北 京 林 业 大 学

2021 学年—2022 学年第一学期计算机图形学(双语)A 课程设计

实验报告书

专　　业：数字媒体技术

小组成员：数媒 191 陈秋莹 191002426

数媒 191 李晓敏 191002425

数媒 191 何京　191002209

数媒 192 张雪婷 191002725

数媒 192 简斯怡 191002419

实验地点：计算中心　　任课教师：曹卫群

实验题目：计算机图形学(双语)A 课程设计

实验环境：VC++, OpenGL

设计目标：

本项目实现了沉浸式海滩漫游以及贝壳知识科普，场景结合了声、光、影，使画面更加真实，向玩家传递了热爱自然、向往自由的理念。

团队任务分工：

小组成员共同搭建起游戏框架，讨论代码方案，分工明确，各司其职，最终实现了可观的效果。

导入模型：简斯怡、何京；
显示图鉴：李晓敏、张雪婷；
添加音频：简斯怡；
Shader 实现天空盒：陈秋莹、简斯怡、李晓敏；
灯光渲染：简斯怡、陈秋莹、何京；
碰撞检测：何京、李晓敏、张雪婷；
模板检测：李晓敏、张雪婷；
Shader 实现沙滩和海浪：陈秋莹；
Demo 渲染：何京。

实验内容：

主要构建了两大场景：

一是在夜晚的海面上有一尊女神像在静静的旋转，当打开手电筒向前推进时，女神像周围慢慢变亮，跳转到第二个场景。

二是在沙滩上，由海洋，沙滩，海边小屋，女神像和散落的贝壳组成。进入场景便可以听到海浪涌动的声音，可以通过键盘和鼠标进行漫游，当靠近贝壳时会出现提示，该贝壳会变大并出现轮廓，也可以选择展示当前贝壳的介绍图鉴；并且添加了碰撞检测，在靠近女神像一定距离时便不能继续向前移动。

实现方法：

一、模型导入

在 3dmax 中建模。因为 obj 类型的文件结构简单，文件体积小，而且能够导出材质文件，因此最终将模型导出为 obj 格式的文件。这里要注意我们在 3dmax 中建模的时候，不能采用 vray 材质，只能采用 3dmax 自带的标准材质，这样才能成功导出为 obj 类型的文件。在导出 obj 模型的时候，还需要注意一个模型坐标和世界坐标的问题，这个会影响到模型文件直接导入 OpenGL 后位于场景的位置。

模型文件的读入我们采用了 assimp 库，参考网上的代码，用了一个 model.h 和 mesh.h 文件来读入模型。具体代码可见工程文件。

二、真实感场景的交互展示

1.与贝壳交互

1）获取各个贝壳模型的位置，并设置一个包围球

2）实时监测当前摄像机的位置，并判断是否在包围球内

3）选中包围球内的模型中最近的模型，包括模型变大并显示物体轮廓；

4）检测是否按下左键，当按下左键时，获取当前最近物体的序号并根据序号弹出对应介绍

2.显示物体轮廓的实现方法：使用模板检测

1）在绘制当前物体之前，将模板函数设置为 GL_ALWAYS，每当物体的片段被渲染时，将模板缓冲更新为 1。

2）禁用模板写入以及深度测试。

3）将当前物体放大至开始的 1.5 倍，实现放大显示

4）使用一个不同的片段着色器，输出一个单独的（边框）颜色，设置为白色

5）再次绘制物体，在放大 1.5 倍的前提下再放大 1.2 倍，但只在它们片段的模板值不等于 1 时才绘制。

6）再次启用模板写入和深度测试。

3.显示介绍的实现方法：使用纹理显示：

1）定义顶点数组 vertex array

2）生成纹理: glGenTextures

3）绑定纹理: glBindTexture: 参数 GL_TEXTURE_2D 告诉 opengl，这应该被作为一个二维纹理对待

4）加载纹理到 Opengl: texImage2D,把位图数据复制到当前绑定的纹理对象

5）增加 vertex shader: 顶点坐标（-1 到 1 之间），纹理坐标（0 到 1 之间，S 坐标和 T 坐标，二维数组）

6）增加 fragment shader：着色：gl_FragColor = texture2D（纹理单元 ID, 纹理坐标 ）：读出纹理中那个纹理坐标处的颜色值

7）将顶点和纹理坐标数组分别绑定到 vertex shader 和 fragment shader 中去

8）激活纹理单元 0（glActiveTexture），并把纹理绑定到这个纹理单元 0(glBindTexture) ， 告 诉 片 段 着 色 器 使 用 纹 理 单 元 0（glUniform1i(uTexture->UnitLocation, 0)）

