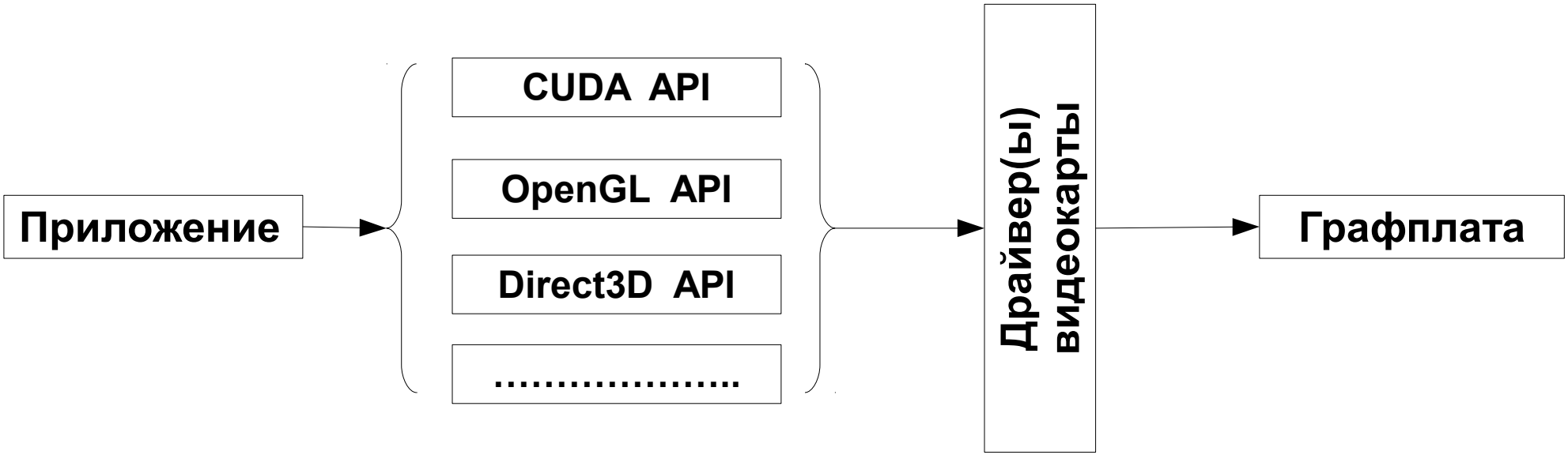
Лекция 10

- Интерфейсы программирования GPU.
- Расширения OpenGL.
- Контекст OpenGL и взаимодействие с подсистемами ОС.
- Автоматное программирование.
- Шейдеры.
- Конвейер OpenGL.
- Библиотека GLM.

