Canvas, game loop, CD

Математические модели в Game Design

Пшеничный Евгений
@pshenichnyy

Павел Оганесян
@PavelOganesyan

Agenda

Введение

Html и события

Canvas

Game Loop

Collision Detection D

Мат. Модели в Game Design

1. QuadTree and Collision detection
2. Space Invaders / Tetris
3. Стиринги
4. Игровой баланс
5. Процедурная генерация уровня
6. NFT 

x3 SCORE: 0 RECORD: 0

CORE: 0 HI-SCORE: 0 E: 60

100% 100% 100% 100%

AMMO


PHASER

Press ESC to return

Подземелья наводнили желе-мутанты,
объявлена награда за их истребление!
Сразитесь ли вы с соперником за звание
самого успешного охотника на желе?

Начать игру

ЧЕЛОВЕЧЕСТВО УНИЧТОЖЕНО ИНОПЛАНЕТНЫМИ ЗАХВАТЧИКАМИ
ТЫ - БЕЗДУШНАЯ МАШИНА
ИХ КРОВЬ - ТВОЁ ТОПЛИВО

НАЧАТЬ ИГРУ

Жёлтая зона - зона боевых дейс
Зеленая зона - активная зона д
Красная зона - ваша статистика

Ваша задача: не подпускать их

WASD - перемещение
Прицеливание мышью
Стрельба - левая кнопка мыши
Клавиша C - включить/выключ

HTML w/ Canvas

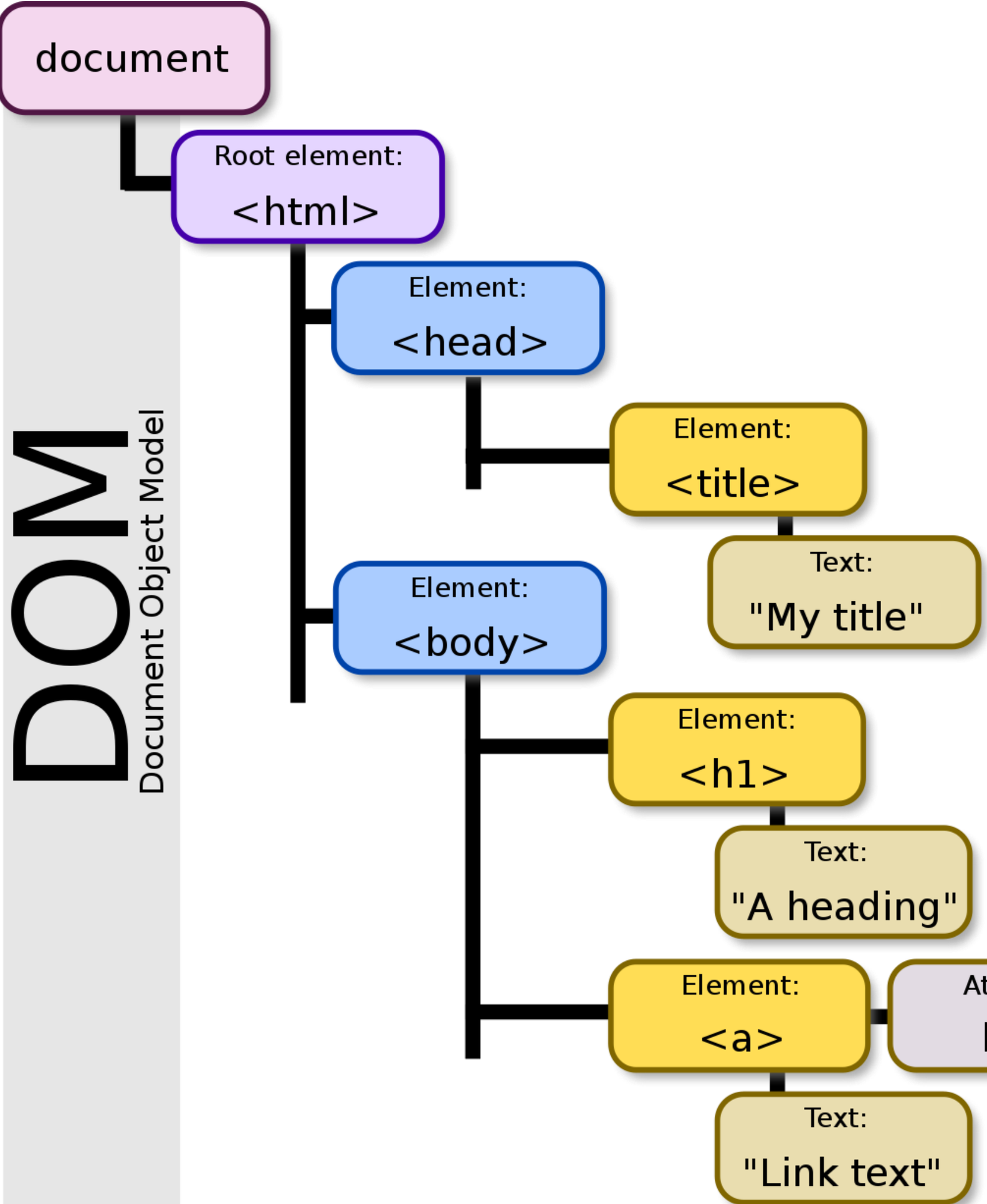
index.html > ...

```
1   <!DOCTYPE html>
2   <html>
3   <head>
4     <title>Canvas demo</title>
5     <style>
6       * { margin:0; padding:0; }
7       html, body { width:100%; height:100%;}
8       canvas { background: ■black; }
9     </style>
10  </head>
11
12  <body>
13    <canvas id='cnvs'></canvas>
14    <script src="./index.js"></script>
15  </body>
16  </html>
```

DOM

Объектная Модель Документа (**DOM**) – это программный интерфейс (API) для HTML документов.

DOM предоставляет структурированное представление документа и определяет то, как эта структура может быть доступна из программ, которые могут изменять содержимое, стиль и структуру документа.

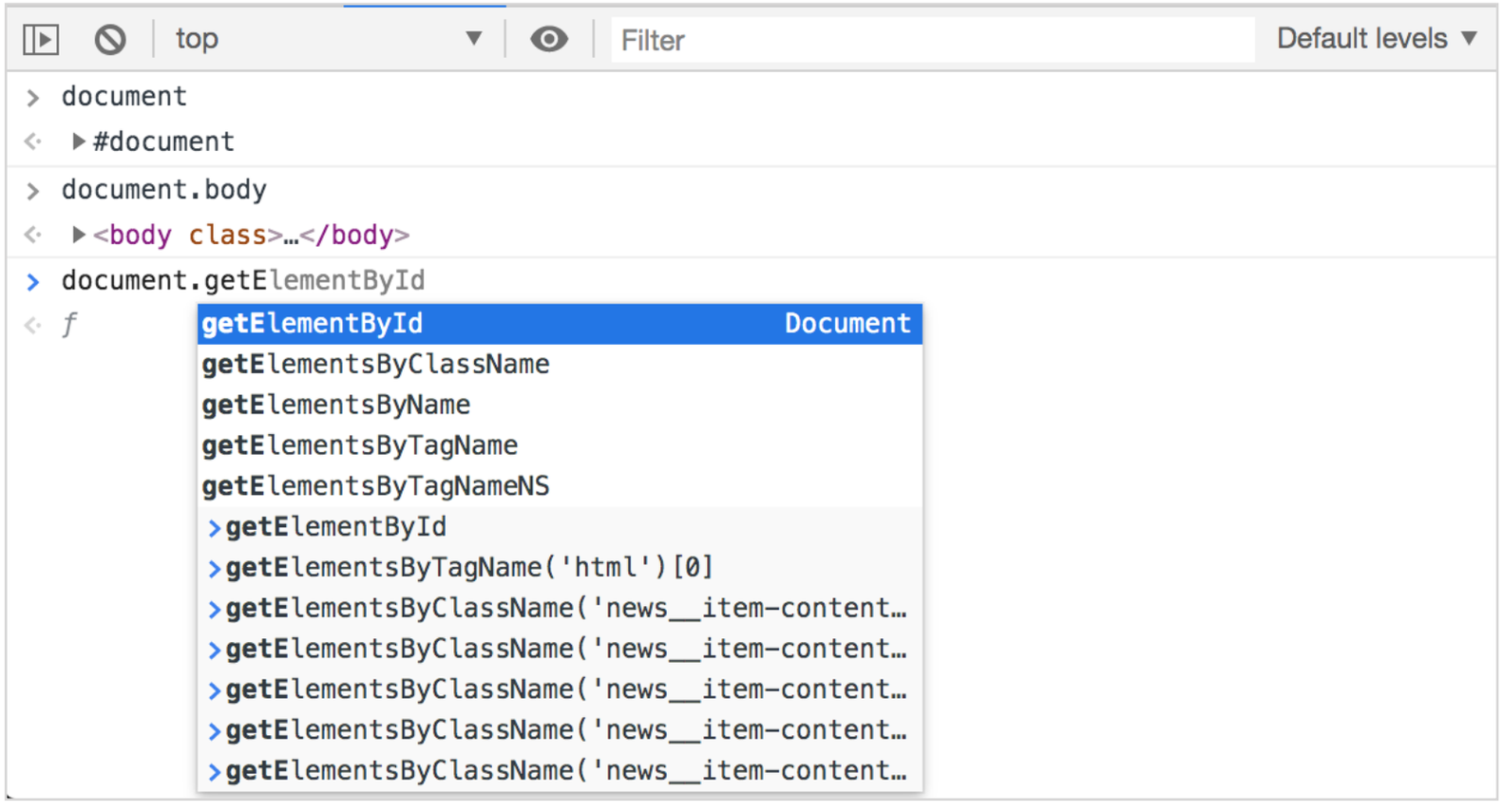


Каждый **HTML**-тег страницы является объектом.

Все эти объекты доступны из JavaScript, их можно использовать для изменения страницы.

document

Все операции с DOM начинаются с объекта document. Это главная «точка входа» в DOM.



getElementById

Самый простой способ получить доступ к элементу HTML – это использовать атрибут ***id*** элемента.

HTML:

```
<p id="myId">My first web page</p>
```

JavaScript:

```
const el = document.getElementById('myId')
```

Браузерные события

Событие – это сигнал от браузера о том, что что-то произошло. Все DOM-узлы предоставляют возможность подписаться на их события.

- click – происходит по клику на элемент.
- mouseover/mouseout – когда курсор мыши попадает или покидает элемент.
- mousedown/mouseup – когда нажали / отжали кнопку мыши на элементе.
- mousemove – при движении курсора мыши.
- contextmenu – происходит по клику правой кнопкой мыши.

Обработчики событий

Событию можно назначить обработчик, то есть функцию, которая сработает, как только событие произошло.

Именно благодаря обработчикам JavaScript-код может реагировать на действия пользователя.

