Project report for the CG 100433 course

——Tongji Mapping Game

Team members

第 6 组	
1652526	谢楚阳
1751876	肖金雨
1754244	郭维康
1759266	Sophie
1753844	房洛成

1. Motivation

Today’s most large-scale event venues, such as shopping malls, parks, schools, etc., mostly use 2D flat maps in their guide. Reading such a 2D map is quite unintuitive and inconvenient, since some personnal efforts and skills are required to identify buildings, roads and other key elements in the map. We hope we can help to bring a new and simpler way for people to get familiar with unknown places, without the need to read 2D maps anymore.

Computer Graphics has some advantages in the purpose of reproducing 3D real scenes. Based on what we learned during class and some background requirements, we managed to design a 3D map exploration game. We chosed the campus of our school as an exploration ground, in which the user is given a main character perspective to travel and discover the scene map.

At the same time, Tongji's 3D map also helps freshmen to become familiar with every corner of the campus. If it eventually becomes even more beautiful and realistic in the subsequent improvements, then maybe in the future it can also achieve the role of promoting our beautiful campus.

2. Goal of the project

Considering the personal capabilities of each member of the group and the realizability of the project, we fixed a few target objectives in order to demonstrate the skills we developped with the Computer Graphics class.

2.1 First Goals

- Complete the construction of basic models of all landmark buildings, roads, and shops in a specific area of the school to realize the simulation of the 3D campus scene. (completed)
- Observe the user from a first-person perspective in the view the map scene, so that the user's view can rise and fall, rotate left and right, and move forward and backward. (completed)

2.2 Extensions

- The user can select a specific building's name as target destination for the simulation to generate a red path to it. (completed)
- Add some random moving passerby models in the simulation to increase realism of the scenes. (completed)
- Provide the user the ability to modify specific target models in the map, such as moving or destruct objects like trees and trash cans. (undone)

2.3 Others(Other features implemented after)

- A variety of lighting are implemented and can be controlled through the user's interaction with keyboard
- The skybox was added
- Real physics engine (collision detection, jumping, etc.)

3. Scope of the project

The scope of project's areas and features is defined as follows:

- Because the entire Jiading campus area of Tongji University is too large for computation to be simulated, we have limited the scope of the selected scene. After selecting the area to be modeled (the teaching buildings region), it was decided that Tongji University overlooked by a starry sky.
- In order to reduce the modeling workload, the regions of space that cannot be seen by the user from every possible positions on the roads are reduced in accuracy or even not modeled at all. (Moreover: Due to the limited loading capacity of computer hardware, models created using modeling software such as sketchup are intentionally not particularly detailed.)
- The details and resemblance of the models to the real life buildings are not too high, but the actual locations are consistents to the real scene, and the buildings are properly scaled. Those properties also apply to the roads.

