий, известны даты проведения лекций. Естественно, что возможны корректировки, но всегда стоит иметь детализированный план. Поэтому имеет смысл добавить календарь к карте. Календарь полезен и в том плане, что показывает хронологический порядок лекций, отсутствующий в дереве. Чтобы добавить календарь в рисунок, в TikZ используется библиотека `calendar`, определяющая команду `\calendar`. Эта команда имеет много опций и позволяет создать практически любой вид календаря. Для карты курса вполне подойдет календарь в виде простого списка дней, расположенных по нисходящей сверху вниз (опция `day list downward`).

```

\tiny
\begin{tikzpicture}
\calendar [day list downward,
name=cal,
dates=2011-04-01 to 2011-04-14]
if (weekend)
[black!25];
\end{tikzpicture}

```

```

1
2
3
4
5
6
7
8
9
10
11
12
13
14

```

Опция `name` позволяет определить имя календаря, что позволит позже сослаться на узлы, представляющие конкретные дни календаря. Например, на прямоугольный узел, содержащий 1, которая представляет 1-е апреля 2009 года, можно сослаться как на `(cal-2011-04-01)`. Опция `dates` позволяет определить интервал, для которого должен строиться календарь. Отметим конструкцию `if (weekend)`. Команда `\calendar` сопровождается опциями с последующими `if`-утверждениями. Эти условные операторы проверяются для каждого дня календаря и, когда дата удовлетворяют этому условию, опции или код, расположенные после условного оператора, выполняются. В примере выше, субботы и воскресенья выделены цветом, чтобы отличить их от рабочих дней.

Как было сказано, можно сослаться на узлы, используя дни недели. Напомним, что в макросе `\lecture` была определена дата, которая еще не использовалась. Теперь можно переопределить макросе `\lecture`, чтобы использовать ее, помещая заголовок лекции рядом с датой ее проведения:

```

\def\lecture#1#2#3#4#5#6{ % Все, как и в более раннем определении:
  \node [annotation,#3,scale=0.65,text width=4cm,inner sep=2mm] at (#4)
  { Lecture #1: \textcolor{orange}{\textbf{#2}}
    \list{--}{\topsep=2pt\itemsep=0pt\parsep=0pt
      \parskip=0pt\labelwidth=8pt\leftmargin=8pt
      \itemindent=0pt\labelsep=2pt }
    #5 \endlist
  }; % Далее, новый код:
\node [anchor=base west] at (cal-#6.base east)
{\textcolor{orange}{\textbf{#2}}};
}% конец определения макроса

```

Теперь можно использовать новую версию макроса `\lecture` следующим образом (в примере, используется только новая часть определения):

```

\small \begin{tikzpicture}
\calendar[day list downward,name=cal,
dates=2011-04-01 to 2011-04-14]
if (weekend) [black!25];
% Как и раньше:
\lecture{1}{Computational Problems}{above,xshift=-3mm}
{Computational Problems.north}{
  \item Knowledge of several key problems
  \item Knowledge of problem encodings
  \item Being able to formalize problems}{2011-04-08}
\end{tikzpicture}

```

	1
	2
Lecture 1: Computational Problems	3
- Knowledge of several key problems	4
- Knowledge of problem encodings	5
- Being able to formalize problems	6
	7
	8 <b>Computational Problems</b>
	9
	10
	11
	12
	13
	14

Наконец, нужно добавить к команде `\calendar` опцию `month text`, чтобы определить, как представить название месяца (см. [1, глава 27]), а затем набрать текст месяца в конкретной позиции в начале каждого месяца.

```
\small \begin{tikzpicture}
\calendar [day list downward, month text=\%mt\ \%y0,month yshift=3.5em,
name=cal,dates=2011-04-01 to 2011-04-14]
if (weekend) [black!25]
if (day of month=1)
{\node at (0pt,1.5em) [anchor=base west]{\small\tikzmonthtext}};
\lecture{1}{Computational Problems} {above,xshift=-3mm}
{Computational Problems.north}{
\item Knowledge of several key problems
\item Knowledge of problem encodings
\item Being able to formalize problems
}{2011-04-08}
\end{tikzpicture}
```

	April 2011
	1
	2
Lecture 1: Computational Problems	3
- Knowledge of several key problems	4
- Knowledge of problem encodings	5
- Being able to formalize problems	6
	7
	8 <b>Computational Problems</b>
	9
	10
	11
	12
	13
	14

## 6.7 Полный код карты курса

Соединим вместе все рассмотренные идеи, начиная с определение макроса `\lecture`:

```
\def\lecture#1#2#3#4#5#6
{
  \node [annotation,#3,scale=0.65,text width=4cm,inner sep=2mm,fill=white]
      at (#4)
  {
    Lecture #1: \textcolor{orange}{\textbf{#2}}
    \list{--}{\topsep=2pt\itemsep=0pt\parsep=0pt
      \parskip=0pt\labelwidth=8pt\leftmargin=8pt
      \itemindent=0pt\labelsep=2pt}
    #5 \endlist
  };
  \node [anchor=base west] at (cal-#6.base east)
      {\textcolor{orange}{\textbf{#2}} };
}
```

Определим основные установки, влияющие на внешний вид карты курса ...

```
\begin{tikzpicture}% Начало кода карты курса
\begin{scope}
[ mindmap,text=white,grow cyclic,
  every node/.style={concept,circular drop shadow,
    execute at begin node=\hskip0pt},
  root concept/.append style={concept color=black,fill=white,
    line width=1ex,text=black, font=\large\scshape},
  computational problems/.style={concept color=red,
    faded/.style={concept color=red!50}},
  computational models/.style={concept color=blue,
    faded/.style={concept color=blue!50}},
  measuring complexity/.style={concept color=orange,
    faded/.style={concept color=orange!50}},
  solving problems/.style={concept color=green!50!black,
    faded/.style={concept color=green!50!black!50}},
  level 1/.append style={level distance=4.5cm,
    sibling angle=90,font=\scshape},
  level 2/.append style={level distance=3cm,
    sibling angle=45,font=\scriptsize}
]
]
```

Определим узлы и их содержимое ...

```
\node [root concept]
  (Computational Complexity){Computational Complexity}
child [computational problems]
{
  node [yshift=-1cm]
    (Computational Problems) {Computational Problems}
```

```

    child      {node (Problem Measures) {Problem Measures}}
    child      {node (Problem Aspects) {Problem Aspects}}
    child[faded] {node (problem Domains) {Problem Domains}}
    child      {node (Key Problems) {Key Problems}}
  }
child [computational models]
{
  node [yshift=-1cm]
    (Computational Models) {Computational Models}
  child      {node (Turing Machines) {Turing Machines}}
  child[faded] {node (Random-Access Machines) {Random-Access Machines}}
  child      {node (Circuits) {Circuits}}
  child[faded] {node (Binary Decision Diagrams){Binary Decision Diagrams}}
  child      {node (Oracle Machines) {Oracle Machines}}
  child      {node (Programming in Logic) {Programming in Logic}}
}
child [measuring complexity]
{
  node [yshift=1cm]
    (Measuring Complexity) {Measuring Complexity}
  child      {node (Complexity Measures) {Complexity Measures}}
  child      {node (Classifying Complexity) {Classifying Complexity}}
  child[faded] {node (Describing Complexity) {Describing Complexity}}
  child      {node (Comparing Complexity) {Comparing Complexity}}
}
child [concept color=green!50!black]
{
  node [yshift=1cm]
    (Solving Problems) {Solving Problems}
  child      {node (Exact Algorithms) {Exact Algorithms}}
  child      {node (Randomization) {Randomization}}
  child      {node (Fixed-Parameter Algorithms)
              {Fixed-Parameter Algorithms}}
  child      {node (Parallel Computation) {Parallel Computation}}
  child      {node (Partial Solutions) {Partial Solutions}}
  child      {node (Approximation) {Approximation}}
};
\end{scope}

```

Код, определяющий календарь, ...

```

\tiny
\calendar [ day list downward,
            month text=\% mt\ \%y0,
            month yshift=3.5em,
            name=cal,
            at={(-.5\textwidth-5mm,.5\textheight-1cm)},
            dates=2011-04-01 to 2011-06-last
          ]
  if (weekend) [black!25]

```

```

if (day of month=1)
  {
    \node at (0pt,1.5em) [anchor=base west] {\small\tikzmonthtext};
  };

```

Код, определяющий аннотации лекций, ...

```

\lecture{1}
  {Computational Problems}
  {above,xshift=-5mm,yshift=5mm}
  {Computational Problems.north}
  { \item Knowledge of several key problems
    \item Knowledge of problem encodings
    \item Being able to formalize problems }
  {2011-04-08}
\lecture{2}
  {Computational Models}
  {above left}
  {Computational Models.west}
  { \item Knowledge of Turing machines
    \item Being able to compare the computational power
      of different models }
  {2011-04-15}

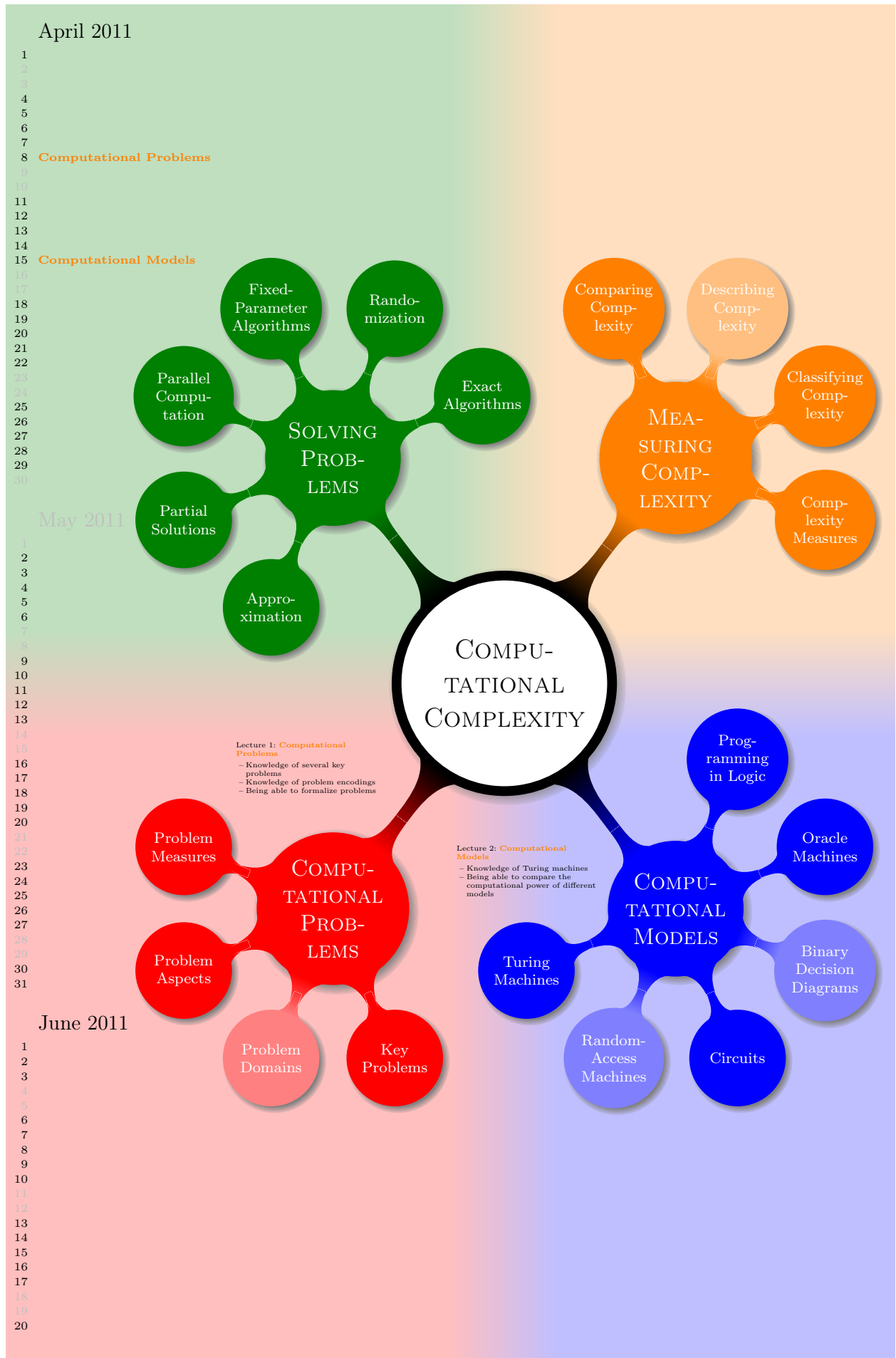
```

Код фонового уровня ...

```

\begin{pgfonlayer}{background}
\clip[xshift=-1cm]
  (-.5\textwidth,-.5\textheight) rectangle ++(\textwidth,\textheight);
% Большие прямоугольники фона ...
\fill [color=green!50!black!25]
  (Computational Complexity) rectangle ++(-20,20);
\fill [color=orange!25]
  (Computational Complexity) rectangle ++(20,20);
\fill [color=red!25]
  (Computational Complexity) rectangle ++(-20,-20);
\fill [color=blue!25]
  (Computational Complexity) rectangle ++(20,-20);
% Затенение ...
\shade [left color=green!50!black!25,right color=orange!25]
  ([xshift=-1cm]Computational Complexity) rectangle ++(2,20);
\shade [left color=red!25,right color=blue!25]
  ([xshift=-1cm]Computational Complexity) rectangle ++(2,-20);
\shade [top color=green!50!black!25,bottom color=red!25]
  ([yshift=-1cm]Computational Complexity) rectangle ++(-20,2);
\shade [top color=orange!25,bottom color=blue!25]
  ([yshift=-1cm]Computational Complexity) rectangle ++(20,2);
\end{pgfonlayer}
\end{tikzpicture}% конец кода карты курса

```





# Глава 7

## Рекомендации по графике

Оставим на время `pgf` и `TikZ`, и рассмотрим некоторые рекомендации и общие принципы создания графики для научных презентаций, статей и книг. Рекомендации взяты из разных источников. Многие из них отражают идеальную ситуацию, некоторые проистекают из личного опыта автора документации по `pgf` и `TikZ` Тилла Тантау, некоторые взяты из разных книг по графическому дизайну и книгопечатанию. Много взято из блестящих книг Эдварда Тафти (Edward Tufte) (его книга «Представление информации» переведена на русский). Хотя можно и не соглашаться со всем тем, что написано в этих книгах, но большинство аргументов Тафти весьма убедительны.

Первое, что нужно спросить у себя, когда кто-то представляет набор рекомендаций, нужно ли действительно им следовать? Часто ведь существуют серьезные основания не следовать общим рекомендациям! Человек, который давал рекомендации, возможно, имел цели, отличные от ваших. Например, есть рекомендация использовать красный цвет для выделений в тексте. Для представления, использующего проектор, это приемлемо, но для черно-белой печати такая рекомендация не годится. Рекомендации почти всегда определяются ситуацией. Если ситуация другая, следование рекомендации может принести больше вреда, чем пользы.

Второе, что обязательно следует знать — основное правило книгопечатания: каждое правило может быть нарушено, но не может быть проигнорировано. Когда правило известно, и вы решили, что его нарушение дает желаемый результат, нарушайте!

### 7.1 Планирование времени

Когда создается статья с многочисленными графиками и рисунками, время на их создание становится важным фактором. Сколько времени потребуется? Как правило, можно предположить, что графический объект потребует столько времени, сколько текст такой же длины. Например, когда создается статья, требуется приблизительно один час на страницу. Позже, еще потребуется от двух до четырех часов на проверку страниц. Таким образом, можно предположить, что потребуется полчаса на создание чернового графика в половину страницы и еще от часа до двух на его завершение.

Во многих даже хороших публикациях кажется, что авторы и редакторы тратят много часов на текст и минуты на создание графики. Возникает впечатление, что графики были добавлены в последнюю минуту, часто они похожи на снимок с экрана, созданный внешней программой, подобной `gnuplot`. Такие графики как правило достаточно низкого качества.

Создание информативных графических объектов, которые помогают читателю понять основной текст и соответствуют ему — трудный и длительный процесс.

- Следует трактовать графические объекты как полноправных граждан статей и книг. Они заслуживают на разработку, по крайней мере, такого же времени и энергии, как и текст.
- Следует планировать на создание и просмотр графического объект столько же времени, как и на текст такого же размера.
- Сложные графические объекты с высокой информационной плотностью могут требовать существенно больше времени.
- Очень простые графические объекты требуют меньше времени, но, вероятнее всего, не украсят статью или книгу.

## 7.2 Процесс создания графического объекта

Когда пишется научный труд, то, вероятно, это означает, что автор получил некоторые результаты (идеи), о которых хотел бы сообщить миру. Создание такого документа обычно начинается со сбора материала и создания его первой черновой схемы. Затем возникают различные разделы с текстом, будет создана уже не схема, а первый черновой вариант документа. После многократных просмотров и исправлений, часто после существенной переработки, появится окончательный вариант. Создание графических объектов должно происходить точно так же:

- Решить, что должен содержать графический объект, о чем он должен рассказать читателю.
- Создать грубую схему графического объекта, содержащую самые главные элементы. Часто, полезно сделать это в карандаше, используя бумагу.
- Заполнить деталями графику, создавая черновой вариант.
- Неоднократно проанализировать и отредактировать графический объект, наряду с остальной частью документа.

## 7.3 Соединение графики и текста

Графики могут помещаться в разные места текста: линейно, следуя друг за другом, развивая тему, или автономно. Так как принтеры (и люди) любят заполненные страницы (и по эстетическим и по экономическим причинам), автономный рисунок может размещаться на странице, далекой от страницы текста, к которому он относится. Но по  $\text{T}_{\text{E}}\text{X}$ ническим причинам  $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$  и  $\text{T}_{\text{E}}\text{X}$  поощряют такую политику.

Когда графика размещается линейно, она автоматически оказывается более или менее связана с текстом в том смысле, что графика поясняется близко расположенным текстом. Кроме того, из текста становится ясно, что представляет собой график и что он отображает.

Автономный график часто возникает в то время, когда основной текст, поясняющий графический объект, или еще не читался или уже был прочитан. Поэтому, создавая автономный график, нужно следовать следующим рекомендациям:

- Автономный графический объект должен иметь заголовок, его поясняющий. Например, если графический объект показывает различные стадии алгоритма быстрой сортировки (quicksort), то заголовок должен, по крайней мере, сообщить читателю, что рисунок показывает различные стадии алгоритма quicksort, введенного на странице хуз, а не только то, что рисунок показывает алгоритм quicksort.

- Хороший заголовок должен добавлять настолько много контекстной информации, насколько это возможно. Не следует бояться длинных заголовков. Хотя информацию из заголовка можно поместить в основной текст, но, помещая ее в заголовок, можно гарантировать лучшее понимание графики.
- В основном тексте следует расставлять ссылки на графические объекты.
- Большинство книг по стилю и книгопечатанию рекомендуют не использовать сокращения, такие как Рис. 2.1, а писать Рисунок 2.1. Аргумент против сокращений тот, что точка в основном тексте Рис. может ввести читателя в заблуждение, заставляя его предположить, что это конец предложения, и требуется возвратиться к началу предложения, чтобы понять, что предложение еще не закончилось. Аргумент в пользу сокращений тот, что они сохраняют место. Видимо оба аргумента спорны. С одной стороны, трудно найти того, кому сокращения в основном тексте мешали бы его понимать. С другой стороны, сокращая Рисунок 2.1 до Рис. 2.1 вряд ли удастся в большинстве документов сократить даже одну строку. Но все-таки стоит избегать сокращений.

## 7.4 Согласованность между графикой и текстом

Возможно, самые общие ошибки люди делают, создавая графические объекты, когда забывают, что должно иметь место соответствие между внешним видом графических объектов и внешним видом текста. Весьма часто используется несколько программ для создания графических объектов одного документа. Некоторые графики создаются, используя `gnuplot`, диаграммы, используя `xfig`, а включаемые рисунки в формате `*.eps` — неизвестной программой. Все эти графические объекты будут иметь разные размеры, использовать, вероятнее всего, разную толщину линий, разные шрифты. Кроме того, авторы часто используют опции вида `[height=5cm]`, когда включают рисунок в документ, чтобы масштабировать его до некоторого приемлемого размера.

Если такой подход применить к созданию основного текста, то каждый раздел писался бы разными шрифтами разного размера. В некоторых разделах все теоремы были бы подчеркнуты, в других они были бы напечатаны прописными буквами, а в третьих — красным цветом. Кроме того, размеры полей были бы свои на каждой странице. Читатели и редакторы считали бы недопустимым такой текст, но с похожими проблемами в графических объектах они часто соглашались.

Чтобы согласовать графические объекты и текст, надо придерживаться следующих рекомендаций:

- Не масштабировать графические объекты, то есть, создавая графические объекты внешней программой, сразу следует создавать их нужного размера.
- Использовать одни и те же шрифты в графических объектах и основном тексте.
- Использовать одну и ту же ширину линий в тексте и графике. Ширина линии нормального текста — ширина ножки буквы T. Для TeX'a, это обычно `0.4pt`. Однако, некоторые журналы не принимают графику с нормальной шириной линии меньше `0.5pt`.
- Когда используется цвет, нужно согласовывать цвета в тексте и в графических объектах. Например, если в основном тексте красный цвет — цвет выделения, то красными в графических объектах должны быть их важные части. Если синий цвет используется для структурных элементов (шапок и заголовков разделов) текста, то синий цвет должен также использоваться для структурных элементов графического объекта.

Однако, графические объекты могут использовать логику, присущую цвету. Например, читатели обычно предполагают, что зеленый цвет — цвет, разрешающий некоторое действие, а красный цвет — предупреждение об опасности.

Добиться согласованности, используя разные программы создания графики, почти невозможно. Поэтому следует придерживаться одной такой программы.

## 7.5 Метки в графических объектах

Почти все графические объекты содержат метки, то есть, части текста, объясняющие части графических объектов. Когда метки размещаются, нужно придерживаться следующих рекомендаций:

- Следовать правилам согласования: сначала согласовать метки с основным текстом, затем между собой (например, использовать одну и ту же форму).
- В дополнение к использованию одних и тех же шрифтов в тексте и графических объектах, следует также использовать одну и ту же нотацию. Например, если в тексте написано  $1/2$ , то метки в графических объектах должны быть тоже  $1/2$ , а не 0.5, и так далее.
- Метки должны быть четкими. Они должны иметь не только разумно большой размер, но они также не должны перекрываться другими линиями или другим текстом. Это также относится к линиям и тексту позади меток.
- Метки должны быть всегда на нужном месте. Всякий раз, когда достаточно места, метки должны помещаться рядом с тем объектом, который они маркируют. Только в случае необходимости, можно протянуть линию от метки до маркируемого ею объекта. Следует избегать меток, которые только ссылаются на объяснения во внешних источниках информации. Читатель должен «прыгать» только между описанием и объектом, который описан.
- Рассмотреть возможность «сокрытия» неосновных меток, используя метки, например, серого цвета. Это позволит читателю сконцентрироваться на самом графическом объекте.

## 7.6 Графики и диаграммы

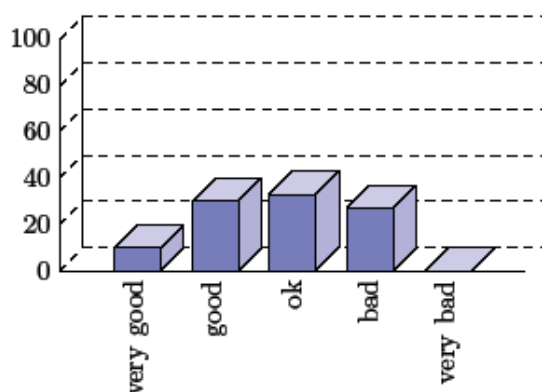
Одним из часто встречающихся видов графических объектов, особенно в научных документах, являются графики. К ним относятся простые линейные графики, параметрические графики, трехмерные графики, всевозможные диаграммы, и еще много других видов. К сожалению, графики, как общеизвестно, трудно правильно понять. Частично, в этом виноваты настройки по умолчанию таких программ, как gnuplot или Excel, позволяя очень просто создавать плохие графики.

Первый вопрос, который следует задать себе, создавая график: есть ли достаточно много данных, чтобы создать график? Если ответ — нет, следует использовать таблицу.

Типичная ситуация ненужности графика возникает тогда, когда надо представить несколько чисел в виде столбчатой диаграммы. Вот реальный пример: в конце семинара лектор просит участников заполнить форму по оценке семинара. Из 50 участников, 30 ее заполнили. Согласно формам, три участника оценили семинар как “очень хороший”, девять посчитали его “хорошим”, десять дали оценку “нормальный”, восемь — “плохой”, и никто не сказал, что семинар “очень плохой”. Простой способ подвести итог — следующая таблица:

Оценка семинара	Число участников (из 50), давших ее	Процент
очень хорошо	3	6%
хорошо	9	18%
нормально	10	20%
плохо	8	16%
очень плохо	0	0%
не ответили	20	40%

То, что сделал лектор, должно было визуализировать данные, используя трехмерную столбчатую диаграмму. Диаграмма была примерно такой:



И таблица, и график имеют почти один и тот же размер. Если вы подумали, что график смотрится лучше таблицы, попробуйте ответить на следующие вопросы, основанные на информации в таблице и на графике:

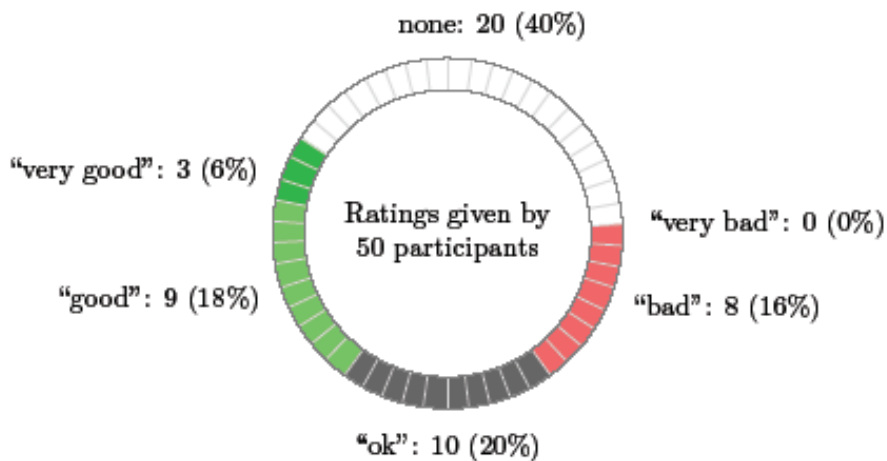
1. Сколько было участников семинара?
2. Сколько участников возвратило заполненную форму?
3. Какой процент участников возвратил заполненную форму?
4. Сколько участников сказал “очень хорошо”?
5. Какой процент участников сказал “очень плохо”?
6. Больше четверти участников сказали “плохо” или “очень плохо”?
7. Какой процент участников, возвративших форму, сказали “очень хорошо”?

К сожалению, график не позволяет ответить ни на один из этих вопросов. Таблица отвечает на все, за исключением последнего. Информационная плотность графического объекта почти равна нулю. Таблица имеет намного более высокую информационную плотность; несмотря на то, что она использует довольно много незаполненного пространства для представления нескольких чисел. Вот список того, что плохо в трехмерной столбчатой диаграмме:

- Весь график во власти раздражающих фоновых линий.
- Не ясно, что слева означают числа, видимо, проценты, но могло бы и абсолютное число участников.
- Метки в основании перевернуты, их трудно читать.
- Третье измерение добавляет сложность графику, не добавляя информативности.
- Трехмерность не дает возможности правильно оценить высоту столбцов.

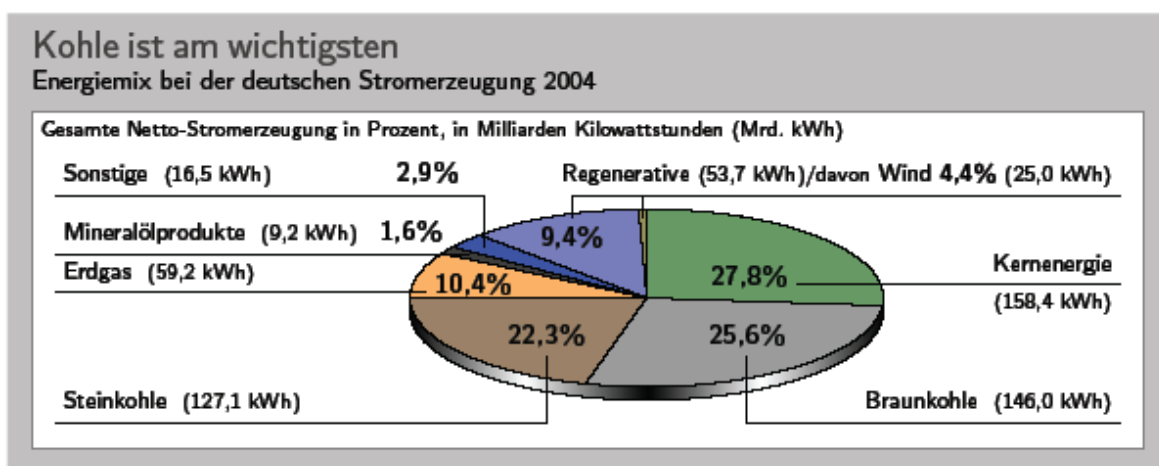
- Невозможно сказать, какие числа представлены столбцами. Таким образом, столбцы скрывают информацию, а их предназначение прямо противоположное.
- Что представляет самый высокий столбец? Это 100% или 60%?
- Столбец оценки “очень плохо” представляет 0 или 1?
- Почему столбцы синие?

Можно было бы сказать, что в примере точные числа не важны для графика. Важнее мера, по которой оценки “очень хорошо” и “хорошо” превышают оценки “очень плохо” и “плохо”. Однако чтобы передать это сообщение, можно использовать предложение, утверждающее это, или использовать другой вариант графика, который передает это сообщение более точно и ясно:



Этот график имеет ту же информационную плотность, что и таблица (при том же размере и с теми же числами). Кроме того, позволяет непосредственно увидеть, что больше хороших или очень хороших оценок, чем плохих. Можно также увидеть число людей, которые не дали оценки семинара вообще, и оно значительно.

Диаграммы — не всегда хороший способ представления информации. Рассмотрим пример круговой диаграммы, перерисованной из Die Zeit (от 4-го июня 2005):



Этот график на первый взгляд, хорошо смотрится и информативен, но есть несколько вещей, которые не так хороши, как кажется:

- Диаграмма трехмерная. Однако растушевывание не добавляет информативности, а скорее отвлекает.

- В трехмерной круговой диаграмме очень сильно искажены относительные размеры. Например, серая область больше зеленой, несмотря на то, что серая отражает меньший процент, чем зеленая.
- Трехмерное искажение ухудшает восприятие маленьких областей. Голубая область Regenerative несколько больше оранжевой области Erdgas. Область Wind (между зеленой и голубой) намного меньше, чем область Mineralolprodukte, хотя отображает процент почти в три раза больший. В последнем случае, различие в размерах происходит только частично из-за искажения. Проектировщик оригинала графика сделал область Wind слишком маленькой, даже принимая во внимание искажение.
- Согласно заголовку, диаграмма должна сообщить, что уголь был самым важным источником энергии в Германии в 2004 году. Но понять это, даже игнорируя искажения, весьма сложно, так как уголь, как источник энергии, разделен на две области: Steinkohle и Braunkohle (две разновидности угля). Если их сложить, то сумма даст меньше половины круговой диаграммы. Две эти области для угля визуально не связаны вообще: выбраны очень далекие цвета, метки расположены в разных сторонах диаграммы. По сравнению с ними, области Regenerative и Wind связаны теснее.
- Цвета на графике не имеют никакого логического объяснения. Почему зеленым выделена ядерная энергия? Почему восстанавливаемые источники энергии обозначены светло-голубым цветом, “другие источники” синим? Выглядит больше как шутка то, что область Braunkohle (буквально переводится как бурый уголь) — серая, в то время как область для Steinkohle (буквально переводится как каменный уголь) — коричневая.
- Область с самым легким цветом используется для Erdgas (энергия на газе). Эта область выделяется больше всего из-за яркого цвета. Однако на этой диаграммы зона Erdgas не является самой важной.

Вот несколько рекомендаций, которые могут помочь избежать производства плохих диаграмм:

- Не использовать трехмерные круговые диаграммы.
- По возможности использовать таблицы вместо круговых диаграмм.
- Не применять цвета беспорядочно; использовать цвет для привлечения внимания читателей и для объединения сущностей в группы.
- Не использовать фоновые шаблоны, типа штриховок или диагональных линий, вместо цветного фона. Они отвлекают. Фоновые шаблоны в информационных графиках не допустимы.

## 7.7 Внимание и отвлечение

Если взглянуть на типичную страницу хорошо изданного романа, то легко заметить, что его страница очень однородна. Ничто не должно отвлекать читателя; нет больших заголовков, нет полужирного текста, нет больших белых областей. Даже когда автор желает что-то подчеркнуть, он это делает, используя курсив. Такие буквы хорошо сочетаются с основным текстом: на расстоянии нельзя сказать, содержит ли страница курсив, но сразу же заметно единственное полужирное слово. Причина такого набора — следование следующей парадигме: избегать отвлекающих моментов.

Хорошее книгопечатание (как и хорошая организация) — нечто не очень заметное. Основная задача при печатании книги — сделать чтение текста, то есть, впитывание его информационного содержания, настолько естественным, насколько это возможно. Читатели поглощают содержание романа, читая текст, строка за строкой, как будто они слушают кого-то, рассказывающего интересную историю. В этой ситуации что-либо на странице, отвлекающее глаз от быстрого и равномерного движения по строкам, делает текст тяжелее, его станет труднее читать.

Теперь возьмем любой еженедельный журнал или газету и взглянем на их типичную страницу. Легко заметить, что на странице “много чего есть”. Используются шрифты разного размера и различного написания, текст организован в узкие столбцы, часто чередуется с рисунками. Причина набора журналов таким образом — другая парадигма: управлять вниманием.

Читатели не будут читать журнал как роман. Вместо того, чтобы читать журнал строку за строкой, они выхватят короткие резюме и заголовки, чтобы понять, надо ли читать данную статью или нет. Задача печатающего — управлять нашим вниманием такими резюме и заголовкам, это во-первых. Но как только мы решили, что мы будем читать данную статью, вы не должны отвлекаться, вот почему главный текст статей набирается точно как же, как и роман.

Эти два принципа — “не отвлекать” и “управлять вниманием” относятся и к графическим объектам. Когда проектируется графический объект, следует устранить все, что будет “резать глаз”. В то же время, следует активно помогать читателю с помощью графики, используя шрифты/цвет/ширину линий для выделения различных информационных частей. Вот только малая часть тех вещей, которые могут отвлечь читателей:

- Сильные контрасты всегда бросаются в глаза.
- Подчеркивание штриховой линией создает много контрастных черно-белых точек. Поэтому нужно избегать подчеркивания штриховой или пунктирной линией. Не следует использовать различные штриховые линии и для разделения кривых на графике. Для глаз много лучше группировать сущности по цвету, а не по штриховке.
- Фоновые шаблоны заполняют области, используя диагональные или горизонтальные и вертикальные линии или только точки, почти всегда только отвлекают внимание и, обычно, не достигают никакой реальной цели.
- Фоновые рисунки и растушевывание отвлекают и редко добавляют что-нибудь важное в графический объект.
- Симпатичная небольшая иллюстративная вставка может легко привлечь внимание, уводя далеко от данных.



Часть II  
Интерфейс TikZ

```

\begin{center}
\colorbox{yellow!20}{
\begin{tikzpicture}
  \draw[fill=yellow] (0,0) -- (60:.75cm) arc (60:180:.75cm);
  \draw(120:0.4cm) node {\alpha};

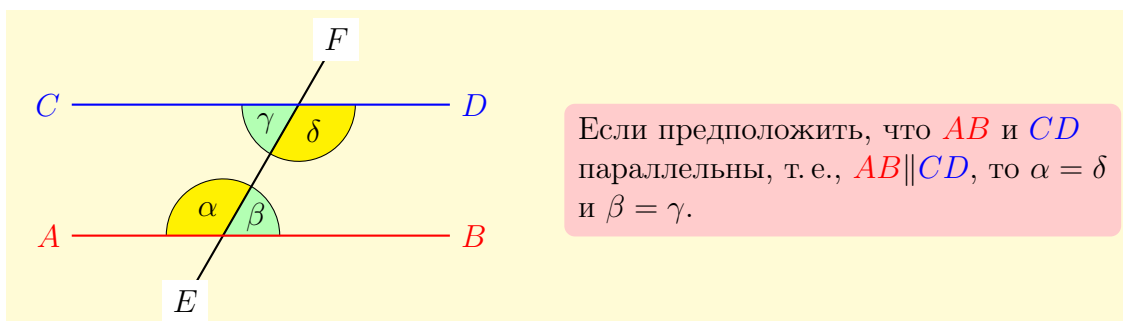
  \draw[fill=green!30] (0,0) -- (right:.75cm) arc (0:60:.75cm);
  \draw(30:0.5cm) node {\beta};

  \begin{scope}[shift={(60:2cm)}]
    \draw[fill=green!30] (0,0) -- (180:.75cm) arc (180:240:.75cm);
    \draw (30:-0.5cm) node {\gamma};
    \draw[fill=yellow] (0,0) -- (240:.75cm) arc (240:360:.75cm);
    \draw (-60:0.4cm) node {\delta};
  \end{scope}

  \begin{scope}[thick]
    \draw (60:-1cm) node[fill=white] {$E$} --
      (60:3cm) node[fill=white] {$F$};
    \draw[red] (-2,0) node[left] {$A$} -- (3,0) node[right]{$B$};
    \draw[blue,shift={(60:2cm)}] (-3,0) node[left] {$C$} --
      (2,0) node[right]{$D$};

    \draw[shift={(60:1cm)},xshift=4cm]
    node [right,text width=7cm,rounded corners,fill=red!20,inner sep=1ex]
    {
      Если предположить, что  $\color{red}AB$  и  $\color{blue}CD$ 
      параллельны, т.\,е.,  $\color{red}AB \parallel \color{blue}CD$ ,
      то  $\alpha = \delta$  и  $\beta = \gamma$ .
    };
  \end{scope}
\end{tikzpicture}}
\end{center}

```



# Глава 8

## Принципы проектирования

Рассмотрим принципы проектирования, скрытые под *внешним интерфейсом* TikZ (TikZ означает: *TikZ ist kein Zeichenprogramm* — TikZ не является программой для рисования). Работа в TikZ должна сделать процесс создания графики простым, обеспечивая легко изучаемым и удобным в работе синтаксисом для описания графических операций.

Команды и синтаксис TikZ навеяны несколькими источниками. Имена основных команд и понятие операций пути взяты из *metafont*, механизм опций — из *pstricks*, понятие стилей напоминает о *svg*. Чтобы заставить все это работать вместе, были необходимы компромиссы. Некоторые идеи были добавлены Тиллом Тантау, например, координатные преобразования.

В основе TikZ лежат следующие базовые принципы дизайна:

1. Специальный синтаксис для определения точек.
2. Специальный синтаксис для спецификации пути.
3. Действия на пути.
4. Синтаксис “ключ-значение” для графических параметров.
5. Специальный синтаксис для узлов.
6. Специальный синтаксис для деревьев.
7. Группировка графических параметров.
8. Система преобразования координат.

### 8.1 Синтаксис для определения точек

TikZ предоставляет специальный синтаксис для определения точек и координат. В самом простом случае двумерная точка задается ее координатами, отделенными запятыми, в круглых скобках, например, `(1cm,2pt)`. Точку можно задать в полярных координатах, если использовать двоеточие вместо запятой: `(30:1cm)`, что означает: *1cm* в направлении  $30^\circ$ .

Если не задаются единицы измерения, например, `(2,1)`, точка определяется в системе *xy*-координат пакета *pgf*. По умолчанию, единица *x*-вектора — это *1cm* вправо, единица *y*-вектора — *1cm* вверх.

Задание трех чисел, как `(1,1,1)`, задает точку в системе *xyz*-координат пакета *pgf*.

Также можно указать точку (позицию), используя якоря ранее определенной формы, например, `(first node.south)`.

Можно поставить два знака "плюс" перед точкой, например, `++(1cm, 0pt)`, что означает: *1cm* вправо от последней используемой точки. Это позволяет легко опреде-

лять относительные передвижения. Например, запись  $(1,0) ++ (1,0) ++ (0,1)$  определяет три точки:  $(1,0)$ ,  $(2,0)$  и  $(2,1)$ .

Наконец, вместо двух знаков "плюс", можно поставить один знак "плюс". Такая запись также задаст точку, определяемую суммой координат, но не изменит текущую точку, используемую в последующих командах. Например, запись  $(1,0) + (1,0) + (0,1)$  определяет три точки:  $(1,0)$ ,  $(2,0)$  и  $(1,1)$ .

## 8.2 Синтаксис для спецификации пути

Основная работа при создании рисунка — создание путей. Путь — ряд прямых линий или кривых, которые не обязательно должны быть связаны. Например, определим путь в виде треугольника:

```
\tikz \draw (15pt, 0pt) -- (0pt, 0pt) -- (0pt, 15pt) -- cycle;
```



## 8.3 Действия на путях

Путь — только ряд прямых линий и кривых, но надо еще определить, что должно с ним произойти. Можно нарисовать (`draw`) путь, заполнить (`fill`) путь, оттенить (`shade`) его, отсечь (`clip`) часть, или выполнить любую их комбинацию. Создание рисунка (рисование или вычерчивание) аналогично рисованию пером определенной толщины при его перемещении по бумаге или холсту. Заполнение означает, что внутренняя часть пути заливается однородным цветом. Очевидно, заполнение имеет смысл только для замкнутых путей, и, в случае необходимости, путь автоматически замыкается перед выполнением заполнения.

Задавая путь выражением `\path (0,0) rectangle (2ex, 1ex);`, можно его нарисовать, добавляя опцию `[draw]`:

```
\tikz \path [draw] (0,0) rectangle (4ex,2ex);
```



Команда `\draw` — только сокращение для `\path[draw]`. Чтобы заполнить путь, используется опция `[fill]` или команда `\fill`, которая является сокращением для `\path[fill]`. Команда `\filldraw` — сокращение для `\path[fill,draw]`. Оттенивание вызывается опцией `[shade]` (есть команды `\shade` и `\shadedraw`), а отсечение — опцией `[clip]`. Есть также команда `\clip`, которая делает то же, что и `\path[clip]`, но нет команды `\drawclip`. Вместо последней используют следующие две команды: `\draw[clip]` или `\path[draw,clip]`.

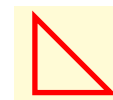
Все эти команды могут работать только внутри окружения `{tikzpicture}`. TikZ позволяет использовать разные краски для заполнения областей и рисования линий.

## 8.4 Синтаксис «ключ-значение» для параметров

Когда TikZ рисует или заполняет путь, на результат влияет большое число графических параметров: цвет, ширина линией, область отсечения, и многие другие. В TikZ, все такие опции определяются в виде списка так называемых пар «ключ-значение» (например, `color=red`), которые передаются как необязательные параметры командам

рисования пути или заполнения пути. Как пример использования опций, нарисуем толстый, красный треугольник:

```
\tikz \draw[line width=2pt,color=red]
      (1,0) -- (0,0) -- (0,1) -- cycle;
```

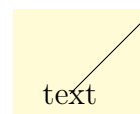


Задавая пары «ключ-значение», ключи `arrow`, `color` и `shape` можно опускать.

## 8.5 Специальный синтаксис для узлов

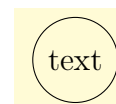
TikZ вводит специальный синтаксис, позволяющий добавить текст или, более широко, узел в графический объект. В следующем примере определяется путь, в который добавляется узел с текстом:

```
\tikz \draw (1,1) node {text} -- (2,2);
```



Узел вставляется в текущей точке (позиции) пути, но только после того, как путь сформирован. Когда задаются специальные опции, как в следующем примере,

```
\tikz \draw (1,1) node[circle,draw] {text};
```

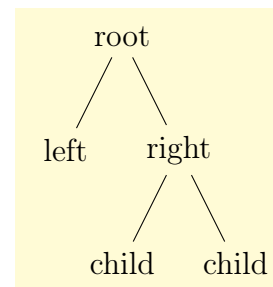


сам текст не помещается в текущей позиции, а помещается в круг, который рисуется. Для последующей ссылки на узел можно дать ему имя или опцией `name=<node name>`, или заключая имя в круглые скобки: `node[circle](name){text}`. Предопределенными формами являются `rectangle` (прямоугольник), `circle` (круг) и `ellipse` (эллипс), но можно определять и новые формы.

## 8.6 Специальный синтаксис для деревьев

В дополнение к синтаксису узлов, TikZ вводит специальный синтаксис для создания деревьев, который интегрирован с синтаксисом для узлов и только добавляет несколько новых команд. Эти команды следует запомнить. Основная из них вводит дочерний узел ключевым словом `child`. Каждый узел может сопровождаться любым числом дочерних узлов. Например,

```
\begin{tikzpicture}
\node {root}
  child {node {left}}
  child {node {right}}
    child {node {child}}
    child {node {child}}};
\end{tikzpicture}
```

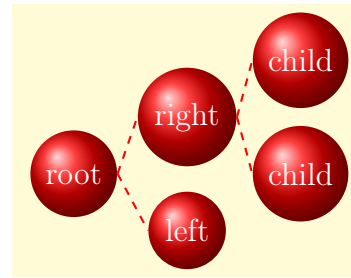


Так как деревья составлены из узлов, можно, используя опции, изменить способ, которым рисуется дерево. Приведем примеры двух деревьев, нарисованных с разными опциями:

```

\begin{tikzpicture}[parent anchor=east,
  child anchor=west,grow=east,
  every node/.style={ball color=red,
    circle,text=white},
  edge from parent/.style={draw,dashed,
    thick,red}]
\node {root}
  child {node {left}}
  child {node {right}}
    child {node {child}}
    child {node {child}}};
\end{tikzpicture}

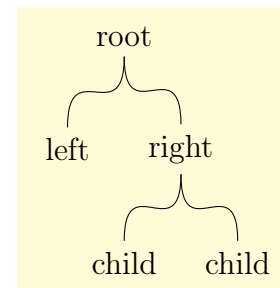
```



```

\begin{tikzpicture}[edge from parent path=
{(\tikzparentnode.south) .. controls +(0,-1)
  and +(0,1) .. (\tikzchildnode.north)}]
\node {root}
  child {node {left}}
  child {node {right}}
    child {node {child}}
    child {node {child}}};
\end{tikzpicture}

```



## 8.7 Группировка графических параметров

Часто графические параметры применяются к нескольким командам рисования или заполнения путей, например, нужно нарисовать несколько линий одной ширины, скажем, 1pt. Для этого нужно поместить такие команды в окружение `scope`, в котором определим желаемые графические опции как необязательные параметры. Они будут применяться ко всем командам рисования и заполнения внутри окружения `scope`. Вложенные окружения `scope` или конкретные команды рисования или заполнения могут отменять (переопределять) графические параметры, заданные внешними окружениями `scope`. Например,

```

\begin{tikzpicture}
\begin{scope}[red] \draw (0mm,10mm) -- (10mm,10mm);
                  \draw (0mm, 8mm) -- (10mm, 8mm);
                  \draw (0mm, 6mm) -- (10mm, 6mm);
\end{scope}
\begin{scope}[green] \draw (0mm, 4mm) -- (10mm, 4mm);
                   \draw (0mm, 2mm) -- (10mm, 2mm);
\draw[blue] (0mm, 0mm) -- (10mm, 0mm);
\end{scope}
\end{tikzpicture}

```



Окружение `{tikzpicture}` ведет себя так же, как окружение `scope`, то есть, можно определить для окружения графические параметры как необязательные и они будут применяться ко всем командам в рисунке.

## 8.8 Преобразования системы координат

TikZ поддерживает преобразования системы координат `pgf` так, чтобы реально выполнять эти преобразования, как преобразования холста на более низком уровне (детали различий между преобразованиями координат и преобразованиями холста см. в [1, 68.4]). Синтаксис определен так, что сложнее использовать преобразования холста, чем преобразования координат. Для этого есть две причины:

- преобразование холста должно использоваться с большой осторожностью, поскольку часто приводит к „плохим“ графическим объектам с меняющейся шириной линий и текстом с неправильными размерами символов;
- когда используются преобразования холста, `pgf` теряет информацию о том, где располагаются узлы и формы.

Таким образом, почти при всех обстоятельствах, нужно использовать преобразования координат, а не преобразования холста.

# Глава 9

## Иерархические структуры

В TikZ используются следующие иерархические структуры: пакеты (`packages`), окружения (`environments`), области видимости (`scopes`), стили (`styles`). Они нужны при структурировании `tex`-файла, когда используется TikZ. На верхнем уровне, нужно подключить пакет `tikz`. В основном тексте, каждый графический объект должен быть помещен в окружение `{tikzpicture}`. Внутри него можно использовать окружение `scope`, чтобы создать внутренние группы. Внутри `scope` можно использовать команды `\path`, чтобы фактически нарисовать что-либо. На всех уровнях (за исключением уровня пакета), можно задавать графические опции, которые относятся ко всему, что находится в пределах окружения. Опции, полезные в окружении или нескольких окружениях, следует оформлять в виде стилей.

Выражение `\usepackage{tikz}` в преамбуле  $\text{\LaTeX}$ 'а не допускает никаких опций и автоматически загружает пакеты `pgf` и `pgffor`. Пакет `pgf` должен знать, какой драйвер использовать, и в большинстве случаев `pgf` достаточно умен, чтобы самому определить правильный драйвер (в особенности, если используется  $\text{\LaTeX 2}_\epsilon$ ).

Выражение `\usetikzlibrary {<list libraries>}` должно располагаться после загрузки пакета `tikz`, оно загружает библиотеки из указанного списка, (библиотеки в списке должны отделяться запятыми). При попытке загрузить библиотеку во второй раз ничто не происходит. Реально такое выражение для каждой библиотеки загружает файл `tikzlibrary<library'>.code.tex`. Таким образом, чтобы написать собственную библиотеку, все, что нужно сделать, это поместить файл с соответствующим именем там, где  $\text{\TeX}$  сможет его найти.

### 9.1 Создание рисунка

Основные возможности TikZ реализует окружение `{tikzpicture}`. Рисовать можно только в нем. Особенности создания рисунка реализуются через опции, которые используются в виде «ключ-значение» (см. раздел 8.4). Все графические опции локальны по отношению к тому окружению `{tikzpicture}`, в котором они определены, и применяются ко всему, что находится внутри этого окружения. Все TikZ-команды нужно задавать только в этом окружении, за исключением команды `\tikzset`, которая позволяет определить, например, в преамбуле `tex`-файла стиль, доступный всем окружениям из этого `tex`-файла.



В конце окружения, `pgf` пробует сделать предположение в размере ограничивающего прямоугольника для графического объекта, а затем изменяет размеры бокса до таких размеров. Чтобы сделать такие предположения, каждый раз, когда `pgf` сталкивается с координатами, он обновляет размер ограничивающего прямоугольника так, чтобы тот



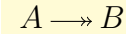
охватывал все такие координаты. Обычно получается хорошее приближение для ограничивающего прямоугольника, но не всегда точное. Во-первых, сначала не принимается во внимание толщина диагональных линий. Во-вторых, контрольные точки кривой могут лежат далеко от кривой, что делает ограничивающий прямоугольник слишком большим. В этом случае следует использовать опцию `[use as bounding box]`.

Следующий ключ влияет на базовую линию получающегося рисунка:

`/tikz/baseline=<dimension or coordinate or default>` (default 0pt)

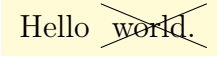
Обычно, нижняя часть рисунка помещается на базовой линии окружающего рисунок текста. Например, когда задан код `\tikz\draw(0,0)circle(.5ex);`, pgf вычислит, что нижний край рисунка равен  $-0.5ex$ , а верхний край  $0.5ex$ . Затем, нижний край поднимается до базовой линии, что приведет к следующему: . Используя эту опцию, можно поднять или опустить рисунок так, чтобы он находился на высоте `<dimension>` над базовой линией. Например, код `\tikz [baseline=0pt]\draw(0,0)circle(.5ex);` даст , так как теперь базовая линия находится на высоте  $x$ -оси. Эти опции часто нужны для встраивания графики в текст или формулу.

```
$A \mathbin{\tikz[baseline] \draw[->]} B$
(Opt, .5ex) -- (3ex, .5ex);}B$
```




Вместо указания `<dimension>` можно указать координаты в круглых скобках. Это приведет к тому, что в конце рисунка, вычисляется `<coordinate>`, а затем базовая линия помещается в  $y$ -координату вычисленной точки. Это облегчает ссылку на  $y$ -координату, скажем, базовой линии узлов.

```
Hello \tikz[baseline=(X.base)]
\node[cross out,draw] (X) {world.};
```



```
Top align:
\tikz[baseline=(current bounding box.north)]
\draw (0,0) rectangle (1cm,1ex);
```



Опция `baseline=default` устанавливает опцию `baseline` в начальное значение.

`/tikz/execute at begin picture=<code>` (no default)

Опция заставляет `<code>` выполняться в начале рисунка. Ее нужно задавать как параметр окружения `{tikzpicture}`, иначе опция не даст результата. Эффект от нескольких таких опций накапливается. Данная опция главным образом используется в стилях, чтобы выполнить некоторый код в начале рисунка.

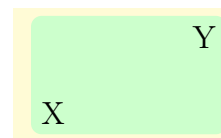
`/tikz/execute at end picture=<code>` (no default)

Опция заставляет `<code>` выполняться в конце рисунка и задается как параметр окружения `{tikzpicture}`. Эффект от нескольких таких опций накапливается.

```

\begin{tikzpicture}[execute at end picture=%
{\begin{pgfonlayer}{background}
  \path[fill=green!40!white,rounded corners]
    (current bounding box.south west) rectangle
    (current bounding box.north east);
  \end{pgfonlayer}}]
\node at (0,0) {X}; \node at (2,1) {Y};
\end{tikzpicture}

```



Все опции завершаются в конце рисунка. Чтобы установить опцию глобально, нужно изменить следующий стиль:

```
/tikz/every picture=<code> (style, initially empty)
```

Этот стиль устанавливается в начале каждого рисунка. Например, можно определить: `\tikzset{every picture/.style=semithick}`.

Однако не следует использовать `\tikzset` для явной установки опций. Например, если нужно установить всюду ширину линий в `1pt`, нельзя в начале документа сказать `\tikzset{line width=1pt}`. Это не работает, так как ширина линий меняется во многих местах. Надо сказать `\tikzset{every picture/.style={line width=1pt}}`, что даст желаемый результат.

Для небольшого рисунка, создание которого требует одной–двух команд, вместо окружения `{tikzpicture}` следует использовать команду

```
\tikz[<options>]{<path commands>}
```

Эта команда помещает `<path commands>` в окружение `{tikzpicture}` и добавляет в конце точку с запятой. Команды `<path commands>` могут содержать параграфы и хрупкий материал (подобный дословному тексту). Если используется только одна команда, фигурные скобки можно не ставить, если несколько — ставить обязательно.

```

\tikz{\draw (0,0) rectangle (2ex,1ex);}
\tikz \draw (0,0) rectangle (2ex,1ex);

```



По умолчанию, рисунки не имеют фона, то есть, они прозрачны всюду, где ничего не нарисовано. Вместо этого можно определить любой цветной фон позади рисунка, либо черную рамку вокруг него, либо линии выше или ниже, либо другое художественное оформление. Так как фон часто вообще не нужен, определение стилей для размещения фона собрано в библиотеку `backgrounds` (см. [1, chapter 25]). С использованием фона мы уже встречались в первой части и будем встречаться далее.

## 9.2 Структурирование рисунка, используя `scope`

Только в окружении `{tikzpicture}` можно, используя окружение `{scope}`, создать изолированные области видимости.

```

\begin{scope}[<options>]
  <environment contents>
\end{scope}

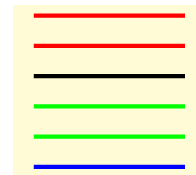
```

Все опции `<options>` локальны для `<environment contents>`. Кроме того, путь отсечения также локален для этого окружения, то есть, любое отсечение, сделанное в пределах окружения с ним и заканчивается.

```

\begin{tikzpicture}[ultra thick]
\begin{scope}[red] \draw (0mm,10mm) -- (10mm,10mm);
                  \draw (0mm,8mm) -- (10mm,8mm);
\end{scope}
\draw (0mm,6mm) -- (10mm,6mm);
\begin{scope}[green] \draw (0mm,4mm) -- (10mm,4mm);
                   \draw (0mm,2mm) -- (10mm,2mm);
\draw[blue] (0mm,0mm) -- (10mm,0mm);
\end{scope}
\end{tikzpicture}

```



Возможности окружения `scope` определяет следующий стиль, который устанавливается в начале каждого окружения:

`/tikz/every scope` (style, initially empty)

Часто бывают полезны следующие опции, которые применяется только в текущем окружении, но не во вложенных окружениях `scope`:

`/tikz/execute at begin scope=<code>` (no default)

`/tikz/execute at end scope=<code>` (no default)

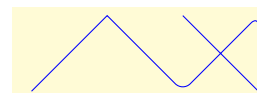
Есть небольшая библиотека `scopes`, которая позволяет проще использовать окружение `scope`. Когда она загружена, `\begin{scope}` можно заменить на `{`, а `\end{scope}` на `}`, но только в том случае, если есть опции, которые размещаются в квадратных скобках сразу за `{`, иначе `{...}` это обычный блок ЛАТЭХ'а.

Окружение `{scope}` можно использовать при построении пути. Когда команда `\path` принимает графические опции, они локальны для пути. Кроме того, в пределах пути можно создавать локальные окружения `{scope}`, используя фигурные скобки:

```

\tikz \draw[blue] (0,0) -- (1,1)
{[rounded corners] -- (2,0) -- (3,1)}
-- (3,0) -- (2,1);

```

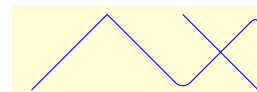


Отметим, что многие опции применяются только к пути в целом (например, цвет) и не могут быть изменены в области видимости, расположенной в этом пути.

```

\tikz \draw[blue] (0,0) -- (1,1)
{[rounded corners,red] -- (2,0) -- (3,1)}
-- (3,0) -- (2,1);

```



Наконец, некоторые элементы, которые определяются в команде `\path`, например, `node`, также могут принимать локальные опции. В этом случае, опции, определенные для элемента, применяются только к этому элементу (например, узлу), а не к пути.

## 9.3 Использование графических опций

Команды и окружения в TikZ принимают опции в виде списков «ключ-значение». Для обработки опций может использоваться команда `\tikzset{<options>}`. Но лучше

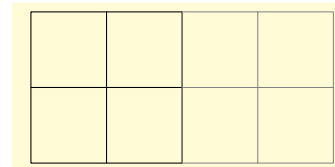
не вызывать эту команду явно, поскольку она обработает список `<options>`, используя команду `\pgfkeys` (см. [1, р. 481–503]).

При нормальных обстоятельствах, `<options>` — список пар `<key>=<value>`, отделенных запятыми, и в команде `\pgfkeys` нет необходимости (но есть причудливые ситуации, когда все же приходится использовать всю мощь механизма `pgfkeys`). Когда обрабатывается пара `<key>=<value>`, происходит следующее:

1. Если `<key>` начинается со слэша, он обрабатывается непосредственно (см. [1, р. 481–503]).
2. Иначе (что обычно имеет место), проверяется, является ли `<key>` ключом в TikZ, и, если это так, выполняется.
3. Иначе, проверяется, является ли `<key>` ключом в pgf, и, если это так, выполняется.
4. Иначе, проверяется, является ли ключ названием цвета и, если это так, выполняется код `color=<key>`.
5. Иначе, проверяется, содержит ли ключ черточку и, если это так, выполняется код `arrows=<key>`.
6. Иначе, проверяется, является ли ключ названием формы и, если это так, выполняется код `shape=<key>`.
7. Иначе напечатается сообщение об ошибке.

Есть способ организовать наборы графических опций через механизм нормальной области видимости. Например, чтобы нарисовать все линии серыми и тонкими, можно определить стиль. Стиль — ключ, который, когда используется, заставляет обрабатывать последовательность графических опций. Как только стиль определен, его можно использовать, как любой другой ключ. Например, предопределен стиль `help lines`, который следует использовать для линий в виде сетки на заднем плане (фоне).

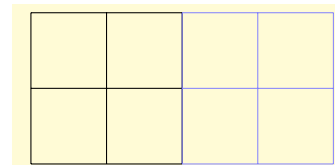
```
\begin{tikzpicture}
\draw (0,0) grid +(2,2);
\draw[help lines] (2,0) grid +(2,2);
\end{tikzpicture}
```



Определение стиля происходит, используя опции. Пусть нужно определить стиль `my style`, который, рисует линии красным цветом (`red`), а заполняет области бледно-красным цветом (`red!20`). Чтобы добиться этого, используем следующее определение: `my style/.style={draw=red,fill=red!20}`. Далее, используя ключ `my style`, получим тот же результат, что и при использовании опций `draw=red,fill=red!20`.

Возвращаясь к стилю `help lines`, предположим, что нужны синие линии. Этого можно добиться следующим образом:

```
\begin{tikzpicture}
[help lines/.style={color=blue!50,very thin}]
\draw (0,0) grid +(2,2);
\draw[help lines] (2,0) grid +(2,2);
\end{tikzpicture}
```

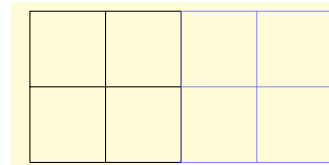


Одна из главных идей определения стилей в том, что их можно использовать в разных рисунках. Для этого, где-то в начале нужно использовать команду `\tikzset`.

```
\tikzset={help lines/.style={color=blue!50,very thin}}
```

Так как стили — только специальные случаи общего механизма стилей `pgfkeys`, реально можно сделать больше: добавить некоторые опции к уже существующему стилю. Для этого следует использовать `/.append style` вместо `/.style`:

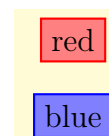
```
\begin{tikzpicture}
  [help lines/.append style={blue!50}]
  \draw (0,0) grid +(2,2);
  \draw [help lines] (2,0) grid +(2,2);
\end{tikzpicture}
```



В примере выше, опция `blue!50` добавляется к стилю `help line`, который теперь дает то же результат, что и код `black!50,very thin,blue!50`. Когда установлены два цвета, выполняется последний. Также существует обработчик стилей `/.prefix style`, который добавляет опции в начало стиля.

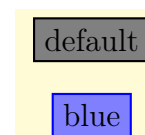
Так же, как и нормальные ключи, стили можно параметризовать. Покажем это на примерах.

```
\begin{tikzpicture}
  [outline/.style={draw=#1,thick,fill=#1!50}]
  \node [outline=red] at (0,1) {red};
  \node [outline=blue] at (0,0) {blue};
\end{tikzpicture}
```



Для параметрических стилей можно установить значение по умолчанию, используя обработчик `/.default` (детали в см. [1, p. 481–503]):

```
\begin{tikzpicture}
  [outline/.style={draw=#1,thick,fill=#1!50},
   outline/.default=black]
  \node [outline] at (0,1) {default};
  \node [outline=blue] at (0,0) {blue};
\end{tikzpicture}
```



# Глава 10

## Определение координат

Координата — позиция на холсте, на котором создается рисунок. TikZ использует специальный синтаксис для определения координат. Координаты всегда задаются в круглых скобках. Общий синтаксис определения координат таков:

([<options>]<coordinate specification>)

Параметр <coordinate specification> определяет координаты, используя одну из многих возможных систем координат. Например, декартову систему координат в  $\mathbb{R}^2$  и  $\mathbb{R}^3$ , либо полярную систему координат в  $\mathbb{R}^2$ , либо сферическую систему координат в  $\mathbb{R}^3$ . Не важно, какая система координат используется, важно, что точка на холсте представляется координатами. Есть два способа определить, какая система координат должна использоваться.

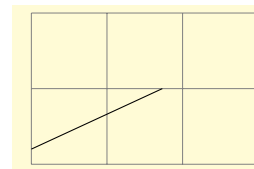
**Можно определить систему координат явно**, для чего надо задать вначале название системе координат, которое сопровождается ключевым словом **cs**: (coordinate system), за которым следует список пар «ключ-значение», характерных для данной системы координат:

(<coordinate system> cs: <list of key-value pairs specific to the coordinate system>)

Когда нужно задать набор точек, явное задание оказывается слишком многословным. Поэтому для системы координат часто используется специальный синтаксис, обеспечивающий *неявное задание*. TikZ заметит использование координат в специальном синтаксисе и автоматически выберет правильную систему координат.

