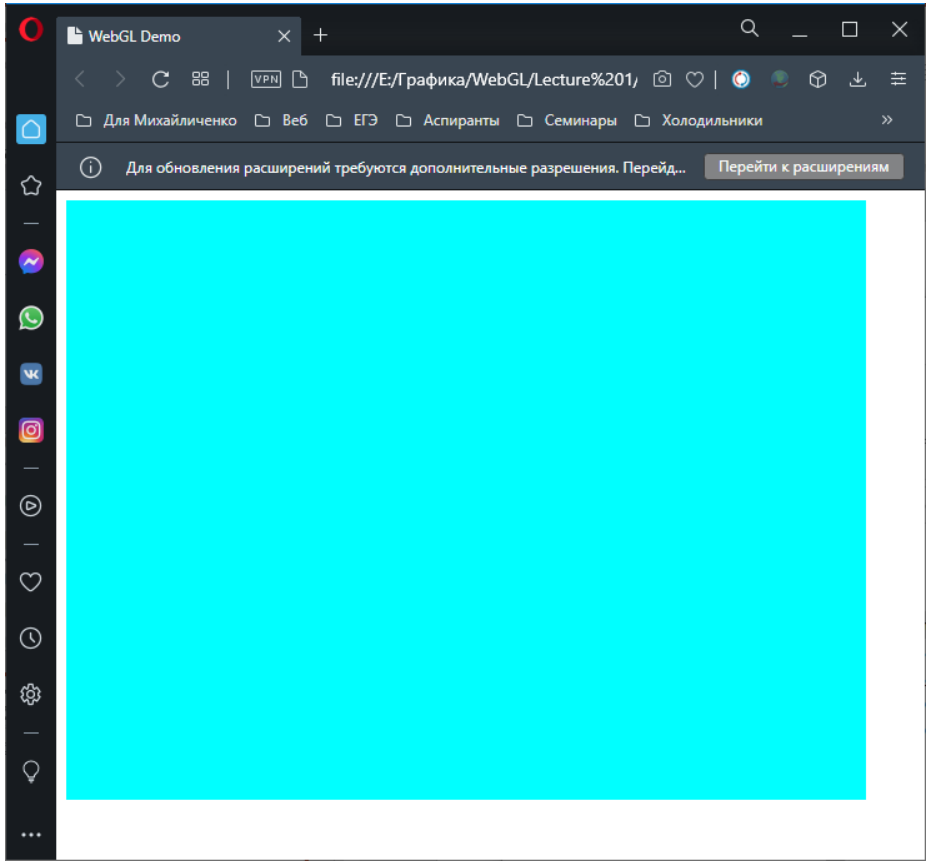
Компьютерная графика
WebGL

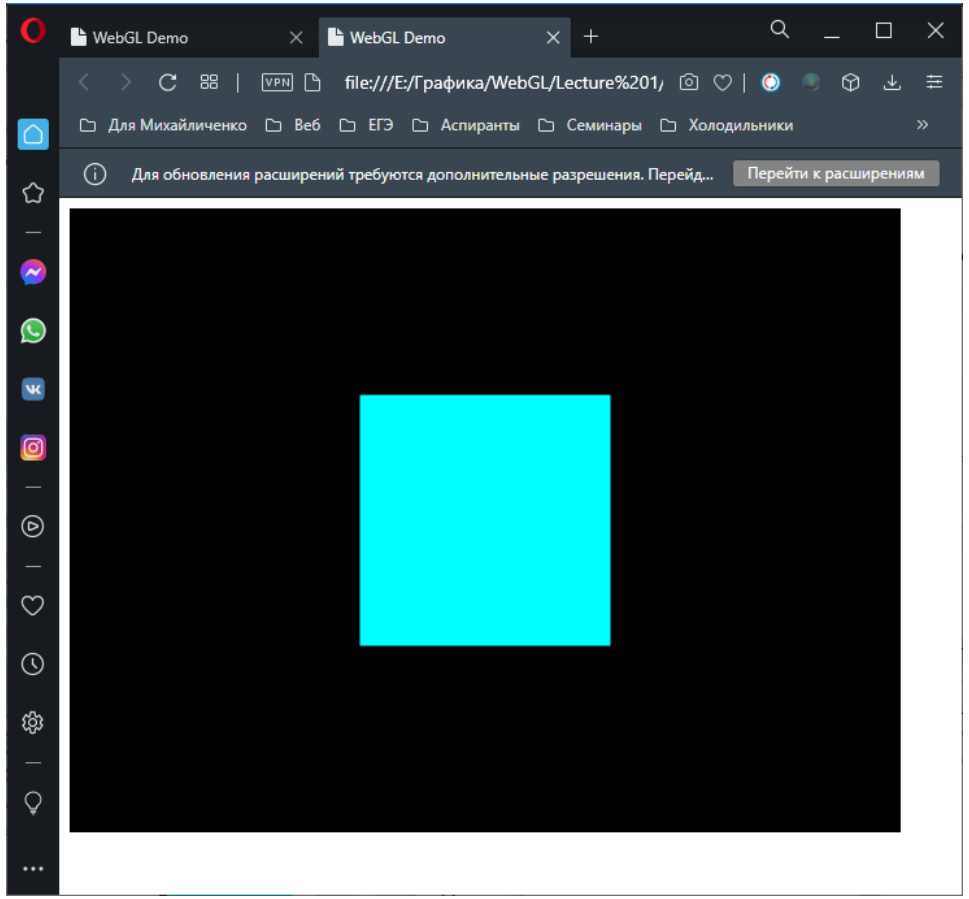
Лекция 2

Демяненко Я.М. ЮФУ 2025

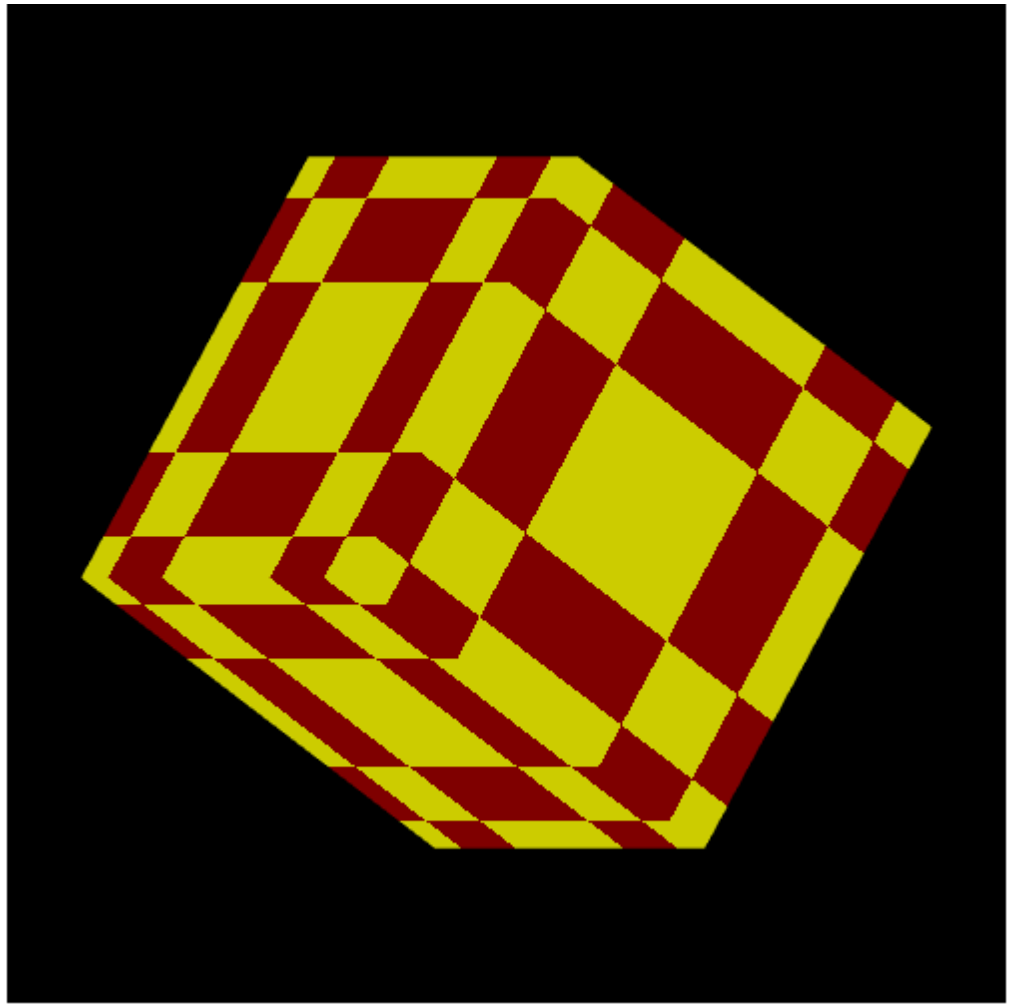
```
 // Vertex shader program
const vsSource = `
    attribute vec4 aVertexPosition;
    uniform mat4 uModelViewMatrix;
    uniform mat4 uProjectionMatrix;
    void main() {
        gl_Position = aVertexPosition;
} `;

// Fragment shader program
    const fsSource = `
    void main() {
        gl_FragColor = vec4(0.0, 1.0, 1.0, 1.0);
     }
`;
```



```
 // Vertex shader program
const vsSource = `
   attribute vec4 aVertexPosition;
   uniform mat4 uModelViewMatrix;
   uniform mat4 uProjectionMatrix;
   void main() {
      gl_Position = uProjectionMatrix * uModelViewMatrix * aVertexPosition;
} `;

// Fragment shader program
   const fsSource = `
   void main() {
       gl_FragColor = vec4(0.0, 1.0, 1.0, 1.0);
    }
`;
```



Что хотим получить?