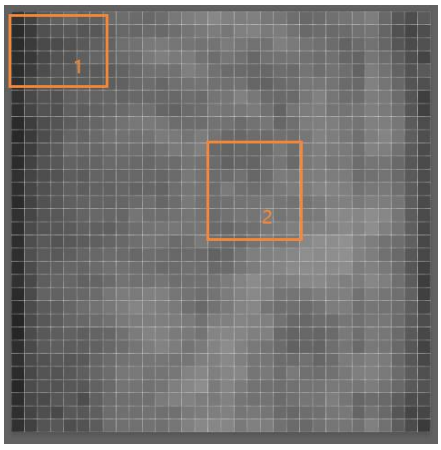
9）渲染：glDrawArrays

三、三维场景沙滩海浪生成

1、思路：

由于大海和沙滩面积较大，导入模型会使程序卡顿，并且还需要动态的海浪，我们最终采用了 shader 绘制场景。

沙滩地形的特点是凹凸不平，也就是 Y 坐标随机。为了实现自然的起伏效果，我们绘制了一个灰度地形图，如下所示。



在 RGB 颜色空间中，(0,0,0)为黑色，(255,255,255)为白色。方框 1 中，左边缘颜色最黑（见图 X），此处地势较低，用来模拟沙滩和海的连接处。方框 2 中，（见图 X）地形图为不均匀的灰色，用来模拟沙滩中间部分。

2、沙滩实现方法：

1）生成网格顶点

沙滩平面大小 row=1000，column=800，利用 for 循环生成网格顶点，传递给 VBO (Vertex Buffer Objects)和 EBO (Element Buffer Object)。VBO 中前 3 位是顶点坐标，后 2 位是纹理坐标，EBO 记录了画三角形的索引，可以有效减少 VBO 的数据量。

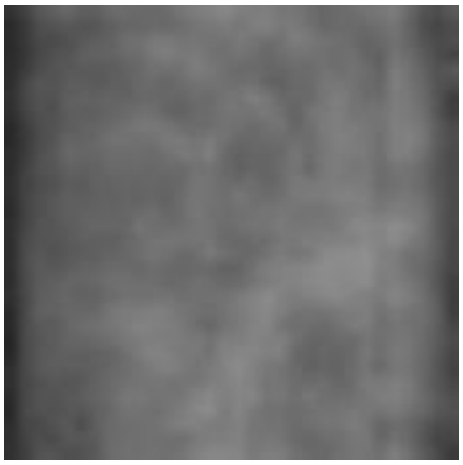
```
// 生成沙滩
//row_num表示网格行数，col_num表示网格列数
int row_num = 1000, col_num = 800;
std::vector<float> p;
float xx, zz = -50;
for (int i = 0; i < row_num; i++)
{
    xx = -5;
    for (int j = 0; j < col_num; j++)
    {
        p.push_back(xx);
        p.push_back(0);
        p.push_back(zz);
        p.push_back(1.0f / col_num * j);
        p.push_back(1 - i * 1.0f / row_num);
        xx += 0.1;//-5~75
    }
    zz += 0.1;
}
std::vector<unsigned int> indicess;
for (int i = 1; i < row_num; i++) {
    for (int j = 1; j < col_num; j++) {
        indicess.push_back((i - 1) * col_num + j - 1);
        indicess.push_back((i - 1) * col_num + j);
        indicess.push_back(i * col_num + j - 1);

        indicess.push_back(i * col_num + j - 1);
        indicess.push_back((i - 1) * col_num + j);
        indicess.push_back(i * col_num + j);
    }
}
```

```
unsigned int VBOsand, VAOsand, EBOsand;
glGenVertexArrays(1, &VAOsand);
glGenBuffers(1, &VBOsand);
glGenBuffers(1, &EBOsand);
// bind the Vertex Array Object first, then bind and set vertex buffer(s), and then configure vertex attributes(s).
glBindVertexArray(VAOsand);

glBindBuffer(GL_ARRAY_BUFFER, VBOsand);
glBufferData(GL_ARRAY_BUFFER, p.size() * sizeof(float), &p[0], GL_STATIC_DRAW);

glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EBOsand);
glBufferData(GL_ELEMENT_ARRAY_BUFFER, indicess.size() * sizeof(unsigned int), &indicess[0], GL_STATIC_DRAW);

glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 5 * sizeof(float), (void*)0);
glEnableVertexAttribArray(0);
// color attribute
glVertexAttribPointer(1, 2, GL_FLOAT, GL_FALSE, 5 * sizeof(float), (void*)(3 * sizeof(float)));
glEnableVertexAttribArray(1);
// note that this is allowed, the call to glVertexAttribPointer registered VBO as the vertex attribute's bound vertex
glBindBuffer(GL_ARRAY_BUFFER, 0);
glBindVertexArray(0);
```

