Technical University of Cluj-Napoca

CTI En Group 30433

2019-2020 1st semester

Graphical Processing Systems
OpenGL Project

Student: Simina Radu-Alexandru

Summary

Project Requirements

The subject of the project consists in the photorealistic presentation of 3D objects using OpenGL library. The user directly manipulates by mouse and keyboard inputs the scene of objects.

- **(2p)** visualization of the scene: scaling, translation, rotation, camera movement
  - using keyboard and mouse
  - using animation
- **(1p)** specification of light sources (minimum two different lights)

- **(0.5p)** viewing solid, wireframe objects, polygonal and smooth surfaces
- **(1p)** texture mapping and materials

  - textures quality and level of detail

  - textures mapping on objects

- **(1p)** exemplify shadow computation
- **(0.5p)** exemplify animation of object components
- **(3p)** photo-realism, scene complexity, detailed modeling, algorithms development and implementation (objects generation, collision detection, shadow generation, fog, rain, wind), animation quality, different types of light sources (global, local, spotlights)

- **(1p)** documentation (mandatory)

Scene composition

The initial scene

When the application is run, two windows will pop up, one that contains the OpenGL application and a terminal. To use the app at full capabilities one should split screen because the terminal is somehow used to present the story behind the scene. For the first minute, each user will have a presentation of the scene, its capabilities and functionalities and after that the user will be able to control and traverse through the scene on his own.

The scene:

- ground component, from the lab 😊
- scene component: the most complex one. It contains every static, non animated object that composes the scene: the 4 groups of houses, the main road, the Welcome signs, the surrounding fence, the trees
- the business man, looking to buy a haunted house
- the two cars performing dangerous take overs

Scenario

As stated before, after the initial scene is presented via a custom animation the user will be able to dwell around the scene. Take a closer look to the old haunted houses, analyze the creepy salesman, supervise the traffic on the main road, rest in the shadows of the trees and much more.

Moreover, the user will be able to observe just the wireframes of the objects, the polygonal and smooth surfaces as well as the solid objects present in the scene.

For this to happen, the user must follow the instructions, thus use the W,A,S,D keys to move forwards/backwards/left/right, the keys Q,E to rotate the entire scene, the keys Z,X,C to change the mode the objects are presented.

Implementation

This sort of applications is extremely delicate and for those reasons they take a lot of time. Usually, for a really ambitious program, a team will be required since there are many aspects that need to be taken into account. For my project, there is a lot to improve. Mostly, I would have loved to have more complex animations. Also, the free objects found on the internet, are not the best in terms of quality of the form and textures. Also, if there would have been a larger variety of free objects online, the scene could have been much more complex.

In terms of pure implementation I have followed multiple tutorials on OpenGL, from the course website, from the LearnOpenGL website and from multiple YouTube tutorials.

Graphical Model

The graphical model used is the one that has been covered in the laboratories and is base on ShadowMapping model, Phong model and basic light dispersion components: diffuse light, Ambiental light and specular light.

Data Structures

For the calculations performed during operation of the project I have used data structures already available in the GLM library, ex: vec2, vec3, mat3, mat4, as well as OpenGL based data structures such as GLfloat, GLuint etc.