Есть несколько способов назначить событию обработчик. Мы будем использовать addEventListener

addEventListener

```
const el = document.getElementById('myId')

el.addEventListener("click", onClick)

function onClick() {
  console.log("Hello World!")
  el.removeEventListener("click", onClick)
}
```

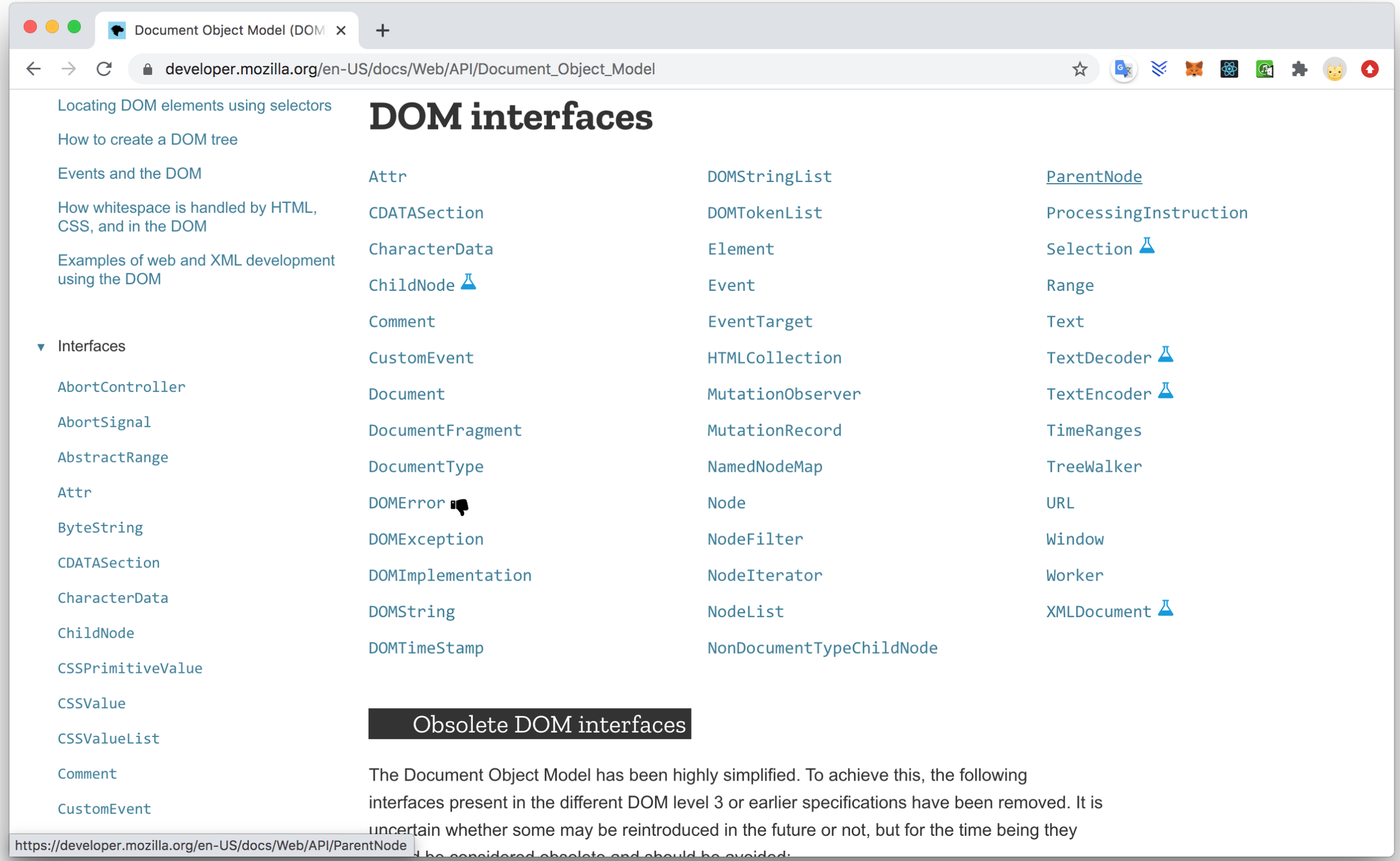
Объект события

Чтобы обработать событие, могут понадобиться детали того, что произошло. Например, координаты курсора мыши.

Когда происходит событие, браузер создаёт объект события, записывает в него доп. информацию и передаёт его в качестве аргумента функции-обработчику.

```
const body = document.body
body.addEventListener('mousemove', onMouseMove)

function onMouseMove(event) {
  console.log('x', event.pageX, 'y', event.pageY)
}
```

MDN Dom


Document Object Model (DOM
developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model
Locating DOM elements using selectors
How to create a DOM tree
Events and the DOM
How whitespace is handled by HTML, CSS, and in the DOM
Examples of web and XML development using the DOM
Interfaces
AbortController
AbortSignal
AbstractRange
Attr
ByteString
CDATASection
CharacterData
ChildNode
CSSPrimitiveValue
CSSValue
CSSValueList
Comment
CustomEvent
DOM interfaces
Attr
CDATASection
CharacterData
ChildNode
Comment
CustomEvent
Document
DocumentFragment
DocumentType
DOMError
DOMException
DOMImplementation
DOMString
DOMTimeStamp
DOMStringList
DOMTokenList
Element
Event
EventTarget
HTMLCollection
MutationObserver
MutationRecord
NamedNodeMap
Node
NodeFilter
NodeIterator
NodeList
NonDocumentTypeChildNode
ParentNode
ProcessingInstruction
Selection
Range
Text
TextDecoder
TextEncoder
TimeRanges
TreeWalker
URL
Window
Worker
XMLDocument
Obsolete DOM interfaces
The Document Object Model has been highly simplified. To achieve this, the following interfaces present in the different DOM level 3 or earlier specifications have been removed. It is uncertain whether some may be reintroduced in the future or not, but for the time being they
https://developer.mozilla.org/en-US/docs/Web/API/ParentNode

Canvas

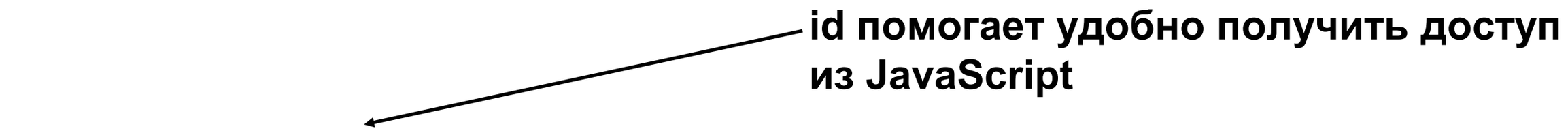
```
<canvas id="cnvs" width="100%" height="100%"></canvas>
```

Канвас — это специальное поле для рисования в браузере на JavaScript. Холст изначально пустой и прозрачный.

Canvas

```
<canvas id="cnvs" width="100%" height="100%"></canvas>
```

Canvas




```
<canvas id="cnvs" width="100%" height="100%"></canvas>
```

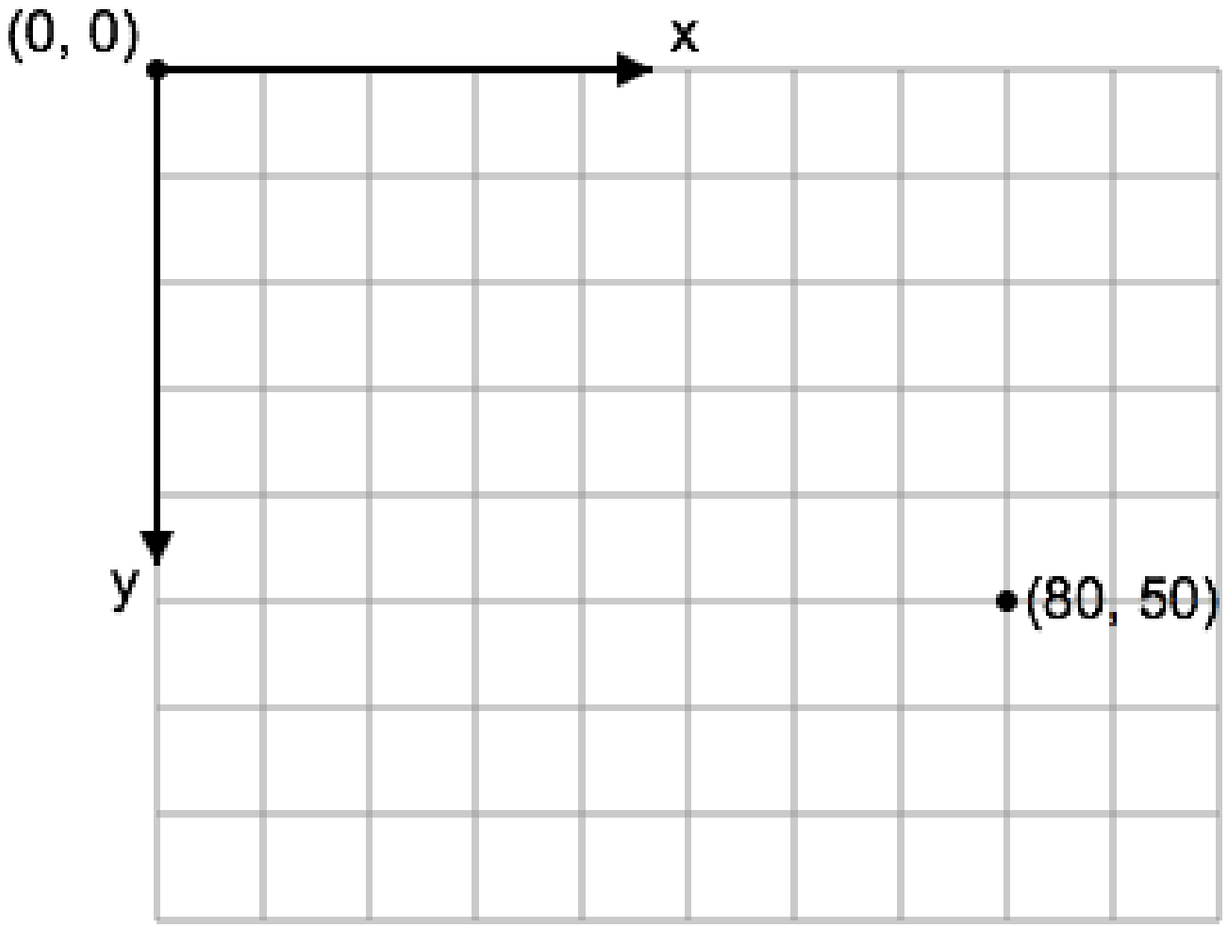
Канвас — это специальное поле для рисования в браузере на JavaScript. Холст изначально пустой и прозрачный.

Canvas

```
const canvas = document.getElementById('cnvs')
const ctx = canvas.getContext('2d')
```

getContext('2d') возвращает контекст через который осуществляется рисование 2D графики.

