Компьютерная графика
Лекция 19
GLSL. VBO

Я.М.Демяненко
dem@math.sfedu.ru

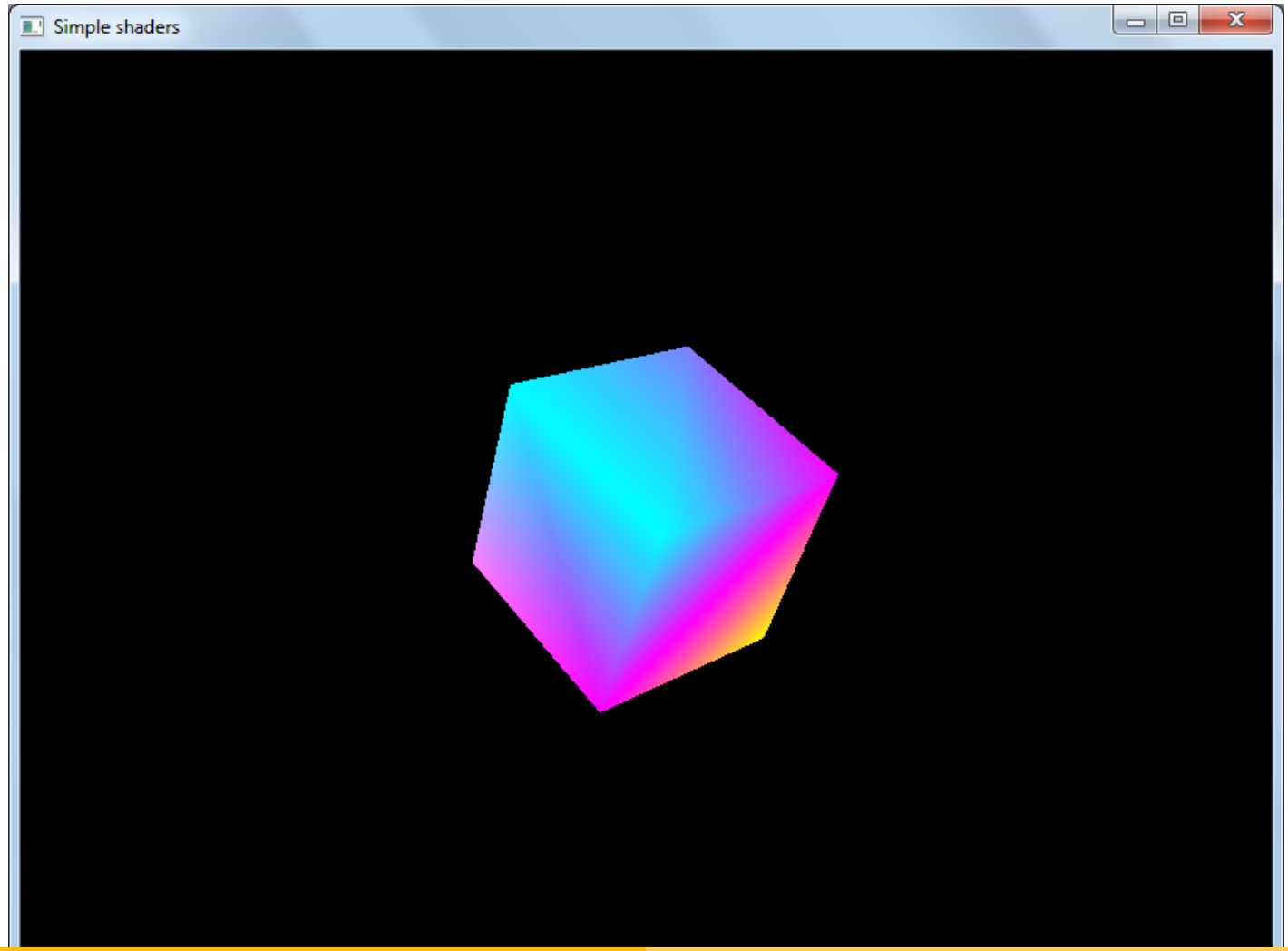
Южный федеральный университет
Институт математики, механики и компьютерных наук

2017

Содержание

Содержание

Что хотим получить?



