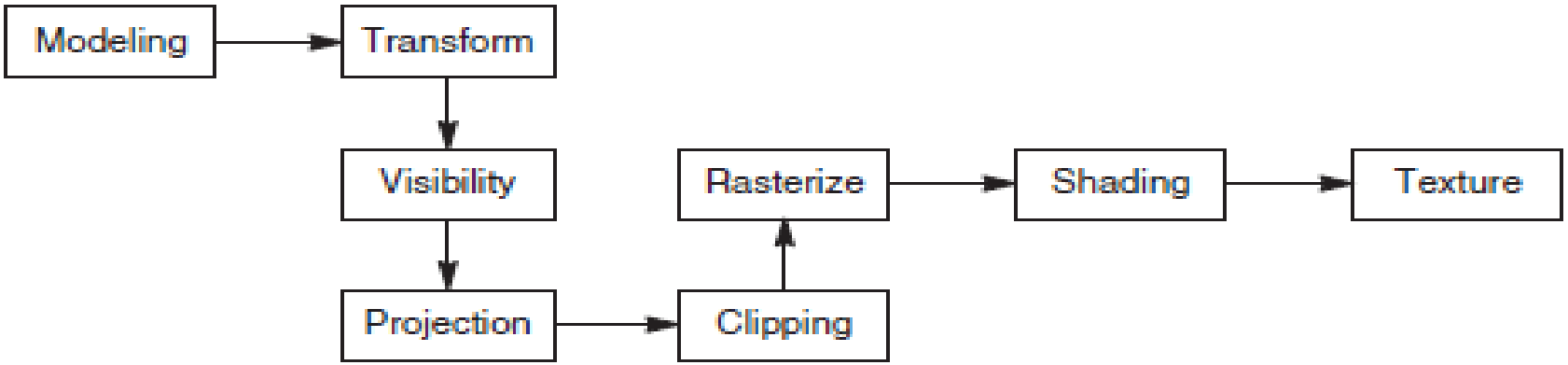
Компьютерная графика

Курс лекций

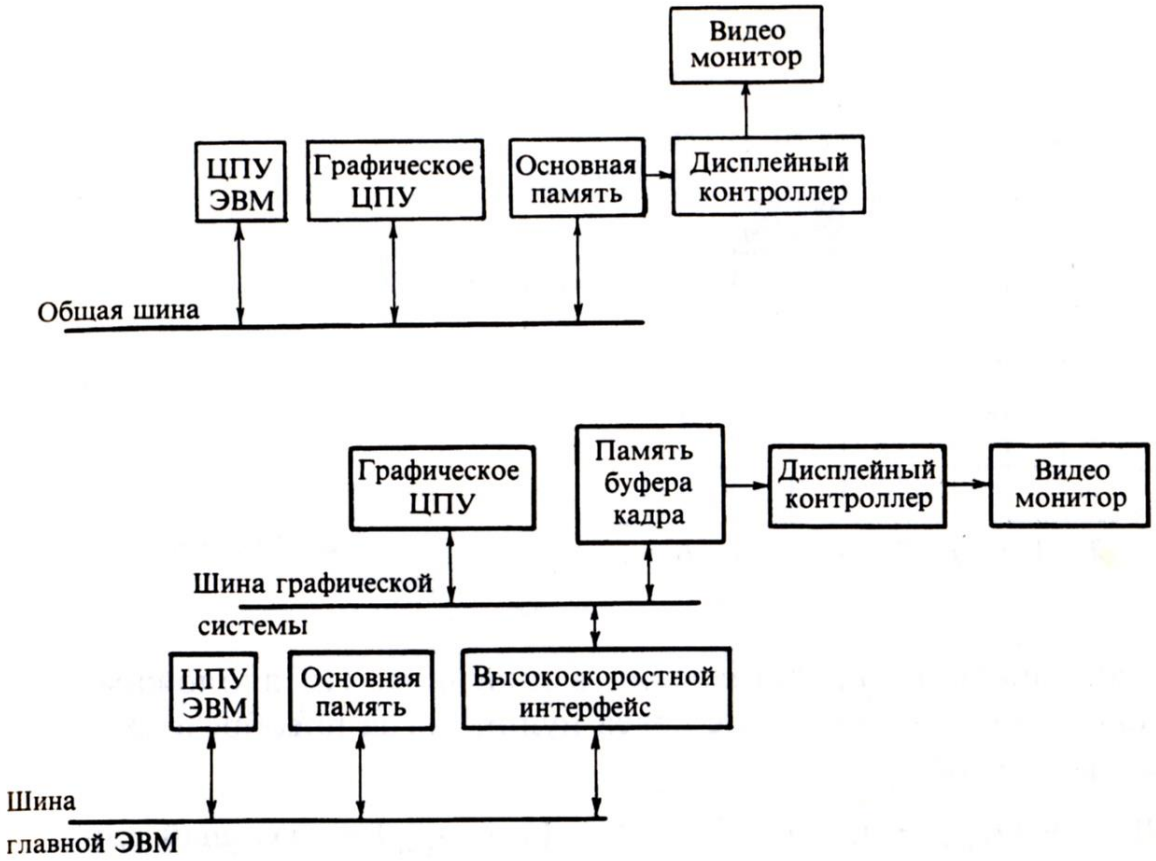
Тема №11. Использование возможностей графических процессоров в приложениях.

