

# Компьютерная графика WebGL

Лекция 1

Демяненко Я.М. ЮФУ 2025

# WebGL



WebGL (Web-based Graphics Library) — кроссплатформенный API для 3D-графики в браузере, разработан некоммерческой организацией Khronos Group

Тип	API
Автор	Mozilla Foundation
Разработчик	Khronos Group
Операционная система	Cross-platform
Первый выпуск	3 марта 2011
Последняя версия	2.0 (30 августа 2018)
Лицензия	Various
Сайт	<a href="http://khronos.org/webgl/">khronos.org/webgl/</a>

Современная версия 2.0 (несовместима с версией 1.0) доступна с весны 2017 года (January 17, 2017 :-)

# WebGPU



WebGPU — это один из современных API, предназначенных для работы с компьютерной графикой.

WebGPU — это API JavaScript , C++ , Rust и C , который позволяет переносимо и эффективно использовать графический процессор (GPU) устройства . Это достигается с помощью базовых системных API Vulkan , Metal или Direct3D 12.

В JavaScript WebGPU может быть предоставлен веб-браузером или другой средой JavaScript, такой как node.js и Deno .

Google Chrome включил первоначальную поддержку WebGPU в апреле 2023 года. Safari и Firefox пока не включили свою поддержку.

Таким образом, **стандарт W3C находится на стадии рабочего проекта.**

WebGPU позволяет использовать 3D-графику в HTML- холсте . Он также имеет надежную поддержку для вычислений общего назначения на GPU .

WebGPU использует собственный язык шейдеров , называемый WGSL.

# WebGL

WebGL использует язык программирования шейдеров GLSL

WebGL — это контекст элемента canvas HTML, который обеспечивает API 3D графики без использования плагинов

За счёт использования низкоуровневых средств поддержки OpenGL часть кода на WebGL может выполняться непосредственно на видеокартах

WebGL исполняется как элемент HTML5 и поэтому является полноценной частью объектной модели документа (DOM API) браузера

Может использоваться с любыми языками программирования, которые умеют работать с DOM API, например, JavaScript, Rust, Java, Kotlin и другими.

Все ведущие разработчики браузеров Google (Chrome), Mozilla (Firefox), и Apple (Safari), являются членами Khronos и реализуют WebGL в своих браузерах.

# Desktop браузеры

**Mozilla Firefox** — WebGL был включён на всех платформах, у которых есть нужная графическая карта с актуальными драйверами, начиная с версии 4.0.

**Google Chrome** — WebGL включён по умолчанию во всех версиях, начиная с 9.

**Safari** — экспериментально поддерживает WebGL, начиная с версии 5.1, полная поддержка реализована и включена по умолчанию в версии 8.0.

**Opera** — WebGL реализован в версии Opera 12.0, но отключён по умолчанию.

**Internet Explorer** — начиная с Internet Explorer 11 WebGL официально поддерживается. До выхода 11 версии независимыми разработчиками были выпущены плагины Chrome Frame и IEWebGL, предусматривающие опции, необходимые для поддержки WebGL в Internet Explorer.

**Microsoft Edge**

# Мобильные браузеры

[Maemo 5](#) — WebGL доступен во встроенном браузере microB начиная с версии прошивки PR1.2 (май 2011)

[BlackBerry PlayBook](#) — WebGL доступен посредством WebWorks и браузера в PlayBook OS 2.0

[Google Chrome](#) — WebGL доступен для Android-устройств в бета-версиях начиная с января 2013

[Firefox for mobile](#) — WebGL доступен для Android устройств в бета-версиях с начала 2011

[Sony Ericsson Xperia](#) — многие устройства на базе Android имеют поддержку WebGL после обновления прошивки

[Opera Mobile 12 final](#) поддерживает WebGL (только на Android)

[Safari для iOS](#) — поддержка реализована в версии 8.0

# Поддержка WebGL

Чтобы получить обновленный список веб-браузеров, которые поддерживают WebGL, перейдите по следующей [ссылке](http://www.khronos.org/webgl/wiki/Getting_a_WebGL_Implementation) ([http://www.khronos.org/webgl/wiki/Getting\\_a\\_WebGL\\_Implementation](http://www.khronos.org/webgl/wiki/Getting_a_WebGL_Implementation)).

Если вы хотите быстро проверить, поддерживает ли ваша конфигурация WebGL, то перейдите по этой [ссылке](http://get.webgl.org/) (<http://get.webgl.org/>).

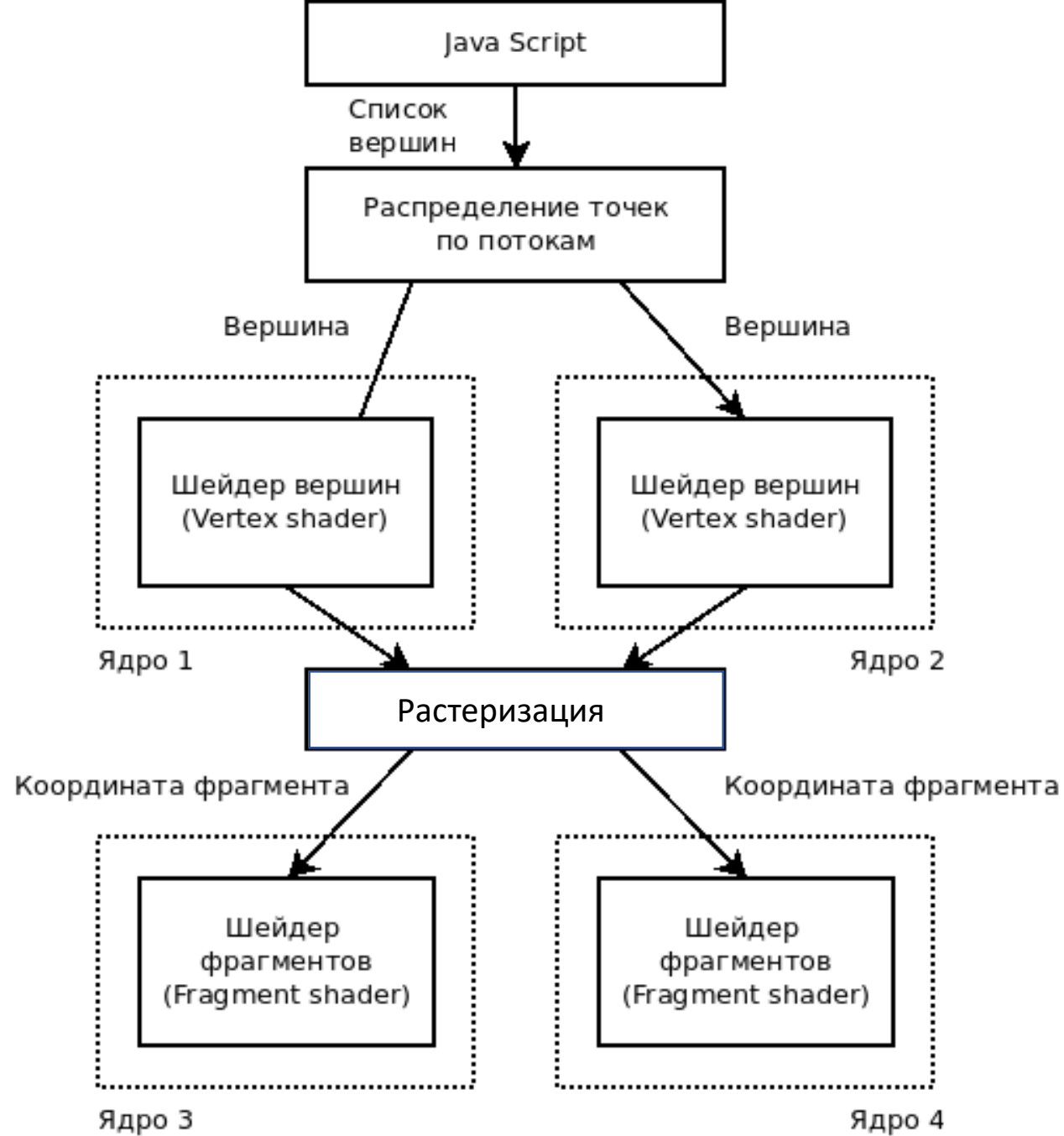
# Основные графические операции WebGL

- конструирует фигуры из геометрических примитивов
- позиционирует объекты в трехмерном пространстве и выбирает точку наблюдения
- вычисляет цвета для всех объектов
  - могут быть определены приложением
  - получены из расчета условий освещенности
  - вычислены при помощи текстур, наложенных на объекты
  - или из любой комбинации этих факторов
- выполняет растеризацию (растровую развертку)

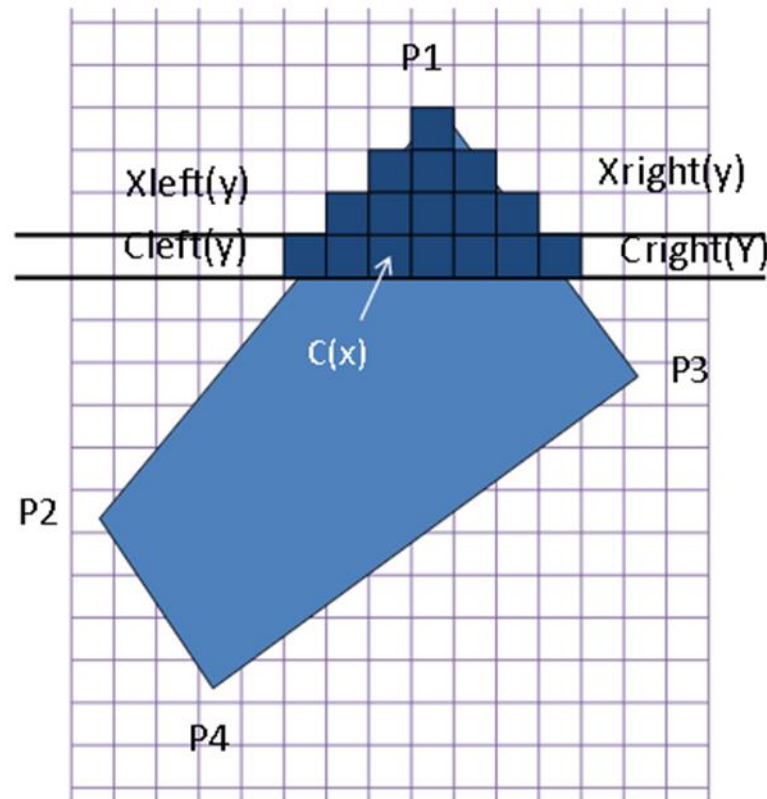


# Чего нет в WebGL?

- функций для создания окон
- захвата пользовательского ввода
- высокоуровневых функций для описания моделей трёхмерных объектов



# Растреризация: процесс вычисления пикселей растра, принадлежащих примитиву



## Вход:

- Координаты P1, P2, P3, P4 (экранные)
- Цвета вершин  $C_i$  или другие атрибуты

# Типы шейдеров

- вершинный (vertex)
- фрагментный (fragment)
- геометрический (geometry)
- 2 тесселяционных шейдера (tesselation), отвечающие за 2 разных этапа тесселяции

# Передача данных в/из/между шейдерами

- Атрибуты (данные, характеризующие вершины, берутся из буфера) – in/out (varying)
- Uniform-переменные (значения, постоянные для всех вершин на протяжении одного вызова отрисовки)
- in/out – переменные (для передачи между шейдерами)
- Текстуры (данные из пикселей/текселей)

# Схема работы

1. Получаем WebGL контекст из canvas'a
2. Загружаем программу шейдеров
  - создаем программу шейдеров
  - получаем исходный код отдельно для вершинного и фрагментного шейдеров
  - компилируем коды шейдеров
  - присоединяем к программе
  - активируем программу
3. Устанавливаем две матрицы: model-view и projection
4. Размещаем, заполняем, активируем буферы данных вершин
5. Рисуем

# WebGL и OpenGL ES 2.0

WebGL имеет методы, названия которых очень похожи на функции OpenGL ES.

