计算机图形学基础

A 类作业实验报告

杨宇菲 计算机系 2020215224
yangyufe20@mails.tsinghua.edu.cn

实验环境：Windows 10, Microsoft Visual Studio 2012

题目一：Terrain Engine

- **实验原理**

  基于以下部分的实现，进行真实可探索的 3D 场景绘制：

  ✧ 天空盒及其倒影的绘制

  ✧ 读取地形高度图（greyscale），逐地块进行地形及其倒影的绘制

  ✧ 带透明度的海面的绘制，实现海面上投射出倒影的效果，并通过移动纹理坐标实现波浪移动

- **实验步骤**

1. 相机

   计算 view 和 projection 矩阵，传入 ourShader 和 ourShaderTerrain 的顶点着色器。

2. 绘制天空盒及其倒影

   设顶点着色器中的 uniform vec2 变量 offset 为(0.0, 0.0)，不进行纹理移动。设片段着色器中的 uniform float 变量为 1.0，不透明。

   绘制天空盒时，设顶点着色器中的 uniform int 变量 reflection 值为 0，不进行翻转，分别绘制天空盒的顶面和四个侧面。

   绘制天空倒影时，设顶点着色器中的 reflection 值为 1，将所有顶点的 y 坐标取负，同样绘制天空盒的顶面和四个侧面。

3. 地形及其倒影的绘制

   首先，用 SOIL_load_image 函数读入地形灰度图为 unsigned char* heightmap，SOIL_LOAD 模式设为 SOIL_LOAD_L。原图像共有 width 列，height 行（此项目中 width = height），此函数是从左上角开始按行读取的，每个像素的灰度值 heightmap[i]对应一个地块中心的相对高度，其绝对高度为 height[i] * unitHeight + Y0。

   设 Terrain 总长度为 terrainLength，则单位地块长度为 unitLength = terrainLength / width，对应地形纹理图中的单位长度为 unitTexLength = 1.0 / width。先将以原点为中心，左下角纹理坐标位于(0.0, 0.0)的地块顶点信息存入地形 VBO，并再设一位 float 类型输入变量 index，来标记顶点是左上角（index = 0.0）、右上角（index = 1.0）、右下角（index = 2.0）还是左下角（index = 3.0），方便进行高度的赋值。

```
GLfloat verticesTerrain[] = {
    -unitLength/2.0f, -unitLength/2.0f, 0.0f, unitTexLength, 0.0f,
    unitLength/2.0f, -unitLength/2.0f, unitTexLength, unitTexLength, 1.0f,
    unitLength/2.0f, unitLength/2.0f, unitTexLength, 0.0f, 2.0f,
    -unitLength/2.0f, unitLength/2.0f, 0.0f, 0.0f, 3.0f
};
```

   Terrain 左上角的 x 和 z 坐标为 X0 = - terrainLength / 2.0f，Z0 = - terrainLength / 2.0f。

从左上角开始按行遍历所有地块（除最右一列和最下一行），每一地块的中心点（index = 0.0）与其右侧（index = 1.0）、下侧（index = 2.0）、右下侧（index = 3.0）的地块中心点组成两个三角面片。对第 i 行第 j 列的地块，两个三角面片相对 VBO 中的顶点(x, z)坐标需移动(X0 + (j + 1) * unitLength, Z0 + (i + 1) * unitLength)，相对 VBO 中的纹理坐标需移动((j + 0.5) * unitTexLength, 1.0 - (i + 1.5) * unitTexLength)，此地块对应第 tempIndex = i * width + j 个高度，四顶点的相对高度分别为{heightmap[tempIndex]，heightmap[tempIndex + 1], heightmap[tempIndex + width + 1], heightmap[tempIndex + width]}。把(x, z)坐标位移、纹理坐标位移和绝对高度组成的 vec4 传入顶点着色器进行计算，顶点的 y 坐标取绝对高度的第(int)index 个元素。