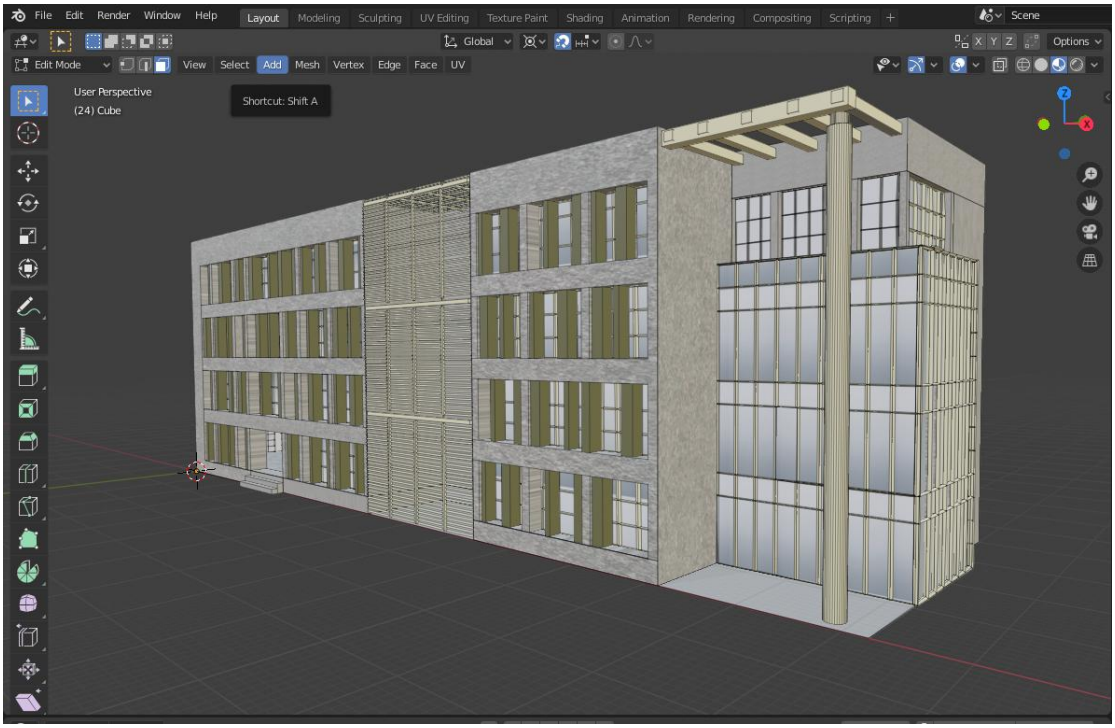
4. Involved CG techniques

Design plan: build the 3D map's models in a modeling software → import the map's models into OPENGL → realize the user's browsings by adjusting the position of the camera → add auxiliary functions such as skybox and lightings.

4.1 Scene modelings

Building's and other objects are modelled and exported into model files from *sketchup* and *Blender*. The buildings were divided among the team members to be modelled. Based on the shape of the buildings (whose geometry is pretty basic), it sufficed to add adjust vertices positions and add edges to get the physical characteristics of the buildings. After the shapes were modeled, we further optimized the models by adding texture maps. The other objects such as flowers or trees can be directly imported from the network. Finally, the buildings are combined together as they will be in the 3D map, according to their actual location, and the file of the combined

buildings can be exported after this final integration. The exported model file automatically includes all the generated vertices, their coordinates, normals, texture coords, and other secondary informations.



Note : because of OS compatibility, *blender* was used to model some of the buildings. However, we found some difficulties to render those models in the same way they are in *blender*

4.2 Model Loading

The models are loaded using the models importing library, *Assimp*. We first needed to install and compile Assimp. https://www.jianshu.com/p/4f3a1271ce0b is an online tutorial that helped to do the loading operation. Before importing the model, the C++ *Model* and *Mesh* classes need to be coded in order to be able to load and use data structures in Assimp. When drawing the models, we need to render the independent meshes that make up the

model, instead of rendering the entire model as a whole. After the definition of the loading's structure, Assimp's *loadModel* functions can be used to import the model datas, get the indexes array to the model, get each array and process it, including vertex datas, indexes, material datas, and finally return to us the final structure.

(Note that the imported model's obj files need to be added an appropriate format to ensure that the rendering will be performed properly.)

```
Shader modelShader("../Shaders/model_loading.vs", "../Shaders/model_loading.fs");
//Shader groundShader("../Shaders/model_ground.vs", "../Shaders/model_ground.fs");
// 加载模型 Load Model
Model ourModel("../Objects/man/man.obj");//新的人物
Model ground("../Objects/ground/ground.obj");//场景模型
Model A("../Objects/A/A.obj");
Model B("../Objects/B/B.obj");
Model C("../Objects/C/C.obj");
Model F("../Objects/F/F.obj");
Model xingkong("../Objects/xingkong/xingkong.obj");
Model qiao("../Objects/qiao/qiao.obj");
Model cat("../Objects/cat/cat.obj");
Model D("../Objects/D/D.obj");
Model H("../Objects/H/H.obj");
Model E("../Objects/E/E.obj");
Model G("../Objects/G/G.obj");
Model luzhang("../Objects/luzhang/luzhang.obj");
Model lamp("../Objects/lamp/lamp.obj");
Model bicycle("../Objects/bicycle/bicycle.obj");
```