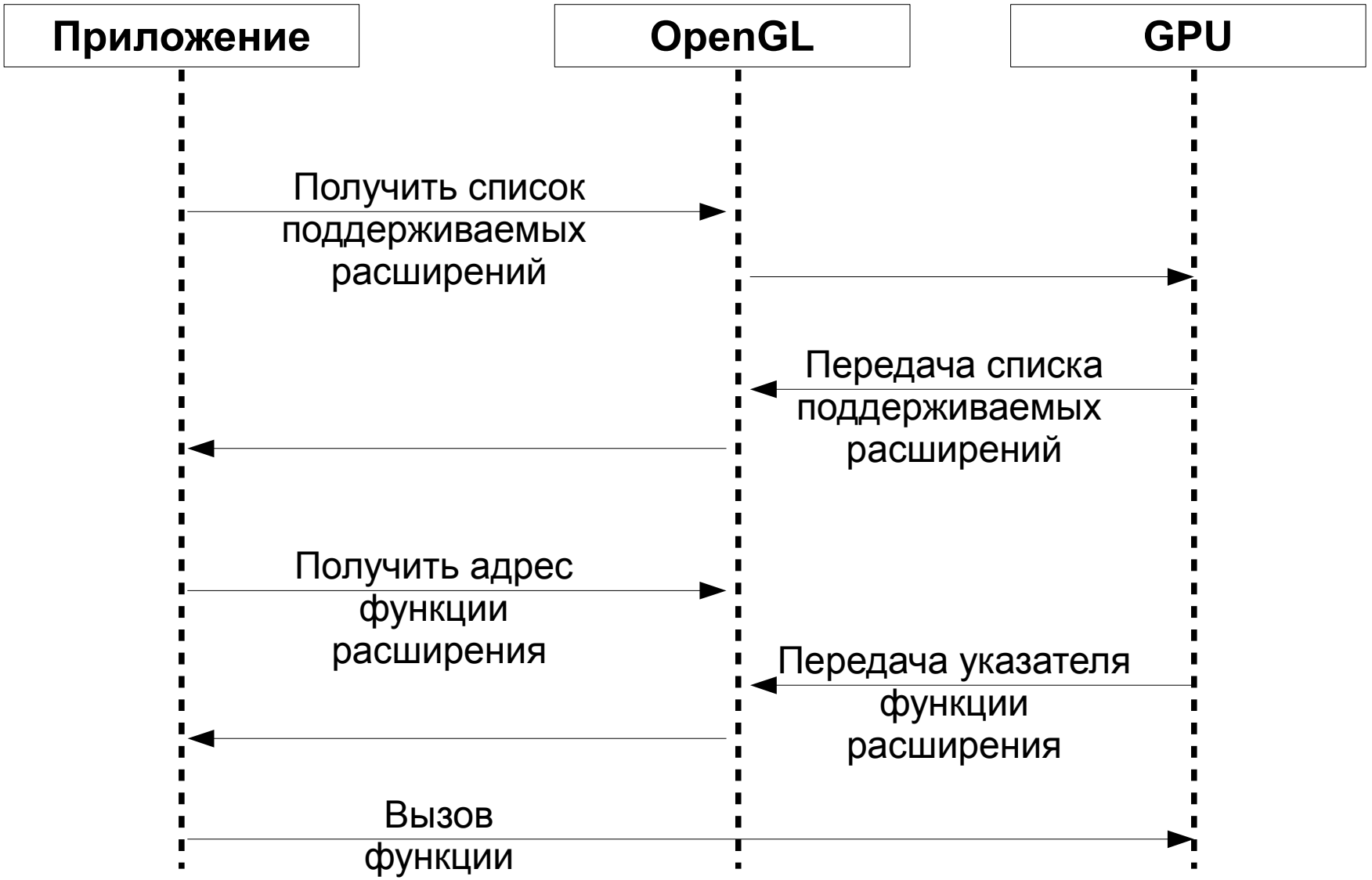
Интерфейсы программирования GPU



main.cpp

```
#include <GL/glew.h>
#include <GLFW/glfw3.h>

#include <stdio.h>
#include <string>
#include <stdlib.h>

void initGL();
int initBuffer();
void display();
void myCleanup();

GLFWwindow* window;

const unsigned int window_width  = 512;
const unsigned int window_height = 512;
```

Загрузка расширений



GLEW (Open**GL** **E**xtension **W**rangler Library) — библиотека для загрузки расширений.

GLEW обеспечивает доступ ко всем функциям GL (не требуется включать gl.h glext.hи т.д.)

Все необходимые ресурсы конкретного OpenGL приложения составляют контекст. OpenGL-контекст создаётся средствами операционной системы.

GLFW и **GLUT** — наиболее популярные библиотеки для создания контекста и организации взаимодействия с оконной и файловой системами.

GLM(*Open**GL M**athematics*) — математическая библиотека для графичиских приложений, базирующаяся на спецификации GLSL.