对于地形，设顶点着色器中 uniform int 变量 reflection 值为 0，不进行翻转，绘制两个三角面片。对于倒影，设顶点着色器中 reflection 值为 0，对 y 坐标取负，绘制两个三角面片。

```
for (int i = 0; i < height - 1; i++)
{
    for (int j = 0; j < width - 1; j++)
    {
        GLfloat x = X0 + (j + 1) * unitLength;
        GLfloat z = Z0 + (i + 1) * unitLength;
        glUniform1f(xLocTerrain, x);
        glUniform1f(zLocTerrain, z);
        int tempIndex = i * width + j;
        glUniform4f(heightLocTerrain, ((GLfloat)heightmap[tempIndex] + Y0) * unitHeight,
                                      ((GLfloat)heightmap[tempIndex + 1] + Y0) * unitHeight ,
                                      ((GLfloat)heightmap[tempIndex + width + 1] + Y0) * unitHeight,
                                      ((GLfloat)heightmap[tempIndex + width] + Y0) * unitHeight);
        glUniform2f(offsetLocTerrain, (j + 0.5) * unitTexLength, 1.0f - (i + 1.5) * unitTexLength);

        glUniform1i(reflectLocTerrain, 0);
        glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_INT, 0);

        glUniform1i(reflectLocTerrain, 1);
        glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_INT, 0);
    }
}
```

4. 添加地形细节纹理

将地形图和细节图分别绑定在 GL_TEXTURE1 和 GL_TEXTURE2 上，并设地形片段着色器中的 uniform sampler2D ourTexture1 和 ourTexture2 分别为 1 和 2。

设细节纹理相对于地形纹理缩小 detailScale = 10.0 倍，即将地形纹理坐标 TexCoord1 乘以 detailScale 倍赋给细节纹理坐标 TexCoord2。

对根据两纹理坐标分别对相应纹理进行插值得到的颜色进行 ADD_SIGNED 类型的相加，即

```
vec4 ourcolor = vec4(color1.x+color2.x-0.5f, color1.y+color2.y-0.5, color1.z+color2.z-0.5f, alpha);
```

5. 删除水面下地形

在地形顶点着色器中设输出变量 float FragY 为顶点对应的绝对高度(若 reflection = 1，FrayY 为未翻转前的高度)，传入片段着色器。片段着色器对 FragY 进行插值后，若 FragY < 0.0，alpha 值设为 0.0 并 discard。

6. 绘制水面和波浪

设顶点着色器中的 uniform vec2 变量 offset 为 (waveSpeedX * (GLfloat)glfwGetTime(),waveSpeedY * (GLfloat)glfwGetTime())，水面以速度(waveSpeedX, waveSpeedY)进行向右上角平移。设片段着色器中的 uniform float 变量为 0.8，绘制时开启 GL_BLEND。

7. 需注意的点

每次绘制前，要激活对应的 shader 和 GL_TEXTURE，绑定 VAO、EBO，对 GL_TEXTURE_0，要更换绑定的纹理。绘制结束后，解除 VAO 和 EBO 的绑定。

