# Computer Graphics Project

## 1 Introduction

This project aims to design an interactive 3D scene consisting of a small cottage with terrace. There are many objects loaded in the scene which are covered in textures and some of them cast shadows too. The users can move the camera in multiple positions to change the viewpoint by using keyboard input. Furthermore, some light sources were added in the scene which can be turned on/off at the press of a button. The simulation of natural phenomena is also present in this project - I created rain particles and fog. Last but not least, there is a race car track set on the table found on the terrace and the users can play with it by driving the small car.

Implementation wise I chose OpenGL, since I studied it during my bachelor degree and have more experience with it. To make things easier, I attached an .exe file near the source code so that the users who want to run the application can open it directly without the need to recompile the whole solution.

## 2 Implementation details

### 2.1 Scene and objects description

As can be seen in Figure 1, the scene consists of a small cottage with a terrace located somewhere in the desert. On the terrace there can be found multiple 3D objects: for instance chairs or lamps (on top), wooden table with race car track (center) and a bench or a teddy bear (on bottom). There are also some cactuses outside the house and an animated windmill.

Three skyboxes were used in order to create this landscape: one for the overall scene (with the ground plane and the sky), another one for the house (with front, back, left and right sides + roof) and ultimately a skybox for the terrace (with the wooden floor and surrounding fence).

As we get closer to the scene, the light intensity decreases and everything becomes darker, as shown in Figures 2 and 3. Additionally, some rain particles can be observed in Figure 2.

Figure 3 illustrates how the scene looks like when the user zooms it in. One may notice that the terrace is now surrounded in complete darkness and the user needs to turn on the light sources for brightening up the nocturnal scene. In Figure 3, one light source is enabled (top right lamp), whereas in Figure 4 all three light sources are enabled.

Ultimately, Figure 5 presents the toy car found on the wooden table which the users can manipulate.

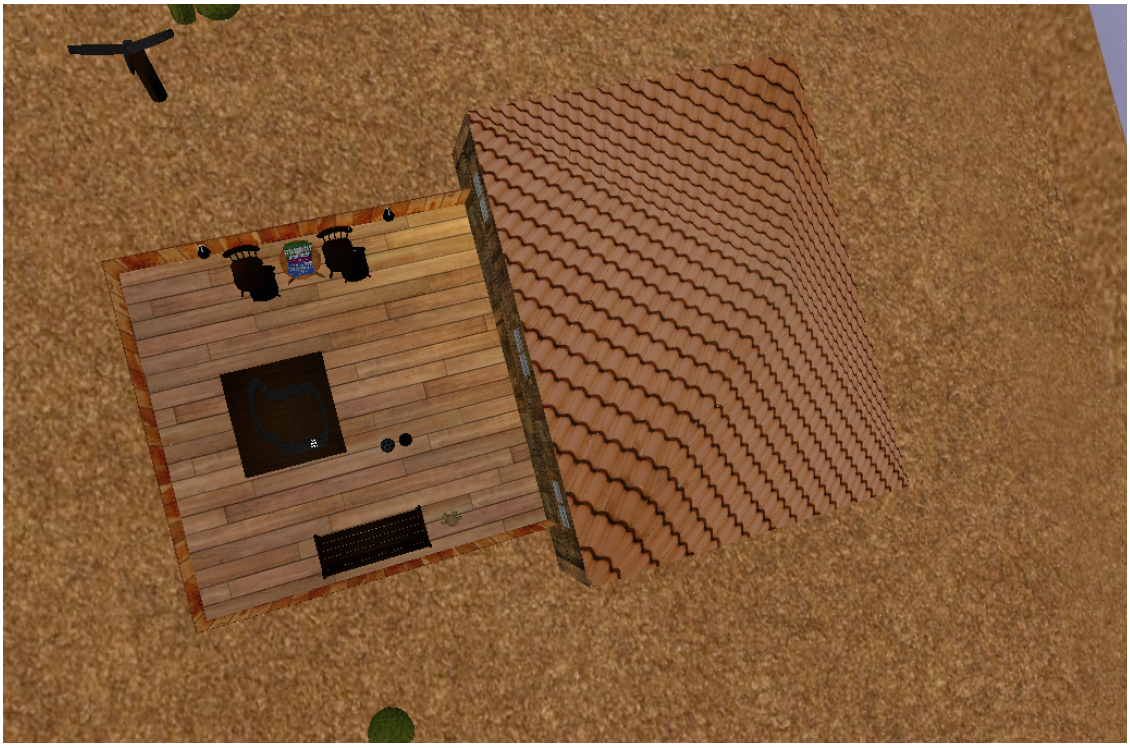

Figure 1: Aerial view of the scene

## 2.2 Functionalities

Some of the main functionalities found within this application will be detailed next. I will also describe shortly how I managed to implement each feature in OpenGL.