Make a call to the *Model* class constructor to read the corresponding .obj file

```
modelShader.use();

modelShader.setMat4("projection", projection);
modelShader.setMat4("view", view);

glm::mat4 model = glm::mat4(1.0f);
model = glm::translate(model, camera.Target); // translate i
model = glm::rotate(model, glm::radians(-camera.Yaw + 90), g
model = glm::scale(model, glm::vec3(0.15f, 0.15f, 0.15f));
modelShader.setMat4("model", model);
ourModel.Draw(modelShader);

model = glm::mat4(1.0f);
model = glm::translate(model, glm::vec3(0.0f, 0.0f, 0.0f));
model = glm::scale(model, glm::vec3(1.0f, 1.0f, 1.0f)); // i
modelShader.setMat4("model", model);
ground.Draw(modelShader);
```

Instantiate the model Shader, and render the model

4.3 Camera and Main Character

Keys W-A-S-D permit to control the position of camera. The camera moves back and forth, left and right, and the mouse's roll controls the zooming.

The camera variable is associated with the input of the W-A-S-D keys, and the input value is immediately fed into the *lookAt* function to change the value of the view. In addition, we define global variables to record the

current run time of the simulation to ensure that the frames are refreshed according to the running time, so the running speed is independent to the system on which it runs.

Mouse rotation changes the angle of view and involves knowledge of Euler angles. Pitch means that the camera is rotated upwards and downwards, yaw means left and right, and roll means scroll. There is no need to implement roll here. When the mouse moves, the horizontal and vertical movements represent respectively the values of yaw and pitch. To compute the rotation angle, we need to know the difference to the current mouse position, so we continuously store the last position of the mouse. The offset value is then added to the camera's yaw and pitch values. We set some pertinent maximum and minimum possible values for pitch and yaw, to prevent excessive rotation.

In order to offer a good game player experience, we bind the position and the main character together. In other words, the camera ‘s position is always at the back of the main character and the camera initially looks down at the role and the angle of the view can be changed through the mouse later.

Since we have successfully connect the camera and the hero, now keys W-A-S-D permit to control not only the position of the camera but also the position of the hero.

4.4 Skybox and Lighting

The skybox is a particular case of a texture box, that is, six texture maps with different orientations are loaded to the corresponding faces of the texture box. In the process of computing the camera’s transformations in the previous step, we determined the range of positions that the camera can take to prevent it from moving outside the skybox.

Lighting – the effects achieved include:

- Global Directional lighting
- Point lights implemented as street lights
- Spotlight: the player can use flashlight
- Multi-lights effects
- Different materials reflection properties
- Bicycle’s metal effect
- Day and night switching effect

4.4.1 Point Light implementation

- **Code Implementation**

```
struct PointLight {
    vec3 position;

    float constant;
    float linear;
    float quadratic;

    vec3 ambient;
    vec3 diffuse;
    vec3 specular;
};
```

```
// calculates the color when using a point light.
vec3 CalPointLight(PointLight light, vec3 normal, vec3 fragPos, vec3 viewDir)
{
    vec3 lightDir = normalize(light.position - fragPos);
    // diffuse shading
    float diff = max(dot(normal, lightDir), 0.0);
    // specular shading
    vec3 reflectDir = reflect(-lightDir, normal);
    float spec = pow(max(dot(viewDir, reflectDir), 0.0), material.shininess);
    // attenuation
    float distance = length(light.position - fragPos);
    float attenuation = 1.0 / (light.constant + light.linear * distance +
light.quadratic * (distance * distance));
    // combine results
    vec3 ambient = light.ambient * vec3(texture(material.diffuse, TexCoords));
    vec3 diffuse = light.diffuse * diff * vec3(texture(material.diffuse, TexCoords));
    vec3 specular = light.specular * spec * vec3(texture(material.specular, TexCoords));
    ambient *= attenuation;
    diffuse *= attenuation;
    specular *= attenuation;
    return (ambient + diffuse + specular);
}
```