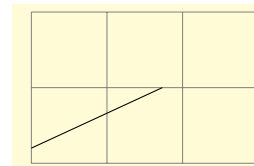
Вот пример, в котором система координат задается явно:

```
\begin{tikzpicture}
  \draw[help lines] (0,0) grid (3,2);
  \draw (canvas cs:x=0cm,y=2mm)
    -- (canvas polar cs:radius=2cm,angle=30);
\end{tikzpicture}
```



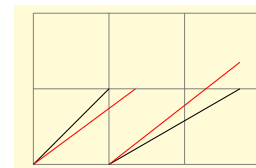
Теперь тот же пример, в котором система координат задается неявно:

```
\begin{tikzpicture}
  \draw[help lines] (0,0) grid (3,2);
  \draw (0cm,2mm) -- (30:2cm);
\end{tikzpicture}
```



Можно задавать опции, применяемые к единственной координате. Это имеет смысл только для опций преобразования и, чтобы задать опции преобразования для единственной координаты, их надо указать в начале в квадратных скобках:

```
\begin{tikzpicture}
\draw[help lines] (0,0) grid (3,2);
\draw (0,0) -- (1,1);
\draw[red] (0,0) -- ([xshift=10pt] 1,1);
\draw (1,0) -- +(30:2cm);
\draw[red] (1,0) -- +([shift=(90:10pt)] 30:2cm);
\end{tikzpicture}
```

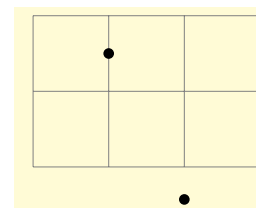


## 10.1 Системы координат

### Система координат **canvas**

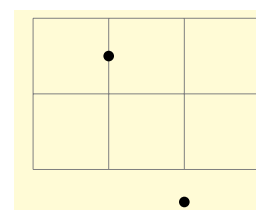
Самый простой способ определить точку — использовать систему координат **canvas** (декартову систему координат в  $\mathbb{R}^2$ ), в которой точка задается относительно начала координат опциями `x=<dimension-x>` и `y=<dimension-y>`. При этом можно определять опции в виде арифметических выражений, например, например `1cm+2pt`.

```
\begin{tikzpicture}
\draw[help lines] (0,0) grid (3,2);
\fill (canvas cs:x=1cm,y=1.5cm) circle (2pt);
\fill (canvas cs:x=2cm,y=-5mm+2pt) circle (2pt);
\end{tikzpicture}
```



Чтобы в системе координат **canvas** определить точку неявно, используются две размерности, отделяемые запятой: `(0cm,3pt)` или `(2cm, \texttheight)`.

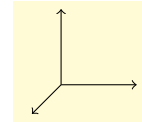
```
\begin{tikzpicture}
\draw[help lines] (0,0) grid (3,2);
\fill (1cm,1.5cm) circle (2pt);
\fill (2cm,-5mm+2pt) circle (2pt);
\end{tikzpicture}
```



### Система координат **xуз**

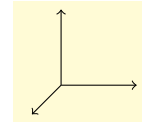
Система координат **xуз** позволяет определить точку как линейную комбинацию трех векторов, названных *x*-вектор, *y*-вектор и *z*-вектор. По умолчанию, *x*-вектор направлен вправо и имеет длину `1cm`, *y*-вектор направлен вверх и имеет длину `1cm` (при необходимости все это можно изменить, см. раздел 19.2). По умолчанию *z*-вектор указывает в точку `(-3.85mm,-3.85mm)`. Чтобы определить множители, на которые должны умножаться эти векторы, следует использовать следующие три опции `x=`, `y=`, `z=`. Например, это можно сделать явно:

```
\begin{tikzpicture}[->]
  \draw (0,0) -- (xyz cs:x=1);
  \draw (0,0) -- (xyz cs:y=1);
  \draw (0,0) -- (xyz cs:z=1);
\end{tikzpicture}
```



Эту же систему координат можно указать неявно, указывая для каждой точки два или три множителя (не расстояния) разделяемых запятыми.

```
\begin{tikzpicture}[->]
  \draw (0,0) -- (1,0);
  \draw (0,0) -- (0,1,0);
  \draw (0,0) -- (0,0,1);
\end{tikzpicture}
```



Отметим, что можно использовать координаты вида  $(1, 2\text{cm})$ , которые не являются ни координатами `canvas`, ни координатами `xyz`. Правило следующее: если координаты имеют неявную форму  $(\langle x \rangle, \langle y \rangle)$ , то  $\langle x \rangle$  и  $\langle y \rangle$  проверяются, независимо от того, имеют ли они единицы измерения или являются безразмерными. Если оба числа имеют измерение, используется система координат `canvas`. Если обе единицы измерения отсутствуют, используется система координат `xyz`. Если  $\langle x \rangle$  имеет измерение, а  $\langle y \rangle$  не имеет, то используется сумма двух координат  $(\langle x \rangle, 0\text{pt})$  и  $(0, \langle y \rangle)$ . Если  $\langle y \rangle$  имеет измерение, а  $\langle x \rangle$  нет, то используется сумма двух координат  $(\langle x \rangle, 0)$  и  $(0\text{pt}, \langle y \rangle)$ .

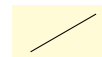
Отметим, что выражение вида  $(2+3\text{cm}, 0)$  не то же самое, что  $(2\text{cm}+3\text{cm}, 0)$ . Вместо этого, если  $\langle x \rangle$  или  $\langle y \rangle$  использует смесь значений с единицами измерения и безразмерных значений, тогда все безразмерные значения интерпретируются как значения с единицей измерения `pt`. Таким образом,  $2+3\text{cm}$  — это  $2\text{pt}+3\text{cm}$ .

## Система координат `canvas polar`

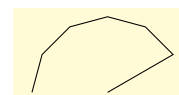
Система координат `canvas polar` позволяет определять точки в полярной системе координат. Можно задать полярный угол и радиус, используя опции `angle=` и `radius=`, которые определяют точку на холсте на заданном радиусом расстоянии от начала координат и с углом поворота на заданный угол ( $90^\circ$  — направление вверх). Угол нужно задавать в промежутке от  $-360$  до  $720$ . Расстояние — всегда положительное число.

Если вместо опции `radius=` задать опции `x radius=` и `y radius=`, то точка указывается не на окружности, а на эллипсе. Неявная форма задания полярной системы координат: угол и расстояние, отделенные двоеточием. Например,  $(30:1\text{cm})$ . Два разных радиуса (для одного угла) определяются специальной записью:  $(30:1\text{cm and } 2\text{cm})$ .

```
\tikz \draw (0,0) -- (canvas polar cs:angle=30,radius=1cm);
```



```
\tikz \draw (0cm, 0cm) -- (30:1cm) -- (60:1cm)
  -- (90:1cm) -- (120:1cm)
  -- (150:1cm) -- (180:1cm);
```





Для неявной формы, вместо угла, заданного в виде числа, можно использовать определенные слова. Например, `up` вместо `90`, так что можно написать

```
\tikz \draw (0,0) -- (2ex,0pt) -- +(up:1ex);
```



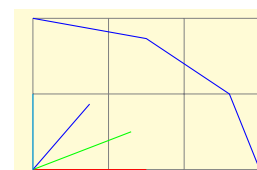
Кроме `up` можно использовать слова `down`, `left`, `right`, `north`, `south`, `west`, `east`, `north east`, `north west`, `south east`, `south west`, каждое из которых соответствует их естественному значению.

## Система координат **xyz polar**

Эта система координат подобна системе `canvas polar`. Однако, радиус и угол интерпретируются в системе координат `xy`, а не в системе `canvas`. Более точно, рассмотрим круг или эллипс, полуоси которого задаются текущими  $x$ -вектором и  $y$ -вектором. Затем, рассмотрим точку, которая находится под заданным углом на этом эллипсе, где угол  $0^\circ$  — это то же, что и  $x$ -вектор, а угол  $90^\circ$  —  $y$ -вектор. Наконец, умножим получившийся вектор на множитель, равный радиусу.

Итак, опция `angle=<degrees>` интерпретируется на эллипсе, полуоси которого —  $x$ -вектор и  $y$ -вектор, опция `radius=<factor>` — множитель, на который умножаются  $x$ - и  $y$ -векторы до формирования эллипса. Опция `x radius=<dimension>` — специальный множитель, на который умножается только  $x$ -вектор, опция `y radius=<dimension>` — специальный множитель, на который умножается только  $y$ -вектор.

```
\begin{tikzpicture}[x=1.5cm,y=1cm]
\draw[help lines] (0cm,0cm) grid (3cm,2cm);
\draw[red] (0,0) -- (xyz polar cs:angle=0,radius=1);
\draw[green] (0,0) -- (xyz polar cs:angle=30,radius=1);
\draw[blue] (0,0) -- (xyz polar cs:angle=60,radius=1);
\draw[cian] (0,0) -- (xyz polar cs:angle=90,radius=1);
\draw[blue] (xyz polar cs:angle=0,radius=2)
-- (xyz polar cs:angle=30,radius=2)
-- (xyz polar cs:angle=60,radius=2)
-- (xyz polar cs:angle=90,radius=2);
\end{tikzpicture}
```



Неявная версия этой опции та же, что и неявная версия `canvas polar`, только без указания единиц измерения.

```
\tikz[x={(0cm,1cm)},y={(-1cm,0cm)}]
\draw (0,0) -- (30:1) -- (60:1) -- (90:1)
-- (120:1) -- (150:1) -- (180:1);
```



## Система координат **xy polar**

Это другое название для системы `xyz polar`, которому можно отдать предпочтение, поскольку в системе координат `xyz polar` реально нет никакой  $z$ -координаты.

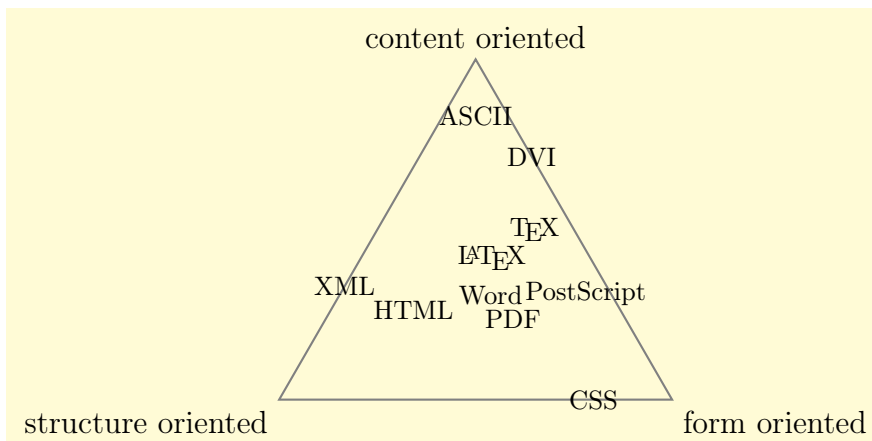
## Барицентрическая система координат

В барицентрической системе координат (`barycentric cs`) точка выражается в виде линейной комбинации множества векторов: следует определить векторы  $v_1, v_2, \dots, v_n$  и числа  $\alpha_1, \alpha_2, \dots, \alpha_n$ . Тогда барицентрические координаты, определяемые этими векторами и числами вычисляются как

$$\frac{\alpha_1 v_1 + \alpha_2 v_2 + \dots + \alpha_n v_n}{\alpha_1 + \alpha_2 + \dots + \alpha_n}.$$

Для этой системы координат список `<coordinate specification>` должен быть списком пар вида `<node name>=<number>`, разделенных запятыми. Отметим, что список не должен содержать пробелов до или после `<node name>` (в отличие от нормальных пар «ключ-значение»). Специальная координата вычисляется следующим образом: каждая пара обеспечивает вектор и число. Вектор — якорь `center` узла `<node name>`. Число — это `<number>`. Отметим, что пока нельзя определить другой якорь, и, чтобы использовать, скажем, якорь `north` узла, нужно создать новую координату на этом якоре (используя, например, `\coordinate(mynorth) at (mynode.north);`).

```
\begin{tikzpicture}
  \coordinate (content) at (90:3cm);
  \coordinate (structure) at (210:3cm);
  \coordinate (form) at (-30:3cm);
  \node [above] at (content) {content oriented};
  \node [below left] at (structure) {structure oriented};
  \node [below right] at (form) {form oriented};
  \draw [thick,gray] (content.south) -- (structure.north east)
    -- (form.north west) -- cycle;
\small
\node at (barycentric cs:content=0.5,structure=0.1 ,form=1) {PostScript};
\node at (barycentric cs:content=1 ,structure=0 ,form=0.4) {DVI};
\node at (barycentric cs:content=0.5,structure=0.5 ,form=1) {PDF};
\node at (barycentric cs:content=0 ,structure=0.25,form=1) {CSS};
\node at (barycentric cs:content=0.5,structure=1 ,form=0) {XML};
\node at (barycentric cs:content=0.5,structure=1 ,form=0.4) {HTML};
\node at (barycentric cs:content=1 ,structure=0.2 ,form=0.8) {\TeX};
\node at (barycentric cs:content=1 ,structure=0.6 ,form=0.8) {\LaTeX};
\node at (barycentric cs:content=0.8,structure=0.8 ,form=1) {Word};
\node at (barycentric cs:content=1 ,structure=0.05,form=0.05){ASCII};
\end{tikzpicture}
```



## Система координат **node**

В pgf и TikZ весьма просто определить узел, на который можно сослаться в последующем. Когда уже есть определенный узел, существуют разные способы сослаться на точки узла. Система координат **node** позволяет указать и сослаться на конкретную точку внутри или на границе узла. Для этого можно использовать три опции:

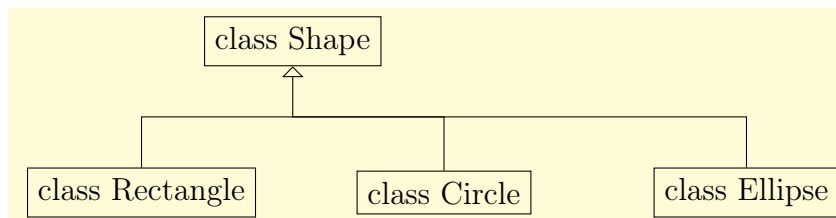
**/tikz/cs/name=<node name>** (no default)

Определяет узел, в котором определяется координата, <node name> — имя, которое ранее использовалось для именованя узла, используя специальный синтаксис или опцию `name=<node name>`.

**/tikz/anchor=<anchor>** (no default)

Определяет якорь узла. Вот поясняющий пример:

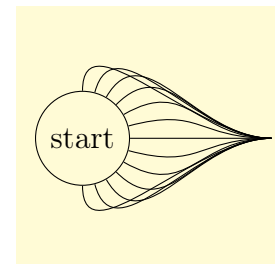
```
\begin{tikzpicture}
  \node (shape) at (0,2) [draw] {\class Shape|};
  \node (rect) at (-2,0) [draw] {\class Rectangle|};
  \node (circle) at (2,0) [draw] {\class Circle|};
  \node (ellipse) at (6,0) [draw] {\class Ellipse|};
  \draw (node cs:name=circle,anchor=north) |- (0,1);
  \draw (node cs:name=ellipse,anchor=north) |- (0,1);
  \draw[-open triangle 90] (node cs:name=rect,anchor=north)
    |- (0,1) -| (node cs:name=shape,anchor=south);
\end{tikzpicture}
```



**/tikz/cs/angle=<degrees>** (no default)

Обеспечивает угол вместо якоря. Эта координата ссылается на точку границы узла, в которой часть луча, проведенного из центра, пересекает границу под указанным углом.

```
\begin{tikzpicture}
  \node (start) [draw,shape=circle] {start};
  \foreach \angle in {-90, -75, ..., 90}
  \draw (node cs:name=start,angle=\angle)
    .. controls +(\angle:1cm) and +(-1,0) .. (2.5,0);
\end{tikzpicture}
```



Можно не указывать ни опцию `anchor=`, ни опцию `angle=`. В этом случае, TikZ сам вычислит соответствующую точку границы. Например:

```

\begin{tikzpicture}
\path (0,0) node(a)
      [ellipse,rotate=10,draw] {Эллипс}
      (3,-1) node(b) [circle,draw] {Круг};
\draw [thick]
      (node cs:name=a) -- (node cs:name=b);
\end{tikzpicture}

```



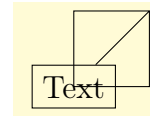
TikZ «достаточно умен» и часто в состоянии определить граничные точки, но он может иногда и ошибаться. Если TikZ не в состоянии определить соответствующую граничную точку, он вместо нее будет использовать центр. Автоматическое вычисление якорей работает только с операциями `-- (line-to)`, `|-`, `-|`, `.. (curve-to)`. Для других команд пути, типа `parabola` или `plot`, будет использоваться центр. Если это не желательно, следует явно задать именованный якорь или угловой якорь. Заметим, если используется автоматическая координата и для начала и для конца линии, как в `-- (node cs:name=b) --`, тогда две координаты границы вычисляются с перемещением между ними. Обычно это то, что нужно.

Если используются относительные координаты вместе с автоматическими координатами якоря, относительные координаты вычисляются относительно центра узла, а не относительно точки границы:

```

\tikz \draw (0,0) node(x) [draw] {Text}
          rectangle (1,1)
          (node cs:name=x) -- +(1,1);

```

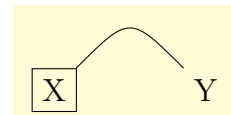


В следующем примере обе контрольные точки — точки (1,1):

```

\tikz \draw (0,0) node(x) [draw] {X}
          (2,0) node(y) {Y}
          (node cs:name=x) .. controls +(1,1) and +(-1,1) ..
          (node cs:name=y);

```

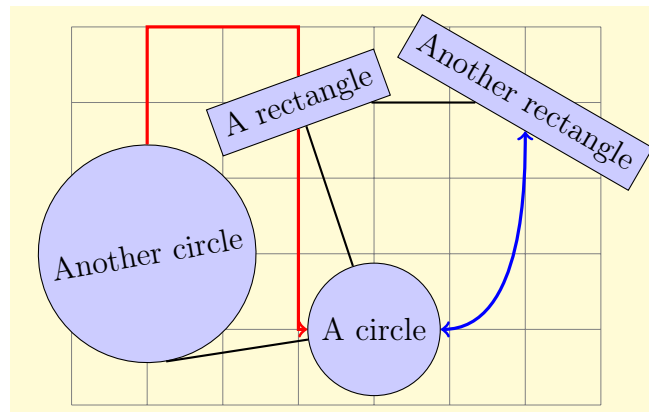


Неявный способ определения системы координат узла состоит в том, чтобы просто использовать имя узла в круглых скобках или определить имя узла вместе с якорем или углом, отделенными точкой, как в `(a.north)` или `(a.10)`. Приведем более сложный пример:

```

\begin{tikzpicture}[fill=blue!20]
\draw[help lines] (-1,-2) grid (6,3);
\path (0,0) node(a) [shape=circle,rotate=10,draw,fill]{Another circle}
      (3,-1) node(b) [shape=circle,draw,fill] {A circle}
      (2,2) node(c) [rectangle,rotate=20,draw,fill] {A rectangle}
      (5,2) node(d) [rectangle,rotate=-30,draw,fill] {Another rectangle};
\draw[thick] (a.south) -- (b) -- (c) -- (d);
\draw[very thick,color=red,->] (a) |- +(1,3) -| (c) |- (b);
\draw[very thick,color=blue,<->]
      (b) .. controls +(right:2cm) and +(down:1cm) .. (d);
\end{tikzpicture}

```



## Система координат **tangent**

Система координат **tangent** доступна только тогда, когда загружена `tikz`-библиотека `calc`. Она позволяет вычислить точку, лежащую на касательной к форме. Точнее, рассмотрим узел `<node>` и точку `<point>`. Нарисуем прямую от `<point>` так, чтобы прямая касалась узла `<node>` (являлась касательной к узлу). Точка, в которой прямая касается формы, называется точкой системы координат **tangent**. Можно задавать следующие опции:

`/tikz/cs/node=<node>` (no default)

Определяет узел, на границе которого должна лежать точка касания.

`/tikz/cs/point=<point>` (no default)

Определяет точку, через которую должна пройти касательная.

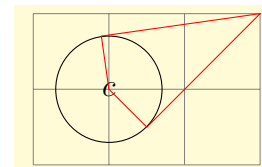
`/tikz/cs/solution=<number>` (no default)

Определяет, какая точка касания должна использоваться, если их несколько.

Для этой системы координат нет неявного синтаксиса.

Необходим специальный алгоритм, позволяющий вычислить касательную для данной формы. В настоящее время касательная может быть вычислена для узлов, форма которых координата (`coordinate`) или круг (`circle`).

```
\begin{tikzpicture}
  \draw[help lines] (0,0) grid (3,2);
  \coordinate (a) at (3,2);
  \node [shape=circle,draw] (c) at (1,1)
    [minimum size=40pt] {$c$};
  \draw[red]
    (a) -- (tangent cs:node=c,point={a},solution=1) --
    (c.center) -- (tangent cs:node=c,point={a},solution=2)
    -- cycle;
\end{tikzpicture}
```



Заметим, что можно определять новую именованную систему координат (см. [1, p. 130]).

## 10.2 Координаты точек пересечения

Часто требуется вычислить точку пересечения двух путей. Для специального и частого случая двух перпендикулярных прямых, доступна специальная система координат `perpendicular`. Для общих случаев нужна библиотека `intersection`.

### Пересечения перпендикулярных прямых

Распространенный частный случай пересечения путей — пересечение вертикальной прямой линии, проходящей через точку  $p$ , и горизонтальной прямой линии, проходящая другую точку  $q$ . Для этой ситуации существует полезная система координат `perpendicular`. В ней можно определить две прямые, используя следующие ключи:

`/tikz/cs/horizontal line through={(<coordinate>)}` (no default)

Определяет горизонтальную прямую линию, проходящую через заданную точку.

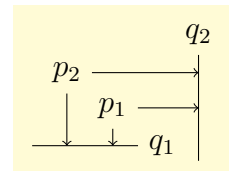
`/tikz/cs/vertical line through={(<coordinate>)}` (no default)

Определяет вертикальную прямую линию, проходящую через заданную точку.

Однако, почти во всех случаях следует использовать неявный синтаксис, записывая `(<p> |- <q>)` или `(<q> -| <p>)`. Например, выражения `(2,1 |- 3,4)` и `(3,4 -| 2,1)` дадут точку  $(2,4)$  (в  $xy$ -системе координат).

Самое полезное применение такого синтаксиса — провести линию до некоторой точки на вертикальной или горизонтальной прямой линия. Например:

```
\begin{tikzpicture}
\path (30:1cm) node(p1) {$p_1$} (75:1cm) node(p2) {$p_2$};
\draw (-0.2,0) -- (1.2,0) node(xline)[right] {$q_1$};
\draw (2,-0.2) -- (2,1.2) node(yline)[above] {$q_2$};
\draw[->] (p1) -- (p1 |- xline);
\draw[->] (p2) -- (p2 |- xline);
\draw[->] (p1) -- (p1 -| yline);
\draw[->] (p2) -- (p2 -| yline);
\end{tikzpicture}
```



### Пересечение произвольных путей

При поиске точек пересечения двух произвольных путей, пути не должны быть слишком сложными, поскольку точность вычислений в  $\text{T}_\text{E}_\text{X}$ 'е не велика. В частности, не нужно пытаться вычислять точки пересечения путей, состоящих их большого числа небольших сегментов, типа графиков или декорированных путей.

Чтобы найти точки пересечения двух путей в `TikZ`, они должны иметь имена. Имя пути задается, используя следующие ключи:

`/tikz/name path=<name>` (no default)

`/tikz/name path global=<name>` (no default)

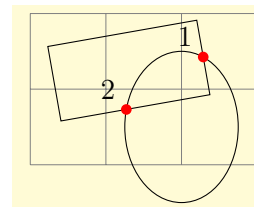
После того, как путь создан, но перед тем, как его использовать, он связывается с именем `<name>`. Для `name path`, эта связь возникает до финальной точки с запятой пути, но не в конце области видимости, содержащей путь. Для `name path global` связь возникает вне любой области видимости. С этими ключами надо обращаться осторожно.

Чтобы найти точки пересечения именованных путей, используется ключ

`/tikz/name intersections={<options>}` (no default)

Опции определяют, какие пути использовать для пересечения. Точка создается для каждого пересечения, и, по умолчанию, получает имя `intersection-1`, `intersection-2`, и так далее. Префикс `intersection` можно изменить, а общее количество пересечений хранится в `TeX`-макросе.

```
\begin{tikzpicture}
[every node/.style={black, above left}]
\draw [help lines] grid (3,2);
\draw [name path=ellipse]
(2,0.5) ellipse (0.75cm and 1cm);
\draw [name path=rectangle, rotate=10]
(0.5,0.5) rectangle +(2,1);
\fill [red,name intersections={of=ellipse and rectangle}]
(intersection-1) circle (2pt) node {\small 1}
(intersection-2) circle (2pt) node {\small 2};
\end{tikzpicture}
```



В `<options>` можно использовать следующие опции:

`/tikz/intersection/of=<name path 1> and <name path 2>` (no default)

Определяет имена путей, используемых для пересечения.

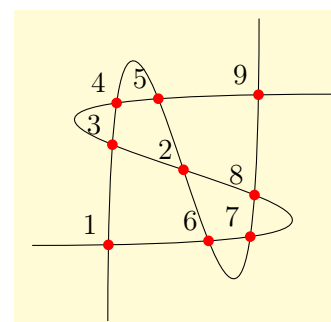
`/tikz/intersection/name=<prefix>` (no default, initially `intersection`)

Определяет имя (префикс) для точек пересечения.

`/tikz/intersection/total=<macro>` (no default)

Указывает, что число найденных точек пересечения будет хранить макрос `macro`.

```
\begin{tikzpicture}
\clip (-2,-2) rectangle (2,2);
\draw [name path=curve 1]
(-2,-1) .. controls (8,-1) and (-8,1) .. (2,1);
\draw [name path=curve 2]
(-1,-2) .. controls (-1,8) and (1,-8) .. (1,2);
\fill [red,name intersections=
{of=curve 1 and curve 2, name=i, total=\t}]
\foreach \s in {1,...,\t}{(i-\s) circle (2pt)
node [above left, black] {\small \s}};
\end{tikzpicture}
```



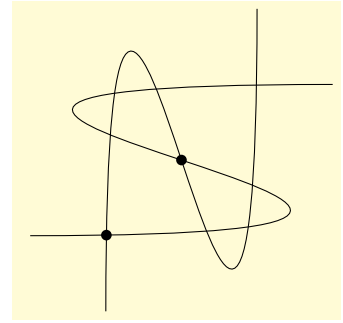
`/tikz/intersection/by=<comma-separated list>` (no default)

Позволяет определить список имен для точек пересечения. Точки пересечения все еще именуется в нотации `<prefix>-<number>`, но, дополнительно, первая точка также именуется как первый элемент списка `<comma-separated list>`, и так далее. Затем список `<comma-separated list>` передается утверждению `\foreach`, и для элементов списка точки пересечения создаются с новыми именами.

```

\begin{tikzpicture}
\clip (-2,-2) rectangle (2,2);
\draw [name path=curve 1]
(-2,-1) .. controls (8,-1) and (-8,1) .. (2,1);
\draw [name path=curve 2]
(-1,-2) .. controls (-1,8) and (1,-8) .. (1,2);
\fill [name intersections=
      {of=curve 1 and curve 2, by={a,b}}]
(a) circle (2pt) (b) circle (2pt);
\end{tikzpicture}

```

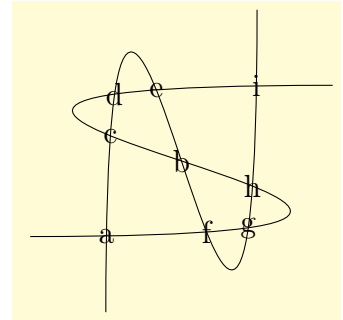


Внутри списка `<comma-separated list>` можно использовать обозначение `..`, присущее `\foreach`. Если элемент `<comma-separated list>` начинается с опции в квадратных скобках, она используется при создании точек. Имя точки может, но не должно, следовать за опциями. Это позволяет легко добавлять метки к точкам пересечения:

```

\begin{tikzpicture}
\clip (-2,-2) rectangle (2,2);
\draw [name path=curve 1]
(-2,-1) .. controls (8,-1) and (-8,1) .. (2,1);
\draw [name path=curve 2]
(-1,-2) .. controls (-1,8) and (1,-8) .. (1,2);
\fill [name intersections={
      of=curve 1 and curve 2, by={[label=center:a],
      [label=center:...], [label=center:i]}]}]
\end{tikzpicture}

```



`/tikz/intersection/sort by=<path name>`

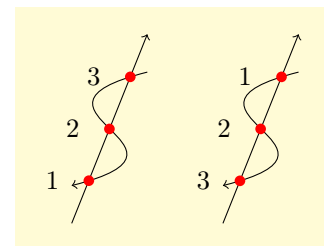
(no default)

По умолчанию, точки пересечения возвращаются в том порядке, в котором их вычисляет используемый алгоритм. Это не всегда разумный порядок. Опция позволяет отсортировать точки пересечения вдоль пути, определяемого параметром `<path name>`, который должен быть одним из пересекающихся путей.

```

\begin{tikzpicture}
\clip (-0.5,-0.75) rectangle (3.25,2.25);
\foreach \pathname/\shift in
  {line/0cm, curve/2cm}{\tikzset{xshift=\shift}}
\draw [->, name path=curve]
(1,1.5) .. controls (-1,1) and (2,0.5) .. (0,0);
\draw [->, name path=line] (0,-.5) -- (1,2);
\fill [red, name intersections=
      {of=line and curve, sort by=\pathname, name=i}]
\foreach \s in {1,2,3}{(i-\s) circle (2pt)
      node [left=.25cm, black] {\footnotesize\s}};
\end{tikzpicture}

```

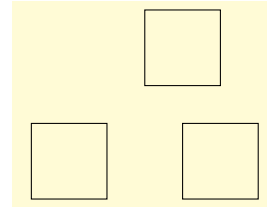




## 10.3 Относительные координаты

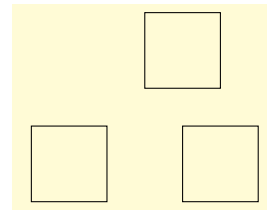
Можно предварить координаты префиксом `++`, делая их относительными. Координаты, типа `++(1cm, 1pt)` означают «на 1cm вправо и на 1pt вверх от предыдущей позиции». Относительные координаты часто полезны в локальных контекстах:

```
\begin{tikzpicture}
\draw (0,0) -- ++(1,0) -- ++(0,1)
           -- ++(-1,0) -- cycle;
\draw (2,0) -- ++(1,0) -- ++(0,1)
           -- ++(-1,0) -- cycle;
\draw (1.5,1.5) -- ++(1,0) -- ++(0,1)
           -- ++(-1,0) -- cycle;
\end{tikzpicture}
```



Вместо `++` можно использовать единственный `+`, также определяя относительные координаты, но не изменяя текущую точку для последующих вычислений относительных координат. Таким образом, чтобы нарисовать предыдущий рисунок, можно использовать только одиночный `+`:

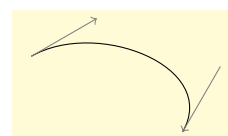
```
\begin{tikzpicture}
\draw (0,0) -- +(1,0) -- +(1,1) -- +(0,1) -- cycle;
\draw (2,0) -- +(1,0) -- +(1,1) -- +(0,1) -- cycle;
\draw (1.5,1.5) -- +(1,0) -- +(1,1) -- +(0,1)
                -- cycle;
\end{tikzpicture}
```



Есть специальные ситуации, когда относительные координаты интерпретируются по-другому. Если относительные координаты используются в качестве контрольной точки при построении кривой Безье, применяется следующее правило: (1) первая относительная контрольная точка вычисляется относительно начала кривой; (2) вторая относительная контрольная точка вычисляется относительно конца кривой; (3) относительная точка конца кривой вычисляется относительно начала кривой.

Такое специальное поведение облегчает определение того, что кривая в начальной или конечной точках должна покидать точку или прибывать в точку в определенном направлении. В следующем примере, кривая покидает начальную точку в направлении  $30^\circ$  и прибывает в конечную точку в направлении  $60^\circ$ :

```
\begin{tikzpicture}
\draw (1,0) .. controls +(30:1cm)
                and +(60:1cm) .. (3,-1);
\draw[gray,->] (1,0) -- +(30:1cm);
\draw[gray,<-] (3,-1) -- +(60:1cm);
\end{tikzpicture}
```



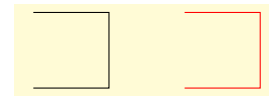
## Относительные координаты и области видимости

Возникает вопрос: как относительные координаты ведут себя в присутствии областей видимости? Если в пути используются фигурные скобки, делая его часть локальной, как это воздействует на текущую позицию? С одной стороны, текущая позиция, конечно, изменится, так как область видимости воздействует только на опции, а не на сам путь. С другой стороны, иногда нужно временно отказаться от обновления текущей точки. Так как обе интерпретации взаимодействия текущей точки и область видимости полезны, есть локальная опция, которая позволяет решить, что делать.

`/tikz/current point is local=<boolean>` (no default, initially false)

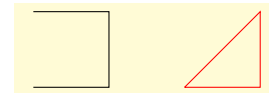
Обычно, операция по изменению области видимости в пути не влияет на текущую точку. Таким образом, фигурные скобки в пути не влияют и на текущую позицию:

```
\begin{tikzpicture}
\draw (0,0) -- ++ (1,0) -- ++ (0,1) -- ++ (-1,0);
\draw [red] (2,0) -- ++ (1,0) {-- ++ (0,1)}
-- ++ (-1,0);
\end{tikzpicture}
```



Если установить значение этой опции в `true` (истина), поведение изменится. В таком случае, в конце группы, созданной в пути, последняя текущая позиция возвратится к тому значению, которое было в начале области видимости. Более точно, когда TikZ сталкивается с `}` на пути, он проверяет, имеет ли опция значение `true`. Если это так, текущая позиция возвращается к значению, перед прочтением `{`.

```
\begin{tikzpicture}
\draw (0,0) -- ++ (1,0) -- ++ (0,1) -- ++ (-1,0);
\draw [red] (2,0) -- ++ (1,0)
{[current point is local] -- ++ (0,1)} -- ++ (-1,0);
\end{tikzpicture}
```

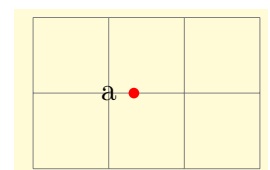


В примере выше, можно задать значение опции вне локальной области видимости, например, как параметр окружения `scope`.

## 10.4 Вычисление координат

Чтобы использовать функции вычисления координат (сложение и вычитание, умножение на число или масштабирование, вычисление середины, проектирование), нужно загрузить библиотеку `calc`. Например, код  $(a) + 1/3*(1\text{cm},0)$  возвратит точку, расположенную на  $1/3\text{cm}$  правее точки  $a$ :

```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (3,2);
\node (a) at (1,1) {a};
\fill [red] ($(a) + 1/3*(1\text{cm},0)$) circle (2pt);
\end{tikzpicture}
```



Основной синтаксис (`[<options>] $<coordinate computation>$`) библиотеки `calc` использует символы `$. . $` среды `TeX`, чтобы указать на вычисления. Но никакого другого смысла они здесь не имеют, в частности, не указывают на набор математического текста. Структура `<coordinate computation>` имеет следующий вид:

1. Она начинается с выражения `<factor>*<coordinate><modifiers>`.
2. Такое выражение может сопровождаться знаками `+` или `-`, а затем другим выражением вида `<factor>*<coordinate><modifiers>`.
3. Затем еще раз может ставиться знак `+` или `-` и записываться другая модификация координат; и так далее.

Далее этот синтаксис рассматривается более подробно, в том числе и на примерах.

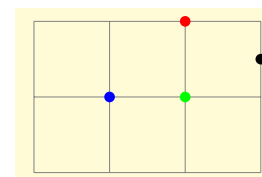
## Синтаксис коэффициентов

Множитель (или коэффициент) `<factor>` — необязателен и обнаруживается при проверке, начинается ли опция `<coordinate computation>` с открывающей круглой скобки `(`. Кроме того, после каждого знака  $\pm$  множитель `<factor>` присутствует, если и только если, знаки `+` или `-` непосредственно не сопровождаются скобкой `(`.

Если `<factor>` присутствует, он вычисляется макросом `\pgfmathparse`. Это означает, что можно использовать сложные вычисления в самих коэффициентах; `<factor>` может даже содержать открывающие круглые скобки, которые усложняют анализ. Как `TikZ` узнает, где `<factor>` заканчивается и где начинаются координаты? В случае, если `<coordinate computation>` начинается с `2*(3+4 . . .` не ясно, является ли `3+4` частью `<coordinate>` или частью `<factor>`. Поэтому, используется следующее правило: как только определено, что `<factor>` присутствует, `<factor>` содержит все до следующих символов `*`. При этом не должно быть пробела между звездочкой и круглой скобкой.

Допустимо помещать `<factor>` в фигурные скобки, что можно использовать всякий раз, когда неясно где `<factor>` заканчивается. Приведем примеры вычисления координат, которые состоят точно из одного `<factor>` и одного `<coordinate>`.

```
\begin{tikzpicture}
\draw[help lines] (0,0) grid (3,2);
\fill[red] ($2*(1,1)$) circle (2pt);
\fill[green] ($\{1+1\}*(1,.5)$) circle (2pt);
\fill[blue] ($\cos(0)*\sin(90)*(1,1)$) circle (2pt);
\fill[black] ($\{3*(4-3)\}*(1,0.5)$) circle (2pt);
\end{tikzpicture}
```



## Синтаксис модификаторов пути

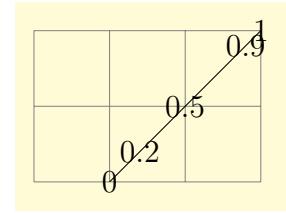
`<Coordinate>` может сопровождаться разными модификаторами. Первый вид модификатора — модификатор пути. Его синтаксис (подсмотрен в пакете `xcolor`!) таков:

`<coordinate>!<number>!<angle>:<second coordinate>`

Код `(1,2)!.75!(3,4)` означает: используется точка, расположенная в трех четвертях пути от точки `(1,2)` до точки `(3,4)`. Вообще говоря, выражение `<x>!<n>!<y>` приводит к точке `(1 - <n>)<x> + <n><y>`. Когда число `<n>` лежит между 0 и 1, получаем точку на прямой между `<x>` и `<y>`. Допустимо использовать числа, которые меньше 0

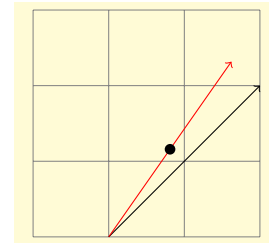
или больше 1. Число  $\langle n \rangle$  вычисляется, используя команду `\pgfmathparse`, что позволяет выполнять сложные вычисления.

```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (3,2);
\draw (1,0) -- (3,2);
\foreach \i in {0,0.2,0.5,0.9,1}
  \node at ($(1,0)!\i!(3,2)$) {\i};
\end{tikzpicture}
```

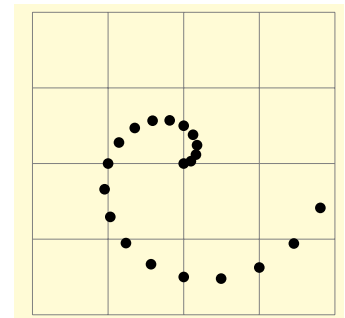


Опция `<second coordinate>` может иметь префикс `<angle>`, отделенный двоеточием, например, `(1,1)!.5!60:(2,2)`. Вообще говоря, `<x>!\factor!\angle:<y>`, означает: провести прямую от точки `<x>` до точки `<y>`, повернуть прямую на угол `<angle>` вокруг точки `<x>`; получить две конечные точки новой прямой `<x>` и `<z>`, используя их, выполнить модификацию пути.

```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (3,3);
\coordinate (a) at (1,0); \coordinate (b) at (3,2);
\draw[->] (a) -- (b);
\coordinate (c) at ($(a)!.5!10:(b)$);
\draw[->,red] (a) -- (c);
\fill ($(a)!.5!10:(b)$) circle (2pt);
\end{tikzpicture}
```

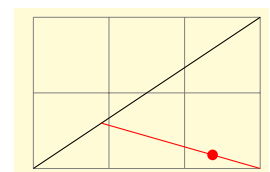


```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (4,4);
\foreach \i in {0,0.1,...,2}
  \fill ($(2,2)!\i!\i*180:(3,2)$) circle (2pt);
\end{tikzpicture}
```



Модификаторы можно применять многократно. Таким образом, после любого модификатора можно добавить другой (возможно отличный от предыдущего).

```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (3,2);
\draw (0,0) -- (3,2);
\draw[red] ($(0,0)!.3!(3,2)$) -- (3,0);
\fill[red] ($(0,0)!.3!(3,2)!.7!(3,0)$) circle (2pt);
\end{tikzpicture}
```



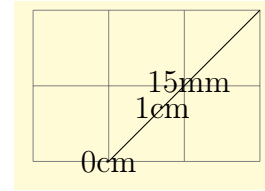
## Синтаксис модификаторов расстояния

Модификатор расстояния имеет почти тот же синтаксис, что и модификатор пути, только используется `<dimension>` (например, `1cm`), вместо `<factor>` (например, `5`):

`<coordinate>!\color{red}<dimension>!\color{green}<angle>:\color{red}<second coordinate>`

Код `<a>!<dimension>!<b>`, означает: использовать точку, которая находится на расстоянии `<dimension>` от точки `<a>` на прямой, проведенной через точки `<a>` и `<b>`.

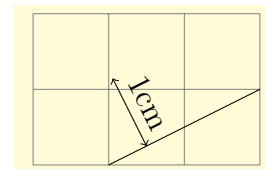
```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (3,2);
\draw (1,0) -- (3,2);
\foreach \i in {0cm,1cm,15mm}
  \node at ($(1,0)! \i!(3,2)$) {\i};
\end{tikzpicture}
```



Как и прежде, если используется `<angle>`, точка `<second coordinate>` вращается вокруг `<coordinate>` на указанный угол, а затем используется в вычислениях.

Поворот на угол `<angle>` в  $90^\circ$  и опцию `<dimension>` можно использоваться для сдвига точки относительно прямой. Предположим, что вычислена точка (c), лежащая на прямой от (a) до (b), и нужно ее сдвинуть на 1cm так, чтобы расстояние от сдвинутой точка до прямой было равно 1cm. Это можно сделать следующим так:

```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (3,2);
\coordinate (a) at (1,0);
\coordinate (b) at (3,1);
\draw (a) -- (b);
\coordinate (c) at ($(a)!.25!(b)$);
\coordinate (d) at ($(c)!1cm!90:(b)$);
\draw [<->] (c) -- (d) node [sloped,midway,above] {1cm};
\end{tikzpicture}
```



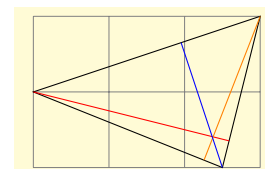
## Синтаксис модификаторов проектирования

Синтаксис модификатора проектирования подобен синтаксису вышеупомянутых модификаторов:

`<coordinate>!<projection coordinate>!<angle>:<second coordinate>`

Например, код `(1,2)!(0,5)!(3,4)` означает: ортогонально спроектировать точку `<projection coordinate>` (в примере, точку `(0,5)`) на прямую линию, проходящую через точки `<coordinate>` и `<second coordinate>`, то есть найти проекцию точки на прямую (в примере, на прямую, проходящую через точки `(1,2)` и `(3,4)`).

```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (3,2);
\coordinate (a) at (0,1);
\coordinate (b) at (3,2);
\coordinate (c) at (2.5,0);
\draw (a) -- (b) -- (c) -- cycle; \draw[red] (a) -- ($(b)!(a)!(c)$);
\draw[orange] (b) -- ($(a)!(b)!(c)$); \draw[blue] (c) -- ($(a)!(c)!(b)$);
\end{tikzpicture}
```



# Глава 11

## Синтаксис спецификаций пути

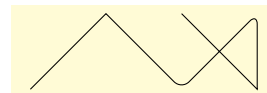
Путь — ряд прямолинейных и криволинейных сегментов, вводимых после команды `\path` и следующих специальному синтаксису, который описывается в этой главе.

`\path`<specification>;

Команда доступна только в окружении `{tikzpicture}`, <specification> — поток операций пути, которые «говорят» TikZ, как строить путь. Например, код `-- (0,0)` использует операцию `line-to`, которая продолжает путь в виде прямой линией от текущей точки до точки  $(0,0)$ . В любой точке, где TikZ ожидает операцию пути, можно задавать в квадратных скобках список графических опций, например, `[rounded corners]`.

1. Некоторые опции действуют немедленно и применяются ко всем последующим операциям пути. Например, опция скругления углов `[rounded corners]` будет скруглять все последующие углы, но не углы, расположенные до этой опции. Если позже будет задана опция `sharp corners` (в новых квадратных скобках), операция по скруглению углов перестанет работать.

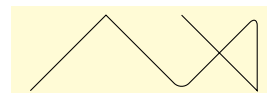
```
\tikz \draw (0,0) -- (1,1)
[rounded corners] -- (2,0) -- (3,1)
[sharp corners] -- (3,0) -- (2,1);
```



Другой пример — опции преобразования, которые применяются только к последующим координатам.

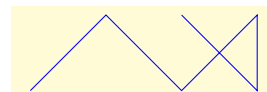
2. Опции, которые дают непосредственный результат, можно поместить в область видимости, например, помещая часть пути в фигурные скобки. Так, пример, приведенный выше, можно написать и следующим образом:

```
\tikz \draw (0,0) -- (1,1)
{[rounded corners] -- (2,0) -- (3,1)}
-- (3,0) -- (2,1);
```



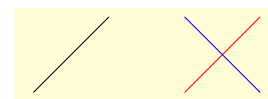
3. Некоторые опции относятся только к пути в целом. Такой является, например, опция `color=`, определяющая цвет пути. Если по мере создания пути задается несколько опций цвета, применяется ко всему пути только последняя:

```
\tikz \draw (0,0) -- (1,1)
[red] -- (2,0) -- (3,1) [blue] -- (3,0) -- (2,1);
```



Большинство опций имеет именно такой тип. В примере выше, чтобы нарисовать разноцветный путь, его надо расщепить на несколько команд `\path`:

```
\tikz{\draw (0,0) -- (1,1);
\draw [red] (2,0) -- (3,1);
\draw [blue] (3,0) -- (2,1);}
```



По умолчанию, команда `\path` ничего не делает с путем, а только приказывает начать создавать путь. Таким образом, если написать `\path (0,0) -- (1,1);`, ничего не будет нарисовано. Единственное, что произойдет, под рисунок будет выделена достаточная для него область. Фактически, чтобы что-то сделать с путем, нужно где-то на пути задать опции, подобные `draw` или `fill`.

Наконец, в любом месте пути можно задать спецификацию `node`. В основном, узлы применяются для того, чтобы набрать нормальный текст TeX'a и поместить его в текущей позиции пути (см. раздел 13).

Важно отметить, что узлы не являются частью пути. После того, как путь создан в соответствии с опциями пути, узлы добавляются в путь. Следующий стиль оказывает влияние на область видимости.

`/tikz/every path` (style, initially empty)

Стиль устанавливается в начале каждого пути, что полезно для временного добавления нужных опций.

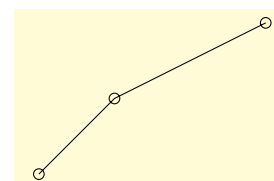
```
\begin{tikzpicture}
[fill=yellow,every path/.style={draw}]
\fill (0,0) rectangle +(1,1);
\shade (2,0) rectangle +(1,1);
\end{tikzpicture}
```



`/tikz/insert path=<path>` (no default)

Ключ может использоваться внутри опции, позволяя добавить что-либо к текущему пути. Главным образом используется в определении стилей, создающих графический контекст. Надо быть осторожным и не использовать опцию в качестве параметра, скажем, узла. В примере, используя стиль, добавим в путь небольшие круги.

```
\tikz
[c/.style={insert path={circle[radius=2pt]}}]
\draw (0,0) [c] -- (1,1) [c] -- (3,2) [c];
```



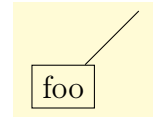
Код `(0,0) -- (1,1) circle[radius=2pt] -- (3,2) circle[radius=2pt]` дает тот же результат.

Следующие две опции — только для опытных пользователей.

`/tikz/append after command=<path>` (no default)

Некоторые из команд пути принимают необязательные опции. Для таких команд, когда внутри их опций используется этот ключ, путь `<path>` будет вставляться после выполнения команды пути. Например, когда такая команда задается в списке опций узла, путь `<path>` будет рисоваться после узла.

```
\tikz \draw node [append after command=
                {(foo)--(1,1)},draw] (foo){foo};
```



Если этот ключ вызывать несколько раз, эффект накапливается, то есть, указанные в ключе пути добавляются в том порядке, в каком были расположены сами ключи.

```
/tikz/prefix after command=<path> (no default)
```

Работает подобно `append after command`, только порядок накопления обратный: путь `<path>` добавляется до любого ранее добавленного пути, при добавлении которого использовался ключ `append after command` или `prefix after command`.

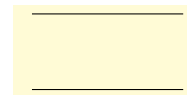
## 11.1 Операция перемещения

Возможно самая простая операция — это операция перемещения (`move-to`), которая определяется только заданием координаты, через которую проходит путь:

```
\path ... <coordinate> ...;
```

Операция перемещения начинает путь в указанной точке, которая становится началом следующего сегмента. Если ранее уже были созданы сегменты пути, операция перемещения начнет строить новый сегмент пути, не связанный с предыдущими.

```
\begin {tikzpicture}
\draw (0,0) -- (2,0) (0,1) -- (2,1);
\end {tikzpicture}
```



В коде `(0,0) -- (2,0) (0,1) -- (2,1)` есть две операции перемещения: `(0,0)` и `(0,1)`. А операции `-- (2,0)` и `-- (2,1)` — это операции построения линии `line-to`.

## 11.2 Операции построения линий

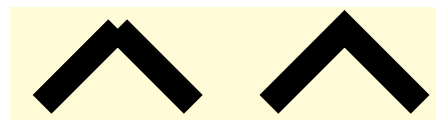
### Прямые линии

```
\path ... -- <coordinate>...;
```

Операция построения линии `--` (`line-to`) расширяет текущий путь от текущей точки до заданной точки по прямой. Текущая точка является конечной точкой предыдущей операции рисования или точкой, определяющей операцию перемещения.

Когда используется операция построения линии, и некоторый сегмент пути уже создан, например, другая прямая линия, эти два линейных сегмента соединяются. Это означает что, если они рисуются, точка, в которой они встречаются, «сглаживается». Чтобы оценить различия, рассмотрим следующие два примера: в левом примере, путь состоит из двух сегментов, которые не соединяются, но, так случилось, имеют общую точку, в то время как в правом примере показано гладкое соединение.

```
\begin{tikzpicture}[line width=10pt]
\draw (0,0) -- (1,1) (1,1) -- (2,0);
\draw (3,0) -- (4,1) -- (5,0);
\end{tikzpicture}
```





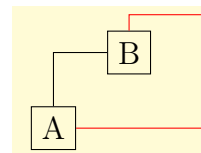
## Горизонтальные и вертикальные линии

Иногда надо соединить две точки только горизонтальными и вертикальными прямыми. Для этого можно использовать две операции конструирования пути.

```
\path ... -| <coordinate>...; \path ... |- <coordinate>...;
```

Сначала провести горизонтальную (вертикальную) линию, а затем вертикальную (горизонтальную).

```
\begin{tikzpicture}
\draw (0,0) node(a) [draw] {A}
      (1,1) node(b) [draw] {B};
\draw (a.north) |- (b.west);
\draw[red] (a.east) -| (2,1.5) -| (b.north);
\end{tikzpicture}
```

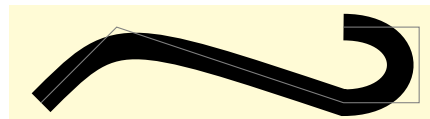


## 11.3 Кривые Безье

```
\path ... ..<controls> <c>and<d> ..<y> ...;
```

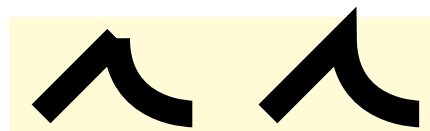
Рисует путь в виде кубической кривой Безье, соединяя текущую точку  $\langle x \rangle$  с точкой  $\langle y \rangle$  с помощью двух контрольных точек  $\langle c \rangle$  и  $\langle d \rangle$ : касательная к кривой в точке  $\langle x \rangle$  проходит через  $\langle c \rangle$ , а касательная к кривой в точке  $\langle y \rangle$  проходит через  $\langle d \rangle$ . Если часть `and<d>` отсутствует, считается, что  $\langle d \rangle = \langle c \rangle$ .

```
\begin{tikzpicture}
\draw[line width=10pt]
  (0,0) .. controls (1,1) .. (4,0)
  .. controls (5,0) and (5,1) .. (4,1);
\draw[gray] (0,0) -- (1,1) -- (4,0) -- (5,0) -- (5,1) -- (4,1);
\end{tikzpicture}
```



Как и с прямыми линиями, есть различие: связываются ли две кривые или только имеют общую точку.

```
\begin{tikzpicture}[line width=10pt]
\draw (0,0) -- (1,1) (1,1) .. controls
      (1,0) and (2,0) .. (2,0);
\draw (3,0) -- (4,1) .. controls
      (4,0) and (5,0) .. (5,0);
\end{tikzpicture}
```



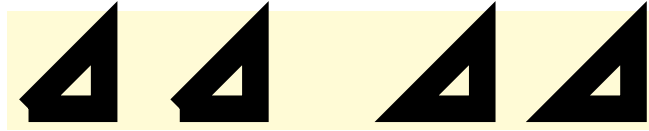
## 11.4 Операция цикла

```
\path ... -- cycle ...;
```

Операция рисует прямую от текущей точки до последней точки, определенной операцией перемещения (это не обязательно точка начала пути). Кроме того, создается

гладкое соединение. Например, слева два треугольника создаются, используя три прямые, но они не связаны на концах, справа используются операции цикла.

```
\begin{tikzpicture}[line width=10pt]
\draw (0,0) -- (1,1) -- (1,0) -- (0,0) (2,0) -- (3,1) -- (3,0) -- (2,0);
\draw (5,0) -- (6,1) -- (6,0) -- cycle (7,0) -- (8,1) -- (8,0) -- cycle;
\end{tikzpicture}
```



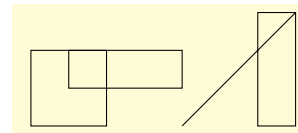
## 11.5 Операция `rectangle`

Прямоугольник можно создать, используя четыре прямые и операцию цикла. Но так как прямоугольники нужны часто, для их создания есть специальный синтаксис.

```
\path ... rectangle<corner> ...;
```

Когда используется эта операция, один угол — текущая точка, другой угол задается параметром `<corner>`, который становится новой текущей точкой.

```
\begin{tikzpicture}
\draw (0,0) rectangle (1,1);
\draw (.5,1) rectangle (2,0.5)
      (3,0) rectangle (3.5,1.5) -- (2,0);
\end{tikzpicture}
```

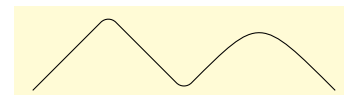


## 11.6 Скругление углов

На все операции конструирования пути, упомянутые выше, влияет опция `/tikz/rounded corners=<inset>` (default 4pt)

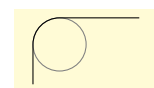
Все углы (места, где линия продолжается или операцией построения прямой или операцией построения кривой) заменяются небольшими дугами так, чтобы угол стал гладким.

```
\tikz \draw [rounded corners] (0,0) -- (1,1)
      -- (2,0) .. controls (3,1) .. (4,0);
```



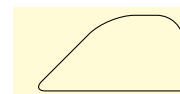
Параметр `<inset>` описывает, насколько велико скругление. Отметим, что `<inset>` не масштабируется, если используется такая опция, как `scale=2`.

```
\begin{tikzpicture}
\draw[gray,thin] (10pt,15pt) circle[radius=10pt];
\draw[rounded corners=10pt]
      (0,0) -- (0pt,25pt) -- (40pt,25pt);
\end{tikzpicture}
```



Можно включать скругление углов только в середину пути, можно использовать в одном и том же пути разные радиусы скругления:

```
\begin{tikzpicture}
\draw (0,0)[rounded corners=10pt] -- (1,1) -- (2,1)
      [sharp corners] -- (2,0)
      [rounded corners=5pt] -- cycle;
\end{tikzpicture}
```



А вот маленький прямоугольник со скругленными углами:

```
\tikz \draw[rounded corners=1ex](0,0) rectangle (20pt,2ex);
```



Используя данную опцию, надо иметь ввиду несколько особенностей. Первое, скругление угла дугой (частью круга) возможно, если угол равен  $90^\circ$ . В других случаях, скругление возможно, но может дать плохой результат. Второе, если в пути есть короткие линейные сегменты, скругление может вызвать корявые разрывы пути. В таких случаях стоит использовать опцию `sharp corners`, которая отменяет скругление на последующих углах пути.

## 11.7 Круг и эллипс

Круги и эллипсы — общие элементы пути, для которых есть специальные операции.

```
\path ... circle[<options>] ...;
```

Команда добавляет в текущий путь круг, центр круга — по умолчанию текущая точка, но его можно с помощью опции изменить. Новой текущей точкой пути обычно остается центр круга. Радиус круга определяется, используя следующие опции:

```
/tikz/x radius=<value> (no default)
```

Устанавливает горизонтальный радиус круга (который, когда это значение отличается от вертикального радиуса, является эллипсом); `<value>` может быть измеримой или безразмерной величиной. В последнем случае, число интерпретируется в  $xy$ -системе координат (если «единица» на оси  $OX$  равна 2cm, то код `x radius=3` будет представлять радиус в 6cm).

```
/tikz/y radius=<value> (no default)
```

Работает как `x radius`.

```
/tikz/radius=<value> (no default)
```

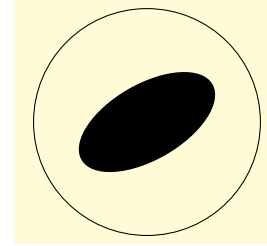
Устанавливает равными `<value>` и `x radius`, и `y radius`.

```
/tikz/at=<coordinate> (no default)
```

Если опция установлена явно в `<options>` (или косвенно через стиль `every circle`), `<coordinate>` используется как центр круга вместо текущей точки. Установка `at` к некоторому значению в замкнутой области видимости не даст результата.

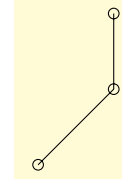
Список опций `<options>` может содержать и другие необязательные опции, скажем, `rotate` или `scale`, которые будут воздействовать только на круг.

```
\begin{tikzpicture}
\draw (1,0) circle [radius=1.5];
\fill (1,0) circle [x radius=1cm,
y radius=5mm,
rotate=30];
\end{tikzpicture}
```



Можно установить радиус в некоторой замкнутой области видимости, но в этом случае опции могут теряться (см. замечание ниже).

```
\begin{tikzpicture}[radius=2pt]
\draw (0,0) circle -- (1,1)
circle -- ++(0,1) circle;
\end{tikzpicture}
```



С каждым кругом используется стиль

`/tikz/every circle`

(style, no default)

Стиль можно использовать, скажем, для установки радиуса по умолчанию для каждого круга. Этот же стиль используется и с операцией `ellipse`.

В случае, если по каким-то причинам, имена `radius` и `x radius` слишком длинные, можно легко создать для них более короткие псевдонимы:

```
\tikzset{r/.style={radius=#1},rx/.style={x radius=#1},
ry/.style={y radius=#1}}
```

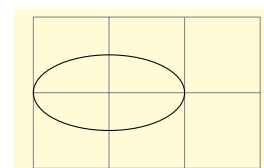
После этого можно сказать `circle[r=1cm]` или `circle[rx=1,ry=1.5]`. Причина, по которой TikZ использует по умолчанию более длинные имена состоит в том, чтобы поощрять людей писать более понятный код.

**Замечание.** Существует старый синтаксис определения круга, где радиус круга задается в круглых скобках прямо после команды `circle`: `circle (1pt)`. Хотя этот синтаксис немного короче, он тяжелее понимается читателем кода, а использование круглых скобок для чего-то другого, отличного от координат, тоже не приносит ясности. TikZ использует следующее правило определения, какой синтаксис используется: если `circle` сопровождается чем-то, что вводится в круглых скобках, то используется старый синтаксис; в круглых скобках должно стоять единственное число, представляющее радиус. Во всех других случаях используется новый синтаксис.

```
\path ... ellipse[<options>] ...;
```

Команда имеет то же смысл, что и `circle`. Старый синтаксис для этой команды `ellipse (<x radius> and <y radius>)`, но он хуже стандартного.

```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (3,2);
\draw (1,1) ellipse
[x radius=1cm,y radius=.5cm];
\end{tikzpicture}
```



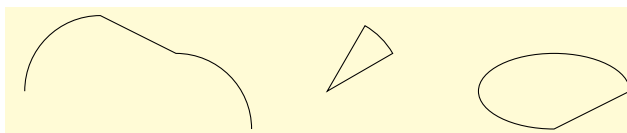
## 11.8 Операция arc

Операция `arc` позволяет добавить часть эллипса (дугу) в текущий путь.

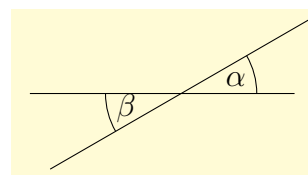
```
\path ... arc[<options>] ...;
```

В `<options>` задаются значения `x radius` и `y radius`. Дуга в пути начинается в текущей точке и заканчивается в конце указанной дуги. Дуга эллипса определяется по углам, вычисляемым по ключам `start angle`, `end angle`, `delta angle`. Обычно, начальный и конечный угол дуги определяют первые два ключа. Если один из них пуст, он вычисляется по другому ключу плюс или минус `delta angle`. Если установлены все три ключа, ключ `delta angle` игнорируется.

```
\begin{tikzpicture}[radius=1cm]
\draw (0,0) arc[start angle=180,end angle=90] -- (2,.5)
          arc[start angle=90,delta angle=-90];
\draw (4,0) -- +(30:1cm) arc [start angle=30,delta angle=30] -- cycle;
\draw (8,0) arc [start angle=0,end angle=270,x radius=1cm,y radius=5mm]
          -- cycle;
\end{tikzpicture}
```



```
\begin{tikzpicture}[radius=1cm,delta angle=30]
\draw (-1,0) -- +(3.5,0);
\draw (1,0) ++(210:2cm) -- +(30:4cm);
\draw (1,0) +(0:1cm) arc [start angle=0];
\draw (1,0) +(180:1cm) arc [start angle=180];
\path (1,0) ++(15:.75cm) node{\alpha};
\path (1,0) ++(15:-.75cm) node{\beta};
\end{tikzpicture}
```



Существует короткий синтаксис для операции `arc`:

```
arc (<start angle>:<end angle>:<radius>),
```

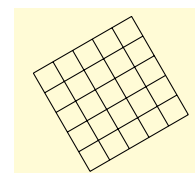
но этот синтаксис тяжелее читать; следует предпочесть нормальный синтаксис.

## 11.9 Операция grid

```
\path ... grid[<options>]<corner> ...;
```

Добавляет сетку, заполняющую прямоугольник, два угла которого задаются параметром `<corner>` и предыдущей точкой. Типичный способ создания сетки таков: `\draw (1,1) grid (3,3)`. Все координатные преобразования применимы к сетке.

```
\tikz[rotate=30]
\draw[step=3mm] (0,0) grid (1.5,1.5);
```

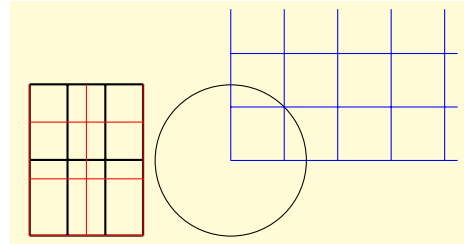


Список `<options>`, который локален для `grid`, может использоваться для воздействия на внешний вид сетки. Размерами ячеек сетки управляют следующие опции:

`/tikz/step=<number or dimension or coordinate>` (no default, initially 1cm)

Устанавливает шаг сетки по  $x$ - и  $y$ - координате. Если есть размерность, она и используется. Если задано только число, число интерпретируется в  $xy$ -системе координат. Например, если задано число 2, то  $x$ -шаг — удвоенный  $x$ -вектор, а  $y$ -шаг — удвоенный  $y$ -вектор, установленные опциями `x=` и `y=`. Если же задана точка, то ее  $x$ -координата —  $x$ -шаг, а  $y$ -координата —  $y$ -шаг.

```
\begin{tikzpicture}[x=.5cm]
\draw[thick] (0,0)
  grid [step=1] (3,2);
\draw[red] (0,0)
  grid [step=.75cm] (3,2);
\end{tikzpicture}
\begin{tikzpicture}
\draw (0,0) circle [radius=1];
\draw[blue] (0,0) grid [step=(45:1)] (3,2);
\end{tikzpicture}
```



Сложности возникают, когда по осям не указан  $x$ - и/или  $y$ - вектор. Тогда правило вычисления  $x$ -шага и  $y$ -шага следующее: в качестве  $x$ -шага и  $y$ -шага используются  $x$ - и  $y$ - компоненты или следующие два вектора: первый вектор (`<x-grid-step-number>;0`) или (`<x-grid-step-dimension>;0pt`), второй вектор (`0;<y-grid-step-number>`) или (`0pt;<x-grid-step-dimension>`).