Координаты

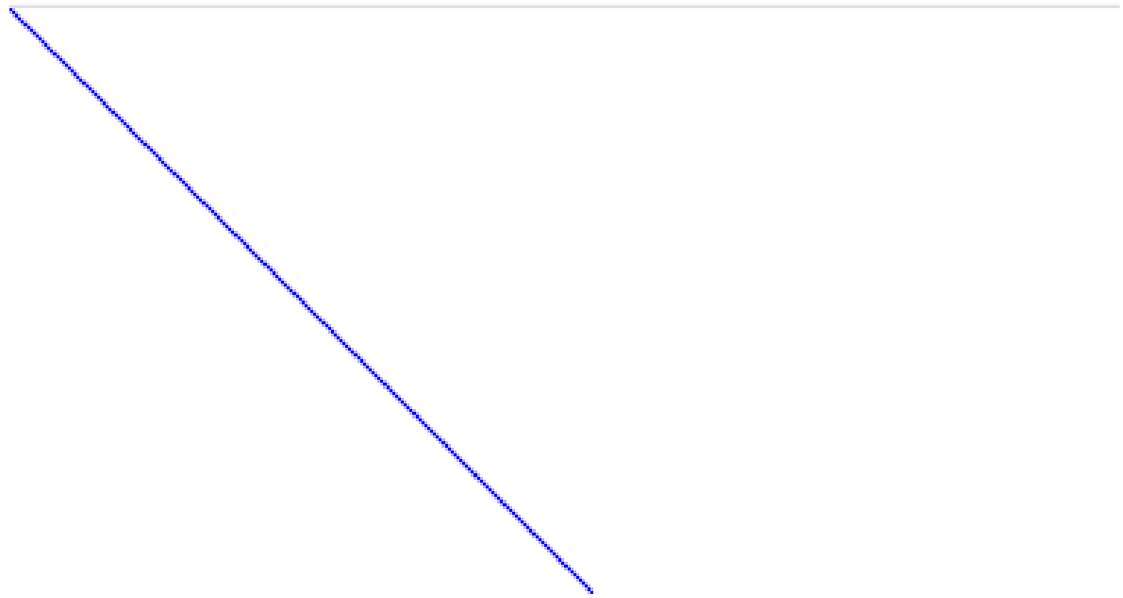


Path

```
beginPath()
closePath()
moveTo()
lineTo()
bezierCurveTo()
quadraticCurveTo()
arc()
arcTo()
ellipse()
rect()
```

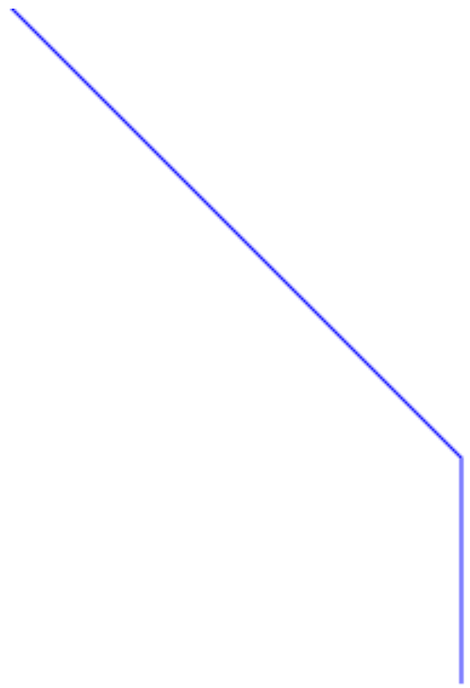
Path 101

```
context.beginPath()
context.strokeStyle = 'blue'
context.moveTo(0, 0)
context.lineTo(200, 200)
context.stroke()
```



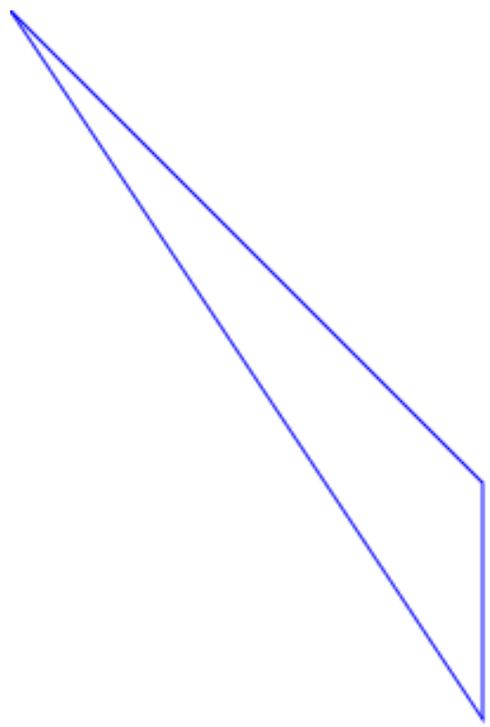
Path 101

```
context.beginPath()
context.strokeStyle = 'blue'
context.moveTo(0, 0)
context.lineTo(200, 200)
context.lineTo(200, 300)
context.stroke()
context.closePath()
```



Path 101

```
context.beginPath()
context.strokeStyle = 'blue'
context.moveTo(0, 0)
context.lineTo(200, 200)
context.lineTo(200, 300)
context.closePath()
context.stroke()
```

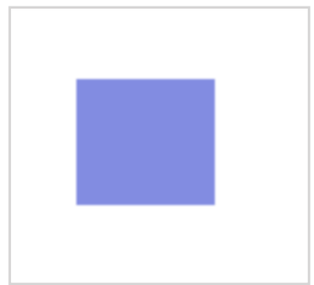


Прямоугольник

```
const canvas = document.getElementById('cnvs');
const context = canvas.getContext('2d');

context.fillStyle = "rgba(0, 0, 200, 0.5)";
context.fillRect (30, 30, 55, 50);
```

Прямоугольник

```
const canvas = document.getElementById('cnvs');
const context = canvas.getContext('2d');

context.fillStyle = "rgba(0, 0, 200, 0.5)";
context.fillRect (30, 30, 55, 50);
```



Круг

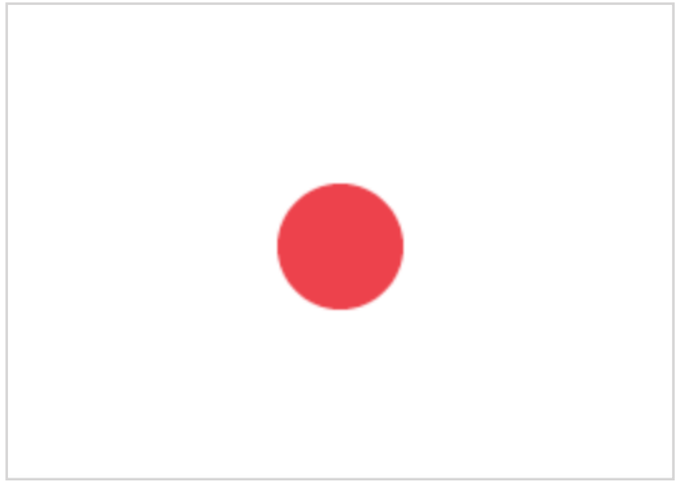
```
const canvas = document.getElementById('cnvs');
const context = canvas.getContext('2d');

// clear canvas
context.clearRect(0, 0, canvas.width, canvas.height);
```

```
// drow circle
context.beginPath();
context.arc(x, y, radius, 0, 2 * Math.PI);
context.fillStyle = "#F04047";
context.fill();
context.closePath();
```

Круг

```
const canvas = document.getElementById('cnvs');
const context = canvas.getContext('2d');

// clear canvas
context.clearRect(0, 0, canvas.width, canvas.height);
```

```
// drow circle
context.beginPath();
context.arc(x, y, radius, 0, 2 * Math.PI);
context.fillStyle = "#F04047";
context.fill();
context.closePath();
```



2D context API

2D context API включает в себя множество функций:

- Геометрические примитивы - линия, прямоугольник, окружность
- Кривые Безье
- Контуры
- Вставка изображений (drawImage)
- Доступ к пикселям
- Текст
- Тени и Градиенты