Исходный код фрагментного шейдера

```
// Исходный код вершинного шейдера
const vsSource = `#version 300 es
// Координаты вершины. Атрибут, инициализируется через буфер
in vec3 vertexPosition;
// Выходной параметр с координатами вершины, интерполируется и передаётся во фрагментный шейдер
 out vec3 vPosition;
void main() {
…
}`;
```

Исходный код вершинного шейдера - продолжениес

```
void main() {
 // углы поворота    float x_angle = 1.0;    float y_angle = 1.0;
mat3 transform = mat3(  1, 0, 0,
                              0, cos(x_angle),  sin(x_angle),
                                   0, -sin(x_angle), cos(x_angle)    )
                         * mat3( cos(y_angle), 0, sin(y_angle),
                                      0, 1, 0,
                                     -sin(y_angle), 0, cos(y_angle)    );
    // Поворачиаем вершину
   gl_Position = vec4(vertexPosition * transform, 1.0);
    // Передаём непреобразованную координату во фрагментный шейдер
    vPosition = vertexPosition;
}`;
```

Исходный код фрагментного шейдера

```
const fsSource = `#version 300 es
precision mediump float;
// Интерполированные непреобразованные координаты вершины,
// передаются из вершинного шейдера
in vec3 vPosition;
// Цвет, которым будем отрисовывать
out vec4 color;

 void main() {
…
}`;
```

Формируем заливку квадратиками

```
void main() {
  float k = 5.0;
  int sum = int(vPosition.x * k) + int(vPosition.y * k) + int(vPosition.z * k);
  if ( (sum - (sum / 2 * 2)) == 0 ) {
      color = vec4(0.8, 0.8, 0, 1);
  }
  else {
      color = vec4(0.5, 0.0, 0, 1);
   }
}`;
```

Заполнение матриц

Матрицы можно заполнять вручную или с использованием библиотек, например,
glMatrix Sylvester
и другие

Операции с матрицами

mat4.identity(mvMatrix); //https://glmatrix.net/docs/v4/classes/Mat4.html

Метод **mat4.translate(output, input, vec)**,
где output — итоговая выходная матрица, которая получается после перемещения матрицы input на трёхмерный вектор vec

Метод mat4.**rotate(output, input, rad, axis)** , где output — итоговая матрица, которая получается поворотом матрицы input на угол rad (в радианах) вокруг оси axis.

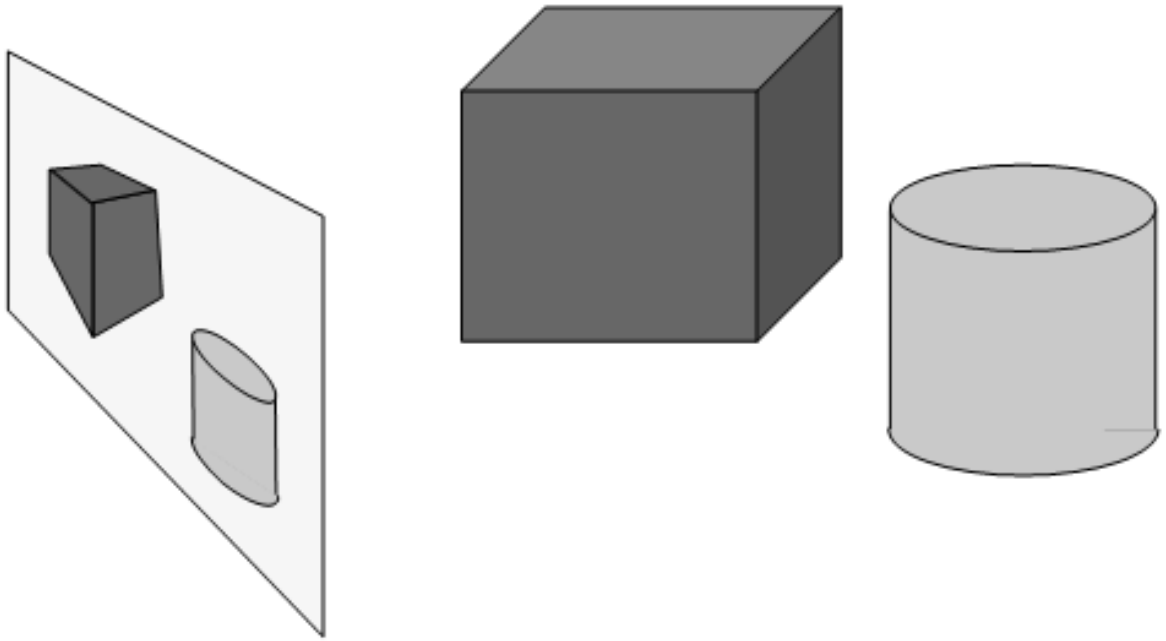
Или
mat4.rotateX(output, input, rad),
mat4.rotateY(output, input, rad)
mat4.rotateZ(output, input, rad)

Метод **mat4.scale(output, input, vec)**. Вектор vec указывает масштаб, на который изменяются значения матрицы input

Аффинные преобразования

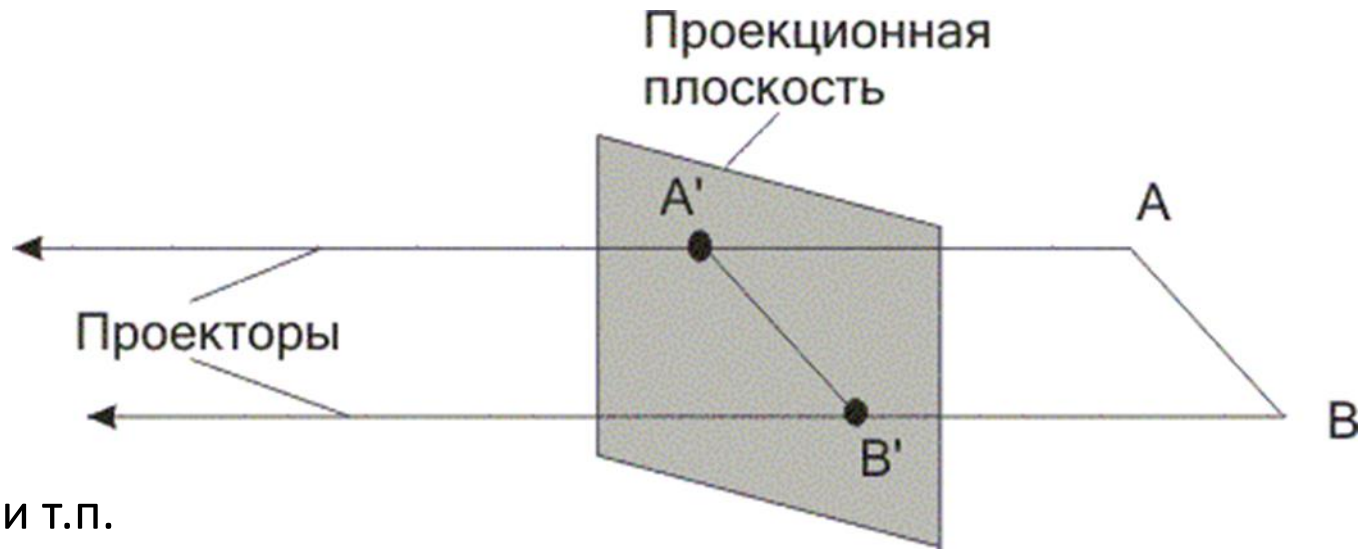
```
 // Set the drawing position to the "identity" point, which is  the center of the scene.
const modelViewMatrix = mat4.create();

// Now move the drawing position a bit to where we want to start drawing the square.
mat4.translate(modelViewMatrix,     // destination matrix
                modelViewMatrix,     // matrix to translate
                [-0.0, 0.0, -6.0]);         // amount to translate
```