2）绑定纹理

① 纹理图片

地形高度图：



沙子纹理：

沙滩与海交界纹理：这是一张带透明通道的图片

② 在读取有透明通道的图片时，需要开启 Blend 混合模式 `glEnable(GL_BLEND);`。

Blend 混合是将源色和目标色以某种方式混合生成特效的技术。混合常用来绘制透明或半透明的物体。在混合中起关键作用的α值实际上是将源色和目标色按给定比率进行混合，以达到不同程度的透明。α值为 0 则完全透明，α值为 1 则完全不透明。混合操作只能在 RGBA 模式下进行，颜色索引模式下无法指定α值。物体的绘制顺序会影响到 OpenGL 的混合处理。

③ 绑定纹理代码如下：

```
//加载纹理，创建shader并连接到程序上
unsigned int tex1 = loadTexture("file/res/SAND3.bmp");
unsigned int tex2 = loadTexture("file/wavenew9.png");//这个有透明通道
unsigned int dep = loadTexture("file/Terrain5.bmp");
beachshader.use();
beachshader.setInt("dep", 0);
beachshader.setInt("tex1", 1);
beachshader.setInt("tex2", 2);
```

3）沙滩着色器：

① vertex shader：

地形：读取灰度图纹理颜色 r 分量，赋值到顶点的 y 分量

```
void main()
{
    float height=texture(dep,texcoord).r;

    vec4 pos_local = matLocal * vec4(position, 1.0);
    vec3 new_pos=vec3(pos_local.x,height*5,pos_local.z);
    gl_Position = projection*view*model*vec4(new_pos, 1.0f);
    n_p=new_pos;
    coord=texcoord;
}
```

② fragment shader:

tex1 为沙滩纹理，tex2 为沙滩海洋交界的纹理。

```
#version 330 core
out vec4 FragColor;

in vec2 coord;
uniform sampler2D tex1;
uniform sampler2D tex2;
uniform sampler2D dep;
in vec3 n_p;

void main()
{
    int m=2;
    if(n_p.y>m)
    {
        FragColor=texture2D(tex1,coord);
    }
    else
    {
        float alpha=n_p.y/(m);  // 比例系数
        FragColor=texture2D(tex1,coord)*alpha+texture2D(tex2,coord)*(1-alpha);
        //FragColor=texture2D(tex2,coord);
        //FragColor=texture2D(tex1,coord)*0.9+texture2D(tex2,coord)*0.1;
    }
}
```

3、海的实现

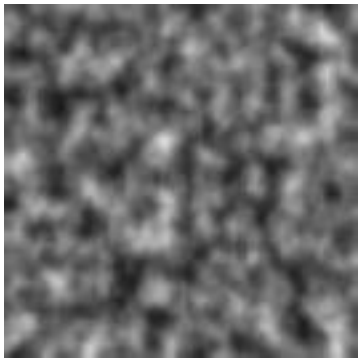
1）生成网格顶点

和沙滩类似，首先生成海洋的网格顶点，传递给 VBO (Vertex Buffer Objects) 和 EBO (Element Buffer Object)。VBO 中读入储存在 vector 容器的 vec3 向量，EBO 记录了画三角形的索引。

```
Ocean() {
    glGenVertexArrays(1, &VAO);
    glGenBuffers(1, &VBO);
    glGenBuffers(1, &EBO);
    // ...
    float length = 20;
    float width = 20;
    int resX = 4;
    int resZ = 4;
    for (int z = 0; z < resZ; z++)
    {
        // [ -length / 2, length / 2 ]
        float zPos = ((float)z / (resZ - 1) - .5f) * length;
        for (int x = 0; x < resX; x++)
        {
            // [ -width / 2, width / 2 ]
            float xPos = ((float)x / (resX - 1) - .5f) * width;
            vertices.push_back(glm::vec3(xPos, 0.0f, zPos));
        }
    }
    int nbFaces = (resX - 1) * (resZ - 1);
    int t = 0;
    for (int face = 0; face < nbFaces; face++)
    {
        // Retrieve lower left corner from face ind
        int i = face % (resX - 1) + (face / (resZ - 1) * resX);

        indices.push_back(i + resX);
        indices.push_back(i + 1);
        indices.push_back(i);

        indices.push_back(i + resX);
        indices.push_back(i + resX + 1);
        indices.push_back(i + 1);
    }
```