Например, функция

для OpenGL ES 2.0

```
glclear(COLOR_BUFFER_BIT)
```

для WebGL

```
gl.clear(gl.COLOR_BUFFER_BIT)
```

Но не все функции WebGL имеют такой же синтаксис, как и функции OpenGL ES 2.0.

# Подготовка контекста WebGL

```
var gl; // глобальная переменная для контекста WebGL
```

```
function start() {
```

```
  var canvas = document.getElementById("glcanvas");
```

```
  gl = initWebGL(canvas); // инициализация контекста GL – сами пишем
```

```
  // продолжать только если WebGL доступен и работает
```

```
  ...
```



# function initWebGL

```
function initWebGL(canvas) {  
    gl = null;  
  
    try {    // Попытаться получить стандартный контекст.  
            // Если не получится, попробовать получить экспериментальный.  
        gl = canvas.getContext("webgl2") || canvas.getContext("webgl") || canvas.getContext("experimental-  
webgl");  
    }  
    catch(e) {}  
  
    // Если мы не получили контекст GL, завершить работу  
    if (!gl) {  
        alert("Unable to initialize WebGL. Your browser may not support it.");  
        gl = null;  
    }  
  
    return gl;  
}
```

# Расширенный вариант

```
var names = ["webgl2", "webgl", "experimental-webgl", "webkit-3d", "moz-webgl"];
gl = null;
for (var ii = 0; ii < names.length; ++ii) {
    try {
        gl = canvas.getContext(names[ii]);
    }
    catch(e) {}
    if (gl) {
        break;
    }
}
```

# А можно так начать

```
main();  
//  
// Start here  
//  
function main() {  
    const canvas = document.querySelector('#glcanvas');  
    const gl = canvas.getContext('webgl2');  
    // If we don't have a GL context, give up now  
    if (!gl) {  
        alert('Unable to initialize WebGL. Your browser or machine may not support it.');
```

```
        return;  
    }  
    ...  
}
```

# Если WebGL доступен

```
if (gl) { // продолжать только если WebGL доступен и работает

    // Устанавливаем размер вьюпорта
    gl.viewport(0, 0, gl.canvas.width, gl.canvas.height);

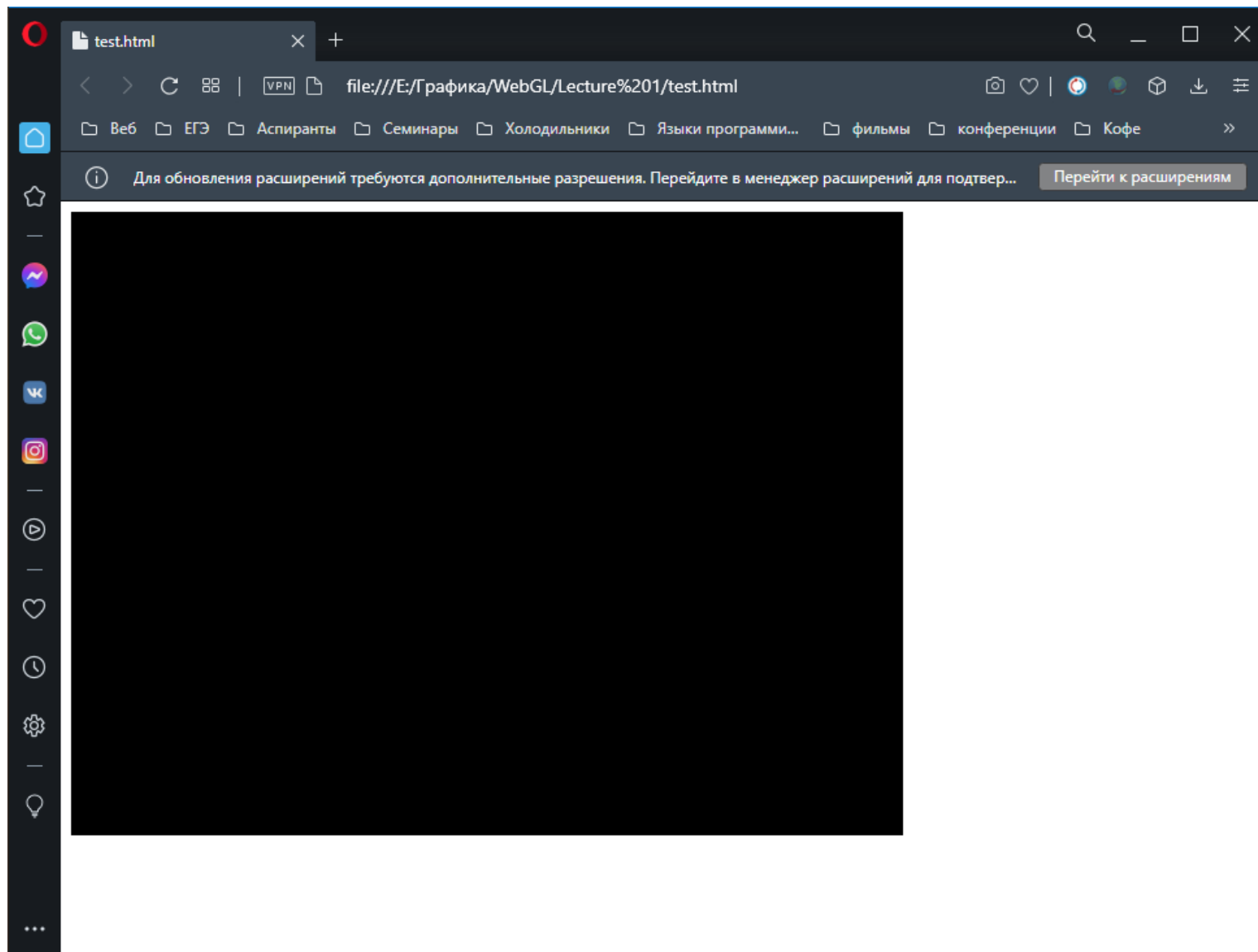
    // установить в качестве цвета очистки буфера цвета черный, полная непрозрачность
    gl.clearColor(0.0, 0.0, 0.0, 1.0);

    // включает использование буфера глубины
    gl.enable(gl.DEPTH_TEST);

    // определяет работу буфера глубины: более ближние объекты перекрывают дальние
    gl.depthFunc(gl.LEQUAL);

    // очистить буфер цвета и буфер глубины
    gl.clear(gl.COLOR_BUFFER_BIT|gl.DEPTH_BUFFER_BIT);

}
} // function start
```



# Вершинный шейдер (js)

```
const vsSource = '  
    attribute vec3 aVertexPosition;  
    attribute vec4 aVertexColor;  
    uniform mat4 mvMatrix;  
    uniform mat4 prMatrix;  
    out vec4 color;  
  
    void main(void) {  
        gl_Position = prMatrix * mvMatrix * vec4 ( aVertexPosition, 1.0 );  
        color = aVertexColor;  
    }  
';
```

# Исходный код вершинного шейдера - новая версия

```
"use strict";
```

```
// Исходный код вершинного шейдера
```

```
const vsSource = `#version 300 es
```

```
// Координаты вершины. Атрибут, инициализируется через буфер.
```

```
in vec2 vertexPosition;
```

```
void main() {
```

```
    gl_Position = vec4(vertexPosition, 0.0, 1.0);
```

```
};
```

# Фрагментный шейдер

```
const fsSource = '  
    #ifdef GL_ES  
    precision highp float;  
    #endif  
    in vec4 color;  
  
    void main(void) {  
        gl_FragColor = color;  
    }  
';
```



# Исходный код фрагментного шейдера - новая версия

```
// Исходный код фрагментного шейдера
const fsSource = '#version 300 es
// WebGL требует явно установить точность флоатов, так что ставим 32 бита
precision mediump float;
out vec4 color;
void main() {
    color = vec4(0, 1, 0, 1);
}';
```

# Вершинный шейдер (html)

```
<script id="shader-vs" type="x-shader/x-vertex">  
  attribute vec3 aVertexPosition;  
  
  uniform mat4 uMVMatrix;  
  uniform mat4 uPMatrix;  
  
  void main(void) {  
    gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);  
  }  
</script>
```

# Фрагментный шейдер

```
<script id="shader-fs" type="x-shader/x-fragment">  
  void main(void) {  
    gl_FragColor = vec4(1.0, 1.0, 1.0, 1.0);  
  }  
</script>
```

# Инициализация шейдеров

```
// внутри main()
```

```
const shaderProgram = initShaderProgram(gl, vsSource, fsSource);
```

# initShaderProgram

```
function initShaderProgram(gl, vsSource, fsSource) {  
    const vertexShader = loadShader(gl, gl.VERTEX_SHADER, vsSource);  
    const fragmentShader = loadShader(gl, gl.FRAGMENT_SHADER, fsSource);  
  
    // Create the shader program  
  
    ...  
  
}
```

# Загрузка шейдеров

```
function loadShader(gl, type, source) {  
  
    const shader = gl.createShader(type);  
  
    // Send the source to the shader object  
    gl.shaderSource(shader, source);  
  
    // Compile the shader program  
    gl.compileShader(shader);  
  
    // See if it compiled successfully  
    if (!gl.getShaderParameter(shader, gl.COMPILE_STATUS)) {  
        alert('An error occurred compiling the shaders: ' + gl.getShaderInfoLog(shader));  
        gl.deleteShader(shader);  
        return null;  
    }  
    return shader;  
}
```

# Create the shader program

...

```
// Create the shader program  
const shaderProgram = gl.createProgram();
```

```
gl.attachShader(shaderProgram, vertexShader);  
gl.attachShader(shaderProgram, fragmentShader);
```

```
gl.linkProgram(shaderProgram);
```

```
// If creating the shader program failed, alert  
if (!gl.getProgramParameter(shaderProgram, gl.LINK_STATUS)) {  
    alert('Unable to initialize the shader program: ' + gl.getProgramInfoLog(shaderProgram));  
    return null;  
}  
return shaderProgram;
```

```
}
```

# Использование шейдерной программы

```
gl.useProgram(shaderProgram);
```

```
vertexPositionAttribute = gl.getAttribLocation(shaderProgram, "aVertexPosition");  
gl.enableVertexAttribArray(vertexPositionAttribute);
```



# Создание объекта

```
function initBuffers() {
    squareVerticesBuffer = gl.createBuffer();

    gl.bindBuffer(gl.ARRAY_BUFFER, squareVerticesBuffer);

    var vertices = [
        1.0, 1.0, 0.0,
        -1.0, 1.0, 0.0,
        1.0, -1.0, 0.0,
        -1.0, -1.0, 0.0
    ];

    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices), gl.STATIC_DRAW);
}
```

Вначале метод `createBuffer()` объекта `gl` создает буфер, в котором мы будем хранить вершины.