Акценты

- Построения матрицы проекции
- Библиотека GML (OpenGL Mathematics)
- Использование множества VBO
- Передача данных из вершинного шейдера во фрагментный
- Отрисовка более сложных объектов

Содержание

Shader.h

```
#ifndef GLSHADER_H
#define GLSHADER_H

...
#include "glm.hpp"

...
using namespace glm;
class GlShader {...};

#endif
```

class GlShader 1

```
private:
    GLuint ShaderProgram;
    GLuint vertex_shader;
    GLuint fragment_shader;
```

class GlShader 2

```
private:
 //Функции печати лога шейдера
 void printInfoLogShader(GLuint shader);
 //Функция печати лога шейдерной программы
 void printInfoLogProgram(GLuint shader);

 GLuint loadSourcefile(const string& source_file_name,
                       GLuint shader_type);

 GLuint compileSource(const GLchar* source,
                      GLuint shader_type);

 void linkProgram();
```

compileSource(...)

```
...
GLuint compileSource(const GLchar* source,
                     GLuint shader_type)
{
  GLuint shader = glCreateShader(shader_type);
  glShaderSource(shader, 1, &source, NULL);
  glCompileShader(shader);

  #ifdef _DEBUG
    printInfoLogShader(shader);
  #endif
  return shader;
}
```

linkProgram()

```
...
ShaderProgram = glCreateProgram();
glAttachShader(ShaderProgram, vertex_shader);
glAttachShader(ShaderProgram, fragment_shader);
glLinkProgram(ShaderProgram);
...
```

class GlShader 3

```
public:
GlShader():ShaderProgram(0){}
~GlShader();

GLuint loadFiles(const string& vertex_file_name,
    const string& fragment_file_name);
GLuint load(const string& vertex_source,
    const string& fragment_source);
GLuint load(const GLchar* vertex_source,
    const GLchar* fragment_source);

void use(){glUseProgram(ShaderProgram);}
```

load(...)

```
load(const string& vertex_source,
           const string&  fragment_source)
{
vertex_shader= compileSource(vertex_source.c_str(),
                                    GL_VERTEX_SHADER);
fragment_shade= compileSource(fragment_source.c_str(),
                                    GL_FRAGMENT_SHADER);
 linkProgram();
}
```

деструктор для GlShader

```
glUseProgram(0);
glDeleteShader(vertex_shader);
glDeleteShader(fragment_shader);
glDeleteProgram(ShaderProgram);
```

class GlShader 4

```
public:
GLuint getIDProgram() {return ShaderProgram;}

bool isLoad() {return ShaderProgram != 0;}

//Attribute
GLint getAttribLocation(const GLchar* name)const;
GLint getAttribLocation(const string& name)const;
```

class GlShader 5

```
public:
    //Uniform get
    GLint getUniformLocation(const GLchar* name) const;
    GLint getUniformLocation(const string& name) const;

    //Uniform set
    void setUniform(GLint location, const vec4& value);
    void setUniform(GLint location, const vec3& value);
    void setUniform(GLint location, const vec2& value);
    void setUniform(GLint location, const mat4& value);
    void setUniform(GLint location, const GLint value);
```

Состояние шагов компилирования

```
void glGetShaderiv(GLuint object, GLenum type,
                                  int *param);
//Параметры:

//object - Дескриптор шейдера
//type - GL_COMPILE_STATUS
//param - возвращаемое значение, GL_TRUE если всё OK,
//                                  иначе GL_FALSE
```

Состояние шагов линковки

```
void glGetProgramiv(GLuint object, GLenum type,
                                   int *param);
//Параметры:

//object - Дескриптор программы
//type - GL_LINK_STATUS
//param - возвращаемое значение, GL_TRUE если всё OK,
//                               иначе GL_FALSE
```