- Import 3D objects - I downloaded .obj models from the internet and used them in my application. The implementation here was pretty straightforward: I first set the correct position where I wanted to place the objects in the scene (translate), rotate or scale them if it was the case, apply a texture on their surface and then draw them. As a side note, I chose low poly objects for achieving a higher speed.

  As an example, for loading the car object I did the following:

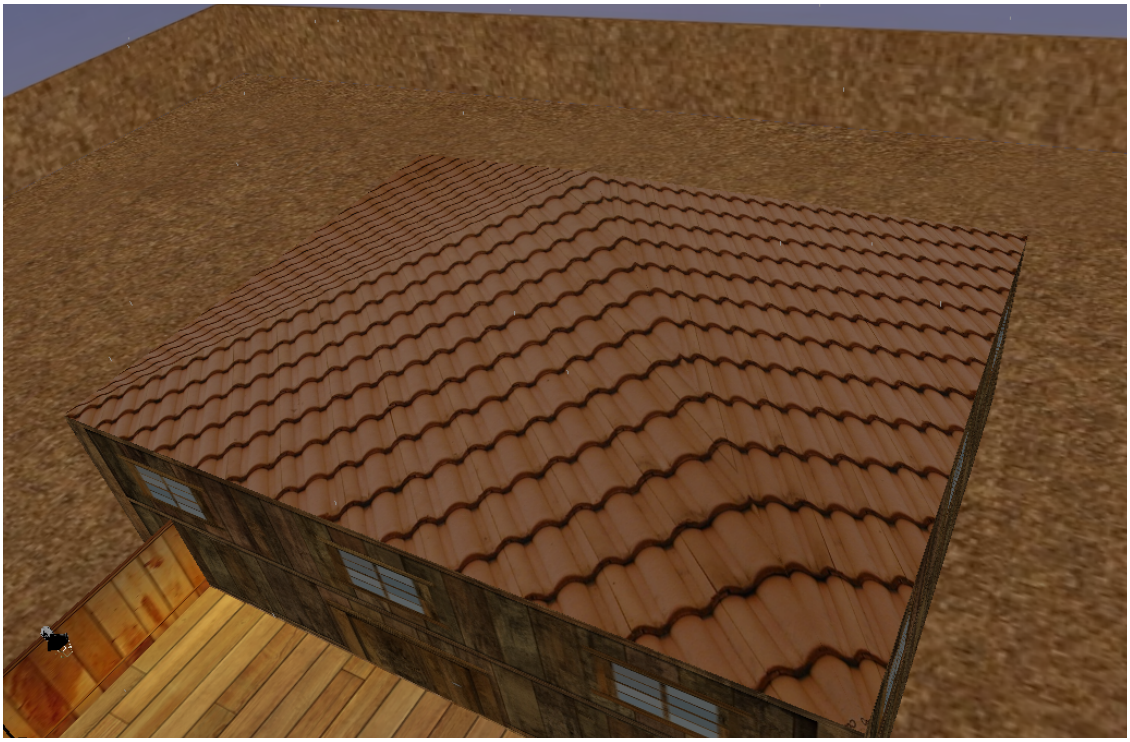

Figure 2: Getting closer

```
GLMmodel *car;
void drawCar() {
    glPushMatrix(); // start current object transformations
        glEnable(GL_TEXTURE_2D);
        glTranslatef(xCar, 25.5, zCar); // set car position
        glRotatef(-90, 1, 0, 0);
        glRotatef(angle2, 0, 0, 1);
        glBindTexture(GL_TEXTURE_2D, carTexture); // apply texture
        drawModel(&car, "OBJ/Car/car5.obj", mode); // draw car object
        glDisable(GL_TEXTURE_2D);
    glPopMatrix(); // end current object transformations
}
```

- Apply textures - I used only .tga textures which had the size equal to a power of two for a better performance. Setting a texture to an object was done the following way:

```
GLuint carTexture; // variable used for the ID of the car texture
glGenTextures(1, &carTexture); // return 1 texture name in carTexture
loadTGA("Obj/Car/carTex.tga", carTexture); // load .tga texture
```

