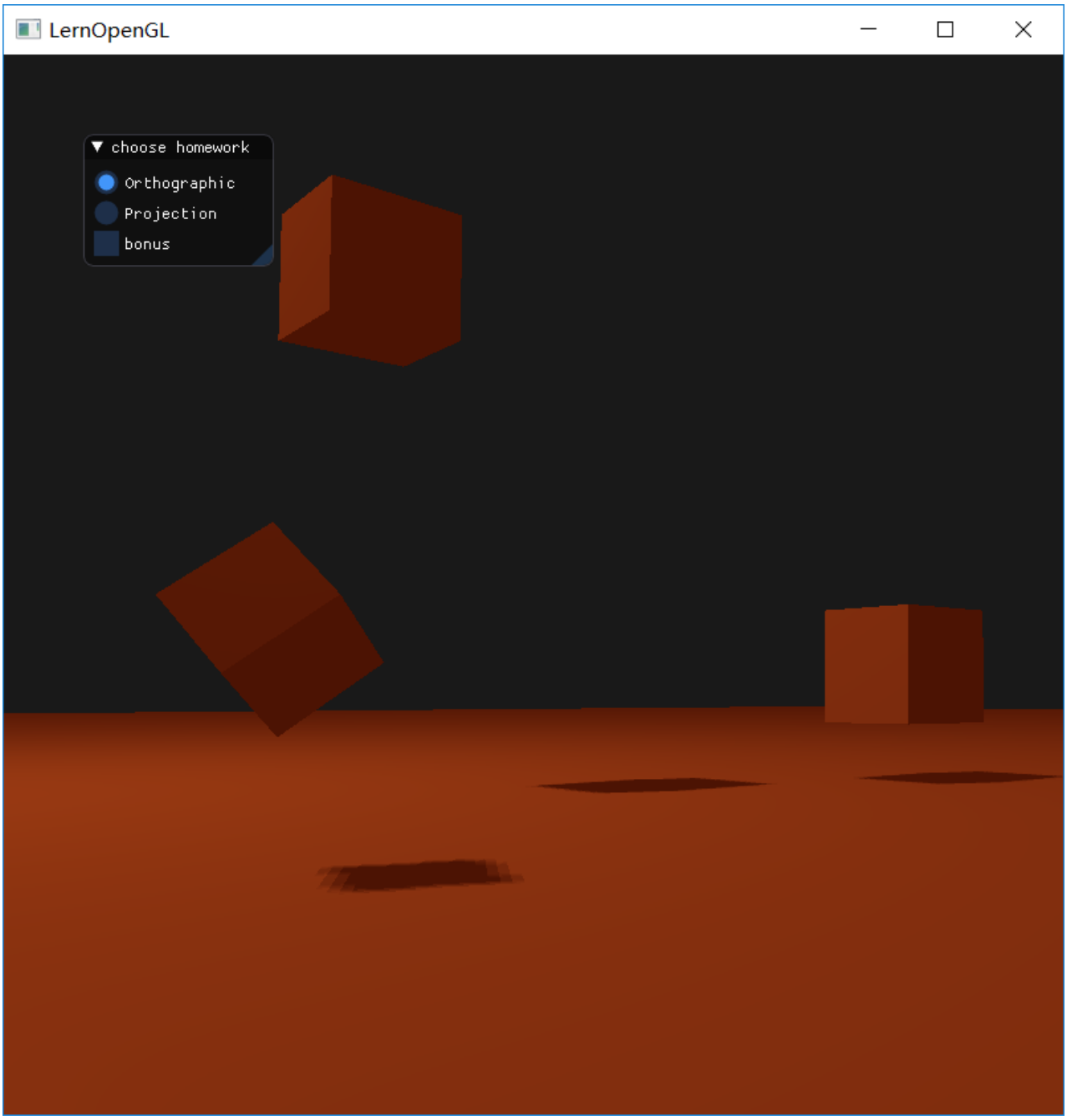
实验报告

Basic

Shadow Mapping

1. 实验结果



2. 实现原理
    1. 渲染深度贴图：用来计算某个片段是否在阴影里。即将视角变成光源的视角，重新判断各个部分的深度值（遮挡与被遮挡）。

1. 创建帧缓冲对象：

```
unsigned int depthMapFBO;
glGenFramebuffers(1, &depthMapFBO);
```