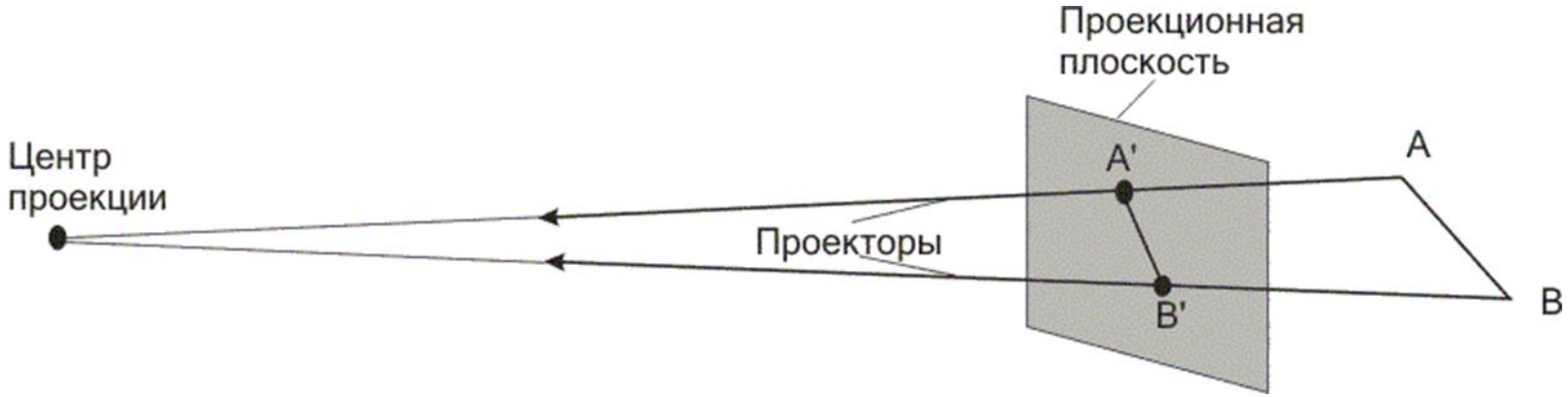
Проецирование



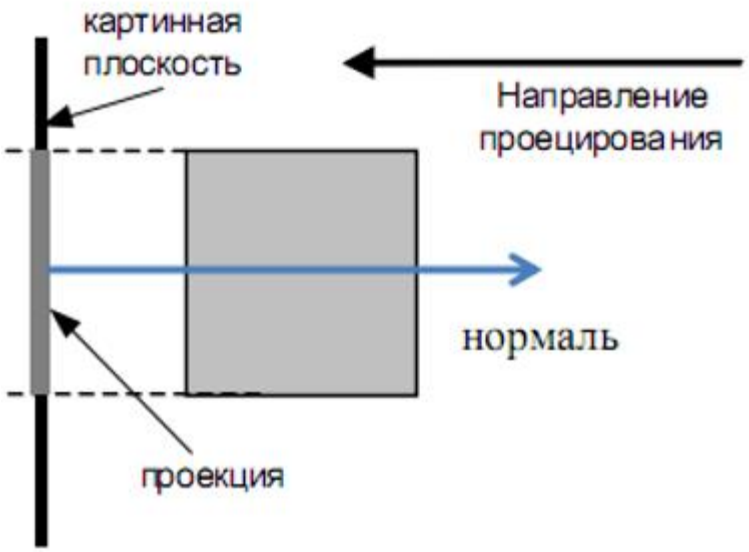
Основные типы проекций

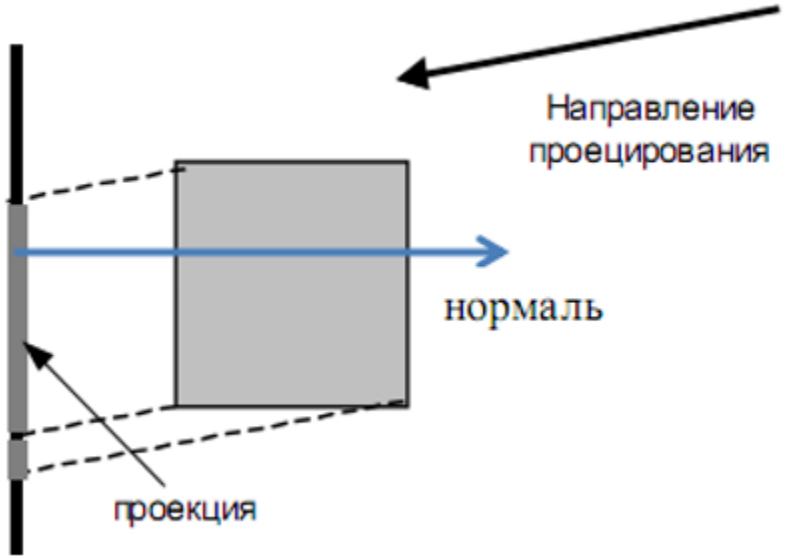
- Параллельная
  - (x,y,z) → (x,y)
  - используется в САПР (CAD), архитектуре и т.п.
  - выглядит нереалистично
- Центральная (перспективная) – так работают камеры
  - (x,y,z) → (x/z,y/z)
  - уменьшение с удалением
  - выглядит реалистично





Классификация параллельных проекций

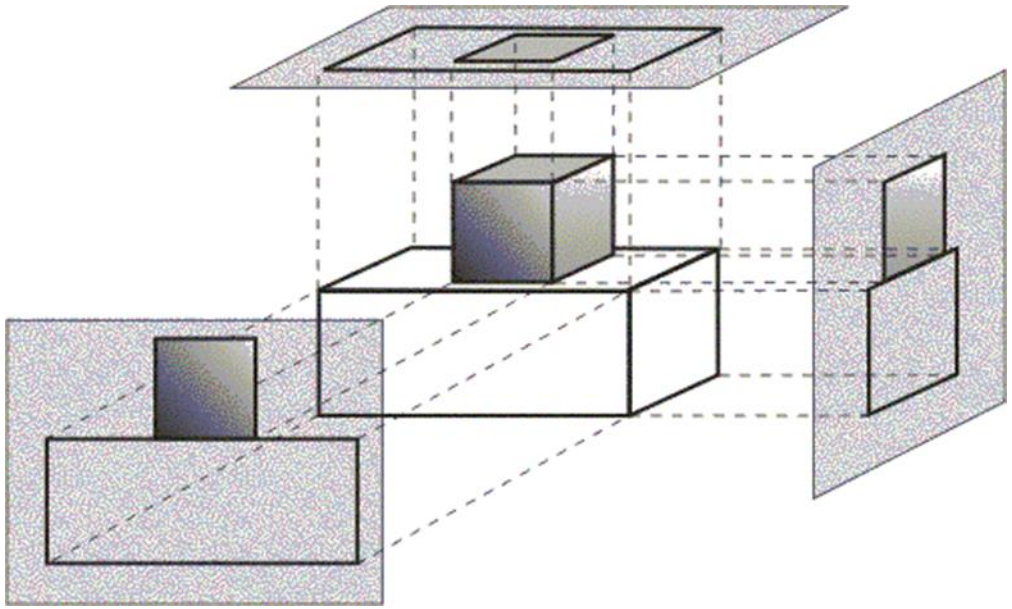




кавалье

кабине

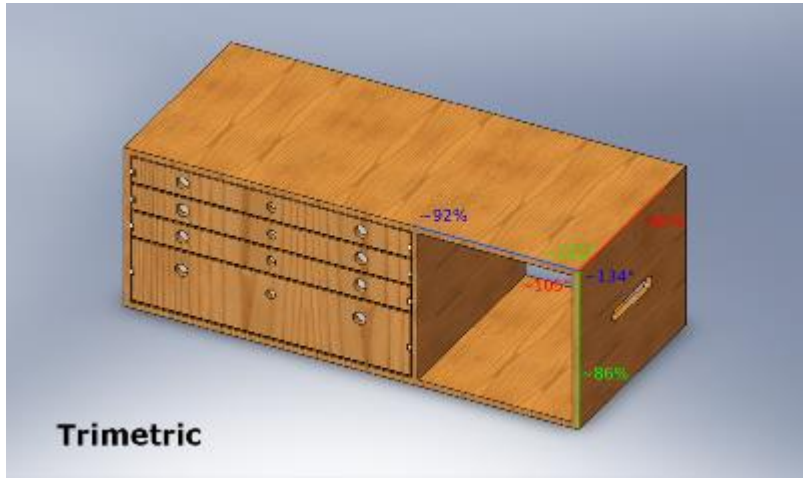
Ортографические проекции

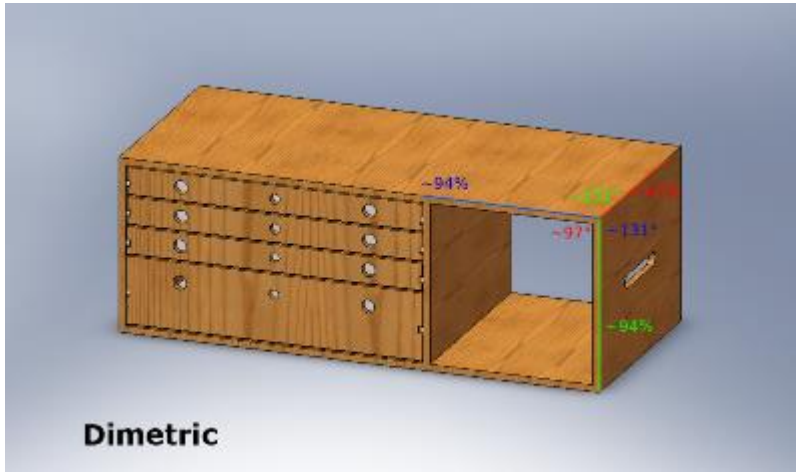


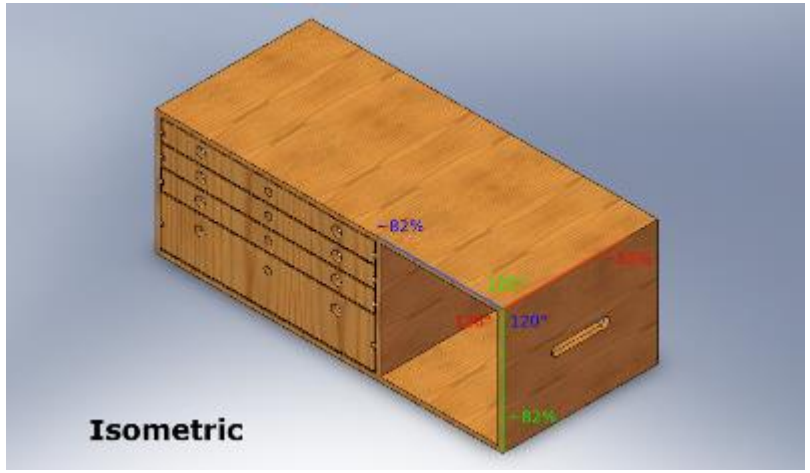
$$[P_x] = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad [P_x] \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ p & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ p & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & q & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & r & 1 \end{bmatrix}$$