Определение длины записи: лог компиляции

```
void glGetShaderiv(GLuint object,GLenum type,
                                        int *infoLogLen);
//Параметры:

//object - Дескриптор шейдера или программы
//type - GL_INFO_LOG_LENGTH
//infoLogLen - указатель на  возвращаемое значение
//целочисленной переменной для длины сообщения
```

Определение длины записи: лог линковки

```
void glGetProgramiv(GLuint object,GLenum type,
                                        int *infoLogLen);
//Параметры:

//object - Дескриптор программы
//type - GL_INFO_LOG_LENGTH
//infoLogLen - указатель на  возвращаемое значение
//целочисленной переменной для длины сообщения
```

Извлечение записи из журнала

```
glGetShaderInfoLog(GLuint object, int infoLogLen,
                                  int *Len,GLchar* infoLog);
glGetProgramInfoLog(shader, infologLen,
                                  int *Len,GLchar* infoLog);
```

Содержание

VBO и шейдерные идентификаторы

```
//ID атрибута вершин
GLint  Attrib_vertex;
//ID атрибута цветов
GLint  Attrib_color;
//ID юниформ матрицы проекции
GLint  Unif_matrix;
//ID Vertex Buffer Object
GLuint VBO_vertex;
//ID Vertex Buffer Object
GLuint VBO_color;
//ID VBO for element indices
GLuint VBO_element;
```

ещё идентификаторы

```
//Количество индексов
GLint Indices_count;
//Матрица проекции 4x4 из библиотеки GLM
mat4 Matrix_projection;
//glm/gtc/matrix_transform.hpp
```

ещё идентификаторы

```
void initGL()
{
   glClearColor(0, 0, 0, 0);
   glEnable(GL_DEPTH_TEST);
}
```

Исходный код шейдеров

```
const GLchar* vsSource =
    "attribute vec3 coord;\n"
    "attribute vec3 color;\n"
    "varying vec3 var_color;\n"
    "uniform mat4 matrix;\n"
    "void main() {\n"
    "gl_Position = matrix * vec4(coord, 1.0);\n"
    "var_color = color;\n"
    "}\n";
  const GLchar* fsSource =
    "varying vec3 var_color;\n"
    "void main() {\n"
    "gl_FragColor = vec4(var_color, 1.0);\n"
    "}\n";
```