Затем буфер привязывается к контексту, вызовом метода `bindBuffer()`.

Затем создаем JavaScript массив, содержащий координаты для каждой вершины квадрата.

Затем этот массив преобразуется в массив вещественных чисел WebGL и передается в метод `bufferData()` объекта `gl` для назначения вершин объекту.

# Отрисовка сцены без анимации

```
function drawScene() {  
    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
```

```
    perspectiveMatrix = makePerspective(45, 640.0/480.0, 0.1, 100.0);
```

```
    loadIdentity();
```

```
    mvTranslate([-0.0, 0.0, -6.0]);
```

угол обзора -- 45°

соотношение w к h -- 640/480 (размеры canvas)

near -- 0.1

far -- 100 единиц от камеры

```
    gl.bindBuffer(gl.ARRAY_BUFFER, squareVerticesBuffer);
```

```
    gl.vertexAttribPointer(vertexPositionAttribute, 3, gl.FLOAT, false, 0, 0);
```

```
    setMatrixUniforms();
```

```
    gl.drawArrays(gl.TRIANGLE_STRIP, 0, 4);
```

```
}
```

# gl.vertexAttribPointer(index, size, type, norm, stride, offset)

```
gl.vertexAttribPointer(vertexPositionAttribute, 3, gl.FLOAT, false, 0, 0);
```

index: индекс атрибута, который мы сопоставляем с буфером вершин

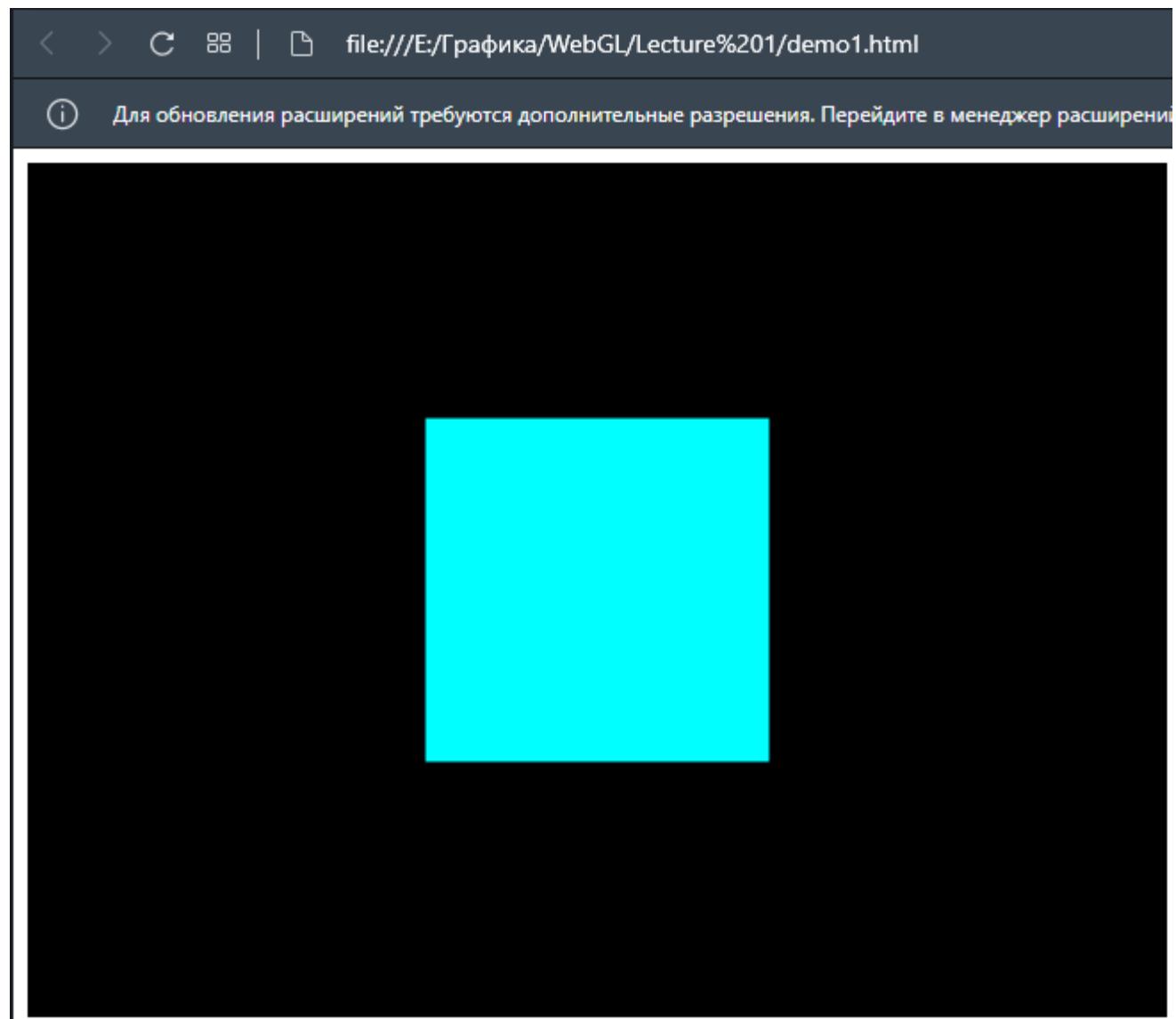
size: число значений для каждой вершины, которые хранятся в буфере. Например, у нас три координаты на вершину, поэтому и число передаваемое в качестве данного параметра будет равно 3

type: тип значений, который хранятся в буфере. В качестве значения параметр может принимать следующие константы: FIXED, BYTE, UNSIGNED\_BYTE, FLOAT, SHORT и UNSIGNED\_SHORT

norm: True, если будут нормализованы

stride: шаг между значениями. При установке в качестве значения 0 элементы будут последовательно обрабатываться

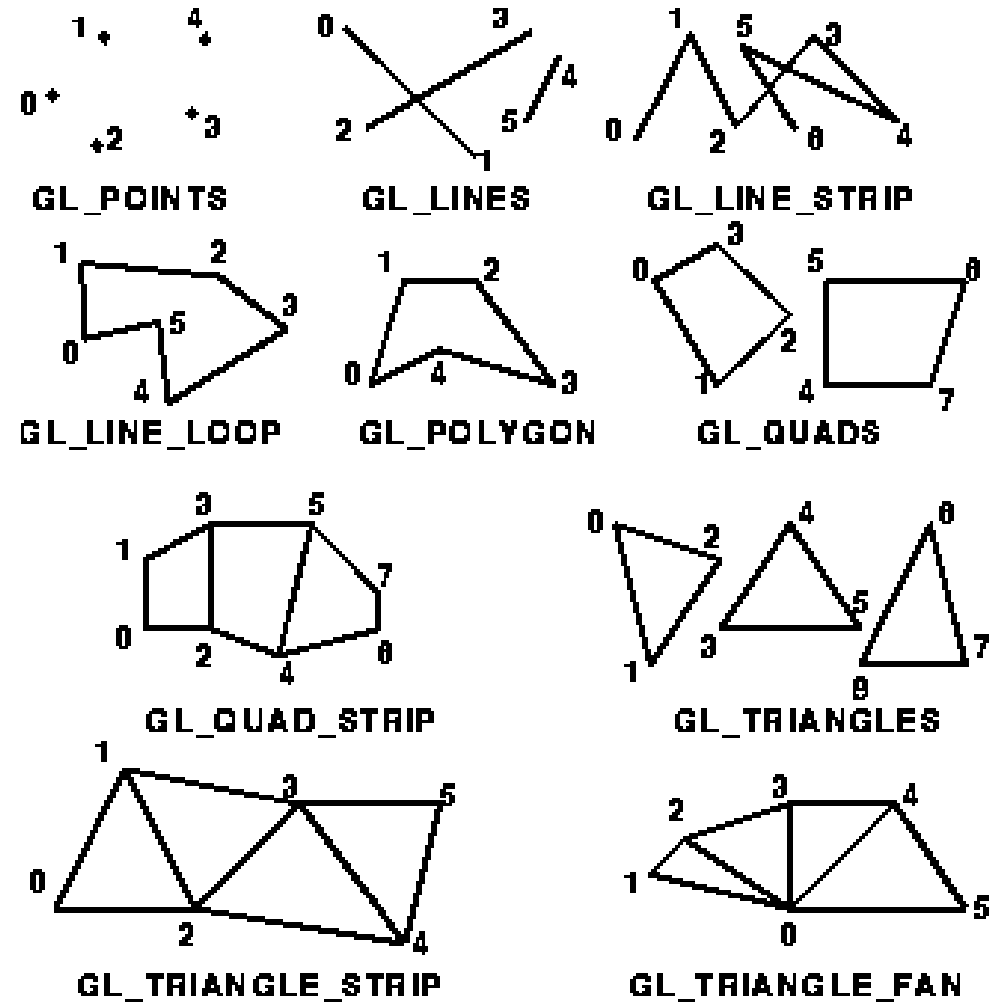
offset: смещение - позиция в буфере, с которой начинается обработка