2. 创建2D纹理，给帧缓冲的深度缓冲使用

```
unsigned int depthMap;
glGenTextures(1, &depthMap);
glBindTexture(GL_TEXTURE_2D, depthMap);
glTexImage2D(GL_TEXTURE_2D, 0, GL_DEPTH_COMPONENT, SHADOW_WIDTH,
SHADOW_HEIGHT, 0, GL_DEPTH_COMPONENT, GL_FLOAT, NULL);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_BORDER);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_BORDER);
```

（其中，`GL_CLAMP_TO_BORDER` 是用于防止采样过多）

3. 把深度纹理作为帧缓冲的深度缓冲，同时不需要渲染颜色缓冲

```
glBindFramebuffer(GL_FRAMEBUFFER, depthMapFBO);
glFramebufferTexture2D(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, GL_TEXTURE_2D,
depthMap, 0);
glDrawBuffer(GL_NONE);
glReadBuffer(GL_NONE);
glBindFramebuffer(GL_FRAMEBUFFER, 0);
```

4. 光源空间的变换：用于判断哪些部分应该处于阴影处，即用光源的视角观察物体，找出被遮挡的部分即阴影。
    1. 投影矩阵：这里使用正交投影

    ```
    float nearPlane = 1.0f, farPlane = 7.5f;
    lightProjection = glm::ortho(-10.0f, 10.0f, -10.0f, 10.0f, nearPlane,
    farPlane);
    ```

    2. 视图矩阵：即光的方向，这里设置为从光源的位置看向场景中央

    ```
    lightView = glm::lookAt(lightPos, glm::vec3(0.0f), glm::vec3(0.0, 1.0,
    0.0));
    ```

    3. 光源空间变换矩阵：通过透视矩阵*视图矩阵获得

    ```
    lightSpaceMatrix = lightProjection * lightView;
    ```

5. 渲染到深度贴图：将原来的内容转换到光源空间
    1. 顶点着色器：将顶点转换为光源空间

```
#version 330 core
layout (location = 0) in vec3 position;

uniform mat4 lightSpaceMatrix;
uniform mat4 model;

void main()
{
    gl_Position = lightSpaceMatrix * model * vec4(position, 1.0f);
}
```

2. 片段着色器：只需要一个空着色器就好

```
#version 330 core

void main()
{
}
```

6. 将深度贴图渲染到场景中

1. 顶点着色器：

```
#version 330 core
layout (location = 0) in vec2 position;
layout (location = 1) in vec2 texCoords;

out vec2 TexCoords;

void main()
{
    gl_Position = vec4(position, 0.0f, 1.0f);
    TexCoords = texCoords;
}
```

2. 片段着色器

```
#version 330 core
out vec4 FragColor;
in  vec2 TexCoords;

uniform sampler2D fboAttachment;

void main()
{
    FragColor = vec4(texture(fboAttachment, TexCoords).r, 1.0);

}
```

2. 生成阴影

1. 检验那一块是否在阴影中：

1. 顶点着色器：将世界空间的定点位置转换成光空间，传递世界空间的顶点位置、世界空间的法向量、颜色和光空间的顶点位置给片段着色器

```
#version 330 core
layout (location = 0) in vec3 aPos;
layout (location = 1) in vec3 aNormal;
layout (location = 2) in vec3 aColor;

out vec3 FragPos;
out vec3 Normal;
out vec3 color;
out vec4 FragPosLightSpace;

uniform mat4 projection;
uniform mat4 view;
uniform mat4 model;
uniform mat4 lightSpaceMatrix;

void main()
{
    FragPos = vec3(model * vec4(aPos, 1.0));
    Normal = transpose(inverse(mat3(model))) * aNormal;
    color = aColor;
    FragPosLightSpace = lightSpaceMatrix * vec4(FragPos, 1.0);
    gl_Position = projection * view * model * vec4(aPos, 1.0);
}
```

