Graphics Lab 3

29/09/2022

Structure of an OpenGL programme

main() - Your main() function will always be relatively unchanged

```
int main(int argc, char** argv) {

    /* initialise glut and setup display window */
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowSize(width, height);
    glutCreateWindow("Hello Noise!");

    /* register callback functions */
    glutDisplayFunc(display);
    glutIdleFunc(updateScene);
    glutKeyboardFunc(keypress);
    glutSpecialFunc(specialKeyPress);
    glutMotionFunc(pressAndMoveCallback);
    glutMouseWheelFunc(wheelCallback);

    /* initialise glew and check for errors */
    glewExperimental = GL_TRUE;
    GLenum res = glewInit();
    if (res != GLEW_OK) {
        fprintf(stderr, "Error: '%s'\n", glewGetErrorString(res));
        return 1;
    }

    init(); /* Set up your objects and shaders */
    glutMainLoop(); /* Begin infinite event loop */
    return 0;
}
```

Your **main()** should:

1. Create your **display window**

2. **Register call back functions**. Every programme requires a display callback – registered using glutDisplayFunc(). Other call back functions allow for interactive control, e.g. mouse and keyboard input

3. **Initialize glew**

4. **Set up** any meshes and shaders in your **init()** function
5. Begin the **infinite event loop**.

Structure of an OpenGL programme

init() – set everything up here for later use

```
void init()
{
	glClearColor(0.1f, 0.1f, 0.1f, 1.0f);

	/* load shaders */
	Shader woodShader("../Shaders/good/cperlin3VSWood.txt", "../Shaders/good/cperlin3FSWood.txt");
	shaderPrograms.push_back(woodShader);
	Shader marbleShader("../Shaders/good/cperlin3VSMarble.txt", "../Shaders/good/cperlin3FSMarble.txt");
	shaderPrograms.push_back(marbleShader);
	Shader cloudShader("../Shaders/good/cperlin4VSCloud.txt", "../Shaders/good/cperlin4FSCloud.txt");
	shaderPrograms.push_back(cloudShader);
	Shader fireShader("../Shaders/good/cperlin4VSFire.txt", "../Shaders/good/cperlin4FSFire.txt");
	shaderPrograms.push_back(fireShader);

	/* load meshes */
	Model teapot("../Models/teapot.obj");
	models.push_back(teapot);
}
```

- Set up any shader programmes you plan to use
- Load all objects you want to use

- **For each shader** we want, we create an **instance of our shader class**
- **For each model** we want to use, we create **an instance of our model class**
- Note: you need to do this in such a way that you can **access them in your display() function**
- In this example all shader programmes are added to a global list called shaderProgrammes and similarly all objects to a global list called models.