Аксонометрические проекции



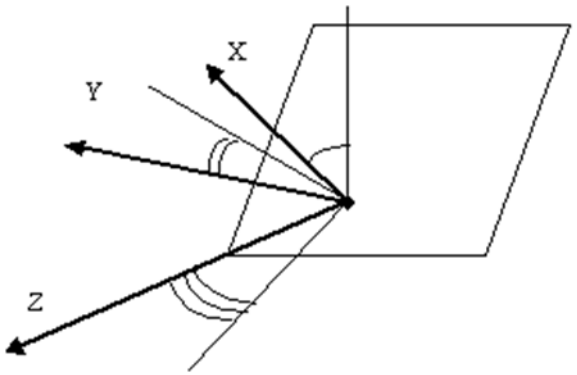


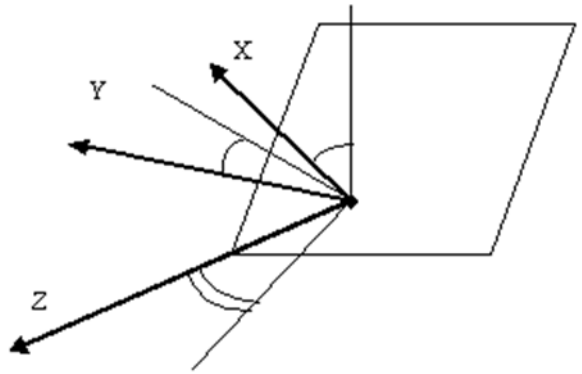


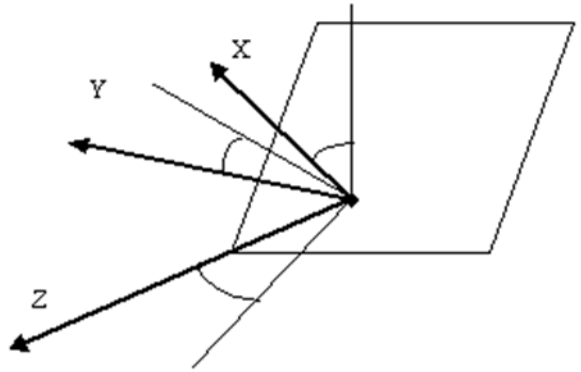
Аксонометрические проекции

$$\begin{bmatrix} \cos\psi & 0 & -\sin\psi & 0 \\ 0 & 1 & 0 & 0 \\ \sin\psi & 0 & \cos\psi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\varphi & \sin\varphi & 0 \\ 0 & -\sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

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

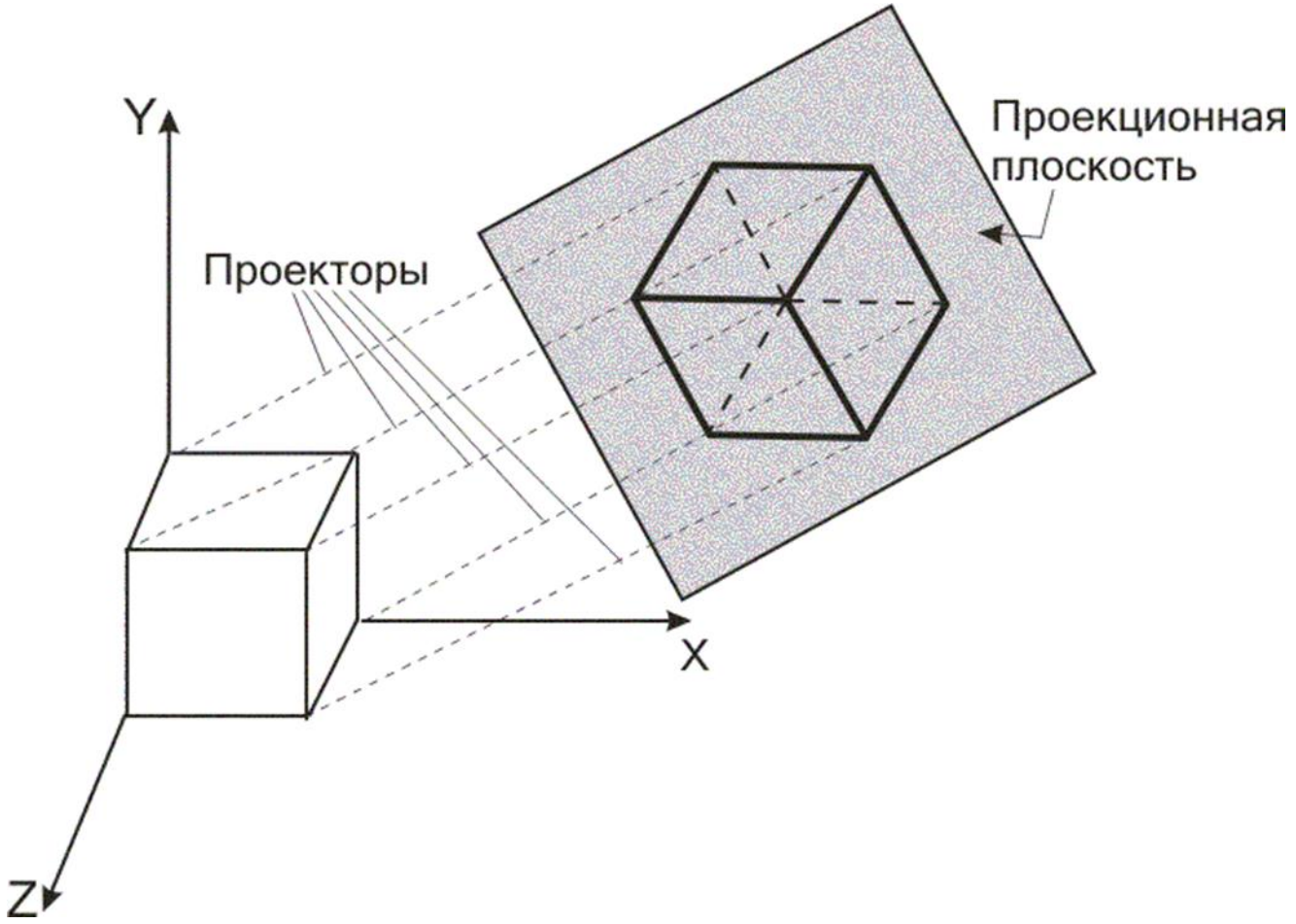




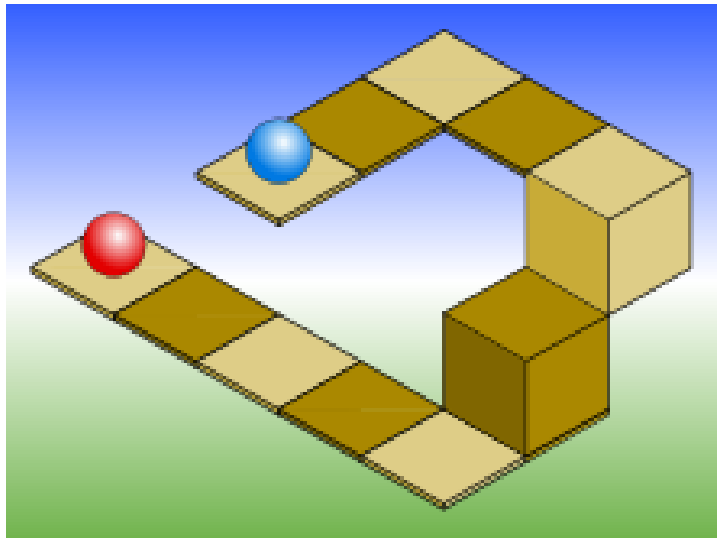


Ограничения аксонометрической проекции

- Как и в других видах параллельных проекций, объекты в аксонометрической проекции не выглядят больше или меньше при приближении или удалении от наблюдателя.
- Это удобно в спрайто-ориентированных компьютерных играх, но, в отличие от перспективной проекции, приводит к ощущению искривления, поскольку человеческий глаз работает иначе.

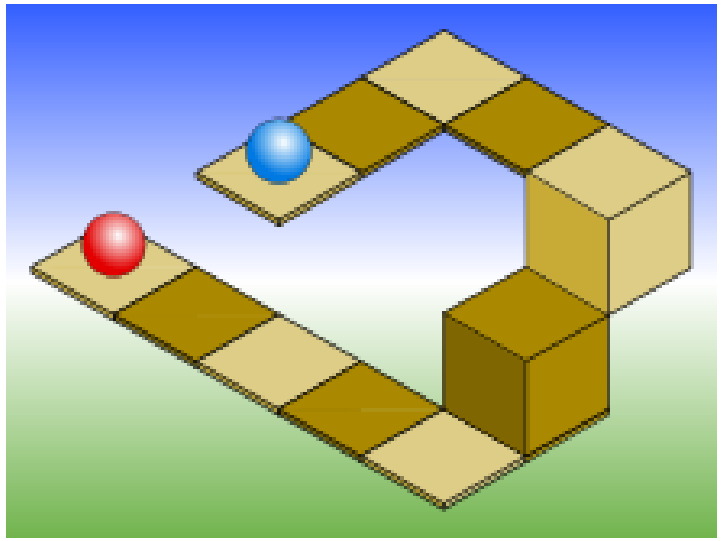


Ограничения аксонометрической проекции



Голубой и красный шары на одном уровне или на разных?

Ограничения аксонометрической проекции



Голубой шар на два уровня выше красного

«Водопад» — литография голландского художника Эшера (октябрь 1961)

Изображён парадокс — падающая вода водопада управляет колесом, которое направляет воду на вершину водопада.

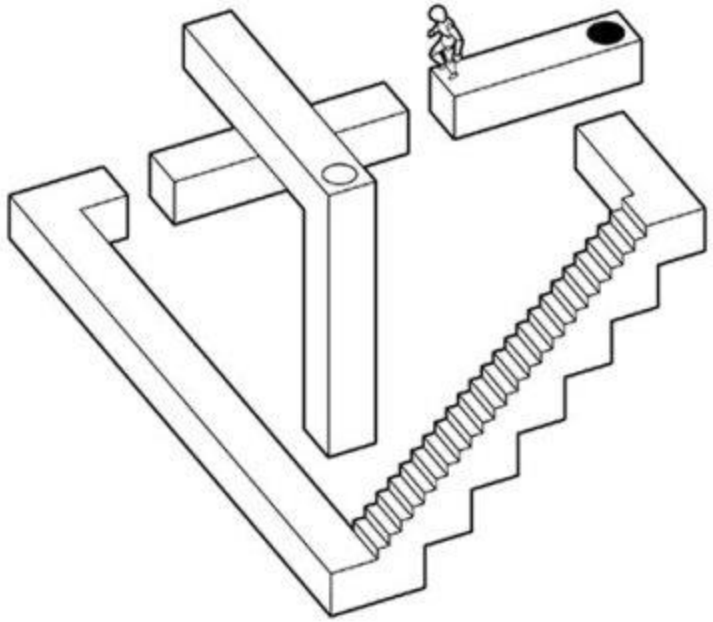
Водопад имеет структуру «невозможного» треугольника Пенроуза: Водопад на литографии работает как вечный двигатель.