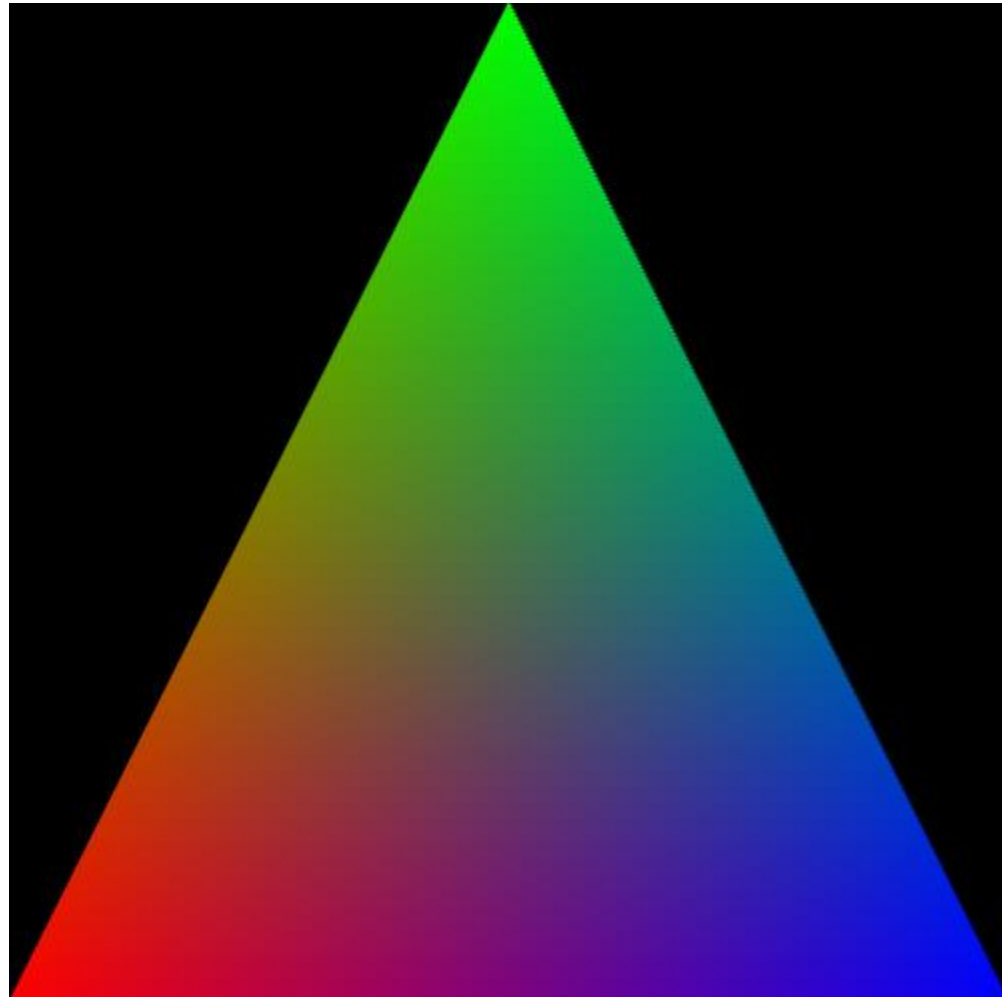


glDrawArrays(тип, смещение, количество);

```
gl.drawArrays(gl.TRIANGLE_STRIP, 0, 4);
```

```
glDrawArrays(GL_TRIANGLES, 0, n*3);
```





```
// Треугольник
```

```
// Превращаем в плоский массив
```

```
const positions = [ [-1, -1], [+0, +1], [+1, -1], ].flat()
```

```
// Координаты вершин
```

```
// Цвета вершин
```

```
const colors = [ [1, 0, 0, 1], [0, 1, 0, 1], [0, 0, 1, 1], ].flat()
```

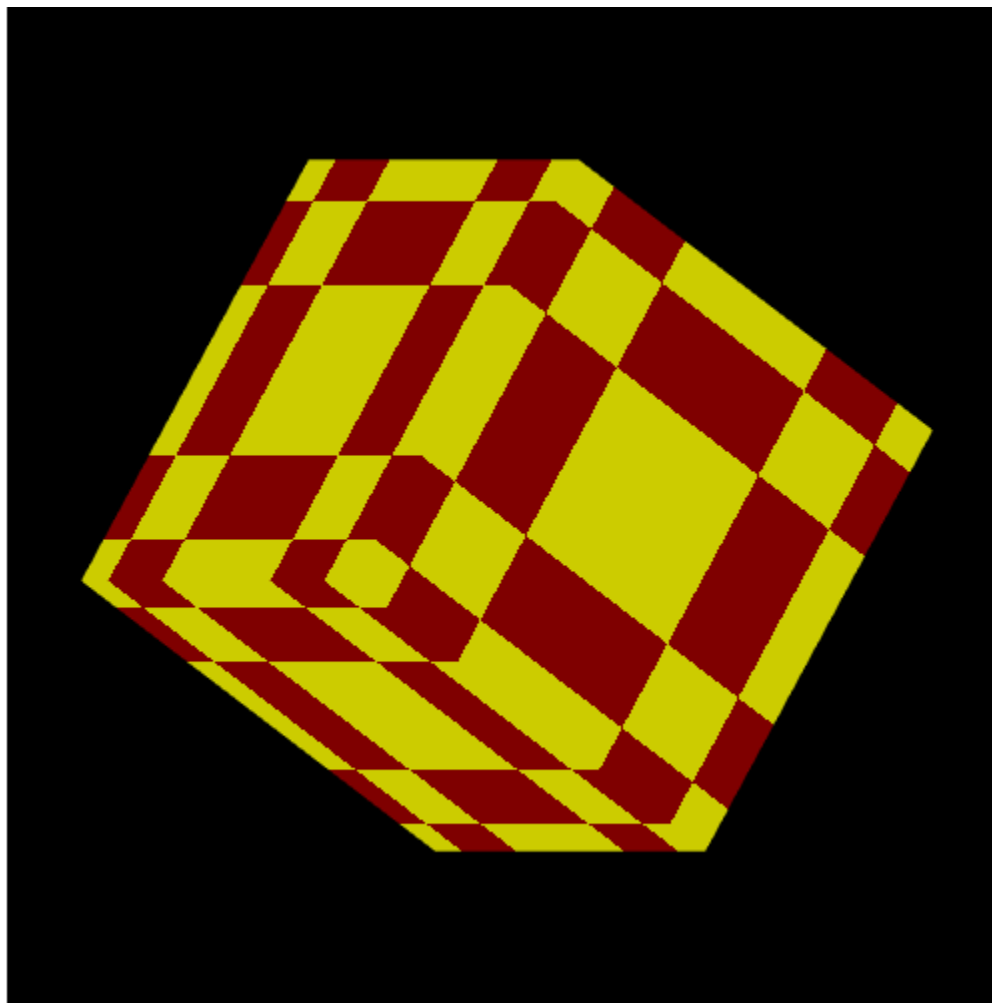
```
const positionBuffer = makeF32ArrayBuffer(gl, positions);
```

```
const colorBuffer = makeF32ArrayBuffer(gl, colors);
```

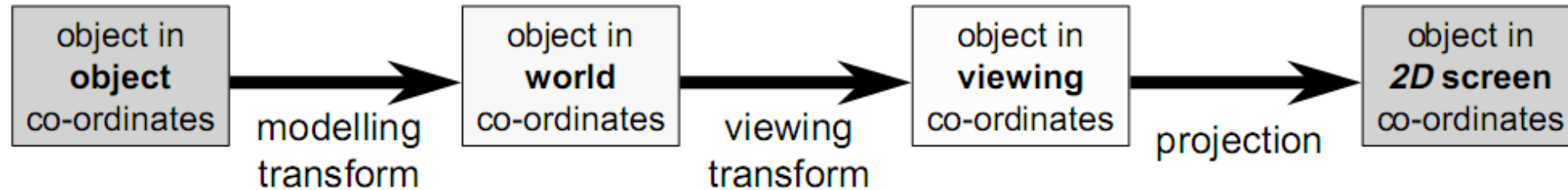
```
function makeF32ArrayBuffer(gl, array) {  
  
    // Создаём буфер  
    const buffer = gl.createBuffer();  
  
    gl.bindBuffer (gl.ARRAY_BUFFER, buffer);  
  
    // Заполняем буффер массивом флоатов  
    gl.bufferData ( gl.ARRAY_BUFFER, new Float32Array(array), gl.STATIC_DRAW );  
  
    return buffer;  
}
```



```
function drawScene(gl, programInfo, buffers) {  
    // Чистим экран  
    gl.clearColor(0.0, 0.0, 0.0, 1.0);  
    gl.clearDepth(1.0);  
    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);  
  
    // ** Подключаем буфер вершин **  
    ...  
    // ** Подключаем буфер цветов **  
    ...  
    // Устанавливаем используемую программу  
    //Рисуем  
}
```



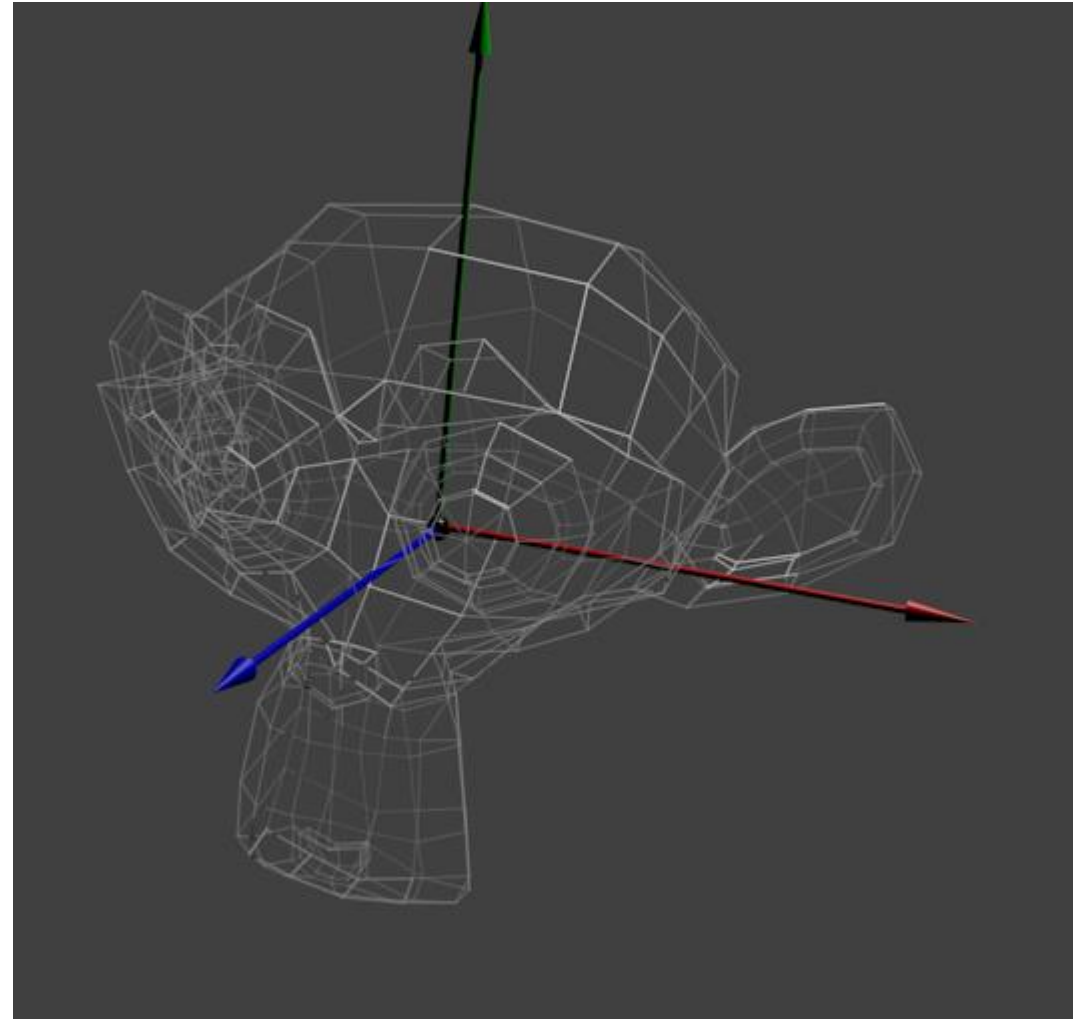
# Разнообразие преобразований – мировая, видовая и проекционная матрицы