Structure of an OpenGL programme

display() – this is where everything happens!

```
void display() {

    // tell GL to only draw onto a pixel if the shape is closer to the viewer
    glEnable(GL_DEPTH_TEST); /* enable depth-testing */
    glDepthFunc(GL_LESS); /* depth-testing interprets a smaller value as "closer" */
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);


    /* transformation matrices */
    glm::mat4 S = glm::scale(glm::mat4(1.0f), glm::vec3(0.1f, 0.1f, 0.1f));
    glm::mat4 T = glm::translate(glm::mat4(1.0f), glm::vec3(0.0f, -1.5f, -10.0f));
    glm::mat4 M = T *  S;
    glm::mat4 V = camera.GetViewMatrix();
    glm::mat4 P = glm::perspective(glm::radians(camera.Zoom), (float)width / (float)height, 0.1f, 100.0f);

    /* top left */
    glViewport(0, height/2 , width / 2, height / 2);
    shaderPrograms[0].use();
    shaderPrograms[0].setMat4("M", M);
    shaderPrograms[0].setMat4("V", V);
    shaderPrograms[0].setMat4("P", P);
    shaderPrograms[0].setFloat("r1", 0.4);
    shaderPrograms[0].setFloat("r2", 3.0);
    shaderPrograms[0].setFloat("r3", 3.0);
    shaderPrograms[0].setVec3("lightPosition", glm::vec3(0.0f, 0.0f, 0.0f));
    models[0].Draw(shaderPrograms[0]);

    /* top right */
    glViewport(width/2, height/2, width/2, height/2);
    shaderPrograms[1].use();
    shaderPrograms[1].setMat4("M", M);
    shaderPrograms[1].setMat4("V", V);
    shaderPrograms[1].setMat4("P", P);
    shaderPrograms[1].setFloat("r1", 4.0);
    shaderPrograms[1].setFloat("r2", 3.0);
    shaderPrograms[1].setFloat("r3", 3.0);
    shaderPrograms[1].setVec3("lightPosition", glm::vec3(0.0f, 00.0f, 0.0f));
    shaderPrograms[1].setVec3("viewPosition", camera.Position);
    models[0].Draw(shaderPrograms[1]);

    /* bottom left */
    glViewport(0, 0, width/2, height/2);
    shaderPrograms[2].use();
    shaderPrograms[2].setMat4("M", M);
    shaderPrograms[2].setMat4("V", V);
    shaderPrograms[2].setMat4("P", P);
    shaderPrograms[2].setFloat("r1", 4.0);
    shaderPrograms[2].setFloat("r2", 3.0);
    shaderPrograms[2].setFloat("r3", 3.0);
    shaderPrograms[2].setFloat("tm", 0.000075);
    shaderPrograms[2].setFloat("time", timeGetTime());
    shaderPrograms[2].setVec3("lightPosition", glm::vec3(0.0f, 00.0f, 0.0f));
    models[0].Draw(shaderPrograms[2]);

    /* bottom right */
    glViewport(width/2, 0, width/2, height/2);
    shaderPrograms[3].use();
    shaderPrograms[3].setMat4("M", M);
    shaderPrograms[3].setMat4("V", V);
    shaderPrograms[3].setMat4("P", P);
    shaderPrograms[3].setFloat("r1", 4.0);
    shaderPrograms[3].setFloat("r2", 3.0);
    shaderPrograms[3].setFloat("r3", 3.0);
    shaderPrograms[3].setFloat("tm", 0.00004);
    shaderPrograms[3].setFloat("time", timeGetTime());
    shaderPrograms[3].setVec3("lightPosition", glm::vec3(0.0f, 0.0f, 0.0f));
    models[0].Draw(shaderPrograms[3]);

    glutSwapBuffers();
    glutPostRedisplay();
}
```

Example 1: where we have used a shader class and model class

```
/* top left (Phong) */
glViewport(0, height / 2, width / 2, height / 2);
ShaderPrograms[0].use();
ShaderPrograms[0].setVec3("ViewPosition", ViewPosition);
ShaderPrograms[0].setMat4("M", mM);
ShaderPrograms[0].setMat4("V", mV);
ShaderPrograms[0].setMat4("P", mP);
ShaderPrograms[0].setMaterial("material", gold);
ShaderPrograms[0].setLight("light", light1);
glBindVertexArray(teapot_vao);
glDrawArrays(GL_TRIANGLES, 0, teapot_vertex_count);
ShaderPrograms[4].use();
glBindVertexArray(light_vao);
glDrawArrays(GL_TRIANGLES, 0, cube_vertex_count);

/* top right (Minn) */

glViewport(width / 2, height / 2, width / 2, height / 2);
ShaderPrograms[2].use();
ShaderPrograms[2].setVec3("ViewPosition", ViewPosition);
ShaderPrograms[2].setMat4("M", mM);
ShaderPrograms[2].setMat4("V", mV);
ShaderPrograms[2].setMat4("P", mP);
ShaderPrograms[2].setMaterial("material", silver);
ShaderPrograms[2].setLight("light", light1);
glBindVertexArray(teapot_vao);
glDrawArrays(GL_TRIANGLES, 0, teapot_vertex_count);
ShaderPrograms[4].use();
glBindVertexArray(light_vao);
glDrawArrays(GL_TRIANGLES, 0, cube_vertex_count);

//bottom left (Toon)
glViewport(0, 0, width / 2, height / 2);
ShaderPrograms[3].use();
ShaderPrograms[3].use();
ShaderPrograms[3].setVec3("ViewPosition", ViewPosition);
ShaderPrograms[3].setMat4("M", mM);
ShaderPrograms[3].setMat4("V", mV);
ShaderPrograms[3].setMat4("P", mP);
ShaderPrograms[3].setMaterial("material", gold);
ShaderPrograms[3].setLight("light", light1);
glBindVertexArray(teapot_vao);
glDrawArrays(GL_TRIANGLES, 0, teapot_vertex_count);
ShaderPrograms[4].use();
glBindVertexArray(light_vao);
glDrawArrays(GL_TRIANGLES, 0, cube_vertex_count);

//bottom right
glViewport(width/2, 0, width / 2, height / 2);
ShaderPrograms[1].use();
ShaderPrograms[1].use();
ShaderPrograms[1].setVec3("ViewPosition", ViewPosition);
ShaderPrograms[1].setMat4("M", mM);
ShaderPrograms[1].setMat4("V", mV);
ShaderPrograms[1].setMat4("P", mP);
ShaderPrograms[1].setMaterial("material", brass);
ShaderPrograms[1].setLight("light", light1);
glBindVertexArray(teapot_vao);
glDrawArrays(GL_TRIANGLES, 0, teapot_vertex_count);
ShaderPrograms[4].use();
glBindVertexArray(light_vao);
glDrawArrays(GL_TRIANGLES, 0, cube_vertex_count);
```

