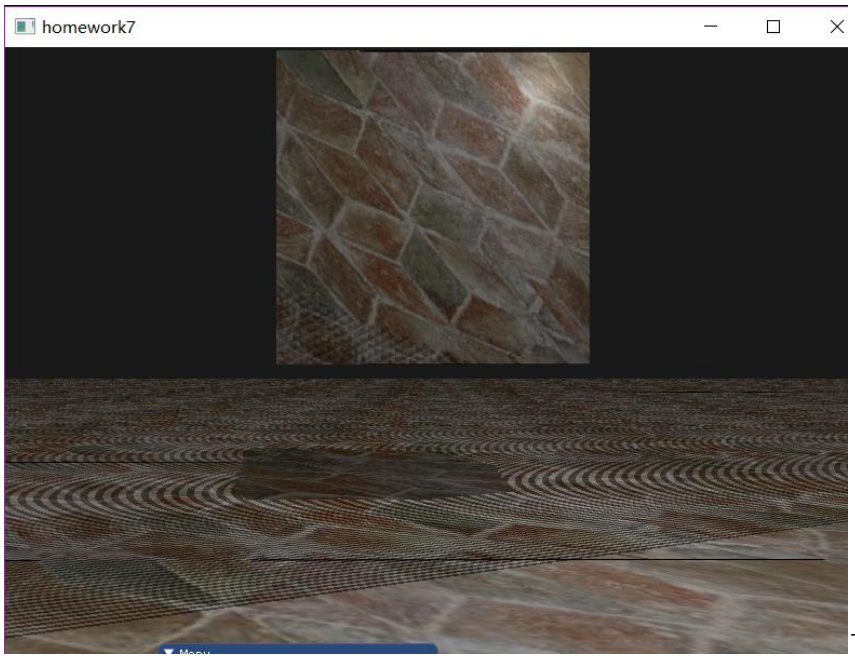
1. 实现方向光源的 shadowing mapping
2. 修改 GUI
3. 实现正交/透视两种投影下的 shadowing Mapping
4. 优化 Shadowing Mapping 算法

Step1：生成一张深度贴图，深度贴图是从光的透视图处理得到的深度纹理（把世界坐标转换到光源“摄像机”坐标，求得其 Zbuffer，生成一张图）。
这个算法的本质是光路可逆。
Step2：计算阴影。在这张图中，最小的深度就是被光线所看到（照射到）的地方。而大于这一部分的都是阴影。

```
float ShadowCalculation(vec4 fragPosLightSpace)
{
    vec3 projCoords = fragPosLightSpace.xyz / fragPosLightSpace.w;
    //归一化
    projCoords = projCoords * 0.5 + 0.5;
    //获得光照坐标系下的最近的深度值
    float closestDepth = texture(shadowMap, projCoords.xy).r;
    //获得当前片段在光照下的深度值
    float currentDepth = projCoords.z;

    //注意此时对应的大概是正交投影??

    vec3 normal = normalize(fs_in.Normal);//得到当前片元的法向量
    vec3 lightDir = normalize(lightPos - fs_in.FragPos);//得到入射方向
    float bias = max(0.05 * (1.0 - dot(normal, lightDir)), 0.005);//平移

    float shadow = 0.0;//PCF
    vec2 texelSize = 1.0 / textureSize(shadowMap, 0);
    for(int x = -1; x <= 1; ++x)//中值滤波卷积
    {
        for(int y = -1; y <= 1; ++y)
        {
            float pcfDepth = texture(shadowMap, projCoords.xy + vec2(x, y) * texelSize).r;
            shadow += currentDepth - bias > pcfDepth  ? 1.0 : 0.0;//比较两个深度值谁大，大的是阴影
        }
    }
    shadow /= 9.0;//去中值

    if(flag>0&&projCoords.z > 1.0)
        shadow = 0.0;
    return shadow;
}

void main()
{
    vec3 color = texture(diffuseTexture, fs_in.TexCoords).rgb;
    vec3 normal = normalize(fs_in.Normal);
    vec3 lightColor = vec3(0.3);

    vec3 ambient = 0.3 * color;//环境光
    vec3 lightDir = normalize(lightPos - fs_in.FragPos);//漫反射
    float diff = max(dot(lightDir, normal), 0.0);
    vec3 diffuse = diff * lightColor;//

    vec3 viewDir = normalize(viewPos - fs_in.FragPos);
    vec3 reflectDir = reflect(-lightDir, normal);//镜面反射
    float spec = 0.0;
    vec3 halfwayDir = normalize(lightDir + viewDir);
    spec = pow(max(dot(normal, halfwayDir), 0.0), 64.0);
    vec3 specular = spec * lightColor;

    float shadow = ShadowCalculation(fs_in.FragPosLightSpace);//获得阴影值

    vec3 lighting = (ambient + (1.0 - shadow) * (diffuse + specular)) * color;//光照模型 注意是（1-阴影），因为阴影等效于对光照的削弱，而且不影响环境光
    FragColor = vec4(lighting, 1.0);
}
```

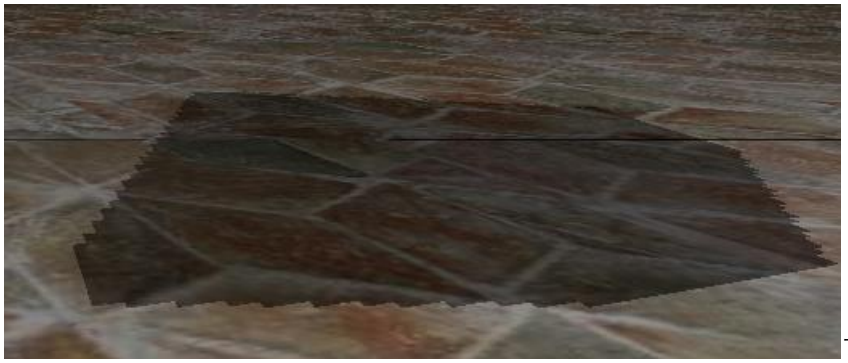


可以看到此时有明显的纹路。这是由于取样非连续导致的。

Step3：增加 GUI 部分，加入 GUI 的切换光源的正交视图和透视视图用来生成纹理。

Step4：对 shadow Mapping 进行改善：增加 bias 解决条纹问题。同时引入了新的问题


悬浮问题


可以看到此时锯齿依然比较严重，所以需要进行中值滤波

Step5：对 shadow Mapping 进行改善：对阴影进行“中值滤波”，重新采样，淡化边缘


Menu
Average 16.681 ms/frame (59.9 FPS
-4.000
light x
4.000
light y
-1.000
light z
IsPerspective