- **Point Light results**

Left : Day + Street lights on

Right : Night + Street lights on

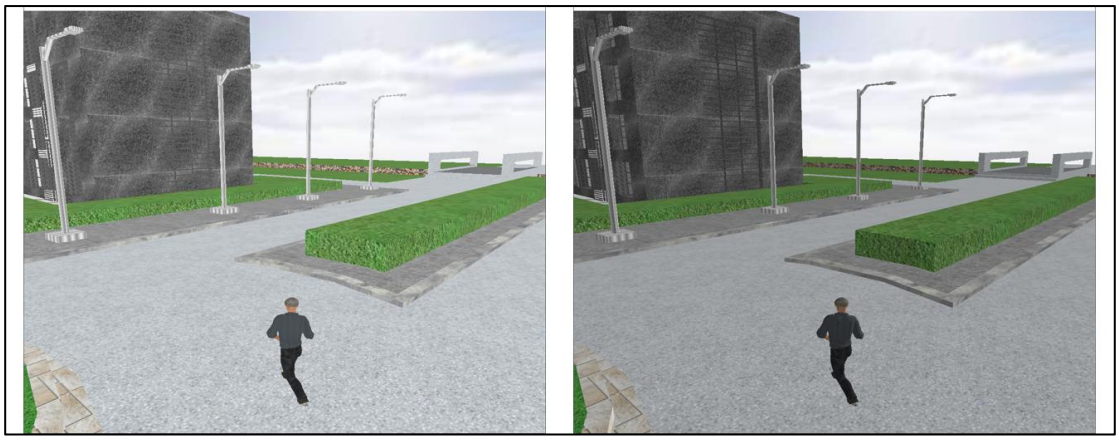
4.4.2 Directional Light implementation

- **Code Implementation**

```
struct DirLight {
    vec3 direction;
    vec3 ambient;
    vec3 diffuse;
    vec3 specular;
};
```

```
// calculates the color when using a directional light.
vec3 CalDirLight(DirLight light, vec3 normal, vec3 viewDir)
{
    vec3 lightDir = normalize(-light.direction);
    // diffuse shading
    float diff = max(dot(normal, lightDir), 0.0);
    // specular shading
    vec3 reflectDir = reflect(-lightDir, normal);
    float spec = pow(max(dot(viewDir, reflectDir), 0.0), material.shininess);
    // combine results
    vec3 ambient = light.ambient * vec3(texture(material.diffuse, TexCoords));
    vec3 diffuse = light.diffuse * diff * vec3(texture(material.diffuse, TexCoords));
    vec3 specular = light.specular * spec * vec3(texture(material.specular, TexCoords));
    return (ambient + diffuse + specular);
}
```