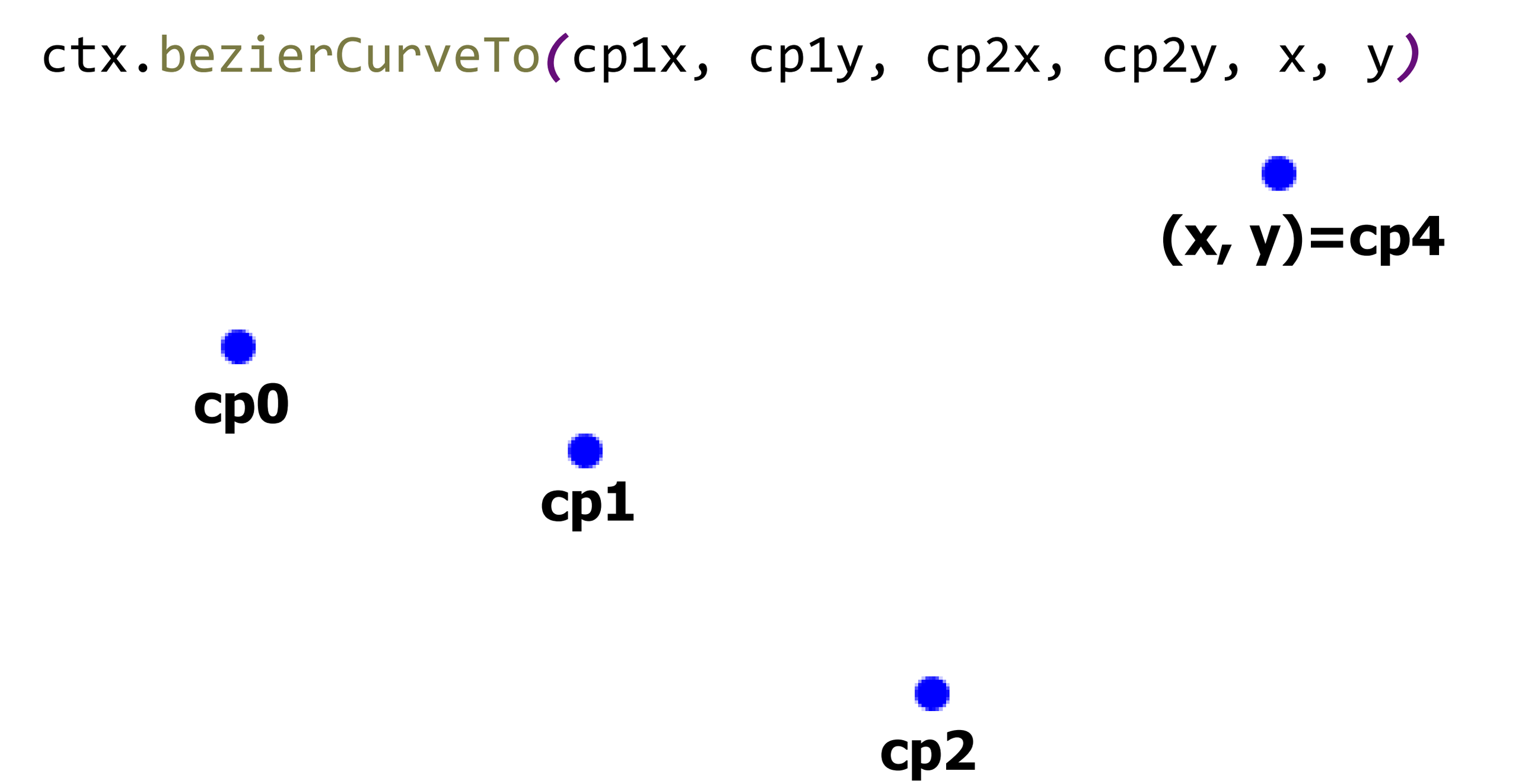
Рисуем кривые Безье

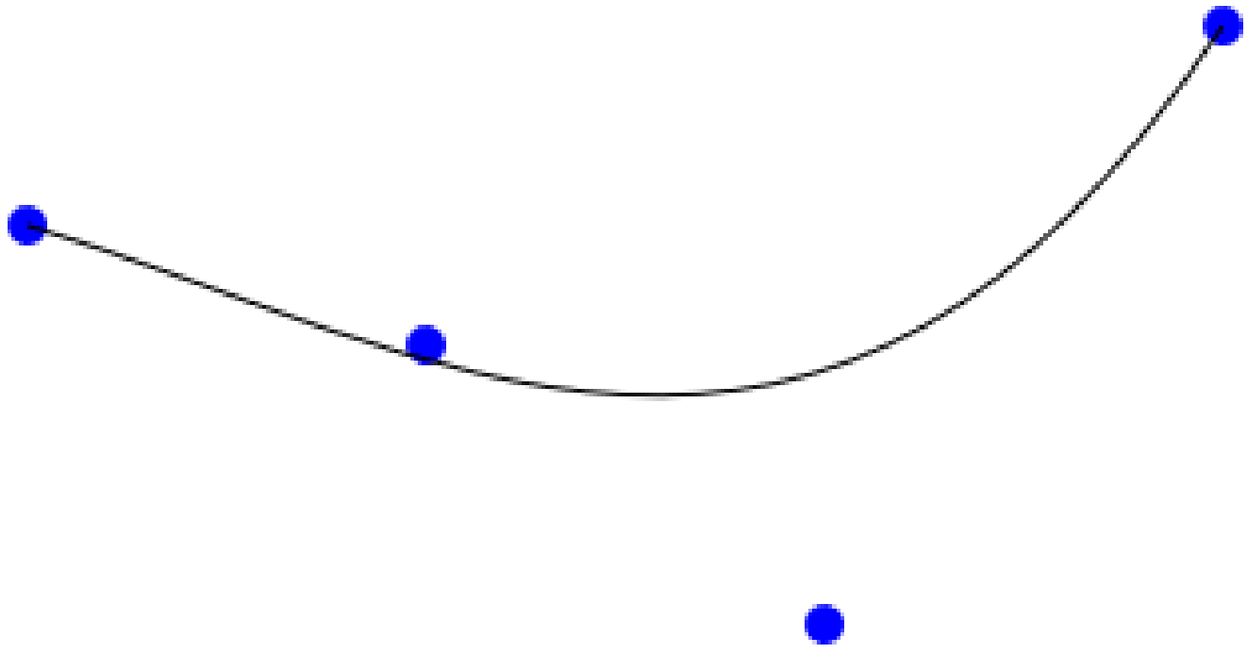
```
ctx.bezierCurveTo(cp1x, cp1y, cp2x, cp2y, x, y)
```

```
ctx.quadraticCurveTo(cpx, cpy, x, y);
```

```
const points = [[100, 100], [200,130],
[300, 200], [400, 50]];
context.fillStyle = "#0000FF";
points.forEach(function (item) {
   context.beginPath();
   context.arc(item[0], item[1], 5, 0, 2*Math.PI);
   context.fill();
});
```

```
ctx.bezierCurveTo(cp1x, cp1y, cp2x, cp2y, x, y)
```





```
context.beginPath();
const [cp0x, cp0y] = points[0];
const [cp1x, cp1y] = points[1];
const [cp2x, cp2y] = points[2];
const [cp3x, cp3y] = points[3];
context.moveTo(cp0x, cp0y);
context.bezierCurveTo(cp1x, cp1y, cp2x, cp2y, cp3x, cp3y);
context.stroke();
```

Понимаем кривые Безье

$$B(t) = \sum_{k=0}^{n} P_k b_{k,n}(t),\ 0 \le t \le 1$$

$$B(t) = (1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3t^2(1-t)P_2 + t^3 P_3$$

Из квадратичной в кубическую кривую

$$(x_0; y_0),\ (x_1; y_1),\ (x_2; y_2)$$

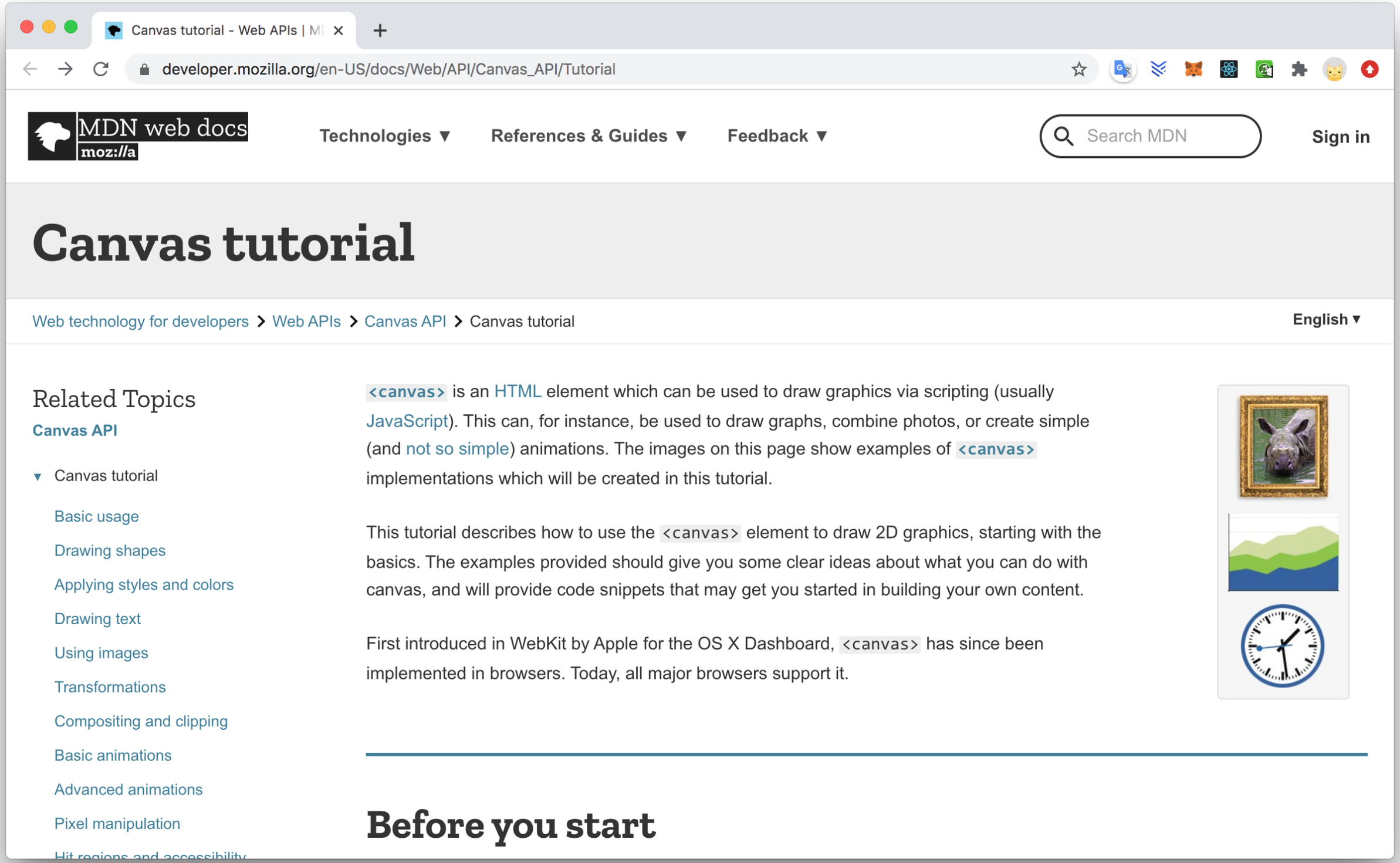
$$(x_0; y_0),\ \left(x_0 + \frac{2 \cdot (x_1 - x_0)}{3}; y_0 + \frac{2 \cdot (y_1 - y_0)}{3}\right),$$

$$\left(x_1 + \frac{x_2 - x_1}{3}; y_1 + \frac{y_2 - y_1}{3}\right),\ (x_2; y_2)$$

Свойства кривых Безье

- КБ Непрерывна
- КБ лежит внутри оболочки, натянутой на контрольные точки
- КБ инвариантна относительно изменения направления траектории
- КБ афинно-инвариантна
- КБ не может описать окружность

Canvas tutorial



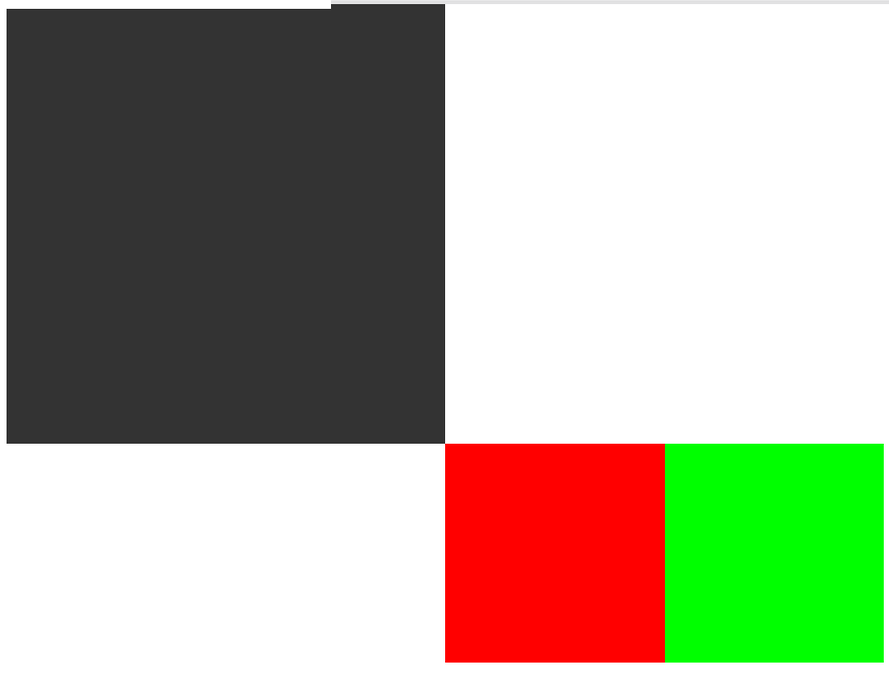
Преобразования

- Перенос
- Масштабирование
- Поворот

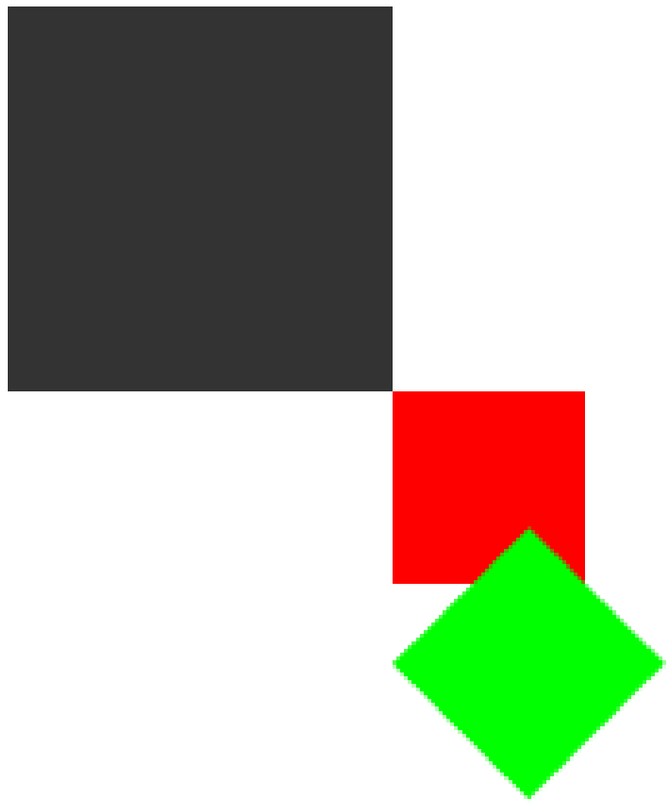
```
translate(x,y)
scale(x,y)
rotate(phi)
```