Загрузка библиотек:

GLEW - http://glew.sourceforge.net/

GLFW - http://www.glfw.org/

GLM - https://glm.g-truc.net

Установка в MS Windows и конфигурация приложений Visual Studio:

https://www.youtube.com/watch?v=gCkcP0GcCe0

```
int main(){
  initGL();
  initBuffer();

  do{
      glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
      //glClearColor(0.7,0.7,0.7,1.0);
      glPointSize(6);
      display();
      glfwSwapBuffers(window);
      glfwPollEvents();
  }while( glfwGetKey(window, GLFW_KEY_ESCAPE ) != GLFW_PRESS &&
                                                  glfwWindowShouldClose(window) == 0 );

  glfwSetInputMode(window, GLFW_STICKY_KEYS, GL_TRUE);

  myCleanup();

  glfwTerminate();
  return 0;
}
```

```
void initGL(){
	if( !glfwInit() )
	{
		fprintf( stderr, "Failed to initialize GLFW\n" );
		getchar();
		return;
	}
	glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 4);
	glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
	glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);
	glfwWindowHint(GLFW_OPENGL_PROFILE,
                                        GLFW_OPENGL_COMPAT_PROFILE);

	window = glfwCreateWindow( window_width, window_height,
                                                        "Template window", NULL, NULL);
	if( window == NULL ){
		fprintf( stderr, "Failed to open GLFW window. \n" );
		getchar();
		glfwTerminate();
		return;
	}
	glfwMakeContextCurrent(window);
```

```
	// Initialize GLEW
	glewExperimental = true;
	if (glewInit() != GLEW_OK) {
		fprintf(stderr, "Failed to initialize GLEW\n");
		getchar();
		glfwTerminate();
		return;
	}

	return;
}
```

util_template.cpp

```
#include <GL/glew.h>
#include <stdio.h>
#include <stdlib.h>
#include <string>

void checkErrors(std::string desc) {
	GLenum e = glGetError();
	if (e != GL_NO_ERROR) {
		fprintf(stderr, "OpenGL error in \"%s\": %s (%d)\n", desc.c_str(),
		                                                    gluErrorString(e), e);
		exit(20);
	}
}

const unsigned int window_width  = 512;
const unsigned int window_height = 512;

GLuint bufferID;
GLuint progHandle;
GLuint genRenderProg();

const int num_of_verticies=3;
```

```
int initBuffer(){

  glGenBuffers( 1,&bufferID);

  glBindBuffer( GL_ARRAY_BUFFER, bufferID);

  static const GLfloat vertex_buffer_data[] = {
                        -0.9f, -0.9f, -0.0f, 1.0f, 0.0f, 0.0f,
                         0.0f,   0.0f,  0.0f, 0.0f, 1.0f, 0.0f,
                         0.9f,  -0.5f,  0.0f, 0.0f, 0.0f, 1.0f,
                        };

  glBufferData( GL_ARRAY_BUFFER, 6*num_of_verticies*sizeof(float),
                                                    vertex_buffer_data, GL_STATIC_DRAW );

  return 0;
}
```

```
void display(){

  progHandle=genRenderProg();
  glUseProgram(progHandle);

  GLint posPtr = glGetAttribLocation(progHandle, "pos");
  glVertexAttribPointer(posPtr, 3, GL_FLOAT, GL_FALSE, 24, 0);
  glEnableVertexAttribArray(posPtr);

  GLint colorPtr = glGetAttribLocation(progHandle, "color");
  glVertexAttribPointer(colorPtr, 3, GL_FLOAT, GL_FALSE, 24, (const
                                                              GLvoid*)12);
  glEnableVertexAttribArray(colorPtr);

  glDrawArrays(GL_TRIANGLES,0, num_of_verticies);

  glDisableVertexAttribArray(posPtr);
  glDisableVertexAttribArray(colorPtr);
}

void myCleanup(){
      glDeleteBuffers(1, &bufferID);
      glDeleteProgram(progHandle);
}
```