Example 1: where we haven't used model class (shows binding VAOs for each object)

For each object you wish to draw you must:

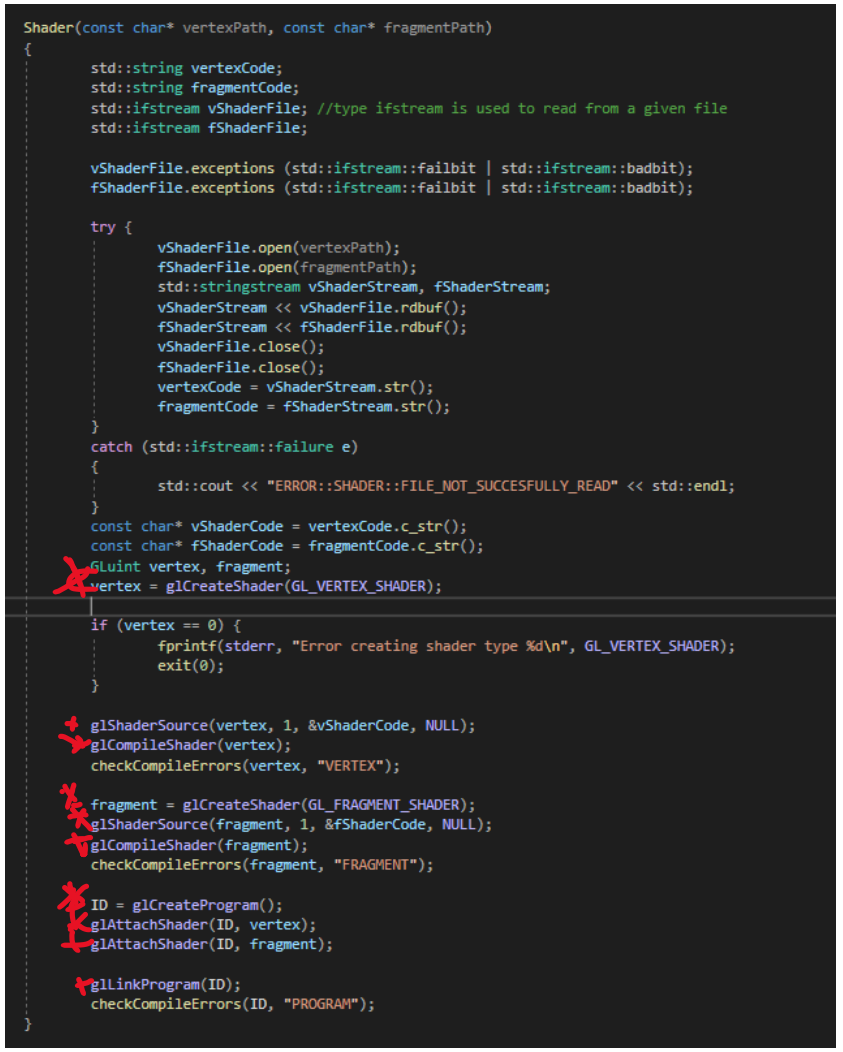
1. Select the shader programme you wish to use using `glUseProgram();`
2. Set values for all the uniform variables used in the shader programme
3. Bind the buffer containing the data for the object you wish to draw using `glBindVertexArray();`
4. Draw the object using `glDrawArrays();`