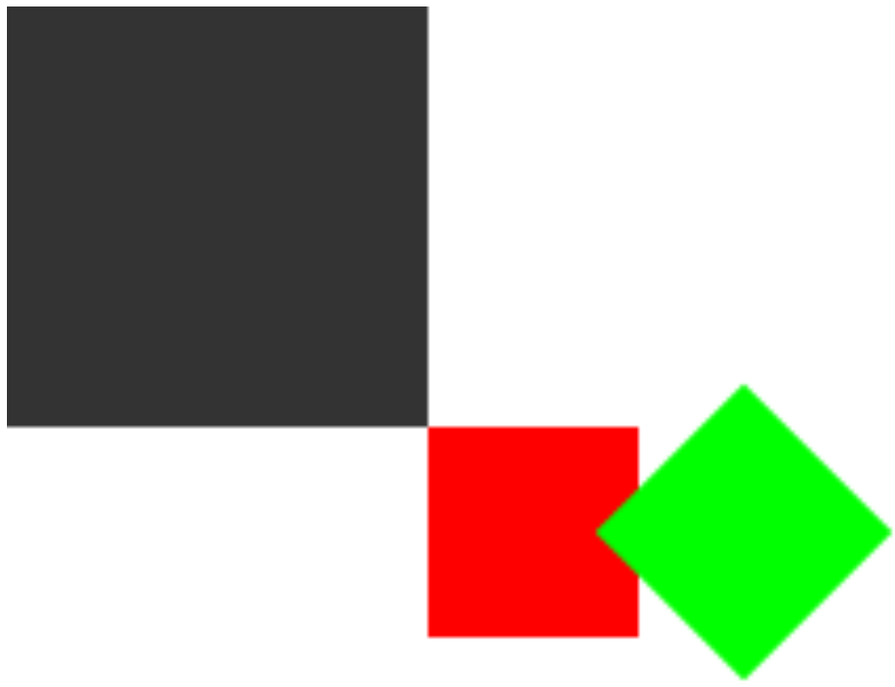
Преобразования производятся в ГСК

```
context.fillStyle = "#333333";
context.fillRect(0, 0, 100, 100);
context.translate(100, 100)
context.fillStyle = "#FF0000";
context.fillRect(0, 0, 50, 50);
context.fillStyle = "#00FF00";
context.fillRect(50, 0, 50, 50);
```



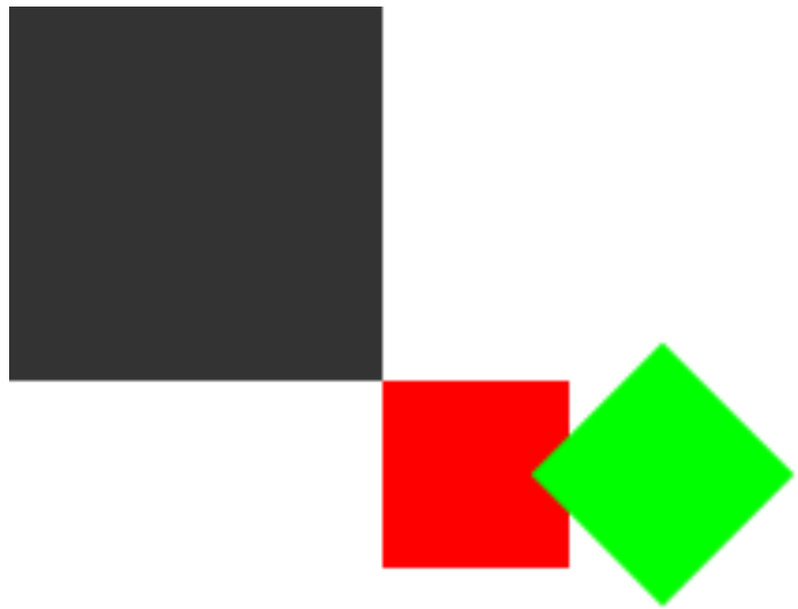
```
context.fillStyle = "#333333";
context.fillRect(0, 0, 100, 100);
context.translate(100, 100)
context.fillStyle = "#FF0000";
context.fillRect(0, 0, 50, 50);
context.fillStyle = "#00FF00";
context.rotate(45*Math.PI/180); //!!!!!
context.fillRect(50, 0, 50, 50);
```





Модифицируем отдельный элемент

```
context.fillStyle = "#333333";
context.fillRect(0, 0, 100, 100);
context.translate(100, 100)
context.fillStyle = "#FF0000";
context.fillRect(0, 0, 50, 50);
context.save(); //!!!!!
context.translate(75,25);
context.rotate(45*Math.PI/180);
context.fillStyle = "#00FF00";
context.fillRect(-25, -25, 50, 50);
context.restore(); //!!!!!
```



Афинные преобразования

Преобразование плоскости называется аффинным, если

1. оно взаимно однозначно;
2. образом любой прямой является прямая.

Преобразование называется взаимно однозначным, если

1. разные точки переходят в разные;
2. в каждую точку переходит какая-то точка.

Совокупность преобразований

Растяжение-сжатие

$$\begin{bmatrix} a_x & 0 & 0 \\ 0 & a_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Поворот

$$\begin{bmatrix} \cos(\varphi) & \sin(\varphi) & 0 \\ -\sin(\varphi) & \cos(\varphi) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Параллельный перенос

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{bmatrix}$$

Однородные координаты

Однородные координаты — координаты, обладающие тем свойством, что определяемый ими объект не меняется при умножении всех координат на одно и то же число.

Однородными координатами вектора (x, y) является тройка чисел (x', y', h), где x = x' / h, y = y'/h, а h — некоторое вещественное число, отличное от 0

(1, 1, 1) и (2,2,2) – одна и та же точка

Произвольное преобразование

$$\begin{bmatrix} x & y & 1 \end{bmatrix} \bullet \begin{bmatrix} R_{1,1} & R_{1,2} & 0 \\ R_{2,1} & R_{2,2} & 0 \\ t_x & t_y & 1 \end{bmatrix}$$

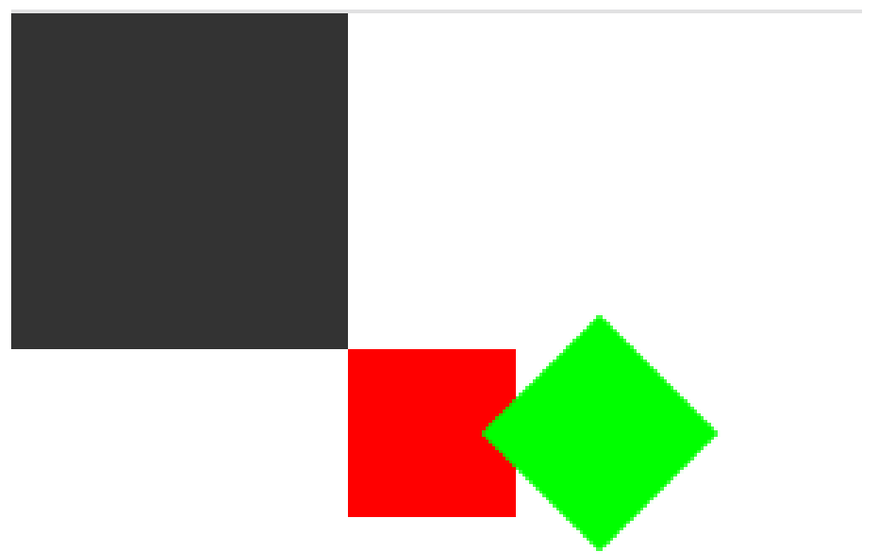
Преобразования в Canvas

```
context.transform(a, b, c, d, e, f);
```

$$\begin{bmatrix} a & c & e \\ b & d & f \\ 0 & 0 & 1 \end{bmatrix}$$

```
context.transform(1, 0, 0, 1, 0, 0);
```

```
context.fillStyle = "#333333";
context.fillRect(0, 0, 100, 100);
context.translate(100, 100);
context.fillStyle = "#FF0000";
context.fillRect(0, 0, 50, 50);
context.save();
const d = 1/Math.sqrt(2);
context.transform(d, d, -d, d,75, 25);
context.fillStyle = "#00FF00";
context.fillRect(-25, -25, 50, 50);
context.restore();
```



Новое преобразование

```
context.setTransform(a, b, c, d, e, f);
```

```
context.resetTransform();
```

Изображения

```
<body>
    <canvas width="100%"
            height="100%"
            id='cnvs’>
     </canvas>
    <img id="pacman" src="img/pacman.png" />
    <script src="./src/index.js"></script>
</body>
```

```
const pacman = document.getElementById("pacman");

context.drawImage(pacman, 10, 10);
```

Взаимодействия с пользователем

```
while (true)  {
      const command = readCommand();
      handleCommand(command);
}
```

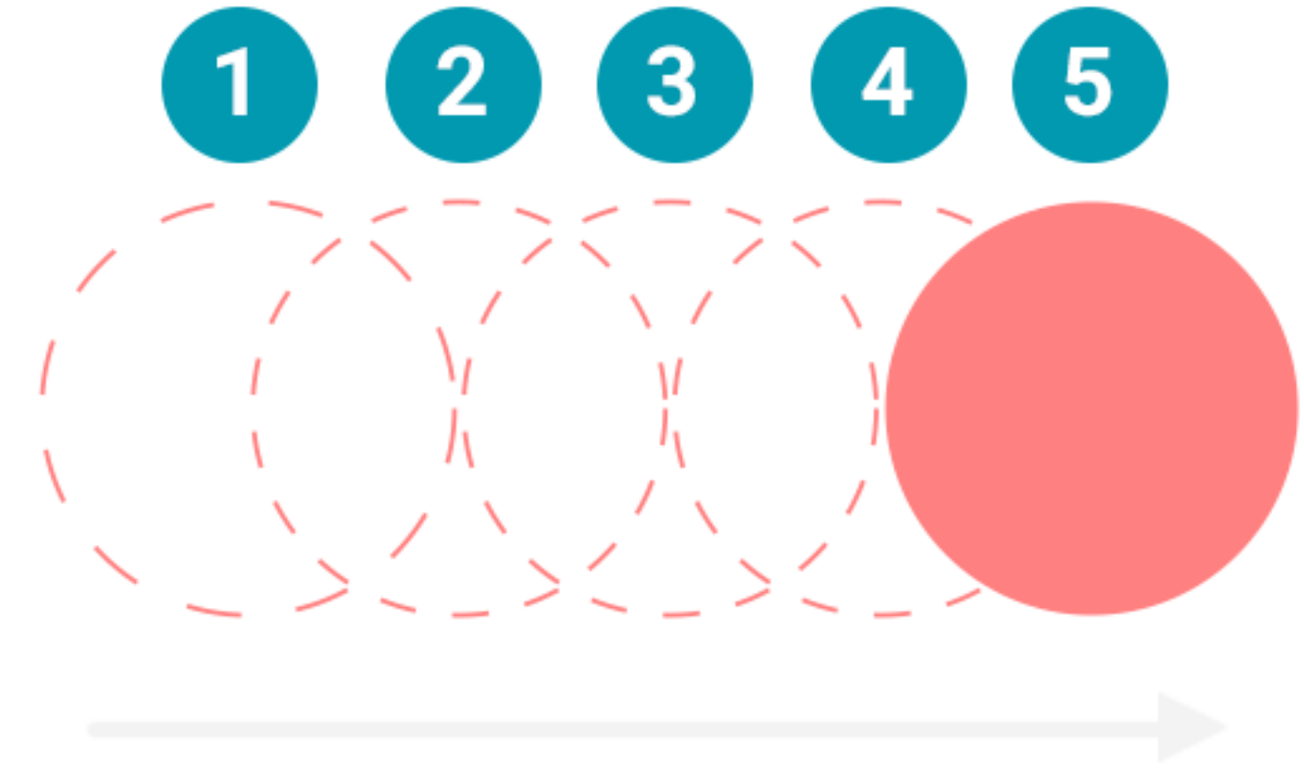
```
What?

> go downstream

You are in a valley in the forest beside a stream tumbling
along a rocky bed.

> go south

At your feet all the water of the stream splashes into a
2-inch slit in the rock.  Downstream the streambed is bare rock.

> go slit

You don't fit through a two-inch slit!

> go rock
```

Game Loop

Покадровая **Анимация**

(Flip Book)

