

Sobel Fragment Shader

梁晨 3180102160

- 实验环境

本课程的所有作业均在macOS Catalina 10.15.7上运用集成开发平台XCode 12.0.1完成。我在文件中附上了XCode的原始工程文件。C++相关源码在与最上层文件夹同名的文件夹中，而GLSL源码（即各个shader）在Debug文件夹中。编译生成的在macOS上可执行的UNIX可执行文件和建模过程中需要用到的.txt, .obj, 以及. tga（相关纹理）文件，也都在Debug文件夹中。渲染的结果实时、动态地显示在GLUT窗口中，我将渲染结果做了截图，统一放进output文件夹中。

- Sobel Fragment Shader的实现

Sobel Fragment可以说也是一种画轮廓线的方法，但其是通过对纹理求梯度来确定轮廓线的，即我们只对梯度足够大的部分进行绘制，rgb每个分量都如是，所以我们可以从绘制的图片中直观地看到颜色的梯度。这种方法在计算机视觉领域有着广泛的应用。其本质是用一个kernel对图片做卷积，估算其x方向、y方向的梯度：

$$G_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} * \text{Image}, \quad G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * \text{Image}$$
$$\text{Gradient} = \sqrt{G_x^2 + G_y^2}.$$

代码如下：

```
uniform sampler2D Texture;
uniform float width;
uniform float height;

varying vec2 pos;

void main()
{
    vec4 kernel[9];
    //求出步长
    float w=1.0/width;
    float h=1.0/height;
    //取当前像素的上下左右相邻8个像素
    for(int i=0;i<9;i++) {
```

```

    float xcoe=-1.0;
    float ycoe=-1.0;
    if(i-3*(i/3)==1) xcoe=0.0;
    else if(i-3*(i/3)==2) xcoe=1.0;
    if(i/3==1) ycoe=0.0;
    else if(i/3==2) ycoe=1.0;
    kernel[i]=texture2D(Texture, pos+vec2(xcoe*w, ycoe*h));
}
//计算梯度
vec4 Gx=kernel[0]+(2.0*kernel[3])+kernel[6]-(kernel[2]+
(2.0*kernel[5])+kernel[8]);
vec4 Gy=kernel[6]+(2.0*kernel[7])+kernel[8]-(kernel[0]+
(2.0*kernel[1])+kernel[2]);
vec4 G= sqrt((Gx*Gx)+(Gy*Gy));
//自己加的优化，如果梯度小于某个值，则像素颜色为纯黑。
//if(length(G)<=4.0) G=vec4(0.0,0.0,0.0,1.0);
gl_FragColor=G;
}

```

同时，我们还需要在openGL的代码中实现纹理的导入和uniform变量的导入，具体代码如下：

```

void loadTexture(char* tex, GLuint p)
{
    //读入纹理图片
    img=Image::LoadTGA(tex);
    int w=img->Width();
    int h=img->Height();
    data=new float[w*h*3];
    for(int i=0;i<h;i++)
        for(int j=0;j<w;j++){
            data[3*(i*w+j)]=img->GetPixel(j,i).r();
            data[3*(i*w+j)+1]=img->GetPixel(j,i).g();
            data[3*(i*w+j)+2]=img->GetPixel(j,i).b();
        }
    //绑定纹理
    glActiveTexture(GL_TEXTURE0);
    glGenTextures(1,&texID);
    glBindTexture(GL_TEXTURE_2D,texID);
    //载入纹理变量texture
    glUniform1i(glGetUniformLocation(p, "texture"),0);
    //对纹理做初始化
    glTexImage2D(GL_TEXTURE_2D, 0,GL_RGB, w, h, 0, GL_RGB, GL_FLOAT, data);
    glGenerateMipmap(GL_TEXTURE_2D);
    glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_LINEAR);
    //载入uniform变量width和height
    float wf=1.0*w;
    float hf=1.0*h;
}