sh_template.cpp

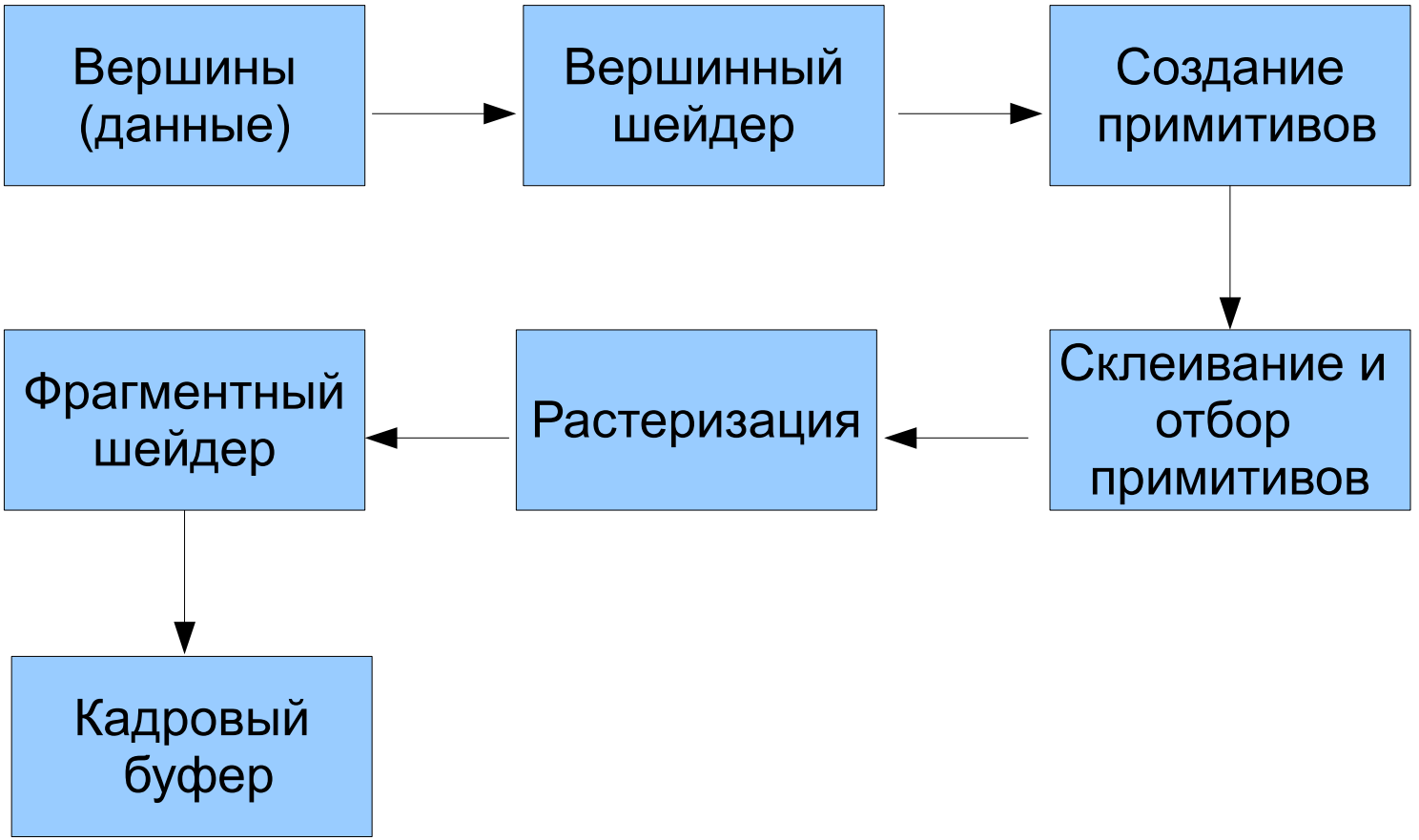
```
#include <GL/glew.h>
#include <stdio.h>
#include <string>
#include <stdlib.h>

void checkErrors(std::string desc);
GLuint genRenderProg() {
    GLuint progHandle = glCreateProgram();
    GLuint vp = glCreateShader(GL_VERTEX_SHADER);
    GLuint fp = glCreateShader(GL_FRAGMENT_SHADER);
```

```
const char *vpSrc[] = {
      "#version 430\n",
      "layout(location = 0) in vec3 pos;\
       layout(location = 1) in vec3 color;\
       out vec4 vs_color;\
       void main() {\
          gl_Position = vec4(pos,1);\
          vs_color=vec4(color,1.0);\
       }"
};

const char *fpSrc[] = {
      "#version 430\n",
       "in vec4 vs_color;\
        out vec4 fcolor;\
        void main() {\
             fcolor = vs_color;\
       }"
};

glShaderSource(vp, 2, vpSrc, NULL);
glShaderSource(fp,  2, fpSrc,  NULL);
```