Движение

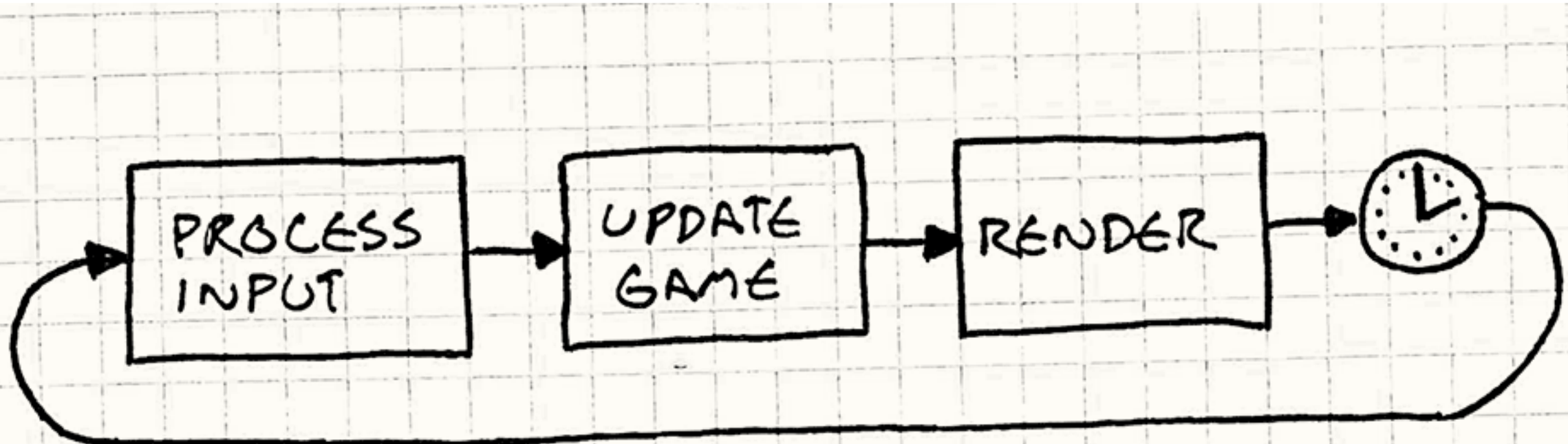


– Очищаем canvas
– Рисуем шарик на новом месте

User Input

– Считываем ввод
– Пересчитываем состояние игры

Game Loop



1 tick = 1000ms / 60fps = 16.6ms

Паттерн проектирования Game Loop

Игровой цикл работает на протяжении всей игры.

На каждом своем цикле игровой цикл

обрабатывает пользовательский ввод без блокировки,

обновляет состояние игры и **рендерит игру**,

управляет скоростью игрового процесса.

https://martalex.gitbooks.io/gameprogrammingpatterns/content/chapter-3/3.2-game-loop.html

Первая мысль

```
function gameLoop() {
  update()
  draw()
}

setInterval(gameLoop, 1000 / 60)
```

Первая мысль

```
function gameLoop() {
  update()
  draw()
}

setInterval(gameLoop, 1000 / 60)
```

плохо!

requestAnimationFrame

```
function fn(msFromStart) {
  console.log(msFromStart)
}

// просим запустить функцию fn в "ближайшее время"
let requestId = requestAnimationFrame(fn)

// отмена запланированного запуска fn
cancelAnimationFrame(requestId)
```

Второй подход

```
function gameLoop(msFromStart) {
  update(msFromStart)
  draw()
  requestAnimationFrame(gameLoop)
}

requestAnimationFrame(gameLoop)
```

Второй подход

```
function gameLoop(msFromStart) {
  update(msFromStart)
  draw()
  requestAnimationFrame(gameLoop)
}

requestAnimationFrame(gameLoop)
```

что будет если код не выполнится за 16ms?

Как заставить функцию выполняться в цикле?

```
setInterval(function, delay)

setTimeout(function, delay)

requestAnimationFrame(callback)
```

Нормальная версия

```
function run() {
  window.requestAnimationFrame( run );
  update();
  draw();
}

function draw() {…}
function update() {…}
```

Версия получше-1

```
(function() {
  function run() {
    gameState.stopCycle = window.requestAnimationFrame( run );
    update();
    draw();
  }
  run();
})();

function draw() {}
function update() {}
```

Версия получше-2

```
const canvas = document.getElementById("cnvs");
const gameState = {…};

canvas.addEventListener("click", onMouseClick, false);

function onMouseClick(e) {
  window.cancelAnimationFrame( gameState.stopCycle );
}
```

Как измерять время/FPS?

```
const tNow = window.performance.now();
```

Эффективный и управляемый Game Loop-1

```
function run(tFrame) {
  gameState.stopCycle = window.requestAnimationFrame( run );

  const nextTick = gameState.lastTick + gameState.tickLength;
  let numTicks = 0;

  if (tFrame > nextTick) {
    const timeSinceTick = tFrame - gameState.lastTick;
    numTicks = Math.floor(timeSinceTick / gameState.tickLength);
  }
  queueUpdates( numTicks );
  draw( tFrame );
  gameState.lastRender = tFrame;
}
```

Эффективный и управляемый Game Loop-1

```
function queueUpdates( numTicks ) {
  for(let i=0; i < numTicks; i++) {
    gameState.lastTick =
              gameState.lastTick + gameState.tickLength;
    update( gameState.lastTick );
  }
}
```

Состояние объектов

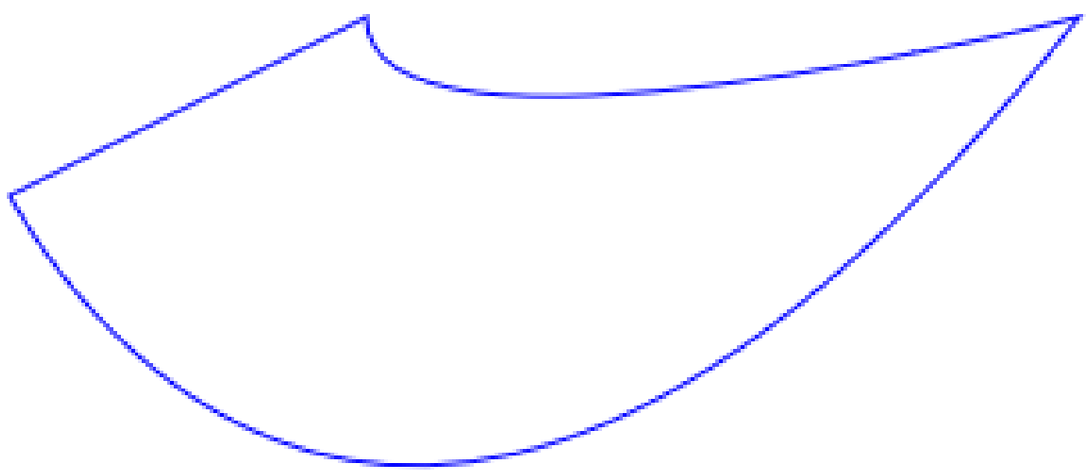
```
gameState.ball = {
   x: canvas.width / 2,
   y: 0,
   radius: 25,
   vx: 0,
   vy: 5
}

gameState.ball.y += gameState.ball.vy;
gameState.ball.x += gameState.ball.vx;

context.beginPath();
context.arc(ball.x, ball.y, ball.radius, 0, 2*Math.PI);
context.fillStyle = "#0000FF";
context.fill();
```

Collision detection

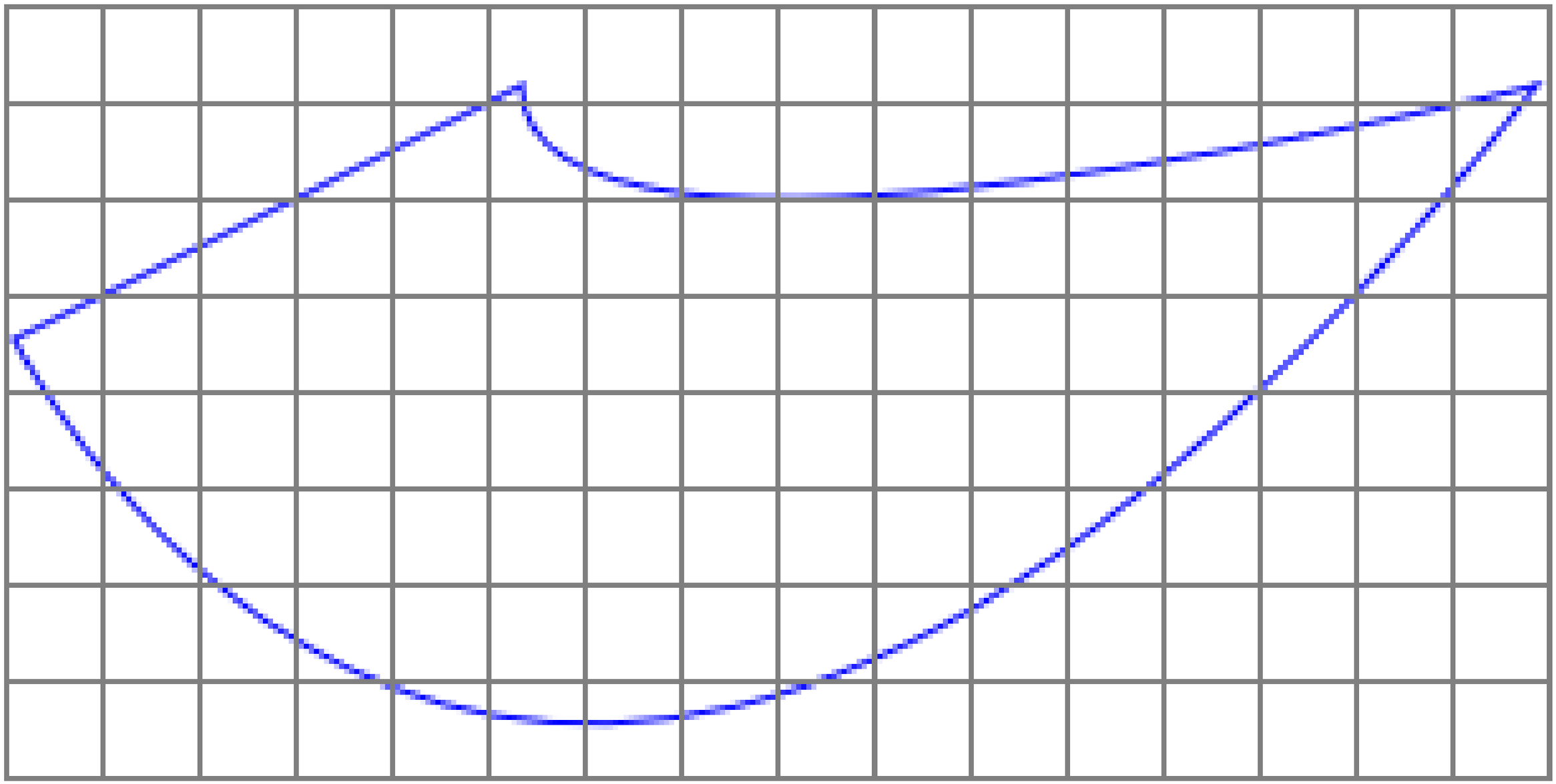
- Объект и граница экрана
- Примитив и примитив
- Сложный объект и сложный объект
- Булевские проверки или оценка столкновения?