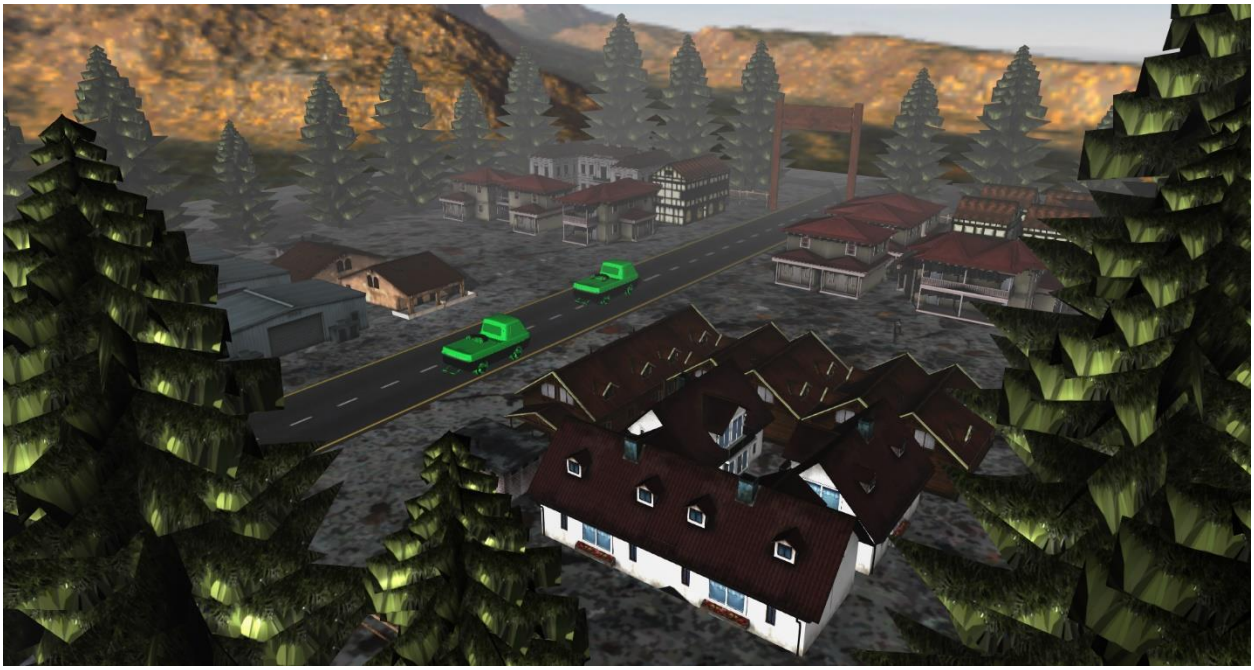
Class Hierarchy

For this project we had to use some libraries as well as user defined methods. For this we have included multiple headers files like:

Pentru a implementa functionalitatile implementate anterior, a fost necesara includerea de headere si .cpp-uri precum:

- Camera.hpp – camera control
- Mesh.hpp – object creation details
- Model3D.hpp – adding objects to scene
- Shader.hpp – adding shaders
- SkyBox.hpp – adding a skybox
- tiny_obj_loader
- GLEW.h si GLFW.h – OpenGL specific libraries

A final look at the chosen scene:



Using the Z,X,C keys one can change the mode the texture is displayed in the scene.

Loading an object in the scene

```
//create model matrix for businessman
	model = glm::mat4(1.0f);
	model = glm::translate(glm::mat4(1.0f), glm::vec3(0.0f + businessMan, 0.0f,
0.0f));
	if (businessMan < 0) {
		left = true;
	}
	if (businessMan > 1.837) {
		left = false;
	}
	if (left) businessMan += 0.005;
	else businessMan -= 0.005;
	//send model matrix data to shader
	glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));

	//compute normal matrix
	normalMatrix = glm::mat3(glm::inverseTranspose(view*model));
	//send normal matrix data to shader
	glUniformMatrix3fv(normalMatrixLoc, 1, GL_FALSE, glm::value_ptr(normalMatrix));

	man1.Draw(myCustomShader);
```

Loading the shadow for the business man:

```
	//create model matrix for man
	model = glm::mat4(1.0f);
	model = glm::translate(glm::mat4(1.0f), glm::vec3(0.0f + businessMan, 0.0f,
0.0f));
	if (businessMan < 0) {
		left = true;
	}
	if (businessMan > 1.837) {
		left = false;
	}
	if (left) businessMan += 0.005;
	else businessMan -= 0.005;

	//send model matrix to shader
	glUniformMatrix4fv(glGetUniformLocation(depthMapShader.shaderProgram, "model"),
		1,
		GL_FALSE,
		glm::value_ptr(model));

	man1.Draw(depthMapShader);
```

This algorithm applies to all the objects in the scene.