Shader progamme

Note: there are both “shader programmes” and “shaders”

For each shader programme you wish to use, you must:

- Create a shader programme using `GLuint ShaderProgramID = glCreateProgram();`
- **For each shader** (i.e. vertex and fragment shaders) you must
  - Create a shader using `GLuint ShaderObj = glCreateShader(ShaderType);`
  - Add the source code for the shader using `glShaderSource();`
  - Compile the shader using `glCompileShader();`
  - Attach the shader to the shader programme using `glAttachShader();`
- Link the shader programme using `glLinkProgram()`
- Switch between shader programmes when drawing using `glUseProgram()`



```
Shader(const char* vertexPath, const char* fragmentPath)
{
	std::string vertexCode;
	std::string fragmentCode;
	std::ifstream vShaderFile; //type ifstream is used to read from a given file
	std::ifstream fShaderFile;

	vShaderFile.exceptions (std::ifstream::failbit | std::ifstream::badbit);
	fShaderFile.exceptions (std::ifstream::failbit | std::ifstream::badbit);

	try {
		vShaderFile.open(vertexPath);
		fShaderFile.open(fragmentPath);
		std::stringstream vShaderStream, fShaderStream;
		vShaderStream << vShaderFile.rdbuf();
		fShaderStream << fShaderFile.rdbuf();
		vShaderFile.close();
		fShaderFile.close();
		vertexCode = vShaderStream.str();
		fragmentCode = fShaderStream.str();
	}
	catch (std::ifstream::failure e)
	{
		std::cout << "ERROR::SHADER::FILE_NOT_SUCCESFULLY_READ" << std::endl;
	}
	const char* vShaderCode = vertexCode.c_str();
	const char* fShaderCode = fragmentCode.c_str();
	GLuint vertex, fragment;
	vertex = glCreateShader(GL_VERTEX_SHADER);

	if (vertex == 0) {
		fprintf(stderr, "Error creating shader type %d\n", GL_VERTEX_SHADER);
		exit(0);
	}

	glShaderSource(vertex, 1, &vShaderCode, NULL);
	glCompileShader(vertex);
	checkCompileErrors(vertex, "VERTEX");

	fragment = glCreateShader(GL_FRAGMENT_SHADER);
	glShaderSource(fragment, 1, &fShaderCode, NULL);
	glCompileShader(fragment);
	checkCompileErrors(fragment, "FRAGMENT");

	ID = glCreateProgram();
	glAttachShader(ID, vertex);
	glAttachShader(ID, fragment);

	glLinkProgram(ID);
	checkCompileErrors(ID, "PROGRAM");
}
```

Callback functions

Allow for interactive control

- See https://www.opengl.org/resources/libraries/glut/spec3/node45.html for a list of different callback types
- The call back function must be registered in main() using something like glut*DesiredCallBackType*(*CallbackName*)
- The argument passed to the callback registration function is the name of the corresponding callback function in your code
- Write your callback function code. This function will be called each time an input event (e.g. keypress) triggers it.

```
int main(int argc, char** argv) {

	/* initialise glut and setup display window */
	glutInit(&argc, argv);
	glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
	glutInitWindowSize(width, height);
	glutCreateWindow("Hello Noise!");

	/* register callback functions */
	glutDisplayFunc(display);
	glutIdleFunc(updateScene);
	glutKeyboardFunc(keypress);
	glutSpecialFunc(specialKeyPress);
	glutMotionFunc(pressAndMoveCallback);
	glutMouseWheelFunc(wheelCallback);

	/* initialise glew and check for errors */
	glewExperimental = GL_TRUE;
	GLenum res = glewInit();
	if (res != GLEW_OK) {
		fprintf(stderr, "Error: '%s'\n", glewGetErrorString(res));
		return 1;
	}

	init(); /* Set up your objects and shaders */
	glutMainLoop(); /* Begin infinite event loop */
	return 0;
}
```

```
void wheelCallback(int wheel, int direction, int x, int y)
{
	/* wheel is wheel numner, direction is 1/-1, x and y are mouse posiion */
	camera.SetZoom(direction);

}
```