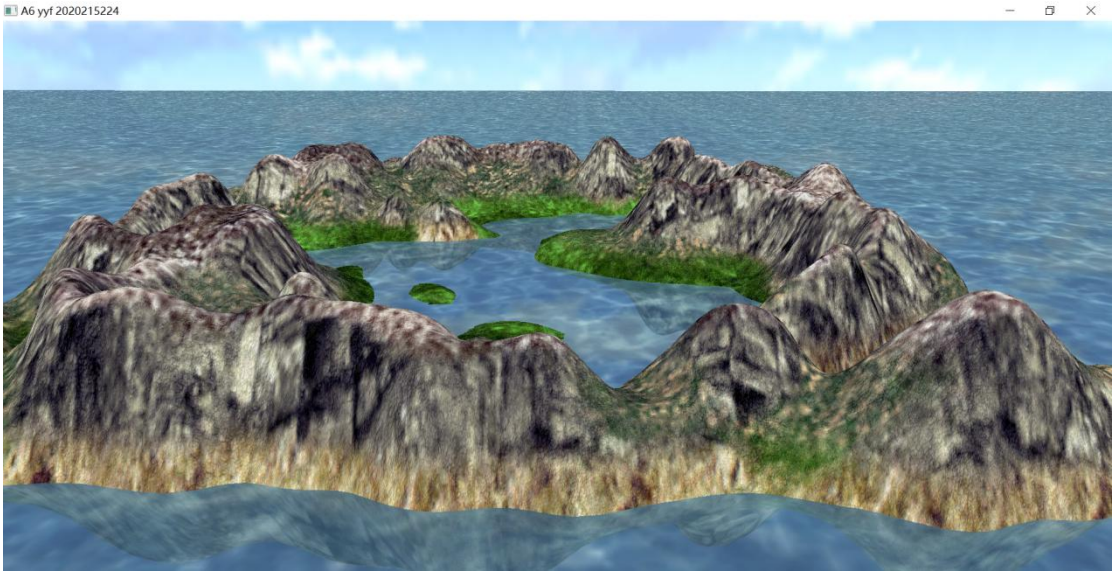
8. 消除卡顿的方法

一开始在绘制地形时，考虑到顶点的高度无法靠整体平移赋值，申请 GL_DYNAMIC_DRAW 类型的 VBO，然后在遍历地块时动态更新 VBO，导致严重的卡顿，无法渲染出画面。

后来通过先在静态 VBO 中存入(x, z)坐标和纹理坐标，绘制时向顶点着色器中传入位移矩阵 model 和纹理偏移 offset，进行(x, z)坐标和纹理坐标的平移，并在 VBO 中存入 float 类型 index 用来区分顶点位置，绘制时向顶点着色器中传入四个顶点的高度组成的 vec4，用索引(int)index 获取每个顶点的高度值，解决了严重卡顿的问题。

但因为在 CPU 中进行了太多 model::translate 计算，仍然有一点卡顿，于是分别计算 x 坐标偏移和 z 坐标偏移，传入顶点着色器中和原坐标相加，而不用 model 矩阵乘位置向量得到平移后的位置，解决了画面的卡顿问题。

- **实验效果（见录屏）**



交互方式：

ESCAPE 键——关闭窗口

W/A/S/D 键——相机位置前/后/左/右（自身坐标系）移动

↑/↓/←/→键——相机位置上/下（世界坐标系）/左/右（自身坐标系）移动

鼠标滚轮——视野缩放

鼠标移动——改变相机俯仰角和偏航角

题目五：光线跟踪算法

- **实验原理**

在光线跟踪算法中，从视点出发，对于视屏上的每一个像素点，从视点作一条到该像素点的射线，调用下面的算法函数就可以确定这个像素点的颜色。

```
RayTracing(start, direction, weight, color)
{
    if( weight > MaxWeight )
```

```
        color = black;
    else
    {
        计算光线与所有物体的交点中离 start 最近的点;
        if( 没有交点 )
            color = black;
        else
        {
            Ilocal = 在交点处用局部光照模型计算出的光强;
            计算反射方向 R;
            RayTracing(最近的交点, R, weight*Ws, Is);
            计算折射方向 T;
            RayTracing(最近的交点, T, weight*Wt, It);
            color=Ilocal + KsIs + KtIt;
        }
    }
}
```

光线跟踪算法的核心是光线与物体的求交，本次作业在原项目中光线与球体、光线与平面求交并获得交点纹理坐标的基础上，实现了光线与矩形、光线与圆柱体求交的算法，以及交点纹理坐标的获取。

- **实验步骤**

1. 光线与矩形的求交

在光线与平面求交的基础上（矩形法向量N为$(D_x * D_y).GetUnitVector()$），获取光线与平面的交点C。矩形的四个顶点按法向顺序依次为$O - D_x + D_y, O - D_x - D_y, O + D_x - D_y, O + D_x + D_y$。计算交点C与四个顶点的连线顺序形成的四个夹角的角度，若计算中发现交点在矩形的边上，则返回默认值的 CollidePrimitive ret，否则累加夹角，夹角的符号由两连线叉乘和矩形法线确定，若两连线叉乘和法线同向，符号为正，若反向，符号为负。

```
double dist = l;
Vector3 C = ray_O + ray_V * dist;
Vector3 vertices[5] = {O - Dx + Dy, O - Dx - Dy, O + Dx - Dy, O + Dx + Dy, O - Dx + Dy};
double addAngles = 0;
for(int i = 0; i < 4; i ++)
{
    Vector3 ray1 = vertices[i] - C;
    ray1 = ray1.GetUnitVector();
    Vector3 ray2 = vertices[i+1] - C;
    ray2 = ray2.GetUnitVector();
    if((ray1 * ray2).Module() < EPS)
        return ret;
    else
    {
        double angle = acos(ray1.Dot(ray2));
        addAngles += (ray1 * ray2).GetUnitVector().Dot(N) * angle;
    }
}
```