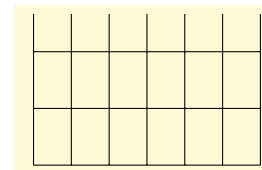
`/tikz/xstep=<number or dimension>` (no default, initially 1cm)

Устанавливает  $x$ -шаг по сетке.

`/tikz/ystep=<number or dimension>` (no default, initially 1cm)

Устанавливает  $y$ -шаг по сетке.

```
\tikz \draw (0,0)
  grid [xstep=.5,ystep=.75] (3,2);
```



Важно отметить, что сетка всегда располагается так, что содержит точку  $(0; 0)$ , если она попадает в прямоугольник. Таким образом, сетка не всегда будет иметь точку пересечения в точке `<corner>`. Отметим, что из-за ошибок округления, последняя линия сетки может оказаться немного ниже. В этом случае, следует прибавить небольшую величину к точке `<corner>`.

При создании сетки используется следующий стиль:

`/tikz/help lines` (style, initially line width=0.2pt,gray)

Стиль используется для построения приглушенной сетки, составленной из тонких серых линий. Однако, он не устанавливается автоматически и его нужно указать явно, например: `\tikz \draw[help lines] (0,0) grid (3,3);`.

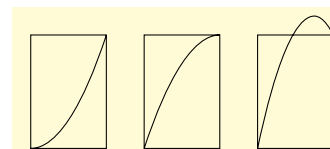
## 11.10 Операция parabola

Операция пути `parabola` позволяет продолжить текущий путь параболой. Парабола — (сдвинутая и отмасштабированная) кривая, задаваемая уравнением  $f(x) = x^2$ .

```
\path ... parabola[<options>]bend<bend coordinate><coordinate> ...;
```

Определяет параболу, проходящую через текущую точку и точку `<coordinate>`. Если параметр `bend` задан, он определяет, где будет находиться вершина параболы; чтобы определить точку расположения вершины параболы может также использоваться `<options>`. По умолчанию, вершина параболы находится в старой текущей точке.

```
\begin{tikzpicture}
\draw (0,0) rectangle (1,1.5)
      (0,0) parabola (1,1.5);
\draw[xshift=1.5cm] (0,0) rectangle (1,1.5)
      (0,0) parabola[bend at end] (1,1.5);
\draw[xshift=3cm] (0,0) rectangle (1,1.5)
      (0,0) parabola bend (.75,1.75) (1,1.5);
\end{tikzpicture}
```



На вид параболы можно повлиять с помощью следующих опций:

`/tikz/bend<coordinate>` (no default)

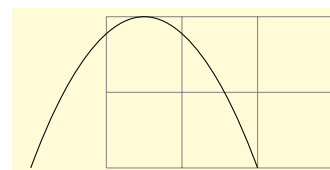
Дает тот же результат, что и код `bend<coordinate>`, расположенный вне `<options>`. Опция определяет, что вершина параболы должна находиться в точке `<coordinate>`. Нужно задавать допустимую точку для вершины параболы; это означает, что если нет параболы вида  $f(x) = ax^2 + bx + c$ , проходящей через старую текущую точку, заданную точку вершины параболы и новую точку, результатом не будет парабола.

Есть одно специальное свойство ключа `<coordinate>`: когда задается относительная координата, например, `+(0,0)`, эта точка лежит где-то на линии от старой текущей точки до новой текущей точки. Точная позиция зависит от опции

`/tikz/bend pos<fraction>` (no default)

Определяет, где располагается предыдущая точка, относительно которой вычисляется вершина параболы. Предыдущая точка будет в точке, определяемой `<fraction>`, на линии от старой текущей точки до новой текущей точкой. Идея состоит в следующем: если сказать `bend pos=0` и `bend +(0,0)`, вершина параболы будет в старой текущей точке; если сказать `bend pos=1` и `bend +(0,0)`, вершина параболы будет в новой текущей точке; если сказать `bend pos=0.5` и `bend +(0,2cm)`, вершина параболы будет на  $2cm$  выше середины линии, соединяющей начальную и конечную точки.

```
\begin{tikzpicture}
\draw[help lines] (0,0) grid (3,2);
\draw (-1,0) parabola[bend pos=0.5]
      bend +(0,2) +(3,0);
\end{tikzpicture}
```

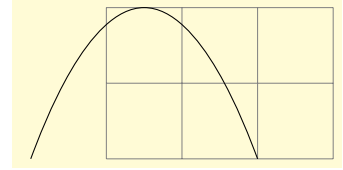


В этом примере код `bend +(0,2)` по существу означает: «парабола высотой  $2cm$ », а код `+(3,0)` означает: «и шириной  $3cm$ ». Поскольку такая ситуация возникает очень часто, есть специальная короткая опция:

`/tikz/parabola height=<dimension>` (no default)

То же, что и код `[bend pos=0.5,bend={+(0pt,<dimension>)}]`

```
\begin{tikzpicture}
\draw[help lines] (0,0) grid (3,2);
\draw (-1,0)
      parabola[parabola height=2cm] +(3,0);
\end{tikzpicture}
```



Часто полезны следующие стили:

`/tikz/bend at start` (style, no value)

Помещает вершину параболы в начало параболы. Это сокращение опции: `bend pos=0,bend={+(0,0)}`.

`/tikz/bend at end` (style, no value)

Помещает вершину параболы в конец параболы.

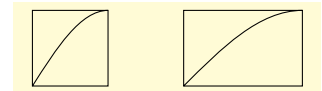
## 11.11 Операции `sin` и `cos`

Операции `sin` и `cos` подобны операции `parabola` и позволяют нарисовать части графиков функций  $y = \sin x$  и  $y = \cos x$ .

`\path ... sin<coordinate> ...;`

Рисует отмасштабированную и сдвинутую синусоиду из сегмента  $[0, \pi/2]$ . Масштабирование и смещение определяются так, чтобы начало синусоиды совпадало со старой текущей точкой, а конец — с точкой `<coordinate>`.

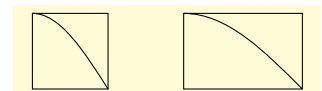
```
\tikz \draw
(0,0) rectangle (1,1) (0,0) sin (1,1)
(2,0) rectangle +(1.57,1) (2,0) sin +(1.57,1);
```



`\path ... cos<coordinate> ...;`

Рисует отмасштабированную и сдвинутую косинусоиду из сегмента  $[0, \pi/2]$ .

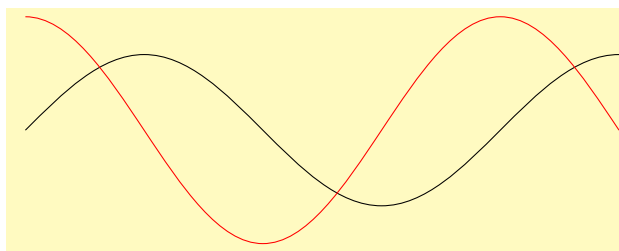
```
\tikz \draw
(0,0) rectangle (1,1) (0,1) cos (1,0)
(2,0) rectangle +(1.57,1) (2,1) cos +(1.57,-1);
```



Чередую операции `sin` и `cos`, можно создать полную синусоиду или косинусоиду:

```
\begin{tikzpicture}[xscale=1.57]
\draw (0,0) sin (1,1) cos (2,0) sin (3,-1) cos (4,0) sin (5,1);
\draw[red] (0,1.5) cos (1,0) sin (2,-1.5) cos (3,0) sin (4,1.5) cos (5,0);
\end{tikzpicture}
```





## 11.12 Операция plot

Операция `plot` позволяет присоединить к пути линию или кривую, проходящую через большое число точек. Эти точки или задаются в простом списке точек, который читается из некоторого файла, или вычисляются на лету. Так как синтаксис и поведение этой команды сложны, она описывается в отдельной главе 16.

## 11.13 Операция to

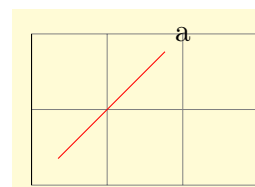
Операция `to` позволяет добавить определенный пользователем путь от предыдущей координаты до следующей координаты. Когда пишут `(a) to (b)`, добавляется прямая от  $a$  до  $b$  точно так же, как если бы было написано `(a) -- (b)`. Однако, если написать `(a) to [out=135, in=45] (b)`, к пути будет добавлена кривая, которая выходит под углом  $135^\circ$  из точки  $a$  и попадает в точку  $b$  под углом  $45^\circ$ . Это происходит потому, что опции `in` и `out` строят специальный путь, который и используется вместо прямой.

```
\path ... to[<options>]<nodes>(<coordinate>) ...;
```

Вставляет в текущий путь управляемый (возможно неявно) опциями `<options>` и сформированный ими путь. До того как путь будет вставлен, настраиваются макросы `\tikztostart`, `\tikztotarget`, `\tikztonodes`, которые могут помочь выполнить операцию (описаны ниже).

**Начальные и целевые точки.** Операция `to` всегда сопровождается целевой точкой `<coordinate>`. Макрос `\tikztotarget` устанавливает эту точку (без круглых скобок). Существует также начальная точка, которая является точкой, предшествующей операции `to`. Эта точка доступна через макрос `\tikztostart`. В следующем примере, для первой операции `to`, макрос `\tikztostart` устанавливает координату `(0pt, 0pt)`, а макрос `\tikztotarget` — `(0,2)`. Для второго `to`, макрос `\tikztostart` устанавливает координату `(10pt,10pt)`, а макрос `\tikztotarget` — `a`.

```
\begin{tikzpicture}
  \draw[help lines] (0,0) grid (3,2);
  \draw (0,0) to (0,2);
  \node (a) at (2,2) {a};
  \draw[color=red] (10pt,10pt) to (a);
\end{tikzpicture}
```

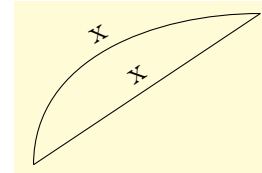


**Узлы для to.** Можно в путь, созданный операцией `to`, встроить узлы. Для этого надо определить узлы между ключевым словом `to` и координатой (если есть опции к операции `to`, они записываются первыми). Код `(a) to node {x} (b)` делает то же, что и код `(a) -- node {x} (b)`, а именно, узел помещается в путь, созданный `to`. Это можно использовать, например, для того, чтобы добавить в путь метку:

```

\begin{tikzpicture}
\draw (0,0) to node [sloped,above] {x} (3,2);
\draw (0,0) to[out=90,in=180]
      node [sloped,above] {x} (3,2);
\end{tikzpicture}

```



**Стили для to.** В дополнение к опциям `<options>`, задаваемым после операции `to`, в начале каждого пути, создаваемого `to`, устанавливается следующий стиль:

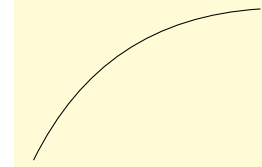
`/tikz/every to`

(style, initially empty)

```

\tikz[every to/.style={bend left}]
\draw (0,0) to (3,2);

```



**Опции.** Задаваемые с `to` опции `<options>` позволяют влиять на вид пути. Это можно использовать для того, чтобы изменить путь, например, от прямой линии до кривой, или расширить путь с помощью опции

`/tikz/to path=<path>`

(no default)

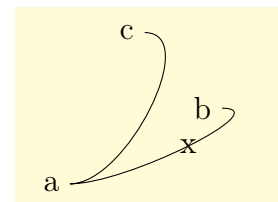
Всякий раз, когда используется операция `to`, вставляется путь `<path>`: точнее, добавляется путь `{[every to,<options>]<path>}`, где `<options>` — опции операции `to`, `<path>` — встраиваемый путь. В `<path>` могут использоваться макросы `\tikztostart`, `\tikztotarget` и `\tikztonodes` (см. предыдущую страницу). Макрос `tikztonodes` хранит узлы между операцией `to` и координатой; кроме того, эти узлы будут иметь опцию `pos`, установленную неявно.

Как явно видоизменить путь? Самый простой способ — использовать кривую.

```

\begin{tikzpicture}
[to path={.. controls +(1,0) and +(1,0) ..
          (\tikztotarget) \tikztonodes}]
\node (a) at (0,0) {a};
\node (b) at (2,1) {b};
\node (c) at (1,2) {c};
\draw (a) to node {x} (b) (a) to (c);
\end{tikzpicture}

```

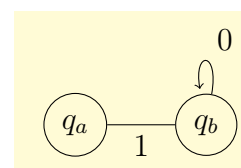


Вот другой пример:

```

\tikzset{my loop/.style={to path={.. controls +(80:1) and +(100:1) ..
          \tikztotarget) \tikztonodes}}, my state/.style={circle,draw}}
\begin{tikzpicture}[shorten >=2pt]
\node [my state] (a) at (210:1) {$q_a$};
\node [my state] (b) at (330:1) {$q_b$};
\draw[->] (a) to node[below] {1} (b)
      to [my loop] node[above right] {0} (b);
\end{tikzpicture}

```



`/tikz/execute at begin to=<code>` (no default)

Код `<code>` выполняется до операции `to`. Может использоваться для того, чтобы нарисовать дополнительные пути или сделать дополнительные вычисления.

`/tikz/execute at end to=<code>` (no default)

Опция аналогично предыдущей, но код выполняется после выполнения операция `to`.

Есть множество predefined вариантов для `to path`, они подгружаются при загрузке библиотеки `topaths` (см. [1, chapter 51]).

## 11.14 Операция let

Операция `let`, одна из тех операций пути, которые не продолжают путь, а дают другой, главным образом, локальный результат.

`\path ... let<assignment>,<assignment>,<assignment> ...in ...;`

Когда операция `let` возникает в пути, выражения `<assignment>` выполняются одно за другим, сохраняя координату и число в специальных регистрах (которые являются локальными в TikZ, и не имеют никакого отношения к TeX-регистрам). Впоследствии, можно обратиться к содержимому этих регистров, используя макросы `\p`, `\x`, `\y`, `\n`.

Первый вид допустимого выражения в `<assignment>` имеет форму

`\n<number register>={<formula>}`.

Когда она используется, выражение `<formula>` вычисляется, используя операцию `\pgfmathparse`. Результат сохраняется в регистре `<number register>`. Если `<formula>` вовлекает в вычисления размерные величины (например,  $2*3\text{cm}/2$ ), то числовой регистр `<number register>` сохраняет получившийся результат в виде размерной величины `pt`. Числовой регистр `<number register>` может иметь произвольное имя и являться обычным TeX-параметром для макроса `\n`. Возможные любые имена: `{left corner}` или даже одна цифра, например, `5`.

Если путь, который следует за операцией `let`, присвоить имя `body`, то в теле `body` макроккоманда `\n` может использоваться для обращения к регистру.

`\n{<number register>}`

Когда этот макрос используется в операции `let` слева от знака `=`, ничего не происходит и он вставляется только для удобочитаемости. Когда этот макрос используется справа от знака `=` или в теле операции `let`, то он разворачивается до значения, сохраненного в числовом регистре `<number register>`. Число может быть безразмерным, как `2.0`, или иметь размерность, как `5.6pt`.

Например, если сказать `let \n1={1pt+2pt}, \n2={1+2} in ...`, то внутри части `...` макрос `\n1` расширится до `3pt`, а `\n2` — до `3`.

Второй вид допустимого выражения в `<assignment>` имеет форму:

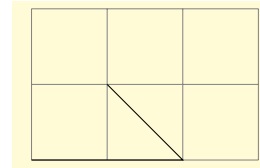
`\p<point register>={<formula>}`.

Регистр точки `<point register>` хранит точку, состоящую из  $x$ -части и  $y$ -части с размерностью в `pt`, но не может хранить узел или имя узла. Рассмотрим пример:

```

\begin{tikzpicture}
\draw [help lines] (0,0) grid (3,2);
\draw let \p{foo} = (1,1), \p2 = (2,0) in
      (0,0) -- (\p2) -- (\p{foo});
\end{tikzpicture}

```



**\p**{<point register>}

Когда макрос `\p` используется в операции `let` слева от `=`, ничего не происходит, он используется только для удобочитаемости. Когда макрос используется справа от `=` или в теле операции `let`, он расширяется до  $x$ -части (в `pt`), запятой и  $y$ -части (в `pt`) точки, сохраненной в регистре. Так если написать `let \p1=(1pt,1pt+2pt) in ...`, то в части `...` макрос `\p1` расширится до `(1pt,3pt)`.

**\x**{<point register>}

Макрос. Расширяется только до  $x$ -части регистра точки. Если написать выражение `let \p1=(1pt,1pt+2pt) in ...`, то в части `...` макрос `\x1` расширяется до `1pt`.

**\y**{<point register>}

Работает как `\x`, только для  $y$ -части регистра точки.

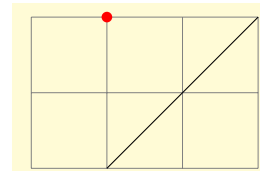
Отметим, что указанные выше макросы доступны только в операции `let`.

Рассмотрим пример, где используется выражение с `let`, чтобы сформировать точку из  $x$ -координаты первой точки и  $y$ -координаты второй точки. Используя операцию `|-`, это же можно было написать немного короче.

```

\begin{tikzpicture}
\draw [help lines] (0,0) grid (3,2);
\draw (1,0) coordinate (first point)
      -- (3,2) coordinate (second point);
\fill[red] let \p1 = (first point),
              \p2 = (second point) in
          (\x1,\y2) circle [radius=2pt];
\end{tikzpicture}

```

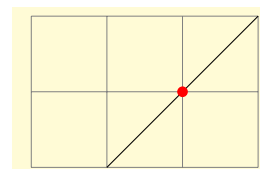


Отметим, что эффект от операции `let` локален для ее тела. Если нужно обратиться к вычисленной точке вне тела `let`, надо использовать операцию пути `coordinate`:

```

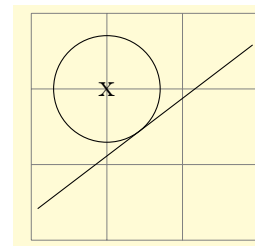
\begin{tikzpicture}
\draw [help lines] (0,0) grid (3,2);
\path % пусть определены некоторые точки:
      let \p1 = (1,0), \p2 = (3,2),
          \p{center} = ($ (\p1)!.5!(\p2) $)
      in coordinate (p1) at (\p1)
          coordinate (p2) at (\p2)
          coordinate (center) at (\p{center});
\draw (p1) -- (p2);
\fill[red] (center) circle [radius=2pt];
\end{tikzpicture}

```



Продemonстрируем более полезное применение операции `let` и нарисуем окружность, которая касается заданной линии:

```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (3,3);
\coordinate (a) at (rnd,rnd);
\coordinate (b) at (3-rnd,3-rnd);
\draw (a) -- (b);
\node (c) at (1,2) {x};
\draw let \p1 = ($ (a)!(c)!(b) - (c) $),
          \n1 = {veclen(\x1,\y1)}
          in circle [at=(c), radius=\n1];
\end{tikzpicture}
```



## 11.15 Операция scope

Когда TikZ в некоторой точке сталкивается и открывающей или закрывающей фигурной скобкой (`{` или `}`), он открывает или закрывает область видимости. Те опции, которые могут применяться локально, будут видны только в данной области видимости. Например, если применить преобразование вида `[xshift=1cm]` в области видимости, сдвиг произойдет только здесь. Но опция типа `color=red` не даст результата внутри области видимости, так как применима только к пути в целом (см. раздел 10.3).

## 11.16 Операции node и edge

Есть еще две операции, которые используются в пути: `node` (узел) и `edge` (ребро, грань, край). Первая используется для добавления узла в путь. Эта операция является особенной в следующем смысле: она никогда не изменяет текущий путь. Другими словами, эта операция реально не является операцией построения пути, а дает эффект, который является внешним для пути. Операция `edge` имеет тот же смысл и добавляет нечто в путь, после того, как основной путь уже нарисован. Однако, она работает подобно операции `to`, добавляя рисунок в путь, после прорисовки основного пути. Так как эти операции сложны, они подробно описываются в главе 13.

## 11.17 Особые PGF-операции

В некоторых случаях, возможно, будут нужны дополнительные вычисления или сущности. Для это нужно временно приостановить построение пути и анализ пути, затем выполнить некоторый TeX-код и возобновить прерванные операции. Этого можно добиться, используя операции пути

```
\pgfextra {<code>}
\pgfextra <code>\endpgfextra
```

Но эти операции должны использоваться только экспертами и только в «умном» макросе (см. [1, p. 152–153]).

# Глава 12

## Действия на пути

Как только путь создан, с ним многое можно сделать: нарисовать пером, заполнить цветом или оттенить, использовать для отсечения последующего рисунка, использовать для расширения рисунка, или выполнить любую комбинацию этих действий. Для определения пути используется команда `\path`, а опции, задаваемые для пути, указывают, что должно быть сделано с ним. Например, `\path (0,0) circle (1cm)`. Такой код создаст путь, состоящий из окружности с центром в начале координат. Опция `draw` или `fill`, вставленные в любом месте пути, нарисует путь «пером» и, соответственно, заполнит его цветом. Опции, передаваемые команде `\path`, накапливаются.


Как уже отмечалось, команда `\draw` — сокращение для `\path[draw]`, `\fill` — сокращение для `\path[fill]`, а `\filldraw` — сокращение для `\path[fill, draw]`. Кроме этих команд, еще определены команды

- `\pattern` — сокращение для `\path[pattern]`,
- `\shade` — сокращение для `\path[shade]`,
- `\shadedraw` — сокращение для `\path[shade,draw]`,
- `\clip` — сокращение для `\path[clip]`,
- `\useasboundingbox` — сокращение для `\path[use as bounding box]`.

Все они используются только внутри окружения `{tikzpicture}` или команды `\tikz`.

### 12.1 Определение цвета

`/tikz/color=<color name>` (no default)

Устанавливает цвет для команд рисования пути, заполнения пути и отображения текста в текущей области видимости. Любая установка цвета реализуется немедленно, при этом `<color name>` — название цвета, определенное пакетами `color` и/или `xcolor`. Например: `\tikz \fill[red!20] (0,0) circle (1ex);` 

Поскольку `pgf` имитирует поведение, определяемое пакетом `xcolor`, можно

- определять новый цвет, используя команду `\definecolor`, которая поддерживает цветовые модели `gray` и `rgb`, например, `\definecolor{orange}{rgb}{1,0.5,0}`;
- использовать команду `\colorlet` для определения нового цвета, основанного на старом, например, `\colorlet{lightgray}{black!25}`;
- использовать команду `\color<color name>`, чтобы установить цвет в текущей `TeX`-группе, а чтобы восстановить цвет после выхода из группы, использовать команду `\aftergroup`.

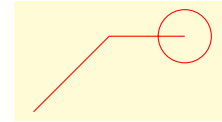
Есть несколько специализированных опций, устанавливающих цвет. Например, опции `draw=` и `fill=`. Все такие опции и их особенности описаны ниже.

## 12.2 Прорисовка пути

`/tikz/draw=<color>`

Рисует путь «пером на холсте». Если цвет задан, используется он. Если цвет не задан, используется последний цвет, используемый в области видимости.

```
\begin{tikzpicture}
\path[draw=red]
(0,0) -- (1,1) -- (2,1) circle (10pt);
\end{tikzpicture}
```



Рассмотрим по порядку все опции, которые влияют на представление пути. Все они имеют смысл, только если указана опция `draw` (прямо или косвенно).

### 12.2.1 Толщина, завершение, соединение

`/tikz/line width=<dimension>`

(no default, initially 0.4pt)

Позволяет явно указать ширину линии.

```
\tikz \draw[line width=5pt] (0,0) -- (1cm,1.5ex);
```

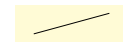


Есть множество predefined стилей, которые дают более естественные способы установки ширины линии. Эти стили можно переопределять.

`/tikz/ultra thin`

Устанавливает ширину линии равной 0.1pt.

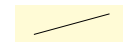
```
\tikz \draw[ultra thin] (0,0) -- (1cm,1.5ex);
```



`/tikz/very thin`

Устанавливает ширину линии равной 0.2pt.

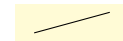
```
\tikz \draw[very thin] (0,0) -- (1cm,1.5ex);
```



`/tikz/thin`

Устанавливает ширину линии равной 0.4pt.

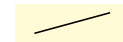
```
\tikz \draw[thin] (0,0) -- (1cm,1.5ex);
```



`/tikz/semithick`

Устанавливает ширину линии равной 0.6pt.

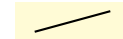
```
\tikz \draw[semithick] (0,0) -- (1cm,1.5ex);
```



`/tikz/thick`

Устанавливает ширину линии равной 0.8pt.

```
\tikz \draw[thick] (0,0) -- (1cm,1.5ex);
```



`/tikz/very thick`

Устанавливает ширину линии равной 1.2pt.

```
\tikz \draw[very thick] (0,0) -- (1cm,1.5ex);
```



`/tikz/ultra thick`

Устанавливает ширину линии равной 1.6pt.

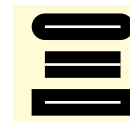
```
\tikz \draw[ultra thick] (0,0) -- (1cm,1.5ex);
```



`/tikz/line cap=` <type> (no default, initially butt)

Определяет, как линии завершаются. Допустимые типы: round, rect, butt.

```
\begin{tikzpicture}
\begin{scope}[line width=10pt]
\draw[line cap=rect] (0,0) -- (1,0);
\draw[line cap=butt] (0,.5) -- (1,.5);
\draw[line cap=round] (0,1) -- (1,1);
\end{scope}
\draw[white,line width=1pt]
(0,0) -- (1,0) (0,.5) -- (1,.5) (0,1) -- (1,1);
\end{tikzpicture}
```



`/tikz/line join=` <type> (no default, initially miter)

Определяет, как линии соединяются. Допустимые типы: round, bevel, miter.

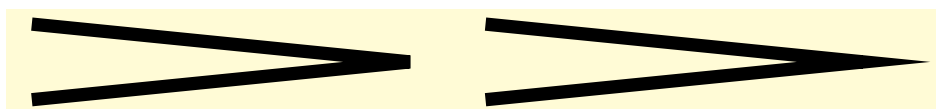
```
\begin{tikzpicture}[line width=7pt]
\draw[line join=round]
(0,0) -- ++(.5,1) -- ++(.5,-1);
\draw[line join=bevel]
(1.25,0) -- ++(.5,1) -- ++(.5,-1);
\draw[line join=miter]
(2.5,0) -- ++(.5,1) -- ++(.5,-1);
\useasboundingbox (0,1.5);% увеличивает ограничивающий прямоугольник
\end{tikzpicture}
```



`/tikz/miter limit=` <factor> (no default, initially 10)

Когда используется тип объединения miter и рисуется очень острый угол, такое объединение может очень далеко выходить за фактическую точку соединения. В этом случае, если размер удлинения будет превышать ширину линии более чем в <factor> раз, тип объединения miter заменяется на тип объединения bevel.

```
\begin{tikzpicture}[line width=5pt]
\draw (0,0) -- ++(5,.5) -- ++(-5,.5);
\draw[miter limit=25] (6,0) -- ++(5,.5) -- ++(-5,.5);
\useasboundingbox (12,0); % увеличивает ограничивающий прямоугольник
\end{tikzpicture}
```



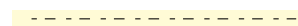
## 12.2.2 Пунктир

`/tikz/dash pattern=` <dash pattern> (no default)

Устанавливает шаблон пунктирной (штриховой линии). Например, следующий шаблон on 2pt off 3pt on 4pt off 4pt означает: нарисовать отрезок длиной 2pt, оставить пробел в 3pt, нарисовать отрезок в 4pt, оставить пробел в 4pt, и все повторить.



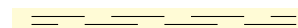
```
\begin{tikzpicture}
  [dash pattern=on 2pt off 3pt on 4pt off 4pt]
  \draw (0pt,0pt) -- (3.5cm,0pt);
\end{tikzpicture}
```



`/tikz/dash phase=` <dash phase> (no default, initially 0pt)

Сдвигает начало шаблона на <dash phase>.

```
\begin{tikzpicture}[dash pattern=on 20pt off 10pt]
  \draw[dash phase=0pt] (0pt,3pt) -- (3.5cm,3pt);
  \draw[dash phase=10pt] (0pt,0pt) -- (3.5cm,0pt);
\end{tikzpicture}
```



Что касается толщины линии, то некоторые predefined стили позволяют легко ее устанавливать.

`/tikz/solid` Устанавливает сплошную линию (значение по умолчанию).

```
\tikz \draw [solid] (0pt, 0pt) -- (50pt, 0pt);
```



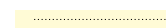
`/tikz/dotted` Устанавливает линию из точек.

```
\tikz \draw [dotted] (0pt, 0pt) -- (50pt, 0pt);
```



`/tikz/densely dotted` Устанавливает плотную точечную линию.

```
\tikz \draw [densely dotted]
  (0pt, 0pt) -- (50pt, 0pt);
```



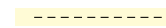
`/tikz/dashed` Устанавливает пунктирную линию.

```
\tikz \draw [dashed] (0pt, 0pt) -- (50pt, 0pt);
```



`/tikz/densely dashed` Устанавливает плотную пунктирную линию.

```
\tikz \draw [densely dashed]
  (0pt, 0pt) -- (50pt, 0pt);
```



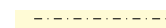
`/tikz/loosely dashed` Устанавливает свободную пунктирную линию.

```
\tikz \draw [loosely dashed]
  (0pt, 0pt) -- (50pt, 0pt);
```




`/tikz/dashdotted` Устанавливает точечно-пунктирную линию.

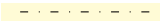
```
\tikz \draw [dashdotted]
  (0pt, 0pt) -- (50pt, 0pt);
```



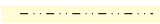
`/tikz/densely dashdotted` Устанавливает плотную точечно-пунктирную линию.

```
\tikz \draw [densely dashdotted] 
      (0pt, 0pt) -- (50pt, 0pt);
```

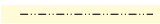
`/tikz/loosely dashdotted` Устанавливает свободную точечно-пунктирную линию.

```
\tikz \draw [loosely dashdotted] 
      (0pt, 0pt) -- (50pt, 0pt);
```

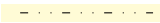
`/tikz/dashdotdotted` Устанавливает точечно-пунктирную линию с большим числом точек.

```
\tikz \draw [dashdotdotted] 
      (0pt, 0pt) -- (50pt, 0pt);
```

`/tikz/densely dashdotdotted` Устанавливает плотную точечно-пунктирную линию с большим числом точек.

```
\tikz \draw [densely dashdotdotted] 
      (0pt, 0pt) -- (50pt, 0pt);
```

`/tikz/loosely dashdotdotted` Устанавливает свободную точечно-пунктирную линию с большим числом точек.

```
\tikz \draw [loosely dashdotdotted] 
      (0pt, 0pt) -- (50pt, 0pt);
```

### 12.2.3 Непрозрачность

Когда линия проведена, она будет обычно темной, перекрывая все, что расположено позади нее, как-будто использовались совершенно непрозрачные чернила. Но можно «попросить TikZ» рисовать чуть-чуть (или совсем) прозрачными чернилами, используя опцию `draw opacity`. Аналогично можно «попросить TikZ» заполнить область чуть-чуть (или совсем) прозрачными красками (см. детали в главе 17).

### 12.2.4 Стрелки

Когда рисуется линия, можно добавить стрелки, расположенные на концах линии. Стрелки могут иметь разные наконечники, причем в начале пути может располагаться стрелка с одним наконечником, а в конце с другим. Поведение для замкнутого пути пока не определено.

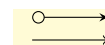
`/tikz/arrows=<start arrow kind>-<end arrow kind>` (no default)

Устанавливает наконечник стрелки и в начале, и в конце линии. Если опция имеет вид `arrows=<start arrow kind>-`, то стрелка рисуется только в начале. Если опция имеет вид `arrows=-<end arrow kind>`, то стрелка рисуется только в конце. Так как опция `arrows=` часто используется, можно опускать в коде имя опции `arrows=`, поскольку каждая опция, содержащая дефис -, интерпретируется как стрелка.

```

\begin{tikzpicture}
\draw[->] (0,0) -- (1,0);
\draw[o-stealth] (0,0.3) -- (1,0.3);
\end{tikzpicture}

```



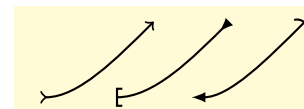
Допустимыми значениями являются все predefined наконечники стрелок, но можно определять новые виды наконечников (см. [1, главу 74]). Часто бывают нужны двойные наконечники и наконечники, имеющие фиксированный размер. Если нужны наконечники, отличные от значений по умолчанию, надо загрузить библиотеку `arrows` и использовать определенные там наконечники (см. [1, глава 23]).

Один вид наконечника особенный: `>`. Он не фиксирован и его можно определить, используя опцию `>=` (см. ниже). Например, можно в одном рисунке комбинировать разные виды наконечников:

```

\begin{tikzpicture}[thick]
\draw[to reversed-to] (0,0) .. controls +(.5,0)
and +(-.5,-.5) .. +(1.5,1);
\draw[[-latex reversed]](1,0) .. controls +(.5,0)
and +(-.5,-.5) .. +(1.5,1);
\draw[latex-]] (2,0) .. controls +(.5,0)
and +(-.5,-.5) .. +(1.5,1);
\useasboundingbox (-.1,-.1) rectangle (3.1,1.1);
\end{tikzpicture}

```



`/tikz/>=<end arrow kind>`

(no default)

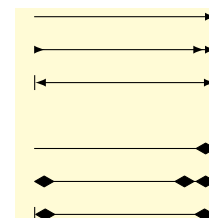
Определяет «стандартный» вид наконечника для стрелок. Например, чтобы сделать стандартным наконечник `\to` из среды `TEX`, нужно написать `>=to`, наконечник `\to` из среды `LATEX`, нужно написать `>=latex`, наконечник, подобной наконечнику из `pstricks`, нужно написать `>=stealth`.

Кроме определения наконечника вида `>` для конца пути (и, наконечника `<` для начала), эта опция также определяет вид наконечника для двойных стрелок `>>`, `<<` и для стрелок `|<` и `|>`, содержащих вертикальный отрезок.

```

\begin{tikzpicture}[scale=2.4]
\begin{scope}[>=latex]
\draw[->] (0pt,6ex) -- (1cm,6ex);
\draw[>->>] (0pt,5ex) -- (1cm,5ex);
\draw[|<->|] (0pt,4ex) -- (1cm,4ex);
\end{scope}
\begin{scope}[>=diamond]
\draw[->] (0pt,2ex) -- (1cm,2ex);
\draw[>->>] (0pt,1ex) -- (1cm,1ex);
\draw[|<->|] (0pt,0ex) -- (1cm,0ex);
\end{scope}
\end{tikzpicture}

```

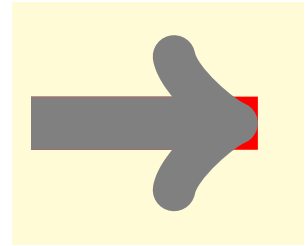


`/tikz/shorten >=<dimension>`

(no default, initially 0pt)

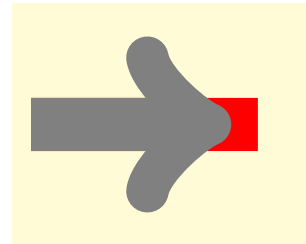
Опция сокращает конец линии на заданную длину `<dimension>`. Если определяется наконечник стрелки, линия сокращается так, что наконечник стрелки касается определенной конечной точки и не высовывается за эту точку. Например:

```
\begin{tikzpicture}[line width=20pt]
\useasboundingbox (0,-1.5) rectangle (3.5,1.5);
\draw[red] (0,0) -- (3,0);
\draw[gray,->] (0,0) -- (3,0);
\end{tikzpicture}
```



Опция `shorten >` позволяет сократить конец линии на дополнительно заданное расстояние и может быть полезной, если вообще нет наконечника.

```
\begin{tikzpicture}[line width=20pt]
\useasboundingbox (0,-1.5) rectangle (3.5,1.5);
\draw[red] (0,0) -- (3,0);
\draw[-to,shorten >=10pt,gray] (0,0) -- (3,0);
\end{tikzpicture}
```



`/tikz/shorten <=<dimension>`

(no default, initially 0pt)

Аналогична опции `shorten >`, но для начала линии.

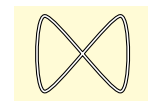
## 12.2.5 Двойные линии и граничные линии

`/tikz/double=<core color>`

(default white)

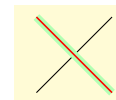
Заставляет нарисовать «две» линии вместо одной. Однако, в действительности, происходит следующее: путь рисуется дважды, сначала нормальным цветом, а затем цветом `<core color>`, который по умолчанию белый, но при рисовании второй линии, ширина линии уменьшается. Как итог, кажется, что были нарисованы две линии. Этот прием работает хорошо даже со сложными, изогнутыми путями:

```
\tikz \draw[double]
plot[smooth cycle]
coordinates{(0,0) (1,1) (1,0) (0,1)};
```



Используя опцию удвоения, можно создать эффект, при котором линия, выглядит, как имеющая некоторую границу:

```
\begin{tikzpicture}
\draw (0,0) -- (1,1);
\draw[draw=green,double=red,very thick] (0,1) -- (1,0);
\end{tikzpicture}
```



`/tikz/double distance=<dimension>`

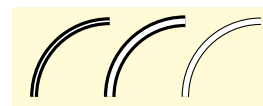
(no default, initially 0.6pt)

Устанавливает «расстояние» между двумя линиями. В действительности, это — толщина линии, рисуемой во второй раз. Толщина первой рисуемой линии — удвоенная нормальная ширина линии плюс `<dimension>`.

```

\begin{tikzpicture}
\draw[very thick,double] (0,0) arc (180:90:1cm);
\draw[very thick,double distance=2pt]
      (1,0) arc (180:90:1cm);
\draw[thin,double distance=2pt]
      (2,0) arc (180:90:1cm);
\end{tikzpicture}

```



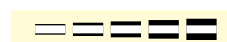
`/tikz/double distance between line centers=<dimension>` (no default)

Опция работает как `double distance`, только расстояние задается не между (внутренними) границами двух основных линий, а между их центрами.

```

\begin{tikzpicture}[double distance=3pt]
\foreach \lw in {0.5,1,1.5,2,2.5}
\draw[line width=\lw pt,double] (\lw,0) -- ++(4mm,0);
\end{tikzpicture}

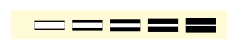
```



```

\begin{tikzpicture}
  [double distance between line centers=3pt]
\foreach \lw in {0.5,1,1.5,2,2.5}
\draw[line width=\lw pt,double] (\lw,0) -- ++(4mm,0);
\end{tikzpicture}

```



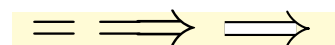
`/tikz/double equal sign distance` (style, no value)

Стиль выбирает расстояние между линиями равным расстоянию между двумя линиями в знаке равенства =.

```

\Huge $\implies$
\tikz[baseline,double equal sign distance]
\draw[double,thick,-implies]
      (0,0.55ex) --++(3ex,0);

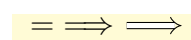
```



```

\normalsize $\implies$
\tikz[baseline,double equal sign distance]
\draw[double,-implies] (0,0.6ex) --++(3ex,0);

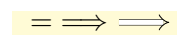
```



```

\tiny $\implies$
\tikz[baseline,double equal sign distance]
\draw[double,very thin,-implies]
      (0,0.5ex) -- ++(3ex,0);

```



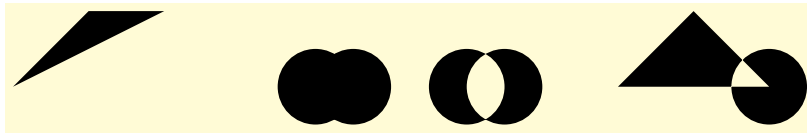
## 12.3 Заполнение пути

`/tikz/fill=<color>` (по умолчанию это цвет в области видимости)

Заполняет путь текущим цветом: сначала все открытые части пути, в случае необходимости, замыкаются, а затем замкнутые области заполняются цветом. Для самопересекающихся путей и для путей, состоящих из нескольких замкнутых областей, определе-

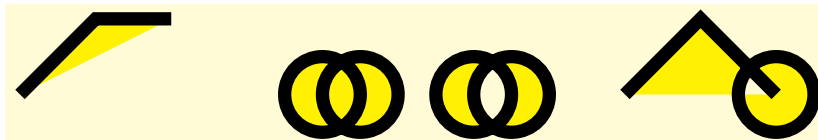
ние замыкания области несколько усложняется, поскольку существуют два различных определения – nonzero winding number rule и even odd rule (о них — ниже).

```
\begin{tikzpicture}
\fill (0,0) -- (1,1) -- (2,1);
\fill (4,0) circle (.5cm) (4.5,0) circle (.5cm);
\fill[even odd rule] (6,0) circle (.5cm) (6.5,0) circle (.5cm);
\fill (8,0) -- (9,1) -- (10,0) circle (.5cm);
\end{tikzpicture}
```



Если опция `fill` используется вместе с опцией `draw` (или потому, что обе задаются как опции пути, или потому, что используется команда `\filldraw`), сначала путь заполняется, затем рисуется. Этот порядок особенно важен, если цвета для рисования и заполнения различны. Даже если используется один и тот же цвет, есть различие между командой `\filldraw` и командой `\fill`: область, заполненная командой `\filldraw`, из-за толщины пера будет немного больше.

```
\begin{tikzpicture}[fill=yellow,line width=5pt]
\filldraw (0,0) -- (1,1) -- (2,1);
\filldraw (4,0) circle (.5cm) (4.5,0) circle (.5cm);
\filldraw[even odd rule] (6,0) circle (.5cm) (6.5,0) circle (.5cm);
\filldraw (8,0) -- (9,1) -- (10,0) circle (.5cm);
\end{tikzpicture}
```



### 12.3.1 Заполнение пути шаблоном

Вместо заполнения пути одним цветом, можно заполнить путь мозаичным шаблоном. Представьте себе небольшую плитку, содержащую простое изображение, например, звезду. Этой плиткой, повторяя ее во всех направлениях, можно заполнить путь.

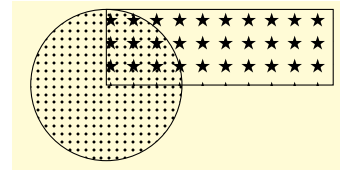
Мозаичные шаблоны существуют в двух вариантах: только окрашенные шаблоны и шаблоны, содержащие только форму. Окрашенный шаблон, скажем, красная звезда с черной границей, будет всегда выглядеть одинаково. Шаблон, содержащий только форму, может иметь разный цвет каждый раз, когда используется, а вот форма шаблона останется той же. Шаблоны, содержащие только форму, не имеют собственных цветов и всегда используют текущий шаблон цвета. Шаблоны не столь гибки, как хотелось бы, в частности, нельзя изменить размер или ориентацию шаблона, не создавая новый.

`/tikz/pattern=<name>` (по умолчанию цвет из области видимости)

Застилает путь шаблоном. Если задается имя `<name>`, используется шаблон с таким именем, иначе используется шаблон, установленный в области видимости. Шаблон работает как цвет заполнения. Нет встроенных именованных шаблонов, допустимых

по умолчанию. Чтобы установить предустановленные шаблоны, следует загрузить библиотеку шаблонов `patterns` (см. [1, глава 41]).

```
\begin{tikzpicture}
\draw[pattern=dots] (0,0) circle (1cm);
\draw[pattern=fivepointed stars]
(0,0) rectangle (3,1);
\end{tikzpicture}
```

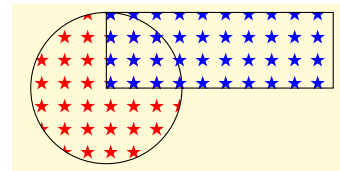


`/tikz/pattern color=<color>`

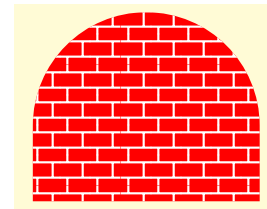
(no default)

Устанавливает цвет, который будет использоваться для шаблонов, содержащих форму. Опция не оказывает влияния на окрашенные шаблоны.

```
\begin{tikzpicture}
\draw[pattern color=red,
pattern=fivepointed stars]
(0,0) circle (1cm);
\draw[pattern color=blue,
pattern=fivepointed stars]
(0,0) rectangle (3,1);
\end{tikzpicture}
```



```
\begin{tikzpicture}
\def\mypath{(0,0) -- +(0,1)
arc (180:0:1.5cm) -- +(0,-1)}
\fill [red] \mypath;
\pattern[pattern color=white,pattern=bricks]
\mypath;
\end{tikzpicture}
```



### 12.3.2 Правила для внутренней точки

Следующие две опции используются для того, чтобы указать, как должны определяться внутренние точки пути.

`/tikz/nonzero rule`

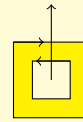
(no value)

Если используется это правило (которое является значением по умолчанию), чтобы определить, является ли заданная точка внутренней точкой пути, применяется следующий метод: из точки выстреливается луч в некотором направлении до бесконечности (направление выбирается так, что никаких странных приграничных ситуаций не происходит). Тогда луч может пересечь путь. Всякий раз, когда он пересекает путь, увеличивается или уменьшается счетчик, который является первоначально нулевым. Если луч пересекает путь при прохождении пути слева направо (относительно луча), счетчик увеличивается, иначе он уменьшается. В конце проверяется, будет ли счетчик отличным от нуля (отсюда название правила). Если так, точку считают внутренней, иначе внешней.

```

\begin{tikzpicture}
\filldraw[fill=yellow]
% Прямоугольник по часовой стрелке
(0,0) -- (0,1) -- (1,1) -- (1,0) -- cycle
% Прямоугольник против часовой стрелки
(0.25,0.25) -- (0.75,0.25) -- (0.75,0.75) -- (0.25,0.75) -- cycle;
\draw[->] (0,1) -- (.4,1);
draw[->] (0.75,0.75) -- (0.3,.75);
\draw[->] (0.5,0.5) -- +(0,1) node[above] {пересечений:  $-1+1 = 0$ };
\end{tikzpicture}

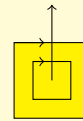
```

пересечений:  $-1 + 1 = 0$ 

```

\begin{tikzpicture}[yshift=-3cm]
\filldraw[fill=yellow]
% Прямоугольник по часовой стрелке
(0,0) -- (0,1) -- (1,1) -- (1,0) -- cycle
% Прямоугольник по часовой стрелке
(0.25,0.25) -- (0.25,0.75) -- (0.75,0.75) -- (0.75,0.25) -- cycle;
\draw[->] (0,1) -- (.4,1);
\draw[->] (0.25,0.75) -- (0.4,.75);
\draw[->] (0.5,0.5) -- +(0,1) node[above] {пересечений:  $1+1 = 2$ };
\end{tikzpicture}

```

пересечений:  $1 + 1 = 2$ 

### `/tikz/even odd rule`

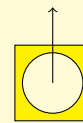
(no value)

Дает другой метод определения того, является ли точка пути внутренней или внешней. Этот метод менее гибкий, но обладает большей наглядностью. В нем также выстреливаются лучи из исследуемой точки. Однако на сей раз считается, как часто луч пересекается путь и точка объявляется внутренней, только если это число нечетное.

```

\begin{tikzpicture}
\filldraw[fill=examplefill,even odd rule]
(0,0) rectangle (1,1)
(0.5,0.5) circle (0.4cm);
\draw[->] (0.5,0.5) -- +(0,1) [above]
node{пересечений:  $1+1 = 2$ };
\end{tikzpicture}

```

пересечений:  $1 + 1 = 2$ 

## 12.4 Заполнение пути произвольным рисунком

Иногда нужно заполнить путь чем-то более сложным, чем шаблон или цвет. Например, можно использовать для заполнения пути сложный рисунок. Следующая опция позволяет это сделать достаточно просто:

`/tikz/path picture=<code>`

(no default)

Когда опция задается в пути и когда `<code>` не пуст, происходит следующее: после того, как все другие операции заполнения пути, вызванные опциями `fill`, `pattern` и `shade`, сделаны, открывается локальная область видимости и путь временно устанавливается



ливается, как путь отсечения; затем выполняется `<code>`, который теперь может нарисовать нечто; и, наконец, локальная область видимости закрывается и, возможно, путь прорисовывается, если задавалась опция `draw`. Опция `path picture` может задаваться только в пути, заданная вне пути она не работает.

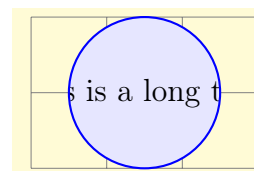
Параметр `<code>` может содержать любой нормальный TikZ-код, такой, например, как `\draw ...` или `\node ...`. Как всегда, когда включается внешняя графика, ее нужно поместить в узел `\node`.

Отметим, что нет действий, позволяющих преобразовать начало координат, то есть точка  $(0,0)$  всегда остается там, где была установлена, когда создавался путь. Это всегда левый нижний угол пути. Однако, можно определить размер пути, для чего нужно использовать следующий специальный предопределенный узел

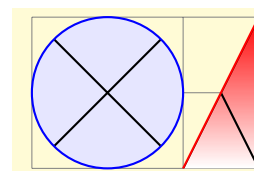
### `path picture bounding box`

Этот узел имеет прямоугольную форму. Его размер и позиция такие же, как и у ограничивающего прямоугольника текущего пути (узла `current path bounding box`, см. [1, раздел 75.4]), вычисленные непосредственно перед тем, как начал выполняться `<code>`. Параметр `<code>` может создать собственные пути, таким образом, обращение к узлу `current path bounding box` в `<code>` возвратит ограничивающий прямоугольник того пути, который в настоящее время создается в `<code>`.

```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (3,2);
\filldraw [fill=blue!10,draw=blue,thick]
(1.5,1) circle (1)
[path picture={
  \node at (path picture bounding box.center) {This is a long text.};}];
\end{tikzpicture}
```



```
\begin{tikzpicture}[cross/.style={path picture={
\draw[black]
(path picture bounding box.south east) --
(path picture bounding box.north west)
(path picture bounding box.south west) --
(path picture bounding box.north east); }}]
\draw [help lines] (0,0) grid (3,2);
\filldraw [cross,fill=blue!10,draw=blue,thick] (1,1) circle (1);
\path [cross,top color=red,draw=red,thick] (2,0) -- (3,2) -- (3,0);
\end{tikzpicture}
```



```
\begin{tikzpicture}[path image/.style={
path picture={
  \node at (path picture bounding box.center)
  {\includegraphics[height=3cm]{#1}};}]
\draw [help lines] (0,0) grid (3,2);
\draw [path image=logotip-sfu0,draw=blue,thick] (0,1) circle (1);
\draw [path image=brave-gnu-world-logo,raw=red,very thick,->]
(1,0) parabola[parabola height=2cm] (3,0);
\end{tikzpicture}
```



## 12.5 Растушевывание пути

Растушевать путь позволяет опция

`/tikz/shade` (no value)

Она по своему действию похожа на заполнение пути цветом, но производит растушевывание, гладко изменяя цвет пути от одного цвета к другому. При этом используется выбранный способ растушевывания (см. ниже). Если эта опция используется вместе с опцией `draw`, то путь сначала выполняется опция `shade`, а затем `draw`. Можно, но лишено смысла, использовать эту опцию вместе с опцией `fill`.

```
\tikz \shade (0,0) circle (1ex);
\tikz \shadedraw (0,0) circle (1ex);
```

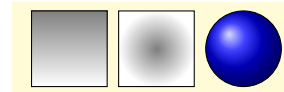


Как, например, растушевать прямоугольник, треугольник, шар? Для этого предусмотрены три способа растушевывания: `ball`, `radial` и `axis`. Растушевывание по умолчанию производит гладкий переход сверху вниз от серого к белому. Однако, возможны и другие варианты, выбор которых позволяет сделать опция `shading=`, автоматически вызывающая опцию `shade`. Отметим, что она не изменяет цвет растушевывания, а меняет только способ варьирования цветов. Для изменения цвета необходимы другие опции, типа `left color=`, `right color=` и т.д.

`/tikz/shading=<name>` (no default)

Устанавливает способ растушевывания с именем `<name>` (`axis`, `radial`, или `ball`).

```
\tikz \shadedraw [shading=axis]
(0,0) rectangle (1,1);
\tikz \shadedraw [shading=radial]
(0,0) rectangle (1,1);
\tikz \shadedraw [shading=ball]
(0,0) circle (.5cm);
```



`/tikz/shading angle=<degrees>` (no default, initially 0)

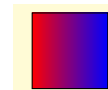
Поворачивает направление растушевывания (не путь!) на заданный угол.

```
\tikz \shadedraw [shading=axis]
(0,0) rectangle (1,1);
\tikz \shadedraw [shading=axis,shading angle=90]
(0,0) rectangle (1,1);
\tikz \shadedraw [shading=axis,shading angle=45]
(0,0) rectangle (1,1);
```



Подробно полный список опций для растушевывания и установки цветов см. в [1, глава 46]. Здесь же ограничимся примером.

```
\tikz \shadedraw [left color=red,right color=blue]
(0,0) rectangle (1,1);
```



Можно определять и новые способы растушевывания (см. [1, 83.3]).

## 12.6 Установка ограничивающего прямоугольника

Система pgf достаточно «разумна», чтобы отслеживать размеры графики и резервировать под нее требуемое пространство. Однако, в некоторых случаях, бывает нужно указать, что рассчитывать выделяемое пространство не следует или графика требует немного больше пространства на странице. Именно для этого можно использовать опцию `use as bounding box` или команду `\useasboundingbox`, которая является только сокращением для `\path [use as bounding box]`.

`/tikz/use as bounding box` (no value)

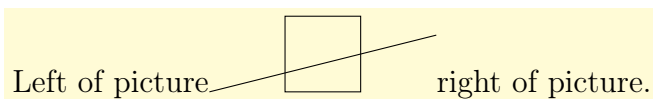
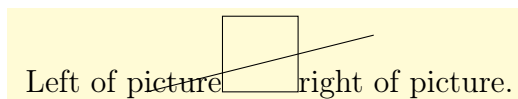
Обычно, когда данная опция задается в пути, используется уже существующий в пути ограничивающий прямоугольник, чтобы определить размеры изображения, а размеры всех подпутей игнорируются. Однако, если предыдущие операции пути уже установили бóльший ограничивающий прямоугольник, он не будет уменьшен новой операцией (обратите внимание на команду `\pgfresetboundingbox`, которая сбрасывает предыдущий ограничивающий прямоугольник, см. [1, 71.13]).

В некотором смысле, `use as bounding box` дает тот же эффект, что и отсечение всего последующего рисунка по отношению к текущему пути, но без реального отсечения.

В первом примере применение этой опции в `{tikzpicture}` приводит к перекрытию основного текста:

```
Left of picture\begin{tikzpicture}
  \draw[use as bounding box]
    (2,0) rectangle (3,1);
  \draw (1,0) -- (4,.75);
\end{tikzpicture}right of picture.
```

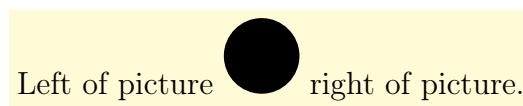
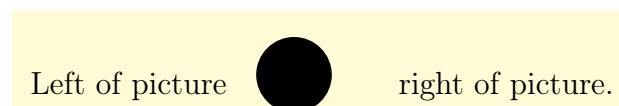
```
Left of picture\begin{tikzpicture}
  \draw (2,0) rectangle (3,1);
  \draw (1,0) -- (4,.75);
\end{tikzpicture}right of picture.
```



Второе применение этой опции позволяет получить больший контроль над пустым пространством вокруг рисунка:

```
Left of picture \begin{tikzpicture}
  \useasboundingbox
  (0,0) rectangle (2,1);
  \fill (.75,.25) circle (.5cm);
\end{tikzpicture}right of picture.
```

```
Left of picture\begin{tikzpicture}
  (0,0) rectangle (2,1);
  \fill (.75,.25) circle (.5cm);
\end{tikzpicture}right of picture.
```



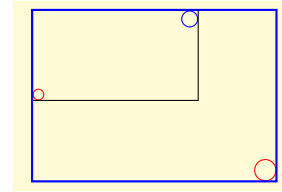
**Замечание:** если эта опция используется в пути, расположенном внутри TeX-группы (области видимости), ее действие завершается в конце этой области видимости.

Есть узел, который позволяет получить размер текущего ограничивающего прямоугольника — это узел `current bounding box`, имеющий форму прямоугольника с размерами, всегда совпадающими с размерами текущего *ограничивающего прямоугольника*. Точно так же, узел `current path bounding box` имеет форму прямоугольника, и его размеры совпадают с размерами *ограничивающего прямоугольника текущего пути*.

```

\begin{tikzpicture}
\draw[red] (0,0) circle (2pt);
\draw[blue] (2,1) circle (3pt);
\draw      (current bounding box.south west)
           rectangle (current bounding box.north east);
\draw[red] (3,-1) circle (4pt);
\draw[thick,blue]
           (current bounding box.south west) rectangle
           (current bounding box.north east);
\end{tikzpicture}

```



Иногда, требуется выровнять несколько окружений `{tikzpicture}` по горизонтали и/или по вертикали в некоторой предписанной позиции. Вертикальное выравнивание можно реализовать посредством опции `baseline`, так как  $\TeX$  поддерживает понятие глубины поля. Горизонтальное выравнивание реализовать немного сложнее. Следующий подход основан на отрицательном горизонтальном промежутке, реализуемом командой `\hspace` до и/или после рисунка, таким образом, удаляя части изображения. Однако, фактическая величина отрицательного горизонтального промежутка обеспечивается посредством координат изображения, используя ключи `trim left` и `trim right`.

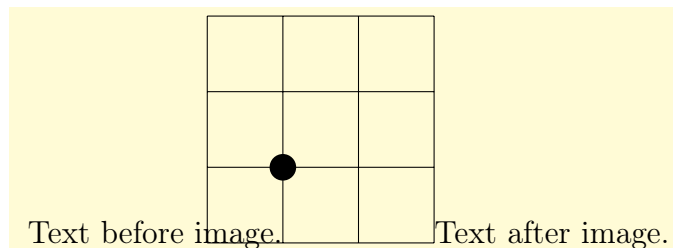
`/tikz/trim left=<dimension or coordinate or default>` (default 0pt)

Ключ `trim left` приказывает `pgf` отбросить все, что остается левее позиции, указанной `<dimension or coordinate>`. Здесь, `<dimension>` —  $x$ -координата изображения, а `<coordinate>` — точка с  $x$ - и  $y$ -координатами (но используется только ее  $x$ -координата). Это дает то же результат, что и использование `\hspace{-s}`, где  $s$  — разность между  $x$ -координатой нижнего левого угла ограничивающего прямоугольника рисунка и  $x$ -координатой, определенной как `<dimension>` или `<coordinate>`. Поскольку `trim left` использует по умолчанию `trim left= 0pt`, все слева от  $x = 0$  будет удалено из ограничивающего прямоугольника. Используя `trim left=default`, можно сбросить значение данного ключа.

```

Text before image.%
\begin{tikzpicture}
[trim left]
\draw (-1,-1) grid (2,2);
\fill (0,0) circle (5pt);
\end{tikzpicture}%
Text after image.

```

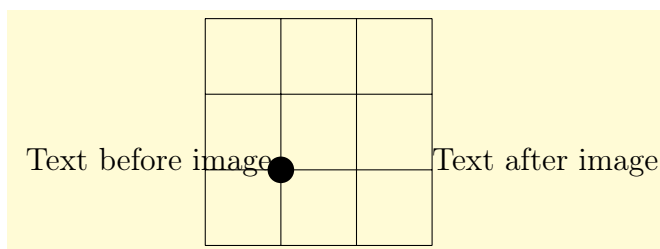


`/tikz/trim right=<dimension or coordinate or default>` (no default)

Подобен ключу `trim left`, но позволяет отбросить все, что лежит правее `<dimension or coordinate>`. Смысл параметров остается тем же. Используя выражение `trim right=default`, можно сбросить значение ключа.

Используем тот же пример, что и ранее, но добавим `trim right`:

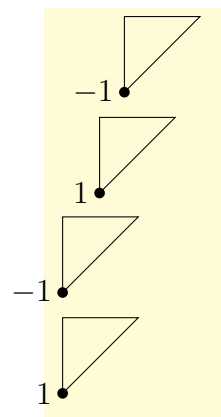
```
Text before image.%
\begin{tikzpicture}
  [trim left,trim right=2cm,
   baseline]
  \draw (-1,-1) grid (2,2);
  \fill (0,0) circle (5pt);
\end{tikzpicture}%
Text after image.
```



В дополнение к `trim left=0pt`, отбрасываем все, что лежит правее  $x = 2cm$ , а ключ `baseline` поддерживает вертикальное выравнивание (по базовой линии  $y = 0cm$ ).

В примере ниже есть достаточно длинная метка `-1` и более короткая метка `1`. Горизонтальное выравнивание относительно начальной позиции устанавливается в последних двух рисунках с помощью опции `trim left`:

```
\begin{tikzpicture}
  \draw (0,1) -- (0,0) -- (1,1) -- cycle;
  \fill (0,0) circle (2pt);
  \node[left] at (0,0) {$-1$};
\end{tikzpicture} \par
\begin{tikzpicture}
  \draw (0,1) -- (0,0) -- (1,1) -- cycle;
  \fill (0,0) circle (2pt);
  \node[left] at (0,0) {$1$};
\end{tikzpicture} \par
\begin{tikzpicture}[trim left]
  \draw (0,1) -- (0,0) -- (1,1) -- cycle;
  \fill (0,0) circle (2pt);
  \node[left] at (0,0) {$-1$};
\end{tikzpicture} \par
\begin{tikzpicture}[trim left]
  \draw (0,1) -- (0,0) -- (1,1) -- cycle;
  \fill (0,0) circle (2pt);
  \node[left] at (0,0) {$1$};
\end{tikzpicture}
```



## 12.7 Отсечение и мягкое обесцвечивание

Отсечение пути означает, что все нарисованное на странице ограничивается определенной областью. Это область не обязана быть прямоугольной, хотелось бы, определяя такую область, использовать произвольный путь. Опция `clip` позволяет определить область, которая должна использоваться для отсечения.

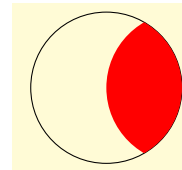
Мягкое обесцвечивание или фединг (`fading`), подобно отсечению, но при постепенном изменении части изображения производится только «отсечение наполовину»: недавно нарисованные пиксели становятся частично прозрачными. Определение и обработка таких постепенных изменений — сложный процесс, он детально описан в разделе 17.

`/tikz/clip`

(no value)

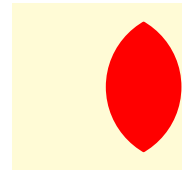
Отсекает последующую прорисовку текущего пути и размеры последующих путей уже не важны для размера картинки. Если отсечение производится самопересекающимся путем, чтобы определить, является ли точка внутренней или внешней по отношению к области отсечения, используется или правило `even-odd` или правило `nonzero winding number`. Путь отсечения — графический параметр и он сбрасывается в конце текущей области видимости. Множество отсечений накапливаются, то есть, отсечение производится по пересечению всех областей отсечения, которые были определены в текущей области видимости. Единственный способ увеличивать область отсечения — закрыть область видимости `scope`.

```
\begin{tikzpicture}
\draw[clip] (0,0) circle (1cm);
\fill[red] (1,0) circle (1cm);
\end{tikzpicture}
```



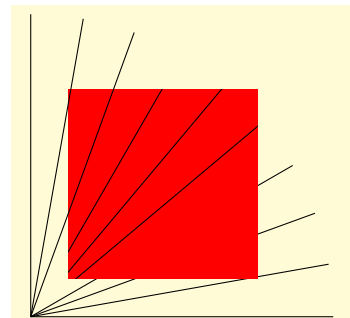
Обычно, следует применять опцию `clip` только в первой команде `\path` из области видимости. Если нужно только отсечь, и не нужно еще что-то рисовать, можно использовать команду `\clip`, которая является сокращением для `\path[clip]`.

```
\begin{tikzpicture}
\clip (0,0) circle (1cm);
\fill[red] (1,0) circle (1cm);
\end{tikzpicture}
```



Чтобы применять отсечение локально, следует использовать окружение `scope`:

```
\begin{tikzpicture}
\draw (0,0) -- (0:4cm); \draw (0,0) -- (10:4cm);
\draw (0,0) -- (20:4cm); \draw (0,0) -- (30:4cm);
\begin{scope}[fill=red]
\fill[clip] (0.5,0.5) rectangle (3,3);
\draw (0,0) -- (40:4cm); \draw (0,0) -- (50:4cm);
\draw (0,0) -- (60:4cm);
\end{scope}
\draw (0,0) -- (70:4cm); \draw (0,0) -- (80:4cm);
\draw (0,0) -- (90:4cm);
\end{tikzpicture}
```



**Предупреждение.** В командах для отсечения, нельзя определять некоторые графические опции. Например, в коде предыдущего примера нельзя переместить опцию `[fill=red]` в команду `\fill`. Причины этого имеют отношение к внутренней организации pdf. Вряд ли стоит сейчас разбираться в деталях, важно просто запомнить, что не следует определять опции в командах для отсечения.