2）绑定纹理

① 海面形态纹理：



② 天空盒的映射：

代码如下：

```
perlintex = LoadMipmapTexture("file/res/perlin_noise3.png");
skytex = loadCubemap(faces_rainbow);
```

```
GLuint LoadMipmapTexture(const char* fname)
{
    int width, height, nrComponents;
    unsigned char* image = stbi_load(fname, &width, &height, &nrComponents, 0);
    if (image == nullptr)
        throw std::exception("Failed to load texture file");

    GLuint textureId;
    glGenTextures(1, &textureId);

    glBindTexture(GL_TEXTURE_2D, textureId);
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, image);
    glGenerateMipmap(GL_TEXTURE_2D);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
    // Set texture filtering parameters
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAX_LOD, 3);
    stbi_image_free(image);
    glBindTexture(GL_TEXTURE_2D, 0);

    return textureId;
}
```

```
unsigned int loadCubemap(vector<std::string> faces)
{
    unsigned int textureID;
    glGenTextures(1, &textureID);
    glBindTexture(GL_TEXTURE_CUBE_MAP, textureID);

    int width, height, nrComponents;
    for (unsigned int i = 0; i < faces.size(); i++)
    {
        unsigned char* data = stbi_load(faces[i].c_str(), &width, &height, &nrComponents, 0);
        if (data)
        {
            glTexImage2D(GL_TEXTURE_CUBE_MAP_POSITIVE_X + i, 0, GL_RGB, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, data);
            stbi_image_free(data);
        }
        else
        {
            std::cout << "Cubemap texture failed to load at path: " << faces[i] << std::endl;
            stbi_image_free(data);
        }
    }
    glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
    glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
    glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_R, GL_CLAMP_TO_EDGE);

    return textureID;
}
```

3）海洋着色器：

vertex shader 起到把世界坐标转换成本地坐标的作用，与沙滩 shader 类似。

fragment shader：

实现海浪的动态绘制：

① 在风向 `#define WIND_DIRECTION { -0.4f, -0.9f } //海洋风向` 的基础上，利用计时器 glfwGetTime()的变化改变当前顶点的颜色。

```
perlinoffset[0] = -w[0] * glfwGetTime() * 0.06f;
perlinoffset[1] = -w[1] * glfwGetTime() * 0.06f;
```

```
vec2 p0 = texture(perlin, ptex * perlinFrequency.x + perlinOffset).rg;
vec2 p1 = texture(perlin, ptex * perlinFrequency.z + perlinOffset).rg;

perl = (p0 * perlinGradient.x + p1 * perlinGradient.z) * 2;
```

```
vec4 grad = vec4(0, 0, 0.20, 1.0);
grad.xy = perl;

vec3 n = normalize(grad.xzy);
vec3 v = normalize(vdir);
vec3 l = reflect(-v, n);

float F0 = 0.5;
float F = F0 + (1.0 - F0) * pow(1.0 - dot(n, l), 5.0);
```

② 计算太阳的反光点：

```
vec3 h = sundir + v;
vec3 x = cross(sundir, n);
vec3 y = cross(x, n);

float mult = (ONE_OVER_4PI * rho / (ax * ay * sqrt(max(1e-5, dot(sundir, n) * dot(v, n)))));
float hdotx = dot(h, x) / ax;
float hdoty = dot(h, y) / ay;
float hdotn = dot(h, n);

float spec = mult * exp(-((hdotx * hdotx) + (hdoty * hdoty)) / (hdotn * hdotn));
```

③ 最后，混合海洋的颜色、动态海浪、反射天空、太阳光三种颜色作为渲染的颜色。

```
my_FragColor0 = vec4(mix(oceanColor, refl, F) + sunColor * spec * 0.5, 0.9);
```

四、碰撞检测

1、实现方法

在被碰撞的物体周围设置两个包围圈，当相机进入内外圈之间则记录相机位置，当相机进入内圈则让相机回退到所记录的位置。

2、核心代码

```
void collideTest()//女神像碰撞检测
{
    House h;
    float current_diatance = pow(abs(camera.Position.x - h.x), 2) + pow(abs(camera.Position.y - h.y), 2) + pow(abs(camera.Position.z - h.z), 2);

    if (current_diatance < outsideD && current_diatance > insideD)//当闯入包围盒内外圈之间时，记录相机此时坐标
    {
        remember_distance[0] = camera.Position.x;
        remember_distance[1] = camera.Position.y;
        remember_distance[2] = camera.Position.z;
    }
    if (current_diatance <= insideD)//当闯入包围盒内圈时，将它弹回去
    {
        camera.Position.x = remember_distance[0];
        camera.Position.y = remember_distance[1];
        camera.Position.z = remember_distance[2];
        //Camera c;
        //c.Position = glm::vec3(remember_distance[0], remember_distance[1], remember_distance[2]);
    }
}
```

五、灯光设置

1、实现方法

灯光的效果我们都写在 Shader 文件中，用 Shader 来控制灯光。第一幕中的环境光慢慢点亮的效果，是在天空盒的 Shader 文件中用了一个雾的效果来实现的。让雾的浓度随着时间而变化，最终就能达到环境慢慢变亮的效果。第一幕中还有一个手电筒的效果，这个是在一个另外的 Shader 文件中写的。第二幕中的光照是全局光照，采用的是一个平行光的写法。

2、核心代码

天空盒 Shader：

```
void main()
{
    vec3 ambient = light.ambient * texture(skybox, TexCoords).rgb;

    float fogcoord = (gl_FragCoord.z/gl_FragCoord.w) * 200;
    //vec4 fogcolor = vec4(1, 1, 1, 0.5);
    float Fog = exp2( - 0.000008 * fogcoord * fogcoord * time);
    Fog = clamp(Fog, 0.0, 1.0);
    vec4 color = vec4(ambient, 1.0);

    if (time == 0)
    {
      FragColor = color;
    }
    if (time > 0)
    {
      FragColor = mix(fogcolor, color, Fog);
    }
}
```