2. 片段着色器：和之前的光照一样计算光照，其中这里用了binn-phong光照模型，（视角向量+光入射向量）·法向量，通过函数`ShadowCalculation`计算是否处于阴影处，如果处于阴影处，该处仅仅展现环境光结果，如果不在阴影处则是漫反射+镜面反射+环境光

```
#version 330 core
out vec4 FragColor;

in vec3 FragPos;
in vec3 Normal;
in vec3 color;
in vec4 FragPosLightSpace;


uniform sampler2D diffuseTexture;
uniform sampler2D shadowMap;

uniform vec3 lightPos;
uniform vec3 viewPos;

float ShadowCalculation(vec4 fragPosLightSpace)
{
    //透视除法
    vec3 projCoords = fragPosLightSpace.xyz / fragPosLightSpace.w;
    //0-1范围
    projCoords = projCoords * 0.5 + 0.5;
```

```
    //最近点的深度
    float closestDepth = texture(shadowMap, projCoords.xy).r;
    //光源视角的深度
    float currentDepth = projCoords.z;
    vec3 normal = normalize(Normal);
    vec3 lightDir = normalize(lightPos - FragPos);
    //检查是否在阴影中
    float shadow = currentDepth > closestDepth  ? 1.0 : 0.0;
    return shadow;