## 12.8 Выполнение нескольких операций на пути

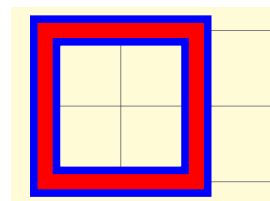
Если на пути нужно выполнить несколько основных операций, например, нарисовать, отсечь и заполнить, они автоматически выполняются в разумном порядке: сначала, путь заполняется, затем прорисовывается, и, наконец, отсекается. Однако, иногда нужно управлять порядком операций. В таких случаях можно использовать следующие две опции.

`/tikz/preactions=<options>` (no default)

Эту опцию можно задавать в команде `\path` (или производной команде, подобной `\draw`, которая вызывает `\path`) или непосредственно, или как часть опций узла `node`. Как опция окружения `scope` она ничего не делает. Если опция используется в `\path`, происходит следующее: когда путь полностью создан и должен использоваться, создается область видимости, внутри которой путь используется не с оригинальными опциями пути, а с `<options>`. Затем путь используется обычным образом. Другими словами, путь используется дважды: первый раз с `<options>`, второй раз с опциями самого пути.

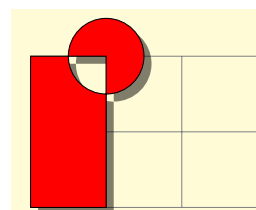
Приведем пример, в котором путь состоит из прямоугольника. Основное действие — нарисовать путь красным цветом. Однако, сначала опция `preaction` заставляет нарисовать путь синим цветом, и потому мы видим синий прямоугольник позади красного.

```
\begin{tikzpicture}
\draw[help lines] (0,0) grid (3,2);
\draw
  [preaction={draw,line width=4mm,blue}]
  [line width=2mm,red]
  (0,0) rectangle (2,2);
\end{tikzpicture}
```



Отметим, что, когда опция `preaction` выполняется, путь уже создан. В частности, применение координатных преобразований к пути уже не даст результата. Для сравнения, применение преобразования холста даст соответствующий преобразованию результат. Например, чтобы добавить к пути тень, используем опцию `preaction` для заполнения пути серым цветом, сдвинутым немного вправо и вниз:

```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (3,2);
\draw [preaction={fill=black,opacity=.5,
  transform canvas={xshift=1mm,yshift=-1mm}}]
  [fill=red] (0,0) rectangle (1,2)
  (1,2) circle (5mm);
\end{tikzpicture}
```



Обычно, следует создать стиль `shadow`, который содержит приведенный выше код. Библиотека `shadows` (см. [1, глава 47]) содержит predefined тени такого вида.

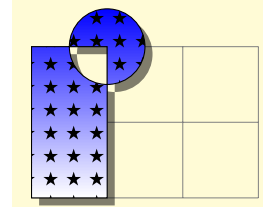
Опцию `preaction` можно использовать многократно. В этом случае, для каждой опции `preaction` путь используется снова (`<options>` не накапливаются для использования в одном пути). Путь используется в порядке заданных опций `preaction`.

В примере ниже, одна опция `preaction` используется, чтобы добавить тень, другая, чтобы обеспечить растушевывание, а в основном действии — использовать шаблон.

```

\begin{tikzpicture}
\draw[help lines] (0,0) grid (3,2);
\draw [pattern=fivepointed stars]
  [preaction={fill=black,opacity=.5,
    transform canvas={xshift=1mm,yshift=-1mm}}]
  [preaction={top color=blue,bottom color=white}]
  (0,0) rectangle (1,2) (1,2) circle (5mm);
\end{tikzpicture}

```

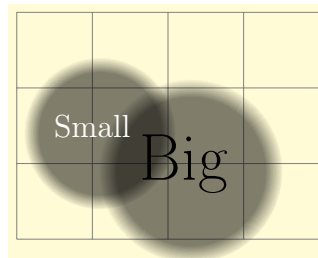


Сложный код создается в следующем примере, где путь используется несколько раз с разными обесцвечиванием (fading) и растушевыванием (shading).

```

\begin{tikzpicture}
[% Создать два интересных стиля для кнопок
  button/.style=
  { % Первая операция preaction: размытая тень
    preaction={fill=black,path fading=circle with fuzzy edge 20 percent,
      opacity=.5,transform canvas={xshift=1mm,yshift=-1mm}},
    % Вторая операция preaction: фоновый шаблон (узор)
    preaction={pattern=#1,path fading=circle with fuzzy edge 15 percent},
    % Третья операция preaction: создать наполненный светом фон
    preaction={top color=white, bottom color=black!50, shading angle=45,
      path fading=circle with fuzzy edge 15 percent,opacity=0.2},
    % Четвертая операция preaction: создать специальное освещение границ
    preaction={path fading=fuzzy ring 15 percent,top color=black!5,
      bottom color=black!80, shading angle=45},
    inner sep=2ex }, % конец определения первого стиля
  button/.default=horizontal lines light blue, circle ]
\draw [help lines] (0,0) grid (4,3);
\node [button] at (2.2,1) {\Huge Big};
\node [button=crosshatch dots light steel blue,text=white]
  at (1,1.5) {Small};
\end{tikzpicture}

```



`/tikz/postaction=<options>` (no default)

Аналогична опции `preaction`, только опции применяются после выполнения основного действия. Можно задавать несколько таких опций, в этом случае путь будет использоваться несколько раз, каждый раз с другим набором опций.

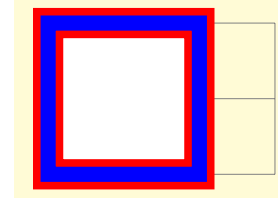
В примере ниже используется опция `postaction`, чтобы нарисовать путь после того, как он уже был нарисован:



```

\begin{tikzpicture}
\draw[help lines] (0,0) grid (3,2);
\draw
  [postaction={draw,line width=2mm,color=blue}]
  [line width=4mm,red,fill=white]
  (0,0) rectangle (2,2);
\end{tikzpicture}

```

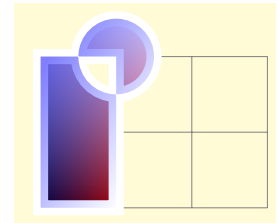


В следующем примере опция `postaction` используется для раскраски пути:

```

\begin{tikzpicture}
\draw[help lines] (0,0) grid (3,2);
\draw [postaction={path fading=south,fill=white}]
  [postaction={path fading=south,
    fading angle=45,fill=blue,opacity=.5}]
  [left color=black,right color=red,
    draw=white,line width=2mm]
  (0,0) rectangle (1,2) (1,2) circle (5mm);
\end{tikzpicture}

```



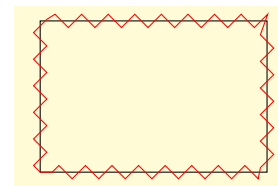
## 12.9 Декорирование и трансформация пути

Прежде, чем путь использовать, его можно сначала декорировать и/или трансформировать. Трансформация означает, что путь заменяется другим путем, который получается из исходного небольшим его изменением. Такие трансформации — частный случай общего декорирования, описанного подробно в разделе 18. Например, в следующем примере путь рисуется дважды: один раз обычным, а затем трансформированным (декорированным).

```

\begin{tikzpicture}
\draw (0,0) rectangle (3,2);
\draw [red, decorate, decoration=zigzag]
  (0,0) rectangle (3,2);
\end{tikzpicture}

```



Естественно, можно объединить эти операции в одну команду, использующую пред- или пост- действие. Можно также деформировать формы.

```

\begin{tikzpicture}
\node [circular drop shadow={shadow scale=1.05},
  minimum size=3.13cm,decorate,draw,thick,
  decoration=zigzag,fill=blue!20,circle]
  {Hello!};
\end{tikzpicture}

```



# Глава 13

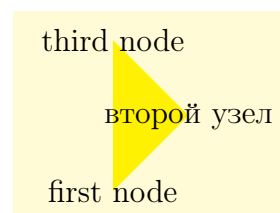
## Узлы и дуги

### 13.1 Узлы и их формы

Узел — это обычно либо прямоугольник, либо круг, либо другая простая форма с некоторым текстом на ней. Узлы создаются всякий раз, когда TikZ сталкивается с `node` или `coordinate` в точке пути, где можно было бы ожидать появления нормальной операции пути. Узлы не являются частью пути, они добавляются к рисунку после того, как путь нарисован. Если есть несколько узлов, они рисуются в том порядке, в котором помещались в путь.

В самом простом случае, узел — только некоторый текст, помечающий узел на рисунке. В коде текст располагается в фигурных скобках за `node`, обычно содержащим многочисленные опции, применяемые только к узлу, и на рисунке помещается в текущую точку пути. Однако, узел может иметь вокруг себя границу или более сложный фон и передний план. Некоторые узлы вообще не имеют текста, и состоят исключительно из фона. Узлы можно именовать, указывая имя в круглых скобках, чтобы ссылаться на их координаты позже в том же рисунке или, если предпринять определенные меры (см. раздел 13.12), в других рисунках.

```
\tikz \fill[fill=yellow] (0,0) node {first node}
      -- (1,1) node {second node}
      -- (0,2) node {third node};
```



Синтаксис определения узла следующий:

```
\path ... node [<options>](<name>)at(<coordinate>){<text>} ...;
```

Код помещает узел в указанную позицию `<coordinate>`, заданную после `at`, а не, как было бы в обычном случае, в последнюю позицию. Синтаксис `at` не доступен, когда узел задается внутри операции пути (там это не имеет смысла). Имя `<name>` используется позже для ссылок, но задавать его не обязательно. Чтобы задать имя, не используя круглые скобки, можно использовать в списке `<options>` опцию `name=<name>`.

```
/tikz/name=<node name> (no default)
```

Назначает имя узлу. Так как это имя «высшего уровня» (драйверы никогда не знают его), можно использовать в имени пробелы, числа, английские буквы (нельзя использовать русский алфавит!). Таким образом, можно назвать узел именем `1`, или `start of chart`, или `y_1`. Имя узла не должно содержать знаков пунктуации таких,

как точка, запятая или двоеточие, так как они используются для указания на то, какой тип координат подразумевается при ссылке на узел.

`/tikz/alias=<another node name>` (no default)

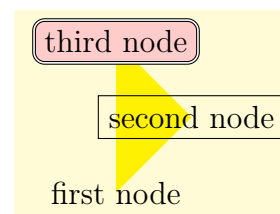
Определяет для узла другое имя (псевдоним). Опцию можно использовать многократно, что позволяет обращаться к узлу посредством нескольких псевдонимов. Используя опции `late options`, можно назначить псевдоним узлу и позже.

`/tikz/at=<coordinate>` (no default)

Это другой способ определить `at`-координату. Заметим, что, обычно, следует заключить `<coordinate>` в фигурные скобки, чтобы запятая в `<coordinate>` не запутала `TeX`.

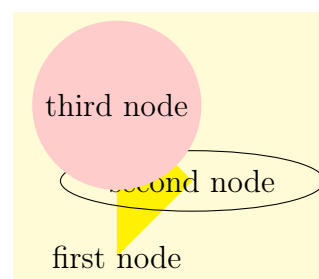
Необязательный список опций `<options>` применяется только к узлу и не оказывает влияние вне узла. А вот большинство внешних опций применимы и к узлу, но не все. Например, внешнее вращение не применимо к узлам (если не используются некоторые специальные опции). Далее, внешние операции пути, такие как `draw` или `fill`, никогда не относятся к узлам и должны быть заданы в узле (если не используются некоторые другие особые опции). Как сказано ранее, в узел можно добавить границу и фон:

```
\tikz \fill[fill=yellow]
      (0,0) node {first node}
      -- (1,1) node [draw] {second node}
      -- (0,2) node [fill=red!20,draw,double,
                    rounded corners]{third node};
```



Граница является специальным случаем более общего механизма. Каждый узел имеет определенную форму (по умолчанию, прямоугольник). Однако, можно использовать форму круга или эллипса (чтобы использовать форму `ellipse` и другие геометрические формы, нужно загрузить библиотеку `shapes.geometric`) (создание новых форм дело хитрое и требует основного уровня `pgf`).

```
\tikz \fill[fill=yellow]
      (0,0) node{first node}
      -- (1,1) node[shape=ellipse,draw]
                  {second node}
      -- (0,2) node[shape=circle,fill=red!20]
                  {third node};
```



Чтобы указать форму узла, используется следующая опция:

`/tikz/shape=<shape name>` (no default, initially rectangle)

Определяет форму или текущего узла, или, когда опция задается не в узле, а где-то вне узла, форму всех узлов в текущей области видимости. Так как опция используется часто, можно опускать слово `shape=`. Когда `TikZ` сталкивается с опцией, которой он не знает, то, после всех проверок, он проверит, не является ли эта опция именем некоторой формы. Если так, то эта форма будет выбрана, как будто было сказано `shape=<shape name>`.

По умолчанию, доступны следующие формы: `rectangle`, `circle`, `coordinate`, а, когда загружена библиотека `shapes.geometric`, также `ellipse` и другие (см. [1, глава 48], но в будущем могут появиться и новые формы). Детали некоторых из этих форм, их якоря и опции определения размеров, обсуждаются в разделе 13.1.1.

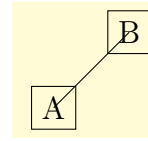
На представление узлов влияют следующие стили:

`/tikz/every node`

(style, initially empty)

Этот стиль устанавливается в начале каждого узла.

```
\begin{tikzpicture}
  [every node/.style={draw}]
  \draw (0,0) node {A} -- (1,1) node {B};
\end{tikzpicture}
```

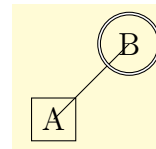


`/tikz/every <shape> node`

(style, initially empty)

Стиль устанавливается в начале каждого узла заданной формы <shape>.

```
\begin{tikzpicture}
  [every rectangle node/.style={draw},
   every circle node/.style={draw,double}]
  \draw (0,0) node[rectangle] {A}
    -- (1,1) node[shape=circle] {B};
\end{tikzpicture}
```



Есть специальный синтаксис для того, чтобы было легче определять узлы:

`\path ... coordinate [<options>](<name>)at(<coordinate>) ...;`

То же, что и `node [shape=coordinate] [<options>] (<name>)at(<coordinate>){}`, где часть `at` может быть опущена.

Так как узлы — часто единственная операция в пути, есть две специальных команды для создания путей, содержащих только узел:

`\node`

Внутри `{tikzpicture}` сокращение для `\path node`.

`\coordinate`

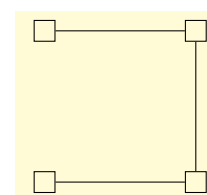
Внутри `{tikzpicture}` сокращение для `\path coordinate`.

### 13.1.1 Предопределенные формы

Кроме указанных выше трех форм по умолчанию и одной дополнительной, `pgf` и `tikz` определяют огромное число других форм, которые становятся доступными после загрузки соответствующих библиотек (см. [1, глава 48]).

Форма `coordinate` обрабатывается специальным образом. Когда узел  $x$ , форма которого `coordinate`, используется как координата  $(x)$ , это то же самое, что `(x.center)`.

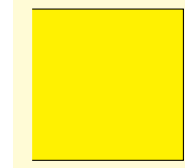
```
\begin{tikzpicture}[every node/.style={draw}]
  \path[yshift=1.5cm,rectangle]
    (0,0) node(a1){} (2,0) node(a2){}
    (2,2) node(a3){} (0,2) node(a4){};
  \filldraw[fill=yellow] (a1) -- (a2) -- (a3) -- (a4);
\end{tikzpicture}
```



```

\begin{tikzpicture}
\path[shape=coordinate]
  (0,0) coordinate(b1) (2,0) coordinate(b2)
  (2,2) coordinate(b3) (0,2) coordinate(b4);
\filldraw[fill=yellow] (b1) -- (b2) -- (b3) -- (b4);
\end{tikzpicture}

```



### 13.1.2 Общие опции узлов

Точное поведение узлов различно, поскольку зависит от их форм, имеющих разное поведение. Однако есть некоторые опции, относящиеся к большинству форм.

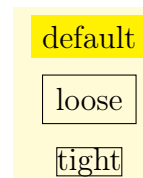
`/pgf/inner sep=<dimension>` (no default, initially .3333em)

Устанавливает дополнительное пространство между текстом и границей формы равным `<dimension>`. Результат тот же, если бы добавлялся соответствующий горизонтальный и вертикальный пробелы в начале и конце текста.

```

\begin{tikzpicture}
\draw (0,0) node[inner sep=0pt,draw] {tight}
  (0cm,2em) node[inner sep=5pt,draw] {loose}
  (0cm,4em) node[fill=yellow] {default};
\end{tikzpicture}

```



`/pgf/inner xsep=<dimension>` (no default, initially .3333em)

Определяет внутреннее дополнительное пространство только в  $x$ -направлении.

`/pgf/inner ysep=<dimension>` (no default, initially .3333em)

Определяет внутреннее дополнительное пространство только в  $y$ -направлении.

`/pgf/outer sep=<dimension>` (no default, initially .5\pgflinewidth)

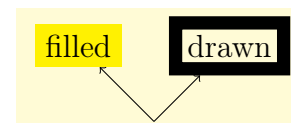
псевдоним: `/tikz/outer sep`

Добавляет дополнительное пространство размером равным `<dimension>` вне фона узла. Главный эффект опции тот, что все якоря узла немного смещаются во вне. Значение по умолчанию — половина ширины строки. Когда используется значение по умолчанию и когда рисуется путь, якоря будут лежать точно на внешней границе пути (не на самом пути!). Когда форма заполняется, но не рисуется, это может оказаться нежелательным. В этом случае нужно установить опцию `outer sep` равной нулю.

```

\begin{tikzpicture}
\draw[line width=5pt]
  (0,0) node[outer sep=0pt,fill=yellow] (f) {filled}
  (2,0) node[inner sep=.5\pgflinewidth+2pt,draw] (d) {drawn};
\draw[->] (1,-1) -- (f); \draw[->] (1,-1) -- (d);
\end{tikzpicture}

```



`/pgf/outer xsep=<dimension>` (no default, initially `.5\pgflinewidth`)

Определяет внешнее дополнительное пространство только в  $x$ -направлении.

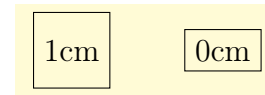
`/pgf/outer ysep=<dimension>` (no default, initially `.5\pgflinewidth`)

Определяет внешнее дополнительное пространство только в  $y$ -направлении.

`/pgf/minimum height=<dimension>` (no default, initially `0pt`)

Гарантируется, что высота формы (включая внутреннее, но игнорируя внешнее дополнительное пространство) будет по крайней мере равна `<dimension>`. Если высота текста плюс дополнительное пространство не больше `<dimension>`, форма увеличится, если высота текста уже больше `<dimension>`, форма не станет меньше.

```
\begin{tikzpicture}
\draw (0,0) node[minimum height=1cm,draw] {1cm}
      (2,0) node[minimum height=0cm,draw] {0cm};
\end{tikzpicture}
```



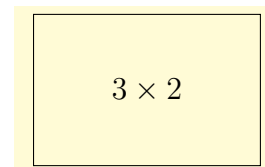
`/pgf/minimum width=<dimension>` (no default, initially `0pt`)

Аналогична опции `minimum height`, но используется для ширины формы.

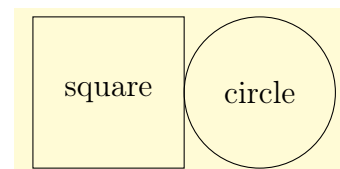
`/pgf/minimum size=<dimension>` (no default)

Устанавливает минимальную высоту и ширину равными `<dimension>`.

```
\begin{tikzpicture}
\begin{tikzpicture}
\draw (0,0) node [minimum height=2cm,
                  minimum width=3cm,draw]
      {$3 \times 2$};
\end{tikzpicture}
\end{tikzpicture}
```



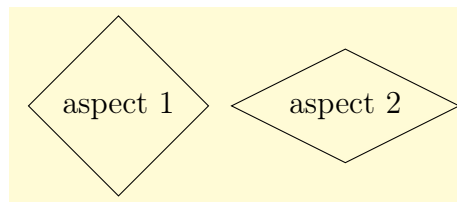
```
\begin{tikzpicture}
\draw (0,0) node[minimum size=2cm,draw]
      {square};
\draw (2,0) node[minimum size=2cm,
                  draw,circle] {circle};
\end{tikzpicture}
```



`/pgf/shape aspect=<aspect ratio>` (no default)

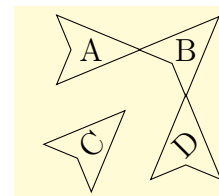
Устанавливает желаемое отношение размеров формы. Для формы `diamond` эта опция устанавливает соотношение между шириной и высотой формы.

```
\begin{tikzpicture}
\draw (0,0) node[shape aspect=1,
diamond,draw] {aspect 1};
\draw (3,0) node[shape aspect=2,
diamond,draw] {aspect 2};
\end{tikzpicture}
```



Некоторые формы (но не все) поддерживают специальный вид вращения, который вращает только границу формы и не зависит от содержимого узла, но только в дополнение к любым другим преобразованиям.

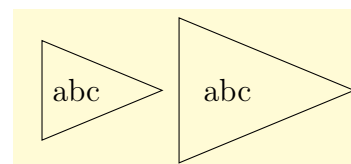
```
\tikzstyle{every node}=[shape=dart,
shape border uses incircle, inner sep=1pt, draw]
\begin{tikzpicture}
\foreach \a/\b/\c in {A/0/0, B/45/0, C/0/45, D/45/45}
\node [shape border rotate=\b, rotate=\c]
at (\b/36,-\c/36) {\a};
\end{tikzpicture}
```



Есть два типа вращения: ограниченное (на некоторые углы) и неограниченное (на любой угол). То, какой тип вращения применяется, определяется тем, как конструируется граница формы. Если граница формы создается, используя вписанную окружность, точнее, окружность, плотно облегающую содержимое узла (включая внутреннее дополнительное пространство), то вращение может быть неограниченным. Если же граница создается, используя естественные размеры содержимого узла, вращение ограничивается углами, кратными  $90^\circ$ .

При использовании вписанной окружности, неограниченное вращение возможно, но граница не будет плотно охватывать содержимое узла. Вот простой пример:

```
\tikzstyle{every node}=[
shape=isosceles triangle, draw]
\begin{tikzpicture}
\node {abc};
\node[shape border uses incircle] at (2,0){abc}; % см. ниже
\end{tikzpicture}
```



Есть pgf-ключи, определяющие то, как создается граница формы и происходит ее вращение. Но не все формы поддерживают эти ключи, и нужно обратиться за информацией к документации по конкретной форме.

`/pgf/shape border uses incircle=<boolean>` (default true)

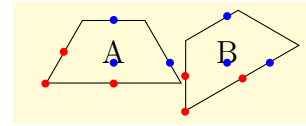
Определяет, что граница формы создается, используя вписанную окружность. Если значению `<boolean>` не задано, по умолчанию используется значение true.

`/pgf/shape border rotate=<angle>` (no default, initially 0)

Вращает границу формы независимо от содержимого узла, но в дополнение к любым другим преобразованиям. Если граница формы создана, не используя вписанную

окружность, то, когда форма рисуется, угол вращения округляется до ближайшего целого, кратного  $90^\circ$ . Если вращается граница формы, компасные якорные точки и якоря текстового поля (`mid east`, `base west`, ...) не вращаются, а другие якоря вращаются.

```
\tikzstyle{every node}=[shape=trapezium,
    draw, shape border uses incircle]
\begin{tikzpicture}
\node [shape border rotate=30] at (1.5,0) (B) {B};
\node at (0,0) (A) {A}; \foreach \s/\t in
    {left side/base east,bottom side/north,bottom left corner/base}{
        \fill[red] (A.\s) circle(1.5pt) (B.\s) circle(1.5pt);
        \fill[blue](A.\t) circle(1.5pt) (B.\t) circle(1.5pt);}
\end{tikzpicture}
```



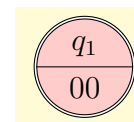
Нежелательный результат вращения границ формы состоит в том, что поддержка форм не различает `outer xsep` и `outer ysep`, и обычно, использует большее число.

## 13.2 Многослойные узлы

Большинство узлов имеют одну простую текстовую метку. Однако, узлы более сложных форм могут состоять из нескольких частей — `<node part>`. Например, в теории автоматов метка состояния состоит из двух независимых текстов: в верхней и в нижней частях круга. Аналогично, форма `uml`-класса имеет часть с именем класса, часть с именами методов и часть с именами атрибутов. Число примеров легко увеличить.

И `pgf`, и `TikZ` поддерживают многослойные узлы. На уровне `TikZ`, различные части узла определяются командой `\nodepart[<options>]{<part name>}`, которая используется только в параметре `<text>` операции `node`. Набор первой части текста начинается после фигурной скобки и заканчивается перед командой `\nodepart`, затем начинается вторая часть текста, которая заканчивается либо командой `\nodepart`, начинающей следующую часть, либо фигурной скобкой, если набор текста завершается.

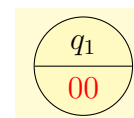
```
\begin{tikzpicture}
\node [shape=circle split,draw,double,fill=red!20]
    {$q_1$ \nodepart{lower} $00$ };
\end{tikzpicture}
```



Следующий стиль используется в начале каждой части узла с именем `<part name>`:

`/tikz/every <part name> node part` (style, initially empty)

```
\tikz [every lower node part/.style={color=red}]
\node [shape=circle split,draw,fill=yellow!30]
    {$q_1$ \nodepart{lower} $00$};
```





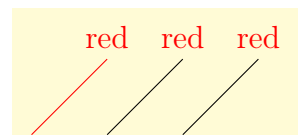
## 13.3 Текст узла

### 13.3.1 Текстовые параметры: цвет и прозрачность

Самая простая опция для текста в узлах — определение его цвета. Обычно, цвет — последний цвет, установленный, используя опцию `color=`, возможно унаследованный от других областей видимости. Однако, можно явно указать нужный цвет для текста в данном узле, используя опцию

`/tikz/text=<color>` (no default)

```
\begin{tikzpicture}
\draw[color=red] (0,0) -- +(1,1) node[above]{red};
\draw[text=red] (1,0) -- +(1,1) node[above]{red};
\draw (2,0) -- +(1,1) node[above,color=red]{red};
\end{tikzpicture}
```



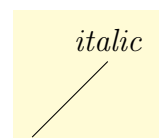
Так же как и цвет, используя опцию `text opacity`, которая детально описана в разделе 17, можно установить степень непрозрачности текста.

### 13.3.2 Текстовые параметры: шрифт

Всегда можно скорректировать шрифт для текста узла, используя опцию

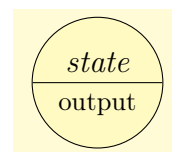
`/tikz/font=<font commands>` (no default)

```
\begin{tikzpicture}
\draw[font=\itshape] (1,0) -- +(1,1)
node[above] {italic};
\end{tikzpicture}
```



Приведем более содержательный пример использования разных шрифтов:

```
\tikz [every text node part/.style={font=\itshape},
every lower node part/.style={font=\footnotesize}]
\node [circle split,draw,fill=yellow!30]
{state \nodepart{lower} output};
```



### 13.3.3 Текстовые параметры: выравнивание и ширина текста

Обычно, когда набирается узел, весь текст узла, задаваемый в фигурных скобках, помещается в одну длинную строку (в `\hbox`, если быть точным), и ширина узла определяется шириной текста. Но, иногда, нужно создать узел, который содержит несколько строк текста. Есть три разных способа добиться этого.

1. В узел можно поместить стандартное окружение, создающее многострочный, выровненный текст. Например, можно использовать внутри узла окружение `{tabular}`:

```
\tikz \node [draw] {
  \begin{tabular}{cc}
    upper left & upper right \\
    lower left & lower right
  \end{tabular}
};
```

upper left	upper right
lower left	lower right

Этот подход часто самый гибкий в том смысле, что позволяет использовать все команды выравнивания, предлагаемые выбранным окружением.

2. В тексте узла можно использовать команду `\`, указывая конец строк, а затем расположить некоторым способом эти строки, задавая опцию `align=`.

```
\tikz[align=left] \node[draw,fill=yellow!30]
  {This is a\ demonstration.};
```

This is a demonstration.
-----------------------------

```
\tikz[align=center] \node[draw,fill=yellow!30]
  {This is a\ demonstration.};
```

This is a demonstration.
-----------------------------

Команду `\` можно использовать с параметром в квадратных скобках.

```
\tikz \node[fill=yellow!30,align=right]
  {This is a \
  demonstration text \[1ex]
  for alignments.};
```

This is a demonstration text for alignments.
--

3. Можно автоматически разбить строку в узле на части, определяя фиксированную ширину текста для узла опцией

`/tikz/text width=<dimension>` (no default)

В этом случае, опция поместит текст в узел заданной ширины (нечто родственное `{minipage}`). Если ширина текста узла будет не больше `<dimension>`, текст будет помещен в поле указанной ширины. Если текст будет шире, то строка будет автоматически разорвана. Разорвать строку можно явно, используя команду `\`. По умолчанию с опцией `text width` используется опция `align=left`. Но, используя `align=` или команду `\centering` можно изменить стиль выравнивания.

```
\tikz \draw (0,0)
  node[fill=yellow!30,text width=4cm]
  {This is a demonstration text for
  showing how line breaking works.};
```

This is a demonstration text for showing how line breaking works.
--

`/tikz/align=<how>` (no default)

Ключ используется для установки характера выравнивания многострочного текста в узле. Параметр `<how>` может принимать следующие значения: `left`, `flushleft`, `right`, `flushright`, `center`, `flushcenter`, `justify`. Существуют некоторые тонкие моменты в выравнивании текста, в зависимости от присутствия других опций узла, таких как `text width` или `node halign header` (см. детали в [1, 16.4.3]).

```
\tikz \node[fill=yellow!30,align=left]
  {This is a\
  demonstration text\
  for alignments.};
```

This is a  
demonstration text  
for alignments.

```
\tikz \node[fill=yellow!30,text width=4cm,
  align=left]
  {This is a demonstration text for
  showing how line breaking works.};
```

This is a de-  
monstration text  
for showing how line  
breaking works.

```
\tikz \node[fill=yellow!30,text width=4cm,
  align=flush left]
  {This is a demonstration text for
  showing how line breaking works.};
```

This is a  
demonstration text  
for showing how line  
breaking works.

```
\tikz \node[fill=yellow!30,text width=4cm,
  align=right]
  {This is a demonstration text for
  showing how line breaking works.};
```

This is a  
demonstration text  
for showing how  
line breaking works.

```
\tikz \node[fill=yellow!30,text width=4cm,
  align=flush right]
  {This is a demonstration text for
  showing how line breaking works.};
```

This is a  
demonstration text  
for showing how line  
breaking works.

```
\tikz \node[fill=yellow!30,align=center]
  {This is a\
  demonstration text\
  for alignments.};
```

This is a  
demonstration text for  
alignments.

```
\tikz \node[fill=yellow!30,text width=4cm,
  align=center]
  {This is a demonstration text for
  showing how line breaking works.};
```

This is a  
demonstration  
text for showing how  
line breaking works.

```
\tikz \node[fill=yellow!30,text width=4cm,
  align=flush center]
  {This is a demonstration text for
  showing how line breaking works.};
```

This is a  
demonstration text  
for showing how line  
breaking works.

```
\tikz \node[fill=yellow!30,text width=4cm,
  align=justify]
  {This is a demonstration text for
  showing how line breaking works.};
```

This is a  
demonstration text  
for showing how line  
breaking works.

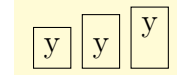
### 13.3.4 Текстовые параметры: высота и глубина текста

В дополнение к изменению ширины узла, можно изменять и высоту узла. Для этого есть два пути: можно использовать опцию `minimum height`, которая гарантирует что высота целого узла будет по крайней мере не меньше заданной высоты, или использовать опцию `/tikz/text height=<dimension>`, которая определяет высоту самого текста. Высота текста обычно не является высотой поля формы: к высоте текста добавляется как дополнительное пространство, так и глубина текста. Кроме специальных ситуаций, следует использовать опцию `minimum size` вместо `minimum height`.

`/tikz/text depth=<dimension>` (no default)

Устанавливает глубину текстовых полей в формах. Эта опция главным образом полезна тогда, когда нужно гарантировать однородную глубину текстовых полей, которые должны быть выровнены.

```
\tikz \node[draw] {y};
\tikz \node[draw,text height=10pt] {y};
\tikz \node[draw,text depth=10pt] {y};
```

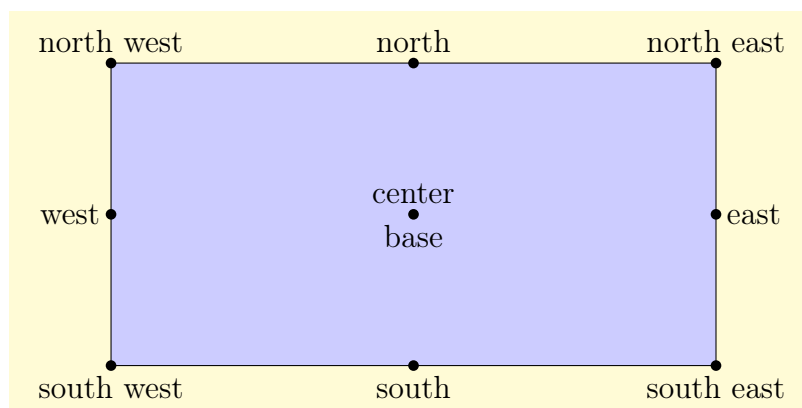


## 13.4 Позиционирование узлов

Когда узел размещается в некоторой точке, по умолчанию он центрируется. Часто такое расположение нежелательно и было бы лучше переместить узел вправо или вверх.

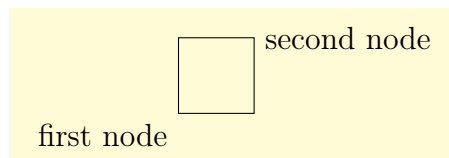
### 13.4.1 Позиционирование узлов по якорям

Pgf, чтобы управлять размещением узлов, использует так называемый механизм якорей, определяя на каждой форме множество якорей позиционирования. Например, верхний правый угол называется не верхним правым якорем, а северо-восточным якорем формы — `north east`. Центр формы — якорь `center`, и так далее. Приведем пример расположения якорей на прямоугольнике.



Теперь, размещая узел в определенной точке, можно переместить узел так, чтобы определенный якорь оказался в указанной точке. В следующем примере, один узел сдвигается так, что его северо-восточный якорь (`north east`) располагается в точке (0,0), а второй так, что его западный якорь (`west`) — в точке (1,1).

```
\tikz \draw (0,0) node[anchor=north east]
      {first node}
      rectangle (1,1) node[anchor=west]
      {second node};
```

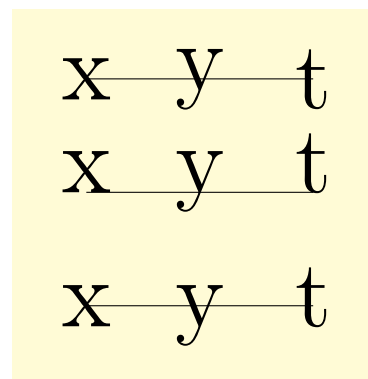


Так как по умолчанию якорь позиционирования — `center`, часто задача состоит в том, чтобы сдвинуть узел так, чтобы его якорь `center` оказался в указанной точке.

`/tikz/anchor=<anchor name>` (no default)

Сдвигает узел так, чтобы его якорь `<anchor name>` оказался в текущей точке. Единственный якорь, который присутствует во всех формах, `center`. Большинство форм имеют якоря во всех компасных направлениях. Кроме того, стандартные формы также определяют якоря `base`, `base west` и `base east`, чтобы можно было проводить размещение на базовой линии текста, и якорь `mid` (а так же `mid west`, `mid east`). Якорь `mid` — половина высоты символа `{x}` выше базовой линии — полезен для вертикального центрирования узлов, которые имеют разную высоту и разную глубину. Например:

```
\begin{tikzpicture}[scale=3,transform shape]
% 1-е выравнивание по центру -> колебания
\draw[anchor=center] (0,1) node{x} -- (0.5,1)
                    node{y} -- (1,1) node{t};
% 2-е выравнивание -> нет колебания, но высоко
\draw[anchor=base] (0,.5) node{x} -- (0.5,.5)
                  node{y} -- (1,.5) node{t};
% 3-е выравнивание по якорю mid
\draw[anchor=mid] (0,0) node{x} -- (0.5,0)
                 node{y} -- (1,0) node{t};
\end{tikzpicture}
```



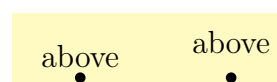
### 13.4.2 Основные опции для размещения узлов

К сожалению, например, то, что для размещения узла выше заданной точки, нужно определить южный якорь `south`, часто противоречит интуиции. Поэтому есть некоторые полезные опции, позволяющие выбирать стандартные якоря более понятно.

`/tikz/above=<offset>` (default 0pt)

Делает то же, что и код `anchor=south`. Если задан параметр `<offset>`, узел дополнительно сдвигается вверх на заданную величину.

```
\tikz \fill (0,0) circle (2pt) node[above] {above};
\tikz \fill (0,0) circle (2pt) node[above=2pt]
      {above};
```



Следующие опции подобны опции `above=<offset>`.

`/tikz/below=<offset>` (default 0pt)

`/tikz/left=<offset>` (default 0pt)

`/tikz/right=<offset>` (default 0pt)

`/tikz/above left`

(no value)

Делает то же, что и код `anchor=south east`. Опция `above left` также может иметь параметр `<offset>`, но здесь использовать параметр можно только с загруженной библиотекой `positioning` (она изменяет смысл этого параметра на нечто более разумное). Отметим, что опция `above left` не то же, что опции `above`, `left`, когда будет действовать только опция `left`.

```
\tikz \fill (0,0) circle (2pt)
      node[above left] {above left};
\tikz \fill (0,0) circle (2pt)
      node[above left=5pt] {above left};
```

above left • above left •

Следующие опции подобны опции `above left`.

`/tikz/above right`

(no value)

`/tikz/below left`

(no value)

`/tikz/below right`

(no value)

Для размещения большого числа узлов простые опции, типа `above` или `left`, не всегда удобны. Для таких ситуаций существуют дополнительные библиотеки, упрощающие позиционирование. Уже упоминавшаяся библиотека `positioning` (см. [1, 13.5.3]) переопределяет стандартные опции размещения узлов, предоставляя больше возможностей по управлению размещением. Библиотека матриц `matrix` (см. [1, глава 38]), определяя дополнительные стили и опции создания матриц, позволяет точно размещать узлы по строкам и столбцам. Библиотека цепочек `chains` (см. [1, глава 28]) определяет опции для создания цепочек — последовательности узлов, упорядоченных по строкам или столбцам и связанных дугами. Цепочки весьма полезны для упрощения размещения узлов в разветвленной сети. Для позиционирования узлов по строкам и столбцам часто используются матрицы, а цепочки используют для описания связей между узлами (см. главу 14).

## 13.5 Установка узлов по набору координат

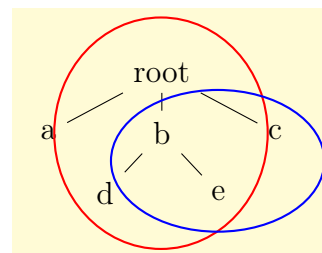
Иногда нужно, чтобы размер и позиция узла не задавались, используя якорь и размерные параметры, а автоматически вычислялись размеры и достаточно большое поле, на котором можно было бы разместить все, что нужно. Эта ситуация обычно возникает тогда, когда рисуется картинка, а потом части картинки нужно выделить или высветить.

В этом случае полезна опция `fit` из библиотеки `fit` (см. [1, глава 34]). Идея состоит в том, что узлу можно задать опцию `fit`, передавая ей список координат (одну за другой без запятых) в качестве параметра. В результате под текстовую область узла будет выделяться столько пространства, чтобы оно содержало все заданные координаты.

```

\begin{tikzpicture}[level distance=8mm]
\node (root) {root}
  child { node (a) {a} }
  child { node (b) {b}
    child { node (d) {d} }
    child { node (e) {e} } }
  child { node (c) {c} };
\node[draw=red,inner sep=0pt,thick,ellipse,fit=(root)(b)(d)(e)]{};
\node[draw=blue,inner sep=0pt,thick,ellipse,fit=(b)(c)(e)]{};
\end{tikzpicture}

```

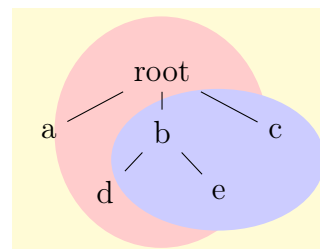


Если нужно заполнить узел цветом, обычно надо разместить его на фоновом уровне (напомним, что для этого нужно загрузить библиотеку `background`).

```

\begin{tikzpicture}[level distance=8mm]
\node (root) {root}
  child { node (a) {a} }
  child { node (b) {b}
    child { node (d) {d} }
    child { node (e) {e} } }
  child { node (c) {c} };
\begin{pgfonlayer}{background}
\node[fill=red!20,inner sep=0pt,ellipse, fit=(root)(b)(d)(e)]{};
\node[fill=blue!20,inner sep=0pt,ellipse,fit=(b)(c)(e)]{};
\end{pgfonlayer}
\end{tikzpicture}

```



## 13.6 Трансформирование узлов

Узлы возможно трансформировать, но, по умолчанию, трансформирование не применяется узлам. Причина в том, что, обычно, не желательно масштабировать или вращать текст, даже когда основной рисунок трансформируется. Следует отметить, что масштабирование текста является «плохой» операцией, да и вращение не намного лучше.

Однако, иногда действительно нужно трансформировать узел, например, повернуть его на  $90^\circ$ . Есть два способа добиться этого.

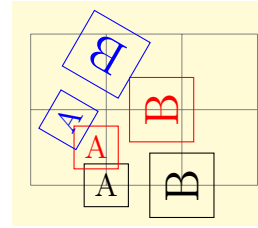
1. Можно использовать опцию `transform shape`, которая применит к форме текущую внешнюю матрицу преобразования. Например, если сказали `\tikz [scale=3]`, а затем сказали узлу `[transform shape] {X}`, то получим на рисунке огромное ( $\backslash huge$ )  $X$ .

2. Можно задать опцию трансформирования в списке опций узла. Такие преобразования всегда относятся только к узлу.

```

\begin{tikzpicture}
  [every node/.style={draw}]
  \draw[help lines](0,0) grid (3,2);
  \draw (1,0) node{A}
        (2,0) node[rotate=90,scale=1.5] {B};
  \draw [rotate=30,red]
        (1,0) node{A}
        (2,0) node[rotate=90,scale=1.5] {B};
  \draw [rotate=60,blue]
        (1,0) node[transform shape] {A}
        (2,0) node[transform shape,rotate=90,scale=1.5] {B};
\end{tikzpicture}

```



### 13.7 Явное размещение узлов на прямой и кривой

До сих пор, узлы размещались в координатах, упомянутых в пути. Но иногда нужно поместить узел, например, на середину прямолинейного отрезка, но нет желания вычислять эту точку вручную. Чтобы облегчить такие размещения, TikZ позволяет лишь указать, что конкретный узел должен размещаться где-то на линии. Есть два способа определить такую позицию: или явно, используя опцию `pos`, или неявно, размещая узел внутри операции пути.

`/tikz/pos=<fraction>` (no default)

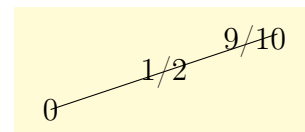
Когда опция задана узлу, узел привязывается не к последней точке пути, а к некоторой точке на линии, расположенной от «предыдущей» до текущей точки. Параметр `<fraction>` определяет, как далеко на линии должна находиться точка. Если параметр `<fraction>` равен 0 — это «предыдущая» точка, если 1 — текущая точка, все остальное — между ними, в частности, 0.5 — посередине.

Что значит «предыдущая» точка? Это зависит от предыдущей операции конструирования пути. В самом простом случае, предыдущей операцией пути может быть операцией `line-to` то есть операцией `-- <coordinate>`:

```

\tikz \draw (0,0) -- (3,1) node[pos=0]{0}
        node[pos=0.5]{1/2} node[pos=0.9]{9/10};

```

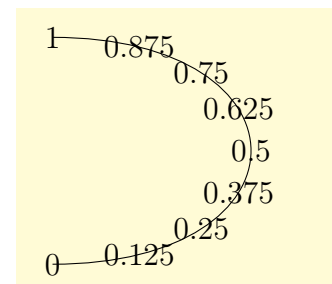


Следующий случай — операция `curve-to` (операция `..`). В этом случае, середина кривой — не обязательно точка точно посередине линии. Скорее это некоторая точка, достигаемая за 0.5 времени прохода от начала кривой до ее конца. Точная математика здесь достаточно сложна и с ней можно познакомиться по любой хорошей книге, посвященной компьютерной графике и кривым Безье.

```

\tikz \draw (0,0) .. controls +(right:3.5cm)
                and +(right:3.5cm) .. (0,3)
  \foreach \p in {0,0.125,...,1}{node[pos=\p]{\p}};

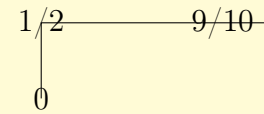
```



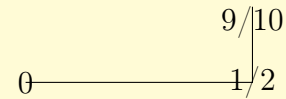


Другой интересный случай — горизонтальная/вертикальная прямая и операции `|-` и `-|`. Для них, позиция (или время) `0.5` — точно точка угла.

```
\tikz \draw (0,0) |- (3,1) node[pos=0]{0}
      node[pos=0.5]{1/2} node[pos=0.9]{9/10};
```



```
\tikz \draw (0,0) -| (3,1) node[pos=0]{0}
      node[pos=0.5]{1/2} node[pos=0.9]{9/10};
```



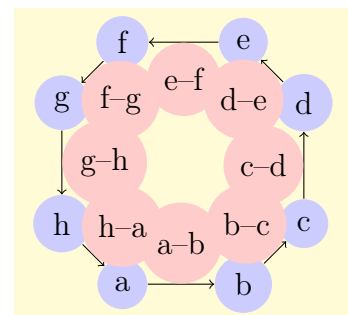
Для всех других операций конструирования пути опция размещения точки пока не работает, но, возможно, изменится в последующих версиях.

`/tikz/auto=<left or right or false>`

(default is scope's setting)

Заставляет автоматически вычислить позиции якорей согласно следующему алгоритму. Рассмотрим линию между точкам. Если задано направление `left`, то выбирается такой якорь, что узел располагается слева от этой линии. Если задано направление `right`, то узел располагается справа от этой линии. Отсутствие направления заставляет автоматически поместить узел в соответствии с последним используемым значением `left` или `right`. Если задано направление `false`, автоматическое размещение отключается. Это происходит всякий раз, когда якорь задается явно опцией `anchor` или одной из опций `above`, `below`, и т.д. Опция `auto` применима только к узлам, которые размещаются на прямых или кривых.

```
\begin{tikzpicture}
[scale=.8,auto=left,every node/.style=
{circle,fill=blue!20}]
\node (a) at (-1,-2){a}; \node (b) at ( 1,-2){b};
\node (c) at ( 2,-1){c}; \node (d) at ( 2, 1){d};
\node (e) at ( 1, 2){e}; \node (f) at (-1, 2){f};
\node (g) at (-2, 1){g}; \node (h) at (-2,-1){h};
\foreach \from/\to in
{a/b,b/c,c/d,d/e,e/f,f/g,g/h,h/a}
\draw [->] (\from) -- (\to)
      node[midway,fill=red!20] {\from--\to};
\end{tikzpicture}
```



`/tikz/swap`

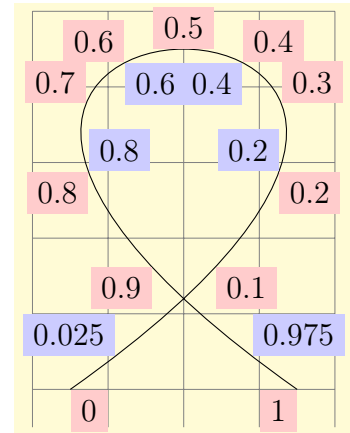
(no value)

Опция меняет роли `left` и `right` при автоматическом размещении узлов. Таким образом, если текущее значение опции `auto` равно `left`, устанавливается значение `right`, и наоборот.

```

\begin{tikzpicture}[auto]
\draw[help lines,use as bounding box]
      (0,-.5) grid (4,5);
\draw (0.5,0) .. controls (9,6)
      and (-5,6) .. (3.5,0)
\foreach \pos in
  {0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1}
  {node[pos=\pos,swap,fill=red!20] {\pos}}
\foreach \pos in {0.025,0.2,0.4,0.6,0.8,0.975}
  {node[pos=\pos,fill=blue!20] {\pos}};
\end{tikzpicture}

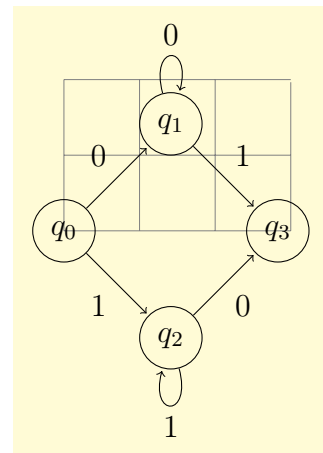
```



```

\begin{tikzpicture}
  [shorten >=1pt,node distance=2cm,auto,
   state/.style={circle,draw}]
\draw[help lines] (0,0) grid (3,2);
\node[state] (q_0) {$q_0$};
\node[state] (q_1) [above right of=q_0]{$q_1$};
\node[state] (q_2) [below right of=q_0]{$q_2$};
\node[state] (q_3) [below right of=q_1]{$q_3$};
\path[->] (q_0) edge node {0} (q_1)
edge node [swap] {1} (q_2)
(q_1) edge node {1} (q_3)
edge [loop above] node {0} ()
(q_2) edge node [swap] {0} (q_3)
edge [loop below] node {1} ();
\end{tikzpicture}

```



### /tikz/sloped

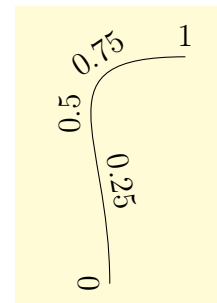
(no value)

Заставляет узел вращаться так, что горизонтальная прямая становится касательной к кривой. Вращение обычно происходит таким образом, что текст никогда не окажется «вверх ногами». Чтобы получить перевернутый текст, можно использовать параметр [rotate=180] или [allow upside down] (см. ниже).

```

\tikz \draw (0,0) .. controls +(up:2cm)
      and +(left:2cm) .. (1,3)
\foreach \p in {0,0.25,...,1}
  {node[sloped,above,pos=\p] {\p}};

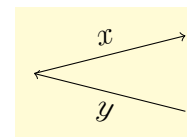
```



```

\begin{tikzpicture}[->]
\draw (0,0) -- (2,0.5) node[midway,sloped,above] {$x$};
\draw (2,-.5) -- (0,0) node[midway,sloped,below] {$y$};
\end{tikzpicture}

```

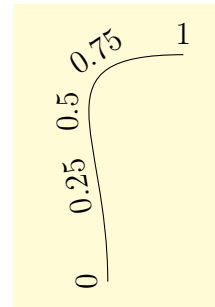


`/tikz/allow upside down=<boolean>`

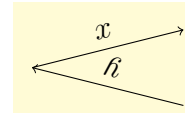
(default true, initially false)

Если установлено значение true, TikZ не будет переворачивать текст.

```
\tikz [allow upside down]
\draw (0,0) .. controls +(up:2cm)
           and +(left:2cm) .. (1,3)
\foreach \p in {0,0.25,...,1}
  {node[sloped,above,pos=\p]{\p}};
```



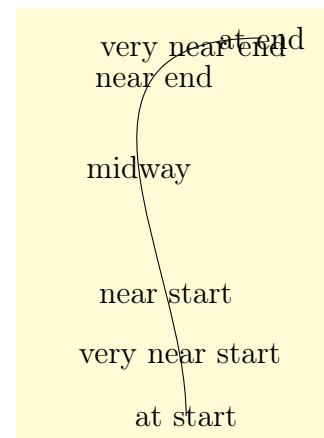
```
\begin{tikzpicture}[->,allow upside down]
\draw (0,0) -- (2,0.5) node[midway,sloped,above] {$x$};
\draw (2,-.5) -- (0,0) node[midway,sloped,below] {$y$};
\end{tikzpicture}
```



Существуют стили для определения позиции узла:

<code>/tikz/midway</code>	То же, что и <code>pos=0.5</code> .
<code>/tikz/near start</code>	То же, что и <code>pos=0.25</code> .
<code>/tikz/near end</code>	То же, что и <code>pos=0.75</code> .
<code>/tikz/very near start</code>	То же, что и <code>pos=0.125</code> .
<code>/tikz/very near end</code>	То же, что и <code>pos=0.875</code> .
<code>/tikz/at start</code>	То же, что и <code>pos=0.0</code> .
<code>/tikz/at end</code>	То же, что и <code>pos=1</code> .

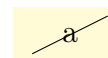
```
\tikz \draw (0,0)
  .. controls +(up:2cm)
     and +(left:3cm) .. (1,5)
  node[at end] {at end}
  node[very near end] {very near end}
  node[near end] {near end}
  node[midway] {midway}
  node[near start] {near start}
  node[very near start] {very near start}
  node[at start] {at start};
```



## 13.8 Неявное размещение узлов на прямой и кривой

Когда нужно поместить узел на прямую  $(0,0) -- (1,1)$ , естественно определить узел не после точки  $(1,1)$ , а где-то посередине. Например, написать

```
\tikz \draw (0,0) -- node{a} (1,1);
```

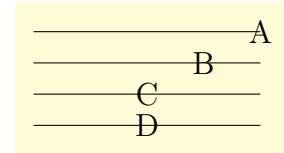


Узел окажется посередине линии. Таким образом можно задать множество узлов.

Отметим две вещи. Первое, нельзя напрямую использовать в них сложный дословный текст (`verbatim`). Если же такой текст нужен, его следует поместить в нормальный узел после координаты и добавить опцию, определяющую позицию.

Второе, какая позиция выбирается для узлов? Позиция наследуется из окружающей среды. Однако, она сохраняется только для узлов, с определенной неявно позицией. Таким образом, если добавить в окружение опцию `[near end]`, это не означает, что все узлы, заданные в этом окружении, будут расположены около конца линий. Так будут располагаться только узлы, для которых позиция определяется неявно. Как правило, это то, что требуется. Рассмотрим примеры, которые должны сделать все это более понятным.

```
\begin{tikzpicture}[near end]
\draw (0cm,4em) -- (3cm,4em) node{A};
\draw (0cm,3em) -- node{B} (3cm,3em);
\draw (0cm,2em) -- node[midway] {C} (3cm,2em);
\draw (0cm,1em) -- (3cm,1em) node[midway] {D} ;
\end{tikzpicture}
```



Как и на прямой, на кривой можно определять узлы внутри операции определения кривой: после первого `..` и после второго `...`. Как и для прямой, узлы с установленной опцией позицией будут собираться и устанавливаться после определения кривой.

## 13.9 Опции `label` и `pin`

В дополнение к операции пути `node`, узлы можно добавлять в путь, используя опции `label` и `pin`. Они главным образом применяются для простых узлов.

`/tikz/label=[<options>]<angle>:<text>` (no default)

Когда опция задается в операции `node`, она добавляет в путь новый узел после установки текущего узла. Дополнительный узел будет иметь текст `<text>`. Он размещается в направлении `<angle>` относительно основного узла, но точные правила сложнее. Пусть узел `node` с именем `main node` в данное время находится в стадии построения, а узел, определяемый опцией `label`, имеет имя `label node`. Тогда произойдет следующее:

1. Используется `<angle>`, чтобы определить позицию на границе узла `main node`. Если параметр `<angle>` отсутствует, вместо него используется значение ключа

`/tikz/label position=<angle>` (no default, initially above)

Устанавливает позицию по умолчанию для узлов `label`.

Параметр `<angle>` определяет позицию на границе формы двумя разными способами. Обычно, позиция на границе задается в виде `main node.<angle>`. Это означает, что `<angle>` может быть числом, например, `0`, или якорем, например, `north`. Специальные углы `above`, `below`, `left`, `right`, `above left`, и так далее, автоматически заменяются соответствующими углами `90`, `270`, `180`, `0`, `135`, и так далее. Специальный случай возникает, когда устанавливается ключ

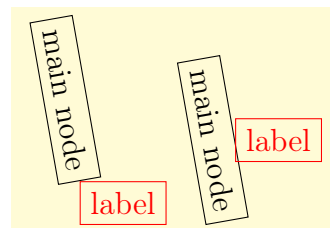
`/tikz/absolute=<true or false>` (default true)

Когда ключ установлен, `<angle>` интерпретируется по-другому: сохраняется использование точки на границе `main node`, но угол измеряется абсолютно, то есть угол обращается к точке на границе, которая лежит на прямой линии от центра `main node` направо (относительно бумаги, а не относительно локальной системы координат, узла или окружения). Различия демонстрирует следующий пример.

```

\tikz[rotate=-80, every label/.style={draw, red}]
\node[transform shape, rectangle, draw,
      label=right:label] {main node};
\tikz[rotate=-80,
      every label/.style={draw, red}, absolute]
\node[transform shape, rectangle, draw,
      label=right:label] {main node};

```

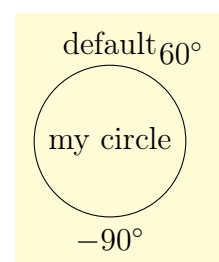


2. Затем используется якорь для узла `label`. Он определяется так, чтобы узел `label` был направлен в противоположную сторону как можно дальше от границы основного узла. Якорь, который выбирается, зависит от позиции выбранной граничной точки, ее позиции относительно центра основного узла и на том, установлена ли опция `transform shape`. Вообще говоря, выбор должен быть ожидаемым, но, иногда, в сложных случаях придется устанавливать якорь самостоятельно.

```

\tikz
\node [shape=circle, draw,
      label=default,
      label=60:$60^\circ$,
      label=below:$-90^\circ$] {my circle};

```

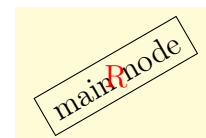


3. Одно значение угла `<angle>` является особым: если `<angle>` равен `center`, то метка размещается в центре основного узла. Это часто нужно, чтобы добавить текст метки в уже существующий узел, особенно если он вращался.

```

\tikz \node [transform shape, rotate=30,
            rectangle, draw, label={[red]center:R}]
            {main node};

```

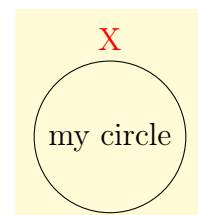


Узлу `label` можно передать опции `<options>`, которые следует поместить в квадратных скобках перед `<angle>`. Если делать так, нужно добавить фигурные скобки вокруг всего параметра опции `label`, и то же надо сделать, если в `<text>` есть скобки, запятые, точки с запятой или что-то специальное.

```

\tikz \node [shape=circle, draw,
            label={[color=red]above:X}]
            {my circle};

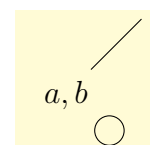
```



```

\begin{tikzpicture}
\node[shape=circle, draw, label={[name=label node]
      above left:$a, b$}] {};
\draw (label node) -- +(1,1);
\end{tikzpicture}

```



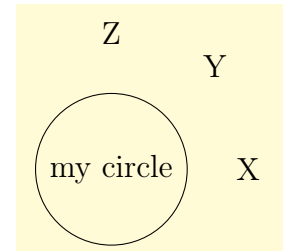
Если вводится несколько опций `label`, то соответствующие дополнительные узлы добавляются в том порядке, в котором задавались опции.

Следующие стили влияют на то, как рисуются метки:

`/tikz/label distance=<distance>` (no default, initially 0pt)

Расстояние `<distance>` дополнительно вставляется между основным узлом и узлом, определенным опцией `label`.

```
\tikz[label distance=5mm]
  \node [shape=circle,draw,label=right:X,
        label=above right:Y,label=above:Z]
        {my circle};
```



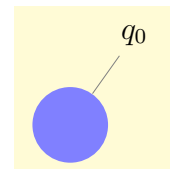
`/tikz/every label` (style, initially `draw=none, fill=none`)

Этот стиль используется в каждом узле, созданном опцией `label`.

`/tikz/pin=[<options>]<angle>:<text>` (no default)

Опция подобна опции `label`, но есть отличие: в дополнение к добавлению узла в рисунок, она также добавляет дугу (штырь) от основного узла к узлу `label`.

```
\tikz \node [circle,fill=blue!50,
            minimum size=1cm,
            pin=60:$q_0$] {};
```

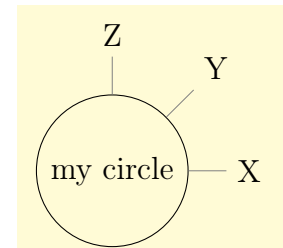


Смысл `<options>`, `<angle>` и `<text>` тот же, что и у опции `node`. Опции и стили по-разному влияют на вид штырька.

`/tikz/pin distance=<distance>` (no default, initially 3ex)

Значение параметра `<distance>` используется вместо `label distance` для определения расстояния между основным узлом и узлом, вводимым опцией `pin`.

```
\tikz[pin distance=5mm]
  \node [circle,draw,pin=right:X,
        pin=above right:Y,
        pin=above:Z] {my circle};
```



`/tikz/ every pin` (style, initially `draw=none, fill=none`)

Стиль используется в каждом узле, созданном опцией `pin`.

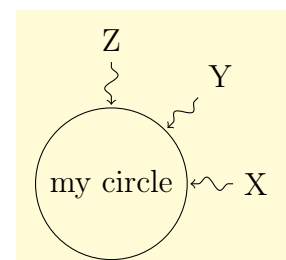
`/tikz/pin position=<angle>` (no default, initially above)

Позиция штырька по умолчанию. Работает так же, как `label position`.

`/tikz/ every pin edge` (style, initially `help lines`)

Стиль используется для каждой дуги (штырька), созданной опцией `pin`.

```
\tikz [pin distance=6mm,every pin edge/.style=
      {<-,shorten <=1pt,decorate,
       decoration={snake,pre length=4pt}}]
  \node [circle,draw,pin=right:X,pin=above right:Y,
        pin=above:Z] {my circle};
```

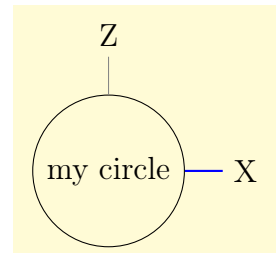


`/tikz/pin edge=<options>`

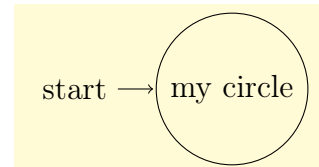
(no default, initially empty)

Опция может использоваться для того, чтобы устанавливать опции, которые должны использоваться дугами, созданными опцией `pin`.

```
\tikz[pin distance=5mm]
\node [circle,draw, pin={[pin edge=
    {blue,thick}]right:X},
    pin=above:Z] {my circle};
```



```
\tikz [every pin edge/.style={},
    initial/.style={pin={[pin distance=5mm,
    pin edge={<-,shorten <=1pt}]left:start}}]
\node [circle,draw,initial] {my circle};
```



## 13.10 Связи, использующие узлы как координаты

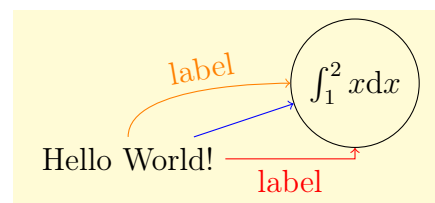
Как только узлы определены и им даны имена, имена можно использовать для ссылки на узлы. Это можно сделать двумя способами. Предположим, что было сказано `\path(0,0) node(x) {Hello World!};`, определяя узел с именем `x`.