- Change camera position - I created a boolean value called "camera" which is set to

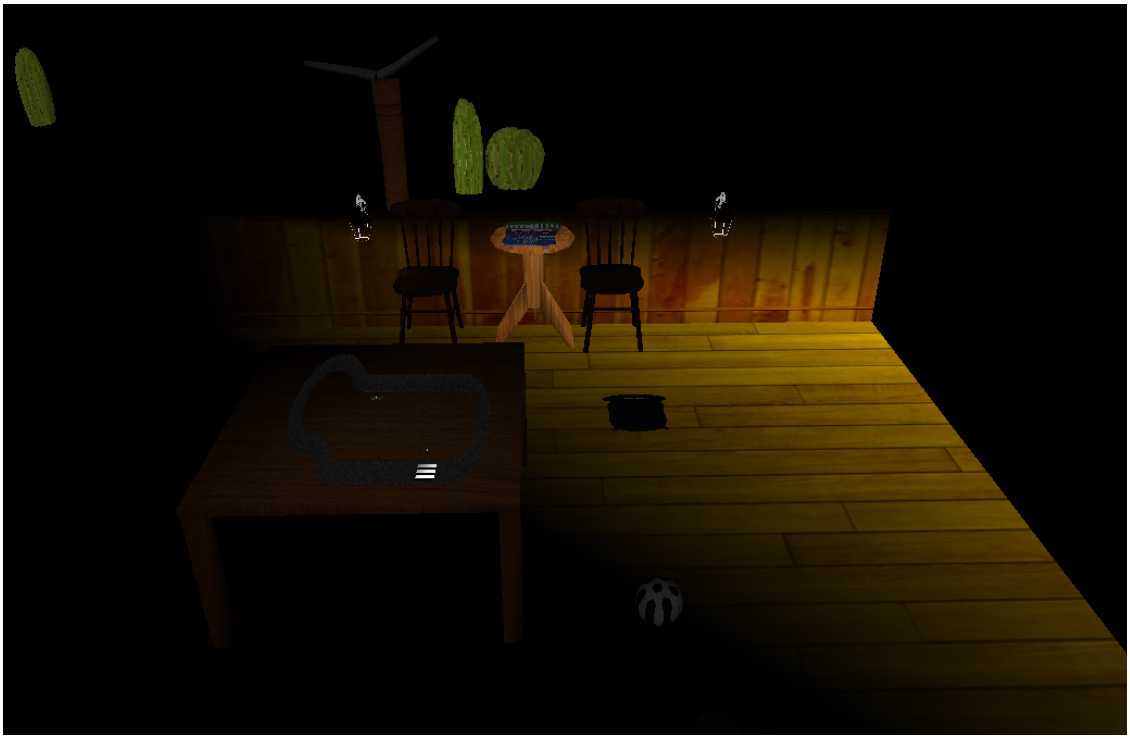

Figure 3: Night time

true when the user presses "c" and wants to focus on the race car found on the table. Whenever the "camera" variable is true, the user can move the car forwards, backwards, to the right and to the left by using W, A, S, D keys. Otherwise, when the camera doesn't focus on the car, pressing these keys will perform no action.

- Rain particles - A single rain particle has the following properties:

```
typedef struct {
    // Life
    bool alive;    // is the particle alive?
    float life;    // particle lifespan
    float fade; // decay
    // color
    float red;
    float green;
    float blue;
    // position/direction
    float xpos;
    float ypos;
    float zpos;
    float vel; // velocity/direction going in y direction
    float gravity; // gravity
}particles;
```

Figure 4: Light sources turned on

Each of these properties is initialized to a random value (apart from rgb values and gravity which are always the same) and each falling particle is drawn as a line segment.

- Fog effect - The fog is computed by using some helper functions found in OpenGL library.

```
void EnableFog() {
    glClearColor(fogColor[0], fogColor[1], fogColor[2], 1.0);
    glFogfv(GL_FOG_COLOR, fogColor);
    glFogi(GL_FOG_MODE, GL_EXP);
    glFogf(GL_FOG_DENSITY, fogdensity);
}
```

- Create animation - To animate the entire 3D scene from above, I simply incremented some variables (zoom and camZ), checking at the same time that the user's view "stays" in the skybox bounds.
- Switch between wireframe/solid objects - Here I used glPolygonMode function, with the second parameter set to GL_LINE for wireframes and GL_FILL for solid objects:

```
glPolygonMode(GL_FRONT_AND_BACK, GL_LINE;
```

- Shadows - I created shadows for 3 objects in the scene, namely the 2 chairs and the ball.