- **Directional Light results**



Left : Directional light is on

Right : Directional light is off

4.4.3 Spot Light implementation

- **Code Implementation**

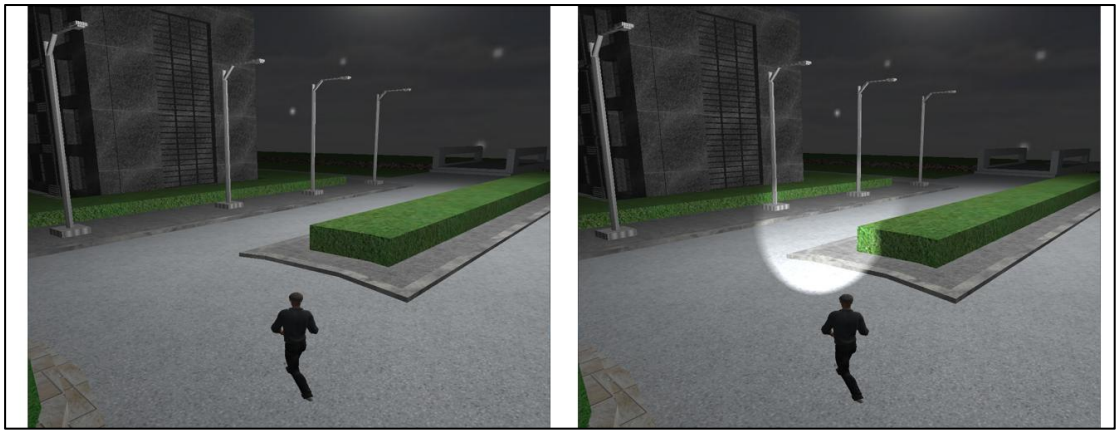
```
struct SpotLight {
    vec3 position;
    vec3 direction;
    float cutOff;
    float
outerCutOff;

    float constant;
    float linear;
    float quadratic;

    vec3 ambient;
    vec3 diffuse;
    vec3 specular;
};
```

```
vec3 CalSpotLight(SpotLight light, vec3 normal, vec3 fragPos, vec3 viewDir)
{
    vec3 lightDir = normalize(light.position - fragPos);
    // diffuse shading
    float diff = max(dot(normal, lightDir), 0.0);
    // specular shading
    vec3 reflectDir = reflect(-lightDir, normal);
    float spec = pow(max(dot(viewDir, reflectDir), 0.0), material.shininess);
    // attenuation
    float distance = length(light.position - fragPos);
    float attenuation = 1.0 / (light.constant + light.linear * distance +
light.quadratic * (distance * distance));
    // spotlight intensity
    float theta = dot(lightDir, normalize(-light.direction));
    float epsilon = light.cutOff - light.outerCutOff;
    float intensity = clamp((theta - light.outerCutOff) / epsilon, 0.0, 1.0);
    // combine results
    vec3 ambient = light.ambient * vec3(texture(material.diffuse, TexCoords));
    vec3 diffuse = light.diffuse * diff * vec3(texture(material.diffuse, TexCoords));
    vec3 specular = light.specular * spec * vec3(texture(material.specular, TexCoords));
    ambient *= attenuation * intensity;
    diffuse *= attenuation * intensity;
    specular *= attenuation * intensity;
    return (ambient + diffuse + specular);
```

- **Spot Light results**



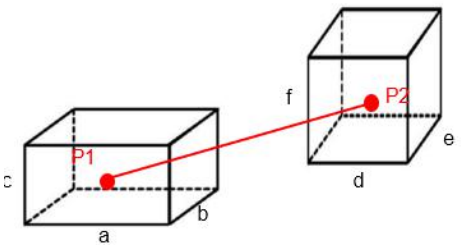
Left : spot light is off　　Right : spot light is on

4.5 Physics Engine (basic)

4.5.1 Collision detection

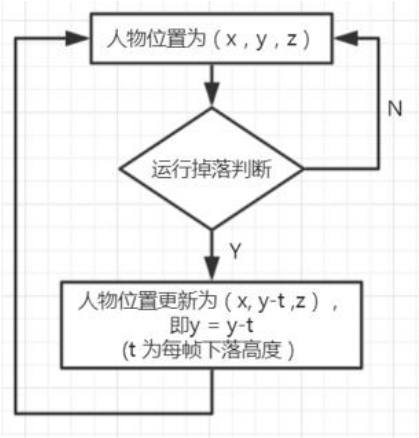
Detects whether the main character collides with other objects, including: collision between cuboids and collision between cuboid and cylinder.