1. Как только узел `x` определен, везде, где использовалась бы обычная координата, можно использовать `(x.<anchor>)`. Это приведет к точке рисунка, в которой находится заданный якорь `<anchor>`. Заметим, что преобразования не меняют этого определения, то есть `(x.north)` будет северным якорем узла `x`, даже если было сказано `scale=3` или `xshift=4cm`.

2. Можно также использовать `(x)` как точку. В большинстве случаев, это дает то же, что и код `(x.center)`. Действительно, если форма узла `x` — это `coordinate`, то `(x)` и `(x.center)` — это одна и та же позиция. Однако, для большинства других форм, некоторые операции конструирования пути, подобные `--`, пытаются быть «умными», когда их просят прочертить линию от такой точки или к такой точке. Если написать `(x) -- (1,1)`, то операция пути `--` будет чертить линию не от центра `x`, а от границы `x` в направлении, идущем к точке `(1,1)`. Аналогично, прямая `(1,1) -- (x)` будет завершаться на границе в направлении от точки `(1,1)`.

В дополнение к операции `--`, также корректно работают без якорей узлов операции пути `..`, `-|` и `|-`. Например (см. также подраздел на странице 107):

```
\begin{tikzpicture}
\path (0,0) node (x) {Hello World!}
      (3,1) node[circle,draw] (y)
          { $\int_1^2 x \mathrm{d} x$ };
\draw [->,blue] (x) -- (y);
\draw [->,red] (x) -|
      node[near start,below] {label} (y);
\draw [->,orange] (x) .. controls +(up:1cm) and +(left:1cm) ..
      node[above,sloped] {label} (y);
\end{tikzpicture}
```



## 13.11 Связи, использующие операцию `edge`

Операция `edge` работает, как и операция `to`, которая добавляется после прорисовки основного пути, позволяя определить вид каждой дуги. Как и операция `node`, операция `edge` временно приостанавливает конструирование текущего пути и создает новый путь `p`, который будет нарисован после того, как будет нарисован основной путь. При этом путь `p` может полностью отличаться от основного пути своими опциями. Если в основном пути есть несколько операций `to` и/или `node`, каждая из них создает свой путь, то они рисуются в том порядке, в котором встраивались в основной путь.

```
\path ... edge[<options>]<nodes> (<coordinate>)...;
```

Как результат операции `edge`, после основного пути добавляется путь

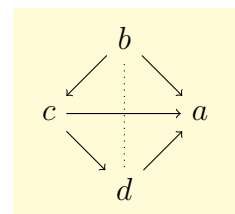
```
\path[every edge,<options>](\tikztostart) <path>;
```

Здесь, `<path>` — `to path`. Заметим, что, в отличие от пути, добавленного операцией `to`, здесь перед `<path>` добавляется координата (`\tikztostart`) (что не нужно для операции `to`, так как она там часть основного пути).

Координата `\tikztostart` — последняя координата в пути, как раз перед операцией `edge`, так же как и для операций `node` и `to`. Однако, есть одно исключение из этого правила: если операция `edge` непосредственно предшествует операции `node`, то этот, только что заявленный узел, является стартовой координатой (а не координата, в которой этот заявленный узел помещается — маленькое, но важное отличие). В этом отношении `edge`, отличается и от `node`, и от `to`.

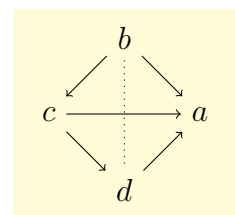
Если подряд стоит несколько операций `edge`, начальная координата одна и та же для всех, а целевые точки нет. Таким образом, начальная координата предшествует первой операции `edge`. Как и операция `node`, операция `edge` не изменяет текущий путь. В частности, она не изменяет последнюю посещенную позицию, например:

```
\begin{tikzpicture}
\node (a) at (0:1) {$a$};
\node (b) at (90:1) {$b$} edge [->] (a);
\node (c) at (180:1) {$c$} edge [->] (a)
edge [->] (b);
\node (d) at (270:1) {$d$} edge [->] (a)
edge [dotted] (b)
edge [->] (c);
\end{tikzpicture}
```



Есть другой способ определить эту же диаграмму, используя операцию `edge`:

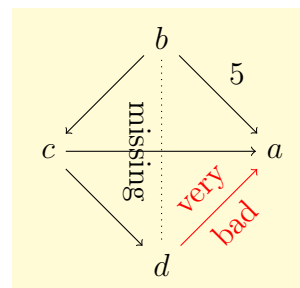
```
\begin{tikzpicture}
\foreach \name/\angle in {a/0,b/90,c/180,d/270}
\node (\name) at (\angle:1) {$\name$};
\path[->] (b) edge (a) edge (c)
edge [-,dotted] (d)
(c) edge (a) edge (d)
(d) edge (a);
\end{tikzpicture}
```





Операция `edge` наследует опции основного пути, но может переопределять их.

```
\begin{tikzpicture}
\foreach \name/\angle in {a/0,b/90,c/180,d/270}
  \node (\name) at (\angle:1.5) {$\name$};
\path[->] (b) edge node[above right] {$5$} (a)
  edge (c)
  edge [-,dotted] node[below,sloped]
    {missing} (d)
  (c) edge (a)
  edge (d)
  (d) edge [red] node[above,sloped] {very}
    node[below,sloped] {bad} (a);
\end{tikzpicture}
```



Вместо `every to`, в начале основного пути устанавливается стиль `every edge`, который выполняется для каждой дуги.

`/tikz/every edge (initially draw)` (style, no value)

## 13.12 Ссылки на узлы вне текущего рисунка

### 13.12.1 Ссылка на узел в другом рисунке

Можно (но не тривиально) ссылаться на узлы в рисунках, отличных от текущего. Это означает, что можно создать рисунок и узел в нем и, позже, начертить линию к этому узлу из некоторой точки другого рисунка. Ссылка на узлы в других рисунках реализуется по следующей схеме.

1. Добавить опцию `remember picture` ко всем рисункам, которые содержат узлы, на которые предстоит ссылаться, а также ко всем рисункам, из которых предстоит ссылаться на узлы в других рисунках.

2. Добавить опцию `overlay` к пути или к целому рисунку, которые содержат ссылки на узлы в других рисунках. Эта опция переключает вычисление ограничивающего прямоугольника.

3. Использовать драйвер, который поддерживает запоминание картинок, и выполнить  $\TeX$ -операцию дважды (детали см. [1, раздел 75.3.2]).

`/tikz/remember picture=<boolean>` (no default, initially false+)

Заставляет TikZ запомнить позицию текущего рисунка на странице. Эта попытка может оказаться неудачной и зависит от того, какой внутренний драйвер используется. Кроме того, даже если запоминание работает, позиция становится доступной только при втором выполнении  $\LaTeX$ 'а. При условии, что запоминание работает, можно сказать

```
\tikzstyle{every picture}+=[remember picture]
```


Это заставит TikZ запоминать все рисунки, добавляя соответствующую запись в aux-файл (который обычно не очень большой) по одной строке для каждого рисунка в документе. После этого не нужно будет волноваться о запоминании рисунков.

`/tikz/overlay` (no value)


Опция предназначена для ссылок на узлы в других рисунках, но ее можно использовать и в других ситуациях. В результате, все в пределах текущего окружения не рассматривается, когда вычисляется ограничивающий прямоугольник текущего рисунка. Нужно определить эту опцию на всех путях (или, по крайней мере, на всех частях путей), которые содержат ссылку на узел в другом рисунке. Иначе, TikZ попытается сделать текущий рисунок достаточно большим, чтобы охватить узлы в других рисунках. Однако, при втором прогоне, TeX создаст еще больший рисунок. Если еще не ясно, что предстоит делать, можно определить опцию `overlay` со всеми рисункам, которые содержат ссылку на другие рисунки.

Рассмотрим несколько примеров, которые работают, только если документ обрабатывается драйвером, поддерживающим запоминание рисунков. Внутри текущего текста поместим два рисунка, содержащих узлы с именами `n1` и `n2` с помощью кода

```
\tikz[remember picture] \node[shape=circle,fill=red!50] (n1) {};
```

определяющего узел  и кода

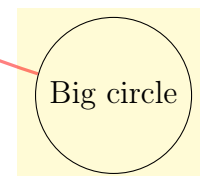
```
\tikz[remember picture] \node[fill=blue!50] (n2) {};
```

определяющего другой узел, отличный от первого . Чтобы связать эти узлы, создадим новый рисунок, используя оверлейную опцию и опцию, запоминающую рисунки.

```
\begin{tikzpicture}[remember picture,overlay]
\draw[->,very thick,opacity=.5] (n1) -- (n2);
\end{tikzpicture}
```

Созданный так рисунок пуст: он имеет нулевой размер и содержит стрелку, которая лежит вне его границ. Наконец, соединим еще один узел из третьего рисунка с первыми двумя узлами. Здесь, оверлейная опция определяется только для линии, которая не является частью этого рисунка.

```
\begin{tikzpicture}[remember picture]
\node (c) [circle,draw] {Big circle};
\draw [overlay,->,very thick,red,opacity=.5]
(c) to[bend left] (n1) (n1) -| (n2);
\end{tikzpicture}
```



### 13.12.2 Ссылка на узел текущей страницы

Есть специальный узел с именем `current page`, который может использоваться для обращения к текущей странице. Это узел прямоугольной формы, юго-западный якорь которого — нижний левый угол страницы, а северо-восточный якорь — верхний правый. Хотя этот узел внутренне обрабатывается специальным способом, можно сослаться на него так, как будто он был определен для ссылки в некотором запоминаемом рисунке, отличном от текущего. Таким образом, задавая опции `remembered picture` и `overlay` для рисунка, можно абсолютно позиционировать узлы на странице. В первом примере размещается некоторый текст в нижнем левом углу текущей страницы:

```
\begin{tikzpicture}[remember picture,overlay]
  \node [xshift=1cm,yshift=1cm] at (current page.south west)
    [text width=6cm,fill=yellow!20,rounded corners,above right]
    {Этот текст располагается в нижнем левом углу страницы.};
\end{tikzpicture}
```

Следующий код рисует круг в середине страницы.

```
\begin{tikzpicture}[remember picture,overlay]
  \draw [line width=1mm,black!20] (current page.center) circle (3cm);
\end{tikzpicture}
```

В следующем примере слово `Example` перекрывает текст страницы.

```
\begin{tikzpicture}[remember picture,overlay]
\node[rotate=60,scale=10,text opacity=.2] at (current page.center){Example};
\end{tikzpicture}
```

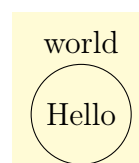
## 13.13 Отложенный код и отложенные опции

Все опции, заданные в узле локально, воздействуют только на этот узел. В большинстве случаев именно это и нужно. Но, иногда, нужно заставить некоторые опции «задержаться», а некоторые добавить к опциям узла позднее. Добавить путь после узла, позволяют опции `append after command` и `prefix after command`. В этих случаях может быть полезен макрос `\tikzlastnode`, который расширяется до последнего узла пути.

Опция узла `late options=<options>` задается в пути через некоторое время после создания узла (но не как параметр в команде `node`) и произойдет следующее: объявляется (способом, описанным ниже) уже существующий узел (`<existing node>`) и, опции `<options>` выполняются в локальной области видимости. Большинство этих опций не дадут никаких последствий, так как нельзя изменить вид узлов, то есть, нельзя изменить красный узел на зеленый узел, используя `late option`. Однако, задавая опции `append after command` и `prefix after command` внутри `<options>` (прямо или косвенно) можно добиться того, что заданный путь будет выполняться с установкой макроса `\tikzlastnode` в объявленном узле.

Результат тот, что можно определить, скажем, опцию `label` внутри `<options>` и добавить метку в узел, который уже был создан. Аналогично, можно использовать опцию `on chain`, чтобы сделать уже существующий узел `<existing node>` частью цепочки. Существующий узел `<existing node>` определяется следующим образом: если внутри `<options>` используется опция `name=<existing node>`, то используется это имя. Иначе, если последняя координата текущего пути имела форму (`<existing node>`), то используется имя (`<existing node>`). Иначе, возникает ошибка.

```
\begin{tikzpicture}
  \node (a) [draw,shape=circle] {Hello};
  \path (a) [late options={label=above:world}];
\end{tikzpicture}
```



Этот текст располагается в нижнем левом углу страницы.

# Глава 14

## Матрицы и выравнивание

При создании картинки часто возникает задача выравнивания его частей. Есть разные способы ее решения. Например, почти все задачи выравнивания можно решить, применяя «умное» использование якорей. Однако, это часто приводит к сложному коду. Более простой путь состоит в использовании матриц. TikZ-матрица подобна среде `{tabular}` или `{array}` ЛАТЭХ'а, только вместо текста каждая клетка содержит небольшой рисунок или вершину. Размеры клеток автоматически корректируются так, чтобы поместилось все содержимое клетки.

Матрицы — мощный инструмент, но сами нуждаются в корректной обработке: каждая строка (и последняя тоже) должна завершаться `\\`. Многие идеи, реализованные в TikZ-матрицах, принадлежат и поддерживаются Марком Виброу (Mark Wibrow). Библиотека `matrix` определяет много полезных опций и стилей для построения матриц.

### 14.1 Матрицы как узлы

Матрицы в большинстве задач обрабатываются как узлы. Это означает, что для ее создания используется команда пути `node` со специальной опцией `matrix`. Поэтому матрица имеет форму, и эту форму можно нарисовать или заполнить, матрицу можно использовать в дереве, можно обращаться к различным якорям матрицы, и так далее.

`/tikz/matrix=<true or false>` (default `true`)

Опция передается команде пути `node`, сообщая, будет ли узел содержать матрицу.

`/tikz/every matrix` (style, initially empty)

Этот стиль используется в каждой матрице.

Матрицы часто являются единственным объектом в пути. Для этого случая внутри `{tikzpicture}` можно использовать для `\path node[matrix]` специальное сокращение:

`/tikz/\matrix`

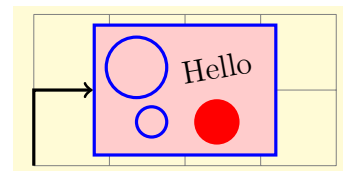
Даже при том, что матрицы — узлы, некоторые опции не дают того же результата, как в случае нормальных узлов:

1. Операции вращения и масштабирования не дают никакого результата для матрицы в целом (хотя можно проводить преобразование содержимого каждой ячейки матрицы).
2. Для многослойных форм можно устанавливать только текстовую часть узла.
3. Все опции, начинающиеся со слова `text`, типа `text width`, не дают никакого результата.

```

\begin{tikzpicture}
\draw[help lines] (0,0) grid (4,2);
\matrix[fill=red!20,draw=blue,very thick,
  ampersand replacement=\&] (my matrix) at (2,1)
{ \draw (0,0) circle (4mm); \&
  \node[rotate=10] {Hello};\&
  \draw (0.2,0) circle (2mm); \&
  \fill[red] (0,0) circle (3mm);\&
};
\draw[very thick,->] (0,0) |- (my matrix.west);
\end{tikzpicture}

```



## 14.2 Изображение ячеек

Матрицы состоят из строк ячеек (или клеток). Каждая строка (включая последнюю!) должна заканчиваться командой `\\`. Символ `&` используется для отделения ячеек. Опция `ampersand replacement=` команды `\matrix` позволяет заменить символ `&` на другой (например, `\&`). В каждой ячейке нужно поместить команды рисования образа, который называется рисунком ячейки. Однако, рисунки ячеек не включаются в окружение `{pgfpicture}`, поскольку рисунки ячеек не имеют уровней. Нет необходимости определять заранее, сколько строк или столбцов содержит матрица, если одна из строк содержит меньше ячеек, чем другая, автоматически добавляются пустые ячейки.

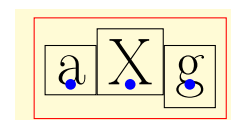
### 14.2.1 Выравнивание ячеек

Для каждого рисунка ячейки вычисляется ограничивающий прямоугольник, которые вместе с точками начала рисунка ячеек определяют, как выравниваются ячейки. Рассмотрим рисунки ячеек в первой строке. Каждый имеет ограничивающий прямоугольник, и где-то в этом ограничивающем прямоугольнике лежит начало рисунка ячейки (начало может даже лежать вне ограничивающего прямоугольника, но пока не будем на это обращать внимания). Рисунки ячеек сдвигаются так, чтобы все их начальные точки лежали на одной горизонтальной прямой. Если необходимо, можно сдвинуть некоторую ячейку вверх или вниз. Вершина строки определяется самым высоким рисунком ячейки в строке, низ строки определяется самой низким рисунком ячеек (высота строки — максимальное  $y$ -значение, а глубина строки — отрицательное минимальное  $y$ -значение ограничивающих прямоугольников).

```

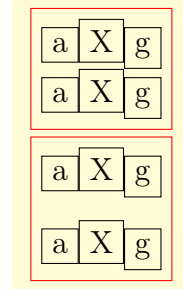
\begin{tikzpicture}
[every node/.style={draw=black,anchor=base,font=\huge}]
\matrix [draw=red,ampersand replacement=\&]
{\node {a}; \fill[blue] (0,0) circle (2pt); \&
 \node {X}; \fill[blue] (0,0) circle (2pt); \&
 \node {g}; \fill[blue] (0,0) circle (2pt); \&
};
\end{tikzpicture}

```



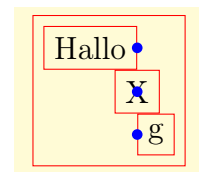
Выравнивание строк происходит следующим образом: для каждой строки рисунки ячеек вертикально выравниваются так, что их начала лежат на горизонтальной линии. Затем вторая строка размещается ниже первой строки так, что низ первой строки касается верха второй строки (если не используется опция `row sep`, чтобы добавить дополнительное пространство). Затем, верх третьей строки касается низа второй строки, и так далее. Как правило, каждая строка будет иметь единую высоту и глубину.

```
\begin{tikzpicture}
[every node/.style={draw=black,anchor=base}]
\matrix [draw=red,ampersand replacement=\&]
{
\node {a}; \& \node {X}; \& \node {g}; \\
\node {a}; \& \node {X}; \& \node {g}; \\
};
\matrix [row sep=3mm,draw=red,
ampersand replacement=\&] at (0,-2)
{
\node {a}; \& \node {X}; \& \node {g}; \\
\node {a}; \& \node {X}; \& \node {g}; \\
};
\end{tikzpicture}
```

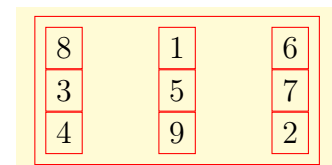


Обратимся к столбцам. Правила выравнивания рисунков ячеек по столбцам похожи на правила выравнивания их по строкам: каждый рисунок ячейки в столбце сдвигается по горизонтали так, что их начала лежат на одной вертикальной линии. Левый край столбца — самый левый край ограничивающих прямоугольников, а самый правый край столбца — самый правый край ограничивающих прямоугольников. Затем второй столбец размещается так, чтобы правый край первого столбца касался левого края второго столбца (если не используется опция `column sep`), и так далее.

```
\begin{tikzpicture}[every node/.style={draw}]
\matrix [draw=red,ampersand replacement=\&]
{
\node[left] {Hallo};\fill[blue](0,0) circle(2pt); \\
\node {X}; \fill[blue](0,0) circle(2pt); \\
\node[right]{g}; \fill[blue](0,0) circle(2pt); \\
};
\end{tikzpicture}
```



```
\begin{tikzpicture}[every node/.style={draw}]
\matrix [draw=red,column sep=1cm,
ampersand replacement=\&]
{\node {8}; \& \node{1}; \& \node {6}; \\
\node {3}; \& \node{5}; \& \node {7}; \\
\node {4}; \& \node{9}; \& \node {2}; \\
};
\end{tikzpicture}
```



## 14.2.2 Корректировка пробелов между столбцами и строками

Есть разные способы установить и отрегулировать интервал между столбцами и строками в матрице. Во-первых, можно использовать опции `column sep` и `row sep`, чтобы установить пробелы между строками и столбцами. Во-вторых, можно добавить опции расстояний к символу `&` и команде `\`, корректируя интервал между двумя конкретными столбцами или строками. Дополнительно, можно определить, нужен ли пробел между центрами ячеек двух столбцов или строк, или между их границами.

`/tikz/column sep=<spacing list>` (no default)

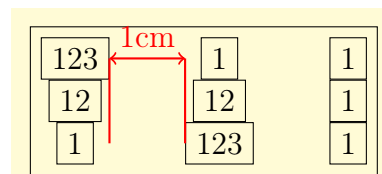
`/tikz/row sep=<spacing list>` (no default)

Опции устанавливают пробелы (по умолчанию 0pt) между столбцами и строками. Параметр `<spacing list>` обычно содержит единственный размер, например, 2pt, или список размерных чисел, разделенных запятыми, а также два ключевых слова `between origins` и `between borders`. Результат от задания такого списка определяется по следующему алгоритму. Сначала, все числа из списка складываются, чтобы вычислить конечный размер. Затем важную роль играет расположение одного из двух ключевых слов: если последним указано `between borders` или их нет вообще, то пробелы между двумя столбцами вставляются как обычно. Однако если последним указано `between origins`, то расстояние между столбцами корректируется так, что разность между началами всех ячеек в первом столбце (все они лежат на прямой линии), и началами всех ячеек во втором столбце и есть заданное расстояние. Опция `between origins` может использоваться для столбцов, упомянутых только в первой строке, то есть, нельзя определить эту опцию для столбцов в последующих строках.

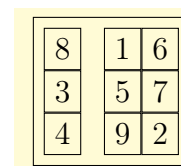
Когда определяется расстояние между столбцами, можно указать, добавляется оно между низом первой строки и верхом второй строки, или является расстоянием между началами этих двух строк.

Особенности установки пробелов демонстрируются ниже на примерах.

```
\begin{tikzpicture}[ampersand replacement=\&]
\matrix [draw,column sep=1cm,nodes=draw]
{\node(a){123};\&\node(b){1}; \&\node{1};\&
\node{12}; \&\node{12}; \&\node{1};\&
\node(c){1}; \&\node(d){123};\&\node{1};\& };
\draw[red,thick](a.east) -- (a.east |- c)
(d.west) -- (d.west |- b);
\draw[<->,red,thick] (a.east) -- (d.west |- b) node[above,midway]{1cm};
\end{tikzpicture}
```



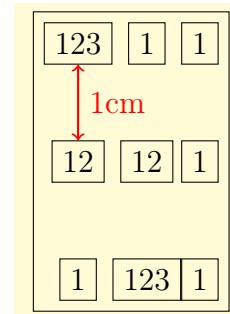
```
\begin{tikzpicture}[ampersand replacement=\&]
\matrix [draw,nodes=draw,column sep=1mm]
{\node {8}; &[2mm] \node{1}; &[-1mm] \node {6};\&
\node {3}; \& \node{5}; \& \node {7}; \&
\node {4}; \& \node{9}; \& \node {2}; \& };
\end{tikzpicture}
```



```

\begin{tikzpicture}[ampersand replacement=\&]
\matrix [draw,row sep=1cm,nodes=draw]
{\node (a) {123}; \& \node {1}; \& \node {1};\\
\node (b) {12}; \& \node {12}; \& \node {1};\\
\node {1}; \& \node {123}; \& \node {1}; \\ };
\draw [<->,red,thick] (a.south) -- (b.north)
node [right,midway] {1cm};
\end{tikzpicture}

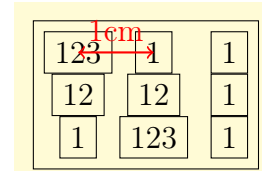
```



```

\begin{tikzpicture}[ampersand replacement=\&]
\matrix [draw,column sep={1cm,between origins},
nodes=draw]
{ \node(a) {123}; \& \node (b) {1}; \& \node {1};\\
\node {12}; \& \node {12}; \& \node {1}; \\
\node {1}; \& \node {123}; \& \node {1}; \\ };
\draw [<->,red,thick] (a.center) -- (b.center) node [above,midway]{1cm};
\end{tikzpicture}

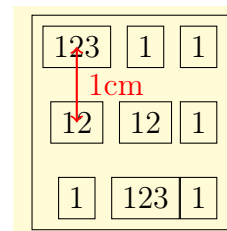
```



```

\begin{tikzpicture}[ampersand replacement=\&]
\matrix[draw,row sep={1cm,between origins},nodes=draw]
{\node (a) {123}; \& \node {1}; \& \node {1};\\
\node (b) {12}; \& \node {12}; \& \node {1};\\
\node {1}; \& \node {123}; \& \node {1}; \\ };
\draw [<->,red,thick] (a.center) -- (b.center)
node [right,midway] {1cm};
\end{tikzpicture}

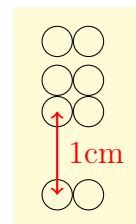
```



```

\begin{tikzpicture}[ampersand replacement=\&]
\matrix [row sep=1mm]
{\draw (0,0) circle (2mm); \& \draw (0,0) circle (2mm);\\
\draw (0,0) circle (2mm); \&
\draw (0,0) circle (2mm);\\[-1mm]
\draw (0,0) coordinate (a) circle (2mm); \&
\draw (0,0) circle (2mm); \\[1cm,between origins]
\draw (0,0) coordinate (b) circle (2mm); \&
\draw (0,0) circle (2mm);\\ };
\draw [<->,red,thick] (a.center) -- (b.center) node [right,midway]{1cm};
\end{tikzpicture}

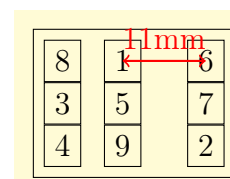
```



```

\begin{tikzpicture}[ampersand replacement=\&]
\matrix [draw,nodes=draw,column sep=1mm]
{\node {8}; \&[2mm] \node(a){1}; \\
\node {3}; \& \node {5}; \& \node {7}; \\
\node {4}; \& \node {9}; \& \node {2}; \\ };
\draw [<->,red,thick] (a.center) -- (b.center) node [above,midway]{11mm};
\end{tikzpicture}

```

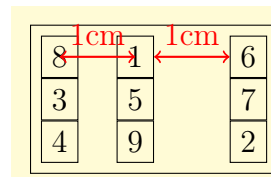




```

\begin{tikzpicture}[ampersand replacement=\&]
\matrix[draw,nodes=draw,
        column sep={1cm,between origins}]
{\node (a) {8}; \& \node (b) {1}; \\
 \node {3}; \& \node {5}; \& \node {7}; \\
 \node {4}; \& \node {9}; \& \node {2}; \\ };
\draw[<->,red,thick] (a.center) -- (b.center) node [above,midway]{1cm};
\draw[<->,red,thick] (b.east) -- (c.west) node [above,midway]{1cm};
\end{tikzpicture}

```



### 14.2.3 Стили и опции ячеек

**/tikz/every cell**={<row>}{<column>} (style, no default, initially empty)

Стиль устанавливается в начале каждого рисунка ячейки с двумя параметрами, являющимися текущей строкой <row> и столбцом <column> ячейки. Заметим, что установка в этом стиле опции `draw`, чтобы нарисовать все узлы, работать не будет, так как опция `draw` должна передаваться каждому узлу. В этом стиле (и во всех ячейках), номера текущей строки <row> и столбца <column> доступны через счетчики `\pgfmatrixcurrentrow` и `\pgfmatrixcurrentcolumn`.

**/tikz/cells**=<options> (no default)

Добавляет опции <options> к стилю `every cell`. Это только сокращение для кода `every cell/.append style=<options>`.

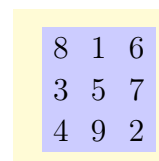
**/tikz/nodes**=<options> (no default)

Добавляет опции <options> к стилю `every node`. Это только сокращение для кода `every node/.append style=<options>`. Основное использование этой опции — установка некоторых опций для узлов матрицы, которые не относятся ко всей матрице.

```

\begin{tikzpicture}[ampersand replacement=\&]
\matrix [nodes={fill=blue!20,minimum size=5mm}]
{ \node {8}; \& \node{1}; \& \node {6}; \\
 \node {3}; \& \node{5}; \& \node {7}; \\
 \node {4}; \& \node{9}; \& \node {2}; \\ };
\end{tikzpicture}

```



Следующие стили используются для изменения вида конкретных строк, столбцов или ячеек. Если определяется несколько стилей, они выполняются в порядке следования (каждый стиль `every cell` выполняется перед расположенными ниже).

**/tikz/column <number>** (style, no value)

Стиль используется для каждой ячейки в столбце с номером <number>.

**/tikz/every odd column** (style, no value)

Стиль используется для каждой ячейки в нечетном столбце.

**/tikz/every even column** (style, no value)

Стиль используется для каждой ячейки в четном столбце.

**/tikz/row <row number> column <column number>** (style, no value)

Стиль используется для ячейки в указанной строке и указанном столбце.

Следующие стили аналогичны описанным выше, но применяются к строкам.

<code>/tikz/row &lt;number&gt;</code>	(style, no value)
<code>/tikz/every odd row</code>	(style, no value)
<code>/tikz/every even row</code>	(style, no value)

```
\begin{tikzpicture}
  [row 1/.style={color=red},
   column 2/.style={color=green!50!black},
   row 3 column 3/.style={color=blue}]
\matrix[ampersand replacement=\&]
{\node {8}; \& \node{1}; \& \node {6}; \\
 \node {3}; \& \node{5}; \& \node {7}; \\
 \node {4}; \& \node{9}; \& \node {2}; \\
}
\end{tikzpicture}
```

8	1	6
3	5	7
4	9	2

```
\begin{tikzpicture}
  [column 1/.style={anchor=base west},
   column 2/.style={anchor=base east},
   column 3/.style={anchor=base}]
\matrix [ampersand replacement=\&]
{\node {123}; \& \node{456}; \& \node {789}; \\
 \node {12}; \& \node{45}; \& \node {78}; \\
 \node {1}; \& \node{4}; \& \node {7}; \\
}
\end{tikzpicture}
```

123	456	789
12	45	78
1	4	7

Во многих матрицах все рисунки ячеек имеют почти один и тот же код. Например, ячейки обычно начинаются с `\node{` и заканчиваются `}`. Следующие опции позволяют определить код, который выполняется в ячейках:

<code>/tikz/execute at begin cell=&lt;code&gt;</code>	(no default)
---	--------------

Код выполняется в начале каждой непустой ячейки.

<code>/tikz/execute at end cell=&lt;code&gt;</code>	(no default)
---	--------------

Код выполняется в конце каждой непустой ячейки.

<code>/tikz/execute at empty cell=&lt;code&gt;</code>	(no default)
---	--------------

Код выполняется в каждой пустой ячейке.

```
\begin{tikzpicture}
  [matrix of nodes/.style={
    execute at begin cell=\node\bgroup,
    execute at end cell=\egroup; }]
\matrix
  [matrix of nodes,ampersand replacement=\&]
{ 8 \& 1 \& 6 \\ \& 5 \& 7 \\ 4 \& 9 \& 2 \\
}
\end{tikzpicture}
```

8	1	6
3	5	7
4	9	2

```

\begin{tikzpicture}
  [matrix of nodes/.style={
    execute at begin cell=\node\bgroup,
    execute at end cell=\egroup;,%
    execute at empty cell=\node{--}; }]
\matrix
  [matrix of nodes, ampersand replacement=\&]
{ 8 \& 1 \& \ 3 \& \ 7 \& \ 2 \& };
\end{tikzpicture}

```

```

8 1 -
3 - 7
- - 2

```

## 14.3 Якоря матрицы

Так как матрица — узел, можно использовать опцию `anchor`. Есть два способа повлиять на размещение матрицы. Во-первых, можно воспользоваться опцией

`/tikz/matrix anchor=<anchor>` (no default)

Эта опция дает тот же самый результат, что и опция `anchor`, но опция `anchor` применима только к самой матрице, а не к ячейкам внутри нее. Если задать в качестве опции к матрице опцию `anchor=north`, все узлы внутри матрицы будут использовать указанный якорь, если только для некоторого узла явно не установлен другой. Для сравнения, опция `matrix anchor` устанавливает якорь для матрицы, а для узлов внутри матрицы значение опции `anchor` остается неизменным.

```

\begin{tikzpicture}
\matrix
[matrix anchor=west] at (0,0)
{ \node {123}; \  % остается якорь center
  \node {12}; \
  \node {1}; \ };
\matrix
[anchor=west] at (0,-2)
{ \node {123}; \  % якорь west
  \node {12}; \
  \node {1}; \ };
\end{tikzpicture}

```

```

123
12
1
123
12
1

```

Второй способ «прикрепить» матрицу состоит в том, чтобы использовать якорь внутреннего узла матрицы. Для этого используется опция

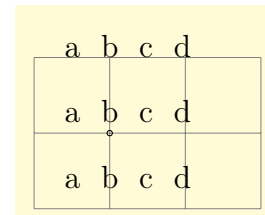
`/tikz/anchor=<anchor or node.anchor>` (no default)

Обычно, параметр этой опции ссылается на якорь матричного узла. Однако можно задать параметр в форме `<node>.<anchor>`, где `<node>` — узел, определенный в матрице, а `<anchor>` — якорь этого узла. В этом случае, вся матрица сдвигается так, что этот якорь перемещается в точку, указанную после опции `at` матрицы. То же самое имеет место и для опции `matrix anchor`.

```

\begin{tikzpicture}[ampersand replacement=\&]
\draw[help lines] (0,0) grid (3,2);
\matrix[matrix anchor=inner node.south,
        anchor=base, row sep=3mm] at (1,1)
{\node{a}; \& \node{b};
  \& \node{c}; \& \node{d};\&
\node{a}; \& \node{inner node}{b};
  \& \node{c}; \& \node{d};\&
\node{a}; \& \node{b};
  \& \node{c}; \& \node{d};\& };
\draw (inner node.south) circle (1pt);
\end{tikzpicture}

```



## 14.4 Определение активных символов

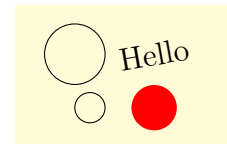
TikZ для отделения ячеек использует символ `&`, а pgf фактически использует для этого команду `\pgfmatrixnextcell` и использование символа `&` обычно приводит к ошибке. Поэтому, чтобы заменить символ для отделения ячеек, рекомендуется использовать опцию `/tikz/ampersand replacement=<macro name or empty>`.

Например, код `ampersand replacement=\&` позволит использовать `\&` для отделения столбцов матрицы (что и было сделано).

```

\tikz \matrix [ampersand replacement=\&]
{\draw (0,0) circle (4mm); \&
  \node[rotate=10] {Hello};\&
\draw (0.2,0) circle (2mm); \&
  \fill[red] (0,0) circle (3mm); \& };

```



## 14.5 Примеры

Первый пример построен на основе кода, написанного Кджеллом Мэйгном Фоском (Kjell Magne Fauske), который основан на работе: K. Bossley, M. Brown, and C. Harris, *Neurofuzzy identification of an autonomous underwater vehicle*, *International Journal of Systems Science*, 1999, 30, 901–913.

Следующие примеры построены на основе кода Марка Виброу и первые два из них являются перерисовками примеров Тимоти ван Занда (Timothy van Zandt) из документации к PStricks.

```

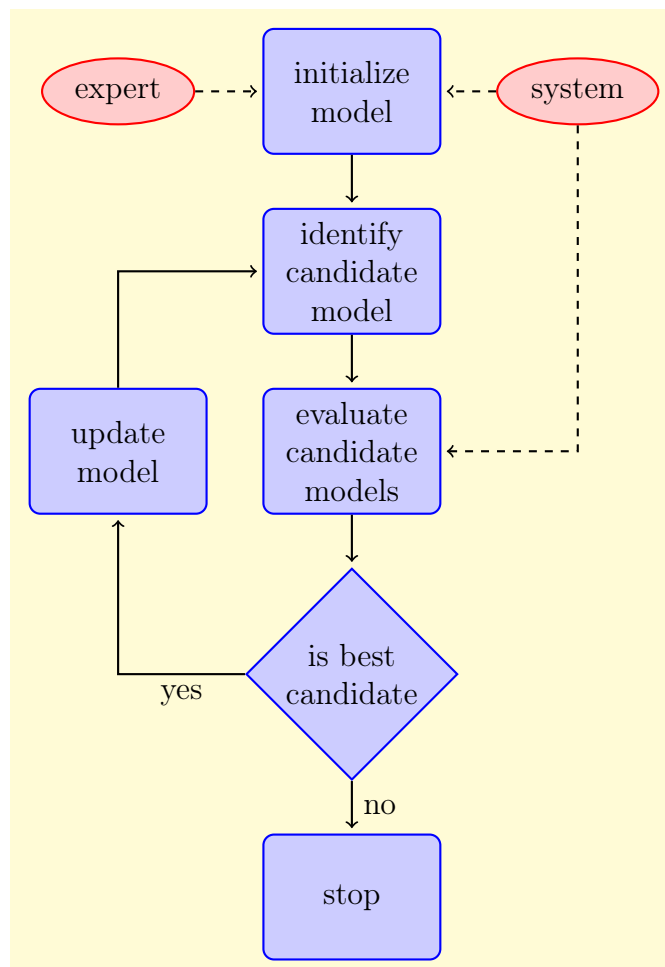
\begin{tikzpicture}
[ auto,decision/.style={diamond, draw=blue, thick, fill=blue!20,
  text width=4.5em,align=flush center,inner sep=1pt},
block/.style = {rectangle,draw=blue,thick,fill=blue!20,
  text width=5em,align=center,rounded corners,minimum size=4em},
line/.style = {draw,thick,-latex',shorten >=2pt},
cloud/.style={draw=red,thick,ellipse,fill=red!20,minimum size=2em} ]

```

```

\matrix [column sep=5mm,row sep=7mm,ampersand replacement=\&]
{ % row 1
  \node [cloud] (expert) {expert}; \&
  \node [block] (init) {initialize model}; \&
  \node [cloud] (system) {system}; \\
% row 2
  \& \node [block] (identify) {identify candidate model}; \& \\
% row 3
  \node [block] (update) {update model}; \&
  \node [block] (evaluate) {evaluate candidate models}; \& \\
% row 4
  \& \node [decision] (decide) {is best candidate}; \& \\
% row 5
  \& \node [block] (stop) {stop}; \& \\
\begin{scope}[every path/.style=line]
\path (init) -- (identify); \path (identify) -- (evaluate);
\path (evaluate) -- (decide); \path (update) |- (identify);
\path (decide) -| node [near start] {yes} (update);
\path (decide) -- node [midway] {no} (stop);
\path [dashed] (expert) -- (init); \path [dashed] (system) -- (init);
\path [dashed] (system) |- (evaluate);
\end{scope}
\end{tikzpicture}

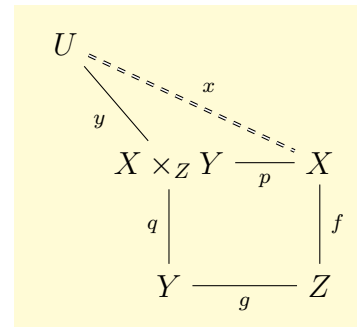
```



```

\begin{tikzpicture}[ampersand replacement=\&]
\matrix [matrix of math nodes,row sep=1cm]
{ |(U)| U \&[2mm] \&[8mm] \\
\& |(XZY)| X \times_Z Y \& |(X)| X \\
\& |(Y)| Y \& |(Z)| Z \\
};
\begin{scope}[every node/.style=
midway,auto,font=\scriptsize]
\draw [double, dashed] (U) -- node {$x$} (X);
\draw (X) -- node {$p$} (X -| XZY.east)
(X) -- node {$f$} (Z) -- node {$g$} (Y)
-- node {$q$} (XZY) -- node {$y$} (U);
\end{scope}
\end{tikzpicture}

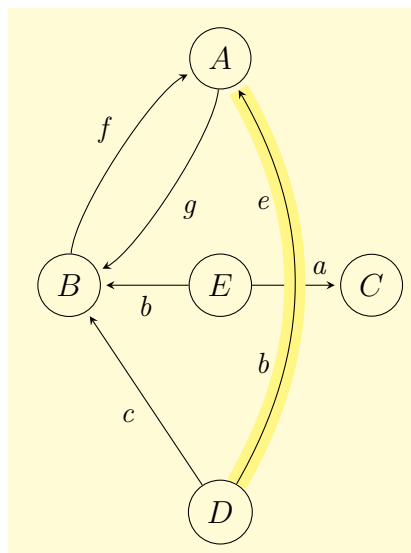
```



```

\begin{tikzpicture}[>=stealth,->,auto,shorten >=2pt,looseness=.5]
\matrix[matrix of math nodes,column sep={2cm,between origins},
row sep={3cm,between origins},ampersand replacement=\&,
nodes={circle,draw,minimum size=7.5mm}]
{ \& |(A)| A \& \\
|(B)| B \& |(E)| E \& |(C)| C \\
\& |(D)| D \\
};
\begin{scope}[every node/.style={font=\small\itshape}]
\draw(A) to [bend left] node[midway] {g} (B);
\draw(B) to [bend left] node[midway] {f} (A);
\draw(D) -- node [midway] {c} (B);
\draw(E) -- node [midway] {b} (B);
\draw(E) -- node [near end] {a} (C);
\draw[-,line width=8pt,draw=yellow!60]
(D) to [bend right,looseness=1] (A);
\draw (D) to [bend right,looseness=1]
node[near start] {b} node[near end] {e} (A);
\end{scope}
\end{tikzpicture}

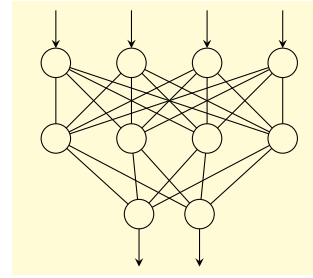
```



```

\begin{tikzpicture}
\matrix (network)
  [ampersand replacement=\&,
  matrix of nodes,nodes in empty cells,
  nodes={outer sep=0pt,circle,
  minimum size=4pt,draw},
  column sep={1cm,between origins},
  row sep={1cm,between origins}]
{
  \&          \&          \& \& \\
  \&          \&          \& \& \\
|[draw=none]|\& |[xshift=1mm]|\& |[xshift=-1mm]|\& \& \& \\
\foreach \a in {1,...,4}
{ \draw (network-3-2) -- (network-2-\a);
  \draw (network-3-3) -- (network-2-\a);
  \draw [-stealth] ([yshift=5mm]network-1-\a.north) -- (network-1-\a);
  \foreach \b in {1,...,4}
    \draw (network-1-\a) -- (network-2-\b);
}
\draw [stealth-] ([yshift=-5mm]network-3-2.south) -- (network-3-2);
\draw [stealth-] ([yshift=-5mm]network-3-3.south) -- (network-3-3);
\end{tikzpicture}

```



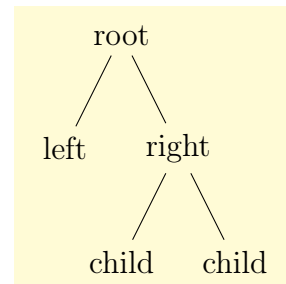
# Глава 15

## Создание растущих деревьев

### 15.1 Операция `child`

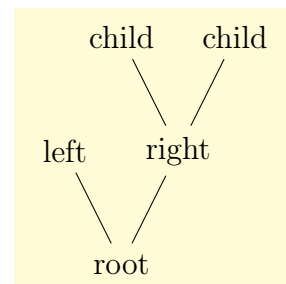
Деревья — общий способ визуализации иерархических структур. Простое дерево выглядит следующим образом:

```
\begin{tikzpicture}
\node {root}
  child {node {left}}
  child {node {right}}
    child {node {child}}
    child {node {child}}};
\end{tikzpicture}
```



Деревья растут вверх, а не вниз, но у математиков и программистов деревья часто растут вниз, так удобнее. Естественный рост дерева можно определить так:

```
\begin{tikzpicture}
\node {root} [grow'=up]
  child {node {left}}
  child {node {right}}
    child {node {child}}
    child {node {child}}};
\end{tikzpicture}
```



Деревья определяются добавлением дочернего дерева к узлу пути операциями

```
\path ... child [<options>] foreach <variables> in {<values>} {<child path>} ...;
```

Эта операция должна следовать непосредственно за операцией `node` или другой операцией `child`, хотя допустимо, чтобы первой операции `child` предшествовали некоторые опции (см. далее).

Когда операция `node` имеет вид `node {X}` и сопровождается операцией `child`, TikZ начинает считать дочерние узлы, которые следуют за первоначальным узлом `{X}`. Для этого, он открывает ввод и сохраняет каждый дочерний узел и его параметры, пока не встретит операцию пути, отличную от операции `child`. Заметим, что при этом фиксируются символы кода всего текста в параметрах дочерних узлов, а это означает, что нельзя использовать дословный текст (`verbatim`) в узлах внутри `child`.



Как только дочерние узлы набраны и посчитаны, TikZ начинает генерировать дочерние узлы. Для каждого дочернего узла основного (корневого) узла, TikZ вычисляет соответствующую позицию размещения. Для каждого дочернего узла система координат преобразуется так, чтобы начало узла попадало в эту позицию. Затем рисуется дочерний путь `<child path>`, который, как правило, состоит только из спецификации `node`, что в результате и приводит к узлу, который будет нарисован в позиции дочернего узла. И, наконец, рисуется дуга от первого узла в `<child path>` к родительскому узлу.

Необязательная часть `foreach` (заметим, что нет наклонной черты перед `foreach`!) позволяет определить множество дочерних узлов единственной командой `child`. Идея состоит в следующем: выражение `\foreach` (внутренне) используется для того, чтобы выполнить итерации по списку `<values>`. Для каждого значения из этого списка к текущему узлу добавляется новый дочерний узел. Синтаксис для `<variables>` и `<values>` то же, что и для команды `\foreach` (см. [1, глава 56]). Например, код

```
node {root} child [red] foreach \name in {1,2} {node {\name}}
```

равносилен коду

```
node {root} child[red] {node {1}} child[red] {node {2}}
```

Аналогично, код

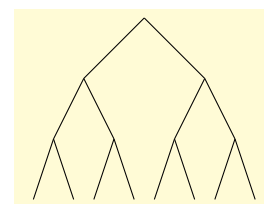
```
node {root} child[\pos] foreach
  \name/\pos in {1/left,2/right} {node[\pos] {\name}}
```

равносилен коду

```
node {root} child[left] {node[left] {1}} child[right] {node[right] {2}}
```

Можно вкладывать объекты один в другой, как в следующем примере:

```
\begin{tikzpicture}[scale=2,level distance=4mm,
  level/.style={sibling distance=8mm/#1}]
\coordinate
  child foreach \x in {0,1}
    {child foreach \y in {0,1}
      {child foreach \z in {0,1}}};
\end{tikzpicture}
```



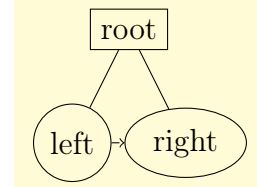
## 15.2 Дочерние пути и дочерние узлы

Для каждого `child` корневого узла, его дочерний путь `<child path>` вставляется в конкретную позицию рисунка (правила размещения см. в разделе 15.5). Первый узел в `<child path>`, если он существует, является специальным и называется дочерним узлом — `<child node>`. Если нет первого узла в `<child path>`, то есть, если `<child path>` отсутствует (включая фигурные скобки) или если он не начинается с `node` или с `coordinate`, то автоматически добавляется пустой дочерний узел формы `coordinate`.

Рассмотрим, например, код `\node {x} child {node {y}} child;`. Для первого дочернего узла `<child path>` состоит из дочернего узла `node {y}`. Для второго дочернего узла нет определенных узлов и, таким образом, он только `coordinate`.

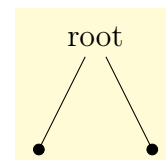
Как и для любого нормального узла, можно дать дочернему узлу имя, повернуть его, или использовать опции, чтобы повлиять на его представление.

```
\begin{tikzpicture}
  \node[rectangle,draw] {root}
    child {node[circle,draw] (left node) {left}}
    child {node[ellipse,draw](right node) {right}};
  \draw[dashed,->](left node) -- (right node);
\end{tikzpicture}
```



Во многих случаях `<child path>` будет состоять только из определенного дочернего узла и, возможно, дочернего узла для этого дочернего узла. Однако, определение узлов может сопровождаться другим произвольным материалом, который будет добавляться в рисунок, преобразованный к системе координат дочернего узла. Ради удобства, операция перемещения в  $(0,0)$  вставляется автоматически в начале пути. Например:

```
\begin{tikzpicture}
  \node {root}
    child {[fill] circle (2pt)}
    child {[fill] circle (2pt)};
\end{tikzpicture}
```



В конце дочернего пути `<child path>` можно добавить специальную операцию пути с именем `edge from parent`. Если эта операция не задается где-нибудь в пути, она автоматически добавляется в конце пути. Она заставляет добавить в путь дугу, соединяющую коренной узел с дочерним узлом. Задавая опции к этой операции, можно влиять на вид дуги. Кроме того, узлы, следующие за операцией `edge from parent` будут размещаться на этой дуге (см. раздел 15.6). Подведем итог.

1. Дочерний путь начинается с спецификации узла. Если он не задан, он добавляется автоматически.

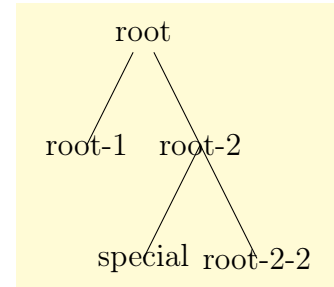
2. Дочерний путь заканчивается операцией `edge from parent`, возможно сопровождаемой узлами, которые будут помещены на эту дугу. Если операция не задана, она добавляется автоматически в конце.

### 15.3 Именованное дочерних узлов

Дочерние узлы можно именовать, как и любой другой узел, используя или опцию `name` или специальный синтаксис, в котором имя узла размещается в круглых скобках между операцией `node` и текстом узла. Если имя дочернего узла не назначается явно, TikZ автоматически назначает ему имя следующим образом: предположим, что имя коренного узла `parent` (если такое имя не назначено, TikZ сделает это сам, но имя не будет доступно пользователям). Первый дочерний узел корневого узла получит имя `parent-1`, второй — `parent-2` и так далее. Соглашение об именах работает рекурсивно. Если второй дочерний узел `parent-2` имеет дочерние узлы, то первый из них получит имя `parent-2-1`, второй — `parent-2-2` и так далее.

Если явно назначить имя дочернему узлу, другое имя уже не генерируется (узел не может иметь двух имен). Однако, внутренний счет узлов продолжается, а это означает, что третий дочерний узел узла `parent` называется `parent-3`, независимо от того, назначено ли было имя на первый и/или второй дочерний узел узла `parent`. Например,

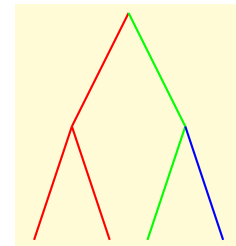
```
\begin{tikzpicture}
\node (root) {root}
  child
  child {child {coordinate (special)}
    child };
\node at (root-1) {root-1};
\node at (root-2) {root-2};
\node at (special) {special};
\node at (root-2-2) {root-2-2};
\end{tikzpicture}
```



## 15.4 Опции деревьев и узлов

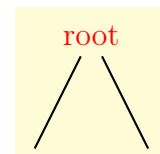
Каждая операция `child` может иметь собственные опции `<options>`, которые относятся к дочернему узлу в целом, включая все его дочерние узлы. Например:

```
\begin{tikzpicture}
[thick,level 2/.style={sibling distance=10mm}]
\coordinate
  child[red] {child child}
  child[green] {child child[blue]};
\end{tikzpicture}
```



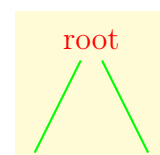
Однако, опции корневого узла не влияют на дочерние узлы, поскольку опции такого узла всегда локальны. Поэтому дуги в следующем дереве черного, а не красного цвета.

```
\begin{tikzpicture}[thick]
\node [red] {root}
  child child;
\end{tikzpicture}
```



Возникает вопрос: как задать опции для всех дочерних узлов? Можно установить опции для всего пути: `\path [red] node {root} child child;`. Но вместо этого проще задать опции перед первым дочерним узлом следующим образом:

```
\begin{tikzpicture}[thick]
\node [red] {root}
  [green] % опция для всех дочерних узлов
  child child;
\end{tikzpicture}
```



Перечислим все правила размещения опций в дереве:

1. Опции для целого дерева задаются **перед** корневым узлом.
2. Опции для корневого узла задаются прямо в его операции `node`.
3. Опции для всех дочерних узлов можно задать между корневым узлом и первым дочерним узлом.
4. Опции конкретного дочернего пути задаются как опции операции `child`.
5. Опции единственного дочернего узла, но не целого дочернего пути, задаются как опции команды `node` в дочернем пути.

```
\begin{tikzpicture}
\path
  [...]          % Опции ... применяются к целому дереву
  node[...] {root} % Опции ... применяются только к корневному узлу
  [...]          % Опции ... применяются ко всем дочерним узлам
  child[...]     % Опции ... применяются к узлу и всем его дочерним узлам
  { node[...] {} % Опции ... применяются только к этому дочернему узлу
    ... }
  child[...]     % Опции ... применяются к узлу и всем его дочерним узлам
;
\end{tikzpicture}
```

Есть дополнительные стили, которые влияют на состояние дочерних узлов:

`/tikz/every child` (style, initially empty)

Стиль используется в начале каждого дочернего пути, как будто бы код стиля был задан в качестве опций каждой операции `child`.

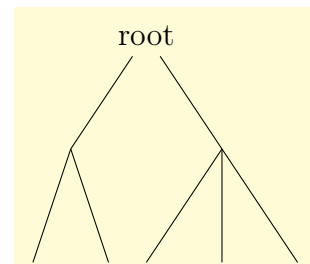
`/tikz/every child node` (style, initially empty)

Стиль используется в начале каждого дочернего узла в дополнение к стилю `every child`.

`/tikz/level=<number>` (style, no default, initially empty)

Стиль используется в начале каждого множества дочерних узлов, где `<number>` — текущий уровень в текущем дереве. Например, если задан код `\node {x} child child;`, то до первого дочернего узла используется `level=1`. Стиль или код этого ключа будет передавать `<number>` как первый параметр. Если первый потомок `child` сам имеет дочерний узел, то для него используется `level=2`.

```
\begin{tikzpicture}
[level/.style={sibling distance=20mm/#1}]
\node {root}
  child { child child }
  child { child child child };
\end{tikzpicture}
```



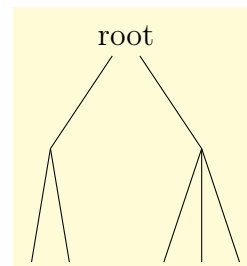
`/tikz/level <number>` (style, initially empty)

Стиль используется в дополнение к стилю `level`. В коде `\node {x} child child;` будет выполняться следующий список ключей: `level=1, level 1`.

```

\begin{tikzpicture}
  [level 1/.style={sibling distance=20mm},
  level 2/.style={sibling distance=5mm}]
  \node {root}
    child { child child }
    child { child child child };
\end{tikzpicture}

```



## 15.5 Размещение дочерних узлов

### 15.5.1 Основная идея

Возможно наиболее трудная часть в процессе рисования дерева — правильное размещение дочерних узлов. Как правило, дочерние узлы имеют разные размеры и их совсем не просто упорядочить так, чтобы не потратить впустую много места, чтобы дочерние узлы не накладывались друг на друга и равномерно располагались по рисунку либо сами узлы, либо их центры. Поиск хорошей позиции задача особенно трудная для первого дочернего узла, поскольку она может зависеть от размера последнего дочернего узла.

В TikZ выбран сравнительно простой подход к размещению дочерних узлов. Чтобы вычислить позицию дочернего узла, во внимание принимается только номер текущего дочернего узла в списке дочерних узлов и число дочерних узлов в этом списке. Таким образом, если узел имеет пять дочерних узлов, то существует фиксированная позиция для первого дочернего узла, для второго дочернего узла, и так далее. Эти позиции не зависят от размера дочерних узлов и, следовательно, узлы могут пересекаться. Однако можно использовать опции, позволяющие немного сдвинуть конкретный дочерний узел, а это не столь уж сложная задача, как кажется.

Хотя размещение дочернего узла зависит только от его номера в списке и общего количества дочерних узлов, все остальное в размещении легко определить опциями. Можно изменить расстояние между дочерними узлами (`sibling distance`) и расстояние между уровнями дерева (`level distance`). Эти расстояния можно менять от уровня к уровню. Направление роста дерева можно изменять и глобально, и для его частей. Можно даже определить собственную функцию роста, чтобы упорядочить дочерние узлы, например, по окружности или по специальным линиям или кривым.

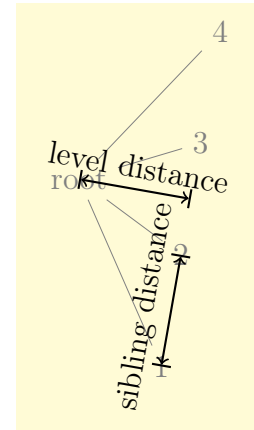
### 15.5.2 Функция роста дерева по умолчанию

Функция роста по умолчанию работает так: если задан узел и пять дочерних узлов, то последние будут размещаться на прямой линии, соединяющей их центры (или, более общо, их якоря) на расстоянии `sibling distance` друг от друга. Эта линия ортогональна текущему направлению роста дерева, которое устанавливается опциями `grow` и `grow'` (последняя опция обращает порядок дочерних узлов). Расстояние от линии до коренного узла задается расстоянием между уровнями `level distance`.

```

\begin{tikzpicture}
\path [help lines]
node (root) {root}
[grow=-10] child {node {1}}
              child {node {2}}
              child {node {3}}
              child {node {4}};
\draw[|<->,thick] (root-1.center)
  -- node[above,sloped] {sibling distance}
  (root-2.center);
\draw[|<->,thick] (root.center)
  -- node[above,sloped]
  {level distance} +(-10:\tikzleveldistance);
\end{tikzpicture}

```



**/tikz/level distance**=<distance>

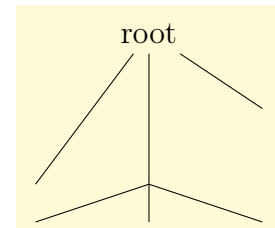
(no default, initially 15mm)

Определяет расстояние между разными уровнями дерева, то есть между узлом и линией, на которой располагаются его дочерние узлы. Когда задан один дочерний узел, то это расстояние до дочернего узла.

```

\begin{tikzpicture}
\node {root}
[level distance=20mm]
  child
  child {[level distance=5mm]
    child
    child
    child}
child[level distance=10mm];
\end{tikzpicture}

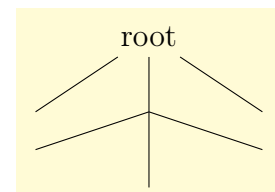
```



```

\begin{tikzpicture}
[level 1/.style={level distance=10mm},
level 2/.style={level distance=5mm}]
\node {root}
  child
  child {
    child
    child[level distance=10mm]
    child}
  child;
\end{tikzpicture}

```



**/tikz/sibling distance**=<distance>

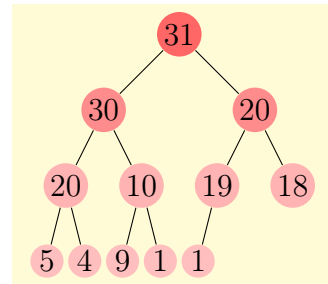
(no default, initially 15mm)

Определяет расстояние между якорями дочерних узлов одного родительского узла.

```

\begin{tikzpicture}
[level distance=10mm,
every node/.style={fill=red!60,shape=circle,inner sep=1pt},
level 1/.style={sibling distance=20mm,nodes={fill=red!45}},
level 2/.style={sibling distance=10mm,nodes={fill=red!30}},
level 3/.style={sibling distance=5mm,nodes={fill=red!25}}]
\node {31}
  child {node {30}}
    child {node {20}}
      child {node {5}}
      child {node {4}}}}
  child {node {10}}
    child {node {9}}
    child {node {1}}}}
  child {node {20}}
    child {node {19}}
      child {node {1}}
      child[missing]}
    child {node {18}}}};
\end{tikzpicture}

```



`/tikz/grow=<direction>`

(no default)

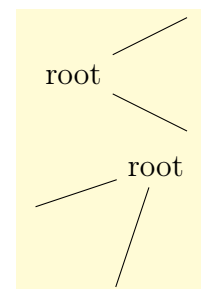
Определяет направление `<direction>` роста дерева. Параметр `<direction>` может быть или углом или одной из следующих специальных текстовых строк: `down`, `up`, `left`, `right`, `north`, `south`, `east`, `west`, `north east`, `north west`, `south east`, `south west`. Все они имеют очевидные значения, скажем, `south west` — то же, что и угол  $135^\circ$ . Как побочный эффект, устанавливает функцию роста по умолчанию (сверху вниз).

Особенности размещения узлов лучше всего продемонстрировать на примерах.

```

\tikz \node {root} [grow=right]
      child child;

```



```

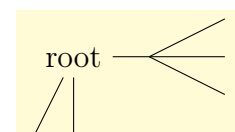
\tikz \node {root} [grow=south west]
      child child;

```

```

\begin{tikzpicture}
[level distance=10mm,sibling distance=5mm]
\node {root} [grow=down]
  child child child[grow=right] {
    child child child};
\end{tikzpicture}

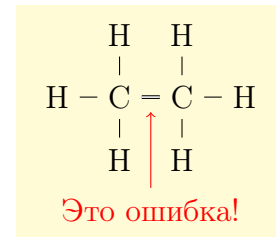
```



```

\begin{tikzpicture}[level distance=2em]
\node {C}
  child[grow=up] {node {H}}
  child[grow=left] {node {H}}
  child[grow=down] {node {H}}
  child[grow=right] {node {C}}
    child[grow=up] {node {H}}
    child[grow=right] {node {H}}
    child[grow=down] {node {H}}
  edge from parent[double] coordinate (wrong)};
\draw[arrows=<- ,color=red]
  ([yshift=-2mm]wrong) -- +(0,-1) node[below]{Это ошибка!};
\end{tikzpicture}

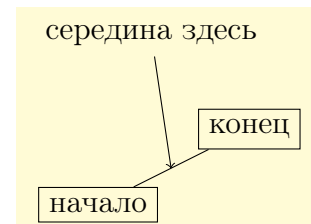
```



```

\begin{tikzpicture}
\node[rectangle,draw] (a) at (0,0) {начало};
\node[rectangle,draw] (b) at (2,1) {конец};
\draw (a) -- (b)
  node[coordinate,midway] {}
  child[grow=100,arrows=<-]
    {node[above] {середина здесь}};
\end{tikzpicture}

```



`/tikz/grow'=<direction>` (no default)

Определяет направление роста, как и `grow`, только располагает дочерние узлы в противоположном порядке.

### 15.5.3 Отсутствие дочерних узлов

Иногда один или даже несколько дочерних узлов должны как бы «отсутствовать», но такие узлы должны рассматриваться как дочерние, учитываться при их подсчете и размещении.

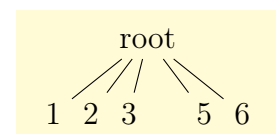
`/tikz/missing=<true or false>` (default `true`)

Задаёт дочерний узел, увеличивает текущий счетчик дочерних узлов, но в остальном такой дочерний узел игнорируется. В частности, игнорируется его содержимое.

```

\begin{tikzpicture}[level distance=10mm,sibling distance=5mm]
\node {root} [grow=down]
  child { node {1} }
  child { node {2} }
  child { node {3} }
  child[missing] { node {4} }
  child { node {5} }
  child { node {6} };
\end{tikzpicture}

```





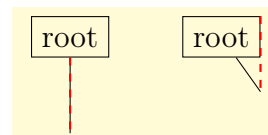
### 15.5.4 Собственные функции роста дерева

`/tikz/growth parent anchor=<anchor>` (no default, initially center)

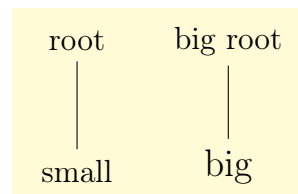
Определяет якорь корневого узла, используемый для вычисления позиции дочерних узлов. Например, когда задан один дочерний узел, и `level distance=2cm`, то дочерний узел размещается на два сантиметра ниже якоря `<anchor>` корневого узла.