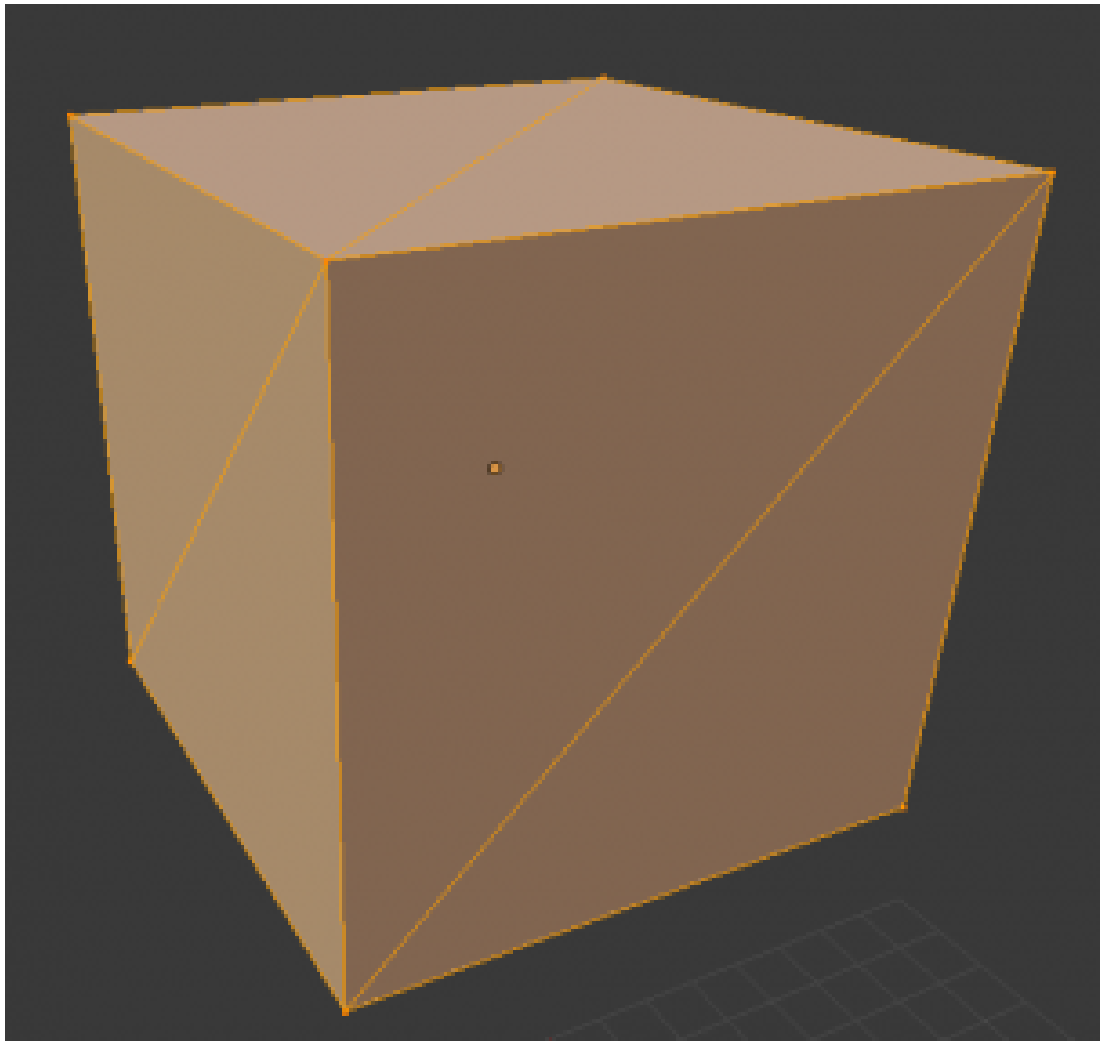
Функция инициализации шейдеров

```
void initShader()
{
...
//Исходный код шейдеров
...
//Вытягиваем ID атрибута из собранной программы
  Attrib_vertex = shader.getAttribLocation("coord");

//Вытягиваематрибута из собранной программы
  Attrib_color = shader.getAttribLocation("color");

//Вытягиваем ID юниформ матрицы проекции
  Unif_matrix = shader.getUniformLocation("matrix");
...
```

Сколько вершин и сколько примитивов?



Вершины куба

```
vertex vertices[] = {
    {-1.0f, -1.0f, -1.0f},
    {1.0f, -1.0f, -1.0f},
    {1.0f,  1.0f, -1.0f},
    {-1.0f, 1.0f, -1.0f},
    {-1.0f, -1.0f,  1.0f},
    {1.0f, -1.0f,  1.0f},
    {1.0f,  1.0f,  1.0f},
    {-1.0f,  1.0f,  1.0f}
};
```

Цвета куба без альфа компонента(RGB)

```
vertex colors[] = {
        {1.0f, 0.5f, 1.0f},
        {1.0f, 0.5f, 0.5f},
        {0.5f, 0.5f, 1.0f},
        {0.0f, 1.0f, 1.0f},
        {1.0f, 0.0f, 1.0f},
        {1.0f, 1.0f, 0.0f},
        {1.0f, 0.0f, 1.0f},
        {0.0f, 1.0f, 1.0f}
    };
```

Индексы вершин, общие и для цветов

```
GLint indices[] = {
  0, 4, 5, 0, 5, 1,
  1, 5, 6, 1, 6, 2,
  2, 6, 7, 2, 7, 3,
  3, 7, 4, 3, 4, 0,
  4, 7, 6, 4, 6, 5,
  3, 0, 1, 3, 1, 2
};
Indices_count = sizeof(indices) / sizeof(indices[0])
```