Графический конвейер

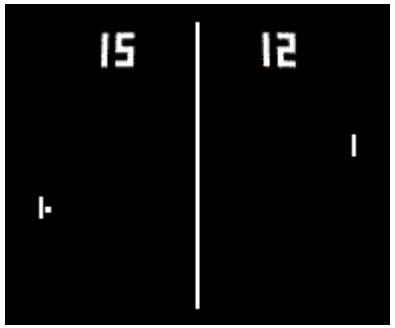


- моделирование (modeling) – математическое описание объектов, всей сцены, источников света, с учётом расположения
- отображение (rendering):
  - преобразования (transformation) – задание местоположения;
  - определение видимости (visibility) – область видимости (field of view) + нелицевые поверхности → отсечение (clipping);
  - проекция на картинную плоскость (projection);
  - растеризация (rasterization);
  - закраска (shading);
  - текстурирование (texturing).

Архитектура растровых графических систем с буфером кадра



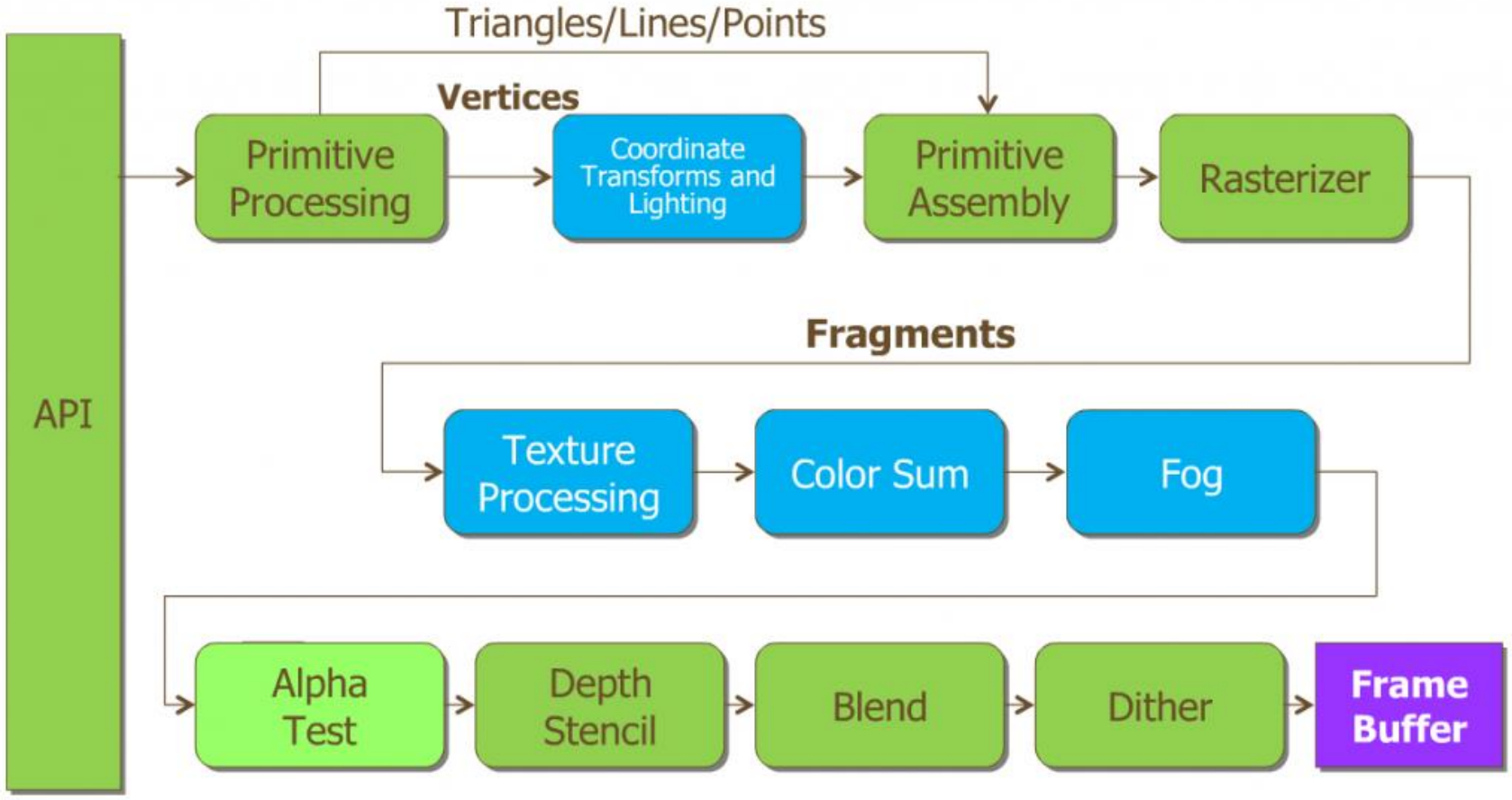
- ≈1960-е
- 1970г. 1K RAM
- 1974 z-буфер, Кэтмулл (Catmull)
- 1984 альфа-канал, Loren Carpenter
- 1982 VLSI-процессор (Geometry Engine, Silicon Graphics) → IRIS (Integrated Raster Imaging System)
- GPU(Geometry/Graphics Processing Unit): *n*VIDIA GeForce 256



