Computer Graphics Report

Ernests Kuznecovs - 17332791

05/01/2021

- Demo link: `https://www.youtube.com/watch?v=4ZjR1sLL12Q`

Midterm Summary

In the miderm, the features that had been implemented were:

- Camera Looking
- Camera Movement
- Model Loading
- Texture Loading (via assimp)
- Mesh Hierarcy

Many of the features above derived from LearnOpenGL.com as a resource to learn from.

Final Project Summary

The additional features that had been implemented since then are:

- Crowd of spiders
- Multiple light sources
- A controllable dynamic light
- Texture on the plane
- Pause/Play feature on the crowd
- Ability to show hitbox of each of the spiders
- Abilty to aim at hitbox and detect intersection with the camera view direction
- Ability to select the spider that is being aimed at
- Ability to move selected spider around

The rest of the report will describe these features in additional detail.

Crowd of Spiders

The crowd is represented as a vector of Model objects. Model objects store mesh data, hierarchy structure, and a drawing function. Models also store their position and rotation in the world as vec2 and flaot that is then used to create the translation matrices (done outside the Model class for quick prototyping)

Using world positions.

The Model class provides functions to update these world positions:

```
void move_forward(float distance) {
  float x = cos(forward_direction_rad) * distance;
  float z = sin(forward_direction_rad) * distance;
  pos.x = pos.x + x;
  pos.y = pos.y + z;
}

void rotate(float rad) {
  forward_direction_rad += rad;
}
```

Using the positions to calculate transformation matrices:

```
auto& spider = crowd[i];
auto translation = glm::translate(glm::mat4(1.0f),
   glm::vec3(-spider.pos.x, 0, -spider.pos.y));
model = glm::rotate(translation,
    -spider.forward_direction_rad,
    glm::vec3(0.0f, 1.0f, 0.0f));
```

Managing crowd datastructure

The crowd is initialised with the same spider model but at different starting positions. In the drawing loop, there is one function that animates the spiders legs via the hierarchy and updates its world position, and another function that applies the transformations and draws them.

e.g animations and world position updates

```
void crowd_iteration(vector<Model>& crowd, Shader& spider_shader) {
  for (int i = 0; i < crowd.size(); i++){
```

```
    auto& spider_model = crowd[i];
    float rotation = i % 2 == 0 ? -0.01 : 0.01;
    spider_model.rotate(rotation * (i + 1));
    spider_model.move_forward(0.055 * i + 0.01);
    animate_spider(spider_model.hierarchy);
  }
}
```

Dynamic Light Source

The most obvious light source that is seen is the yellowish ball that can move around. The ball itself is a sphere that is loaded into the application. The model and the light source share the same coordinate (the ball is translated to the position in the world to where the light source is).

The world space posisition of the light is controlled with the keyboard i,j,k,l.

The position of the light, and camera position is passed to the fragment shader in order to calculate the light that changes position and where the camera changes. e.g calculating power of specular light:

```
vec3 light_direction = normalize(light_position - vertex_world);
vec3 view_direction = normalize(vertex_world - view_position);
vec3 reflection_direction = normalize(reflect(light_direction, tnorm));
float specular_dot_product = max(dot(view_direction, reflection_direction), 0.0);
```

Multple Light Sources

The light sources that are implemented are just ambient + diffuse + specular. The first light source was the dynamic one, but the second one is just a blue point light, 2 units above the origin emmiting blue light.

The light is simply implemented in the shaders themselves for the reason of protyping, but for a biffer project I would pass the multiple light sources as an array of light source parameters for the shader to apply.

Texture on the Plane

The texture on the ground is a png of a stone wall with the $\mathrm{GL}_{\mathrm{REPEAT}}$ option. The texutre coordinates were picked by hand and an indexed array was used to specify which order to apply the texture coordinates for the shaders.

Pause/Play Feature on the Crowd

This feature was implemented as it would be useful to have in order to debug future features that would be added, such as boids.

There is a boolean switch in the global scope of the application. A key is bound to toggle the value of the boolean.

The boolean is used in the drawing loop to decide whether or not skip a crowd iteration.

```
if (next_state){
  crowd_iteration(crowd, spider_shader);
}
draw_crowd(crowd, spider_shader,camera, sphere_shader);
```

Just having the $\text{crowd}_{\text{iteration}}$ in the boolean allows for camera movements and any other additional control features to still work.

Show Hitbox

This is just a unit sphere drawn in wireframe.

The spiders world position is used to determine where to translate the sphere for rendering.

A boolean switch exists that determines if the spheres should be drawn.

Aim at Hitbox

This feature uses some of the mathematic of ray tracing. The camera position, direction, and position of the sphere are used to calculate if there is an intersection.

```
bool camera_intersects_sphere(glm::vec3 cam_pos, glm::vec3 sphere_pos,
 glm::vec3 cam_direction) {
  float b = 2 * cam_direction.x * (cam_pos.x - sphere_pos.x) +
    2 * cam_direction.z * (cam_pos.z - sphere_pos.z) +
    2 * cam_direction.y * (cam_pos.y - sphere_pos.y);
  float c = (cam_pos.x - sphere_pos.x) * (cam_pos.x - sphere_pos.x) +
    (cam_pos.y - sphere_pos.y) * (cam_pos.y - sphere_pos.y) +
    (cam_pos.z - sphere_pos.z) * (cam_pos.z - sphere_pos.z) - 1;

  float d = b * b - 4 * c;
  return d >= 0;
}
```

Select Spider

A key can be pressed to lock in the aimed-at spider. This changes the colour of the hitbox to red to show that its locked.

All of this functionality is controlled with 2 variables, one for keeping which spider is aimed at, and one for determining if the lock has been activated.

```
if (intersected && !aiming_lock)
  aiming_at = i;
if (aiming_at == i)
  sphere_shader.setVec3("ball_colour", glm::vec3(0.3, 0.8, 0.3));
if (aiming_at == i && aiming_lock)
  sphere_shader.setVec3("ball_colour", glm::vec3(0.9, 0.2, 0.0));
```

Move Spider

Once a spider is selected, it can be moved.

Within the key processing, the selected spiders world positions is incremented/decremented.

```
if (glfwGetKey(window, GLFW_KEY_E) == GLFW_PRESS && aiming_lock == true)
  crowd[aiming_at].move_forward(-0.1);
if (glfwGetKey(window, GLFW_KEY_U) == GLFW_PRESS && aiming_lock == true)
  crowd[aiming_at].rotate(0.1);
```