若交点C在矩形内部，得到的夹角和为2π，返回在此交点上的 ret；若交点C在矩形外部，夹角和为0，则返回默认值的 ret。

```
if (addAngles < PI * 2 - EPS)
    return ret;
else
{
    ret.dist = dist;
    ret.front = ( d < 0 );
    ret.C = C;
    ret.N = ( ret.front ) ? N : -N;
    ret.isCollide = true;
    ret.collide_primitive = this;
    return ret;
}
```

2. 获取交点在矩形上的纹理

crash_C距中心O的Dx方向距离为$(\text{crash_C} - \text{O}).\text{Dot}(\text{Dx})/\text{Dx}.\text{Module}()$，转换成纹理坐标中的距离为$(\text{crash_C} - \text{O}).\text{Dot}(\text{Dx})/\text{Dx}.\text{Module}()/(\text{Dx}.\text{Module}() * 2) = (\text{crash_C} - \text{O}).\text{Dot}(\text{Dx})/\text{Dx}.\text{Module2}()/2$。中心O在$D_x$方向的纹理坐标为0.5，则crash_C的纹理坐标为$(\text{crash_C} - \text{O}).\text{Dot}(\text{Dx})/\text{Dx}.\text{Module2}()/2 + 0.5$。Dy方向同理。

3. 光线与圆柱求交

圆柱底面中心为O1，顶面中心为O2，半径为R。设从底面指向顶面的法向量为u，则对无限长圆柱，圆柱面上的点P满足

$$|(\text{P} - \text{O1}) \times \text{u}| = \text{R}$$

将$\text{P} = \text{ray_O} + \text{t} \cdot \text{ray_V}$带入上式

$$|(\text{ray\_O} + \text{t} \cdot \text{ray\_V} - \text{O1}) \times \text{u}| = \text{R}$$

$$|\text{t} \cdot \text{ray\_V} \times \text{u} + (\text{ray\_O} - \text{O1}) \times \text{u}| = \text{R}$$

设

$$\text{a} = \text{ray\_V} \times \text{u}$$

$$\text{b} = (\text{ray\_O} - \text{O1}) \times \text{u}$$

有

$$|\text{t} \cdot \text{a} + \text{b}| = \text{R}$$

$$(\text{t} \cdot \text{a} + \text{b})^2 = \text{R}^2$$

$$\text{a}^2 + 2\text{a} \cdot \text{bt} + \text{b}^2 - \text{R}^2 = 0$$

设$\Delta = (\text{a} \cdot \text{b})^2 - \text{a}^2(\text{b}^2 - \text{R}^2)$。若$\Delta < 0$,一定与圆柱无交点；否则，有

$$\text{x1} = \frac{-\text{a} \cdot \text{b} - \sqrt{\Delta}}{\text{a}^2}, \qquad \text{x2} = \frac{-\text{a} \cdot \text{b} + \sqrt{\Delta}}{\text{a}^2}$$

若$\text{x2} < 0$，光线指向圆柱体的反向，与圆柱无交点；否则，若$\text{x1} > 0$，光线从外部与圆柱相交，$\text{t} = \text{x1}$；若$\text{x1} < 0$，光线从外部与圆柱相交，$\text{t} = \text{x2}$。

考虑有顶面和底面的圆柱，设交点C与底面的距离为$\text{h} = \text{C} \cdot \text{u}$，若光线与圆柱面相交，h需满足$0 < \text{h} < |\text{O2} - \text{O1}|$。若满足，交点即为C。