Ниже, в первом примере, обе красных линии имеют длину в 1cm, а во втором верхние и нижние узлы выравниваются по верху и низу, соответственно.

```
\begin{tikzpicture}[level distance=1cm]
\node [rectangle,draw] (a) at (0,0) {root}
  [growth parent anchor=south] child;
\node [rectangle,draw] (b) at (2,0) {root}
  [growth parent anchor=north east] child;
\draw [red,thick,dashed] (a.south) -- (a-1);
\draw [red,thick,dashed] (b.north east) -- (b-1);
\end{tikzpicture}
```



```
\begin{tikzpicture}
[level distance=2cm,growth parent anchor=north,
 every node/.style={anchor=north,rectangle,draw}
 every child node/.style={anchor=south}]
\node at (0,0) {root} child {node {small}};
\node at (2,0) {big root} child {node {\large big}};
\end{tikzpicture}
```



`/tikz/growth function=<macro name>` (no default, initially an internal function))

Позволяет установить новую функцию роста дерева; при этом `<macro name>` — имя макроса без параметров. Этот макрос будет вызываться для каждого дочернего узла. Первоначальная функция — внутренняя функция, соответствующая росту дерева вниз.

В библиотеке `trees` (см. [1, глава 53]) определены дополнительные функции роста деревьев.

## 15.6 Дуги из корневого узла

Каждый дочерний узел связывается с корневым узлом дугой специального вида с именем `edge from parent`. Эта дуга добавляется в `<child path>`, когда определяется операция пути

```
\path ... edge from parent[<options>]...;
```

Она используется только внутри `<child paths>` и должна задаваться в его конце, возможно, сопровождаемая спецификациями узла. Если `<child path>` не содержит этой операции, она будет добавляться в конце `<child path>` автоматически. Операция имеет несколько последствий. Самое важное то, что она вставляет текущую дугу из родительского пути в дочерний путь.

Дуга, исходящая из корневого пути, устанавливается опцией

`/tikz/edge from parent path=<path>` (no default, initially code shown below)

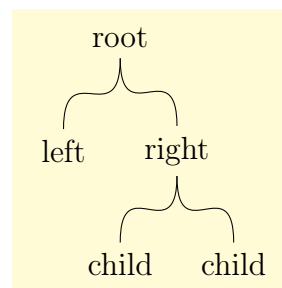
Первоначально, этот путь имеет следующий вид:

`(\tikzparentnode\tikzparentanchor) - (\tikzchildnode\tikzchildanchor)`

Макрос `\tikzparentnode` расширяется до имени корневого узла. Это работает даже тогда, когда корневому узлу не было явно определено имя, но внутреннее имя автоматически генерируется всегда. Макрос `\tikzchildnode` расширяется до имени дочернего узла. Два макроса `...anchor` по умолчанию пусты. Так, что по существу вставляется сегмент пути `(\tikzparentnode) - (\tikzchildnode)`; , являющийся дугой от родителя к потомку.

Можно заменить эту дугу, например, заменить прямую кривой:

```
\begin{tikzpicture}[edge from parent path=
{(\tikzparentnode.south) .. controls +(0,-1)
and +(0,1) .. (\tikzchildnode.north)}]
\node {root} child {node {left}}
              child {node {right}}
              child {node {child}}
              child {node {child}}};
\end{tikzpicture}
```



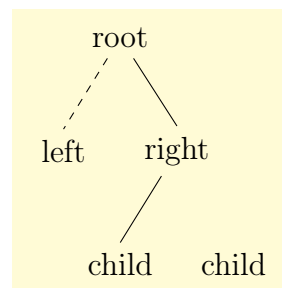
Другие полезные дуги определены в библиотеке `trees` (см. [1, глава 53]).

Как сказано выше, якоря дуги по умолчанию из корневого пути пусты. Однако, можно их установить, используя опцию

`/tikz/child anchor=<anchor>` (no default, initially border)

Определяет якорь, где дуга из корневого пути встречает дочерний узел, устанавливая макрос `\tikzchildanchor` в `.<anchor>`. Если определить якорь как `border`, то макрос `\tikzchildanchor` определяется пустой строкой. В результате дуга из корневого пути встретит дочерний узел на границе в автоматически вычисленной позиции.

```
\begin{tikzpicture}
\node {root} [child anchor=north]
  child {node {left}}
    edge from parent[dashed]}
  child {node {right}}
    child {node {child}}
    child {node {child}}
    edge from parent[draw=none]}};
\end{tikzpicture}
```



`/tikz/parent anchor=<anchor>` (no default, initially border)

Опция аналогична предыдущей, только определяет якорь для корневого узла.

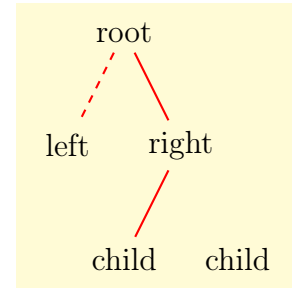
Помимо вставки дуги от коренного пути, операция `edge from parent` имеет побочный эффект: параметр `<options>` вставляется прямо перед дугой из корневого пути, но прежде, чем вставляются эти опции, устанавливается стиль

`/tikz/edge from parent` (style, initially draw)

```

\begin{tikzpicture}
[edge from parent/.style={draw,color=red,thick}]
\node {root}
  child {node {left} edge from parent[dashed]}
  child {node {right}
    child {node {child}}
    child {node {child}
      edge from parent[draw=none]}}};
\end{tikzpicture}

```



Опции `<options>`, вставленные перед дугой из коренного пути, обращаются ко всему дочернему пути. Таким образом, нельзя, скажем, нарисовать как часть дочернего пути красный круг, а затем получить синюю дугу в родительский путь. Однако, как всегда, дочерний узел — это узел и потому может быть нарисован по-другому.

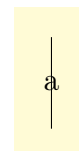
Наконец, операция `edge from parent` дает еще один результат: она вынуждает все узлы, указанные после операции, размещаться на дуге. Результат тот же, что и при добавлении опции `pos` ко всем этим узлам (см. также раздел 13.7).

Как пример, рассмотрим следующий код:

```

\begin{tikzpicture}
\node (root) {}
  child {node (child) {}
    edge from parent node {a}};
\end{tikzpicture}

```



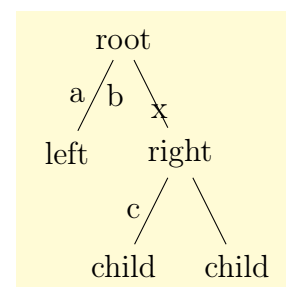
Из примера видно, что операция `edge from parent` и последующая операция `node` вместе дают тот же результат, что и код `(root) -- (child) node [pos=0.5] {a}`.

В заключение, рассмотрим более сложный пример.

```

\begin{tikzpicture}
\node {root}
  child {node {left}
    edge from parent
    node[left] {a}
    node[right] {b}}
  child {node {right}
    child {node {child}
      edge from parent
      node[left] {c}}
    child {node {child}
      edge from parent
      node[near end] {x}}};
\end{tikzpicture}

```



# Глава 16

## Графики функций

### 16.1 TikZ и создание графиков

Существует много мощных программ для построения графиков функций, например, `gnuplot`, `maple` или `mathematica`. Эти программы могут производить вывод двух разных типов: (1) вывод полного графического изображения в определенном формате (например, `pdf`), который включает в себя все команды низкого уровня, необходимые для того, чтобы нарисовать полный график (включая оси координат и метки); (2) вывод простых таблиц данных в форме длинного списка координат.

Заметим, что необходимость использования TikZ для создания графиков функций возникает не часто. Программы типа `gnuplot` могут создавать очень сложные графики и обычно не составляет большого труда включить такие графики в `tex`-файл. Однако, есть несколько причин, по которым иногда следует потратить время и силы на то, чтобы создать график с помощью команд `pgf`. Вот некоторые из них:

- Фактически все графические изображения, произведенные внешними программами, используют шрифты, отличные от используемых в основном документе.
- Еще хуже то, что формулы будут выглядеть совершенно по-другому, если они вообще могут быть созданы.
- Ширина линий обычно либо слишком большая, либо слишком маленькая.
- Масштабирование вложенных рисунков часто приводит к рассогласованию между размерами объектов (строк, линий, цифр, символов, ...) в графике и в тексте.
- Автоматическая сетка, сгенерированная большинством программ, часто достаточно груба и отвлекает внимание от сути рисунка.
- Автоматические метки, сгенерированные большинством программ, часто представляют собой загадочные числовые данные.
- Большинство программ позволяют легко создавать диаграммы низкого качества: форму без содержания.
- Стрелки и метки на графике во внешних рисунках никогда не соответствуют аналогичным объектам, используемым в остальной части документа.

### 16.2 Операция `plot`

Операция пути `plot` может использоваться для того, чтобы присоединить к пути линию или кривую, которая проходит через несколько точек. Эти точки или задаются простым списком, читаемым из некоторого файла, или вычисляются на лету.

Синтаксис операции `plot` имеет несколько версий.

```
\path ... --plot<further arguments>...;
```

Рисует график кривой по точкам, определенным в списке `<further arguments>`. Текущий путь (или подпуть) просто продолжается к первой заданной точке и далее. Детали списка `<further arguments>` объяснены ниже.

```
\path ... plot<further arguments>...;
```

Рисует график кривой по точкам, определенным в списке `<further arguments>`, который начинается с первой заданной точки.

При определении точек, через которые проходит график, `<further arguments>` используются тремя разными способами:

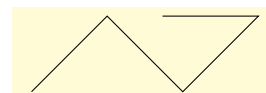
- 1) `--plot[<local options>]coordinates{<coordinate 1> ... <coordinate n>}`
- 2) `--plot[<local options>]file{<filename>}`
- 3) `--plot[<local options>]<coordinate expression>`
- 4) `--plot[<local options>]function{<gnuplot formula>}`

Рассмотрим каждый способ отдельно.

## 16.3 График по явно заданному набору точек

В первых двух примерах точки задаются непосредственно:

```
\tikz \draw plot coordinates
      {(0,0) (1,1) (2,0) (3,1) (2,1) (30:2cm)};
```



Следующий пример показывает различие между операциями `plot` и `--plot`:

```
\begin{tikzpicture}
\draw (0,0) -- (1,1) plot coordinates {(2,0) (4,0)};
\draw [red,xshift=5cm] (0,0) -- (1,1) -- plot coordinates {(2,0) (4,0)};
\end{tikzpicture}
```



## 16.4 График по набору точек из файла

Второй способ определения точек состоит в том, чтобы поместить их во внешний файл с именем `<filename>`. В настоящее время, существует единственный формат такого файла, который TikZ понимает: каждая строка должна содержать два числа, которые отделены пробелом. Все что стоит в строке после двух чисел игнорируется. Кроме того, пустые строки, строки, начинающиеся с `%` или `#` игнорируются (это формат, производимый `gnuplot`, если ввести `set terminal table`). Приведем несколько строк из файла, используемого в следующем примере (это таблица для функции  $\sin x$ ):

```
#Curve 0, 20 points
#x y type
0.00000 0.00000 i
```

```

0.52632 0.50235 i
1.05263 0.86873 i
1.57895 0.99997 i
.....
9.47368 -0.04889 i
10.00000 -0.54402 i

```

Этот файл можно получить, используя `gnuplot`, и выполняя построчно такой код:

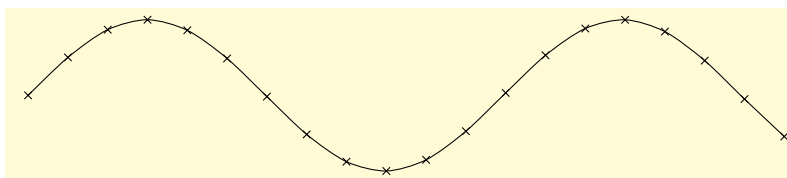
```

set table "pgfmanual-sine.table"
set format "%.5f"
set samples 20
plot [x=0:10] sin(x)

```

Теперь можно построить график функции  $y = \sin x$ :

```
\tikz \draw plot[mark=x,smooth] file {pgfmanual-sine.table};
```



Параметры `<local options>` графической операции `plot` являются локальными для каждого графика и не влияют на другие графики в том же пути. Например, `plot[yshift=1cm]` локально сдвинет график на 1cm вверх. Однако, следует помнить, что большинство опций может применяться только к пути в целом. Например, код `plot[red]` не сделает график красным, поскольку нельзя локально сделать часть пути красным.

## 16.5 График по точкам, вычисляемым на лету

Когда создается график функции, точки данных могут вычисляться, используя математическое выражение. Так как `pgf` содержит механизм математических вычислений, можно определить нужное выражение, а затем заставить `TikZ` автоматически вычислить желаемые координаты.

Так как этот случай весьма распространен, синтаксис такого построения графика функции прост: после команды `plot` и его локальных опций `<local option>`, нужно определить выражение для вычисления точек `<coordinate expression>`. Это похоже на обычную координату, но внутри можно использовать специальный макрос, который по умолчанию есть `\x` (но который можно изменить, используя опцию `variable`). Опция `<coordinate expression>` тогда вычисляется для разных значений `\x` и вычисленные координаты составляют график.

Заметим, что часто нужно будет помещать  $x$ - или  $y$ - координату в фигурные скобки, а именно, всякий раз, когда используется выражение, содержащее круглые скобки.

Следующие опции влияют на то, как вычисляется опция `<coordinate expression>`:

```
/tikz/variable=<macro> (no default, initially x)
```

Определяет макрос, значение которого устанавливается в разные значения, возникающие при вычислении `<coordinate expression>`.

`/tikz/samples=<number>` (no default, initially 25)

Устанавливает число выборок, используемых в графике.

`/tikz/domain=<start>:<end>` (no default, initially -5:5)

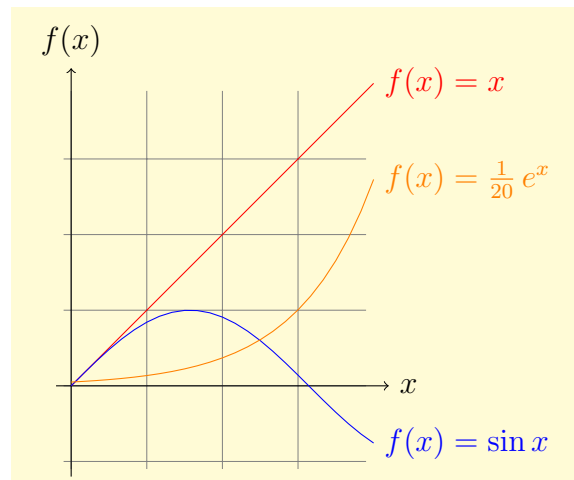
Устанавливает область, из которой должны браться выборки.

`/tikz/samples at=<sample list>` (no default)

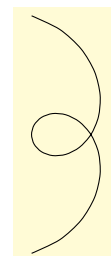
Определяет список позиций, для которых должна вычисляться переменная. Например, можно сказать `samples at={1,2,8,9,10}`, чтобы вычислить переменную для значений 1, 2, 8, 9, 10. Можно использовать синтаксис `\foreach`, так что в `<sample list>` можно использовать операцию (...).

Когда используется опция `samples at`, опции `samples` и `domain` не используются. И, наоборот, установка опций `samples` или `domain` игнорирует опцию `samples at`.

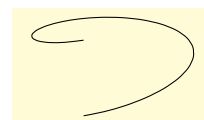
```
\begin{tikzpicture}[domain=0:4]
\draw[very thin,gray]
  -0.1,-1.1) grid (3.9,3.9);
\draw[->] (-0.2,0) -- (4.2,0)
  node[right] {$x$};
\draw[->] (0,-1.2) -- (0,4.2)
  node[above] {$f(x)$};
\draw[red] plot (\x,\x)
  node[right] {$f(x) =x$};
\draw[blue] plot
  (\x,{sin(\x r)})
  node[right]
  {$f(x) = \sin x$};
\draw[orange] plot
  (\x,{0.05*exp(\x)})
  node[right]
  {$f(x) = \frac{1}{20}\, e^x$};
\end{tikzpicture} % r - в радианах
```



```
\tikz \draw[scale=0.5,domain=-3.141:3.141,
  smooth,variable=\t]
  plot ({\t*sin(\t r)},{\t*cos(\t r)});
```



```
\tikz \draw[domain=0:360,smooth,variable=\t]
  plot ({sin(\t)},\t/360,{cos(\t)});
  % отсутствие r - в градусах
```



## 16.6 Построение графика, используя `gnuplot`

Часто требуется составить график из точек, которые задаются через функцию, например,  $f(x) = \sin x$ . К сожалению, сам `TeX` не имеет достаточных вычислительных ресурсов, чтобы генерировать точки для такой функции. Однако, если нужно, `TeX` может вызвать внешнюю программу, которая сможет вычислить необходимые точки. В настоящее время, `TikZ` знает, как вызвать `gnuplot`.

Когда `TikZ` в первый раз сталкивается с кодом `plot[id=<id>] function{x*sin(x)}`, он создает файл с именем `<prefix><id>.gnuplot`, где `<prefix>` — по умолчанию это `\jobname`, то есть имя главного `tex`-файла. Если строится много графиков, лучше использовать разные префиксы. Если `<id>` не задан, то идентификатор будет пуст, что нормально, но лучше, когда каждый график имеет уникальный идентификатор. Идентификатор `<id>` будет частью имени файла, поэтому он не должен содержать ничего особенного, вроде `*` или `$`. Затем, `TikZ` пишет в этот файл некоторый код инициализации, сопровождаемый `plot x*sin(x)`. Код инициализации устанавливает некоторые вещи так, что операция `plot` будет писать координаты в другой файл с именем `<prefix><id>.table`. И, наконец, этот `table`-файл читается, как-будто было написано `plot file{<prefix><id>.table}`. Однако, есть одно несоответствие: `gnuplot` поддерживает некоторый тип поля, сопровождаемого координатами. Если этот тип поля `'u'`, что означает неограниченный, `TikZ` проигнорирует эту координату.

Для работы механизма построения графика должны выполняться два условия:

1. `TeX` должен иметь возможность вызывать внешние программы. Такая возможность по умолчанию в целях безопасности часто отключается. Чтобы предоставить такую возможность используется опция командной строки. Обычно, она называется по подобию `shell-escape` или `enable-write18`.

2. Нужно установить программу `gnuplot` и `TeX` должен найти ее, составляя файл.

К сожалению, эти условия не всегда выполняются. Особенно, если первоисточник передается другому лицу, у которого не установлен `gnuplot`, и потому возникнут проблемы при обработке первоисточника.

Поэтому `TikZ` ведет себя по-разному, когда составляет график: если при достижении кода `plot[id=<id>] function{...}` файл `<prefix><id>.table` уже существует, и если файл `<prefix><id>.gnuplot` содержит нечто, что `TikZ` может рассматривать как корректное содержимое такого файла, `table`-файл немедленно читается, и `TikZ` не пытается вызвать программу `gnuplot`. Этот подход имеет следующие преимущества:

1. Если всю связку необходимых файлов передать другому лицу, он не должен устанавливать `gnuplot`.

2. Если особенность `\write18` отключена в целях безопасности (и это хорошо), то при первой компиляции `tex`-файла, файл `*.gnuplot` все еще будет сгенерирован, а `*.table`-файл нет. Но потом можно запустить программу `gnuplot` вручную для `gnuplot`-файла, чтобы создать необходимый `*.table`-файл.

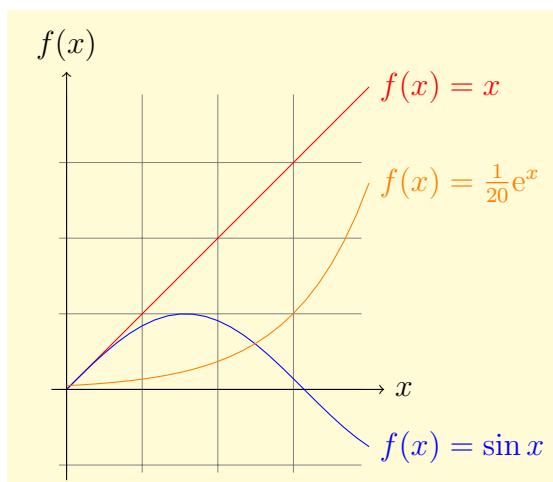
3. Если изменить функцию, график которой нужно составить, или ее область определения, то `TikZ` будет стараться автоматически заново сгенерировать `*.table`-файл.

4. Если не будет задан идентификатор `id`, то один и тот же `*.gnuplot`-файл будет использоваться для создания графиков разных функций, но это не проблема, так как `*.table`-файл будет автоматически восстанавливаться для каждого графика на лету. Но лучше всегда использовать `id`!. Кроме того, наличие уникального идентификатора для каждого графика увеличивает скорость компиляции, поскольку не надо вызывать внешние программы.



Когда есть код `plot function{<gnuplot formula>}`, формулу `{<gnuplot formula>}` нужно задавать в синтаксисе программы `gnuplot`, детали которой следует изучить.

```
\begin{tikzpicture}[domain=0:4]
\draw[very thin,gray] (-0.1,-1.1) grid (3.9,3.9);
\draw[->] (-0.2,0) -- (4.2,0) node[right] {$x$};
\draw[->] (0,-1.2) -- (0,4.2) node[above] {$f(x)$};
\draw[red] plot[id=x] function{x} node[right] {$f(x) = x$};
\draw[blue] plot[id=sin] function{sin(x)}
node[right] {$f(x) = \sin x$};
\draw[orange] plot[id=exp] function{0.05*exp(x)}
node[right] {$f(x) = \frac{1}{20} \mathrm{e}^x$};
\end{tikzpicture}
```

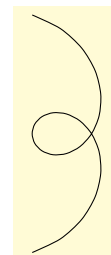


На график функции влияют не только опции `samples` и `domain`, но и следующие опции.

`/tikz/parametric=<boolean>` (default true)

Устанавливает, является ли график параметрически заданным. Если «да» (значение true), вместо параметра  $x$  должен использоваться параметр  $t$  и в `<gnuplot formula>` нужно задавать две функции, разделенные запятой. Например:

```
\tikz \draw[scale=0.5,domain=-3.141:3.141,smooth]
plot[parametric,id=parametric-example]
function{t*sin(t),t*cos(t)};
```



`/tikz/raw gnuplot` (no value)

Заставляет передать параметр `<gnuplot formula>` программе `gnuplot` без настройки выборки или операции `plot`. Таким образом, можно написать

```
plot[raw gnuplot,id=raw-example] function{set samples 25; plot sin(x)}
```

Эта опция полезна в сложных ситуациях, когда нужно передать программе `gnuplot` специальные данные. Однако, для действительно сложных ситуаций следует создавать специальную внешнюю генерацию `gnuplot`-файла и использовать `file`-синтаксис, чтобы включать таблицу вручную.

Следующий стиль также влияет на вид графика:

`/tikz/every plot` (style, initially empty)

Стиль устанавливается для каждого графика, то есть, как-будто было сказано в коде `plot[every plot, ...]`. Для глобальной установки одного и того же префикса `<prefix>` для всех графиков, можно, например, сказать:

```
\tikzset{every plot/.style={prefix=plots/}}
```

## 16.7 Размещение меток на графике

Метки к графику можно добавить, используя опцию `mark`. Когда опция используется, копия графической метки размещается в каждой точке графика. Заметим, что метки размещаются после того, как нарисован/заполнен/растущеван весь путь. Тем, как рисуются метки, управляют следующие опции.

`/tikz/mark=<mark mnemonic>` (no default)

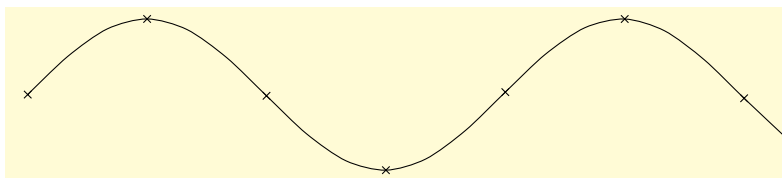
Устанавливает мнемонику для метки, которая ранее была определена, используя `\pgfdeclareplotmark`. По умолчанию используются `*`, `+` и `x`, которые рисуют заполненный круг, плюс, и пересечение в качестве метки. Когда загружена библиотека `plotmarks`, становятся доступными многие другие метки (см. [1, раздел 43.5]).

Одна графическая метка особенная: метка `ball` используется только TikZ. Цвет шара определяет опция `ball color`. Не следует использовать эту опцию с большим числом меток, так как потребуется много времени для их обработки.

`/tikz/mark repeat=<r>` (no default)

Опция просит TikZ рисовать только каждую  $r$ -ую метку.

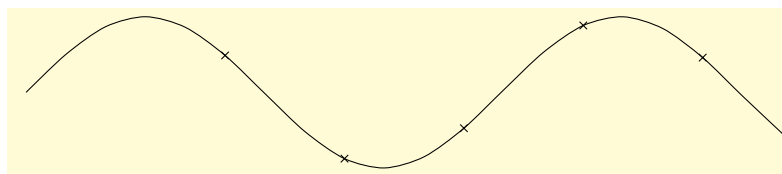
```
\tikz \draw plot[mark=x,mark repeat=3,smooth] file {pgfmanual-sine.table};
```



`/tikz/mark phase=<p>` (no default)

Опция просит TikZ первой нарисовать  $p$ -ую метку, следующей —  $(p+r)$ -ую, затем —  $(p+2r)$ -ую, и так далее.

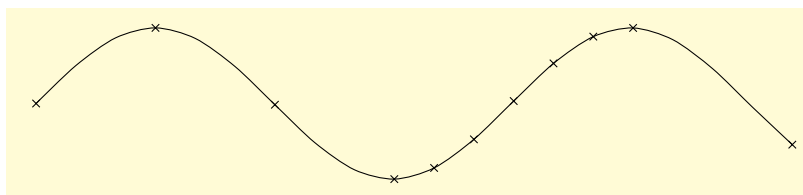
```
\tikz \draw plot[mark=x,mark repeat=3,mark phase=6,smooth]
file {pgfmanual-sine.table};
```



`/tikz/mark indices=<list>` (no default)

Явно определяет индексы точек из списка, в которых должна размещаться метка. Счет начинается с 1. Можно использовать синтаксис `\foreach`, то есть можно использовать операцию `(...)`.

```
\tikz \draw plot[mark=x,mark indices={1,4,...,10,11,12,...,16,20},smooth]
file {pgfmanual-sine.table};
```



`/tikz/mark size=<dimension>` (no default)

Устанавливает размер графических меток. Для круговых меток, `<dimension>` — радиус, для других — `<dimension>` должен приблизительно равняться половине ширины и высоты. Эта опция не так уж и нужна, поскольку тот же результат достигается при определении локальной опции `scale=<factor>`, где `<factor>` — коэффициент для получения желаемого размера от размера по умолчанию. Однако, использование `mark size` и быстрее, и естественнее.

`/tikz/every mark` (style, no value))

Этот стиль устанавливается перед прорисовкой графических меток. Например, в нем можно масштабировать метку или установить ее цвет.

`/tikz/mark options=<options>` (no default)

Переопределяет опцию `every mark` так, что она устанавливает `{<options>}`.

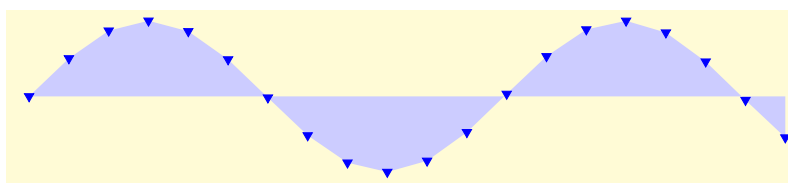
`/tikz/no marks` (style, no value)

Отключает все метки (то же самое, что и `mark=none`).

`/tikz/no markers` (style, no value)

Отключает все метки (то же самое, что и `mark=none`).

```
\tikz \fill[fill=blue!20]
plot[mark=triangle*,mark options={blue,rotate=180}]
file {pgfmanual-sine.table} |- (0,0);
```



## 16.8 Типы графиков

Графическая операция `plot` может работать по-разному с точками, заданными в списке. По умолчанию, она соединяет эти точки прямыми линиями. Однако, существуют опции, меняющие поведение операции `plot`.

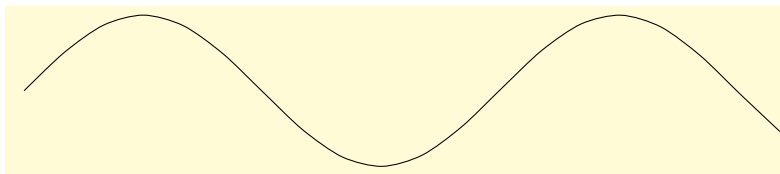
`/tikz/sharp plot` (no value)

Значение по умолчанию: связывает точки прямыми линиями. Эта опция включена только для того, чтобы можно было вернуться назад, если глобально была установлено другое значение, скажем, `smooth`.

`/tikz/smooth` (no value)

Заставляет операцию `plot` связывать точки, используя гладкую кривую.

```
\tikz\draw plot[smooth] file{pgfmanual-sine.table};
```

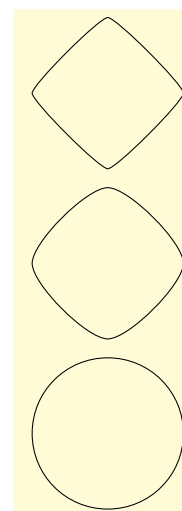


Алгоритм построения гладкой кривой не очень интеллектуален. Лучшие результаты получаются, если угол изгиба брать достаточно малым (меньше  $30^\circ$ ) и, что более важно, расстояния между точками одно и то же на всем промежутке построения графика.

`/tikz/tension=<value>` (no default)

Влияет на степень сглаживания: меньшее значение приведет к более острым углам, а большее значение к более скругленным кривым. Если задать четыре точки на окружности, делящие ее на четыре равные части, то значение 1 даст круг. Значение по умолчанию — 0.55. «Корректное» значение зависит от деталей графика.

```
\begin{tikzpicture}[smooth cycle]
\draw plot[tension=0.2]
  coordinates{(0,0) (1,1) (2,0) (1,-1)};
\draw[yshift=-2.25cm] plot[tension=0.5]
  coordinates{(0,0) (1,1) (2,0) (1,-1)};
\draw[yshift=-4.5cm] plot[tension=1]
  coordinates{(0,0) (1,1) (2,0) (1,-1)};
\end{tikzpicture}
```

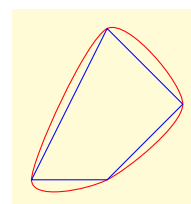


`/tikz/smooth cycle`

(no value)

Связывает точки пути, используя замкнутую гладкую кривую.

```
\begin{tikzpicture}
\draw[red] plot[smooth cycle]
  coordinates{(0,0) (1,0) (2,1) (1,2)};
\draw[blue] plot
  coordinates{(0,0) (1,0) (2,1) (1,2)} -- cycle;
\end{tikzpicture}
```

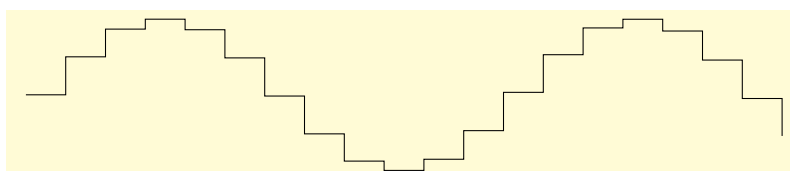


`/tikz/const plot`

(no value)

Связывает точки пути, используя последовательность кусочно-постоянных линий.

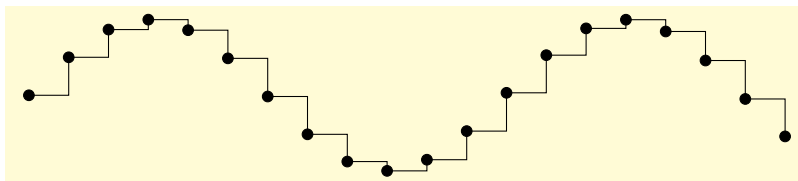
```
\tikz\draw plot[const plot] file{pgfmanual-sine.table};
```



`/tikz/const plot mark left` (no value)

Псевдоним для `const plot`, но если определяется метка, она размещается в левом конце каждой кусочно-постоянной линии.

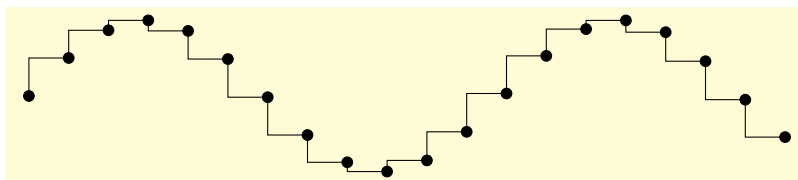
```
\tikz\draw plot[const plot mark left,mark=*] file{pgfmanual-sine.table};
```



`/tikz/const plot mark right` (no value)

Еще один вариант `const plot`, но если определяется метка, она размещается в правом конце каждой кусочно-постоянной линии.

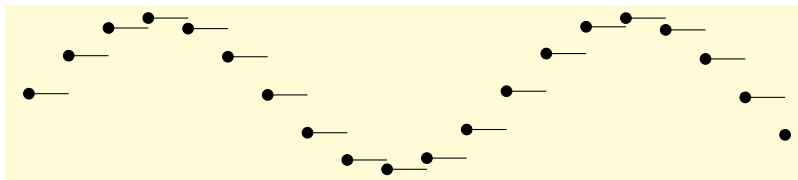
```
\tikz\draw plot[const plot mark right,mark=*] file{pgfmanual-sine.table};
```



`/tikz/jump mark left` (no value)

Связывает точки пути, используя горизонтальные отрезки, и, если метки заданы, они размещаются в левых концах.

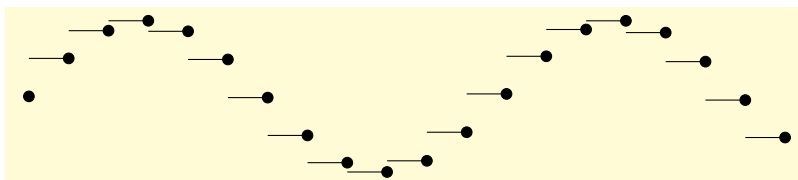
```
\tikz\draw plot[jump mark left, mark=*] file{pgfmanual-sine.table};
```



`/tikz/jump mark right` (no value)

Связывает точки пути, используя горизонтальные отрезки, и, если метки заданы, они размещаются в правых концах.

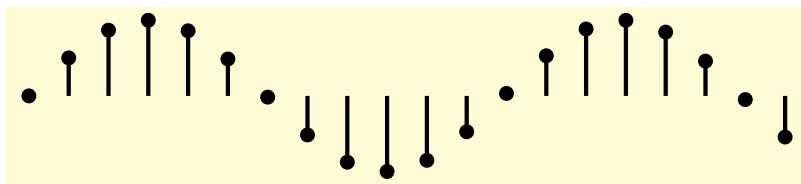
```
\tikz\draw plot[jump mark right, mark=*] file{pgfmanual-sine.table};
```



`/tikz/ycomb` (no value)

Заставляет операцию `plot` по-другому интерпретировать процесс построения графика функции: вместо связывания, для каждой точки графика строится прямая от оси  $OX$  до точки, что дает своеобразную «расческу» или столбчатую диаграмму.

```
\tikz\draw[ultra thick] plot[ycomb,thin,mark=*] file{pgfmanual-sine.table};
```



```
\begin{tikzpicture}[ycomb]
\draw[red,line width=6pt] plot coordinates
  {(0,1)(.5,1.2)(1,.6)(1.5,.7)(2,.9)};
\draw[red!30,line width=4pt,xshift=3pt]
  plot coordinates
  {(0,1.2)(.5,1.3)(1,.5)(1.5,.2)(2,.5)};
\end{tikzpicture}
```

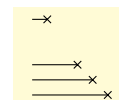


**/tikz/xcomb**

(no value)

Эта опция аналогична опции `ycomb`, но отрезки располагаются горизонтально.

```
\tikz \draw plot[xcomb,mark=x]
  coordinates{(1,0) (0.8,0.2) (0.6,0.4) (0.2,1)};
```

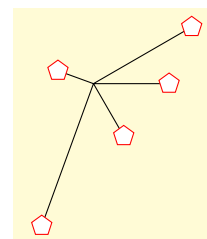


**/tikz/polar comb**

(no value)

Строит прямую линию от начала координат до каждой точки пути из списка.

```
\tikz \draw plot[polar comb,mark=pentagon*,
  mark options={fill=white,draw=red},mark size=4pt]
  coordinates
  {(0:1cm) (30:1.5cm) (160:.5cm) (250:2cm) (-60:.8cm)};
```

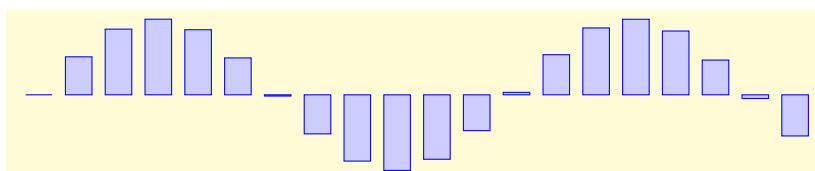


**/tikz/ybar**

(no value)

Порождает столбчатую диаграмму подобно опции `ycomb`, но использует прямоугольники вместо линий. Опция позволяет использовать любой стиль заполнения прямоугольников цветом или шаблоном.

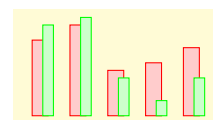
```
\tikz\draw[draw=blue,fill=blue!20] plot[ybar] file{pgfmanual-sine.table};
```



```

\begin{tikzpicture}[ybar]
  \draw[red,fill=red!20,bar width=6pt]
    plot coordinates
      {(0,1) (.5,1.2) (1,.6) (1.5,.7) (2,.9)};
  \draw[green,fill=green!20,bar width=4pt,
    bar shift=3pt] plot coordinates
      {(0,1.2) (.5,1.3) (1,.5) (1.5,.2) (2,.5)};
\end{tikzpicture}

```



Опции `bar width`, `bar shift` устанавливают ширину и сдвиг прямоугольника диаграммы (см. [1, раздел 43.4]).

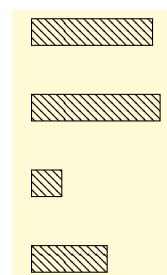
`/tikz/xbar` (no value)

Эта опция аналогична опции `ybar`, но области диаграммы горизонтальны.

```

\tikz \draw[pattern=north west lines] plot[xbar]
  coordinates{(1,0) (0.4,1) (1.7,2) (1.6,3)};

```



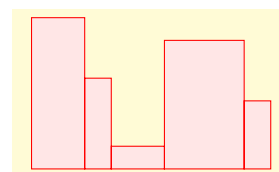
`/tikz/ybar interval` (no value)

Как и `ybar` эта опция строит вертикальные области. Однако, области строятся на интервалах. Так как для  $N + 1$  точки, расположенных на интервале, можно построить  $N$  интервалов  $[x_i; x_{i+1}]$ ,  $i = 1, 2, \dots, N - 1$ , то областей будет всегда на одну меньше, чем точек в списке.

```

\begin{tikzpicture}[ybar interval,x=10pt]
  \draw[red,fill=red!10]
    plot coordinates
      {(0,2) (2,1.2) (3,.3) (5,1.7) (8,.9) (9,.9)};
\end{tikzpicture}

```



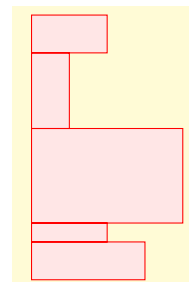
`/tikz/xbar interval` (no value)

Эта опция аналогична опции `ybar interval`, но строит горизонтальные области.

```

\begin{tikzpicture}[xbar interval,x=0.5cm,y=0.5cm]
  \draw[red,fill=red!10]
    plot coordinates
      {(3,0) (2,1) (4,1.5) (1,4) (2,6) (2,7)};
\end{tikzpicture}

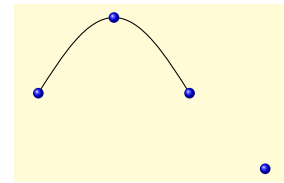
```



`/tikz/only marks` (no value)

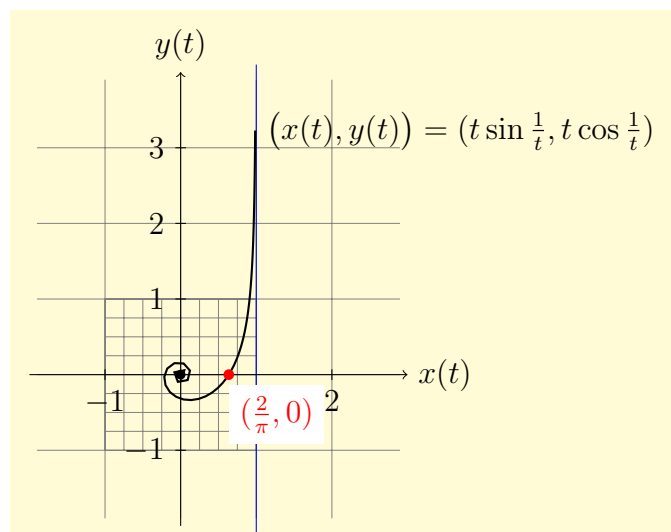
Заставляет рисовать только метки, все другие сегменты пути игнорируются. Опция может оказаться полезной для того, чтобы быстро поставить в пути нужные метки.

```
\tikz \draw (0,0) sin (1,1) cos (2,0)
      plot[only marks,mark=ball]
      coordinates{(0,0)(1,1)(2,0)(3,-1)};
```



И, в завершение темы, красивый пример параметрически заданной функции.

```
\begin{tikzpicture}
\draw[gray,very thin] (-1.9,-1.9) grid (2.9,3.9)
      [step=0.25cm] (-1,-1) grid (1,1);
\draw[blue] (1,-2.1) -- (1,4.1); % asymptote
\draw[->] (-2,0) -- (3,0) node[right] {$x(t)$};
\draw[->] (0,-2) -- (0,4) node[above] {$y(t)$};
\foreach \pos in {-1,2}
  \draw [shift={(\pos,0)}]
    (0pt,2pt) -- (0pt,-2pt) node[below] {$\pos$};
\foreach \pos in {-1,1,2,3}
  \draw [shift={(0,\pos)}]
    (2pt,0pt) -- (-2pt,0pt) node[left] {$\pos$};
\fill (0,0) circle (0.064cm);
\draw [thick,parametric,domain=0.4:1.5,samples=200]
% Реальный график имеет другую параметризацию, чтобы
% сделать проще его вид в окрестности начала координат
plot[id=asymptotic-example]
  function{(t*t*t)*sin(1/(t*t*t)),(t*t*t)*cos(1/(t*t*t))}
  node[right]
  {$\bigl(x(t),y(t)\bigr) = (t\sin \frac{1}{t}, t\cos \frac{1}{t})$};
\fill[red] (0.63662,0) circle (2pt)
  node [below right,fill=white,yshift=-4pt] {$(\frac{2}{\pi},0)$};
\end{tikzpicture}
```





# Глава 17

## Прозрачность

Обычно, когда что-то закрашивается, используя любую из команд TikZ (сглаживание, заполнение, растушевывание, шаблоны или изображения), закрашивание объекта полностью скрывает предметы, ранее нарисованные в этой области. Но такое поведение можно изменить, используя нечто, о чем можно думать, как о прозрачном или полупрозрачном цвете. Такие цвета не делают фон полностью незаметным, скорее они смешивают фон с новым цветом. Хотя использование полупрозрачности цвета выглядит простым и понятным, математика, работающая при ее реализации, далеко не проста.

В главе рассматриваются различные способы определения «прозрачности». Самый простой способ состоит в том, чтобы определить процент прозрачности, например, 60%. Несколько более общий путь состоит в том, чтобы использовать нечто, что называется постепенным изменением цвета или мягким обесцвечиванием (фейдингом — `fading`), также известным как программируемая маска или просто маска (см. [1, глава 33]).

Еще одна задача — создание так называемых групп прозрачности. Эта задача возникает, когда область закрашивается несколько раз полупрозрачным цветом. В этой ситуации иногда желательно добиться эффекта накопления того, что сделано.

Технически прозрачность лучше всего поддерживается pdftex-драйвером. PostScript правильно ее отображает только с новыми версиями Ghostscript. Принтеры и другие программы обычно игнорируют установку непрозрачности.

### 17.1 Определение однородной непрозрачности

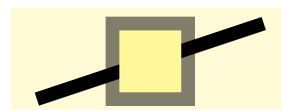
Определить штриховую и/или заполняемую непрозрачность не сложно.

`/tikz/draw opacity=<value>` (no default)

Определяет коэффициент непрозрачности только для *проводимых* линий. Значение 1 означает «полную непрозрачность», значение 0 означает «полную прозрачность» (невидимость). Значение 0.5 означает «полупрозрачность».

Непрозрачность заполнения устанавливает опция `fill opacity`.

```
\begin{tikzpicture}[line width=1ex]
\draw (0,0) -- (3,1);
\filldraw [fill=yellow!50,draw opacity=0.5]
(1,0) rectangle (2,1);
\end{tikzpicture}
```



`/tikz/opacity=<value>` (no default)

Устанавливает коэффициент непрозрачности и для заполнения, и для рисования равным `<value>`, а predefined стили облегчают использование опции `opacity`.

`/tikz/transparent` (style, no value)

Делает все полностью прозрачным и, следовательно, невидимым.

```
\tikz{\fill[red] (0,0) rectangle (1,0.5);
      \fill[transparent,green]
      (0.5,0) rectangle (1.5,0.25);}
```



Следующие стили увеличивают степень непрозрачности от минимальной до полной.

`/tikz/ultra nearly transparent` (style, no value)

```
\tikz{\fill[red] (0,0) rectangle (1,0.5);
      \fill[ultra nearly transparent,green]
      (0.5,0) rectangle (1.5,0.25);}
```



`/tikz/very nearly transparent` (style, no value)

```
\tikz{\fill[red] (0,0) rectangle (1,0.5);
      \fill[very nearly transparent,green]
      (0.5,0) rectangle (1.5,0.25);}
```



`/tikz/nearly transparent` (style, no value)

```
\tikz{\fill[red] (0,0) rectangle (1,0.5);
      \fill[nearly transparent,green]
      (0.5,0) rectangle (1.5,0.25);}
```



`/tikz/semitransparent` (style, no value)

```
\tikz{\fill[red] (0,0) rectangle (1,0.5);
      \fill[semitransparent,green]
      (0.5,0) rectangle (1.5,0.25);}
```



`/tikz/nearly opaque` (style, no value)

```
\tikz{\fill[red] (0,0) rectangle (1,0.5);
      \fill[nearly opaque,green]
      (0.5,0) rectangle (1.5,0.25);}
```



`/tikz/very nearly opaque` (style, no value)

```
\tikz{\fill[red] (0,0) rectangle (1,0.5);
      \fill[very nearly opaque,green]
      (0.5,0) rectangle (1.5,0.25);}
```



`/tikz/ultra nearly opaque`

(style, no value)

```
\tikz{\fill[red] (0,0) rectangle (1,0.5);
       \fill[ultra nearly opaque,green]
         (0.5,0) rectangle (1.5,0.25);}
```

`/tikz/opaque`

(style, no value)

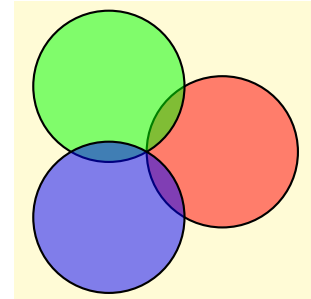
```
\tikz{\fill[red] (0,0) rectangle (1,0.5);
       \fill[opaque,green]
         (0.5,0) rectangle (1.5,0.25);}
```

`/tikz/fill opacity=<value>`

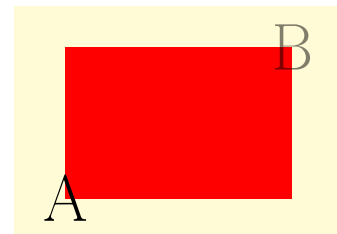
(no default)

Опция устанавливает непрозрачность заполнения. В дополнение к операции заполнения, эта непрозрачность также относится и к тексту, и к изображению.

```
\begin{tikzpicture}
  [thick,fill opacity=0.5]
  \filldraw[fill=red] (0:1cm) circle (10mm);
  \filldraw[fill=green] (120:1cm) circle (10mm);
  \filldraw[fill=blue] (-120:1cm) circle (10mm);
\end{tikzpicture}
```



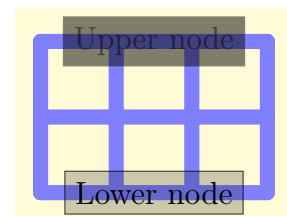
```
\begin{tikzpicture}
  \fill[red] (0,0) rectangle (3,2);
  \node at (0,0) {\huge A};
  \node[fill opacity=0.5] at (3,2) {\huge B};
\end{tikzpicture}
```

`/tikz/text opacity=<value>`

(no default)

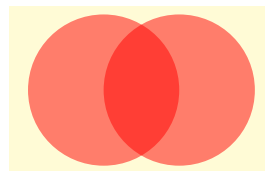
Устанавливает непрозрачность текстовых меток, отменяя установку непрозрачности, заданную опцией `fill opacity`.

```
\begin{tikzpicture}
  [every node/.style={fill,draw}]
  \draw[line width=2mm,blue!50,line cap=round]
    (0,0) grid (3,2);
  \node[opacity=0.5] at (1.5,2) {Upper node};
  \node[draw opacity=0.8,fill opacity=0.2,
        text opacity=1] at (1.5,0) {Lower node};
\end{tikzpicture}
```



Отметим следующий эффект: если установить определенную непрозрачность для штрихования или заполнения, и провести дважды штрихование или заполнение в одной и той же области, то эффект накапливается.

```
\begin{tikzpicture}[fill opacity=0.5]
  \fill[red] (0,0) circle (1);
  \fill[red] (1,0) circle (1);
\end{tikzpicture}
```



Часто, это то, что нужно, но не всегда. Чтобы изменить результат, можно использовать группы прозрачности (см. конец этого раздела).

## 17.2 Постепенное изменение цвета (fading)

Для сложных рисунков, однородные установки прозрачности не всегда достаточны. Предположим, что в процессе раскраски рисунка, нужно, чтобы прозрачность гладко изменялась от полностью непрозрачной до полностью прозрачной. Такую форму прозрачности Тилл Тангау называет федингом (fading) (по-русски, постепенным изменением цвета или затуханием цвета).

### 17.2.1 Создание фединга

Фединг определяет для каждой точки меру прозрачности, измеряемую числом между 0 и 1. Рисунок с федингом — нормальный рисунок, в котором определяется прозрачность его точек. Такой рисунок создается обычными командами рисования. Прозрачность точки определяется светимостью затухающего рисунка в этой точке. Чем больше светимость точки при фединге, тем более непрозрачной является точка. В частности, белая точка в таком рисунке полностью непрозрачна, а черная точка полностью прозрачна. (Фон в рисунке с федингом всегда прозрачен, как будто фон черный.) Такая характеристика противоречит интуиции. Поэтому TikZ определяет полностью прозрачный цвет термином `transparent`, который указывает на черный цвет. Определение `transparent!<percentage>` в рисунке с федингом создает пиксель, который является на `<percentage>%` прозрачным.

Преобразование нормального рисунка в рисунок с федингом достигается с помощью команд, определенных в библиотеке `fadings`, которую следует загрузить. Для работы с рисунками, использующими фединг, библиотека вводит новое окружение

```
\begin{tikzfadingfrompicture}[<options>]
  <environment contents>
\end{tikzfadingfrompicture}
```

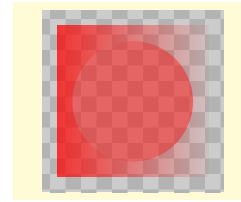
Окружение работает аналогично окружению `{tikzpicture}`, но рисунок не показывается, а на его основе определяется рисунок с федингом. Имя рисунка с федингом определяется, используя опцию `name=` (которая используется при установке имени узла). Но в окружении `{tikzfadingfrompicture}` опция `name=` должна использоваться *обязательно*.

В следующем рисунке 2cm x 2cm, прозрачность меняется слева направо, и 50% прозрачности достигается для большого круга в его середине.

```

\begin{tikzfadingfrompicture}[name=fade right]
  \shade[left color=transparent!20,
    right color=transparent!100]
    (0,0) rectangle (2,2);
\fill[transparent!50] (1,1) circle (0.7);
\end{tikzfadingfrompicture}
% Теперь используем фединг в другом рисунке
\begin{tikzpicture}
  % Создаем фон
  \fill [black!20] (-1.2,-1.2) rectangle (1.2,1.2);
  \pattern [pattern=checkerboard, pattern color=black!30]
    (-1.2,-1.2) rectangle (1.2,1.2);
  % Используем рисунок с федингом
  \fill [path fading=fade right,red] (-1,-1) rectangle (1,1);
\end{tikzpicture}

```



В следующем примере постепенно меняется цвет текста.

```

\begin{tikzfadingfrompicture}[name=tikz]
  \node [text=transparent!20]
    {\fontfamily{cmr}\fontsize{45}{45}
    \bfseries\selectfont Ti\emph{k}Z};
\end{tikzfadingfrompicture}
\begin{tikzpicture}
\fill [black!20] (-2,-1) rectangle (2,1);
\pattern[pattern=checkerboard, pattern color=black!30]
  (-2,-1) rectangle (2,1);
% Используем рисунок с федингом
\shade[path fading=tikz,fit fading=false,
  left color=blue!30,right color=black]
  (-2,-1) rectangle (2,1);
\end{tikzpicture}

```



### `\tikzfading`[<options>]

Команда используется в определении фединга подобно тому, как определяется растушевывание. В <options> следует

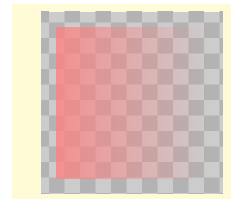
- 1) использовать опцию `name=<name>`, определяя имя для фединга,
- 2) использовать опцию `shading`, устанавливая имя механизма растушевывания, который желательно использовать,
- 3) использовать дополнительные опции для установки цвета растушевывания (обычно они устанавливаются посредством цвета `transparent!<percentage>`).

Тогда, новый рисунок с федингом с именем <name> будет создаваться, основываясь на указанном растушевывании.

```

\tikzfading[name=fade right1,
             left color=transparent!20,
             right color=transparent!100]
% Теперь используем фединг в другом рисунке
\begin{tikzpicture} % Фон
  \fill [black!20] (-1.2,-1.2) rectangle (1.2,1.2);
  \path [pattern=checkerboard, pattern color=black!30]
        (-1.2,-1.2) rectangle (1.2,1.2);
% Используем рисунок с федингом
  \fill [red,path fading=fade right1] (-1,-1) rectangle (1,1);
\end{tikzpicture}

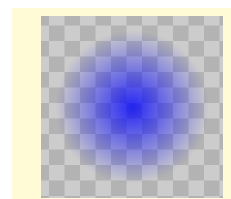
```



```

\tikzfading[name=fade out,inner color=transparent!20,
             outer color=transparent!100]
% Теперь используем фединг в другом рисунке
\begin{tikzpicture}% Фон
  \fill [black!20] (-1.2,-1.2) rectangle (1.2,1.2);
  \path [pattern=checkerboard, pattern color=black!30]
        (-1.2,-1.2) rectangle (1.2,1.2);
  \fill [blue,path fading=fade out] (-1,-1) rectangle (1,1);
\end{tikzpicture}

```



### 17.2.2 Фединг в пути

Следующие опции используются для того, чтобы установить постепенное изменение цвета в текущей области видимости или в пути.

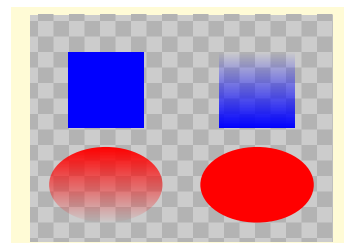
`/tikz/path fading=<name>` (default scope's setting)

Оповещает TikZ, что в текущем пути должен постепенно меняться цвет и этот фединг получает имя `<name>`. Если имя не задано, используется имя текущей области видимости. Подобно опциям `draw` или `fill`, опция сбрасывается после завершения пути, таким образом, ее нужно определять в каждом пути, в котором используется фединг. Можно определить `name=none`, тогда постепенное изменение в пути будет выключено в том случае, если оно было включено предыдущими опциями или стилями.

```

\begin{tikzpicture}[path fading=south]
\fill [black!20] (0,0) rectangle (4,3);
\pattern [pattern=checkerboard,
          pattern color=black!30]
          (0,0) rectangle (4,3);
\fill[blue] (0.5,1.5) rectangle +(1,1);
\fill[blue,path fading=north]
          (2.5,1.5) rectangle +(1,1);
\fill[red,path fading] (1,0.75) ellipse (.75 and .5);
\fill[red]             (3,0.75) ellipse (.75 and .5);
\end{tikzpicture}

```



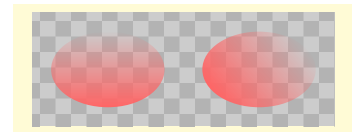
`/tikz/fit fading=<boolean>` (default `true`, initially `true`)

Когда опция принимает значение `true` (истина), фединг сдвигается и изменяет размеры (точно тем же, как и при растушевывании), покрывая текущий путь. Когда опция принимает значение `false` (ложь), фединг только сдвигается, устанавливая свой центр в центр пути, но не изменяет размеры. Это может быть полезно при специальном фединге, например, когда он используется, чтобы «взорвать» что-либо.

`/tikz/fading transform=<transformation options>` (no default)

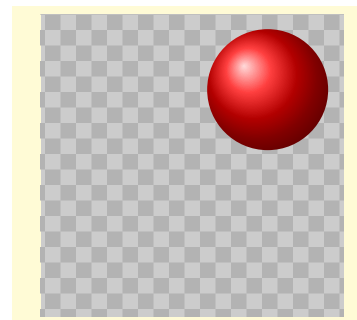
Параметр `<transformation options>` применяется к федингу до его использования. Например, если параметр `<transformation options>` равен `rotate=90`, фединг вращается на  $90^\circ$ .

```
\begin{tikzpicture}[path fading=east]
\fill [black!20] (0,0) rectangle (4,1.5);
\path [pattern=checkerboard,
       pattern color=black!30]
       (0,0) rectangle (4,1.5);
\fill [red!60,path fading,
       fading transform={rotate=90}]
       (1,0.75) ellipse (.75 and .5);
\fill [red!60,path fading,
       fading transform={rotate=30}]
       (3,0.75) ellipse (.75 and .5);
\end{tikzpicture}
```



Постепенное изменение цвета с именем `fade inside` в следующем примере, делает более прозрачным середину, чем внешний край.

```
\tikzfading[name=fade inside,
            inner color=transparent!80,
            outer color=transparent!30]
\begin{tikzpicture}
%Фон: шахматная доска
\fill [black!20] (0,0) rectangle (4,4);
\path [pattern=checkerboard,
       pattern color=black!30]
       (0,0) rectangle (4,4);
\shade[ball color=red](3,3) circle (0.8);
\shade[ball color=white,
       path fading=fade inside]
       (2,2) circle (1.8);
\end{tikzpicture}
```



`/tikz/fading angle=<degree>` (no default)

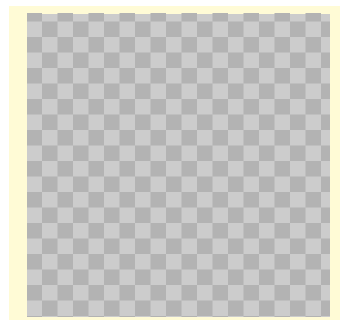
Сокращение для `fading transform={rotate=<degree>}`.

Можно постепенно обесцвечивать все что угодно, в частности, растушевывание.

```

\begin{tikzpicture}
\fill [black!20] (0,0) rectangle (4,4);
\path [pattern=checkerboard,
       pattern color=black!30
       (0,0) rectangle (4,4);
\shade [ball color=blue,path fading=south]
       (2,2) circle (1.8);
\end{tikzpicture}

```



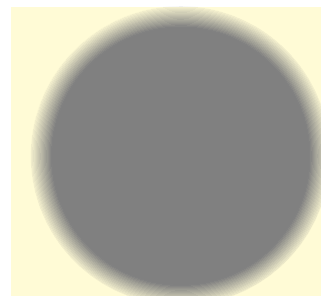
Отметим, что добавление в узел опции `path fading` постепенно изменяет цвет (фона) пути, но не самого текста. Чтобы постепенно изменять цвет текста, нужно использовать фединг в области видимости (см. далее).

Использование фединга совместно с шаблонами может дать весьма интересные визуальные эффекты.

```

\tikzfading[name=middle,
            top color=transparent!50,
            bottom color=transparent!50,
            middle color=transparent!20]
\begin{tikzpicture}
\node [circle,circular drop shadow,
      pattern=horizontal lines dark blue,
      path fading=south,minimum size=3.6cm] {};
\pattern [path fading=north,
         pattern=horizontal lines dark gray]
         (0,0) circle (1.8cm);
\pattern [path fading=middle,
         pattern=crosshatch dots light steel blue]
         (0,0) circle (1.8cm);
\end{tikzpicture}

```



### 17.2.3 Фединг в области видимости

В дополнение к постепенному изменению цвета конкретного пути, можно применить фединг ко всей области видимости, то есть, использовать фединг глобально, определяя прозрачность для всех объектов, нарисованных в области видимости. Для объединения возможностей фединга надо будет использовать опцию `transparency group` (см. конец этого раздела).

`/tikz/scope fading=<fading>` (no default)

В принципе, эта опция работает точно так же, как и опция `path fading`. Единственное отличие то, что постепенное изменение цвета происходит до выхода из области видимости. В этом отношении, опция работает как опция `clip` (заметим, однако, что, в отличие от опции `clip`, возможности фединга не накапливаются, если не используется опция `transparency group`).

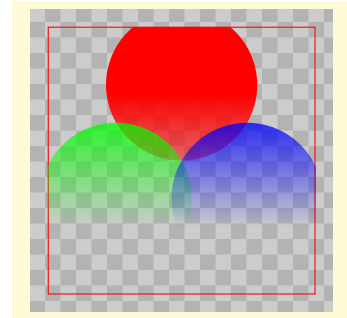
Опции `fit fading` и `fading transform` дают тот же результат, что и `path fading`. Также как и опция `path fading`, предоставляя опцию `scope fading` с `{scope}`, они



только устанавливают имя фединга, которое затем и используется. Следует явно указать опцию `scope fading` в пути, чтобы фактически установить фединг.

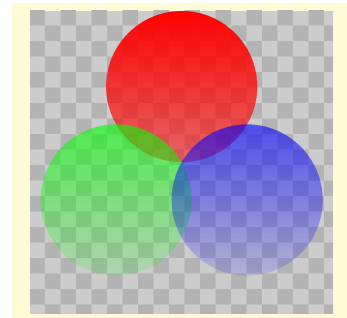
```
\begin{tikzpicture}
\fill [black!20] (-2,-2) rectangle (2,2);
\pattern [pattern=checkerboard,
          pattern color=black!30]
(-2,-2) rectangle (2,2);
% ограничивающий прямоугольник растушевывания
\draw [red]
(-50bp,-50bp) rectangle (50bp,50bp);
\path[scope fading=south,fit fading=false]
(0,0);

% фединг идет из центра
\fill[red] (90:1) circle (1);
\fill[green] (210:1) circle (1);
\fill[blue] (330:1) circle (1);
\end{tikzpicture}
```



В следующем примере размеры фединга изменяются до размеров целого рисунка:

```
\begin{tikzpicture}
\fill [black!20] (-2,-2) rectangle (2,2);
\pattern [pattern=checkerboard,
          pattern color=black!30]
(-2,-2) rectangle (2,2);
\path [scope fading=south]
(-2,-2) rectangle (2,2);
\fill[red] (90:1) circle (1);
\fill[green] (210:1) circle (1);
\fill[blue] (330:1) circle (1);
\end{tikzpicture}
```

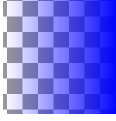

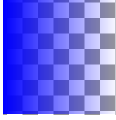
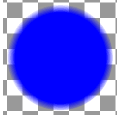
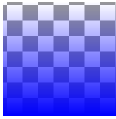
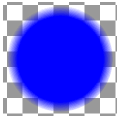
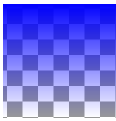
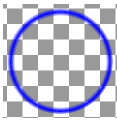


Постепенное изменение цвета в области видимости нужно и тогда, когда желательно постепенно изменить цвет узла.

```
\tikz
\node[scope fading=south,
      fading angle=45,
      text width=45mm]
{Текст будет постепенно исчезать
при перемещении вправо и вниз.
Очень трудно получить такой
результат другими средствами.};
```

Текст будет постепенно исчезать при перемещении вправо и вниз. Очень трудно получить такой результат другими средствами.