手电筒 Shader，实现投射光的效果：

```
void main()
{
    // ambient
    vec3 ambient = light.ambient * texture(material.diffuse, TexCoords).rgb;

    // diffuse
    vec3 norm = normalize(Normal);
    vec3 lightDir = normalize(light.position - FragPos);
    float diff = max(dot(norm, lightDir), 0.0);
    vec3 diffuse = light.diffuse * diff * texture(material.diffuse, TexCoords).rgb;

    // specular
    vec3 viewDir = normalize(viewPos - FragPos);
    vec3 reflectDir = reflect(-lightDir, norm);
    float spec = pow(max(dot(viewDir, reflectDir), 0.0), material.shininess);
    vec3 specular = light.specular * spec * texture(material.specular, TexCoords).rgb;

    // spotlight (soft edges)
    float theta = dot(lightDir, normalize(-light.direction));
    float epsilon = (light.cutOff - light.outerCutOff);
    float intensity = clamp((theta - light.outerCutOff) / epsilon, 0.0, 1.0);
    diffuse  *= intensity;
    specular *= intensity;

    // attenuation
    float distance    = length(light.position - FragPos);
    float attenuation = 1.0 / (light.constant + light.linear * distance + light.quadratic * (distance * distance));
    ambient  *= attenuation;
    diffuse   *= attenuation;
    specular *= attenuation;

    vec3 result = ambient + diffuse + specular;
    FragColor = vec4(result, 1.0);

}
```

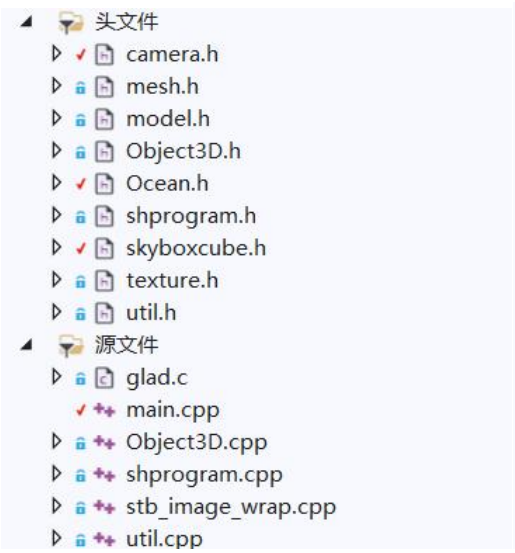
第二幕中全局光照的 Shader：

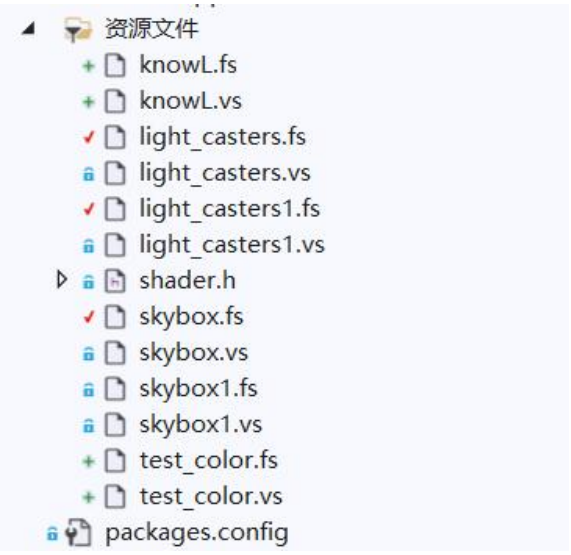
```
void main()
{
    // ambient
    vec3 ambient = light.ambient * texture(skybox, TexCoords).rgb;

    FragColor = vec4(ambient, 1.0);
}
```

实验结果：

1、文件结构介绍：

camera.h 控制观察相机，可以调整视角视点等。
mesh.h 将读入的模型进行绑定，加载纹理。
model.h 读取 obj 模型文件，处理顶点、面、法向量、纹理等，绘制模型。
texture.h 加载纹理和绑定纹理。
skyboxcube.h 绘制天空盒背景。
Ocean.h 为添加海浪
Main.cpp 为主函数