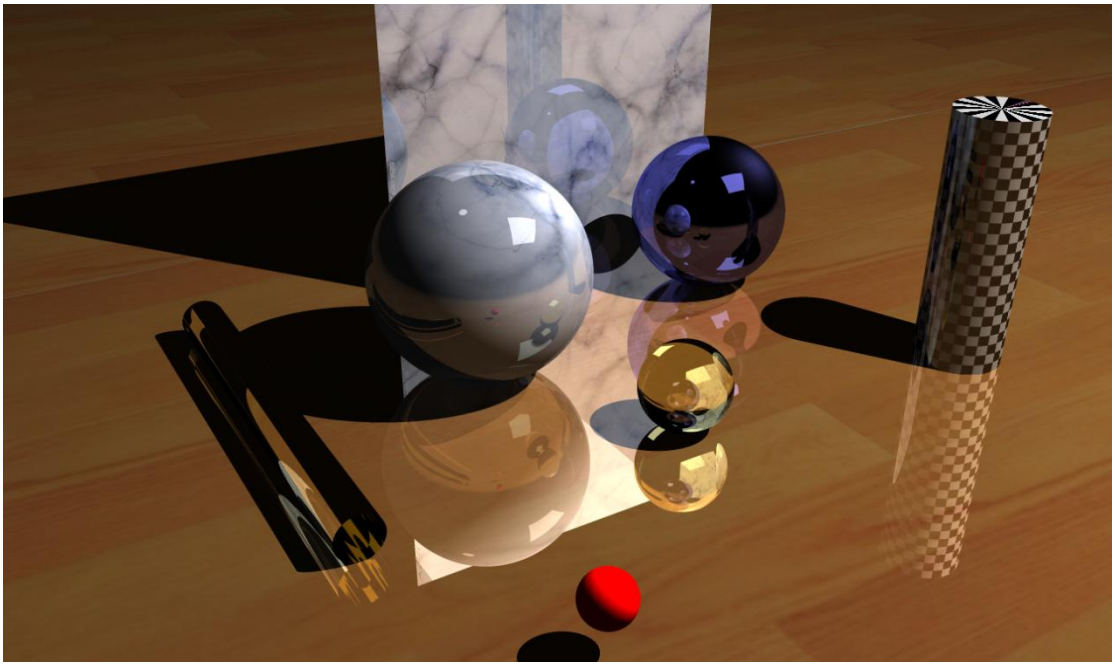
若不满足，可能光线与顶面或底面相交。若$\text{h} < 0$，可能与底面相交，基于光线与平面的相交求出交点C′。若C′距O1距离大于R，则无交点，返回默认值的 ret;若距离小于R，交点即为C′。若$\text{h} > |\text{O2} - \text{O1}|$，可能与顶面相交，同理。注意与顶面和底面相交时法向量垂直平面，方向视内部相交还是外部相交而定。

4. 获取交点在圆柱上的纹理

用$(O2-O1).GetAnVerticalVector()$获取一个垂直于圆柱轴向的单位向量vertical，以此垂直向量为圆柱面展开图的左下角，将圆柱面展开。交点C与底面的距离为h，从轴指向交点C的单位向量为$(C-O1-h(O2-O1).GetUnitVector()).GetUnitVector()$，此单位向量与vertical之间的夹角除以2π即为纹理坐标的v（由于 acos 函数只能求出[0,π]之间的角度，则需要通过$\big((C-O1)*(O2-O1).GetUnitVector()\big)\cdot$ vertical的正负来计算角度的范围）。纹理坐标的$u=h/|O2-O1|$。效果如下：

- **实验效果**

  在原场景的基础上加入一个矩形和两个圆柱的效果如下：



矩形和圆柱的参数为：

primitive square

O= 1 7 -1

Dx= 1 0 0

Dy= 0 0 1

color= 1 1 1

texture= marble.bmp

```
        diff= 0.45
        spec= 0.25
        refl= 0.3
        drefl= 0.25
        blur= exp
end

primitive cylinder
        O1= 2 5 -2
                        O2= 2 5 -0.8
        R= 0.2
        color= 1 1 1
        texture= blackwhite.bmp
        De= 0 0 1
        Dc= 0 1 0
        diff= 0.45
        spec= 0.25
        refl= 0.3
        drefl= 0.25
        blur= exp
end

primitive cylinder
        O1= -1 5 -1.9
                        O2= -1 7 -1.9
        R= 0.1
        refr= 1
        rindex= 1.7
        absor= 0 0 1
end
```