- 实验报告
  - 光源
  - 水体
    - 反射和折射
    - 水面波纹
      - gerstner波
      - 噪声纹理贴图
  - 模型的选中及移动
  - 模型导入和阴影贴图
  - 地面高度图
  - 摄像机类
  - 参考资料

实验报告

光源

这里设置了一个dirLight和四个点光源，后面阴影贴图就是基于dirLight（海边主要是太阳光，太阳光近似平行光）。

对于平行光，有ambient（环境光），diffuse（漫反射光），specular（镜面反射），当平行光比较垂直于地面的法线时，类似中午的太阳，场景会更亮；对于点光源，需要包括衰减系数，不过对于空旷的海滩场景不是很需要点光源。

水体

反射和折射

反射和折射的原理都是通过在一个角度对场景进行渲染，然后创建一个帧缓冲保存到纹理中，然后渲染水体时再将纹理渲染到水面上

渲染的代码见 `utils/render.h/RenderWaterRefraction()`和 `utils/render.h/RenderWaterReflection()`

反射需要将摄像机根据水平面进行对称 `y=2*waterHeight-camPos.y`；折射摄像机不用动，但是需要渲染深度纹理，后面可以通过地面到水面的距离来渲染颜色。

渲染水体颜色时，我们考虑水上和水下：

对于水上，我们计算纹理的uv坐标，从反射（需要倒过来）和折射纹理采样，根据水面的法向量进行一定的扭曲

```
vec2 ndc = glp.xy / glp.w;//得到标准化统一设备坐标
vec2 refractTexCoords = ndc * 0.5 + 0.5 + norm.xz * 0.1;//进行水面扭曲
vec2 reflectTexCoords = vec2(refractTexCoords.x, 1.0 - refractTexCoords.y);
```

之后采样可以得到屏幕的反射和折射颜色

需要考虑fresnel系数，当入射角越大，反射光越强，否则折射光越强，见`background\water.fs`可以看到`vec3 finalColor = mix(refraction, reflection, fresnel *0.6);`

反射需要blin-phong的漫反射、镜面、环境系数相乘，然后折射颜色要和深度相符合，以及水的alpha系数（透明度），当水越深时越暗，最后折射和反射混合，再将最终颜色和深度颜色混合

对于水下，水下没有反射了，直接将折射的颜色和蓝色混合得到最终颜色

水下还有需要注意的是，需要给天空盒和地面增加雾化，天空盒就是直接变成深色，地面根据与摄像机的远近渲染颜色（越远越看不清）

```
//beach-render/background/terrain.fs
float fogFactor = clamp((fogEnd - dist) / (fogEnd - fogStart), 0.0, 1.0);
FragColor = mix(fogColor, FragColor, fogFactor);
```

水面波纹

水面波纹我查阅了很多资料，发现主流的是通过菲利普频谱用FFT渲染波浪gerstner波

gerstner波

这种方法是基于统计学得到的，因为海面的波纹近似摆线波，用多个摆线波叠加来模拟海浪波形，摆线波和正弦波最大的区别就是x和z会进行圆周运动，y进行垂直运动

先生成菲利普频谱

菲利普频谱类似于能量，顺风的能量增大，因此振幅就会更明显；k为采样点

根据菲利普频谱可以得到h_0初始频谱，然后得到每个$h(k, t)$

$$\tilde{h}_0(\vec{k}) = \frac{1}{\sqrt{2}}(\xi_r + i\xi_i)\sqrt{P_h(\vec{k})}$$

水面的速度为于$\sqrt{g * \lambda}$

$$\tilde{h}(\vec{k}, t) = \tilde{h}_0(\vec{k})e^{i\omega(k)t} + \tilde{h}_0^*(-\vec{k})e^{-i\omega(k)t}$$

其中h(k,t)即为waves

因此海洋的idft如下：

$$h(\vec{x}, t) = \sum_{\vec{k}} \tilde{h}(\vec{k}, t)e^{i\vec{k}\cdot\vec{x}}$$

水平位移如下：

$$\vec{D}(\vec{x}, t) = \sum_k -i\frac{\vec{k}}{k}\tilde{h}(\vec{k}, t)e^{i\vec{k}\cdot\vec{x}}$$

然后可以通过fft实现加速，然后海面的发现通过直接对 `h(x,t)` 求导，可以得到如下公式

$$\vec{N} = \text{normalize}\,(-\nabla h_x(\vec{x}, t),\ 1,\ -\nabla h_z(\vec{x}, t))$$

```
//background\waterplane_gerstner.h
float omega = std::sqrt(9.81f * k_length);
float phase = omega * time;
Complex phase_exp(std::cos(phase), std::sin(phase));

int m_conj = (N - m) % N;
int n_conj = (N - n) % N;
int conj_index = m_conj * N + n_conj;

// h̃(k, t)
waves[index] = h0[index] * phase_exp
               + std::conj(h0[conj_index]) * std::conj(phase_exp);

// 位移频谱: D = -i * (k/|k|) * h̃
Complex i_unit(0.0f, 1.0f);
glm::vec2 k_norm = K / k_length;

waves_x[index] = -i_unit * k_norm.x * waves[index];
```

```
waves_z[index] = -i_unit * k_norm.y * waves[index];
waves_y[index] = waves[index];

// 斜率频谱: S = i * k * h̃
// ∂h/∂x ↔ i·k_x·h̃
slopes_x[index] = i_unit * K.x * waves[index];
// ∂h/∂z ↔ i·k_z·h̃
slopes_z[index] = i_unit * K.y * waves[index];
```