```

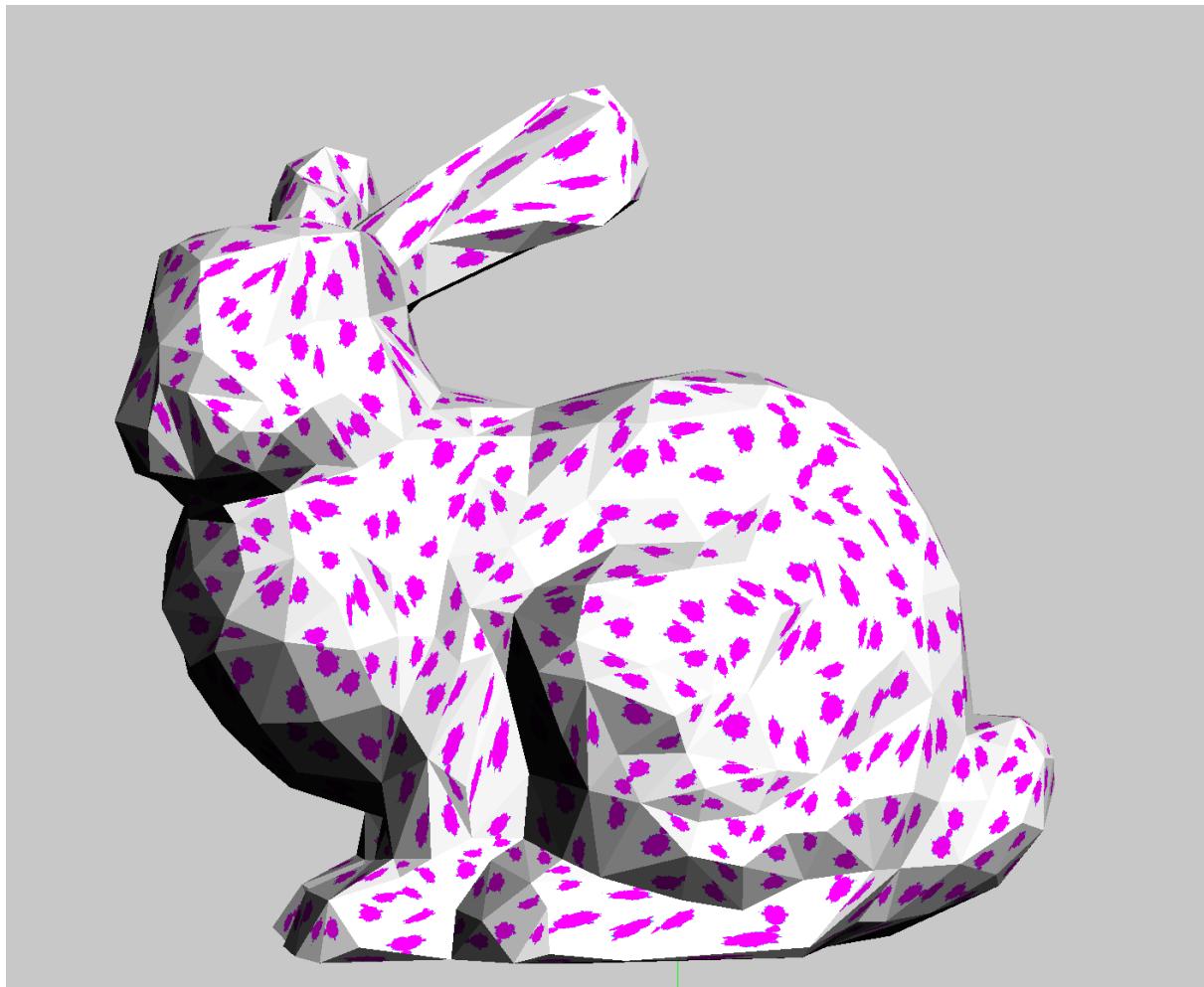
```
glUniform1f(glGetUniformLocation(p, "width"),wf);
glUniform1f(glGetUniformLocation(p, "height"),hf);
}
```