Способы ускорения передачи данных

- Вершинные массивы
- Вершинные буферы
- Индексные буферы

Основные функции VBO

```
GenBuffers(sizei n, uint *buffers);
//Создает n новых VBO и возвращает их ID номера
// как unsigned integer. Id 0 зарезервирован

 BindBuffer(enum target, uint buffer);
// Использует ранее созданный буфер как активный VBO

 BufferData(enum target,uint size,
     const void *data, enum usage);
//Выгружает данные в активный VBO

DeleteBuffers(sizei n, const uint *buffers);
//Удаляет указанные VBO из массива или VBO id
```

Буфер для вершин

```
glGenBuffers(1,&VBO_vertex);
glBindBuffer(GL_ARRAY_BUFFER, VBO_vertex);
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices),
             vertices ,GL_STATIC_DRAW);
```

Буфер для цветов вершин

```
glGenBuffers(1,&VBO_color);
glBindBuffer(GL_ARRAY_BUFFER,VBO_color);
glBufferData(GL_ARRAY_BUFFER,sizeof(colors),
             colors,GL_STATIC_DRAW);
```

Буфер для индексов вершин

```
glGenBuffers(1,&VBO_element);
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER,VBO_element);
glBufferData(GL_ELEMENT_ARRAY_BUFFER,sizeof(indices),
             indices,GL_STATIC_DRAW);
```

Освобождение буфера

```
void freeVBO()
{
    glBindBuffer(GL_ARRAY_BUFFER, 0);
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, 0);

    glDeleteBuffers(1, &VBO_vertex);
    glDeleteBuffers(1, &VBO_element);
    glDeleteBuffers(1, &VBO_color);
}
```

Использование GLM-функций

```
void resizeWindow(int width, int height) {
 GlViewport(0, 0, width, height);
 height=height>0 ? height:1;
 const GLfloat aspectRatio=
                  (GLfloat)width/(GLfloat)height;
 Matrix_projection=glm::perspective(45.0f,
                  aspectRatio,1.0f,200.0f);
// Перемещаем центр нашей оси координат, чтобы увидеть куб
 Matrix_projection=glm::translate(Matrix_projection,
                  vec3(0.0f,0.0f,-10.0f));
// Поворачиваем ось координат (т е  весь мир),
//чтобы развернуть отрисованное
   Matrix_projection=glm::rotate(Matrix_projection,
                  60.0f,vec3(1.0f,1.0f,0.0f));
```

render()

```
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

//Устанавливаем шейдерную программу текущей
shader.use();
//Передаем матрицу в шейдер
shader.setUniform(Unif_matrix, Matrix_projection);

//Подключаем буфер с индексами вершин
//общий для цветов и их вершин
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, VBO_element);
```

ВЕРШИНЫ

```
//Включаем массив атрибутов для вершин
glEnableVertexAttribArray(Attrib_vertex);
//Подключаем VBO
glBindBuffer(GL_ARRAY_BUFFER, VBO_vertex);
//Указывая pointer 0 при подключенном буфере,
//мы указываем что данные в VBO
glVertexAttribPointer(Attrib_vertex, 3,
      GL_FLOAT, GL_FALSE, 0, 0);
```

ЦВЕТА

```
//Включаем массив атрибутов для цветов
glEnableVertexAttribArray(Attrib_color);
glBindBuffer(GL_ARRAY_BUFFER, VBO_color);
glVertexAttribPointer(Attrib_color, 3,
        GL_FLOAT, GL_FALSE, 0, 0);
```

Передаем данные на видеокарту

```
//Передаем данные на видеокарту
glDrawElements(GL_TRIANGLES, Indices_count,
        GL_UNSIGNED_INT, 0);

//Отключаем массив атрибутов
glDisableVertexAttribArray(Attrib_vertex);

//Отключаем массив атрибутов
glDisableVertexAttribArray(Attrib_color);

checkOpenGLerror();

glutSwapBuffers();
```