- 已知每个碰撞盒的长宽高abc以及模型左下角坐标P1（x1,y1,z1）
- 已知人物在底部中心坐标P2（x2,y2,z2）以及人物碰撞盒长宽高def
- 对于所有模型
- |(x1+a/2)-(x2)|<(a+d)/2
- |(y1+b/2)-(z2)|<(b+e)/2
- |(y1+c/2)-(y2+f/2)|<(c+f)/2
- 三个条件同时达成即为长方体发生碰撞



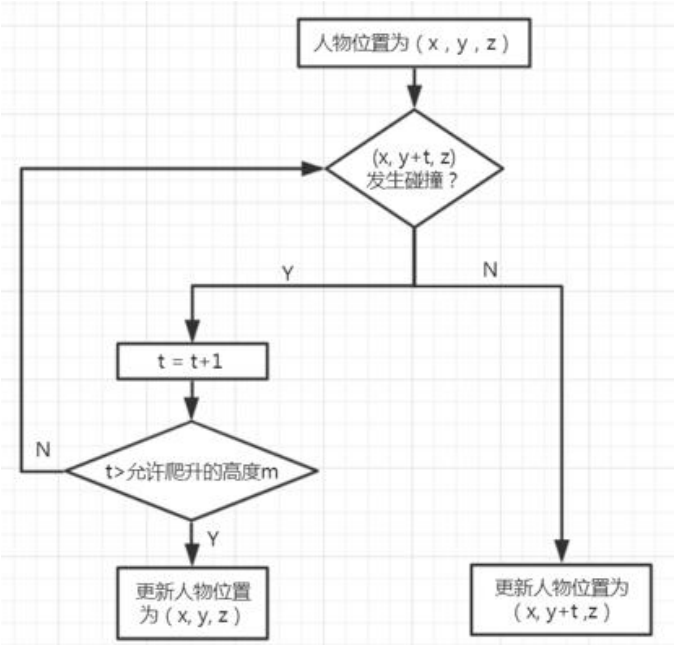
4.5.2 Gravity

Determine whether the model is in a falling state (it is so until it touches the ground's collision box)



4.5.3 Obstacle climbing

The character is allowed to jump over a collision box that doesn't exceed some maximum height.



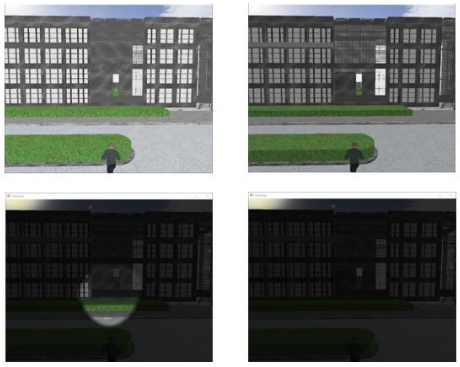
4.6 Material

- **Material structure Implementation**

```
struct Material {
    sampler2D diffuse;
    sampler2D specular;
    float shininess;
};
```