这次我们传参数时，要加上纹理的相关参数：

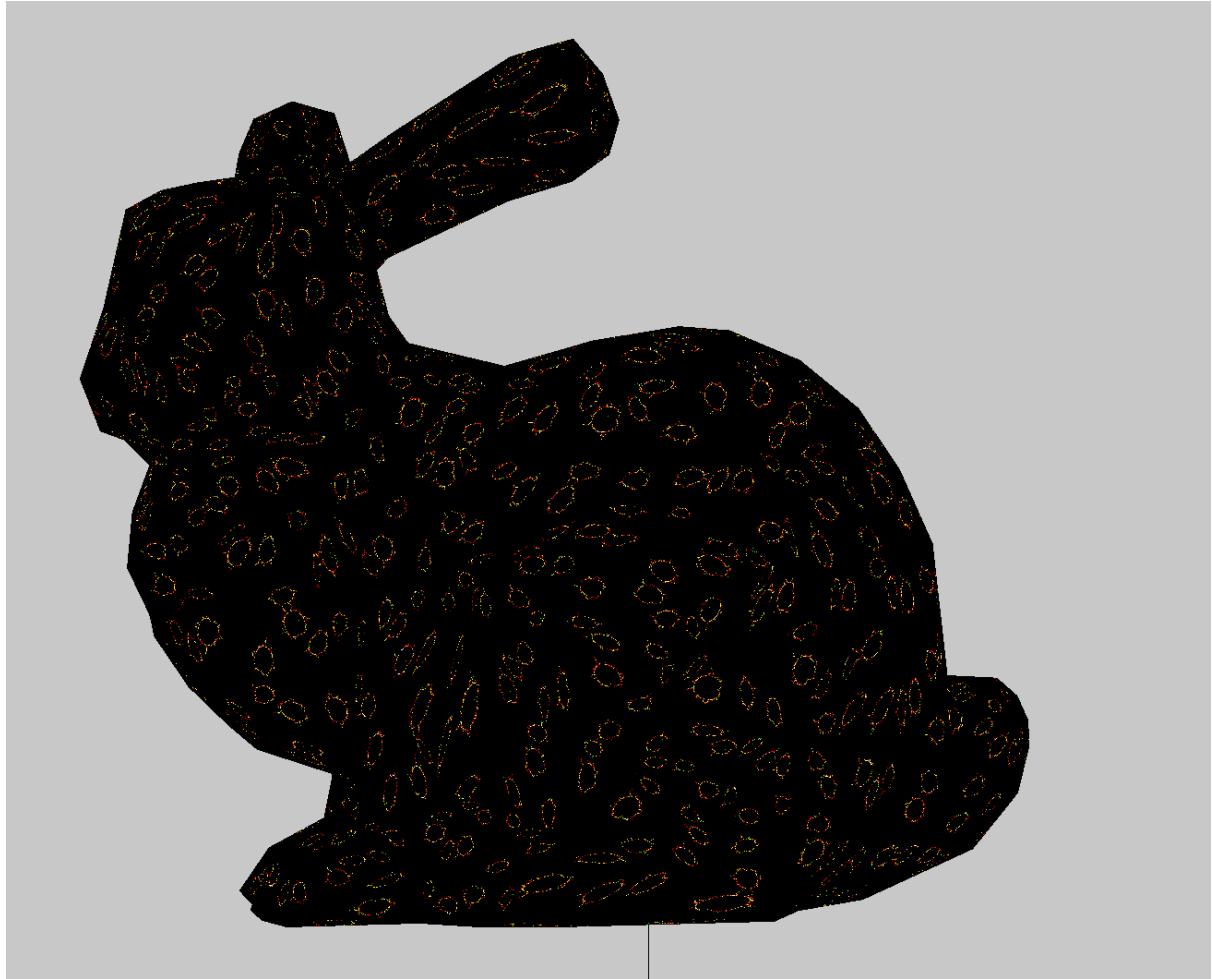
```
-input bunny.txt -vs minimal.vert -fs sobel.frag -texture tex.tga
```