in this example, wheelCallback() will be called everytime a user adjusts the mouse scroll wheel and the camera zoom will be updated accordingly.

Note that any object/variable your callback function effects should be available globally so that it can be accessed elsewhere, e.g. in your display function

LAB 3 Specifics

- Your program should have the following features:
  - Keyboard control of the translation of the root object
  - Create a (simple) model of your own in a modelling package and import it
  - Import free models from the internet

Lab 3 - Keyboard control of the translation of the root object

Register and define your callback function

1. **glutKeyboardFunc()** or **glutSpecialFunc()** to register a callback function for keyboard control in your main() function

```
glutSpecialFunc(specialKeyPress);
```

2. Define your callback function, i.e. specialKeyPress() in this example. It could be something like this:

```
void specialKeyPress(int key, int x, int y)
{
        switch (key) {
        case GLUT_KEY_LEFT:
                translate_base += vec3(-1.0f, 0.0f, 0.0f);
                break;
        case GLUT_KEY_RIGHT:
                translate_base += vec3(1.0f, 0.0f, 0.0f);
                break;
        case GLUT_KEY_DOWN:
                translate_base += vec3(0.0f, 0.0f, 1.0f);
                break;
        case GLUT_KEY_UP:
                translate_base += vec3(0.0f, 0.0f, -1.0f);
                break;
        case GLUT_KEY_INSERT:
                break;
        }
}
```

Note that translate_base needs to be a global variable so that it can be accessed in your display function

Lab 3 - Keyboard control of the translation of the root object

Create a uniform variable

Create a matrix that translates a point

The value of this is altered using your call back function (previous slide)

```
glm::mat4 model = glm::translate(glm::mat4(1.0f), translate_base);
GLuint mMID = glGetUniformLocation(ObjectShaderProgramID, "mM");
glUniformMatrix4fv(mMID, 1, GL_FALSE, glm::value_ptr(model));
```

```
#version 330

in vec3 vertex_position;
in vec3 vertex_normals;

out vec3 n_eye;

uniform mat4 vM;
uniform mat4 pM;
uniform mat4 mM;

void main(){
        n_eye = (vM *mM * vec4 (vertex_normals, 0.0)).xyz;
        gl_Position =  pM * vM * mM * vec4 (vertex_position, 1.0);
}
```

Create an ID for your uniform variable and use this ID to pass the value of the translation matrix to the uniform variable in your shader

The name here must match exactly the name of the uniform variable in your shader programme

Translate every vertex by the value of translate_base

Create a (simple) model of your own in a modelling package

- For example, you could use **Blender** https://www.blender.org/ to create your model

- Don’t spend too long on this (unless you want to) – this isn’t an artistic design module. For example, just make something simple using spheres, cubes and cylinders.

Free models from the internet

- https://www.turbosquid.com/

- https://www.cgtrader.com/

- https://free3d.com/

Import the models you created or downloaded

- Code for a model loader will be provided to you?

- Better and more flexible (but more difficult?) solution:
  - Use the ASSIMP library https://assimp-docs.readthedocs.io/en/v5.1.0/

  - Use https://learnopengl.com/Model-Loading/Assimp to help you install ASSIMP

  - Once you have ASSIMP installed, you should make an object class. Again learnopengl will help you
    - They provide a model class https://learnopengl.com/code_viewer_gh.php?code=includes/learnopengl/model.h
    - and a mesh class https://learnopengl.com/code_viewer_gh.php?code=includes/learnopengl/mesh.h