Треугольник Пенроуза

Скульптура, кажущийся треугольник, Немецкий технический музей

Та же скульптура при изменении точки просмотра

Кадр из игры «echochrome»



Слоган игры — «В этом мире то, что ты видишь, становится реальностью»

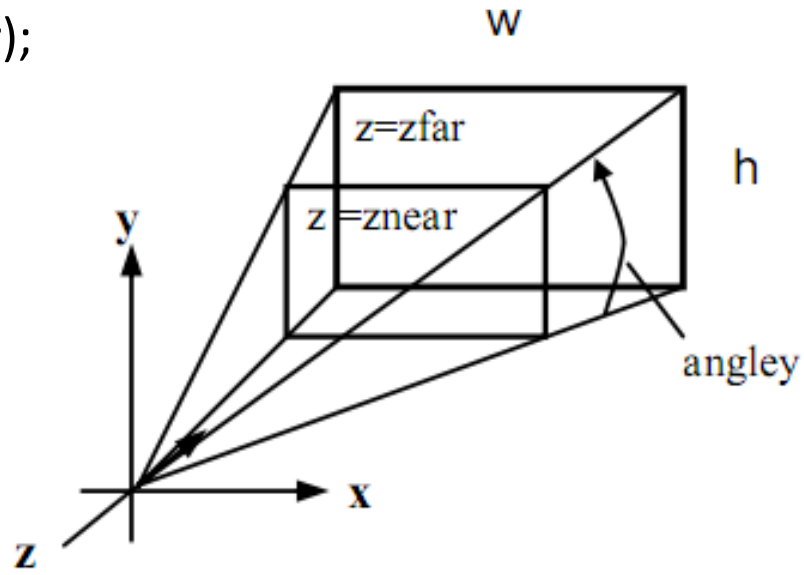
Q*bert (1982)

одна из первых игр с изометрической графикой

аркадные игры начала 1980-х

Матрица проецирования

```
const fieldOfView = 45 * Math.PI / 180;   // in radians
const aspect = gl.canvas.clientWidth / gl.canvas.clientHeight;
const zNear = 0.1;
const zFar = 100.0;
const projectionMatrix = mat4.create();
// the first argument  as the destination to receive the result.
mat4.perspective(projectionMatrix, fieldOfView, aspect,  zNear,  zFar);
```



Установка камеры

Можно менять не свойства объекта, а свойства точки обзора.

Для установки камеры в библиотеке glMatrix используется функция

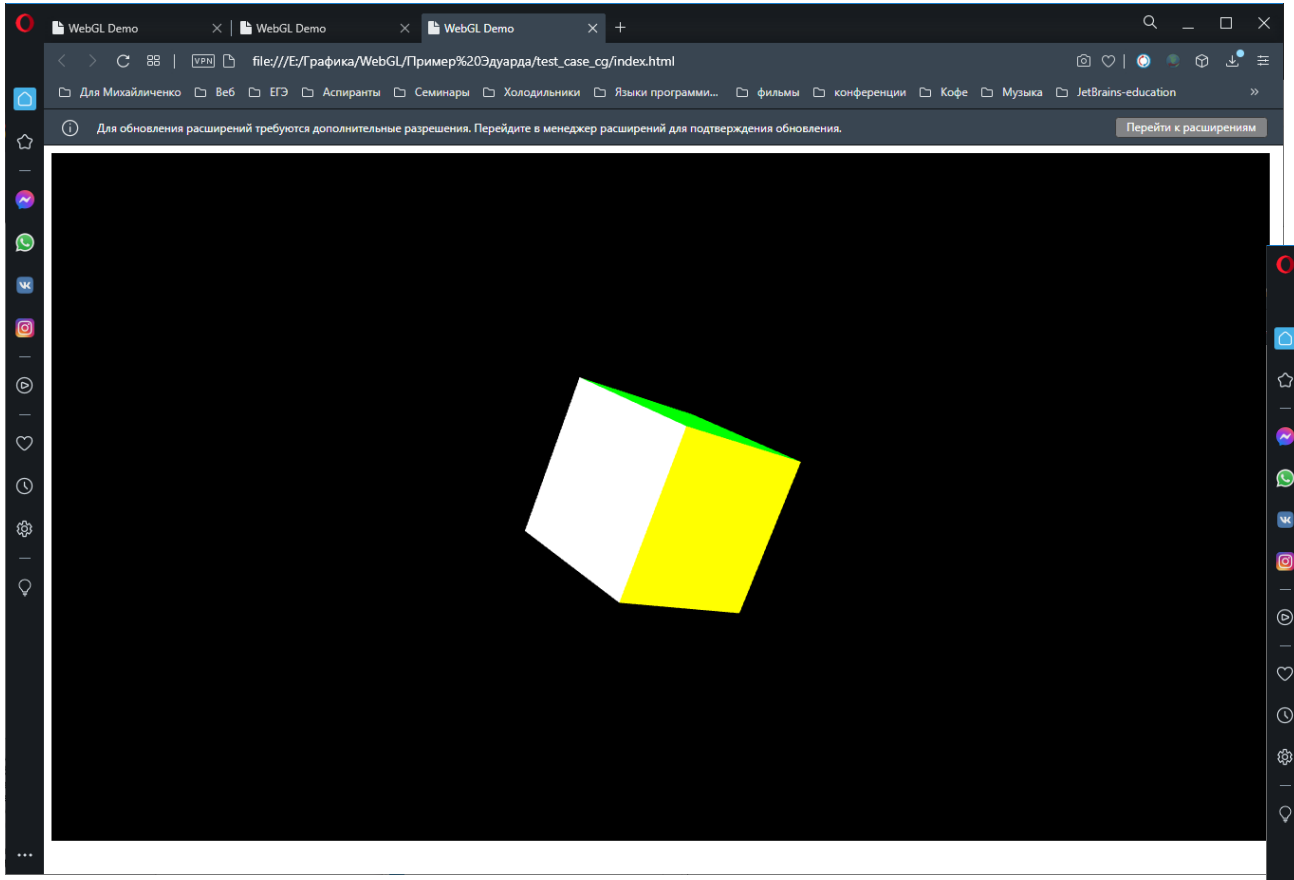
mat4.lookAt(matrix, eye, center, up)

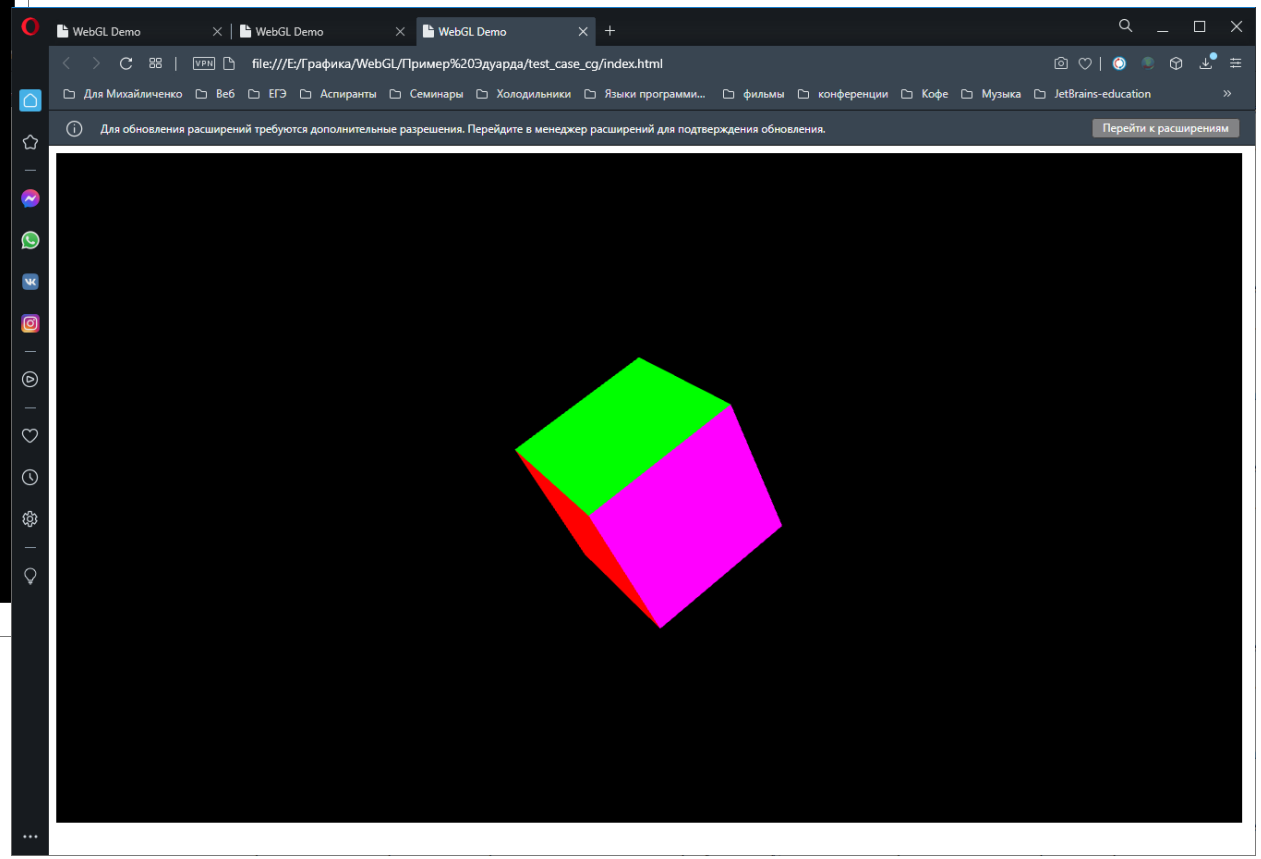
- matrix: матрица модели, которая настраивается в зависимости от свойств камеры
- eye: позиция камеры
- center: точка, на которую направлена камера
- up: вектор вертикальной ориентации

Пример

```
function setupWebGL()
{
   gl.clearColor(0.0, 0.0, 0.0, 1.0);
   gl.clear(gl.COLOR_BUFFER_BIT);

   gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);
   mat4.perspective(pMatrix, Math.PI/2, gl.viewportWidth / gl.viewportHeight, 0.1, 100.0);
   mat4.identity(mvMatrix);
   mat4.lookAt(mvMatrix, [2, 0,-2], [0,0,0], [0,1,0]);
}
```

Что хотим получить?