Произвольный объект

```
context.beginPath();
context.strokeStyle = "#0000FF";
context.bezierCurveTo(100, 100, 200, 300, 400, 50);
context.bezierCurveTo(400, 50, 200, 100, 200, 50);
context.closePath();
context.stroke();
```



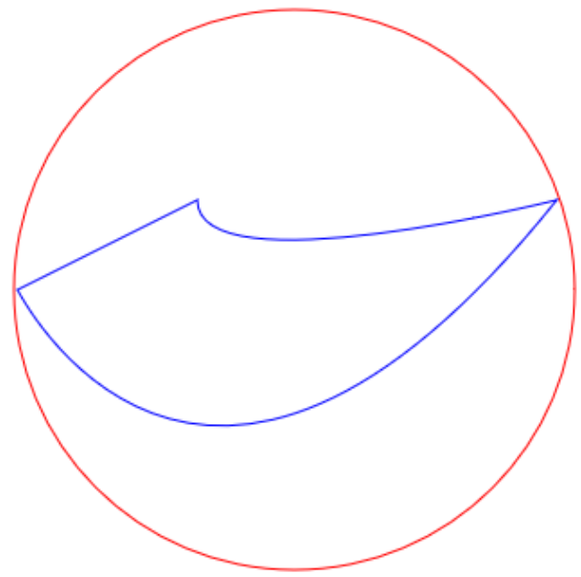
AABB



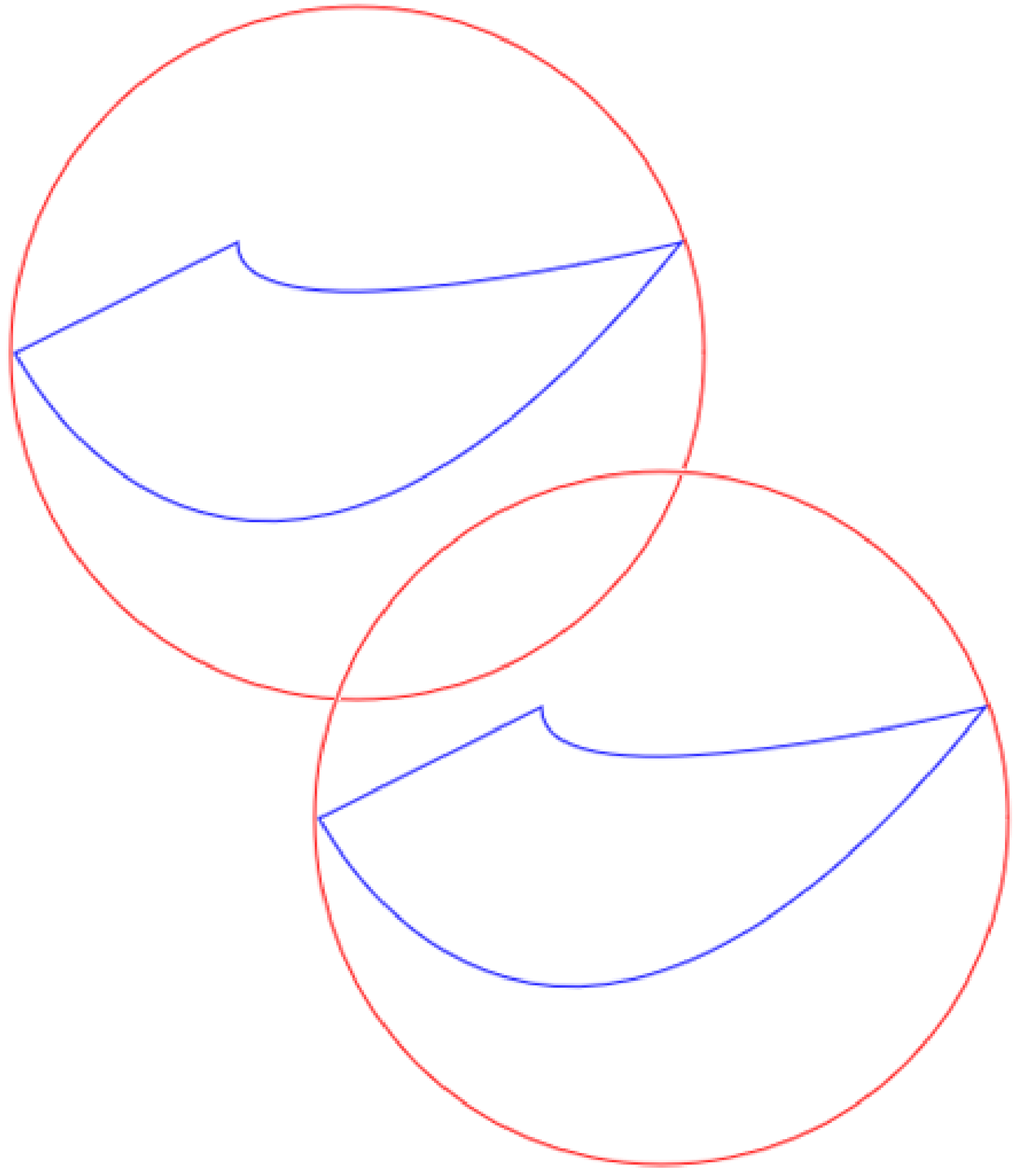
Круги

```
var circle1 = {radius: 20, x: 5, y: 5};
var circle2 = {radius: 12, x: 10, y: 5};

var dx = circle1.x - circle2.x;
var dy = circle1.y - circle2.y;
var distance = Math.sqrt(dx * dx + dy * dy);
```

```
if (distance < circle1.radius + circle2.radius) {
  // collision detected!
}
```



Кривые хит-боксы!



Улучшаем положение

- 1. Найдем пары кандидатов на столкновение (грубая фаза)
- 2. Проверим пары (точная фаза)

Теорема разделяющих осей (SAT)
Теорема об опорной гиперплоскости

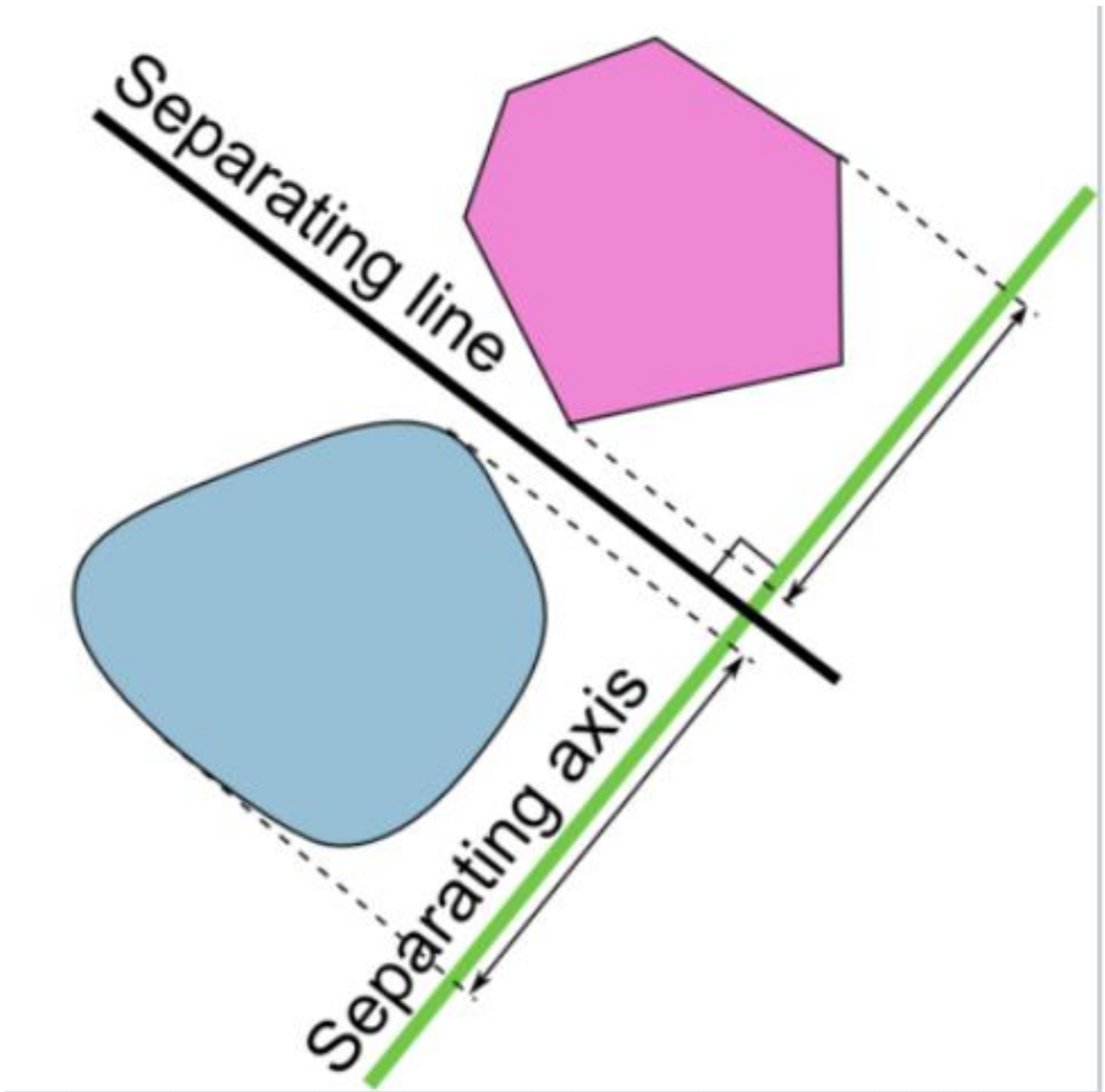
Если заданы замкнутое ограниченное выпуклое множество $C \in \mathbb{R}^m$ и точка $z^* = (z_1^*, z_2^*, \ldots, z_m^*) \in \mathbb{R}^m$, не принадлежащая множеству C, то существуют такие числа $a_1, a_2, \ldots, a_m, b$, что

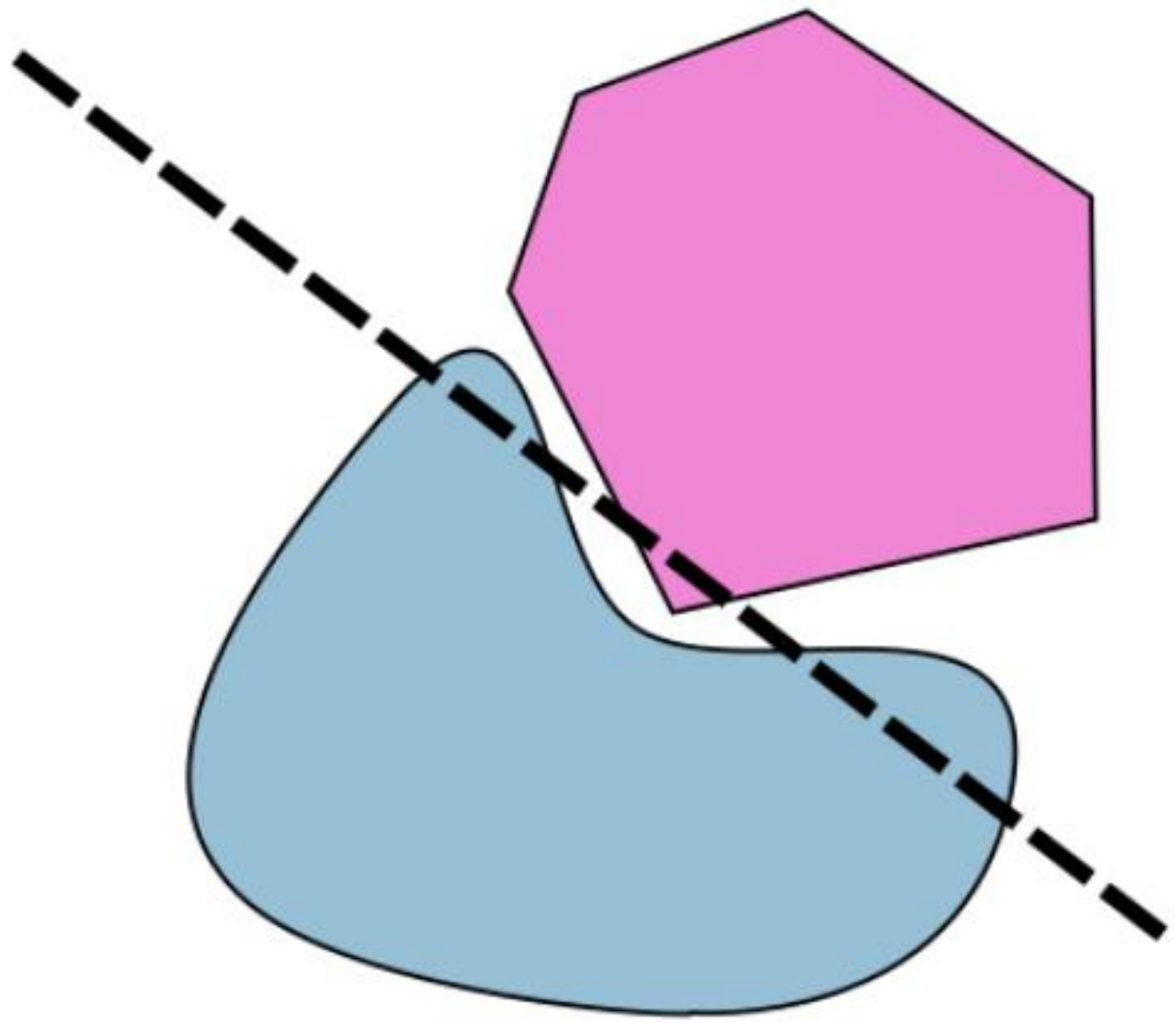
$$a_1 z_1^* + a_2 z_2^* + \ldots + a_m z_m^* = b$$

$$a_1 z_1 + a_2 z_2 + \ldots + a_m z_m > b, \forall z \in C$$

Геометрически это означает, что через точку z^* можно провести гиперплоскость так, что множество C будет лежать «выше» этой гиперплоскости.