辅助库：

assimp 库读取并加载 obj 模型。

glad 库可以简化在运行时获取函数地址并将其保存在一个函数指针中供以后使用的过程，并且依赖了 KHR 库。

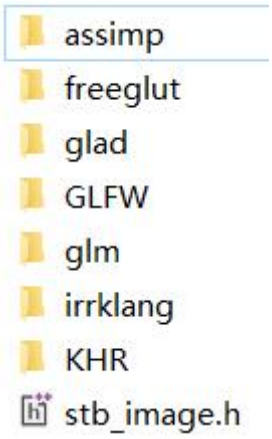
GLFW 为基本库，用于图形、窗口、渲染等等。

glm 库进行向量运算，方便数学计算。

Irrklang 库用于播放音频。

stb_image.h 用于读取图片。

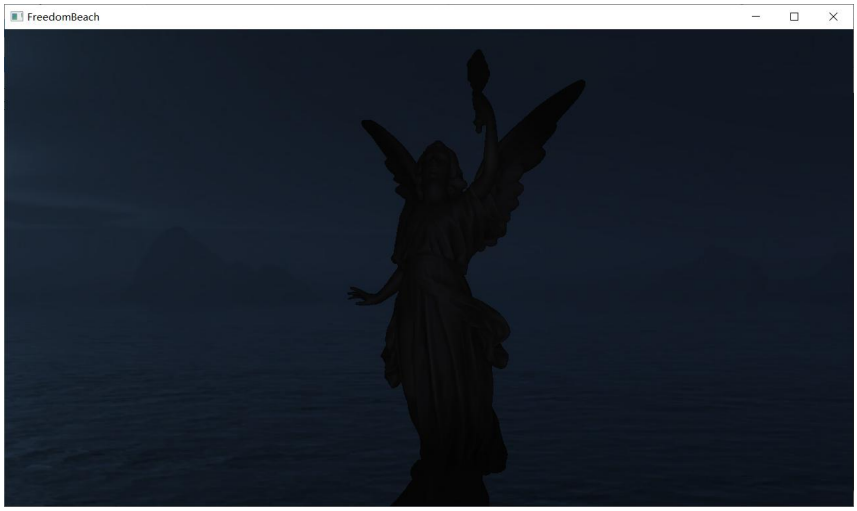
Freeglut 库作为 glut 的替代库进行事件处理



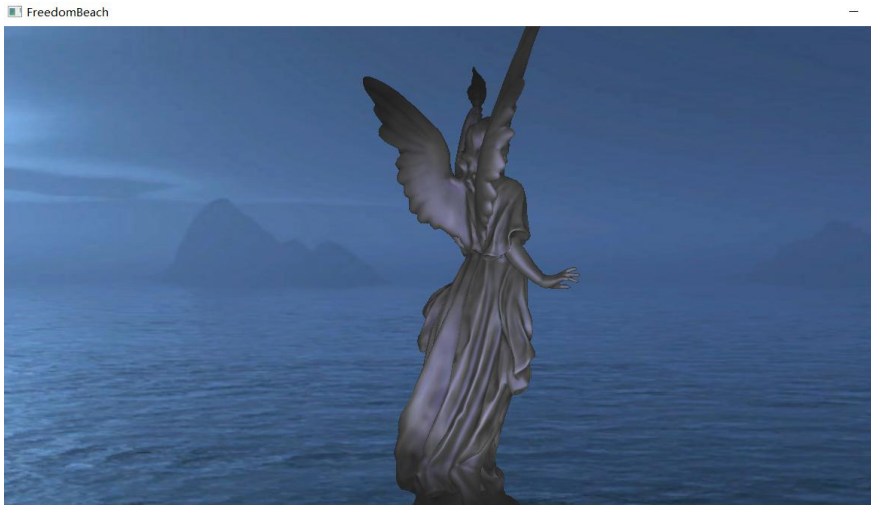
2、交互过程

在进入第一个场景后出现女神像，必须按下空格键打开手电筒，镜头向前推进，场景变亮实现场景跳转，如果不按空格键，15 秒后直接退出；