Использование возможностей современных GPU

- обеспечение гибкого использования возможностей графического конвейера;
  - OpenGL 3.2/4.0, OpenGLSL
- использование вычислительной мощности графического процессора для выполнения вычислений:
  - CUDA (NVidia)
  - OpenCL (Khronos Group)

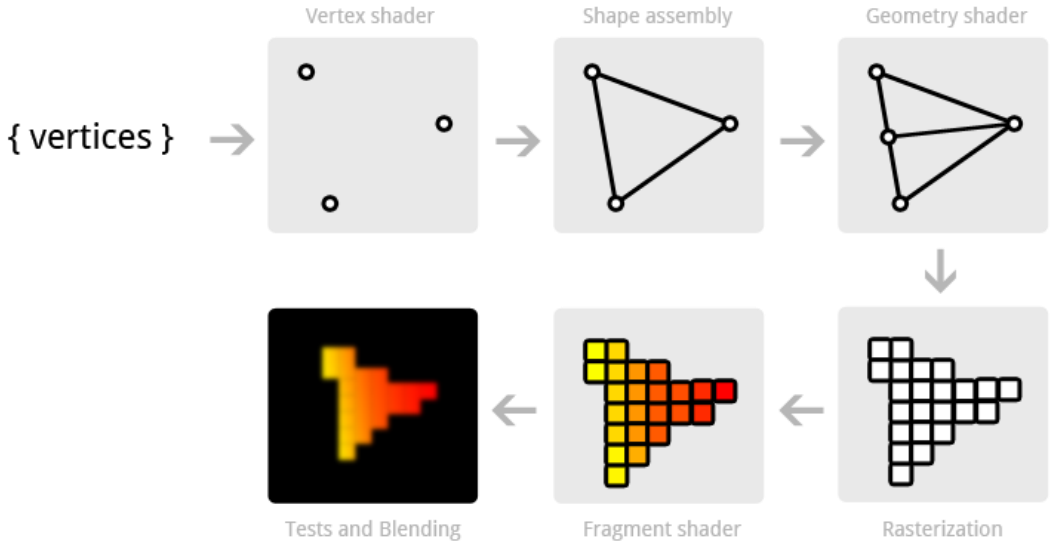
Объекты, обрабатываемые конвейером OpenGL



вершины (vertex) → примитивы (primitive) → фрагменты (fragment)
→ пиксели (pixel)
* fixed function pipeline

Шейдеры

- шейдер – программа, написанная на языке OpenGL Shader Language и выполняющаяся на GPU
  - вершинные шейдеры (vertex shader) – определение положения, цвета, текстурных координат;
  - геометрические шейдеры (geometry shader) – обработка примитивов;
  - фрагментные/пиксельные шейдеры (fragment/pixel shader) – управление растеризацией;
  - тесселяционные шейдеры (OpenGL 4+): (tessellation control shaders = hull shaders, tessellation evaluation shaders = domain shaders) – разбиение граней.