之后对waves数组进行IFFT2D

需要注意归一化，以及法线朝向y轴正半轴方向

```
float scale = 1.0f / (N*N);
for (int i = 0; i < N * N; i++) {
    data[i] *= scale;
}
```

噪声纹理贴图

因为海浪其实是可以提前算出来的，而且该傅里叶变换是有周期的，因此我们可以提前渲染一个周期的海面波纹高度图，然后循环播放，这里用到了3D噪声纹理存储(x,z,t)的y轴

通过这种预处理的方法能够在笔记本上也能不卡顿的实时显示海波

代码见 background\ocean_fft_baker.h和 background\waterplane_baked.h

```
float k_min =  M_PI / L;
float omega_max = std::sqrt(9.81f * k_min);
float minPeriod = 2.0f * M_PI / omega_max;
//minperiod代表一个周期

// 存储到 3D 纹理缓冲
for (int m = 0; m < N; m++) {
    for (int n = 0; n < N; n++) {
        int spatialIdx = m * N + n;
        int volumeIdx = t * N * N + m * N + n;

        // 存储位移量 (不是绝对位置)
        displacementData[volumeIdx] = vertices[spatialIdx] -
originalPos[spatialIdx];
        normalData[volumeIdx] = normals[spatialIdx];
    }
}
```

见water.vs

```
vec3 displacement = texture(displacementMap, uvw).xyz;
vec3 displacedPos = aPos + displacement;//应用海面波浪高度图
```

模型的选中及移动

对于模型的选中，通过右键鼠标事件，用射线和aabb相交检测，代码见 `main.cpp/mouse_button_callback`

1、从摄像机的位置发出一道射线，方向是鼠标的方向（通过逆运算求到鼠标对应世界坐标的方向，具体见代码 `main.cpp/ScreenToWorldRay(doublemouseX, doublemouseY, Camera&camera, glm::vec3&rayOrigin, glm::vec3&rayDir)`）

2、对每个物体判断射线是否穿过该物体，这里我们把物体看成一个长方体，然后对每个轴判断射线进入时间和出去时间（射线方程为$origin + t * direction$，我们可以把t的值看作是射线上的时间），如果一个射线穿入该物体，那么他的最晚进入时间一定小于等于最早出去时间，由此可以判断物体是否在射线范围内，具体见代码 `gameobject.h/RayIntersect(constglm::vec3&rayOrigin, constglm::vec3&rayDir, floatmaxDistance=1000.0f)`

3、然后对于所有候选物体，我们选择离摄像头最近的（远的被挡住了）

为了体现物体选中，在模型加载的fs中增加了 `result = mix(result, selectColor,0.3);// 混合30%的选中颜色`，当物体偏黄时表示选中。

对于物体的放置，我们在右键按下时，如果有物体被选中了，那么将物体的位置改变即可，物体在鼠标射线中，而我们希望物体放置在地面上，因此获取射线与地面的交点即可，而我们的地面起伏在0附近，因此可以设置y=0减少运算

对于鼠标移动时模型的跟随，当鼠标移动时并且存在选中物体时，我们新建一个 `movingObject`用于移动时显示，坐标同样是鼠标射线与地面相交的位置

由于我们前面设置的y为0其实并不标准，因此我这里在每次渲染时增加了一个吸附地面，见代码 `gameobject.h/snaptoterrain`，物体从terrain类中获得地面高度并调整

模型导入和阴影贴图

模型的导入，是通过多个网格（包括贴图，顶点坐标和顶点索引）合并得到的，绘制的时候就非常简单，直接 `mesh.draw()` ，见 `model/model.h`，不过我下载的.obj文件是嵌入式文件，需要特殊处理。

有些模型导入由于坐标系统的不一样会导致翻转，这种情况比较麻烦，需要把所有点都倒置

增加 `GameObject`存储一个model物体，增加静态变量存储所有游戏物体（用于计算阴影时渲染场景）

阴影渲染是摄像机在光源的方向然后渲染场景，获得深度贴图，后面在地面时，如果比深度贴图深，那么就说明被遮挡了，然后渲染阴影颜色

```
// 获取最近点的深度（从光的角度）
float closestDepth = texture(shadowMap, projCoords.xy).r;
// 获取当前片段深度
float currentDepth = projCoords.z;
```

因为分辨率和采样空间不通的问题，可能会出现阴影失真，然后增加一点偏移防止失真 `float bias = max(0.05*(1.0- dot(normal, lightDir)),0.005);`让当前深度变深一点，防止错误渲染成阴影

增加PCF软投影（平滑边缘），减少锯齿块。

地面高度图

为了实现沙滩的纹理和地形，我下载了一张高度图和纹理照片，纹理通过重复10次来使分辨率更高，地面的渲染是通过绘制N*M的网格实现的，然后给(x,z)附上高度图的高度y，法线可以通过三角形平面计算

摄像机类

参考资料

基于OpenGL的动态水面模拟 - 知乎

LearnOpenGL CN

water wave simulation

KoonanHyakukeiZu/2020-ZJUCG-Final-Project

https://zhuanlan.zhihu.com/p/95482541