}

void main()
{
    vec3 color = color;
    vec3 normal = normalize(Normal);
    vec3 lightColor = vec3(0.3);
    // ambient
    vec3 ambient = 0.3 * color;
    // diffuse
    vec3 lightDir = normalize(lightPos - FragPos);
    float diff = max(dot(lightDir, normal), 0.0);
    vec3 diffuse = diff * lightColor;
    // specular
    vec3 viewDir = normalize(viewPos - FragPos);
    vec3 reflectDir = reflect(-lightDir, normal);
    float spec = 0.0;
    vec3 halfwayDir = normalize(lightDir + viewDir);
    spec = pow(max(dot(normal, halfwayDir), 0.0), 64.0);
    vec3 specular = spec * lightColor;
    // calculate shadow
    float shadow = ShadowCalculation(FragPosLightSpace);

    vec3 lighting = (ambient + (1.0 - shadow) * (diffuse + specular)) *
color;

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

计算是否为阴影的方法：

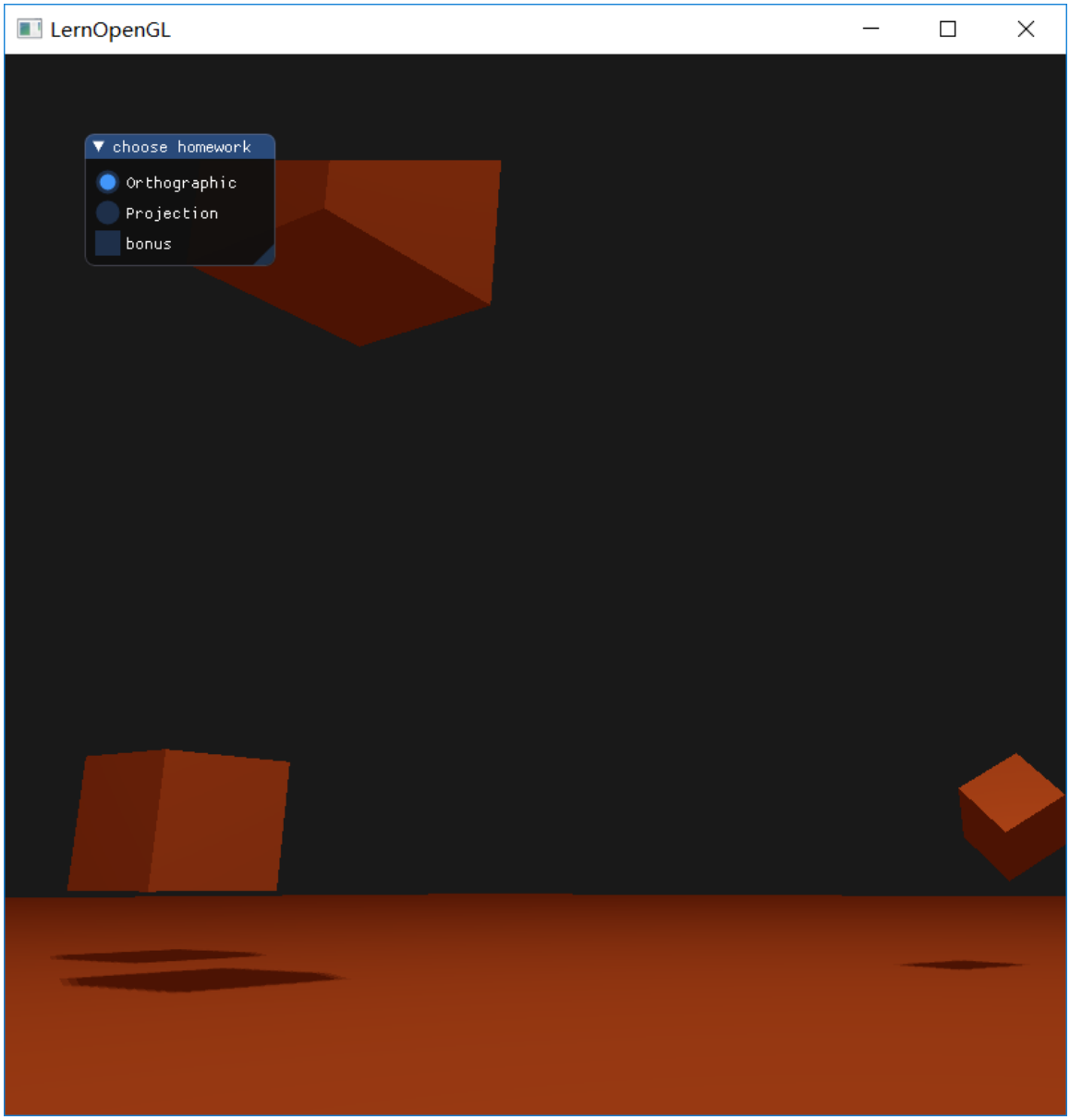