Создание окна и контекста OpenGL: библиотека GLFW

- контекст OpenGL: набор состояний, управляющих формированием изображения

```
glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 2);
glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);

glfwWindowHint(GLFW_RESIZABLE, GL_FALSE);
GLFWwindow* window = glfwCreateWindow(800, 600, "OpenGL", nullptr, nullptr);

attrib = glfwGetWindowAttrib(window, GLFW_CONTEXT_VERSION_MAJOR);
attrib = glfwGetWindowAttrib(window, GLFW_CONTEXT_VERSION_MINOR);
attrib = glfwGetWindowAttrib(window, GLFW_OPENGL_PROFILE);
```

Получение указателей на функции OpenGL: библиотека GLEW

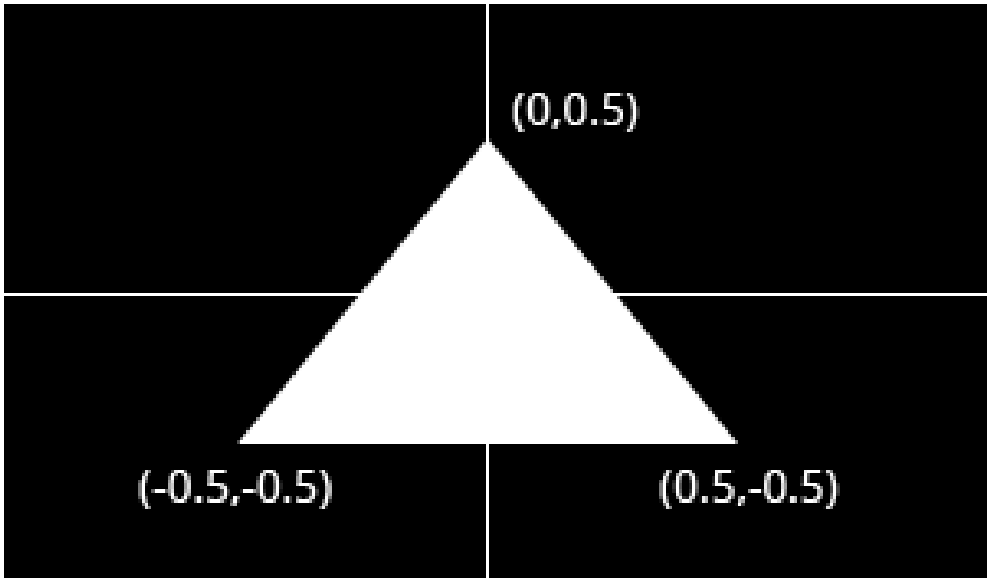
- линковка проекта с библиотекой glew32s.lib
- подключение заголовочного файла до подключения заголовочного файла OpenGL и других библиотек, создающих контекст OpenGL

```
#define GLEW_STATIC
#include <GL/glew.h>
```

- вызов glewInit() после создания окна и контекста OpenGL

```
// Specify prototype of function typedef void
    (*GENBUFFERS) (GLsizei, GLuint*);
// Load address of function and assign it to a function pointer
    GENBUFFERS glGenBuffers =
    (GENBUFFERS)wglGetProcAddress("glGenBuffers");
```

Пример 1



- «усеченные» координаты – в диапазоне [-1;1]

```
float vertices[] = {
        0.0f, 0.5f, // Vertex 1 (X, Y)
        0.5f, -0.5f, // Vertex 2 (X, Y)
        -0.5f, -0.5f // Vertex 3 (X, Y)
};
```

Пример 1: создание буфера вершин

```
GLuint vbo;
glGenBuffers(1, &vbo); // Generate 1 buffer
glBindBuffer(GL_ARRAY_BUFFER, vbo);
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices),
                    vertices, GL_STATIC_DRAW);
```