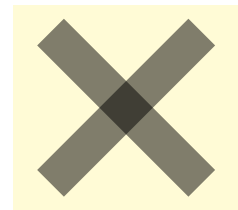
В заключение раздела приведем определенные в библиотеке `fadings` стандартные фединги, некоторые из которых неоднократно использовались в примерах раздела 17.2.

Имя фединга	Пример	Имя фединга	Пример
west		circle with fuzzy edge 10 percent	
east		circle with fuzzy edge 15 percent	
north		circle with fuzzy edge 20 percent	
south		fuzzy ring 15 percent	

### 17.3 Группы прозрачности

Посмотрим на представленные ниже рисунки со знаком зачеркивания и запрещающим знаком «Не курить». Они выглядят плохо, поскольку знак зачеркивания неоднороден, а вторая запрещающая метка прозрачна, кроме того видно, как они сконструированы.

```
\begin{tikzpicture}[opacity=.5]
\draw [line width=5mm] (0,0) -- (2,2);
\draw [line width=5mm] (2,0) -- (0,2);
\end{tikzpicture}
```

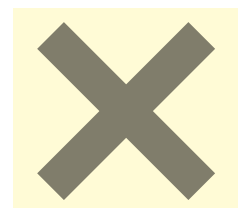


```
\begin{tikzpicture}
\node at (0,0)[forbidden sign,draw=red,
line width=2ex,fill=white] {Smoking};
\node [circle,opacity=.5] at (2,0)
[forbidden sign,line width=2ex,
draw=red,fill=white] {Smoking};
\end{tikzpicture}
```



Используя группы прозрачности, можно все исправить:

```
\begin{tikzpicture}[opacity=.5]
\begin{scope}[transparency group]
\draw [line width=5mm] (0,0) -- (2,2);
\draw [line width=5mm] (2,0) -- (0,2);
\end{scope}
\end{tikzpicture}
```



```

\begin{tikzpicture}
\node at (0,0)
  [forbidden sign,draw=red,
   line width=2ex,fill=white] {Smoking};
\begin{scope}
  [opacity=.5,transparency group]
  \node at (2,0)[circle,forbidden sign,
  line width=2ex,draw=red,fill=white]{Smoking};
\end{scope}
\end{tikzpicture}

```



`/tikz/transparency group` (no value)

Опцию можно задавать для окружения `scope`. Она даст следующий результат: содержимое окружения `scope` штрихуется/заполняется, игнорируя любую внешнюю прозрачность, то есть все предыдущие параметры настройки прозрачности игнорируются. Затем, когда окружение `scope` завершается, оно окрашивается как целое и все параметры настройки прозрачности теперь применяются к полученному рисунку.

Заметим, что, есть некоторые особенности, зависящие от используемого драйвера (детали работы механизма прозрачности см. в [1, 84]).

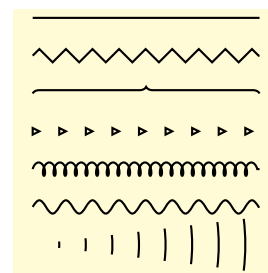
# Глава 18

## Декорирование пути

### 18.1 Основы декорирования

Декорация — общий механизм, позволяющий интереснее (в смысле оформления) представить путь. Прежде, чем рассматривать детали, приведем некоторые примеры:

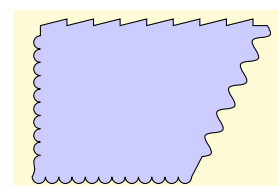
```
\begin{tikzpicture}[thick]
\draw (0,3) -- (3,3);
\draw[decorate,decoration=zigzag] (0,2.5) -- (3,2.5);
\draw[decorate,decoration=brace] (0,2) -- (3,2);
\draw[decorate,decoration=triangles] (0,1.5) -- (3,1.5);
\draw[decorate,decoration={coil,segment length=4pt}]
(0,1) -- (3,1);
\draw[decorate,decoration={coil,aspect=0}]
(0,.5) -- (3,.5);
\draw[decorate,decoration={expanding waves,angle=7}]
(0,0) -- (3,0);
\end{tikzpicture}
```



```
\begin{tikzpicture}
\node [fill=red!20,draw,decorate,decoration={bumps,mirror},
minimum height=1cm] {Бум!};
\end{tikzpicture}
```



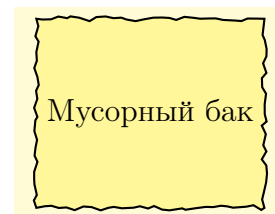
```
\begin{tikzpicture}
\filldraw[fill=blue!20] (0,3)
decorate [decoration=saw] { -- (3,3)}
decorate [decoration={coil,aspect=0}] { -- (2,1)}
decorate [decoration=bumps] { -| (0,3)};
\end{tikzpicture}
```



```

\begin{tikzpicture}
  \node [fill=yellow!50,draw,thick,minimum size=2.5cm,
        decorate,decoration={random steps,
                              segment length=3pt,amplitude=1pt}]
        {Мусорный бак};
\end{tikzpicture}

```



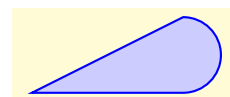
Общая идея декорирования сводится к следующим этапам. Сначала создать путь, используя обычные команды конструирования пути (результат такого построения, как правило, последовательность прямых и кривых). Вместо прямого использования этого пути для заполнения или рисования, определить, что он должен сформировать основу для декорирования. В этом случае, в зависимости от используемой декорации, создается новый путь вдоль ранее определенного пути. Например, с декорацией `zigzag`, новый путь — зигзагообразная линия, которая располагается вдоль старого пути.

Рассмотрим пример: на первом рисунке, располагается путь, который состоит из отрезка, дуги и еще одного отрезка. На втором рисунке, этот путь используется как основа для декорации.

```

\tikz \fill [fill=blue!20,draw=blue,thick]
  (0,0) -- (2,1) arc (90:-90:.5) -- cycle;

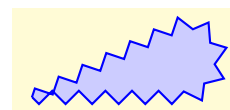
```



```

\tikz \fill [decorate,decoration={zigzag}]
  [fill=blue!20,draw=blue,thick]
  (0,0) -- (2,1) arc (90:-90:.5) -- cycle;

```

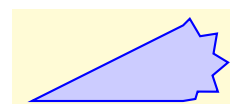


Можно декорировать только часть пути (подпуть) (точный синтаксис будет приведен в этом разделе позже).

```

\tikz \fill [decoration={zigzag}]
  [fill=blue!20,draw=blue,thick]
  (0,0) -- (2,1)
  decorate { arc (90:-90:.5) } -- cycle;

```



Декорацию `zigzag` относится к декорациям трансформации пути, поскольку она только преобразует исходный путь в путь ему топологически эквивалентный. Не все декорации являются трансформациями. Существуют три вида декораций.

1. Декорации трансформации непрерывно меняют путь, не меняя при этом числа его подпутьей. Примеры таких декораций: `zigzag` и `snake`. Много таких декораций определено в библиотеке `decorations.pathmorphing`.

2. Декорации замены пути, полностью заменяющие первоначальный путь другим путем, который только «похож» на первоначальный. Например, декорация `crosses` заменяет декорируемый путь путем, состоящим из последовательности меток пересечений (значков вида `x`). Заметим, что в следующем примере операция заполнения пути не дала бы результата, так как путь состоит из многочисленных изолированных подпутьей (символов `x`) прямой линии.

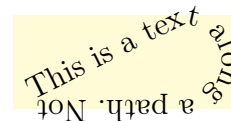
```
\tikz \fill [decorate,decoration={crosses}]
[fill=blue!20,draw=blue,thick]
(0,0) -- (2,1) arc (90:-90:.5) -- cycle;
```



Примерами декораций замены пути являются декорации `shape backgrounds`, `ticks`, `crosses`. Такие декорации определены не только в библиотеке `decorations.pathreplacing`, но и в библиотеке `decorations.shapes`.

3. Декорации удаления пути, которые полностью удаляют декорируемый путь и, обычно, дают множество побочных эффектов. Например, могут написать некоторый текст по (удаленному) пути, или разместить узлы вдоль этого пути.

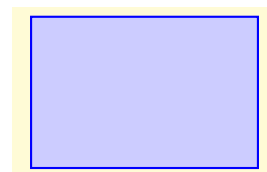
```
\tikz \fill [decorate,decoration={text along path,
text=This is a text along a path.
Note how the path is lost.}]
[fill=blue!20,draw=blue,thick]
(0,0) -- (2,1) arc (90:-90:.5) -- cycle;
```



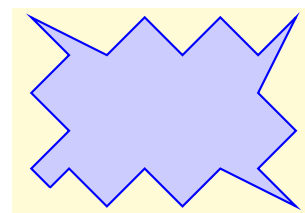
Декорации удаления пути определены в нескольких библиотеках (см. [1, глава 30]). Можно определять и собственные декорации (см. [1, глава 72]), для этого надо использовать основной уровень `pgf` и привлечь немного теории.

Декорации могут использоваться для повторного декорирования уже декорированного пути. В следующих трех рисунках, исходный путь подвергается двум последовательным декорациям.

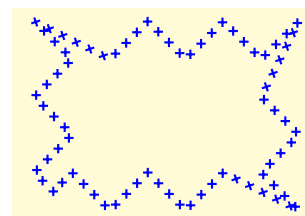
```
\tikz \fill [fill=blue!20,draw=blue,thick]
(0,0) rectangle (3,2);
```



```
\tikz \fill [fill=blue!20,draw=blue,thick]
decorate[decoration={zigzag,
segment length=10mm,amplitude=2.5mm}]
{(0,0) rectangle (3,2) };
```



```
\tikz \fill [fill=blue!20,draw=blue,thick]
decorate[decoration={crosses,
segment length=2mm}] {
decorate[decoration={zigzag,
segment length=10mm,amplitude=2.5mm}] {
(0,0) rectangle (3,2)}};
```



Отметим, что декорирование — медленная операция, и может оказаться немного неточной. Причина в том, что `pgf` должен выполнить большой объем сложных вычислений, а `TeX` не лучшая среда для решения математических задач.

Чтобы использовать декорации, следует загрузить библиотеку `decoration`, определяющую основные опции. Специальные декорации определяются другими библиотеками, например, указанными выше, включающими библиотеку `decoration` автоматически.

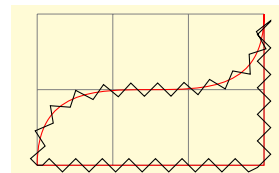
## 18.2 Декорирование, использующее команду `decorate`

Самый общий способ декорировать путь (или подпуть) — следующая команда.

```
\path ... decorate [<options>]{<subpath>}...
```

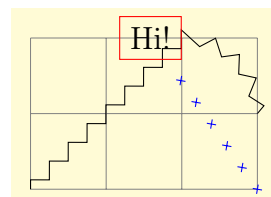
Команда декорирует подпуть `<subpath>`, используя текущую декорацию. В зависимости от декорация, команда может расширять или не расширять текущий путь.

```
\begin{tikzpicture}
\draw [help lines] grid (3,2);
\draw [red] (0,0) .. controls (0,2)
and (3,0) .. (3,2) |- (0,0);
\draw decorate [decoration={name=zigzag}]
{(0,0) .. controls (0,2)
and (3,0) .. (3,2) |- (0,0)};
\end{tikzpicture}
```



Путь может включать в себя прямые линии, кривые, прямоугольники, дуги, круги, эллипсы, и даже уже декорированные пути (см. ниже). Иногда могут возникать неточности при позиционировании путей и в определении точек их пересечения. В параметре `<subpath>` операции `decorate` можно использовать узлы.

```
\begin{tikzpicture}
\draw [help lines] grid (3,2);
\draw decorate [decoration={name=zigzag}]
{(0,0) -- (2,2) node (hi)
[left,draw=red] {Hi!} arc(90:0:1)};
\draw [blue] decorate [decoration={crosses}]{(3,0) -- (hi)};
\end{tikzpicture}
```

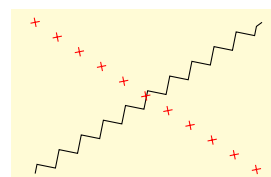


Следующий ключ используется для выбора декорации, а также для выбора последующих опций визуализации выбранной декорации.

```
/pgf/decoration=<decoration options> (no default)
```

Используется только для определения желаемой декорации и ее вида, но не применяет декорацию к пути. Для последней задачи требуется команда пути `decorate` или опция пути `decorate`. Опцию `decoration` можно использовать в начале рисунка или области видимости, определяя ту декорацию, которая будет использоваться с каждой командой пути `decorate`. Естественно, любые локальные опции команды пути `decorate` переопределяют глобальные опции.

```
\begin{tikzpicture}[decoration=zigzag]
\draw decorate {(0,0) -- (3,2)};
\draw [red] decorate [decoration=crosses]
{(0,2) -- (3,0)};
\end{tikzpicture}
```



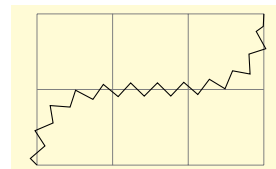
Параметр `<decoration options>` содержит опции, определяющие свойства декорации. Какие опции соответствуют конкретной декорации, зависит только от декорации, так что соответствующие опции нужно искать в документации (см. [1, глава 30]).

Есть одна особая опция, доступная только в TikZ:

`/pgf/decoration/name=<name>` (no default, initially none)

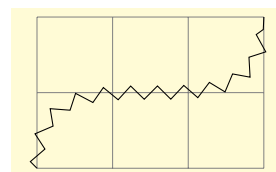
Опция устанавливает, какая декорация будет использоваться. Параметр `<name>` может быть именем декорации или мета-декорации (различия важны для желающих писать собственные декорации). Если `name=none`, то декорации не используются.

```
\begin{tikzpicture}
\draw [help lines] grid (3,2);
\draw decorate [decoration={name=zigzag}]
  {(0,0) .. controls (0,2) and (3,0) .. (3,2)};
\end{tikzpicture}
```



Так как эта опция используется очень часто, можно опустить часть `name=`, и, таким образом, пример, приведенный выше, можно записать короче:

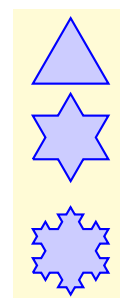
```
\begin{tikzpicture}
\draw [help lines] grid (3,2);
\draw decorate [decoration=zigzag]
  {(0,0) .. controls (0,2) and (3,0) .. (3,2)};
\end{tikzpicture}
```



Есть множество опций, которые позволяют скорректировать позицию декорации относительно декорируемого пути (детали см. ниже в разделе 18.4).

Допустимо вкладывать команды `decorate`. В этом случае, путь, возникающий как результат предыдущей декорации используется как декорируемый путь для последующей. Эту возможность используют при рисовании фракталов. Рассмотрим это на примере снежинки Коха: многократное применение декорации `Koch snowflake` к начальному треугольнику приводит к фракталу, который напоминает снежинку.

```
\begin{tikzpicture}
[decoration=Koch snowflake,draw=blue,fill=blue!20,thick]
\filldraw (0,0) -- ++(60:1) % треугольник
  -- ++(-60:1) -- cycle ;
\filldraw decorate{(0,-1) -- ++(60:1) % звезда
  -- ++(-60:1) -- cycle };
\filldraw decorate{ decorate{(0,-2.5) % снежинка
  -- ++(60:1) -- ++(-60:1) -- cycle }};
\end{tikzpicture}
```



### 18.3 Декорирование пути в целом

Можно декорировать путь, создание которого происходило не явно. Например, путь фона узла создается так, что нельзя использовать команду `decorate`. В этом случае можно использовать опцию, позволяющую декорировать путь «post factum».

`/tikz/decorate=<boolean>` (default true)

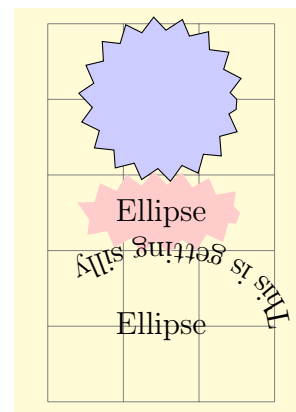


Когда опция установлена (`true`), целый путь декорируется после его создания. Декорация, используемая для декорирования пути, устанавливается через `decoration` так же, как и для команды `decorate`. Следующие две команды дают одинаковый результат:

```
\path decorate [<options>]{<path>}; \path [decorate, <options>] <path>;
```

Опцию `decorate` можно также использовать с узлами, вызывая декорирование фонового пути в узле. Но так декорировать фоновый путь узла можно только один раз, то есть, в отличие от команды пути `decorate`, эту опцию нельзя применить дважды (что просто установило бы значение `true` еще раз).

```
\begin{tikzpicture}[decoration=zigzag]
\draw [help lines] (0,0) grid (3,5);
\draw [fill=blue!20,decorate]
      (1.5,4) circle (1cm);
\node at (1.5,2.5)
      [fill=red!20,decorate,ellipse] {Ellipse};
\node at (1.5,1) [inner sep=6mm,fill=red!20,
      decorate,shape=ellipse,decoration=
      {text along path,text=
      {This is getting silly}}]{Ellipse};
\end{tikzpicture}
```



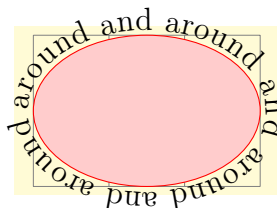
В последнем примере, декорация `text along path` удаляет декорируемый путь. В таких случаях полезно использовать пред- или пост- действие, чтобы вызвать декорацию, которая будет применяться только до или после того, как был использован основной путь. Вот другое приложение опции `decorate`, результат которой нельзя получить, используя команду пути `decorate`.

```
\begin{tikzpicture}[decoration=zigzag]
\node at (1.5,1)[inner sep=6mm,fill=red!20,
  ellipse,postaction={decorate,decoration=
  {text along path,text={This is getting silly}}}]
  {Ellipse};
\end{tikzpicture}
```



Приведем более полезный пример, где используется пост-действие, добавляющее путь после того, как нарисован основной путь.

```
\catcode'\|12
\begin{tikzpicture}
\draw [help lines] grid (3,2);
\fill [draw=red,fill=red!20,
  postaction={decorate,decoration={
  raise=2pt,text along path, text=
  around and around and around and around we go}}]
  (0,1) arc (180:-180:1.5cm and 1cm);
\end{tikzpicture}
```



## 18.4 Корректировка декораций

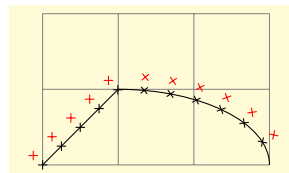
### Позиционирование декораций относительно декорируемого пути

Следующая опция, доступная только в TikZ, позволяет изменить позиционирование декораций относительно декорируемого пути.

`/pgf/decoration/raise=<dimension>` (no default, initially 0pt)

Сегменты декорации поднимаются на `<dimension>` «вверх» относительно декорируемого пути («влево», если двигаться вдоль пути). Подъем происходит после и в дополнение ко всем другим преобразованиям, установленным, используя опцию `transform` (см. ниже). Отрицательная величина `<dimension>` будет сдвигать декорацию «вниз» («вправо», если двигаться вдоль пути).

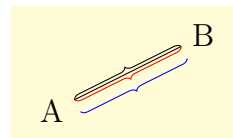
```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (3,2);
\draw (0,0) -- (1,1) arc (90:0:2 and 1);
\draw decorate [decoration=crosses]
  {(0,0) -- (1,1) arc (90:0:2 and 1)};
\draw[red] decorate
  [decoration={crosses,raise=5pt}]{(0,0) -- (1,1) arc (90:0:2 and 1)};
\end{tikzpicture}
```



`/pgf/decoration/mirror=<boolean >` (no default)

Заставляет сегменты декорации зеркально отражаться от декорируемого пути. Это происходит после и в дополнение ко всем другим преобразованиям, установленным, используя опцию `transform` и/или `raise`.

```
\begin{tikzpicture}
\node (a) {A}; \node (b) at (2,1) {B};
\draw (a) -- (b);
\draw[decorate,decoration=brace] (a) -- (b);
\draw[decorate,decoration={brace,mirror},red] (a) -- (b);
\draw[decorate,decoration={brace,mirror,raise=5pt},blue] (a) -- (b);
\end{tikzpicture}
```



`/pgf/decoration/transform=<transformations>` (no default)

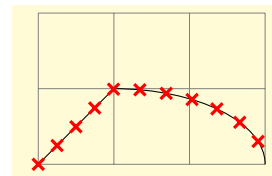
Позволяет определить общие преобразования `<transformations>`, которые будут применяться к сегментам декорации до и независимо от преобразований `raise` и `mirror`. Преобразования в параметре `<transformations>` должны быть обычными преобразованиями TikZ, такими как `shift` и `rotate`.

В следующем примере используется преобразование `shift only`, гарантирующее что пересечение (символ x) не будут наклонены при установке вдоль пути.

```

\begin{tikzpicture}
\draw [help lines] (0,0) grid (3,2);
\draw (0,0) -- (1,1) arc (90:0:2 and 1);
\draw[red,very thick] decorate
    [decoration={crosses,transform={shift only},
                shape size=1.5mm}]
    {(0,0) -- (1,1) arc (90:0:2 and 1)};
\end{tikzpicture}

```



## Начальные и конечные декорации

Иногда нужно остановить декорацию (например, `snake`) в 5mm до конца пути и затем продолжить ее прямой линией. Есть разные способы получить такой результат, но самый простой — использовать опции `pre` (перед) и `post` (пост), которые применимы только в TikZ (и только с декорациями, но не с мета-декорациями).

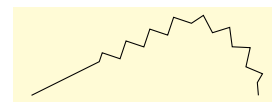
`/pgf/decoration/pre=<decoration>` (no default, initially `lineto`)

Устанавливает декорацию, которая должна использоваться перед началом основной декорации. Параметр `<decoration>` будет использоваться на длине `pre length`, имеющей по умолчанию длину `0pt`. Таким образом, для опции `pre`, чтобы она дала результат, нужно также установить опцию `pre length`.

```

\tikz [decoration={zigzag,
                  pre=lineto,pre length=1cm}]
\draw [decorate] (0,0) -- (2,1) arc (90:0:1);

```



```

\tikz [decoration={zigzag,
                  pre=moveto,pre length=1cm}]
\draw [decorate] (0,0) -- (2,1) arc (90:0:1);

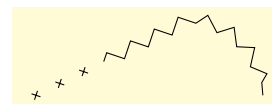
```



```

\tikz [decoration={zigzag,
                  pre=crosses,pre length=1cm}]
\draw [decorate] (0,0) -- (2,1) arc (90:0:1);

```

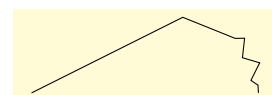


Заметим, что значение по умолчанию опции `pre` — `lineto` (чертит прямую), а не `curveto` (чертит кривую). Поэтому значение по умолчанию для декорации `pre` не может поддерживаться кривой (по естественным причинам). Поэтому следует изменить значение опции `pre` на `curveto`, если нужна криволинейная траектория.

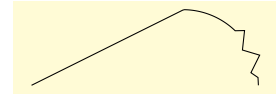
```

\tikz [decoration={zigzag,pre length=3cm}]
\draw [decorate](0,0) -- (2,1) arc (90:0:1);

```



```
\tikz [decoration={zigzag,
  pre=curveto,pre length=3cm}]
\draw [decorate](0,0) -- (2,1) arc (90:0:1);
```



`/pgf/decoration/pre length=<dimension>` (no default, initially 0pt)

Определяет расстояние, на котором должна использоваться пред- декорация. Если пред- декорация не нужна, значение этой опции должно равняться 0pt (именно так, а не иначе, то есть не 0cm, или что-то еще, дающее ту же величину).

`/pgf/decorations/post=<decoration>` (no default, initially lineto)

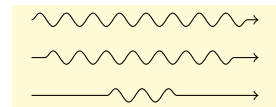
Аналогична опции `pre`, только для конца декорации.

`/pgf/decorations/post length=<dimension>` (no default, initially 0pt)

Аналогична опции `pre length`, только для опции `post`.

Приведем типичный пример, показывающий, как можно использовать эти опции:

```
\begin{tikzpicture}[decoration=snake,
line around/.style={decoration={pre length=#1,
  post length=#1}}]
\draw[->,decorate] (0,0) -- ++(3,0);
\draw[->,decorate,line around=5pt] (0,-5mm) -- ++(3,0);
\draw[->,decorate,line around=1cm] (0,-1cm) -- ++(3,0);
\end{tikzpicture}
```



# Глава 19

## Преобразования

### 19.1 Различные системы координат

Поиск на экране или бумаге позиции, в которую нужно поместить заданную точку, процесс достаточно длинный. Чтобы найти такую позицию, нужно провести некоторые преобразования, среди которых сдвиг, вращение, подгонка, масштабирование. Например, чтобы найти на экране или бумаге позицию, в которую нужно поместить точку (1,2), следует применить такие преобразования:

1. Сначала `pgf` интерпретирует точку (1,2) в  $xu$ -системе координат, складывая, в данном случае, текущий  $x$ -вектор с удвоенным текущим  $y$ -вектором.

2. Затем `pgf` применяет матрицу преобразования координат к результирующей координате, определяя финальную позицию точки внутри рисунка.

3. Внутренний драйвер (например, `dvips` или `pdftex`) добавляет такие команды преобразований, чтобы координату сдвинуть в правильную позицию в системе координат страницы  $\text{T}_\text{E}_\text{X}$ 'а.

4. Программа `pdf` (или `PostScript`) применяет матрицу преобразования холста в точке, что может еще раз изменить положение точки на странице.

5. Наконец, приложение для просмотра или печати применяет матрицу преобразования устройства, чтобы преобразовать координату в ее заключительную позицию на экране или листе бумаги.

В действительности, процесс даже более сложен, но сказанное выше должно прояснить основную идею: точка постоянно преобразуется в связи с изменением используемой в данный момент системы координат.

`TikZ` предоставляет доступ только к  $xu$ -системе координат и к матрице преобразования координат (см. далее), `pgf` позволяет изменить матрицу преобразования холста, но для этого надо использовать команды основного уровня, и хорошо понимать, что происходит. Все такие преобразования требуют непростых вычислений.

### 19.2 Системы координат $xu$ и $xuz$

Самые простые системы координат —  $xu$  и  $xuz$ . Основная идея проста: всякий раз, когда определяется точка, например (2,3), вычисляется выражение  $2v_x + 3v_y$ , где  $v_x$  — текущий  $x$ -вектор, а  $v_y$  — текущий  $y$ -вектор. Точно так же, точка (1,2,3) означает  $v_x + 2v_y + 3v_z$ . В отличие от других пакетов, `pgf` не настаивает, что вектор  $v_x$  имеет  $y$ -компоненту равной 0, то есть, что это горизонтальный вектор. Вместо этого,  $x$ -вектор может быть любым вектором. Но, обычно,  $x$ -вектор будет горизонтальным.

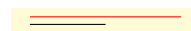
Одно из нежелательных следствий такой общности, то, что нельзя задавать смешанные (размерные и безразмерные) координаты точек, например, (1,2pt).

Чтобы изменять  $x$ -,  $y$ - и  $z$ -векторы, можно использовать следующие опции:

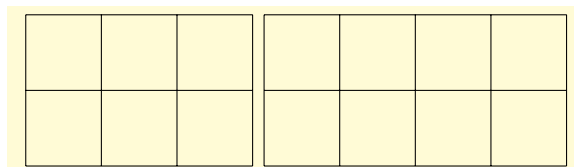
`/tikz/x=<value>` (no default, initially 1cm)

Если `<value>` — размерная величина,  $x$ -вектор системы координат  $xyz$  устанавливается равным (`<value>`, 0pt).

```
\begin{tikzpicture}
\draw (0,0) -- +(1,0); \draw[x=2cm,red] (0,0.1) -- +(1,0);
\end{tikzpicture}
```



```
\tikz \draw[x=1.5cm]
(0,0) grid (2,2);
\tikz \draw[x=2cm]
(0,0) grid (2,2);
```



Примеры показывают, что размер  $x$ -шага в сетке, точно так же, как и все другие размеры, не зависит от  $x$ -вектора. В сетке  $x$ -вектор используется только для того, чтобы определить координаты ее верхнего правого угла.

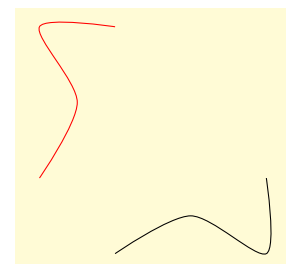
Если `<value>` — точка с двумя координатами,  $x$ -вектор  $xyz$ -системы координат определяется равным этой точке. Если `<value>` содержит запятую, точка должна заключаться в фигурные скобки.

```
\begin{tikzpicture}
\draw (0,0) -- (1,0);
\draw[x={(2cm,0.5cm)},red] (0,0) -- (1,0);
\end{tikzpicture}
```



Это можно использовать для того, например, чтобы изменить «смысл»  $x$ -координаты и  $y$ -координаты.

```
\begin{tikzpicture}[smooth]
\draw plot coordinates{(1,0) (2,0.5) (3,0) (3,1)};
\draw[x={(0cm,1cm)},y={(1cm,0cm)},red]
plot coordinates{(1,0) (2,0.5) (3,0) (3,1)};
\end{tikzpicture}
```



`/tikz/y=<value>` (no default, initially 1cm)

Аналогична опции  $x$ , но для  $y$ -координаты. Если `<value>` — размерная величина,  $y$ -вектор устанавливается равным (0pt, `<value>`).

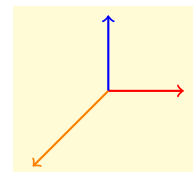
`/tikz/z=<value>` (no default, initially 1cm)

Аналогична опции  $y$ , но теперь, если `<value>` — размерная величина,  $z$ -вектор устанавливается равным (`<value>`, `<value>`).

```

\begin{tikzpicture}[z=-1cm,->,thick]
  \draw[red] (0,0,0) -- (1,0,0);
  \draw[blue] (0,0,0) -- (0,1,0);
  \draw[orange] (0,0,0) -- (0,0,1);
\end{tikzpicture}

```



## 19.3 Координатные преобразования

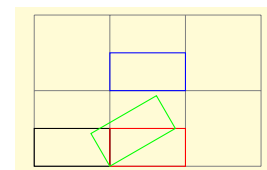
PGF и TikZ позволяют определять координатные преобразования. Всякий раз, когда определяется точка в виде  $(1,0)$  или  $(1\text{cm}, 1\text{pt})$ , эта точка сначала преобразуется в позицию вида: на  $x$  вправо и на  $y$  вверх. Аналогично преобразуется точка вида  $(30:2\text{cm})$ .

Следующий шаг: применить текущую матрицу преобразования координат к точке. Например, матрица преобразования координат может быть установлен так, что она добавляет некоторую константу к значению  $x$ , или может обменивать значения  $x$  и  $y$ . В общем случае, возможно любое стандартное преобразование: перемещение, поворот, отклонение или масштабирование, или любая их комбинации.

```

\begin{tikzpicture}
\draw[help lines] (0,0) grid (3,2);
\draw (0,0) rectangle (1,0.5);
\begin{scope}[xshift=1cm]
  \draw [red] (0,0) rectangle (1,0.5);
  \draw[yshift=1cm] [blue] (0,0) rectangle (1,0.5);
  \draw[rotate=30] [green](0,0) rectangle (1,0.5);
\end{scope}
\end{tikzpicture}

```



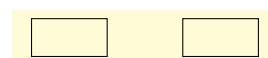
Самый важный аспект матрицы преобразования координат тот, что она применяется только к координатам точки! В частности, координатное преобразование не влияет на такие вещи, как ширина линии, ее пунктирность или растушевывание. Общее правило таково: если нет точки, вовлекаемой в вычисления даже косвенно, матрица не применяется.

Установку матрицы нельзя сделать напрямую. Все, что можно сделать — добавить другое преобразование к текущей матрице. Однако все преобразования являются локальными в текущей TeX-группе. Все преобразования добавляются, используя графические опции, описанные ниже. Преобразования применяются немедленно, когда они встречаются «в середине пути», и применяются только к точкам пути, сопровождаемым опциями преобразования.

```

\tikz \draw (0,0) rectangle (1,0.5)
  [xshift=2cm] (0,0) rectangle (1,0.5);

```



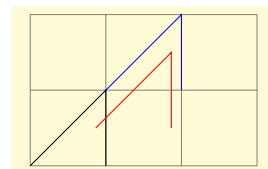
**Замечание.** Следует избегать «агрессивных» преобразований, таких как масштабирование с коэффициентом 1000. Причина в том, что все преобразования используют TeX, имеющий низкую точность вычислений. Кроме того, в определенных ситуациях

TikZ вычисляет обратную матрицу для текущей матрицы преобразования, а это приведет к ошибке, если матрица преобразования плохо обусловлена или сингулярна.

`/tikz/shift=<coordinate>` (no value)

Прибавляет точку с координатами `<coordinate>` ко всем точкам.

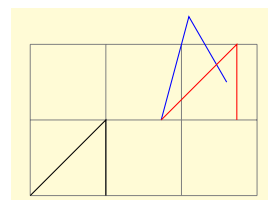
```
\begin{tikzpicture}
\draw[help lines] (0,0) grid (3,2);
\draw (0,0) -- (1,1) -- (1,0);
\draw[shift={(1,1)},blue] (0,0) -- (1,1) -- (1,0);
\draw[shift={(30:1cm)},red] (0,0) -- (1,1) -- (1,0);
\end{tikzpicture}
```



`/tikz/shift only` (no value)

Опция (без параметра!) отменяет все текущие преобразования за исключением смещения. Опция полезна в ситуациях, использующих сложное преобразование.

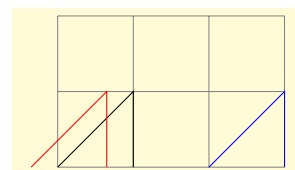
```
\begin{tikzpicture}
\draw[help lines] (0,0) grid (3,2);
\draw (0,0) -- (1,1) -- (1,0);
\draw[rotate=30,xshift=2cm,blue]
(0,0) -- (1,1) -- (1,0);
\draw[rotate=30,xshift=2cm,shift only,red]
(0,0) -- (1,1) -- (1,0);
\end{tikzpicture}
```



`/tikz/xshift=<dimension>` (no default)

Прибавляет `<dimension>` к  $x$ -значению всех точек.

```
\begin{tikzpicture}
\draw[help lines] (0,0) grid (3,2);
\draw (0,0) -- (1,1) -- (1,0);
\draw[xshift=2cm,blue] (0,0) -- (1,1) -- (1,0);
\draw[xshift=-10pt,red] (0,0) -- (1,1) -- (1,0);
\end{tikzpicture}
```



`/tikz/yshift=<dimension>` (no default)

Прибавляет `<dimension>` к  $y$ -значению всех точек.

`/tikz/scale=<factor>` (no default)

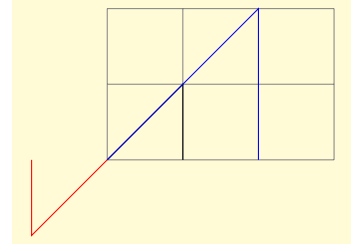
Умножает все координаты точек на заданный множитель `<factor>`, который не должен быть чрезмерно большим или очень маленьким по абсолютной величине.



```

\begin{tikzpicture}
\draw[help lines] (0,0) grid (3,2);
\draw (0,0) -- (1,1) -- (1,0);
\draw[scale=2,blue] (0,0) -- (1,1) -- (1,0);
\draw[scale=-1,red] (0,0) -- (1,1) -- (1,0);
\end{tikzpicture}

```



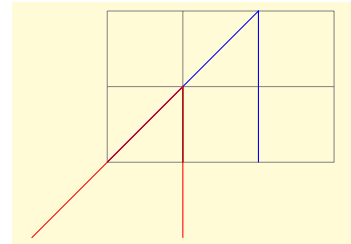
`/tikz/scale around`={<factor>:<coordinate>} (no default)

Масштабирует систему координат на множитель <factor>, помещая точку начала масштабирования не в начало координат, а в точку с координатами <coordinate>.

```

\begin{tikzpicture}
\draw[help lines] (0,0) grid (3,2);
\draw (0,0) -- (1,1) -- (1,0);
\draw[scale=2,blue]
(0,0) -- (1,1) -- (1,0);
\draw[scale around={2:(1,1)},red]
(0,0) -- (1,1) -- (1,0);
\end{tikzpicture}

```



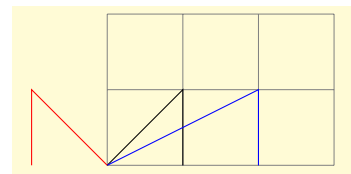
`/tikz/xscale`=<factor> (no default)

Умножает только  $x$ -значение всех точек на заданный множитель <factor>.

```

\begin{tikzpicture}
\draw[help lines] (0,0) grid (3,2);
\draw (0,0) -- (1,1) -- (1,0);
\draw[xscale=2,blue] (0,0) -- (1,1) -- (1,0);
\draw[xscale=-1,red] (0,0) -- (1,1) -- (1,0);
\end{tikzpicture}

```



`/tikz/yscale`=<factor> (no default)

Умножает только  $y$ -значение всех точек на заданный множитель <factor>.

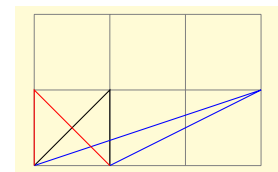
`/tikz/xslant`=<factor> (no default)

Отклоняет точку по горизонтали на заданную величину <factor>.

```

\begin{tikzpicture}
\draw[help lines] (0,0) grid (3,2);
\draw (0,0) -- (1,1) -- (1,0);
\draw[xslant=2,blue] (0,0) -- (1,1) -- (1,0);
\draw[xslant=-1,red] (0,0) -- (1,1) -- (1,0);
\end{tikzpicture}

```



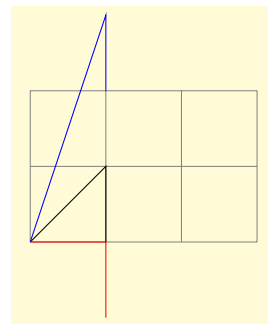
`/tikz/yslant`=<factor> (no default)

Отклоняет точку по вертикали на заданную величину <factor>.

```

\begin{tikzpicture}
\draw[help lines] (0,0) grid (3,2);
\draw (0,0) -- (1,1) -- (1,0);
\draw[yslant=2,blue]
      (0,0) -- (1,1) -- (1,0);
\draw[yslant=-1,red]
      (0,0) -- (1,1) -- (1,0);
\end{tikzpicture}

```



`/tikz/rotate=<degree>`

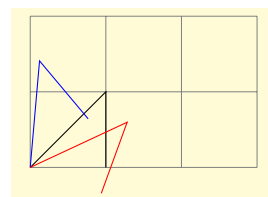
(no default)

Вращает систему координат на угол `<degree>`.

```

\begin{tikzpicture}
\draw[help lines] (0,0) grid (3,2);
\draw (0,0) -- (1,1) -- (1,0);
\draw[rotate=40,blue] (0,0) -- (1,1) -- (1,0);
\draw[rotate=-20,red] (0,0) -- (1,1) -- (1,0);
\end{tikzpicture}

```



`/tikz/rotate around={<degree>:<coordinate>}`

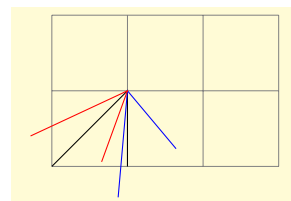
(no default)

Вращает систему координат на угол `<degree>` вокруг точки `<coordinate>`.

```

\begin{tikzpicture}
\draw[help lines] (0,0) grid (3,2);
\draw (0,0) -- (1,1) -- (1,0);
\draw[rotate around={40:(1,1)},blue]
      (0,0) -- (1,1) -- (1,0);
\draw[rotate around={-20:(1,1)},red]
      (0,0) -- (1,1) -- (1,0);
\end{tikzpicture}

```



`/tikz/cm={<a>, <b>, <c>, <d>, <coordinate>}`

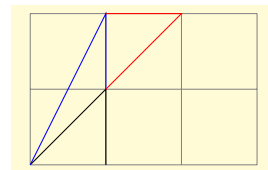
(no default)

Применяет следующее преобразование ко всем точкам: пусть  $(x, y)$  — преобразуемая точка, пусть параметр `<coordinate>` определять точку  $(t_x, t_y)$ ; тогда новая точка определяется по правилу:  $\begin{pmatrix} a & b \\ c & d \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix}$ .

```

\begin{tikzpicture}
\draw[help lines] (0,0) grid (3,2);
\draw (0,0) -- (1,1) -- (1,0);
\draw[cm={1,1,0,1,(0,0)},blue]
      (0,0) -- (1,1) -- (1,0);
\draw[cm={0,1,1,0,(1cm,1cm)},red]
      (0,0) -- (1,1) -- (1,0);
\end{tikzpicture}

```



`/tikz/reset cm`

(no default)

Полностью сбрасывает матрицу преобразования координат в единичную матрицу. Это разрушает не только преобразования, применяемые в текущей области видимости, но также и все преобразования, унаследованные из окружающих областей видимости. Не следует использовать эту опцию, если не знаете, что делаете!

## 19.4 Преобразования холста

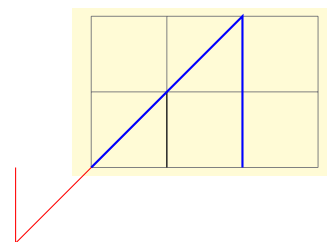
Преобразование холста (детали см. в [1, разделе 68.4]) лучше всего представлять, как преобразование, в котором холст для рисования растягивается или вращается. Нарисуем на воздушном шаре (холсте) рисунок, а затем наполним шар воздухом: мало того, что текст станет больше, но и тонкие линии станут толще. Преобразования холста должны использоваться с большой осторожностью. Важно то, что преобразование холста всегда применяется к пути в целом, и нельзя использовать разные преобразования в разных его частях.

`/tikz/transform canvas=<options>`

(no default)

Параметр `<options>` должен содержать опции преобразования координат, такие как `scale` или `xshift`. Можно задавать множество опций, их результаты накапливаются обычным образом. В результате применения этих опций немедленно меняется текущая матрица преобразования холста. Матрица преобразования координат не меняется. Отслеживание размера картинки (локально) отключается, и координаты узла более не будут соответствовать действительности.

```
\begin{tikzpicture}
\draw[help lines] (0,0) grid (3,2);
\draw (0,0) -- (1,1) -- (1,0);
\draw[transform canvas={scale=2},blue]
      (0,0) -- (1,1) -- (1,0);
\draw[transform canvas={rotate=180},red]
      (0,0) -- (1,1) -- (1,0);
\end{tikzpicture}
```

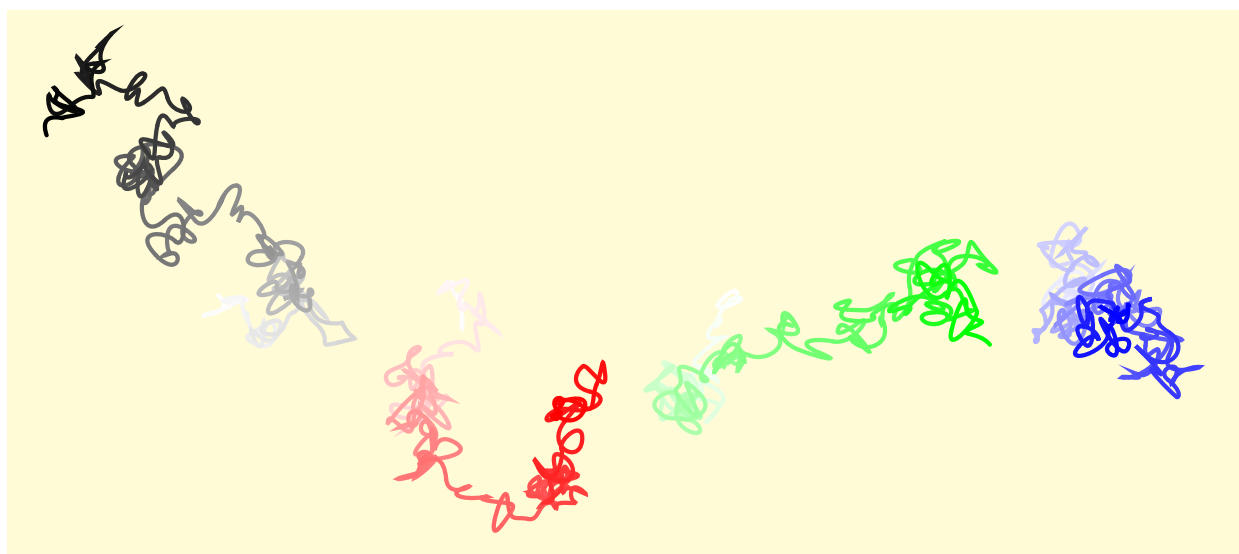


Часть III  
Математика в pgf

Pgf, набирая рисунки, выполняет много вычислений. Для этого pgf использует математический механизм, представленный как часть pgf-пакета, который загружается автоматически при загрузке pgf. Но он может использоваться и независимо от пакета pgf, для этого нужно загрузить только пакет `pgfmath`.

Механизм математики обеспечивает широкие функциональные возможности, выполняя синтаксический анализ математических операций, использующих целые числа и дроби. Могут анализироваться различные функции, включая тригонометрические функции и генераторы случайных чисел (см. раздел 20.2). Дополнительно, каждая операция и функция имеют независимую pgf-команду, связанную с ней (см. раздел 20.4), и доступную вне синтаксического анализатора. Математические возможности pgf используются косвенно, в основном тогда, когда надо найти точку с заданными координатами, определить число или размеры в высокоуровневом макросе.

Поддержку механизму математики обеспечивают две библиотеки: одна для работы с числами с фиксированной точкой, а вторая — с числами с плавающей точкой.



# Глава 20

## Механизм математики

### 20.1 Принципы расчета

#### 20.1.1 Уровни математического механизма

1. **Главный уровень**, который обычно используется непосредственно и определяет команду `\pgfmathparse`, анализирующую математические выражения и вычисляющую их. Кроме того, главный уровень определяет некоторые дополнительные функции, подобные макросам пакета `calc`, для установки размерностей и счетчиков. Такие функции — только обертки вокруг команды `\pgfmathparse`.

2. **Уровень вычисления** определяет макросы для выполнения конкретной вычислительной операции, например, обратной величины или умножения. Синтаксический анализатор использует эти макросы при реальном вычислении.

3. **Уровень реализации** обеспечивает реальное выполнение вычислений. Реализации могут меняться без воздействия на более высокие уровни.

#### 20.1.2 Эффективность и точность математики

В настоящее время все математические алгоритмы реализованы в `TeX`'е. Но `TeX` — язык для набора текстов, а не для математики, и, как в любом языке программирования, возникает необходимость компромисса между точностью и эффективностью.

Математический аппарат, который обеспечивает синтаксический анализатор, иногда приносит необязательные накладные расходы памяти и времени, а в некоторых случаях такие расходы не нужны и нежелательны. Чтобы преодолеть это, реализована следующая особенность: когда математические операции не нужны, выражению должен предшествовать знак `+`, что приведет к отказу от обработки выражения синтаксическим анализатором и назначение пройдет быстрее (см. раздел 20.2.1).

```
\pgfmathsetlength\mydimen{1cm} % анализируется: выполняется медленно.  
\pgfmathsetlength\mydimen{+1cm} % не анализируется: намного быстрее.
```

### 20.2 Вычисление математических выражений

Самый легкий способ использовать математический механизм `pgf` состоит в том, чтобы ввести математическое выражение, заданное в понятной системе обозначений (нотации), например, `1cm+4*2cm/5.5` или `2*3+3*sin(30)`. Такое выражение можно проанализировать и результат поместить в регистр, счетчик или макрос. В любой момент

вычисления все числа не должны превышать  $\pm 16383.99999$ . Это ограничение Т<sub>Е</sub>X'а и оно означает, что все вычисления приближительны и, кроме того, выполняются не очень быстро. Однако механизм вычислений можно заменить (см. раздел 20.5).

Сначала рассмотрим макросы высокого уровня, предназначенные для анализа выражений, а затем синтаксис самих выражений.

### 20.2.1 Команды для анализа выражений

`\pgfmathparse{<expression>}`

Макрос анализирует `<expression>` и возвращает результат без единиц измерения в макрос `\pgfmathresult`. Например, выражение `\pgfmathparse {2pt+3.5pt}` установит в `\pgfmathresult` текст 5.5.

Отметим особенности анализа математических выражений.

- Результат сохраняется в макросе `\pgfmathresult` в виде десятичного числа без единиц измерения. Это не зависит от того, содержит ли `<expression>` число со спецификацией или нет. Все числа с единицами измерения преобразуются к pt.
- Можно проверить, содержало или нет выражение единицы измерения, используя макрос `\ifpgfmathunitsdeclared`: после вызова `\pgfmathparse` оно будет хранить `true`, если в выражении содержалась некоторая единица измерения.
- Синтаксический анализатор может распознавать регистры Т<sub>Е</sub>X'а и модули размерностей, таким образом, могут быть проанализированы выражения `\mydimen`, `0.5\mydimen`, `\wd\mybox`, `0.5\dp\mybox`, `\myscount\mydimen` и так далее.
- Чтобы изменить порядок вычисления, нужно использовать круглые скобки.
- Распознаются различные функции, потому можно проанализировать выражения вида `\sin(.5*\pi r)*60`, которое означает «синус от числа 0.5, умноженного на  $\pi$  радиан, умножить на 60». Аргументом функций может быть любое выражение.
- Научная система обозначений вида `1.234e + 4` распознается (но ограничение на диапазон значений все еще существует). Символ основания степени может быть заглавной или строчной буквой (то есть, *E* или *e*).
- Целое число с нулем-префиксом (исключая, конечно сам нуль), интерпретируется как восьмеричное число и автоматически преобразуется в десятичное.
- Целое число с префиксом `0x` или `0X` интерпретируется как шестнадцатеричное число и автоматически преобразуется в десятичное. Алфавитные цифры могут быть записаны как прописными, так и строчными буквами.
- Целое число с префиксом `0b` или `0B` интерпретируется как двоичное число и автоматически преобразуется в десятичное.
- Выражение (или часть выражения) окруженное двойными кавычками не вычисляется (следует использовать с большой осторожностью!).

`\pgfmathqparse{<expression>}`

Макрос подобен `\pgfmathparse`, но есть два отличия: (1) `\pgfmathqparse` не анализирует функции, научную систему обозначения чисел, префиксы для бинарных, восьмеричных или шестнадцатеричных чисел, и не понимает специального использования символов (`"`), (`?`), (`:`); (2) числа в `<expression>` должны включать единицу Т<sub>Е</sub>X'а (кроме таких случаев, как `0.5\pgf@x`), что упрощает задачу анализа вещественных чисел, а потому `\pgfmathqparse` работает в два раза быстрее `\pgfmathparse`. Результат будет, по-прежнему, числом без единиц измерения.

**`\pgfmathpostparse`**

В конце синтаксического анализа выполняется эта команда, позволяя выполнить некоторое специальное действие на результате синтаксического анализа. Когда она выполняется, макрос `\pgfmathresult` хранит результат синтаксического анализа.

Во всех следующих командах, если параметр `<expression>` начинается со знака `+`, не производится синтаксический анализ и выполняются действия, используя обычные команды `TeX`'а. Это будет быстрее, чем вызов синтаксического анализатора.

**`\pgfmathsetlength`**`{<dimension register>}{<expression>}`

Устанавливает длину `<dimension register>` `TeX`'а к значению (в pt), определяемому выражением `<expression>`, которое анализируется, используя `\pgfmathparse`.

**`\pgfmathaddtolength`**`{<dimension register>}{<expression>}`

Добавляет значение (в pt) выражения `<expression>` в `<dimension register>` `TeX`'а.

**`\pgfmathsetcount`**`{<count register>}{<expression>}`

Устанавливает значение `<count register>` `TeX`'а, равным усеченному (truncated) значению, определяемому выражением `<expression>`.

**`\pgfmathaddtocount`**`{<count register>}{<expression>}`

Добавляет усеченное (truncated) значение `<expression>` к `<count register>` `TeX`'а.

**`\pgfmathsetcounter`**`{<counter>}{<expression>}`

Устанавливает значение `<counter>` `LATeX`'а равным усеченному (truncated) значению, определяемому выражением `<expression>`.

**`\pgfmathaddtocounter`**`{<counter>}{<expression>}`

Добавляет усеченное значение `<expression>` к `<counter>`.

**`\pgfmathsetmacro`**`{<macro>}{<expression>}`

Определяет `<macro>` как значение `<expression>`. Результат — десятичное число без единиц измерения.

**`\pgfmathsetlengthmacro`**`{<macro>}{<expression>}`

Определяет `<macro>` как значение `<expression>` в pt `LATeX`'а.

**`\pgfmathtruncatemacro`**`{<macro>}{<expression>}`

Определяет `<macro>` как усеченное значение `<expression>`.

## 20.3 Синтаксис математических выражений

Далее описываются операторы и функции, распознаваемые по умолчанию.

### 20.3.1 Операторы

Оператор	Описание	Характер	Операция <code>TeX</code> 'а
$x + y$	Сложение	infix	add
$x - y$	Вычитание	infix	subtract
$-x$	Изменение знака	prefix	neg
$x * y$	Умножение	infix	multiply
$x / y$	Деление ( $y \neq 0$ )	infix	divide
$x^y$	Возведение в степень ( $y \neq 0$ )	infix	pow



$x!$	Факториал	postfix	factorial
$xr$	Перевод в градусы	postfix	deg
$x? y: z$	Условный оператор: если $x \neq 0$ , то $y$	conditional	ifthenelse
$x == y$	Сравнение: если $x$ равен $y$ , возвращает 1, иначе - 0	infix	equal
$x > y$	Сравнение: если $x > y$ , возвращает 1, иначе - 0	infix	greater
$x < y$	Сравнение: если $x < y$ , возвращает 1, иначе - 0	infix	less
$x! = y$	Сравнение: если $x$ не равен $y$ , возвращает 1, иначе - 0	infix	notequal
$x >= y$	Сравнение: если $x$ не меньше $y$ , возвращает 1, иначе - 0	infix	notless
$x <= y$	Сравнение: если $x$ не больше $y$ , возвращает 1, иначе - 0	infix	notgreater
$x \& \& y$	Логическое И : возвращает 1, если $x \neq 0$ и $y \neq 0$	infix	and
$x    y$	Логическое ИЛИ: возвращает 1, если $x \neq 0$ или $y \neq 0$	infix	or
$!x$	Логическое отрицание: возвращает 1, если $x = 0$ , иначе - 0	prefix	not
$(...)$	Круглые скобки определяют приоритет вычислений, включая группировку аргументов функций. Для функций с одним аргументом скобки можно не ставить, например, $\sin 30$ или $\sin(30)$ . Функции имеют наивысший приоритет, поэтому, $\sin 30 * 10$ равно $\sin(30) * 10$ .	group	
$\{...\}$	Фигурные скобки используются для указание на структуры, подобные массивам. Массив состоит из разделяемых запятой элементов. Каждый элемент в массиве оценивается после анализа, поэтому можно использовать выражения. Элементами массива могут быть массивы, что позволяет использовать многомерные массивы. При хранении массива в макросе, следует сам массив заключать в фигурные скобки: <code>\def\myarray{{1,2,3}}</code> .	array	
$[j]$	Позволяет получить доступ к элементу массива с индексом $j$ . Индексация начинается с нуля. Если индекс больше или равен количеству значений в массиве, возникает ошибка и возвращается нуль.	array	array
"x"	Операторы кватирования $x$ . Позволяют защитить макросы от преждевременного расширения. В идеале следует избегать таких макросов. Операторы кватирования следует использовать с большой осторожностью, так как с результатом дальнейшие расчеты вряд ли будут возможны.		group

---

Рассмотрим примеры.

```
\def\myarray{{1, 3^2, 2+1, cos(1),3!,10/2,sin(\i*5)}}
\foreach \i in {0,...,6}{\pgfmathparse{\myarray[\i]}\pgfmathresult, }
```

```
1, 9.0, 3.0, 0.99985, 6.0, 5.0, 0.5,
```

```
\def\myarray{{7,-3,4,-9,11}}
\pgfmathparse{\myarray[3]} \pgfmathresult
```

```
-9.0
```

```
\def\print#1{\pgfmathparse{#1}\pgfmathresult}
\def\identitymatrix{{1,0,0},{0,1,0},{0,0,1}}
\tikz[x=0.5cm,y=0.5cm]\foreach \i in {0,1,2}
  \foreach \j in {0,1,2}
  \node at (\j,-\i) [anchor=base] {\print{\identitymatrix[\i][\j]}};
```

```
1 0 0
```

```
0 1 0
```

```
0 0 1
```

### 20.3.2 Функции

Распознаются следующие математические функции, которые имеют pgf-команду, связанную с ней (к имени функции добавляется префикс `\pgfmath`, но есть исключения):

abs	acos	add	and	array	asin	atan
atan2	bin	ceil	cos	cosec	cosh	cot
deg	depth	div	divide	e	equal	factorial
false	floor	frac	greater	height	hex	int
ifthenelse	less	ln	log10	log2	max	min
mod	Mod	multiply	neg	not	notequal	notgreater
notless	oct	or	pi	pow	rad	rand
random	real	rnd	round	sec	sin	sinh
sqrt	subtract	tan	tanh	true	veclen	width

### Основные арифметические функции

add(x,y)		
<code>\pgfmathadd{x}{y}</code>	81.0	<code>\pgfmathparse{add(75,6)}\pgfmathresult</code>
subtract(x,y)		
<code>\pgfmathsubtract{x}{y}</code>	69.0	<code>\pgfmathparse{subtract(75,6)}\pgfmathresult</code>
neg(x)		
<code>\pgfmathneg{x}</code>	-75.0	<code>\pgfmathparse{neg(75)}\pgfmathresult</code>
multiply(x,y)		
<code>\pgfmathmultiply{x}{y}</code>	450.0	<code>\pgfmathparse{multiply(75,6)}\pgfmathresult</code>
divide(x,y)		
<code>\pgfmathdivide{x}{y}</code>	12.5	<code>\pgfmathparse{divide(75,6)}\pgfmathresult</code>
div(x,y)		Округляет результат до ближайшего целого
<code>\pgfmathdiv{x}{y}</code>	12	<code>\pgfmathparse{div(75,6)}\pgfmathresult</code>
factorial(x)		

$\backslash\text{pgfmathfactorial}\{x\}$	120.0	$\backslash\text{pgfmathparse}\{\text{factorial}(5)\}\backslash\text{pgfmathresult}$	Вычисляет $\sqrt{x}$
$\text{sqrt}(x)$			
$\backslash\text{pgfmathsqrt}\{x\}$	2.23606	$\backslash\text{pgfmathparse}\{\text{sqrt}(5)\}\backslash\text{pgfmathresult}$	Вычисляет $x^y$
$\text{pow}(x,y)$			
$\backslash\text{pgfmathpow}\{x\},\{y\}$	25.0	$\backslash\text{pgfmathparse}\{\text{pow}(5,2)\}\backslash\text{pgfmathresult}$	Вычисляет $e=2.718281828$
$e$			
$\backslash\text{pgfmathe}$	7.38902	$\backslash\text{pgfmathparse}\{e^2\}\backslash\text{pgfmathresult}$	Вычисляет $e^x$
$\text{exp}(x)$			
$\backslash\text{pgfmathexp}\{x\}$	2.71825	$\backslash\text{pgfmathparse}\{\text{exp}(1)\}\backslash\text{pgfmathresult}$	Вычисляет $\ln x$
$\ln(x)$			
$\backslash\text{pgfmathln}\{x\}$	0.99995	$\backslash\text{pgfmathparse}\{\ln(e)\}\backslash\text{pgfmathresult}$	Вычисляет $\log_{10} x$
$\log_{10}(x)$			
$\backslash\text{pgfmathlogten}\{x\}$	1.99997	$\backslash\text{pgfmathparse}\{\log_{10}(100)\}\backslash\text{pgfmathresult}$	Вычисляет $\log_2 x$
$\log_2(x)$			
$\backslash\text{pgfmathlogtwo}\{x\}$	2.99997	$\backslash\text{pgfmathparse}\{\log_2(8)\}\backslash\text{pgfmathresult}$	Вычисляет $ x $
$\text{abs}(x)$			
$\backslash\text{pgfmathabs}\{x\}$	8.2	$\backslash\text{pgfmathparse}\{\text{abs}(-8.2)\}\backslash\text{pgfmathresult}$	Остаток целочисленного деления со знаком $x$
$\text{mod}(x,y)$			
$\backslash\text{pgfmathmod}\{x\},\{y\}$	-2.0	$\backslash\text{pgfmathparse}\{\text{mod}(-5,3)\}\backslash\text{pgfmathresult}$	Остаток целочисленного деления со знаком $+$
$\text{Mod}(x,y)$			
$\backslash\text{pgfmathMod}\{x\},\{y\}$	1.0	$\backslash\text{pgfmathparse}\{\text{Mod}(-5,3)\}\backslash\text{pgfmathresult}$	

## Функции округления

$\text{round}(x)$			Округление к ближайшему целому
$\backslash\text{pgfmathround}\{x\}$	2.0	$\backslash\text{pgfmathparse}\{\text{round}(1.5)\}\backslash\text{pgfmathresult}$	
	-2.0	$\backslash\text{pgfmathparse}\{\text{round}(-1.5)\}\backslash\text{pgfmathresult}$	
$\text{floor}(x)$			Округление к ближайшему меньшему целому
$\backslash\text{pgfmathfloor}\{x\}$	1.0	$\backslash\text{pgfmathparse}\{\text{floor}(32.5/17)\}\backslash\text{pgfmathresult}$	
	-2.0	$\backslash\text{pgfmathparse}\{\text{floor}(-32.5/17)\}\backslash\text{pgfmathresult}$	
$\text{ceil}(x)$			Округление к ближайшему большему целому
$\backslash\text{pgfmathceil}\{x\}$	2.0	$\backslash\text{pgfmathparse}\{\text{ceil}(32.5/17)\}\backslash\text{pgfmathresult}$	
	-1.0	$\backslash\text{pgfmathparse}\{\text{ceil}(-32.5/17)\}\backslash\text{pgfmathresult}$	
$\text{int}(x)$			Возвращает целую часть числа
$\backslash\text{pgfmathint}\{x\}$	1	$\backslash\text{pgfmathparse}\{\text{int}(32.5/17)\}\backslash\text{pgfmathresult}$	
	-1	$\backslash\text{pgfmathparse}\{\text{int}(-32.5/17)\}\backslash\text{pgfmathresult}$	
$\text{frac}(x)$			Возвращает дробную часть числа
$\backslash\text{pgfmathfrac}\{x\}$	0.91176	$\backslash\text{pgfmathparse}\{\text{frac}(32.5/17)\}\backslash\text{pgfmathresult}$	
	0.91176	$\backslash\text{pgfmathparse}\{\text{frac}(-32.5/17)\}\backslash\text{pgfmathresult}$	
$\text{real}(x)$			Возвращает вещественное число
$\backslash\text{pgfmathreal}\{x\}$	32.5	$\backslash\text{pgfmathparse}\{\text{real}(32.5)\}\backslash\text{pgfmathresult}$	
	-32.0	$\backslash\text{pgfmathparse}\{\text{real}(-32)\}\backslash\text{pgfmathresult}$	

## Тригонометрические функции

pi		Вычисляет $\pi = 3.141592654$
<code>\pgfmathpi</code>	3.141592654	<code>\pgfmathparse{pi}\pgfmathresult</code>
	179.99962	<code>\pgfmathparse{pi r}\pgfmathresult</code>
rad(x)		Возвращает число в радианах
<code>\pgfmathrad{x}</code>	1.57079	<code>\pgfmathparse{rad(90)}\pgfmathresult</code>
	-0.52359	<code>\pgfmathparse{rad(-30)}\pgfmathresult</code>
deg(x)		Возвращает число в градусах
<code>\pgfmathdeg{x}</code>	269.999	<code>\pgfmathparse{deg(3*pi/2)}\pgfmathresult</code>
	-179.99962	<code>\pgfmathparse{deg(-pi)}\pgfmathresult</code>
sin(x)		Возвращает $\sin x$
<code>\pgfmathsin{x}</code>	-1.0	<code>\pgfmathparse{sin(270)}\pgfmathresult</code>
	-0.99998	<code>\pgfmathparse{sin(-pi/2 r)}\pgfmathresult</code>
cos(x)		Возвращает $\cos x$
<code>\pgfmathcos{x}</code>	0.0	<code>\pgfmathparse{cos(270)}\pgfmathresult</code>
	-0.99998	<code>\pgfmathparse{cos(-pi r)}\pgfmathresult</code>
tan(x)		Возвращает $\operatorname{tg} x$
<code>\pgfmathatan{x}</code>	1.00005	<code>\pgfmathparse{tan(45)}\pgfmathresult</code>
	-1.0	<code>\pgfmathparse{tan(-pi/4 r)}\pgfmathresult</code>
sec(x)		Возвращает $1/\cos x$
<code>\pgfmathsec{x}</code>	1.0	<code>\pgfmathparse{sec(0)}\pgfmathresult</code>
	2.0	<code>\pgfmathparse{sec(-pi/3 r)}\pgfmathresult</code>
cosec(x)		Возвращает $1/\sin x$
<code>\pgfmathcosec{x}</code>	1.0	<code>\pgfmathparse{cosec(90)}\pgfmathresult</code>
	-2.00009	<code>\pgfmathparse{cosec(-pi/6 r)}\pgfmathresult</code>
cot(x)		Возвращает $\operatorname{ctg} x$
<code>\pgfmathcot{x}</code>	1.00005	<code>\pgfmathparse{cot(45)}\pgfmathresult</code>
	-1.00003	<code>\pgfmathparse{cot(-pi/4 r)}\pgfmathresult</code>
asin(x)		Возвращает $\arcsin x$ в градусах
<code>\pgfmathasin{x}</code>	90.0	<code>\pgfmathparse{asin(1)}\pgfmathresult</code>
	-30.0	<code>\pgfmathparse{asin(-0.5)}\pgfmathresult</code>
acos(x)		Возвращает $\arccos x$ в градусах
<code>\pgfmathacos{x}</code>	0.0	<code>\pgfmathparse{acos(1)}\pgfmathresult</code>
	120.0	<code>\pgfmathparse{acos(-0.5)}\pgfmathresult</code>
atan(x)		Возвращает $\operatorname{arctg} x$ в градусах
<code>\pgfmathatan{x}</code>	45.0	<code>\pgfmathparse{atan(1)}\pgfmathresult</code>
	-63.43495	<code>\pgfmathparse{atan(-2)}\pgfmathresult</code>
atan2(x,y)		Остаток целочисленного деления со знаком $x$
<code>\pgfmathatan2two{x},{y}</code>	143.13011	<code>\pgfmathparse{atan2(-4,3)}\pgfmathresult</code>
	36.86989	<code>\pgfmathparse{atan2(4,3)}\pgfmathresult</code>

## Функции сравнения и логические функции

<code>equal(x,y)</code>			
<code>\pgfmathequal{x},{y}</code>	0	<code>\pgfmathparse{equal(-4,3)}</code>	<code>\pgfmathresult</code>
	1	<code>\pgfmathparse{equal(4,4)}</code>	<code>\pgfmathresult</code>
<code>greater(x,y)</code>			
<code>\pgfmathgreater{x},{y}</code>	0	<code>\pgfmathparse{greater(-4,3)}</code>	<code>\pgfmathresult</code>
	1	<code>\pgfmathparse{greater(4,3)}</code>	<code>\pgfmathresult</code>
<code>less(x,y)</code>			
<code>\pgfmathless{x},{y}</code>	1	<code>\pgfmathparse{less(-4,3)}</code>	<code>\pgfmathresult</code>
	0	<code>\pgfmathparse{less(4,3)}</code>	<code>\pgfmathresult</code>
<code>notequal(x,y)</code>			
<code>\pgfmathnotequal{x},{y}</code>	1	<code>\pgfmathparse{notequal(-4,3)}</code>	<code>\pgfmathresult</code>
	0	<code>\pgfmathparse{notequal(4,4)}</code>	<code>\pgfmathresult</code>
<code>notgreater(x,y)</code>			
<code>\pgfmathnotgreater{x},{y}</code>	1	<code>\pgfmathparse{notgreater(-4,3)}</code>	<code>\pgfmathresult</code>
	0	<code>\pgfmathparse{notgreater(4,3)}</code>	<code>\pgfmathresult</code>
<code>notless(x,y)</code>			
<code>\pgfmathnotless{x},{y}</code>	0	<code>\pgfmathparse{notless(-4,3)}</code>	<code>\pgfmathresult</code>
	1	<code>\pgfmathparse{notless(4,3)}</code>	<code>\pgfmathresult</code>
<code>and(x,y)</code>			
<code>\pgfmathand{x},{y}</code>	1	<code>\pgfmathparse{and(-4,3)}</code>	<code>\pgfmathresult</code>
	0	<code>\pgfmathparse{and(4,0)}</code>	<code>\pgfmathresult</code>
<code>or(x,y)</code>			
<code>\pgfmathor{x},{y}</code>	1	<code>\pgfmathparse{or(-4,3)}</code>	<code>\pgfmathresult</code>
	1	<code>\pgfmathparse{or(4,0)}</code>	<code>\pgfmathresult</code>
<code>not(x)</code>			
<code>\pgfmathnot{x}</code>	0	<code>\pgfmathparse{not(4)}</code>	<code>\pgfmathresult</code>
	1	<code>\pgfmathparse{not(0)}</code>	<code>\pgfmathresult</code>
<code>ifthenelse(x,y,z)</code>			
<code>\pgfmathifthenelse{x},{y},{z}</code>	3	<code>\pgfmathparse{ifthenelse(-4,3,7)}</code>	<code>\pgfmathresult</code>
	7	<code>\pgfmathparse{ifthenelse(0,3,7)}</code>	<code>\pgfmathresult</code>
<code>true</code>		Возвращает 1	
<code>\pgfmatrue</code>	3	<code>\pgfmathparse{true ? 3 : 7}</code>	<code>\pgfmathresult</code>
<code>false</code>		Возвращает 0	
<code>\pgfmathfalse</code>	7	<code>\pgfmathparse{false ? 3 : 7}</code>	<code>\pgfmathresult</code>

### Функции для псевдослучайных чисел

<code>rnd</code>		Генерирует случайное число между 0 и 1
<code>\pgfmathrnd</code>	0.35255	<code>\pgfmathparse{rnd}</code>
<code>rand</code>		Генерирует случайное число между -1 и 1
<code>\pgfmathrand</code>	0.62688	<code>\pgfmathparse{rand}</code>
<code>random(x,y)</code>		
<code>\pgfmathrandom{x,y}</code>	0.9081	<code>\pgfmathparse{random()}</code>

20	<code>\pgfmathparse{random(100)}</code>	<code>\pgfmathresult</code>
111	<code>\pgfmathparse{random(100,200)}</code>	<code>\pgfmathresult</code>

## Базовые функции преобразования

<code>hex(x)</code>		Преобразует десятичное целое в шестнадцатеричное
<code>\pgfmathhex{x}</code>	fff	<code>\pgfmathparse{hex(65535)}</code>
<code>Hex(x)</code>		Преобразует десятичное целое в шестнадцатеричное
<code>\pgfmathHex{x}</code>	FFFF	<code>\pgfmathparse{Hex(65535)}</code>
<code>oct(x)</code>		Преобразует десятичное целое в восьмеричное
<code>\pgfmathoct{x}</code>	101	<code>\pgfmathparse{oct(65)}</code>
<code>bin(x)</code>		Преобразует десятичное целое в бинарное
<code>\pgfmathbin{x}</code>	1000001	<code>\pgfmathparse{bin(65)}</code>

## Другие функции

<code>min(x_1, \dots, x_n)</code>		Возвращает минимальное значение
<code>\pgfmathmin{x_1, \dots, x_n}</code>	-5.0	<code>\pgfmathparse{min(3,-5,0)}</code>
<code>max(x_1, \dots, x_n)</code>		Возвращает максимальное значение
<code>\pgfmathmax{x_1, \dots, x_n}</code>	3.0	<code>\pgfmathparse{max(3,-5,0)}</code>
<code>veclen(x,y)</code>		Возвращает $\sqrt{x^2 + y^2}$
<code>\pgfmathveclen{x}, {y}</code>	4.99994	<code>\pgfmathparse{veclen(-4,3)}</code>
<code>array(x,i)</code>		Возвращает $i$ -ый элемент массива $x$
<code>\pgfmatharray{x}, {i}</code>	4	<code>\pgfmathparse{array({4,3,8},0)}</code>
	3	<code>\pgfmathparse{array({4,3,8},1)}</code>
	8	<code>\pgfmathparse{array({4,3,8},2)}</code>
<code>sinh(x)</code>		Вычисляет гиперболический синус
<code>\pgfmathsinh{x}</code>	0.52103	<code>\pgfmathparse{sinh(0.5)}</code>
<code>cosh(x)</code>		Вычисляет гиперболический косинус
<code>\pgfmathcosh{x}</code>	1.12767	<code>\pgfmathparse{cosh(0.5)}</code>
<code>tanh(x)</code>		Вычисляет гиперболический тангенс
<code>\pgfmathtanh{x}</code>	0.462	<code>\pgfmathparse{tanh(0.5)}</code>
<code>width("x")</code>		Вычисляет ширину Т <sub>Е</sub> X-бокса с "x"
<code>\pgfmathwidth{"x"}</code>	27.40999	<code>\pgfmathparse{width("Some")}</code>
<code>height("x")</code>		Вычисляет высоту Т <sub>Е</sub> X-бокса с "x"
<code>\pgfmathheight{"x"}</code>	8.26465	<code>\pgfmathparse{height("Some")}</code>
<code>depth("x")</code>		Вычисляет глубину Т <sub>Е</sub> X-бокса с "x"
<code>\pgfmathdepth{"x"}</code>	2.33276	<code>\pgfmathparse{depth("Sony")}</code>

## 20.4 Дополнительные математические команды

Вместо того, чтобы анализировать и вычислять сложные выражения, можно использовать математический механизм, чтобы вычислить единственную математическую операцию. Макросы, используемые для многих из таких вычислений, перечислены выше. Но `pgf` определяет некоторые дополнительные команды.