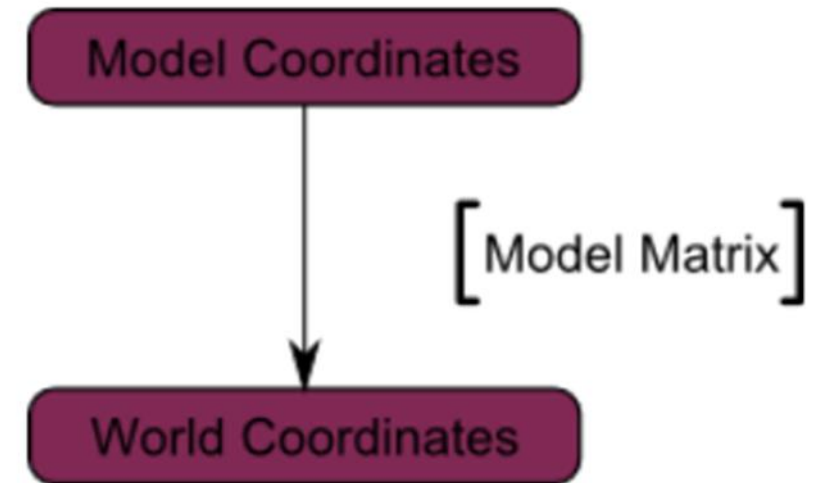
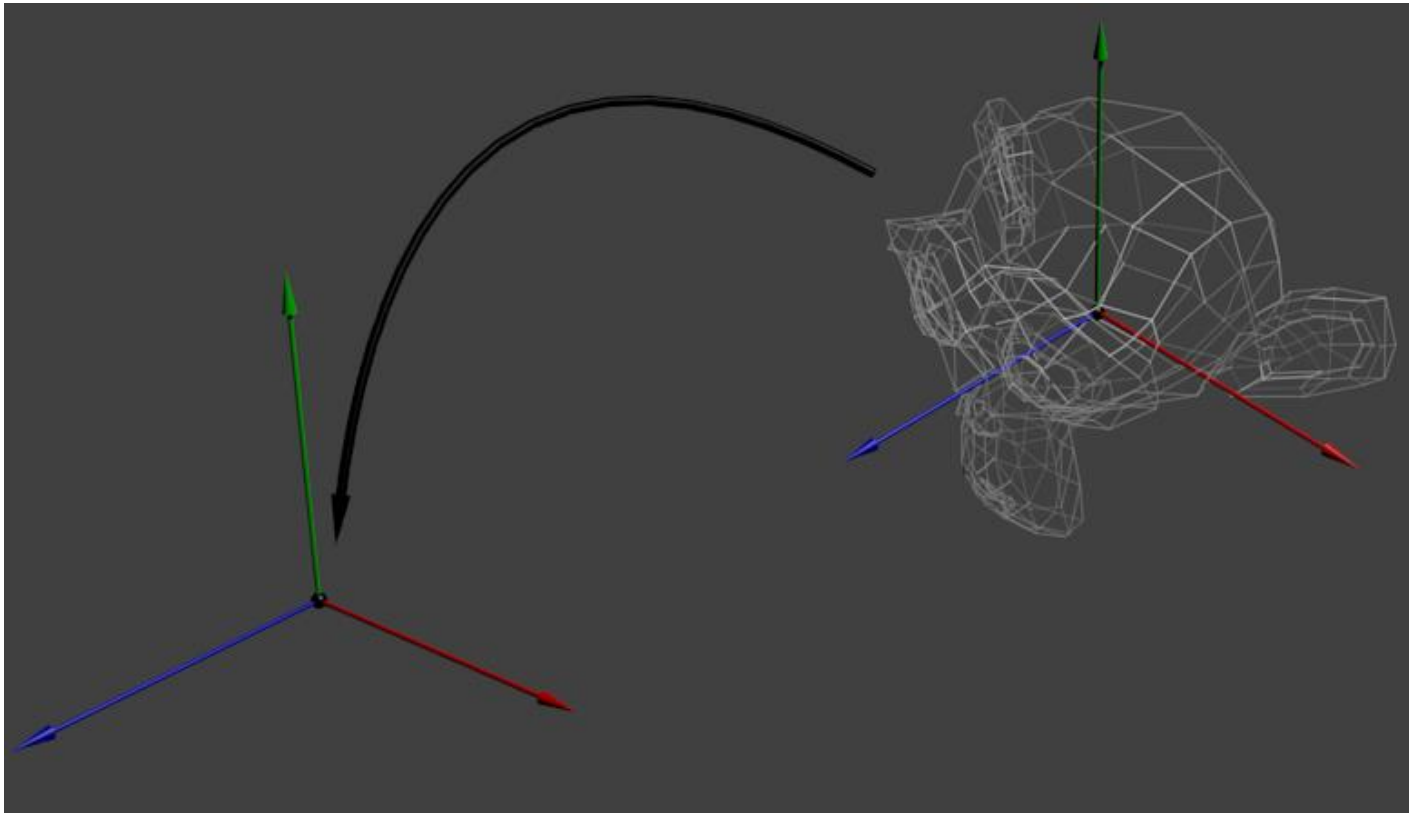
- модельно-видовая как одна матрица (мировая x видовая)
- как мировая, так и видовая могут быть единичными

# Модель в объектных координатах

Модель задается множеством вершин, координаты которых заданы относительно центра объекта, т. е. вершина с координатами  $(0, 0, 0)$  будет находиться в центре объекта.

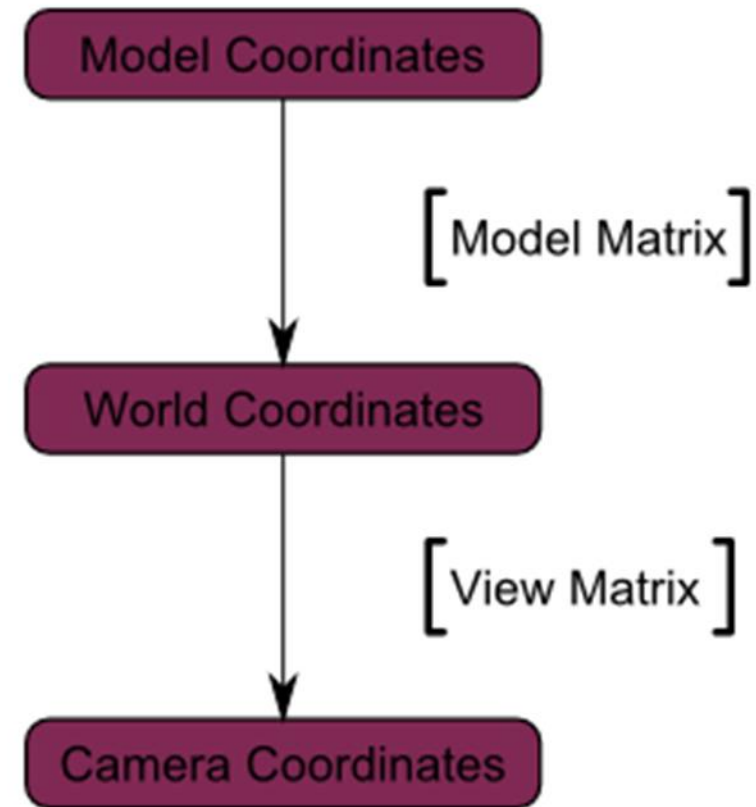
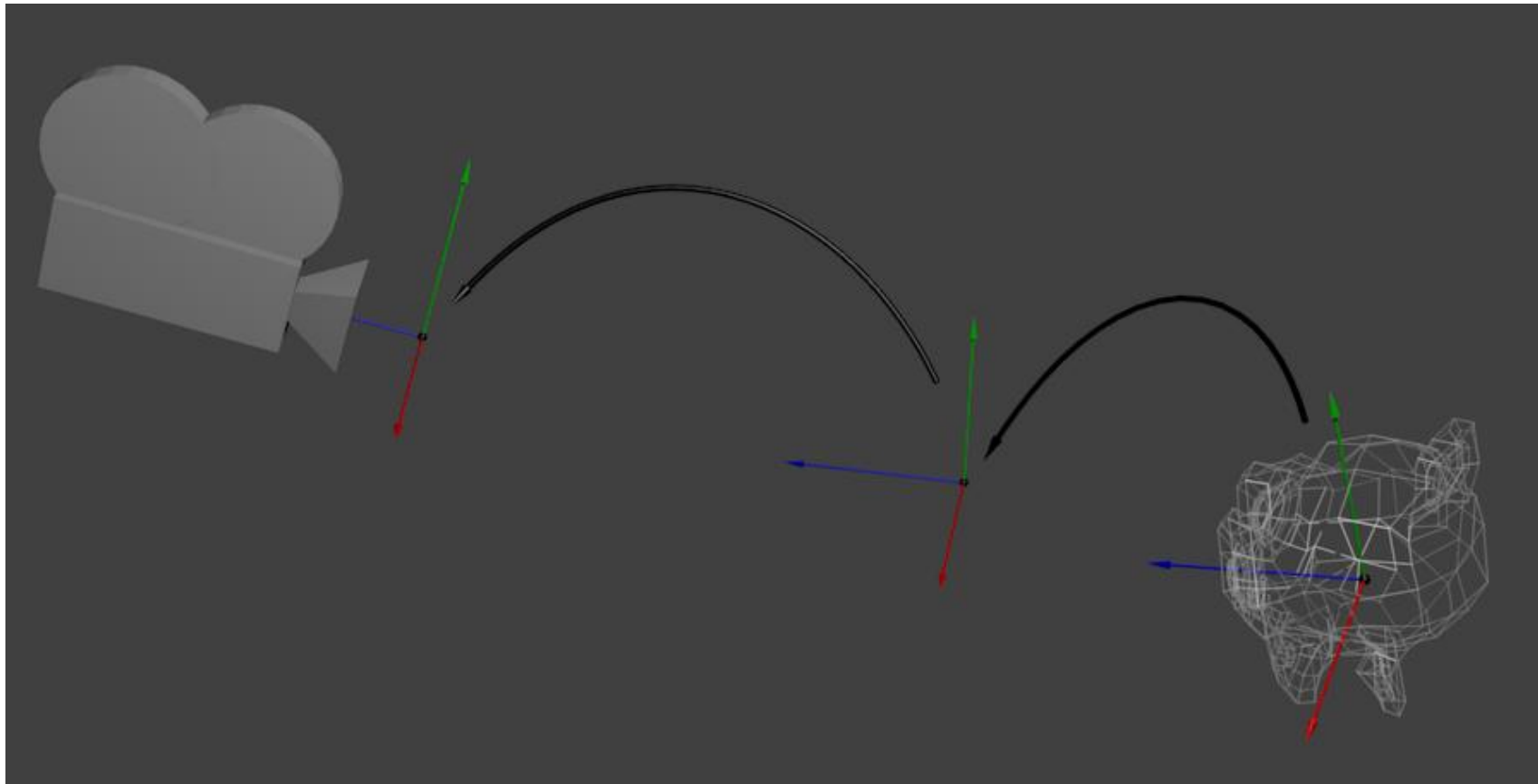