- glGenBuffers() : получение идентификатора буфера
- glBindBuffer() : выбор активного буфера (вершин)
- glBufferData() : передача данных в буфер
  - GL_STATIC_DRAW : статичные данные для множества отрисовок;
  - GL_DYNAMIC_DRAW : редко изменяющиеся данные для множества отрисовок;
  - GL_STREAM_DRAW : постоянно изменяющиеся данные (при каждой отрисовке);

Пример 1: вершинный шейдер

```
#version 150

in vec2 position;
void main()
{
        gl_Position = vec4(position, 0.0, 1.0);
}
```

- встроенные типы vec* и mat* для работы с векторами и матрицами с элементами типа float;
- вызывается для каждой обрабатываемой вершины;
- position – входной параметр (содержит обрабатываемую вершину при каждом вызове);
- gl_Position – специальная переменная, которая должна содержать координаты вершины после вызова шейдера.

Пример 1: фрагментный шейдер

```
#version 150

out vec4 outColor;
void main()
{
        outColor = vec4(1.0, 1.0, 1.0, 1.0);
}
```

- вызывается для каждого пиксела, полученного после растеризации сформированного примитива;
- outColor – цвет пиксела (RGBA с компонентами типа float и значениями в интервале [0.0; 1.0]).

Пример 1: загрузка и компиляция шейдеров

```
GLuint vertexShader;
vertexShader = glCreateShader(GL_VERTEX_SHADER);
glShaderSource(vertexShader, 1, &vertexSource, NULL);

glCompileShader(vertexShader);

GLint status;
glGetShaderiv(vertexShader, GL_COMPILE_STATUS, &status);
char buffer[512];
glGetShaderInfoLog(vertexShader, 512, NULL, buffer);
```

Пример 1: сборка программы

```
GLuint shaderProgram = glCreateProgram();
glAttachShader(shaderProgram, vertexShader);
glAttachShader(shaderProgram, fragmentShader);

glBindFragDataLocation(shaderProgram, 0, "outColor");

glLinkProgram(shaderProgram);

glUseProgram(shaderProgram);

glGetProgramiv(p, GL_LINK_STATUS, &status);

glGetProgramiv(p, GL_INFO_LOG_LENGTH, &infoLogLength);
char* strInfoLog = new char[infoLogLength + 1];
glGetProgramInfoLog(p, infoLogLength, NULL, strInfoLog);
```

- несколько однотипных шейдеров могут присутствовать в одной программе и вместе реализовывать этап обработки.

Пример 1: связывание атрибутов в буфере вершин и параметров шейдера

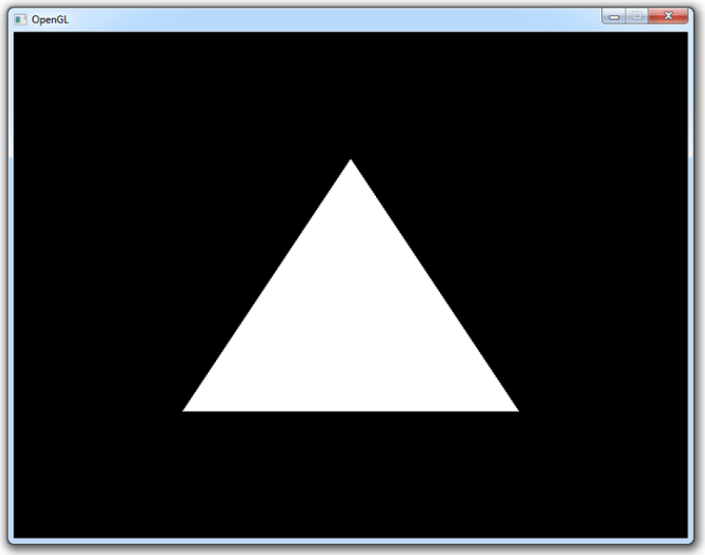
```
GLuint vao;
glGenVertexArrays(1, &vao);
glBindVertexArray(vao);

GLint posAttrib = glGetAttribLocation(shaderProgram, "position");
glVertexAttribPointer(posAttrib, 2, GL_FLOAT, GL_FALSE, 0, 0);
glEnableVertexAttribArray(posAttrib);
```

- VAO (Vertex Array Objects) хранят связи между параметрами и буферами вершин;
- параметры glVertexAttribPointer() (+ неявно сохраняется привязка к активному массиву вершин):
  - позиция (идентификатор) параметра;
  - количество значений;
  - тип значений;
  - нормализация к интервалу [-1.0; 1.0];
  - шаг (stride);
  - смещение (offset).