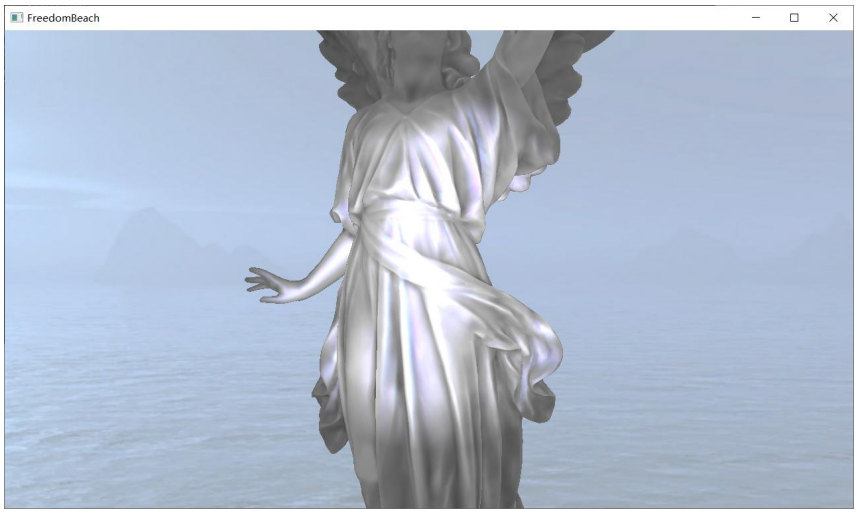
没有打开手电筒的场景：



打开手电筒后的场景：

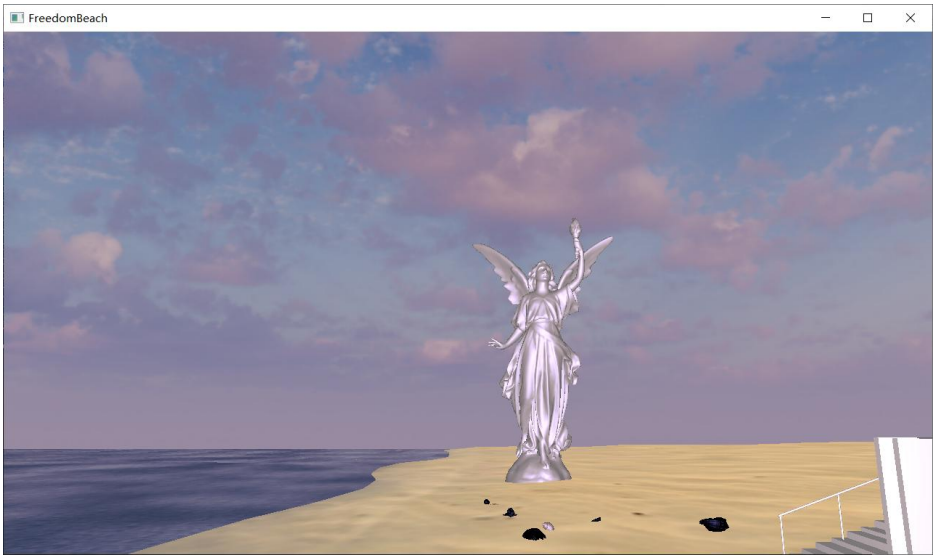


手电筒亮度会越来越亮，周围的环境光也越来越亮，并且女神像会旋转，镜头推进：



打开手电筒后，控制台会输出的一个小彩蛋，我们希望大家的心中都能有这样一团炬火，不断向上，成为黑暗中的光：

在进入第二个场景后可以进行场景漫游，用 WSAD 控制前后左右的移动，用鼠标实现视角的控制；



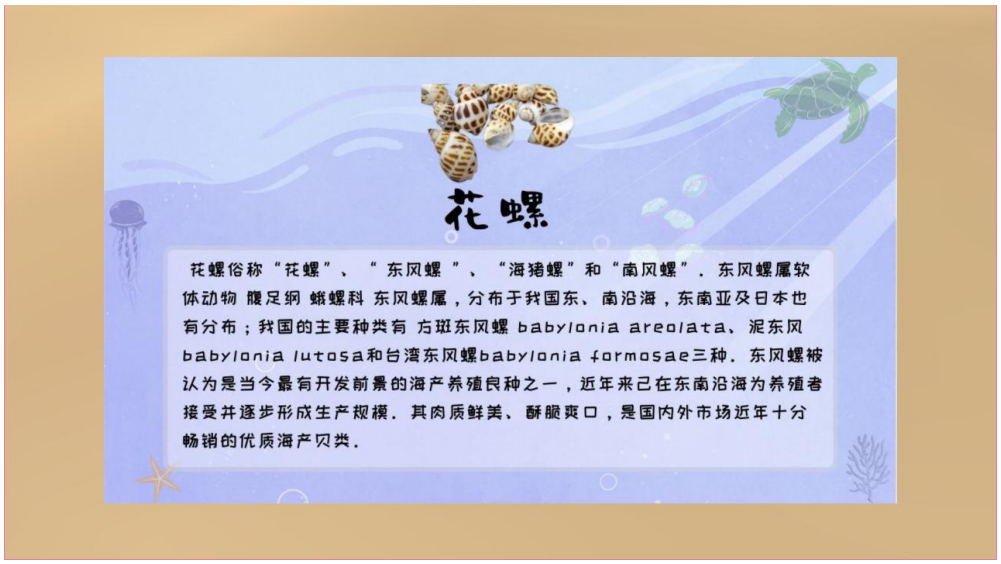
当靠近贝壳时，贝壳会变大并且显示白色轮廓，提示当前最近贝壳是哪一个，此时按下鼠标左键将弹出当前贝壳的介绍图鉴，放开左键图鉴消失，可继续漫游；