# Преобразования к мировым координатам – мировая матрица



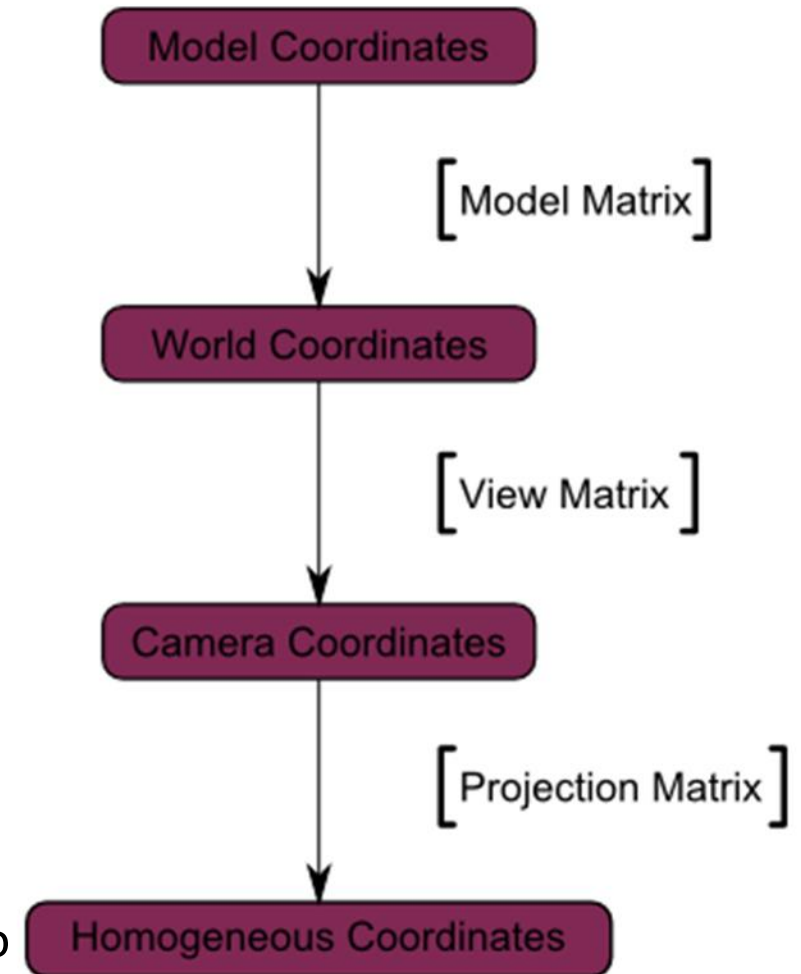
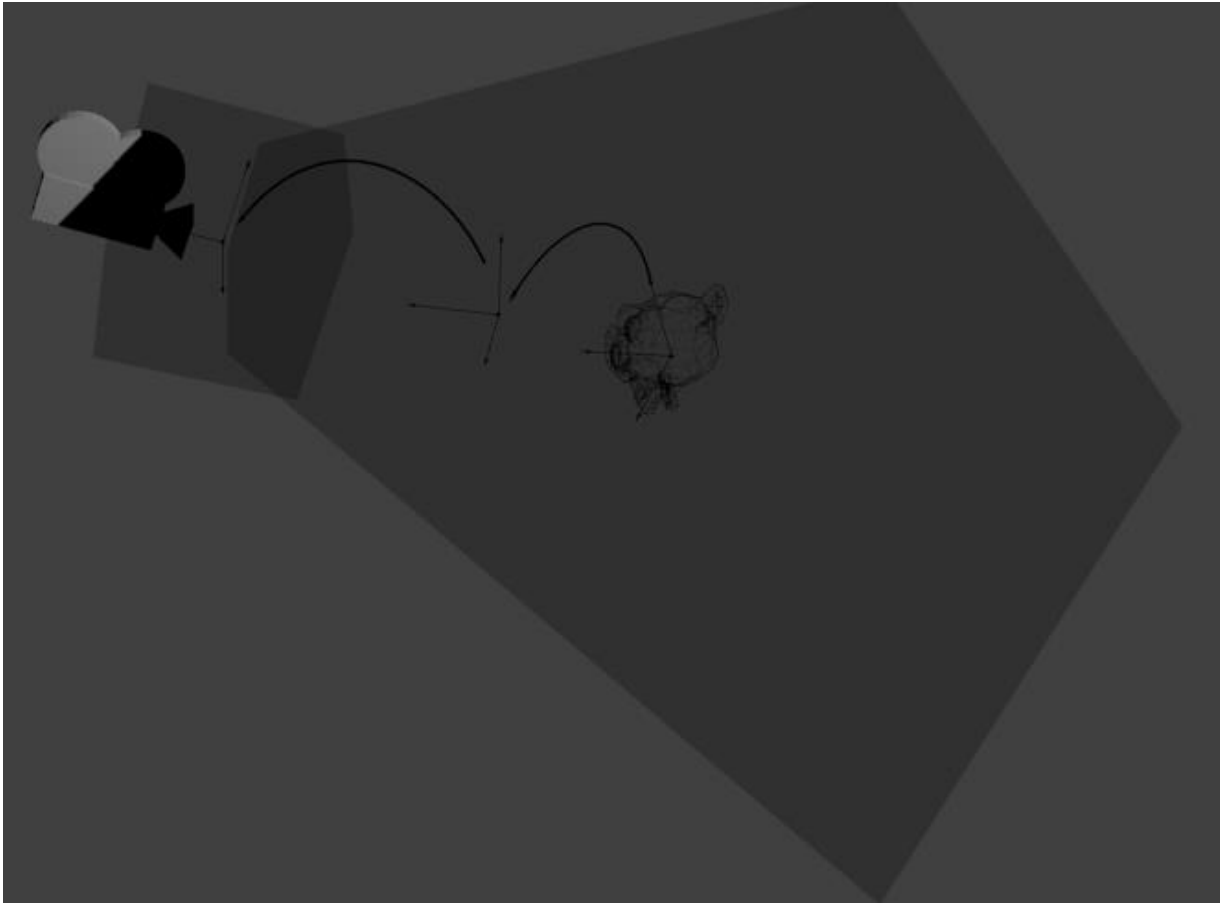
Мы перешли из пространства объекта (все вершины заданы относительно центра объекта) к мировому пространству (все вершины заданы относительно центра мира)

# Переход к системе координат камеры – видовая матрица



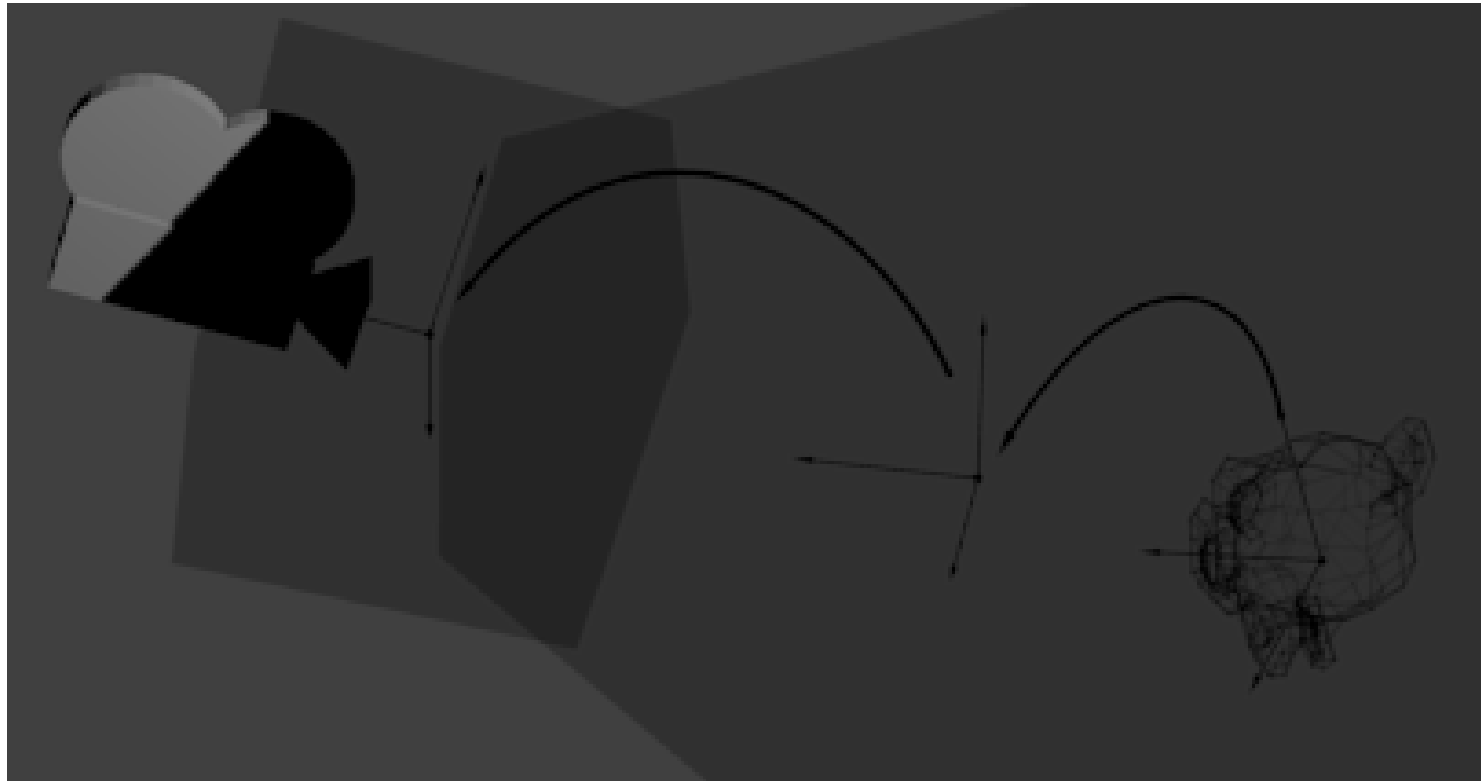
Мы перешли из мировой системы координат (все вершины заданы относительно центра мировой системы) к системе координат камеры (все вершины заданы относительно камеры)

# Переход из пространства камеры в однородное пространство



Мы перешли из Пространства Камеры (все вершины заданы относительно камеры) в Однородное пространство (все вершины находятся в небольшом кубе. Все, что находится внутри куба - выводится на экран).