Пример 1: рисование

glDrawArrays(GL_TRIANGLES, 0, 3);



- вся необходимая для отрисовки информация берется из активного VAO, задаются только
  - тип примитива,
  - количество пропускаемых вершин
  - количество обрабатываемых вершин.

Пример 1: очистка объектов после выполнения программы

```
glDeleteProgram(shaderProgram);
glDeleteShader(fragmentShader);
glDeleteShader(vertexShader);

// glDetachShader(…);

glDeleteBuffers(1, &ebo);
glDeleteBuffers(1, &vbo);

glDeleteVertexArrays(1, &vao);
```

Постоянные параметры (Uniforms)

```
#version 150

uniform vec3 triangleColor;

out vec4 outColor;
void main()
{
        outColor = vec4(triangleColor, 1.0);
}
```

- uniform – параметр (переменная), значение которого одинаково для всех вызовов шейдера

Задание значения постоянного параметра

```
GLint uniColor = glGetUniformLocation(shaderProgram, "triangleColor");

glUniform3f(uniColor, 1.0f, 0.0f, 0.0f);
```

- uniform – параметр (переменная), значение которого одинаково для всех вызовов шейдера

Пример 1a: Задание цвета для каждой вершины – шейдеры

```
#version 150
in vec2 position;
in vec3 color;

out vec3 Color;
void main()
{
	Color = color;
	gl_Position = vec4(position, 0.0, 1.0);
}
```

```
#version 150
in vec3 Color;
out vec4 outColor;

void main()
{
	outColor = vec4(Color, 1.0);
}
```

Пример 1a: Задание цвета для каждой вершины – основная программа

```
GLint posAttrib = glGetAttribLocation(shaderProgram,
        "position");
glEnableVertexAttribArray(posAttrib);
glVertexAttribPointer(posAttrib, 2, GL_FLOAT, GL_FALSE,
                5*sizeof(float), 0);

GLint colAttrib = glGetAttribLocation(shaderProgram,
        "color");
glEnableVertexAttribArray(colAttrib);
glVertexAttribPointer(colAttrib, 3, GL_FLOAT, GL_FALSE,
        5*sizeof(float), (void*)(2*sizeof(float)));
```

Использование буферов идексов

```
GLuint ebo;
glGenBuffers(1, &ebo);

...

glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, ebo);
glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(elements),
elements, GL_STATIC_DRAW);

...

glDrawElements(GL_TRIANGLES, 3, GL_UNSIGNED_INT, 0);
```

Геометрические преобразования: основная программа

```
#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>
#include <glm/gtc/type_ptr.hpp>
        // transformation (model)
glm::mat4 model;
GLint uniModel = glGetUniformLocation(shaderProgram, “model");
glUniformMatrix4fv(uniModel, 1, GL_FALSE, glm::value_ptr(model));
        // camera (view)
glm::mat4 view = glm::lookAt( glm::vec3(1.2f, 1.2f, 1.2f),
        glm::vec3(0.0f, 0.0f, 0.0f), glm::vec3(0.0f, 0.0f, 1.0f) );
GLint uniView = glGetUniformLocation(shaderProgram, "view");
glUniformMatrix4fv(uniView, 1, GL_FALSE, glm::value_ptr(view));
        // projection
glm::mat4 proj = glm::perspective(45.0f, 800.0f / 600.0f, 1.0f, 10.0f);
GLint uniProj = glGetUniformLocation(shaderProgram, "proj");
glUniformMatrix4fv(uniProj, 1, GL_FALSE, glm::value_ptr(proj));
        // Draw a rectangle from the 2 triangles using 6 indices
glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_INT, 0);
```

Геометрические преобразования: вершинный шейдер

```
#version 150

in vec2 position;
in vec3 color;
in vec2 texcoord;

out vec3 Color;
out vec2 Texcoord;

uniform mat4 model;
uniform mat4 view;
uniform mat4 proj;

void main()
{
	Color = color;
	Texcoord = texcoord;
	gl_Position = proj * view * model * vec4(position, 0.0, 1.0);
}
```

Вопросы к экзамену

- Использование шейдеров.