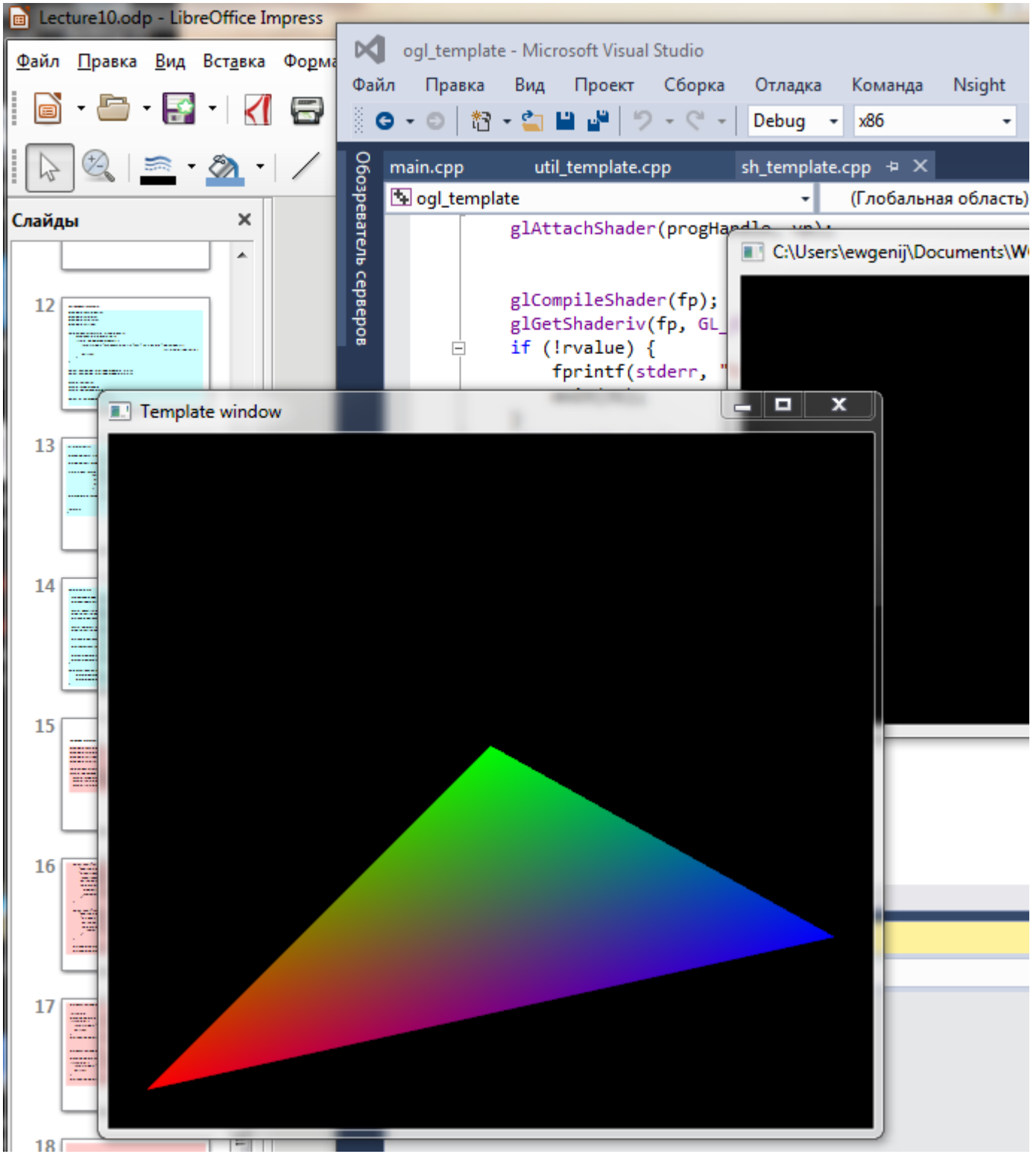
```
glCompileShader(vp);

int rvalue;
glGetShaderiv(vp, GL_COMPILE_STATUS, &rvalue);
if (!rvalue) {
    fprintf(stderr, "Error in compiling vp\n");
    exit(30);
 }
 glAttachShader(progHandle, vp);


 glCompileShader(fp);

 glGetShaderiv(fp, GL_COMPILE_STATUS, &rvalue);
 if (!rvalue) {
    fprintf(stderr, "Error in compiling fp\n");
    exit(31);
 }
glAttachShader(progHandle, fp);
```

```
    glLinkProgram(progHandle);

    glGetProgramiv(progHandle, GL_LINK_STATUS, &rvalue);
    if (!rvalue) {
        fprintf(stderr, "Error in linking sp\n");
        exit(32);
    }

    checkErrors("Render shaders");

    return progHandle;
}
```