3、贝壳交互效果

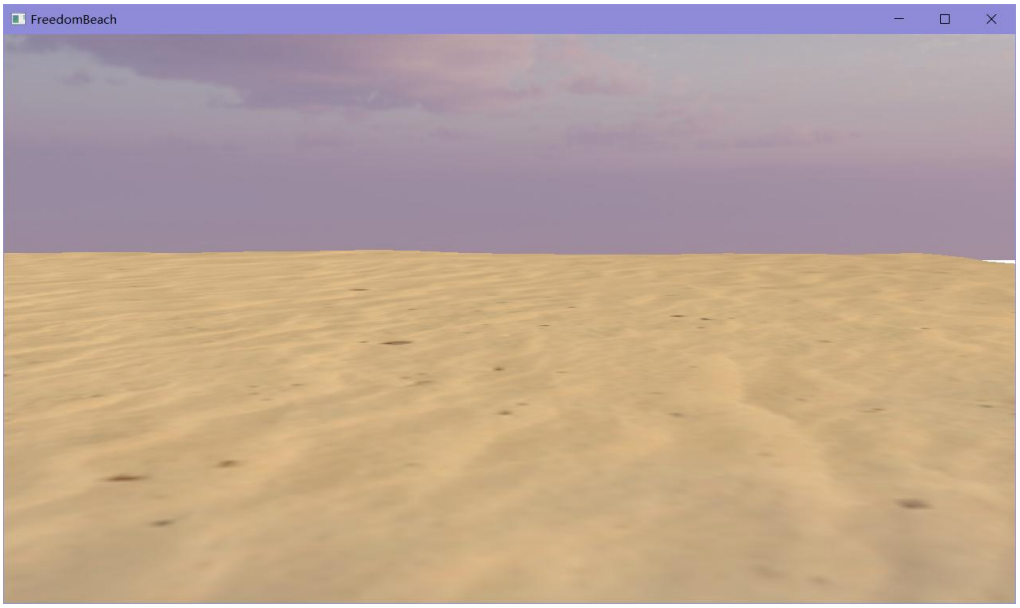
图 1 贝壳没有交互效果

图 2 贝壳有交互效果

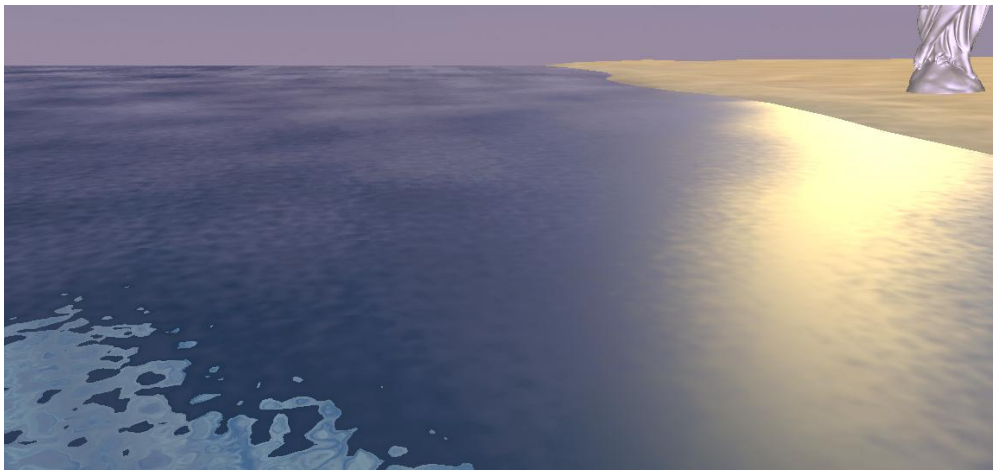
4、科普信息显示



5、沙滩的实现



6、海的实现



结论分析：

计算机图形学的课程设计，我们小组五个人共同完成了这个沙滩漫游的项目。我们采用了 OpenCV 来实现这个项目，并且沙滩和海水并不是建模实现的，而是用 shader 实现的。

课程设计的难度，相较于平时实验的难度，有了一个拔高。在课程设计中我们碰到了很多没有遇到过的问题，甚至这个课程设计一开始我们连该从哪里开始入手都不知道。我们携手一起研究，逐渐一步一步地解决一个个问题，最终搭建起了这片自由沙滩。我们在代码中运用了很多平时课上没有学习到的技术，例如 shader 编程、三维模型的读入、碰撞检测等等。在完成课程设计的四五天时间内，我们都是一边在学习新知识，一边把学到的知识活学活用。最终能在四五天的时间内完成这样一个课程设计，我们的小组成员都尽了自己的力，相当不容易。

虽然相较于一些使用 Unity 完成的课设，我们的完成度看起来会有些不足。但是我们的代码都比较底层，有一定的难度。很多在 unity 中已经封装好的功能，我们在代码中都是自己一行一行研究出来的。

在真实感方面，我们的程序还有些欠缺。有时间有机会我们会继续改进代码的阴影和光照渲染。虽然图形学的课程设计在此告一段落，但是我们在计算机图形学方面的学习并没有结束。希望在今后，我们能够用自己的代码，在二进制的电子世界中构建出一个无限接近现实的美丽世界。