- 实验结果：

对于较为稀疏的纹理，效果较好，我们可以看到原图如下：



用了sobel filter后，效果如下：

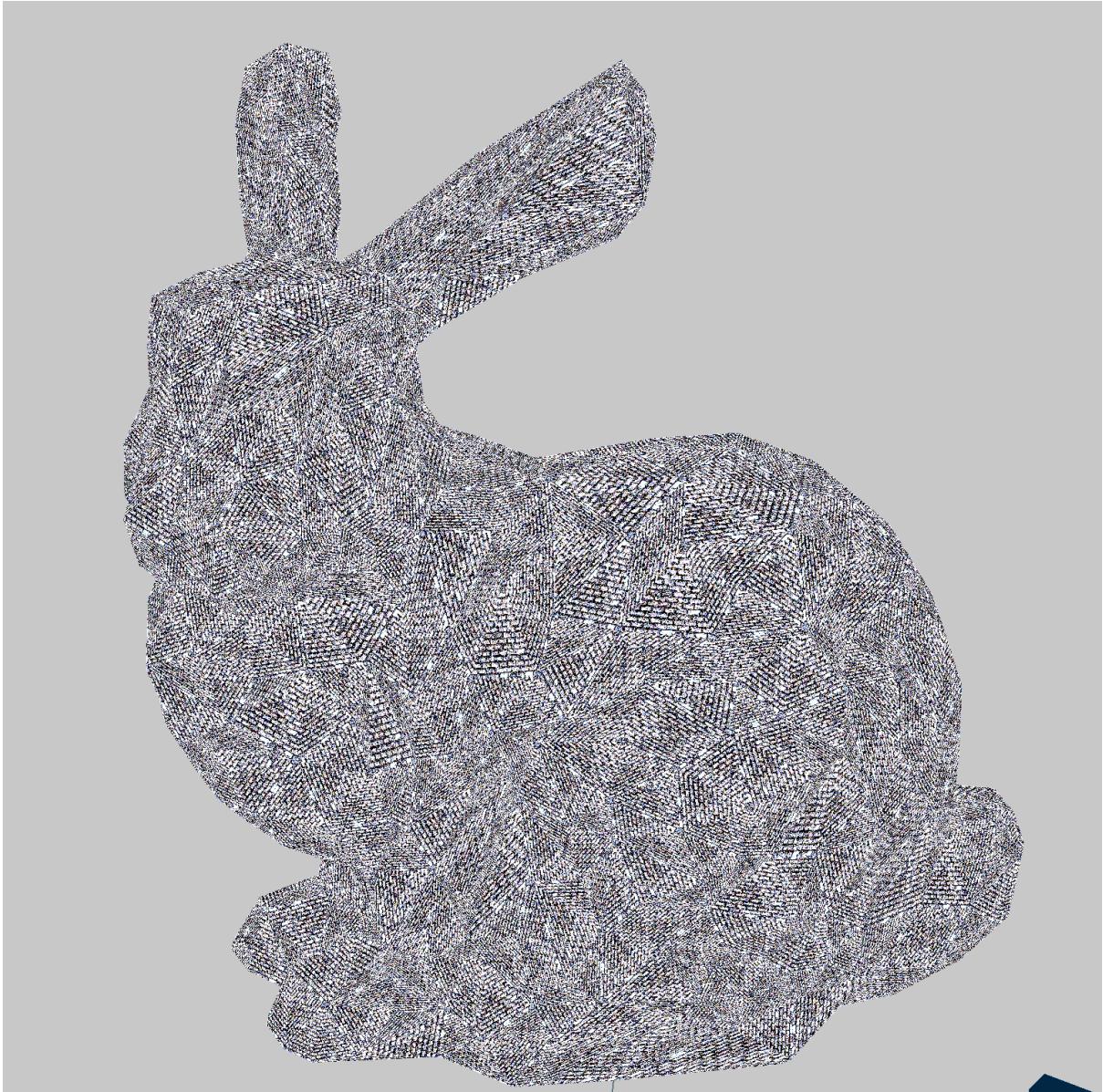


可以看到，只有颜色变化较明显的边界被画出，并且从边界的颜色中可以看出，不同部分颜色梯度不同。

但对于稠密的纹理，sobel filter工作得不是很好，稠密纹理原图如下：



进行sobel filter后，效果如下：



我们可以看到，由于纹理稠密，梯度很大的像素也更为稠密，而且各颜色梯度的区别不明显，过滤出来的图与原图区别不大，且太过稠密，容易密恐...为了获得一个较为稀疏的效果，我自己加了一个优化，即设定一个阈值，如果梯度小于某个阈值，则像素设置为黑色，优化后效果如下：