For the skyBox load we used:

```
mySkyBox.Load(faces);
skyboxShader.loadShader("shaders/skyboxShader.vert", "shaders/skyboxShader.frag");
skyboxShader.useShaderProgram();
```

And then:

```
mySkyBox.Load(faces);

skyboxShader.loadShader("shaders/skyboxShader.vert", "shaders/skyboxShader.frag");
skyboxShader.useShaderProgram();
```

where:

```
faces.push_back("skybox/right.tga");
faces.push_back("skybox/left.tga");
faces.push_back("skybox/top.tga");
faces.push_back("skybox/bottom.tga");
faces.push_back("skybox/back.tga");
faces.push_back("skybox/front.tga");
```

for which:

```
std::vector<const GLchar*> faces;
gps::SkyBox mySkyBox;
```

Below you can see the implementation of the camera movement using the predefined mouseCallback:

```
void mouseCallback(GLFWwindow* window, double xpos, double ypos)
{
	if (firstMouse) {
		lastX = 320;
		lastY = 240;
		firstMouse = false;
	}
	float xoffset = xpos - lastX;
	float yoffset = lastY - ypos;
	lastX = xpos;
	lastY = ypos;
	float sensitivity = 0.3;
	xoffset *= sensitivity;
	yoffset *= sensitivity;
	yaw += xoffset;
	pitch += yoffset;
	if (pitch > 89.0f)
		pitch = 89.0f;
	if (pitch < -89.0f)
		pitch = -89.0f;
	myCamera.rotate(pitch, yaw);
}
```

Where:

```
void Camera::rotate(float pitch, float yaw)
{
		glm::vec3 buff;
		buff.x = cos(glm::radians(yaw)) * cos(glm::radians(pitch));
		buff.y = sin(glm::radians(pitch));
		buff.z = sin(glm::radians(yaw)) * cos(glm::radians(pitch));
		cameraDirection = glm::normalize(buff);
 }
```

We used predefined functions to load the objects used as well as the shaders:

```
void initModels()
{
	myModel = gps::Model3D("objects/gps/v19.obj", "objects/gps/");
	car1 = gps::Model3D("objects/gps/e100.obj", "objects/gps/");
	car2 = gps::Model3D("objects/gps/e200.obj", "objects/gps/");
	man1 = gps::Model3D("objects/gps/GTP_BMan_Jack_07_Stg_Lsn_Adl_Ccs_Gry_Mgr.obj",
"objects/gps/");
	ground = gps::Model3D("objects/ground/ground.obj", "objects/ground/");
	lightCube = gps::Model3D("objects/cube/cube.obj", "objects/cube/");


	faces.push_back("skybox/right.tga");
	faces.push_back("skybox/left.tga");
	faces.push_back("skybox/top.tga");
	faces.push_back("skybox/bottom.tga");
	faces.push_back("skybox/back.tga");
	faces.push_back("skybox/front.tga");


}

void initShaders()
{
	mySkyBox.Load(faces);
	skyboxShader.loadShader("shaders/skyboxShader.vert", "shaders/skyboxShader.frag");
	skyboxShader.useShaderProgram();

	myCustomShader.loadShader("shaders/shaderStart.vert", "shaders/shaderStart.frag");
	lightShader.loadShader("shaders/lightCube.vert", "shaders/lightCube.frag");
	depthMapShader.loadShader("shaders/simpleDepthMap.vert",
"shaders/simpleDepthMap.frag");
}
```

Functionalities

Keyboard interactions:

- “w” – forward
- “s” – backwards
- “a” – left
- “d” – right
- “mouse” – camera rotation
- “q” – left rotation
- “e” – right rotation
- “z,x,c” – texture change
- “j” – left light source rotation
- “l” – right light source rotation

Conclusions

To summarize, although the project was not easy and very time consuming, it was fun, because after a little while until you get used to the tools, it`s very satisfying to see almost instant results. For this reason, and because of the time allocated I feel like I have improved my skills in C++, developed basic skills working with OpenGL and Blender.

This project made me appreciate more the games, even the bad and old ones, because I know how much effort I put into this project and still it is nowhere close to those ones.

Further improvements will most likely start with some cash for better objects, maximizing the scene, adding new objects or animations, and also interactions between these. Also, performance is another aspect that must be taken into account when further developing, since VRAM memory is reduced and expensive.

References:

https://www.youtube.com/

https://docs.google.com/document/d/1njtWPMmOQNIaD_z9ve8iPRUqQTWdIV_PO-NvPD0nOuM/edit

http://glm.g-truc.net/0.9.8/index.html

https://learnopengl.com/

http://www.free3d.com/

https://www.turbosquid.com/

https://moodle.cs.utcluj.ro/mod/page/view.php?id=11922

https://classes.soe.ucsc.edu/cmps161/Winter09/projects/mang/particles.cc

http://cgis.utcluj.ro/teaching/course/view.php?id=6/