Программируемый конвейер OpenGL



>= OpenGL 2.0 — поддержка GLSL


Lecture10.odp - LibreOffice Impress
Файл Правка Вид Вставка Форма
Слайды
ogl_template - Microsoft Visual Studio
Файл Правка Вид Проект Сборка Отладка Команда Nsight
Debug
x86
Обозреватель серверов
main.cpp
util_template.cpp
sh_template.cpp
ogl_template
(Глобальная область)
glAttachShader(progHandle, vp);
glCompileShader(fp);
glGetShaderiv(fp, GL_
if (!rvalue) {
fprintf(stderr, "
C:\Users\ewgenij\Documents\W
Template window

util_template.cpp

```
……
#include <glm/vec3.hpp> // glm::vec3
#include <glm/vec4.hpp> // glm::vec4
#include <glm/mat4x4.hpp> // glm::mat4
#include <glm/gtc/matrix_transform.hpp>
……
```

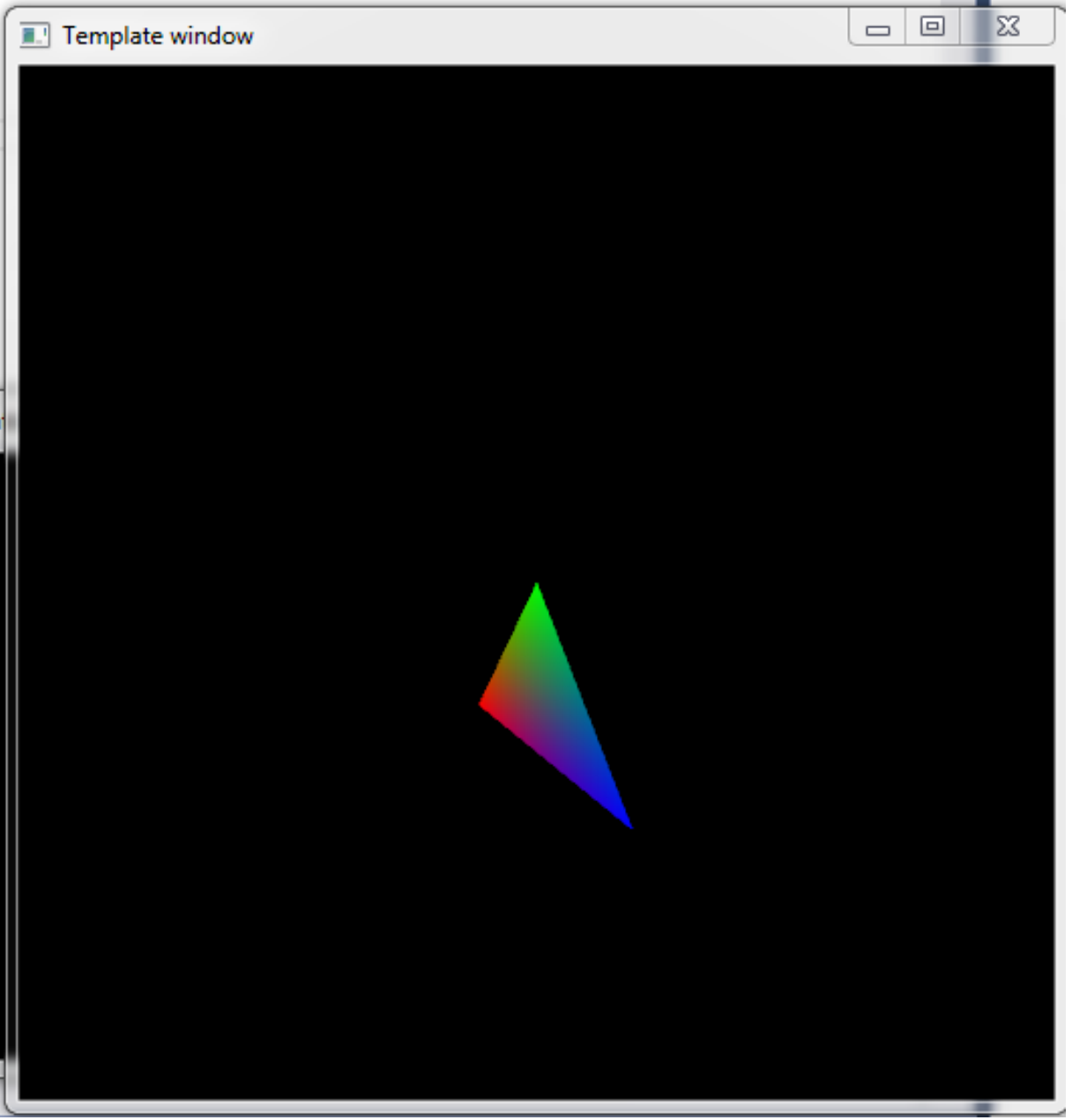
```
……..
void camera(){
  glm::mat4 Projection = glm::perspective(glm::radians(60.0f),
                    (float) window_width / (float)window_height, 0.1f, 0.0f);

  glm::mat4 View = glm::lookAt(