# Перспективная проекция



Вершина, которая получит координаты  $x == 0$  и  $y == 0$  будет отображаться по центру экрана. Однако, при отображении объекта огромную роль играет также дистанция до камеры ( $z$ ). Для двух вершин, с одинаковыми  $x$  и  $y$ , вершина имеющая большее значение по  $z$  будет отображаться ближе, чем другая.



# Аффинные преобразования

**Определение** Аффинное преобразование  $f: \mathbb{R}^n \rightarrow \mathbb{R}^n$  есть преобразование вида

$$f(x) = M \cdot x + v,$$

где  $M$  — обратимая матрица и  $v \in \mathbb{R}^n$ .

## Базовые аффинные преобразования

- Поворот вокруг начала координат на угол  $\varphi$
- Растяжение/Сжатие
- Отражение
- Перенос

# Однородные координаты точки

- Однородными координатами точки называется любая тройка одновременно не равных нулю чисел  $x_1, x_2, x_3$ , связанных с заданными числами  $x$  и  $y$  следующими соотношениями:
- $x_1 / x_3 = x, x_2 / x_3 = y.$

## Произвольная матрица аффинного преобразования

$$\underline{(x^* \cdot y^* \cdot 1)} = \underline{(x \cdot y \cdot 1)} \begin{bmatrix} \alpha & \gamma & 0 \\ \beta & \delta & 0 \\ \lambda & \mu & 1 \end{bmatrix}$$

- → A. Матрица вращения (rotation) ¶

$$[R] = \begin{bmatrix} \cos(\varphi) & \sin(\varphi) & 0 \\ -\sin(\varphi) & \cos(\varphi) & 0 \\ 0 & 0 & 1 \end{bmatrix} ¶$$

- → B. Матрица растяжения/сжатия (dilatation). ¶

$$[D] = \begin{bmatrix} \alpha & 0 & 0 \\ 0 & \delta & 0 \\ 0 & 0 & 1 \end{bmatrix} ¶$$

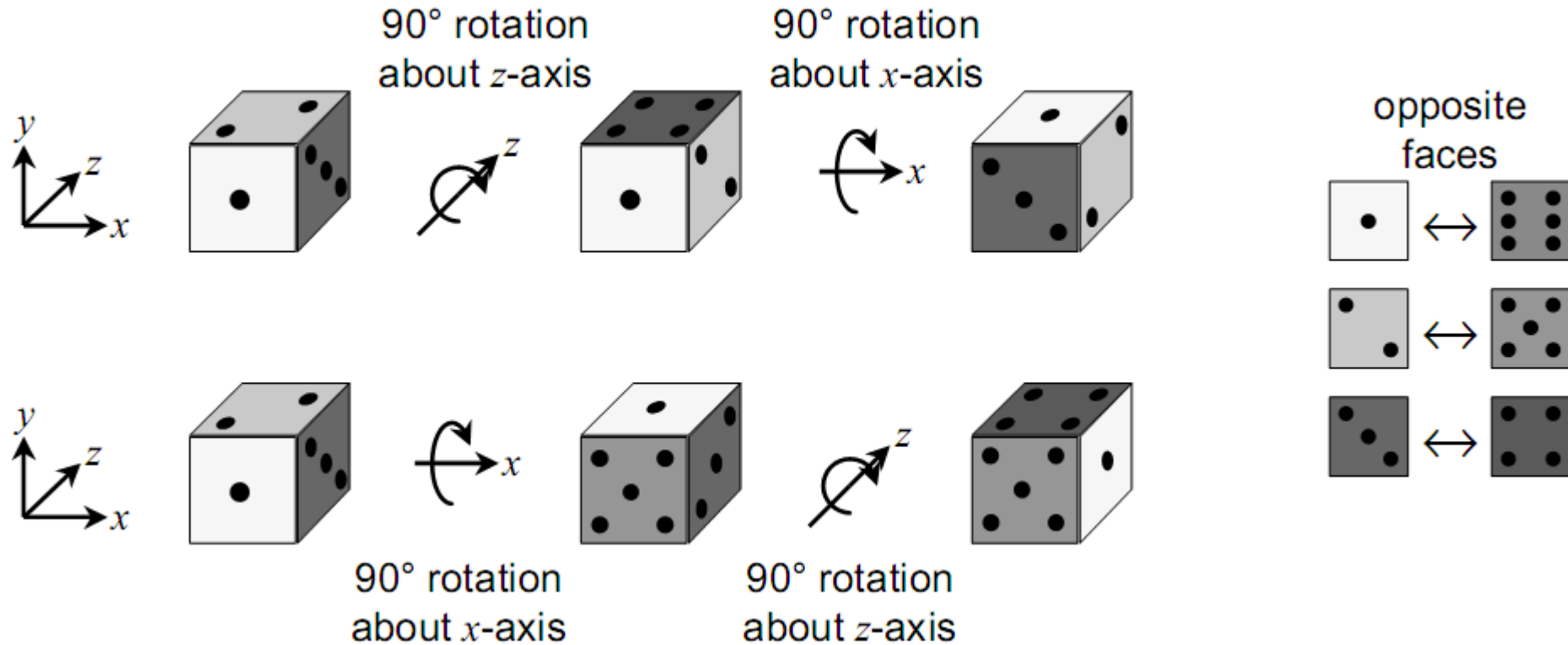
- → C. Матрица отражения (reflection). ¶

$$[M] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} ¶$$

- → D. Матрица переноса (translation). ¶

$$[T] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \lambda & \mu & 1 \end{bmatrix} ¶$$

# Преобразования некоммутативны



# Пример 1

Построить матрицу растяжения с коэффициентами растяжения вдоль оси абсцисс и вдоль оси ординат и с центром в точке  $A(a, b)$ .

**Итоговая матрица**

$$(x^* \quad y^* \quad 1) = (x \quad y \quad 1) \times \begin{bmatrix} \alpha & 0 & 0 \\ 0 & \delta & 0 \\ (1-\alpha) \times a & (1-\delta) \times b & 1 \end{bmatrix}$$

# Решение

- **1-й шаг.** Перенос на вектор  $-A(-a, -b)$  для совмещения центра растяжения с началом координат

$$[T_{-A}] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -a & -b & 1 \end{bmatrix}$$

- **2-й шаг.** Растяжение вдоль координатных осей с коэффициентами  $\alpha$  и  $\delta$  соответственно.

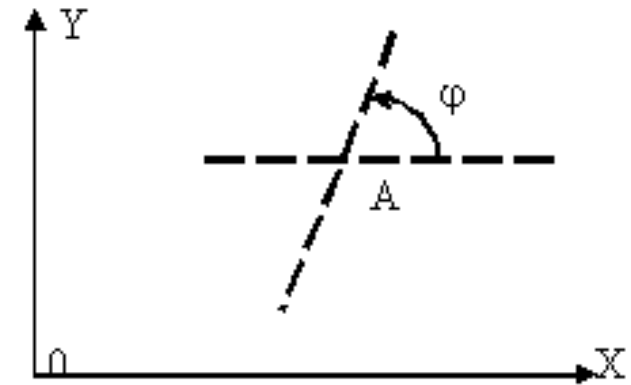
$$[D] = \begin{bmatrix} \alpha & 0 & 0 \\ 0 & \delta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- **3-й шаг.** Перенос на вектор  $A(a, b)$  для возвращения центра растяжения в прежнее положение.

$$[D] = \begin{bmatrix} \alpha & 0 & 0 \\ 0 & \delta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

## Пример2

- Построить матрицу поворота вокруг точки  $A(a, b)$  на угол  $\varphi$

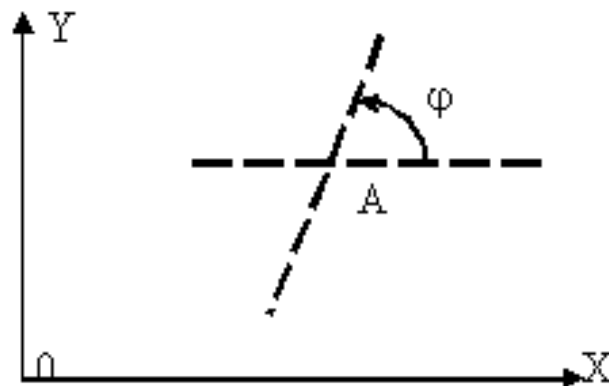


$$\begin{pmatrix} x^* & y^* & 1 \end{pmatrix} = \begin{pmatrix} x & y & 1 \end{pmatrix} \times \begin{bmatrix} \cos(\varphi) & \sin(\varphi) & 0 \\ -\sin(\varphi) & \cos(\varphi) & 0 \\ -a \times \cos(\varphi) + b \times \sin(\varphi) + a & -a \times \sin(\varphi) - b \times \cos(\varphi) + b & 1 \end{bmatrix}$$

# Решение: 1-й шаг

Перенос на вектор  $A(-a, -b)$  для совмещения центра поворота с началом координат.

$$[T_{-A}] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -a & -b & 1 \end{bmatrix}$$

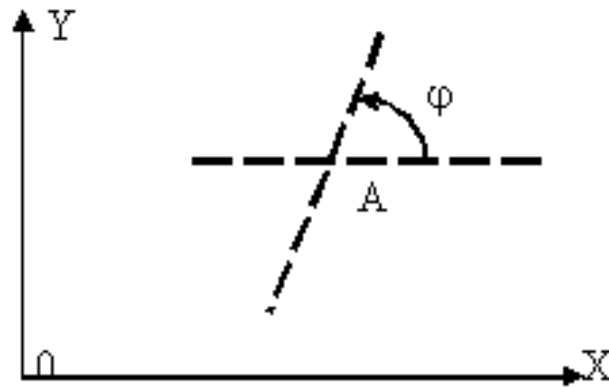




# Решение: 2-й шаг

Поворот на угол  $\varphi$ .