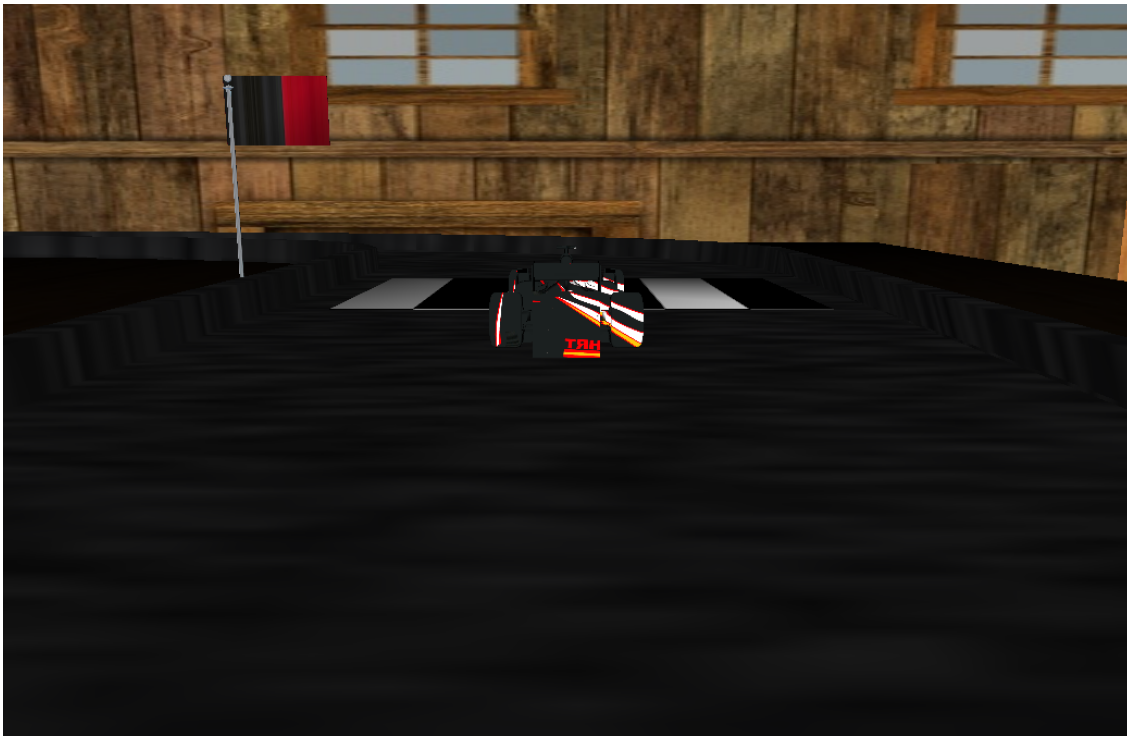

Figure 5: Driving the car

For achieving this, I used linear algebra equations to compute the shadow of a 3D object on a plane (for further details see the functions: PlaneEq, ComputePlaneEquations, ComputeShadowMatrices and ComputeShadowMatrix found in the source code).

## 2.3 User's manual

Since the users can interact with the scene through different keyboard inputs, I will enumerate each one below and briefly describe the action it performs.

- **c** - The camera viewpoint will change, focusing on the small race car found on the table (see Figure 5). The users will able to drive the car alongside the track by using the W, A, S, D keys to change directions. Pressing **c** the second time will revert the camera angle to the initial position (bird's eye view of the scene - similar to the one in Figure 4).
- **h, l** - Zoom in the scene.
- **j, k** - Zoom out the scene.
- **f** - Enable fog.
- **o** - Increase fog density.
- **p** - Decrease fog density.

- **v** - Enable toy car lighting.
- **1** - Disable all light sources.
- **2** - Turn on/off left lamp.
- **3** - Turn on/off right lamp.
- **4** - Change view to wireframe.
- **5** - Start rain.
- **6** - Animate the scene from above.
- **7** - Add shadows to both chairs and football object.

# 3 Conclusion

Developing this application was a very effective way by which I was able to expand my knowledge in Computer Graphics. However there are still multiple improvements that can be added to increase the realism and complexity of the scene, such as: making it possible to enter the house, adding more animations, using high poly objects or improving the shadows.