    glm::vec3(3,1,1), // Камера находится в точке (3,1,1)
    glm::vec3(0,0,0), // и направлена на начало координат.
    glm::vec3(0,1,0)  // Ось Y направлена вверх, ( 0,-1,0) - вниз.
    );

  glm::mat4 Model = glm::mat4(1.0f);
  glm::mat4 mvp = Projection * View * Model;

  GLuint MatrixID = glGetUniformLocation(progHandle, "MVP");
  glUniformMatrix4fv(MatrixID, 1, GL_FALSE, &mvp[0][0]);
}
……..
void display(){
  camera();
……..
```

sh_template.cpp

```
……..
 void main() {\
   gl_Position = MVP*vec4(pos,1);\
……..
```

```
const char *vpSrc[] = {
    "#version 430\n",
    "layout(location = 0) in vec3 pos;\
     layout(location = 1) in vec3 color;\
     out vec4 vs_color;\
     uniform mat4 MVP;\
     void main() {\
        gl_Position = MVP*vec4(pos,1);\
        vs_color=vec4(color,1.0);\
     }"
};

const char *fpSrc[] = {
    "#version 430\n",
     "in vec4 vs_color;\
```

C:\Users\ewgenij\Documents\WORKSHOP\OpenGL\ogl_templa



Спасибо за внимание!