Шейдеры

```
const vsSource = `
   attribute vec4 aVertexPosition;
   attribute vec4 aVertexColor;
   uniform mat4 uModelViewMatrix;
   uniform mat4 uProjectionMatrix;
   out lowp vec4 vColor;

  void main(void) {
    gl_Position = uProjectionMatrix * uModelViewMatrix * aVertexPosition;
    vColor = aVertexColor;
   }
`;

const fsSource = `
   in lowp vec4 vColor;

   void main(void) {
      gl_FragColor = vColor;
   }
`;
```

Передача атрибутов и uniform-переменных

```
const programInfo = {
    program: shaderProgram,
    attribLocations: {
        vertexPosition: gl.getAttribLocation(shaderProgram, 'aVertexPosition’),
        vertexColor: gl.getAttribLocation(shaderProgram, 'aVertexColor’),
    },
    uniformLocations: {
        projectionMatrix: gl.getUniformLocation(shaderProgram, 'uProjectionMatrix’),
        modelViewMatrix: gl.getUniformLocation(shaderProgram, 'uModelViewMatrix’),
    }
};
```

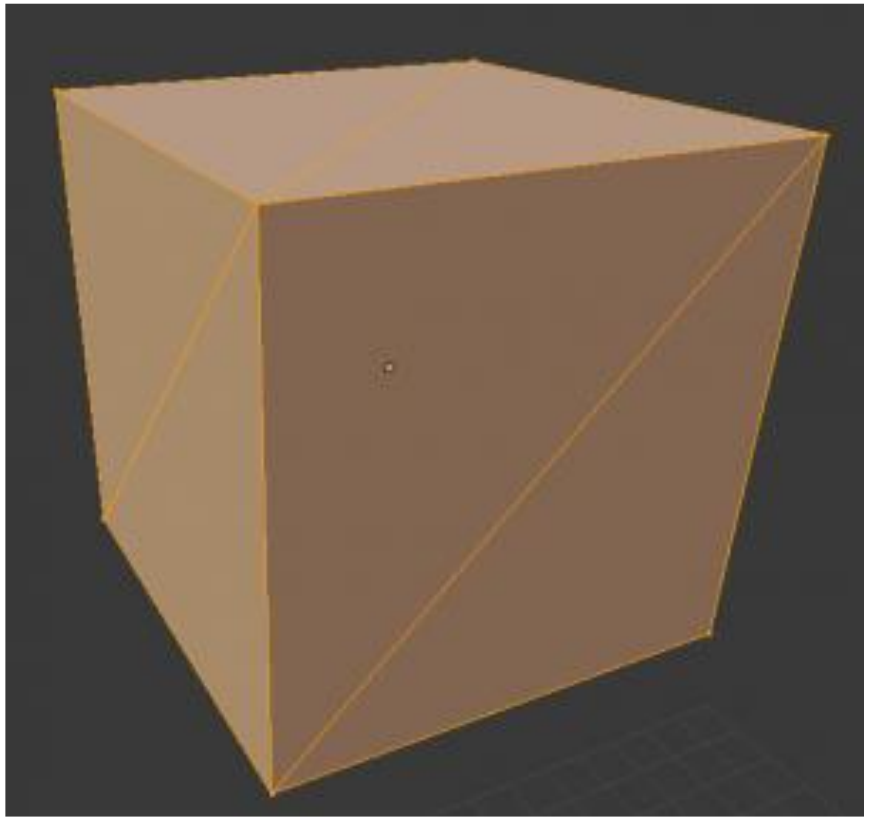
Анимация

```
const buffers = initBuffers(gl);
var then = 0;

function render(now) {
      now *= 0.001;  // convert to seconds
      const deltaTime = now - then;
      then = now;
      drawScene(gl, programInfo, buffers, deltaTime);
     requestAnimationFrame(render);
   }
```

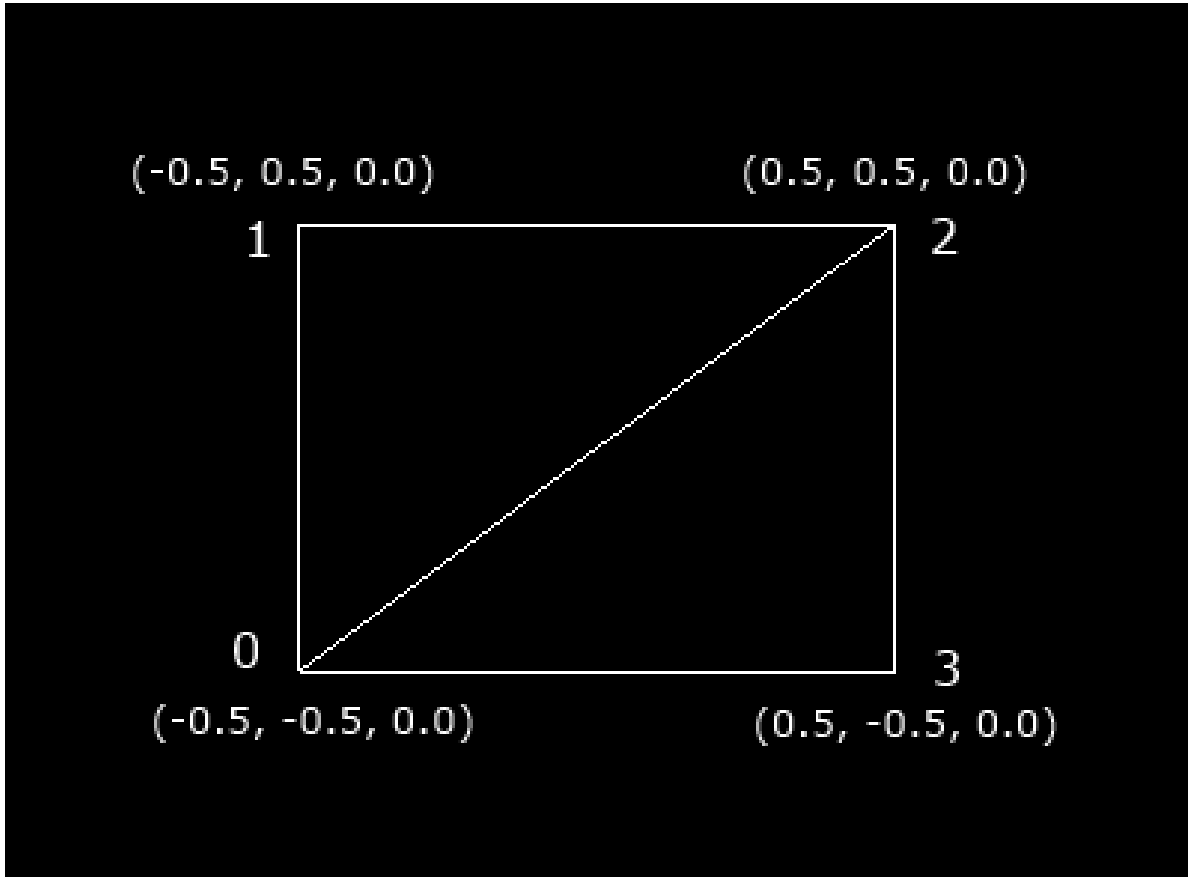
Координаты куба

```
function initBuffers(gl) {
  const positionBuffer = gl.createBuffer();
  gl.bindBuffer(gl.ARRAY_BUFFER, positionBuffer);

const positions = [
    // Front face
    -1.0, -1.0,  1.0,
    1.0, -1.0,  1.0,
    1.0,  1.0,  1.0,
    -1.0,  1.0,  1.0,
    // Back face
    -1.0, -1.0, -1.0,
    -1.0,  1.0, -1.0,
    1.0,  1.0, -1.0,
    1.0, -1.0, -1.0,
    // Top face
    -1.0,  1.0, -1.0,
    -1.0,  1.0,  1.0,
    1.0,  1.0,  1.0,
    1.0,  1.0, -1.0,
    // Bottom face
    -1.0, -1.0, -1.0,
    1.0, -1.0, -1.0,
    1.0, -1.0,  1.0,
    -1.0, -1.0,  1.0,
    // Right face
    1.0, -1.0, -1.0,
    1.0,  1.0, -1.0,
    1.0,  1.0,  1.0,
    1.0, -1.0,  1.0,
    // Left face
    -1.0, -1.0, -1.0,
    -1.0, -1.0,  1.0,
    -1.0,  1.0,  1.0,
    -1.0,  1.0, -1.0,
  ];

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

Сколько вершин и сколько примитивов?



Использование буфера индексов



Цвет граней куба

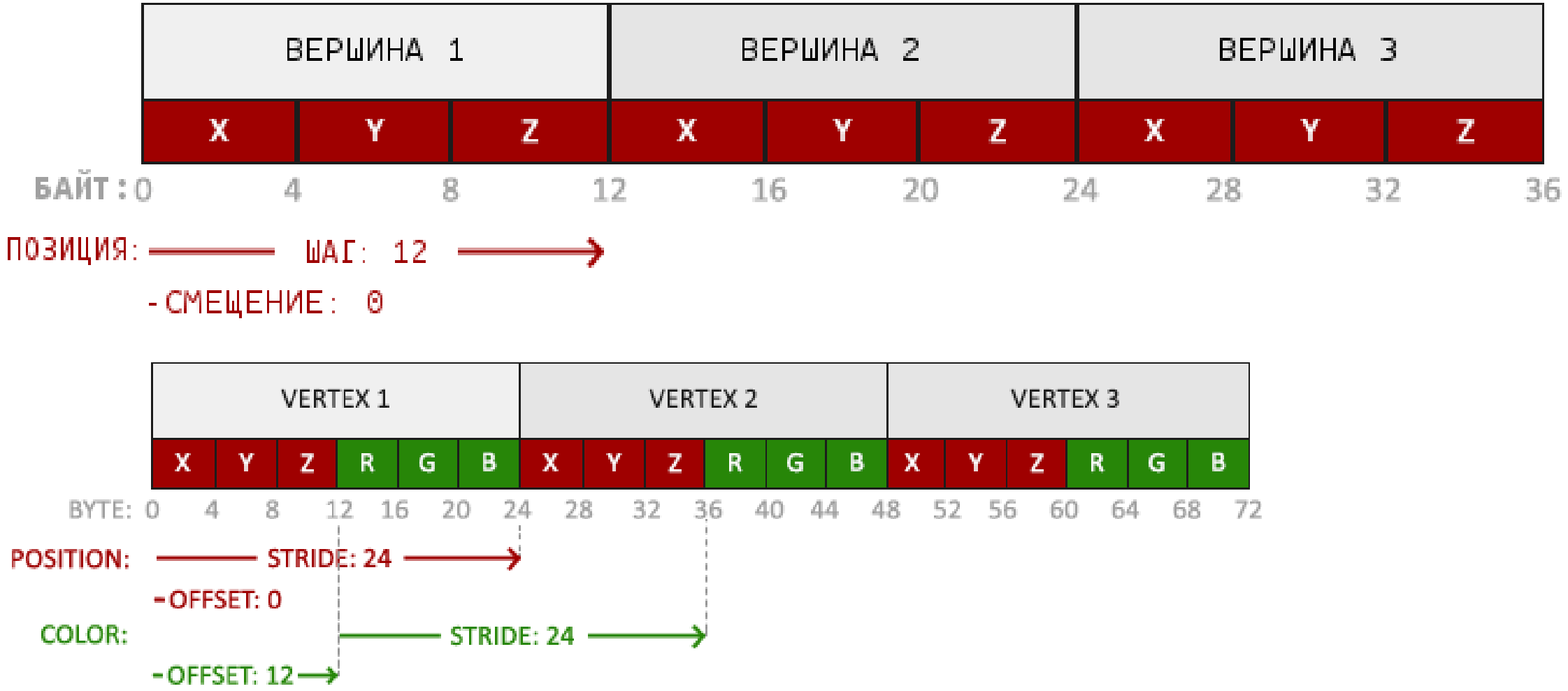
```
const faceColors = [
      [1.0,  1.0,  1.0,  1.0],    // Front face: white
      [1.0,  0.0,  0.0,  1.0],    // Back face: red
      [0.0,  1.0,  0.0,  1.0],    // Top face: green
      [0.0,  0.0,  1.0,  1.0],    // Bottom face: blue
      [1.0,  1.0,  0.0,  1.0],    // Right face: yellow
      [1.0,  0.0,  1.0,  1.0],    // Left face: purple
   ];
```