- **Material results**

Facade material

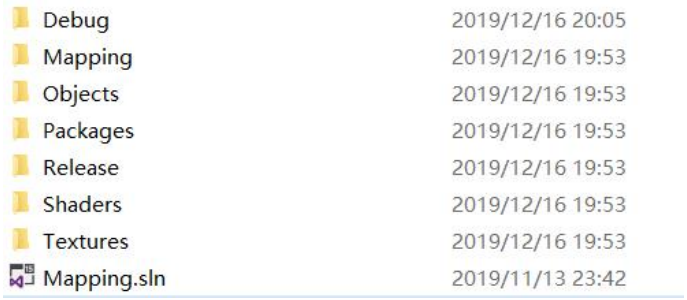


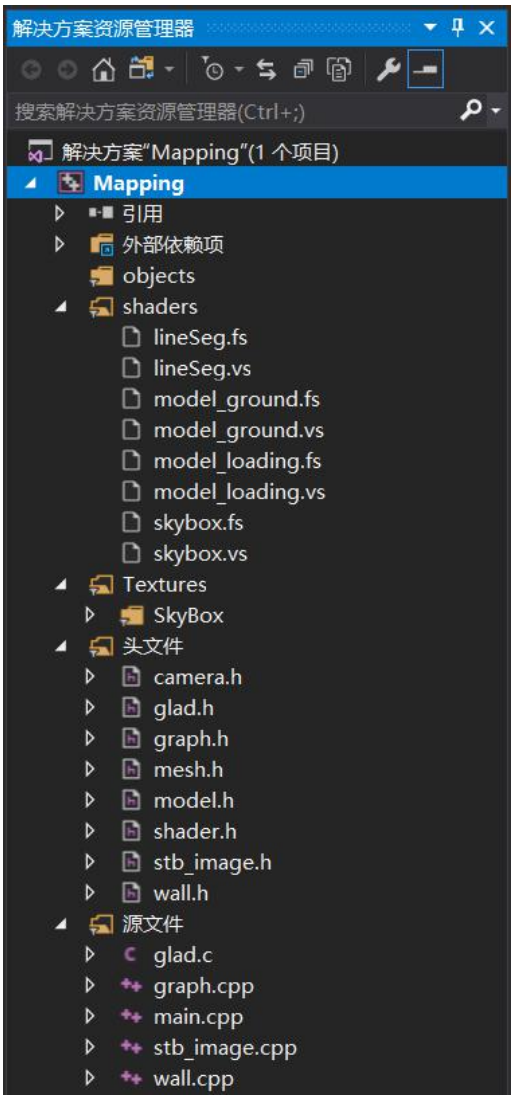
Lamp post

5. Project contents

The project mainly includes:

Objects and buildings modeling, models importation, design and implementation of physics engine, implementation of multiple light sources and materials, loading of textures such as skybox. And here is the structure of the project.





6. Implementation

Week	Missions
Week 9	Build the structure of the project code, establish the code communication platform between team members, and determine the scope of the project.
Week 10	Camera's position and main character binding, Perspective transformations with mouse. Main character behavior: basic movement with WSAD.
Week 11	Learn about library "Assimp" and the header file implementations. Build up basic model import environment. Skybox finished.
Week 12	Design the map of our scene, finish all models' coordinate calibration. Divide the whole team into two groups: model group and behavior group. Finish mid-term report.

Week 13-14	Model group: According to the map designed and the positions of every objects, building all models with Sketchup or Maya.
	Behavior group: Physics engine:jumping, falling, climbing, collision detection. Finish map path guidance. (to Building A) Lighting and materials effects implementation.
Week 15	Merge the work of two groups. Prepare for the final report.

7. Results


Mapping


Mapping

8. Roles in group

Member	Missions
谢楚阳 (Leader)	1. Build the structure of the project. 2. Camera behaviour and main character behaviour 3. Physics engine:jumping, falling, climbing, collision detection. 4. Map path guidance with Sophie. (to Building A)
房洛成	1. Skybox finished. 2. Lighting effects implementation. 3. Materials effects implementation.
郭维康	1. Leader of the Model group. 2. Design the map of our scene. 3. Building and importing all objects in the scene.
肖金雨	1. Build objects in the Model group. 2. Learn about library “Assimp” and the header file implementations and Build up basic model import environment. 3. Help finish all official reports.
Sophie (卓然)	1. Build objects in the Model group. 2. Help implement Map path guidance function. 3. Help finish English reports.

References

[1]邓军勇,李涛,蒋林,韩俊刚,沈绪榜.面向 OpenGL 的图形加速器设计与实现[J].西安电子科 技大学学报,2015,42(06):124-130.
[2]黎华,肖伟.几种三维模型文件在 OpenGL 中的输入与处理[J].物探化探计算技 术,2007(01):83-86+96.
[3]汤彬.基于 OpenGL 的纹理映射研究[J].实验室研究与探索,2006(05):576-579.
[4] https://learnopengl-cn.github.io/ LearnOpenGL-CN
[5] https://www.jianshu.com/p/4f3a1271ce0b/ Assimp 的安装编译及使用过程全纪录
[6] http://ogldev.atspace.co.uk/ OGLDev 现代 OpenGL 教程