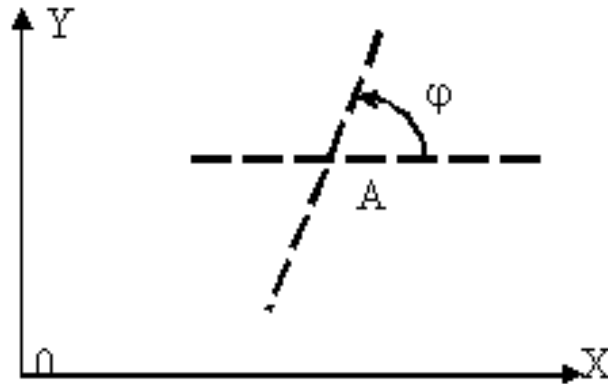
$$[R_{\varphi}] = \begin{bmatrix} \cos(\varphi) & \sin(\varphi) & 0 \\ -\sin(\varphi) & \cos(\varphi) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



## Решение: 3-й шаг

Перенос на вектор  $A(a,b)$  для возвращения центра поворота в прежнее положение

$$[T_A] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ a & b & 1 \end{bmatrix}$$



# Итоговая матрица

$$[T_{-d}] [R_{\varphi}] [T_d]$$

$$[T_{-d}] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -a & -b & 1 \end{bmatrix} \quad [R_{\varphi}] = \begin{bmatrix} \cos(\varphi) & \sin(\varphi) & 0 \\ -\sin(\varphi) & \cos(\varphi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad [T_d] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ a & b & 1 \end{bmatrix}$$

$$(x^* \ y^* \ 1) = (x \ y \ 1) \times \begin{bmatrix} \cos(\varphi) & \sin(\varphi) & 0 \\ -\sin(\varphi) & \cos(\varphi) & 0 \\ -a \times \cos(\varphi) + b \times \sin(\varphi) + a & -a \times \sin(\varphi) - b \times \cos(\varphi) + b & 1 \end{bmatrix}$$

# Базовые аффинные преобразования (при умножении слева на вектор-строку)

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \alpha & \beta & \gamma & 1 \end{bmatrix}$$

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

$$\begin{bmatrix} \alpha & 0 & 0 & 0 \\ 0 & \beta & 0 & 0 \\ 0 & 0 & \gamma & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

# Базовые аффинные преобразования (при умножении слева на вектор-строку)

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & \sin \theta & 0 \\ 0 & -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} \cos \theta & 0 & -\sin \theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin \theta & 0 & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} \cos \theta & \sin \theta & 0 & 0 \\ -\sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

# Базовые аффинные преобразования (при умножении справа на вектор-столбец)

translation

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

identity

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

rotation about  $x$ -axis

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta & 0 \\ 0 & \sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

scale

$$\begin{bmatrix} m_x & 0 & 0 & 0 \\ 0 & m_y & 0 & 0 \\ 0 & 0 & m_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

rotation about  $z$ -axis

$$\begin{bmatrix} \cos \theta & -\sin \theta & 0 & 0 \\ \sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

rotation about  $y$ -axis

$$\begin{bmatrix} \cos \theta & 0 & \sin \theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \theta & 0 & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

# Задача

- Повернуть объект вокруг произвольной прямой  $L$  в пространстве на заданный угол.
- Объект задаётся списком вершин и списком рёбер.
- Прямая задаётся точкой  $A(a,b,c)$ , через которую она проходит, и единичным вектором  $(l,m,n)$ .

# Результирующая матрица

$$\begin{pmatrix} l^2 + \cos \varphi (1 - l^2) & l(1 - \cos \varphi)m + n \sin \varphi & l(1 - \cos \varphi)n - m \sin \varphi & 0 \\ l(1 - \cos \varphi)m - n \sin \varphi & m^2 + \cos \varphi (1 - m^2) & m(1 - \cos \varphi)n + l \sin \varphi & 0 \\ l(1 - \cos \varphi)n + m \sin \varphi & m(1 - \cos \varphi)n - l \sin \varphi & n^2 + \cos \varphi (1 - n^2) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



# План решения

1. Перенести прямую  $L$  в центр координат на  $-A (-a, -b, -c)$
2. Совместить прямую  $L$  с одной из координатных осей
3. Выполнить поворот объекта вокруг прямой  $L$
4. Выполнить преобразования 1 и 2 в обратной последовательности

# Решение

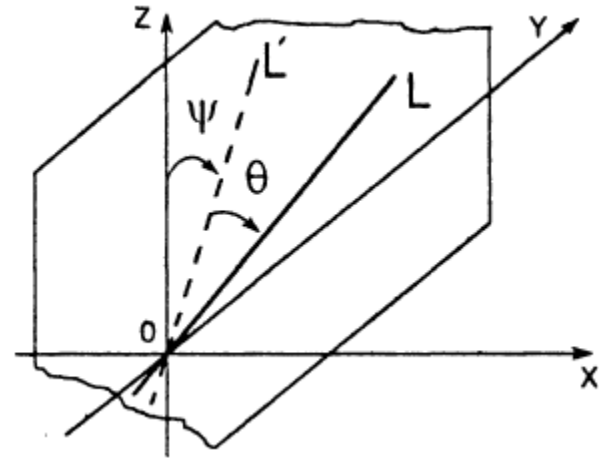
1. Перенести прямую  $L$  в центр координат на  $-A (-a, -b, -c)$
2. Совместить прямую  $L$  с одной из координатных осей, например,  $Z$ 
  - Повернуть прямую  $L$  вокруг  $Ox$
  - Повернуть прямую  $L$  вокруг  $Oy$
3. Выполнить поворот объекта вокруг прямой  $L$
4. Выполнить повороты 2 в обратной последовательности на обратные углы
5. Выполнить перенос на  $A (a, b, c)$

Перенос на  $-A$  ( $-a, -b, -c$ )

$$[T] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -a & -b & -c & 1 \end{bmatrix}$$

# Совмещение прямой L с осью Z

- Повернуть прямую L вокруг  $Ox$  на угол  $\psi$
- Повернуть прямую L вокруг  $Oy$  на угол  $\theta$

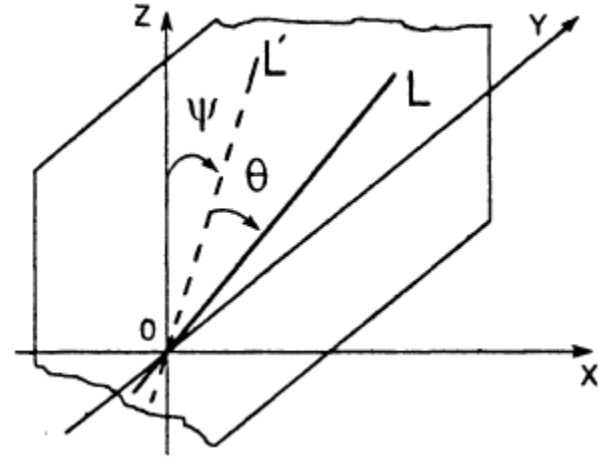


# Поворот прямой L вокруг Oх на угол $\psi$

- Рассмотрим L' – проекцию на YZ – (0,m,n)

$$[R_x] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{n}{d} & \frac{m}{d} & 0 \\ 0 & -\frac{m}{d} & \frac{n}{d} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$(l, m, n, 1)[R_x] = (l, 0, d, 1)$$



$$\cos \psi = \frac{n}{d}, \quad \sin \psi = \frac{m}{d},$$

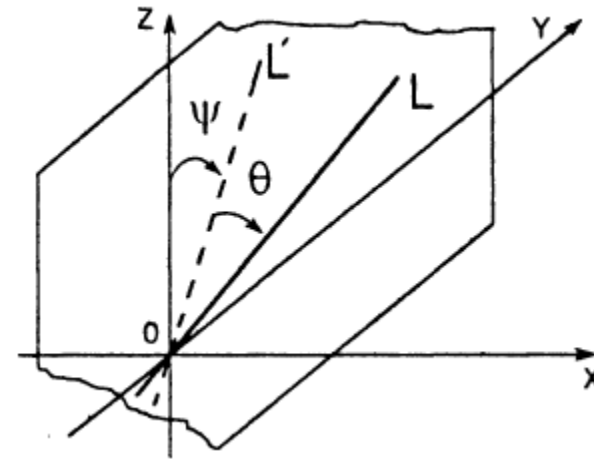
$$d = \sqrt{m^2 + n^2}.$$

# Поворот прямой L вокруг Oy на угол $\theta$

$$(l, m, n, l)[R_x] = (l, 0, d, l)$$

$$\cos \theta = l, \quad \sin \theta = -d.$$

$$[R_y] = \begin{bmatrix} l & 0 & d & 0 \\ 0 & 1 & 0 & 0 \\ -d & 0 & l & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



$$\cos \psi = \frac{n}{d}, \quad \sin \psi = \frac{m}{d},$$

$$d = \sqrt{m^2 + n^2}.$$

Поворот объекта вокруг прямой L на угол  $\varphi$

$$[R_z] = \begin{bmatrix} \cos \varphi & \sin \varphi & 0 & 0 \\ -\sin \varphi & \cos \varphi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

# Обратные преобразования

- Поворот прямой L вокруг O<sub>x</sub> на угол -θ
- Поворот прямой L вокруг O<sub>y</sub> на угол -ψ
- Перенос на A (a, b, c)

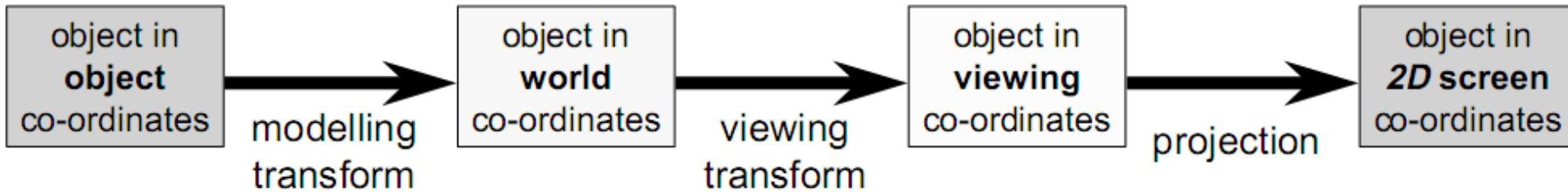
$$[T][R_x][R_y][R_z][R_y]^{-1}[R_x]^{-1}[T]^{-1}$$



# Результирующая матрица

$$\begin{pmatrix} l^2 + \cos \varphi (1 - l^2) & l(1 - \cos \varphi)m + n \sin \varphi & l(1 - \cos \varphi)n - m \sin \varphi & 0 \\ l(1 - \cos \varphi)m - n \sin \varphi & m^2 + \cos \varphi (1 - m^2) & m(1 - \cos \varphi)n + l \sin \varphi & 0 \\ l(1 - \cos \varphi)n + m \sin \varphi & m(1 - \cos \varphi)n - l \sin \varphi & n^2 + \cos \varphi (1 - n^2) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

# Разнообразии преобразований – мировая, видовая и проекционная матрицы



- **модельно-видовая как одна матрица (мировая x видовая)**
- **как мировая, так и видовая могут быть единичными**