```
var colors = [];
for (var j = 0; j < faceColors.length; ++j) {
    const c = faceColors[j];
    colors = colors.concat(c, c, c, c);
 }
 const colorBuffer = gl.createBuffer();

 gl.bindBuffer(gl.ARRAY_BUFFER, colorBuffer);
 gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(colors), gl.STATIC_DRAW);
```

Индексный буфер

```
const indexBuffer = gl.createBuffer();
gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, indexBuffer);
const indices = [
      0,  1,  2,     0,  2,  3,    // front
      4,  5,  6,     4,  6,  7,    // back
      8,  9,  10,    8,  10, 11,   // top
      12, 13, 14,    12, 14, 15,   // bottom
      16, 17, 18,    16, 18, 19,   // right
      20, 21, 22,    20, 22, 23,   // left
   ];
gl.bufferData(gl.ELEMENT_ARRAY_BUFFER, new Uint16Array(indices), gl.STATIC_DRAW);

  return {
      position: positionBuffer,
      color: colorBuffer,
      indices: indexBuffer,
    };
 } //initBuffers
```

Структура буфера



```
function drawScene(gl, programInfo, buffers, deltaTime) {
  gl.clearColor(0.0, 0.0, 0.0, 1.0);
  gl.clearDepth(1.0);
  gl.enable(gl.DEPTH_TEST);
  gl.depthFunc(gl.LEQUAL);
  gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

  const numComponents = 3;
  const type = gl.FLOAT;
  const normalize = false;
  const stride = 0;
  const offset = 0;
  gl.bindBuffer(gl.ARRAY_BUFFER, buffers.position);

  gl.vertexAttribPointer(
      programInfo.attribLocations.vertexPosition,
      numComponents,
      type,     normalize,     stride,     offset);

  gl.enableVertexAttribArray(programInfo.attribLocations.vertexPosition);
```

Кроме установки указателя нам еще надо включить атрибут с помощью метода gl.enableVertexAttribArray(),
в который передается ранее установленный атрибут:

```
gl.enableVertexAttribArray(shaderProgram.vertexPositionAttribute);
```

После этого мы сможем передать каждую вершину в вершинный шейдер через переменную attribute vec3 aVertexPosition.

Передача uniform-параметров

```
gl.uniformMatrix4fv(
      programInfo.uniformLocations.projectionMatrix,
      false,
      projectionMatrix);

gl.uniformMatrix4fv(
     programInfo.uniformLocations.modelViewMatrix,
      false,
      modelViewMatrix);
```

Вывод или отрисовка

Для отрисовки фигур в WebGL используется следующие методы:

gl.drawArrays()

gl.drawElements()

```
const vertexCount = 36;
const type = gl.UNSIGNED_SHORT;
const offset = 0;
gl.drawElements(gl.TRIANGLES, vertexCount, type, offset);
```

Метод gl.drawElements()

Метод gl.drawElements() работает с буфером индексов. Он имеет следующую сигнатуру:
gl.drawElements(mode, count, type, offset):

mode: режим, указывающий на тип примитива. В качестве примитивов используются те же, что и для метода gl.drawArrays()

count: число элементов для отрисовки

type: тип значений в буфере индексов. Может иметь значение UNSIGNED_BYTE или UNSIGNED_SHORT

offset: смещение - с какого индекса будет проводиться отрисовка

Например,
gl.**drawElements**(gl.TRIANGLES, vertexCount, type, offset);

Примитивные типы GLSL

void: функция не возвращает никакого значения

bool: логические значения true или false

int: целочисленные значения

float: числовые значения с плавающей точкой

Примитивные типы GLSL

vec2, vec3, vec4: двух-, трех- и четырехмерные векторы соответственно, которые содержат объекты типа float

ivec2, ivec3, ivec4: двух-, трех- и четырехмерные векторы соответственно, которые содержат объекты типа int

bvec2, bvec3, bvec4: двух-, трех- и четырехмерные векторы соответственно, которые содержат объекты типа bool

mat2, mat3, mat4: матрицы размера 2x2, 3x3 и 4x4 соответственно, которые содержат объекты типа float

sampler2D, samplerCube: специальные типы - семплеры для работы с текстурами. С помощью сэмплеров во фрагментном шейдере мы можем получить цветовые значения текстур и передать их примитиву

Структуры