### `\pgfmathreciprocal{<x>}`

Определяет в `\pgfmathresult` число  $1/x$ , обеспечивая при вычислении большую точность, когда  $x$  мало.

```
\pgfmathreciprocal{2}\pgfmathresult 0.5
```

### `\pgfmathapproxequalto{<x>}{<y>}`

Определяет в `\pgfmathresult` 1.0, если  $|<x> - <y>| < 0.0001$ , иначе 0.0. Как побочный эффект, в `\ifpgfmathcomparison` устанавливается соответствующее логическое значение.

```
\pgfmathapproxequalto{e}{2.718}\pgfmathresult 0.0
\pgfmathapproxequalto{e}{2.7182818}\pgfmathresult 1.0
```

### `\pgfmathgeneratepseudorandomnumber`

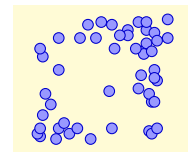
Определяет в `\pgfmathresult` псевдо-случайное число из промежутка  $(1, 2^{31} - 1)$ . Для вычислений используется линейный генератор, основанный на идеях Эриха Янки.

```
\pgfmathgeneratepseudorandomnumber\pgfmathresult 1959706271
```

### `\pgfmathrandominteger{<macro>}{<maximum>}{<minimum>}`

Определяет `<macro>` как псевдослучайно сгенерированное целое число из сегмента  $[<maximum>, <minimum>]$ .

```
\begin{pgfpicture}
\foreach \x in {1,...,50}{
\pgfmathrandominteger{\a}{1}{50}
\pgfmathrandominteger{\b}{1}{50}
\pgfpathcircle{\pgfpoint{+\a pt}{+\b pt}}{+2pt}
\color{blue!40!white}
\pgfsetstrokecolor{blue!80!black}
\pgfusepath{stroke, fill}
}
\end{pgfpicture}
```



### `\pgfmathdeclarerandomlist{<list name>}{<item-1>}{<item-2>...}`

Создает список элементов с именем `<list name>`.

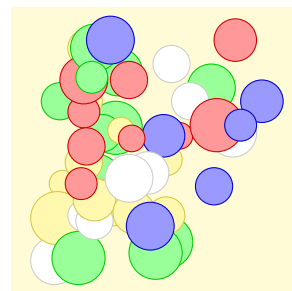
### `\pgfmathrandomitem{<macro>}{<list name>}`

Выбирает элемент из случайного списка `<list name>` и помещает его в `<macro>`.

```

\begin{pgfpicture}
\pgfmathdeclarerandomlist{color}{
  {red}{blue}{green}{yellow}{white}}
\foreach \a in {1,...,50}{
  \pgfmathrandominteger{\x}{1}{85}
  \pgfmathrandominteger{\y}{1}{85}
  \pgfmathrandominteger{\r}{5}{10}
  \pgfmathrandomitem{\c}{color}
  \pgfpathcircle{\pgfpoint{+\x pt}{+\y pt}}{+\r pt}
  \color{\c!40!white} \pgfsetstrokecolor{\c!80!black}
  \pgfusepath{stroke,fill}}
\end{pgfpicture}

```



`\pgfmathsetseed{<integer>}`

Явно устанавливает начальное значение для генератора псевдослучайного числа. По умолчанию используется значение `\time × \year`.

### 20.4.1 Преобразования чисел

Pgf обеспечивает ограниченную поддержку преобразованиям чисел из одной системы счисления в другую. В настоящее время числа должны быть положительными от 0 до  $2^{31} - 1$ , а основание системы счисления из диапазона 2–36. Все цифры для представления числа в системах счисления с основанием большим 10 являются алфавитными и могут быть заглавными или строчными буквами английского алфавита.

`\pgfmathbasetodec{<macro>}{<number>}{<base>}`

Устанавливает в `<macro>` результат преобразования числа `<number>` из системы счисления по основанию `<base>` в десятичное число.

```

\pgfmathbasetodec\mynumber{107f}{16} \mynumber 4223
\pgfmathbasetodec\mynumber{33FC}{20} \mynumber 25512

```

`\pgfmathdectobase{<macro>}{<number>}{<base>}`

Устанавливает в `<macro>` результат преобразования десятичного числа `<number>` в число по основанию `<base>` (в результате используются строчные буквы).

```

\pgfmathdectobase\mynumber{10712}{16} \mynumber 29d8
\pgfmathdectobase\mynumber{65535}{8} \mynumber 177777

```

`\pgfmathdectoBase{<macro>}{<number>}{<base>}`

Устанавливает в `<macro>` результат преобразования десятичного числа `<number>` в число по основанию `<base>` (в результате используются заглавные буквы).

```

\pgfmathdectoBase\mynumber{10712}{16} \mynumber 29D8
\pgfmathdectoBase\mynumber{65535}{20} \mynumber 83GF

```

`\pgfmathbasetobase{<macro>}{<number>}{<base-1>}{<base-2>}`

Устанавливает в `<macro>` результат преобразования числа `<number>` по основанию `<base-1>` в число по основанию `<base-2>` (в результате используются строчные буквы).

```

\pgfmathbasetobase\mynumber{1A7F}{16}{20} \mynumber gj3
\pgfmathbasetobase\mynumber{65535}{8}{16} \mynumber 6b5d

```



`\pgfmathbasetoBase{<macro>}{<number>}{<base-1>}{<base-2>}`

Устанавливает в `<macro>` результат преобразования числа `<number>` из системы счисления по основанию `<base-1>` в число по основанию `<base-2>` (в результате используются заглавные буквы).

```
\pgfmathbasetoBase\mynumber{1A7F}{16}{20} \mynumber      GJ3
\pgfmathbasetoBase\mynumber{65535}{8}{16} \mynumber      6B5D
```

`\pgfmathsetbasenumberlength{<integer>}`

Устанавливает число цифр в результате преобразования равным `<integer>`. Если результат преобразования имеет меньше цифр чем заявлено, дописываются нули.

```
\pgfmathsetbasenumberlength{8}                          00001111
\pgfmathdectobase\mynumber{15}{2} \mynumber
```

## 20.5 Настройка математического механизма

Иногда нужно определить функцию, которой нет в pgf. Иногда нужна большая точность вычислений или более эффективный алгоритм. В этих случаях можно добавить функцию в синтаксический анализатор или заменить текущий алгоритм. Математический механизм был спроектирован с учетом возможности таких изменений. Однако, отметим, что, хотя добавление новых операторов, возможно, это довольно хитрое программирование и рекомендуется только для «продвинутых» пользователей.

Чтобы добавить новую функцию к математическому механизму, может использоваться следующая команда:

`\pgfmathdeclarefunction{<function name>}{<number of arguments>} {<code>}`

Команда установит синтаксический анализатор так, чтобы он распознавал функцию с именем `<name>`. Имя функции может состоять из прописных и строчных букв, чисел или символа подчеркивания `_`. В соответствии с правилами многих языков программирования, имя функции не может начинаться с числа или содержать пробелы.

Параметр `<number arguments>` может быть любым положительным целым числом, нулем, или значением `...`, что указывает на переменное число аргументов. Pgf обрабатывает постоянные, типа  $\pi$  и  $e$ , как функции без аргументов. Функции с больше чем девятью аргументами или с переменным числом аргументами рассматриваются как специальные и обсуждаются ниже.

Результат выполнения кода `<code>` — установить правильное значение макроса `\pgfmathresult` (к результату вычисления без единиц измерения). Кроме того, функция не должна иметь побочных эффектов, то есть, она не должна изменять никаких глобальных значений. Как пример, рассмотрим создание новой функции `double`, которая принимает один аргумент, и возвращает его значение, умноженное на два.

```
\makeatletter
\pgfmathdeclarefunction{double}{1}{
  \begingroup \pgf@x=#1pt\relax
              \multiply\pgf@x by2\relax
              \pgfmathreturn\pgf@x
  \endgroup}
\makeatother
```

Для функций, объявленных с менее чем десятью аргументами, используется стандартное для Т<sub>Е</sub>X'a обозначение переменных — #1, ..., #9. Для функций с большим числом или с переменным числом аргументов, макросы определяются как функции с одним аргументом, который является списком, разделяемым запятыми.

Чтобы переопределить уже существующую функцию используется следующая команда, при этом нельзя изменить число аргументов существующей функции:

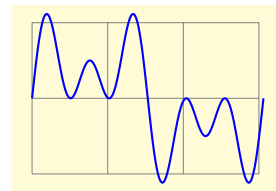
`\pgfmathredeclarefunction`{<function name>}{<algorithm code>}

Чтобы быстро создать простые специальные функции, предоставляется следующая опция, которая позволяет улучшить удобочитаемость кода, и особенно полезна в TikZ:

`/pgf/declare function`=<function definitions> (no default)

Использование этой опции лучше всего проиллюстрировать примером:

```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (3,2);
\draw [blue, thick, x=0.0085cm, y=1cm,
declare function={
sines(\t,\a,\b)=1 + 0.5*(sin(\t)+
sin(\t*\a)+sin(\t*\b));}]
plot [domain=0:360, samples=144, smooth] (\x,{sines(\x,3,5)});
\end{tikzpicture}
```

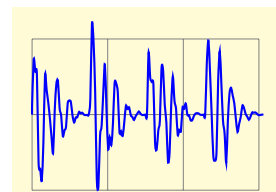


Каждое определение в <function definitions> имеет форму

`<name>(<arguments>)=<definition>;`

(точка с запятой в конце важна). Если определяется множество функций, для их разделения используется точка с запятой (не запятая). Имя функции <name> может быть любым, отсутствующим в текущей области видимости. Список аргументов <arguments> состоит из команд типа \x или \y (используя эту опцию, нельзя объявить функцию с переменным числом аргументов). Если функция не имеет аргументов, не должны использоваться круглые скобки. Определение <definitions> должно быть выражением, которое может быть проанализировано математическим механизмом и должно использовать команды, определенные в <arguments>. Когда определяется несколько функций, последующие могут обращаться к функциям, определенным ранее:

```
\begin{tikzpicture}[
declare function={excitation(\t,\w)=sin(\t*\w);
noise = rnd - 0.5;
source(\t) = excitation(\t,20) + noise;
filter(\t) = 1 - abs(sin(mod(\t, 90)));
speech(\t) = 1 + source(\t)*filter(\t);}]
\draw [help lines] (0,0) grid (3,2);
\draw [blue, thick, x=0.0085cm, y=1cm] (0,1) --
plot [domain=0:360, samples=144, smooth] \x,{speech(\x)};
\end{tikzpicture}
```



## 20.6 Печать числа

Pgf поддерживает печать числа в различных стилях и произвольной точности.

`\pgfmathprintnumber{<x>}`

Генерирует печатаемый вывод для (действительного) числа `<x>`. Число `<x>` анализируется, используя команду `\pgfmathfloatparsenumber`, которая позволяет получить произвольную точность. Числа набираются в математической моде, используя текущее множество параметров печати числа (см. ниже). Можно указать дополнительные аргументы, используя команду `\pgfmathprintnumber[<options>]{<x>}`.

`\pgfmathprintnumberto{<x>}{<\macro>}`

Возвращает получающееся число в `{<\macro>}` вместо того, чтобы непосредственно набирать его.

`/pgf/number format/fixed` (no value)

Заставляет команду `\pgfmathprintnumber` округлить число до фиксированного числа цифр после десятичной точки, отказываясь от всех конечных нулей.

```
\pgfkeys{/pgf/number format/.cd,fixed,precision=2}
\pgfmathprintnumber{4.568}\hspace{1em}
\pgfmathprintnumber{5e-04}\hspace{1em}
\pgfmathprintnumber{0.1}\hspace{1em}
\pgfmathprintnumber{24415.98123}\hspace{1em}
\pgfmathprintnumber{123456.12345}
```

4.57 0 0.1 24,415.98 123,456.12

`/pgf/number format/fixed zerofill={<boolean>}` (default true)

Позволяет или не позволяет заполнять нулями поля в конце числа до указанного числа разрядов.

```
\pgfkeys{/pgf/number format/.cd,fixed,fixed zerofill,precision=2}
\pgfmathprintnumber{4.568}\hspace{1em}
\pgfmathprintnumber{5e-04}\hspace{1em}
\pgfmathprintnumber{0.1}\hspace{1em}
\pgfmathprintnumber{24415.98123}\hspace{1em}
\pgfmathprintnumber{123456.12345}
```

4.57 0.00 0.10 24,415.98 123,456.12

Этот ключ эффективен для чисел, печатаемых в стилях `fixed` или `std` (для последнего, только если не был выбран научный формат числа).

```
\pgfkeys{/pgf/number format/.cd,std,fixed zerofill,precision=2}
\pgfmathprintnumber{4.568}\hspace{1em}
\pgfmathprintnumber{5e-05}\hspace{1em}
\pgfmathprintnumber{1}\hspace{1em}
```

4.57 5 · 10<sup>-5</sup> 1.00

`/pgf/number format/sci` (no value)

Заставляет команду `\pgfmathprintnumber` отображать числа в научном формате, который состоит из знака, мантиссы и показателя степени (с основанием 10). Мантисса округляется до желаемой точности (или до научной точности `sci precision`, см. ниже).

```
\pgfkeys{/pgf/number format/.cd,sci,precision=2}
\pgfmathprintnumber{4.568}\hspace{1em}
\pgfmathprintnumber{5e-04}\hspace{1em}
\pgfmathprintnumber{0.1}\hspace{1em}
\pgfmathprintnumber{24415.98123}\hspace{1em}
\pgfmathprintnumber{123456.12345}
```

4.57 · 10<sup>0</sup>   5 · 10<sup>-4</sup>   1 · 10<sup>-1</sup>   2.44 · 10<sup>4</sup>   1.23 · 10<sup>5</sup>

`/pgf/number format/sci zerofill={<boolean>}` (default true)

Позволяет или не позволяет заполнять нулями поля в конце числа до указанного числа разрядов в научном формате.

```
\pgfkeys{/pgf/number format/.cd,sci,sci zerofill,precision=2}
\pgfmathprintnumber{4.568}\hspace{1em}
\pgfmathprintnumber{5e-04}\hspace{1em}
\pgfmathprintnumber{0.1}\hspace{1em}
\pgfmathprintnumber{24415.98123}\hspace{1em}
\pgfmathprintnumber{123456.12345}
```

4.57 · 10<sup>0</sup>   5.00 · 10<sup>-4</sup>   1.00 · 10<sup>-1</sup>   2.44 · 10<sup>4</sup>   1.23 · 10<sup>5</sup>

Как и в случае опции `fixed zerofill`, эта опция дает эффект только для чисел в научном формате `sci` (или формате `std`, если был выбран научный формат).

`/pgf/number format/zerofill={<boolean>}` (style, default true)

Устанавливает форматы `fixed zerofill` и `sci zerofill` одновременно.

`/pgf/number format/std` (no value)

`/pgf/number format/std=<lower e>` (no default)

`/pgf/number format/std=<lower e>:<upper e>` (no default)

Заставляет команду `\pgfmathprintnumber` использовать стандартный алгоритм: выбирая или `fixed`, или `sci`, в зависимости от порядка величины. Пусть должно выводиться число  $n = s \cdot m \cdot 10^e$  ( $s$  — знак,  $m$  — мантисса,  $e$  — показатель степени), а  $p$  — текущая точность. Если  $-p/2 \leq e \leq 4$ , число отображается, используя формат `fixed`. Иначе, число отображается, используя научный формат `sci`.

```
\pgfkeys{/pgf/number format/.cd,std,precision=2}
\pgfmathprintnumber{4.568}\hspace{1em}
\pgfmathprintnumber{5e-04}\hspace{1em}
\pgfmathprintnumber{0.1}\hspace{1em}
\pgfmathprintnumber{24415.98123}\hspace{1em}
\pgfmathprintnumber{123456.12345}
```

4.57   5 · 10<sup>-4</sup>   0.1   24,415.98   1.23 · 10<sup>5</sup>

Параметры можно настроить, используя дополнительный целочисленный аргумент (ы): если  $\langle \text{lower } e \rangle \leq e \leq \langle \text{upper } e \rangle$ , число представляется в формате `fixed`, иначе в научном формате `sci`. Отметим, что для полезного результата степень  $\langle \text{lower } e \rangle$  должна быть отрицательной. Точность, используемая для научного формата, в случае необходимости может быть согласована с точностью `sci precision`.

`/pgf/number format/int detect` (no value)

Заставляет команду `\pgfmathprintnumber` автоматически распознавать целые числа. Если число целое, десятичная точка не отображается вообще. В противном случае выбирается научный формат.

```
\pgfkeys{/pgf/number format/.cd,int detect,precision=2}
\pgfmathprintnumber{15}\hspace{1em}
\pgfmathprintnumber{20}\hspace{1em}
\pgfmathprintnumber{20.4}\hspace{1em}
\pgfmathprintnumber{0.01}\hspace{1em}
\pgfmathprintnumber{0}
```

15 20 2.04 · 10<sup>1</sup> 1 · 10<sup>-2</sup> 0

`\pgfmathifisint`{<number constant>}{<true code>}{<false code>}

Команда, которая делает тот же самый выбор, что и `int detect`, но выполняет `<true code>`, если число `<number constant>` фактически целое число, и `<false code>` если нет. Как побочный эффект, макрос `\pgfretval` будет содержать анализируемое число или в целочисленном формате или в виде числа с плавающей точкой. Аргумент `<number constant>` анализируется `\pgfmathfloatparsenumber`.

```
15 \pgfmathifisint{15}{is an int: \pgfretval.}{is no int}\hspace{1em}
15.5 \pgfmathifisint{15.5}{is an int: \pgfretval.}{is no int}
```

15 is an int: 15. 15.5 is no int

`/pgf/number format/int trunc` (no value)

Усекает каждое число до целого числа (отказываясь от цифры после десятичной точки).

```
\pgfkeys{/pgf/number format/.cd,int trunc}
\pgfmathprintnumber{4.568}\hspace{1em}
\pgfmathprintnumber{5e-04}\hspace{1em}
\pgfmathprintnumber{0.1}\hspace{1em}
\pgfmathprintnumber{24415.98123}\hspace{1em}
\pgfmathprintnumber{123456.12345}
```

4 0 0 24,415 123,456

`/pgf/number format/trunc` (no value)

Отображает все числа как рациональные дроби.

```

\pgfkeys{/pgf/number format/frac}
\pgfmathprintnumber{0.333333333333333}\hspace{1em}
\pgfmathprintnumber{0.5}\hspace{1em}
\pgfmathprintnumber{2.13333333333325e-01}\hspace{1em}
\pgfmathprintnumber{0.12}\hspace{1em}
\pgfmathprintnumber{2.66666666666646e-02}\hspace{1em}
\pgfmathprintnumber{-1.33333333333334e-02}\hspace{1em}
\pgfmathprintnumber{7.20000000000000e-01}\hspace{1em}
\pgfmathprintnumber{6.66666666666667e-02}\hspace{1em}
\pgfmathprintnumber{1.33333333333333e-01}\hspace{1em}
\pgfmathprintnumber{-1.33333333333333e-02}\hspace{1em}
\pgfmathprintnumber{3.3333333}\hspace{1em}
\pgfmathprintnumber{1.2345}\hspace{1em}
\pgfmathprintnumber{1}\hspace{1em}
\pgfmathprintnumber{-6}

```

$$\frac{1}{3} \quad \frac{1}{2} \quad \frac{16}{75} \quad \frac{3}{25} \quad \frac{2}{75} \quad -\frac{1}{75} \quad \frac{18}{25} \quad \frac{1}{15} \quad \frac{2}{15} \quad -\frac{1}{75} \quad 3\frac{1}{3} \quad 1\frac{22657}{96620} \quad 1 \quad -6$$

`/pgf/number format/frac TeX`={<\macro>} (no default, initially `\frac`)

Позволяет использовать другую реализацию `\frac`.

`/pgf/number format/frac demon`=<int> (no default, initially empty)

Позволяет указать пользовательский знаменатель для представления дроби.

```

\pgfkeys{/pgf/number format/.cd,frac, frac denom=10}
\pgfmathprintnumber{0.1}\hspace{1em}
\pgfmathprintnumber{0.5}\hspace{1em}
\pgfmathprintnumber{1.2}\hspace{1em}
\pgfmathprintnumber{-0.6}\hspace{1em}
\pgfmathprintnumber{-1.4}\hspace{1em}

```

$$\frac{1}{10} \quad \frac{5}{10} \quad 1\frac{2}{10} \quad -\frac{6}{10} \quad -1\frac{4}{10}$$

`/pgf/number format/frac whole`=true|false (no default, initially true)

Определяет, должна ли целая часть числа помещаться перед дробной частью. В этом случае, дробная часть будет меньше 1. Используя `frac whole=false`, можно избежать выделения целой части числа.

```

\pgfkeys{/pgf/number format/.cd,frac, frac whole=false}
\pgfmathprintnumber{20.1}\hspace{1em}
\pgfmathprintnumber{5.5}\hspace{1em}
\pgfmathprintnumber{1.2}\hspace{1em}
\pgfmathprintnumber{-5.6}\hspace{1em}
\pgfmathprintnumber{-1.4}\hspace{1em}

```

$$\frac{201}{10} \quad \frac{11}{2} \quad \frac{6}{5} \quad -\frac{28}{5} \quad -\frac{7}{5}$$

`/pgf/number format/frac shift`={<integer>} (no default, initially 4)

В случае, если возникают проблемы из-за нестабильности, следует поэкспериментировать с разными значениями `frac shift` (при этом следует использовать библиотеку `fp` в преамбуле).

`/pgf/number format/precision`={<number>} (no default)

Устанавливает желаемую точность округления для любой операции отображения. В случае научного формата, это влияет на мантиссу.

`/pgf/number format/sci precision`={<number or empty>} (no default, initially empty)

Устанавливает желаемую точность округления только для научных стилей. Следует использовать `sci precision`={}, чтобы восстановить начальную конфигурацию (которая использует аргумент, определяющий точность всех стилей числа).

### 20.6.1 Изменение стилей отображения чисел

Всегда можно изменить стиль отображения чисел. Например, если используется стиль `fixed`, число округляется до указанной точности и отображается число с фиксированным числом десятичных знаков. Это же применимо к любому другому формату: сначала число округляется некоторой программой, чтобы получить правильные цифры, а затем, в соответствии с выбранным стилем, генерирует нужный TEX-код.

`/pgf/number format/set decimal separator`={<text>} (no default)

Назначает `<text>` как десятичный разделитель для любого числа с фиксированной точкой (включая мантиссу в научном формате). Следует использовать код

```
\pgfkeysgetvalue{/pgf/number format/set decimal separator}\value
```

чтобы получить в `\value` значение текущего разделителя.

`/pgf/number format/dec sep`={<text>} (style, no default)

Это только другое название для определения разделителя десятичного числа.

`/pgf/number format/set thousands separator`={<text>} (no default)

Назначает `<text>` как разделитель тысяч для любого числа с фиксированной точкой (включая мантиссу в научном формате).

```
\pgfkeys{/pgf/number format/.cd,
  fixed, fixed zerofill, precision=2,
  set thousands separator={}}
\pgfmathprintnumber{1234.56}
```

1234.56

```
\pgfkeys{/pgf/number format/.cd,
  fixed, fixed zerofill, precision=2,
  set thousands separator={}}
\pgfmathprintnumber{1234567890}
```

1234567890.00

```
\pgfkeys{/pgf/number format/.cd,
  fixed, fixed zerofill, precision=2,
  set thousands separator={.}}
\pgfmathprintnumber{1234567890}
```

1.234.567.890.00

```
\pgfkeys{/pgf/number format/.cd,
  fixed, fixed zerofill,precision=2,
  set thousands separator={,}
\pgfmathprintnumber{1234567890}
```

1, 234, 567, 890.00

```
\pgfkeys{/pgf/number format/.cd,
  fixed, fixed zerofill,precision=2,
  set thousands separator={{,}}}
\pgfmathprintnumber{1234567890}
```

1,234,567,890.00

Последний пример использует запятые и, по умолчанию, блокирует пробел после запятой. Следует использовать код

```
\pgfkeysgetvalue{/pgf/number format/set thousands separator}\value
```

чтобы получить в `\value` значение текущего разделителя.

```
/pgf/number format/1000 sep={<text>} (style, no default)
```

Это только другое название для установки разделителя тысяч.

```
/pgf/number format/min exponent for 1000 sep={<number>} (no default, initially 0)
```

Определяет наименьший показатель степени в той научной системе обозначений, которая требует указывать разделитель тысяч. Показатель степени — число цифр минус один, таким образом, `<number> = 4` использует разделитель тысяч, начиная с  $1e^4 = 10000$ . Значение 0 блокирует эту возможность (отрицательные числа игнорируются).

```
\pgfkeys{/pgf/number format/.cd,
  int detect,1000 sep={\,},
  min exponent for 1000 sep=0}
\pgfmathprintnumber{5000};
\pgfmathprintnumber{1000000}
```

5 000; 1 000 000

```
\pgfkeys{/pgf/number format/.cd,
  int detect, 1000 sep={\,},
  min exponent for 1000 sep=4}
\pgfmathprintnumber{1000};
\pgfmathprintnumber{5000}
```

1000; 5000

```
\pgfkeys{/pgf/number format/.cd,
  int detect, 1000 sep={\,},
  min exponent for 1000 sep=4}
\pgfmathprintnumber{10000};
\pgfmathprintnumber{1000000}
```

10 000; 1 000 000

```
/pgf/number format/use period (no value)
```

Предопределенный стиль, который устанавливает точку `'.'` как десятичный разделитель и запятую `'.'` как разделитель тысяч. Это стиль по умолчанию.



```
\pgfkeys{/pgf/number format/.cd,
  fixed,precision=2,use period}
\pgfmathprintnumber{12.3456}
```

12.35

```
\pgfkeys{/pgf/number format/.cd,
  fixed,precision=2,use period}
\pgfmathprintnumber{1234.56}
```

1,234.56

**/pgf/number format/use comma** (no value)

Предопределенный стиль, который устанавливает запятую ',' как десятичный разделитель и точку '.' как разделитель тысяч.

```
\pgfkeys{/pgf/number format/.cd,
  fixed,precision=2,use comma}
\pgfmathprintnumber{12.3456}
```

12,35

```
\pgfkeys{/pgf/number format/.cd,
  fixed,precision=2,use comma}
\pgfmathprintnumber{1234.56}
```

1.234,56

**/pgf//number format/skip 0.=**{<boolean>} (no default, initially false)

Определяет, должны ли числа типа 0.1 набираться как .1 или нет.

```
\pgfkeys{/pgf/number format/.cd,fixed,
  fixed zerofill,precision=2, skip 0.}
\pgfmathprintnumber{0.56}
```

.56

```
\pgfkeys{/pgf/number format/.cd,fixed,
  fixed zerofill,precision=2,skip 0.=false}
\pgfmathprintnumber{0.56}
```

0.56

**/pgf//number format/showpos=**{<boolean>} (no default, initially false)

Позволяет или блокирует показ знака "плюс" для неотрицательных чисел.

```
\pgfkeys{/pgf/number format/showpos}
\pgfmathprintnumber{12.345}
```

+12.35

```
\pgfkeys{/pgf/number format/showpos=false}
\pgfmathprintnumber{12.345}
```

12.35

```
\pgfkeys{/pgf/number format/.cd,showpos,sci}
\pgfmathprintnumber{12.345}
```

+1.23 · 10<sup>1</sup>

**/pgf//number format/print sign=**{<boolean>} (style, no default)

Стиль, который является просто псевдонимом для `showpos={<boolean>}`.

**/pgf//number format/sci 10e** (no value)

Использует представление вида  $m \cdot 10^e$  для любого числа, отображаемого в научном формате.

```
\pgfkeys{/pgf/number format/.cd,sci,sci 10e}
\pgfmathprintnumber{12.345}
```

1.23 · 10<sup>1</sup>

`/pgf/number format/sci 10e` (no value)

То же самое, что `sci 10e`.

`/pgf/number format/sci e` (no value)

Использует формат `'1e+0'`, который генерируется общим научным инструментом для любого числа, отображаемого в научном формате.

```
\pgfkeys{/pgf/number format/.cd,sci,sci e}
\pgfmathprintnumber{12.345}
```

1.23e+1

`/pgf/number format/sci E` (no value)

То же самое, но с заглавным `'E'`.

```
\pgfkeys{/pgf/number format/.cd,sci,sci E}
\pgfmathprintnumber{12.345}
```

1.23E+1

`/pgf/number format/sci subscript` (no value)

Набирает показатель степени как нижний индекс для любого числа, отображаемого в научном формате. Этот стиль требует меньше места.

```
\pgfkeys{/pgf/number format/.cd,sci,sci subscript}
\pgfmathprintnumber{12.345}
```

1.23<sub>1</sub>

`/pgf/number format/sci subscript` (no value)

Набирает показатель степени как верхний индекс для любого числа, отображаемого в научном формате. Этот стиль требует меньше места.

```
\pgfkeys{/pgf/number format/.cd,sci,sci superscript}
\pgfmathprintnumber{12.345}
```

1.23<sup>1</sup>

`/pgf/number format/sci generic={<keys>}` (no default)

Позволяет определить собственный стиль числа для научного формата. Параметр `<keys>` может быть одним из следующих ключей:

`mantissa sep={<text>}` (no default, initially empty)

Обеспечивает разделитель между мантиссой и показателем степени. Это может быть, например, `\cdot`.

`exponent={<text>}` (no default, initially empty)

Обеспечивает текст, чтобы сформатировать показатель степени. Фактический показатель степени доступен как аргумент `#1` (см. ниже).

```
\pgfkeys{/pgf/number format/.cd,sci,
  sci generic={mantissa sep=\times,
  exponent={10^{#1}}}}
\pgfmathprintnumber{12.345};
\pgfmathprintnumber{0.00012345}
```

1.23 × 10<sup>1</sup>; 1.23 × 10<sup>-4</sup>

Параметр `<keys>` может зависеть от трех параметров, а именно: `#1`, который является показателем степени, `#2`, который содержит признак числа с плавающей точкой, `#3` — (необработанная и неформатированная) мантисса.

Отметим, что опция `sci generic` не может использоваться для изменения вида числа с фиксированной точкой, и не может использоваться для того, чтобы отформатировать мантиссу (которая набирается подобно числу с фиксированной точкой). Следует использовать опции `dec sep`, `1000 sep` и `print sign`, чтобы настроить мантиссу.

```
/pgf/number format/@dec sep mark={<text>} (no default)
```

Прямо перед местом, где располагается десятичный разделитель, вставляет текст `{<text>}`, который будет вставляться, даже если нет десятичного разделителя. Текст предназначен в качестве метки-заполнителя для вспомогательных программ, позволяя им найти позицию выравнивания. Этот ключ никогда не должен использоваться, чтобы изменить десятичный разделитель! Должен использоваться ключ `dec sep`.

```
/pgf/number format/@sci exponent mark={<text>} (no default)
```

Вставляет `{<text>}` перед показателями степени в научном формате. Текст предназначен в качестве метки-заполнителя для вспомогательных программ, позволяя им найти позицию выравнивания. Этот ключ никогда не должен использоваться для изменения показателя степени!

```
/pgf/number format/assume math mode={<boolean>} (default true)
```

Если не нужна проверка на математический режим, установить `true`. Начальная установка проверяет был ли математический режим активным, используя команду `\pgfutilensuremath` для каждого финального числа. Если знаете, что математический режим активен, следует установить `assume math mode=true`. В этом случае, финальное число будет набираться «как есть», и дальнейшая проверка выполняться не будет.

```
/pgf/number format/verbatim (style, no value)
```

Стиль, который определяет печать числа так, чтобы произвести его дословно, поэтому оно не должно содержать макросов `TeX`'а.

```
\pgfkeys{/pgf/fpu,/pgf/number format/.cd,sci,verbatim}
\pgfmathprintnumber{12.345};
\pgfmathprintnumber{0.00012345};
\pgfmathparse{exp(15)}
\pgfmathprintnumber{\pgfmathresult}
```

1.23e1; 1.23e-4; 3.27e6

Стиль вновь устанавливает ключи `1000 sep`, `dec sep`, `print sign`, `skip 0.` и устанавливает `assume math mode`. Кроме того, он устанавливает формат `sci generic` для дословного вывода научного числа. Однако, будет все еще признавать опции `precision`, `fixed zerofill`, `sci zerofill` и полные стили `fixed`, `sci`, `int detect` (и их варианты). Это может оказаться полезным, если надо написать выходные файлы.

# Глава 21

## Математические библиотеки

В дополнение к общему математическому механизму `pgf`, существуют две математические библиотеки. Первая библиотека `fixedpointarithmetic` обеспечивает интерфейс к пакетом `fp` среды `LATEX` (он должен быть загружен перед загрузкой библиотеки!) для выполнения арифметических операций над числами с фиксированной точкой. Вторая библиотека `fp` (`Floating Point Unit`) позволяет вести полный объем вычислений с научными данными, представленными числами с плавающей точкой.

### 21.1 Библиотека `fixedpointarithmetic`

Точность арифметических операций, предоставляемых `pgf`, довольно низкая, особенно для выражений, содержащих много операций. Область вычисляемых значений мала:  $\pm 16383.99999$ , но есть хороший синтаксический анализатор. Наоборот, пакет `fp` имеет разумно высокую точность, и может выполнять вычисления в широком диапазоне значений (приблизительно  $\pm 9.999 \times 10^{17}$ ), но сравнительно медленно и не очень гибко, особенно в части выполнения синтаксического анализа. Библиотека `fixedpointarithmetic` комбинирует два достоинства: гибкость синтаксического анализатора `pgf` с высокой точностью вычислений `fp`, но полная скорость вычислений будет меньше, чем при использовании только математического механизма `pgf`: это цена за вычислительную мощь `fp` (см. детали в [1, глава 35]).

Для выполнения арифметических операций с числами с фиксированной точкой, используя пакет `fp` в среде `pgf&TikZ`, существуют следующие ключи.

`/pgf/fixed point arithmetic=<options>` (no default)

Этот ключ установит ключ пути равным `/pgf/fixed point`, и выполнит `<options>`. Затем он установит необходимые команды так, чтобы синтаксический анализатор `pgf` смог использовать `fp` для выполнения вычислений. Лучший способ использовать этот ключ — использовать его как параметр области видимости (`scope`) или рисунка. Это означает, что пакет `fp` не всегда должен использоваться и `pgf` может использовать в других ситуациях собственный математический механизм, что может привести к существенному уменьшению времени на обработку документа.

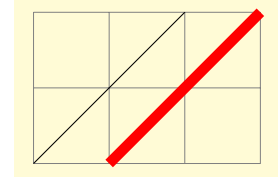
В настоящее время есть всего несколько ключей, допустимых для `<options>`:

`/pgf/fixed point/scale results=<factor>` (no default)

Как сказано выше, `fp` может обрабатывать больший числовой диапазон, чем `pgf` и `TikZ`. Чтобы использовать результаты от `fp` в `{pgfpicture}` или `{tikzpicture}` их нужно масштабировать. Когда используется этот ключ, `pgf` масштабирует результат всех вычислений, используя `<factor>`. Поскольку не желательно масштабировать резуль-

тат каждого выражения, масштабирование будет производиться, только если в начале выражения используется специальный префикс \*. Тогда выражение вычисляется и результат умножается на `<factor>`.

```
\begin{tikzpicture}
[fixed point arithmetic={scale results=10^-6}]
\draw [help lines] grid (3,2);
\draw (0,0) -- (2,2);
\draw [color=red, line width=4pt]
(*1.0e6,0) -- (*3.0e6,*2.0e6);
\end{tikzpicture}
```



Частный случай масштабирования включает в себя графики данных, содержащие большие числа из файлов. Можно выполнить предварительную обработку файла, обычно используя приложение, которое генерирует данные, чтобы разместить их в столбец с префиксом \*, или выполнить масштабирование как часть процесса вычисления. Иногда желательно иметь данные для графика так, как они есть, так что потребуются два файла, один исходный для составления графика, и один переработанный. Такая дополнительная работа может оказаться нежелательной. Поэтому поставляются следующие ключи:

```
/pgf/fixed point/scale file plot x=<factor> (no default)
/pgf/fixed point/scale file plot y=<factor> (no default)
/pgf/fixed point/scale file plot z=<factor> (no default)
```

Ключи масштабируют 1-ый, 2-ой и 3-ий, соответственно, столбец данных, читаемых из файла, до составления графика (они не зависят от ключа `scale results`).

## 21.2 Библиотека fpu

Библиотека `fpu` поддерживает арифметические операции над числами с плавающей точкой, широко используемыми в научных вычислениях. Допустимый диапазон данных:  $-1 \cdot 10^{324}, \dots, 1 \cdot 10^{324}$ . Абсолютная величина самого маленького числа, большего нуля,  $1 \cdot 10^{-324}$ . Относительная точность сопроцессора для операций с плавающей точкой — по крайней мере  $1 \cdot 10^{-4}$ , хотя такие операции, как сложение, имеют относительную точность  $1 \cdot 10^{-6}$ . Как отмечает Христиан Фейерзангер, автор описания этой библиотеки в руководстве [1], библиотека не проверялась вместе со *всеми* операциями рисования.

```
/pgf/fpu=<boolean> (default true)
```

Ключ устанавливает или не устанавливает блок FPU. При установке заменяются все подпрограммы стандартной математики на подпрограммы для операций с плавающей точкой: `\pgfmathadd` на `\pgfmathfloatadd` и так далее. Кроме того, любое число анализируется анализатором `\pgfmathfloatparsenumber`.

```
\pgfkeys{/pgf/fpu} \pgfmathparse{1+1}\pgfmathresult 1Y2.0e0]
```

Блок FPU использует для чисел представление низкого уровня, состоящее из флагов, мантиссы и экспоненты. Чтобы избежать ненужных форматных преобразований, `\pgfmathresult` обычно содержит некое загадочное число. В зависимости от контекста, результат, возможно, должен преобразовываться в нечто, что можно обработать в `pgf`.

Если использовать `fpu=false`, то блок FPU отключается. Многократные вызовы `fpu=true` или `fpu=false` не ведут к неприятностям. Если pgf-библиотека `fixedpointarithmetic` будет активизирована после библиотеки `fpu`, блок FPU отключается автоматически.

`/pgf/fpu/output format=float/sci/fixed` (no default, initially float)

Позволяет изменить формат числа, назначаемый блоком FPU для `\pgfmathresult`. Предопределенный выбор `float` использует формат низкого уровня для числа с плавающей точкой. Он полезен для дальнейшей обработки чисел в любой библиотеке. Выбор `sci` возвращает числа в формате `<mantissa>e<exponent>`, что затрудняет последующие вычисления. Выбор `fixed` возвращает число с фиксированной точкой и обеспечивает самую высокую совместимость с механизмами pgf. Этот вывод активизируется автоматически в случае, если блок FPU масштабирует результаты.

```
\pgfkeys{/pgf/fpu,/pgf/fpu/output format=float}
\pgfmathparse{exp(50)*42}\pgfmathresult
```

1Y2.17765411e23]

```
\pgfkeys{/pgf/fpu,/pgf/fpu/output format=sci}
\pgfmathparse{4.22e-8^-2}\pgfmathresult
```

5.6154816e14

```
\pgfkeys{/pgf/fpu,/pgf/fpu/output format=fixed}
\pgfmathparse{sqrt(1e-12)}\pgfmathresult
```

0.000000999985

`/pgf/fpu/scale results=<scale>` (no default)

Позволяет полуавтоматически масштабировать результат. Установка этой опции дает два следствия: во-первых, формат вывода для любых вычислений будет установлен как `fixed` (принимаемые результаты будут обрабатываться ядром pgf); во-вторых, любое выражение, которое начинается со звездочки `*`, будет умножаться на `<scale>`.

`/pgf/fpu/scale file plot x=<scale>` (no default)

`/pgf/fpu/scale file plot y=<scale>` (no default)

`/pgf/fpu/scale file plot z=<scale>` (no default)

Ключи исправляют pgf-команду `plot file` так, чтобы автоматически масштабировалась на `{<scale>}` одна координата.

`\pgflibraryfpuifactive{<true-code>}{<false-code>}`

Команда используется для того, чтобы выполнить некоторый код, в зависимости от того, был ли активирован или нет блок FPU.

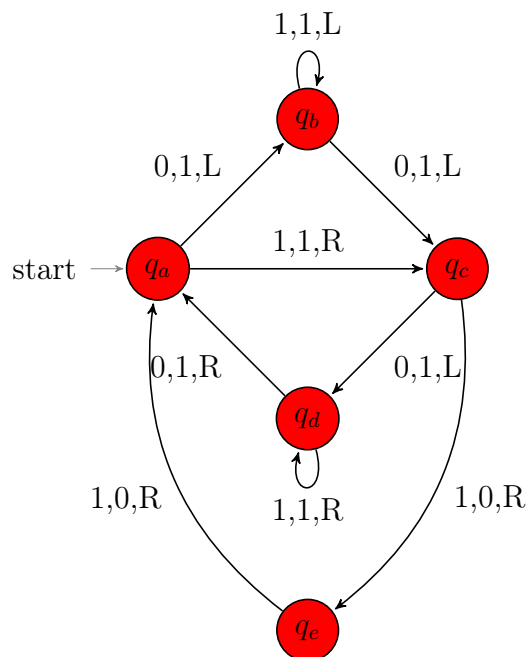
Отметим в заключение, что между библиотеками `fp+fixedpointarithmetic` и `fpu` есть существенные различия:

- `fpu` поддерживает полный диапазон чисел с двойной точностью в формате IEEE, в то время как `fp` охватывает только числа диапазона  $pm1 \cdot 10^{17}$ ;
- `fpu` обеспечивает однородную относительную точность приблизительно в 4–5 правильных цифр. Библиотека `fp` имеет абсолютную точность, которая имеет место во многих случаях, но приводит к ошибкам в конце диапазона данных (так происходит с каждой подпрограммой этой библиотеки);
- `fpu` потенциально быстрее `fp`, поскольку имеет доступ к быстрым операциям мантиссы, используя математические возможности pgf.

Подробную справочную информацию о командах, операциях округления, особенностях математических операций можно найти в [1, 36.4].

# Литература

- [1] Tantau T. The TikZ and PGF Packages. Manual for version 2.10 — Institut für Theoretische Informatik, Universität zu Lübeck. — October 25, 2010 (<http://sourceforge.net/projects/pgf>).
- [2] Кнут Д. Е. Все про  $\text{T}_\text{E}\text{X}$ . — Протвино: РД $\text{T}_\text{E}\text{X}$ , 1993. — 592 с.: ил.
- [3] Гуссенс, М. Путеводитель по пакету  $\text{L}_\text{A}\text{T}_\text{E}\text{X}$  и его расширению  $\text{L}_\text{A}\text{T}_\text{E}\text{X}_2_\epsilon$ . Пер. с англ. — М.: «Мир», 1999. — 606 с., илл.
- [4] Львовский С. М. Набор и вёрстка в системе  $\text{L}_\text{A}\text{T}_\text{E}\text{X}$ , 3-е изд., испр. и доп. — М.: МЦНМО, 2003. — 448 с.
- [5] Котельников И. А., Чеботаев П. З.  $\text{L}_\text{A}\text{T}_\text{E}\text{X}$  по-русски, 3-е изд., перераб. и доп. — Новосибирск: «Сибирский хронограф», 2004. — 496 с.



# Оглавление

<b>1</b>	<b>Введение</b>	<b>3</b>
<b>I</b>	<b>Примеры и рекомендации</b>	<b>6</b>
<b>2</b>	<b>Определение синуса и косинуса</b>	<b>8</b>
2.1	Постановка задачи . . . . .	8
2.2	Конструирование прямого пути . . . . .	9
2.3	Конструирование кривой . . . . .	9
2.4	Конструирование окружности и круга . . . . .	10
2.5	Конструирование прямоугольника . . . . .	11
2.6	Конструирование сетки . . . . .	11
2.7	Понятие о стиле . . . . .	12
2.8	Опции рисунка . . . . .	12
2.9	Конструирование дуги . . . . .	13
2.10	Отсечение . . . . .	13
2.11	Конструирование параболы и синусоиды . . . . .	14
2.12	Области и границы . . . . .	14
2.13	Растушевывание . . . . .	15
2.14	Определение координат . . . . .	16
2.15	Построение стрелок . . . . .	18
2.16	Окружение {scope} . . . . .	19
2.17	Преобразования . . . . .	20
2.18	Итерации и циклы . . . . .	21
2.19	Добавление текста в рисунок . . . . .	22
2.20	Полный код примера . . . . .	25
<b>3</b>	<b>Сеть Петри</b>	<b>27</b>
3.1	Введение в узлы . . . . .	27
3.2	Использование стилей . . . . .	29
3.3	Размеры узлов . . . . .	29
3.4	Именованые узлы . . . . .	30
3.5	Относительное размещение узлов . . . . .	31
3.6	Добавление текста рядом с узлами . . . . .	32
3.7	Соединение узлов . . . . .	33
3.8	Метки рядом с дугами . . . . .	35
3.9	Волнистые линии и многострочный текст . . . . .	36
3.10	Фоновые прямоугольники . . . . .	37
3.11	Полный код примера . . . . .	38



<b>4</b>	<b>Элементы геометрии Евклида</b>	<b>40</b>
4.1	Начала. Книга I. Предложение 1 . . . . .	40
4.1.1	Построение прямой . . . . .	40
4.1.2	Построение окружности . . . . .	41
4.1.3	Пересечение окружностей . . . . .	43
4.1.4	Полный код примера . . . . .	44
4.2	Начала. Книга I. Предложение 2 . . . . .	45
4.2.1	Другое построение равностороннего треугольника . . . . .	46
4.2.2	Пересечение прямой и окружности . . . . .	47
4.2.3	Полный код примера . . . . .	48
<b>5</b>	<b>Цепочки и матрицы</b>	<b>50</b>
5.1	Стиль узлов . . . . .	50
5.2	Выравнивание узлов позиционированием . . . . .	52
5.3	Выравнивание узлов, используя матрицы . . . . .	55
5.4	Использование библиотеки цепочек . . . . .	56
5.4.1	Создание простой цепочки . . . . .	56
5.4.2	Ветвление и связывание в цепочке . . . . .	57
5.4.3	Связи в цепочках с уже созданными узлами . . . . .	59
5.4.4	Совместное использование матриц и цепочек . . . . .	60
<b>6</b>	<b>Дерево для расписания лекций</b>	<b>62</b>
6.1	Постановка задачи . . . . .	62
6.2	Понятие дерева . . . . .	62
6.3	Создание карты курса . . . . .	66
6.4	Добавление аннотаций к лекциям . . . . .	71
6.5	Добавление фона . . . . .	73
6.6	Создание календаря . . . . .	74
6.7	Полный код карты курса . . . . .	77
<b>7</b>	<b>Рекомендации по графике</b>	<b>81</b>
7.1	Планирование времени . . . . .	81
7.2	Процесс создания графического объекта . . . . .	82
7.3	Соединение графики и текста . . . . .	82
7.4	Согласованность между графикой и текстом . . . . .	83
7.5	Метки в графических объектах . . . . .	84
7.6	Графики и диаграммы . . . . .	84
7.7	Внимание и отвлечение . . . . .	87
<b>II</b>	<b>Интерфейс TikZ</b>	<b>89</b>
<b>8</b>	<b>Принципы проектирования</b>	<b>91</b>
8.1	Синтаксис для определения точек . . . . .	91
8.2	Синтаксис для спецификации пути . . . . .	92
8.3	Действия на путях . . . . .	92
8.4	Синтаксис «ключ-значение» для параметров . . . . .	92
8.5	Специальный синтаксис для узлов . . . . .	93
8.6	Специальный синтаксис для деревьев . . . . .	93
8.7	Группировка графических параметров . . . . .	94

8.8	Преобразования системы координат . . . . .	95
<b>9</b>	<b>Иерархические структуры</b>	<b>96</b>
9.1	Создание рисунка . . . . .	96
9.2	Структурирование рисунка, используя <code>scope</code> . . . . .	98
9.3	Использование графических опций . . . . .	99
<b>10</b>	<b>Определение координат</b>	<b>102</b>
10.1	Системы координат . . . . .	103
10.2	Координаты точек пересечения . . . . .	110
10.3	Относительные координаты . . . . .	113
10.4	Вычисление координат . . . . .	114
<b>11</b>	<b>Синтаксис спецификаций пути</b>	<b>118</b>
11.1	Операция перемещения . . . . .	120
11.2	Операции построения линий . . . . .	120
11.3	Кривые Безье . . . . .	121
11.4	Операция цикла . . . . .	121
11.5	Операция <code>rectangle</code> . . . . .	122
11.6	Скругление углов . . . . .	122
11.7	Круг и эллипс . . . . .	123
11.8	Операция <code>arc</code> . . . . .	125
11.9	Операция <code>grid</code> . . . . .	125
11.10	Операция <code>parabola</code> . . . . .	127
11.11	Операции <code>sin</code> и <code>cos</code> . . . . .	128
11.12	Операция <code>plot</code> . . . . .	129
11.13	Операция <code>to</code> . . . . .	129
11.14	Операция <code>let</code> . . . . .	131
11.15	Операция <code>scope</code> . . . . .	133
11.16	Операции <code>node</code> и <code>edge</code> . . . . .	133
11.17	Особые PGF-операции . . . . .	133
<b>12</b>	<b>Действия на пути</b>	<b>134</b>
12.1	Определение цвета . . . . .	134
12.2	Прорисовка пути . . . . .	135
12.2.1	Толщина, завершение, соединение . . . . .	135
12.2.2	Пунктир . . . . .	136
12.2.3	Непрозрачность . . . . .	138
12.2.4	Стрелки . . . . .	138
12.2.5	Двойные линии и граничные линии . . . . .	140
12.3	Заполнение пути . . . . .	141
12.3.1	Заполнение пути шаблоном . . . . .	142
12.3.2	Правила для внутренней точки . . . . .	143
12.4	Заполнение пути произвольным рисунком . . . . .	144
12.5	Растушевывание пути . . . . .	146
12.6	Установка ограничивающего прямоугольника . . . . .	147
12.7	Отсечение и мягкое обесцвечивание . . . . .	149
12.8	Выполнение нескольких операций на пути . . . . .	151
12.9	Декорирование и трансформация пути . . . . .	153

<b>13 Узлы и дуги</b>	<b>154</b>
13.1 Узлы и их формы . . . . .	154
13.1.1 Предопределенные формы . . . . .	156
13.1.2 Общие опции узлов . . . . .	157
13.2 Многослойные узлы . . . . .	160
13.3 Текст узла . . . . .	161
13.3.1 Текстовые параметры: цвет и прозрачность . . . . .	161
13.3.2 Текстовые параметры: шрифт . . . . .	161
13.3.3 Текстовые параметры: выравнивание и ширина текста . . . . .	161
13.3.4 Текстовые параметры: высота и глубина текста . . . . .	164
13.4 Позиционирование узлов . . . . .	164
13.4.1 Позиционирование узлов по якорям . . . . .	164
13.4.2 Основные опции для размещения узлов . . . . .	165
13.5 Установка узлов по набору координат . . . . .	166
13.6 Трансформирование узлов . . . . .	167
13.7 Явное размещение узлов на прямой и кривой . . . . .	168
13.8 Неявное размещение узлов на прямой и кривой . . . . .	171
13.9 Опции <code>label</code> и <code>pin</code> . . . . .	172
13.10Связи, использующие узлы как координаты . . . . .	175
13.11Связи, использующие операцию <code>edge</code> . . . . .	176
13.12Ссылки на узлы вне текущего рисунка . . . . .	177
13.12.1 Ссылка на узел в другом рисунке . . . . .	177
13.12.2 Ссылка на узел текущей страницы . . . . .	178
13.13Отложенный код и отложенные опции . . . . .	179
<b>14 Матрицы и выравнивание</b>	<b>180</b>
14.1 Матрицы как узлы . . . . .	180
14.2 Изображение ячеек . . . . .	181
14.2.1 Выравнивание ячеек . . . . .	181
14.2.2 Корректировка пробелов между столбцами и строками . . . . .	183
14.2.3 Стили и опции ячеек . . . . .	185
14.3 Якоря матрицы . . . . .	187
14.4 Определение активных символов . . . . .	188
14.5 Примеры . . . . .	188
<b>15 Создание растущих деревьев</b>	<b>192</b>
15.1 Операция <code>child</code> . . . . .	192
15.2 Дочерние пути и дочерние узлы . . . . .	193
15.3 Именованние дочерних узлов . . . . .	194
15.4 Опции деревьев и узлов . . . . .	195
15.5 Размещение дочерних узлов . . . . .	197
15.5.1 Основная идея . . . . .	197
15.5.2 Функция роста дерева по умолчанию . . . . .	197
15.5.3 Отсутствие дочерних узлов . . . . .	200
15.5.4 Собственные функции роста дерева . . . . .	201
15.6 Дуги из корневого узла . . . . .	201

<b>16</b>	<b>Графики функций</b>	<b>204</b>
16.1	TikZ и создание графиков . . . . .	204
16.2	Операция <code>plot</code> . . . . .	204
16.3	График по явно заданному набору точек . . . . .	205
16.4	График по набору точек из файла . . . . .	205
16.5	График по точкам, вычисляемым на лету . . . . .	206
16.6	Построение графика, используя <code>gnuplot</code> . . . . .	208
16.7	Размещение меток на графике . . . . .	210
16.8	Типы графиков . . . . .	211
<b>17</b>	<b>Прозрачность</b>	<b>217</b>
17.1	Определение однородной непрозрачности . . . . .	217
17.2	Постепенное изменение цвета ( <code>fading</code> ) . . . . .	220
17.2.1	Создание фединга . . . . .	220
17.2.2	Фединг в пути . . . . .	222
17.2.3	Фединг в области видимости . . . . .	224
17.3	Группы прозрачности . . . . .	226
<b>18</b>	<b>Декорирование пути</b>	<b>228</b>
18.1	Основы декорирования . . . . .	228
18.2	Декорирование, использующее команду <code>decorate</code> . . . . .	231
18.3	Декорирование пути в целом . . . . .	232
18.4	Корректировка декораций . . . . .	234
<b>19</b>	<b>Преобразования</b>	<b>237</b>
19.1	Различные системы координат . . . . .	237
19.2	Системы координат $xu$ и $xuz$ . . . . .	237
19.3	Координатные преобразования . . . . .	239
19.4	Преобразования холста . . . . .	243
<b>III</b>	<b>Математика в <code>pgf</code></b>	<b>244</b>
<b>20</b>	<b>Механизм математики</b>	<b>246</b>
20.1	Принципы расчета . . . . .	246
20.1.1	Уровни математического механизма . . . . .	246
20.1.2	Эффективность и точность математики . . . . .	246
20.2	Вычисление математических выражений . . . . .	246
20.2.1	Команды для анализа выражений . . . . .	247
20.3	Синтаксис математических выражений . . . . .	248
20.3.1	Операторы . . . . .	248
20.3.2	Функции . . . . .	250
20.4	Дополнительные математические команды . . . . .	255
20.4.1	Преобразования чисел . . . . .	256
20.5	Настройка математического механизма . . . . .	257
20.6	Печать числа . . . . .	259
20.6.1	Изменение стилей отображения чисел . . . . .	263

<i>ОГЛАВЛЕНИЕ</i>	277
<b>21 Математические библиотеки</b>	<b>268</b>
21.1 Библиотека <code>fixedpointarithmetic</code> . . . . .	268
21.2 Библиотека <code>fpu</code> . . . . .	269
<b>Литература</b>	<b>271</b>