Геометрический смысл





Попроще

Если два выпуклых объекта не имеют общих точек, то существует ось, на которой проекции этих объектов не пересекаются

Алгоритм

1. Знать вершины объектов
2. Построить ребра объектов
3. Построить нормали к ребрам - оси
4. Проверить все оси для всех пар объектов, построив проекции

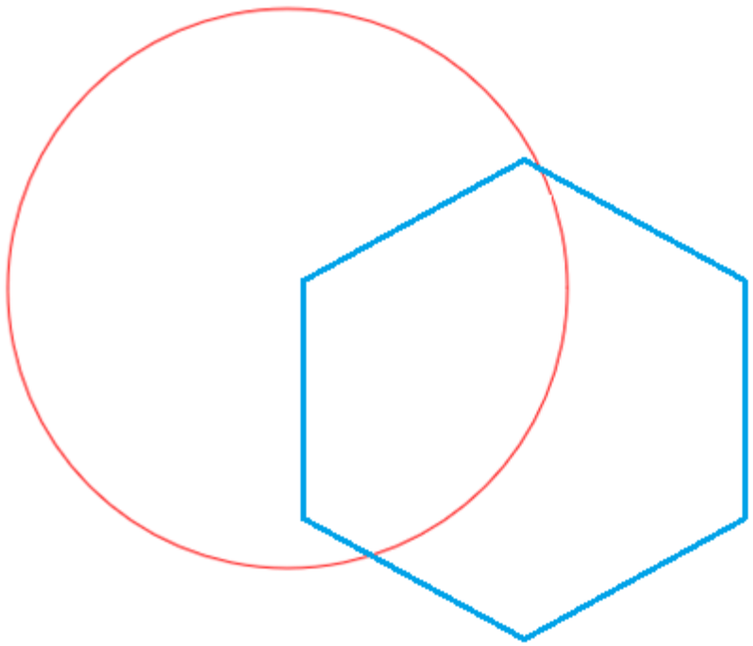
Алгоритм с MTV

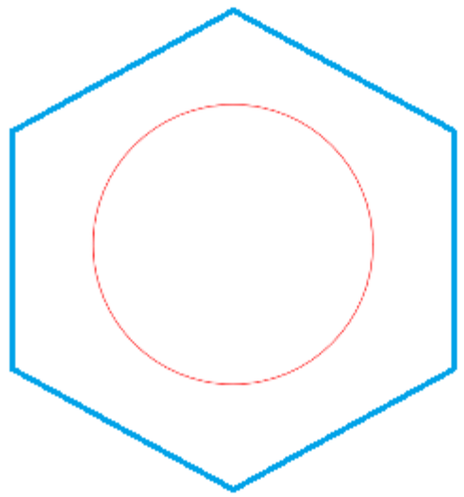
1. Знать вершины объектов
2. Построить ребра объектов
3. Построить нормальные нормали к ребрам - оси
4. Проверить все оси для всех пар объектов, построив проекции
5. Минимальная проекция - MTV

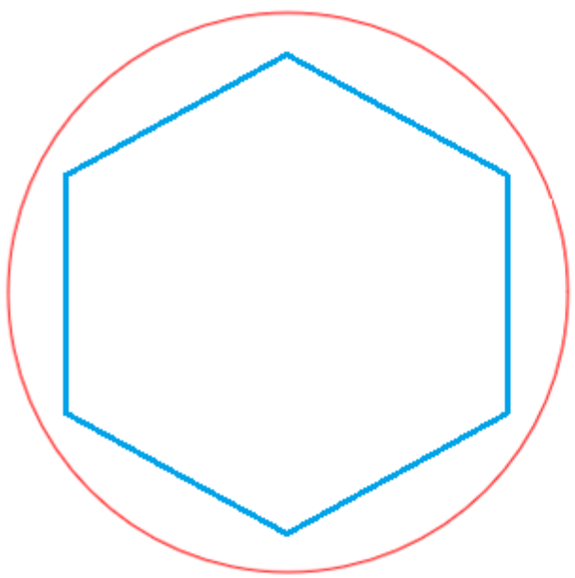
Плюсы, минусы, подводные камни

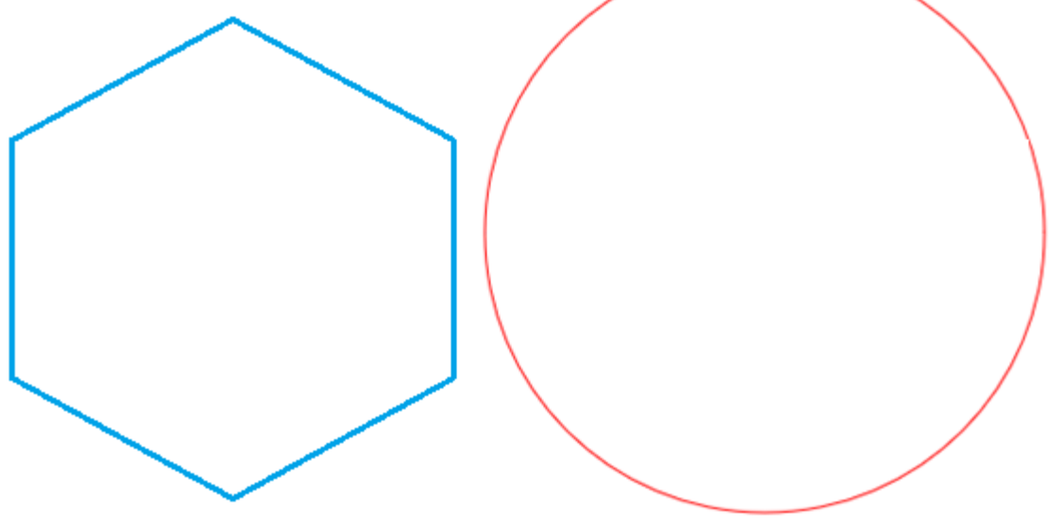
- Как проверять круги?
- Хорошо, когда мало столкновений, а объектов много
- Можно получать MTV

Но как проверять круги??









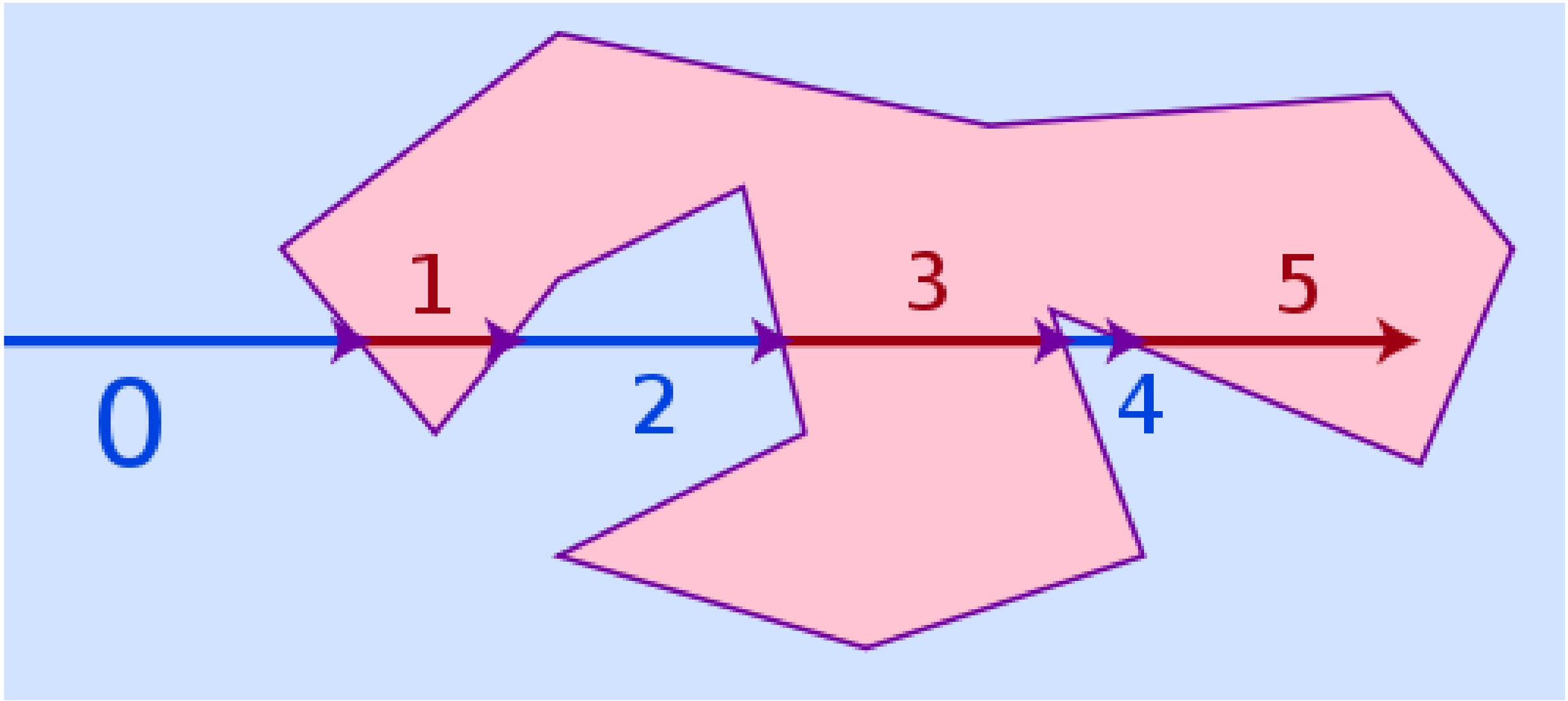
Проблема «круг-многоугольник»

- Проблема «круг-отрезок»
- Проблема «точка-многоугольник»

Проблема «круг-отрезок»

1. Строим проекцию центра круга на прямую
2. Проверяем, принадлежит ли полученная точка отрезку (сумма расстояний от точки до концов отрезка равна длине отрезка)

Ray Casting



Материалы

- **Анатомия видеоигры** https://developer.mozilla.org/en-US/docs/Games/Anatomy
- **Canvas Tutorial** https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Tutorial