```
struct someStruct{
    int someInt;
    vec4 someVec;
}
```

Квалификаторы (модификаторы)

- attribute: атрибут или часть описания вершины, которое передается из программы на WebGL в вершинный шейдер

- const: константы, эти переменные определяют свое значение только один раз и в процессе программы его уже не меняют

- uniform: по сути то же переменные с константными значениями, только эти значения задаются для всего примитива

- varying: переменная, которая задается в вершинном шейдере и затем передается во фрагментный шейдер, где может быть использована – устаревшая
- out/in с одинаковым именем в обоих шейдерах

Квалификаторы для чисел с плавающей точкой

highp: число с плавающей точкой сохраняет максимальную точность

mediump: число со средней степенью точности

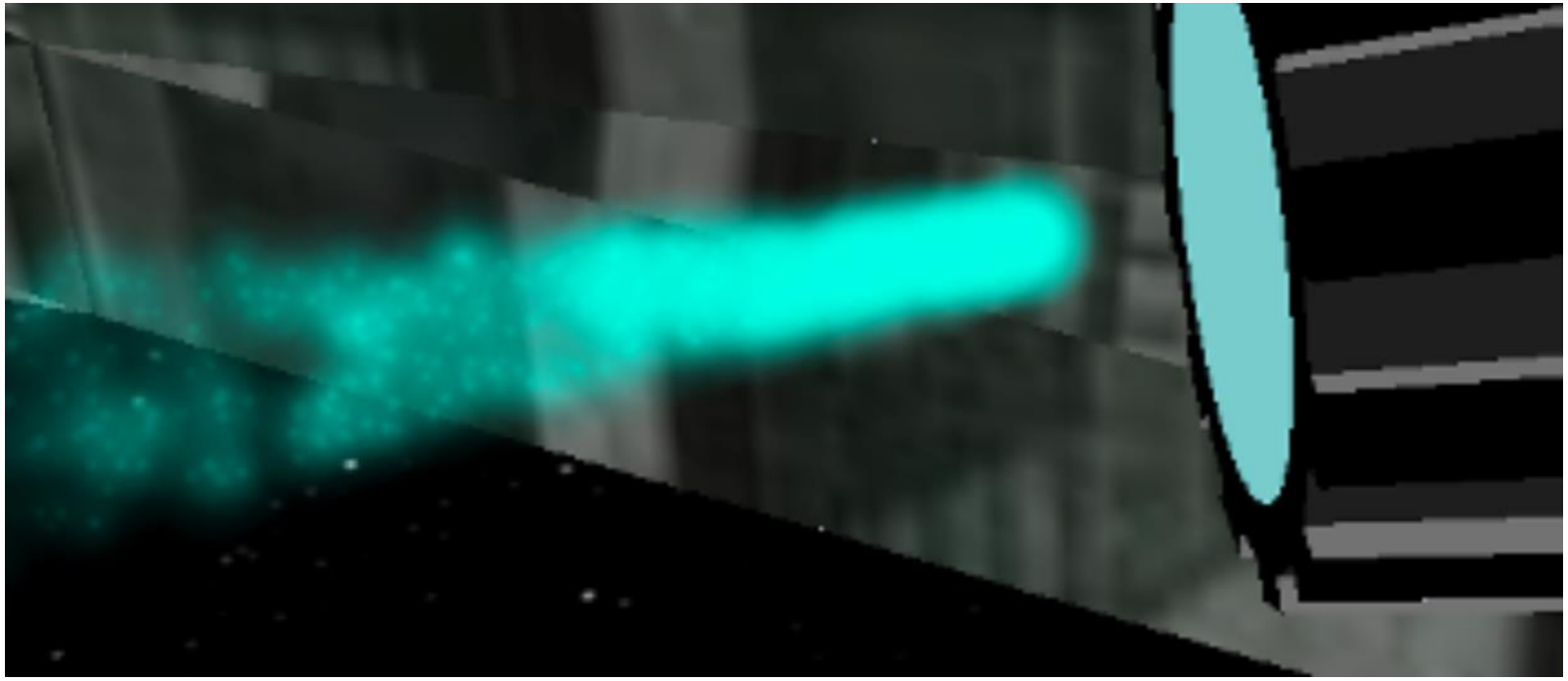
lowp: диапазон плавающей запятой от -2 до 2

Например,
varying highp vec4 vColor;

Встроенные глобальные переменные GLSL

gl_Position: переменная имеет тип vec4 и указывает на положение вершины. Используется в **вершинном** шейдере в качестве в**ыходного** параметра

gl_PointSize: имеет тип float и содержит размер точки. Используется в **вершинном** шейдере в качестве **выходного** параметра



Встроенные глобальные переменные GLSL

gl_FragCoord: имеет тип vec4 и указывает на положение фрагмента в буфере фреймов. Используется во **фрагментном** шейдере в качестве **входного** параметра

gl_FontFacing: имеет тип bool и определяет, принадлежит ли фрагмент лицевому примитиву. Используется во **фрагментном** шейдере в качестве **входного** параметра

gl_PointCoord: имеет тип vec2 и указывает на позицию фрагмента. Используется во **фрагментном** шейдере в качестве **входного** параметра

gl_FragColor: имеет тип vec4 и указывает на итоговый цвет фрагмента. Используется во **фрагментном** шейдере в качестве **выходного** параметра

gl_FragData[n]: имеет тип vec4 и указывает на цвет фрагмента для прикрепления цвета n. Используется во **фрагментном** шейдере в качестве **выходного** параметра

Встроенные функции

abs(x);
round(x)
mod(x);
fmod(x)
sqrt(x);
pow(x, y)
max(x, y);
min(x, y)
sin(angle);
cos(angle);
tan(angle)
log(x)
dot(x, y) скалярное произведение векторов
cross(x, y) векторное произведение векторов
matrixCompMult(mat x, mat y) произведение матриц одной размерности
normalize(x) нормализация вектора
reflect(t, n) отражает вектор t вдоль вектора n
vec4 texture2D (sampler2D sampler,vec2 coord) выборка текстур

Создаем свою функцию

```
// матрица для двумерного вращения
mat2 rot ( in float a ) {
return mat2 ( cos (a), sin (a), -sin (a), cos (a) ) ;
}

// Используем
vec2 pos1 = rot (17.0) * pos ;
```

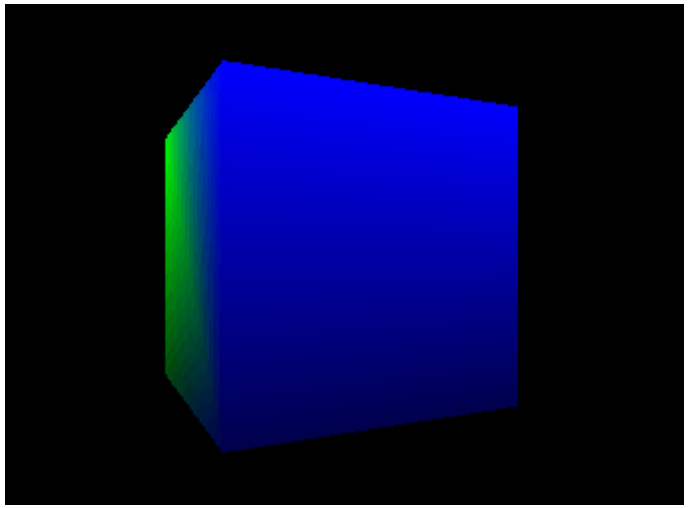
Операторы управления

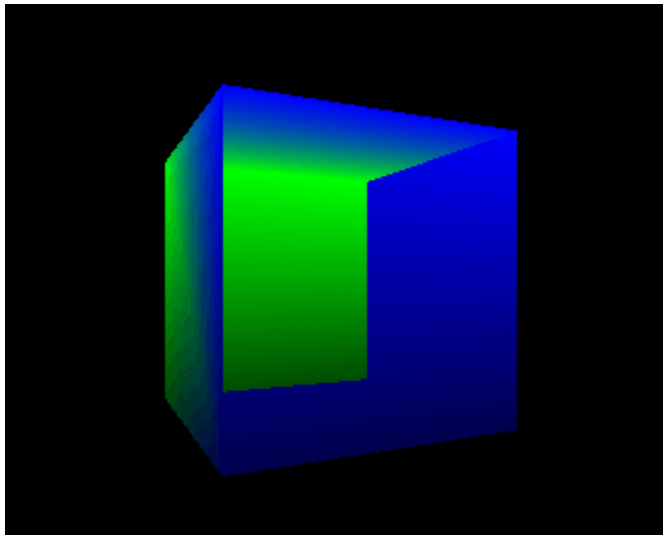
if
switch
discard //специфическое ветвление для пиксельного шейдера
break
continue
do
for
while

Старайтесь использовать встроенные функции

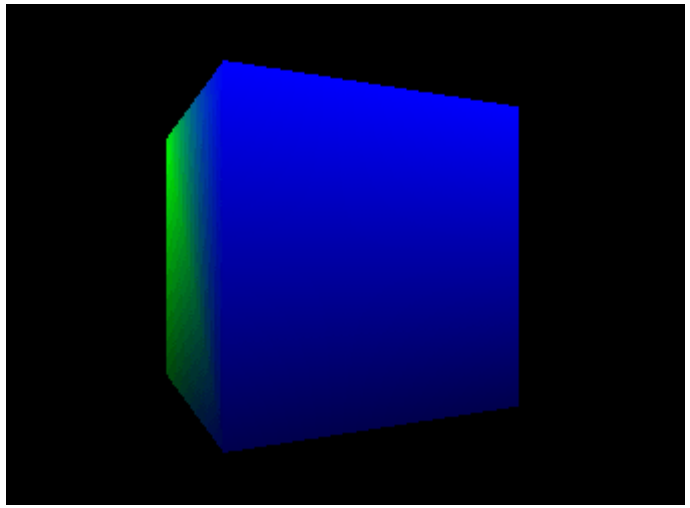
```
float f1 , f2 , f3 , f4 ; . . .
float c = f1+f2+f3+f4 ; // медленно
float a = dot (vec4( f1, f2, f3, f4 ), vec4(1.0)) ; // быстро
```

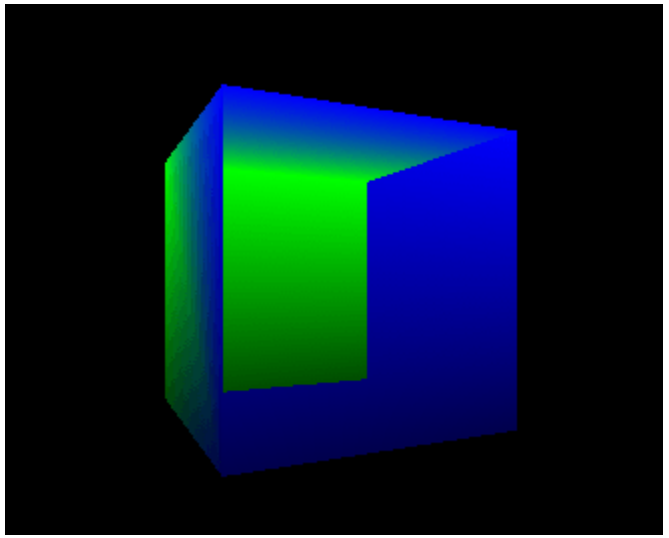
Некоторые ошибки – в чём проблема?





Некоторые ошибки – в чём проблема?





gl.enable(gl.DEPTH_TEST); //если забыли
gl.drawElements(gl.TRIANGLES, indexBuffer.numberOfItems, gl.UNSIGNED_SHORT,0);