1. 将位置转换为设备坐标，用于对于帧缓存：将`fragPosLightSpace`的xyz元素处以w元素
2. 将它从[-1,1]范围转换为[0,1]范围
3. 对于这个设备坐标，获得帧缓存中的内容，即得到光的位置视野下，最近的深度。
4. 获得当前位置的深度
5. 如果当前深度高于最近深度，则处于阴影中。

3. 画出平面、立方体，对平面和立方体依次使用深度贴图着色器和阴影着色器渲染

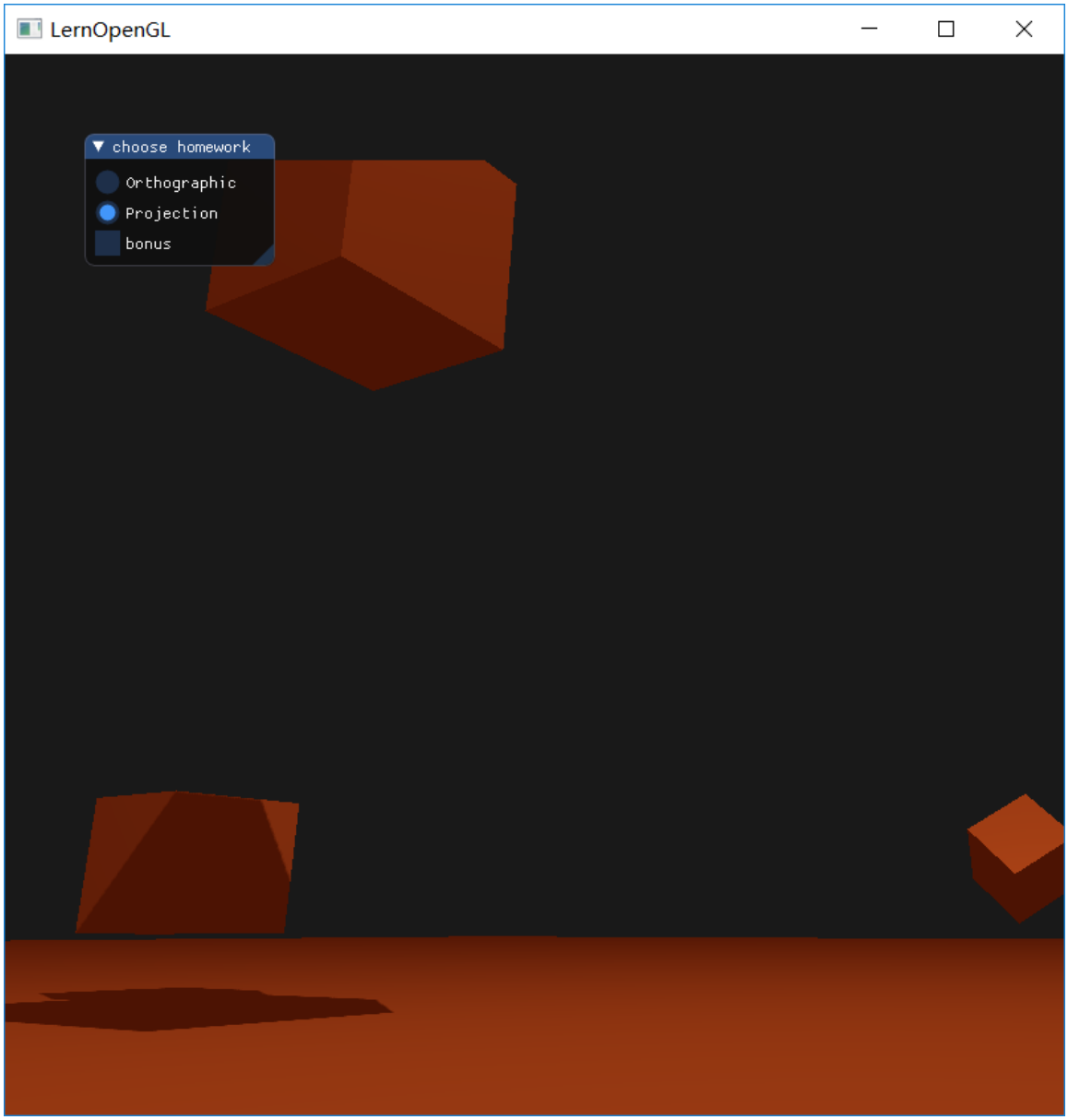
Bonus

1. 正交和透视
    1. 实验结果

1. 正交


LernOpenGL
choose homework
Orthographic
Projection
bonus

2. 透视



2. 实现原理

    1. 在光源空间的变换里面，将透视矩阵换成：

```
lightProjection = glm::perspective(glm::radians(45.0f), (GLfloat)SHADOW_WIDTH
/ (GLfloat)SHADOW_HEIGHT, nearPlane, farPlane);
```

    2. 透视投影下，深度变成非线性深度，因此需要将非线性的变成线性的

```
float LinearizeDepth(float depth)
{
    float z = depth * 2.0 - 1.0; // Back to NDC
    return (2.0 * near_plane * far_plane) / (far_plane + near_plane - z *
(far_plane - near_plane));
}
```

2. 阴影失真：由于在距离光源很远的地方，多个片元会从深度贴图中的一个值中采样，会出现当光以同一个角度射向斜面的时候，有时候取样到的深度实际上在斜面以下的情况（即实际上深度应该随斜面上升，但是由于取值一样，就处于斜面以下），因此对取到的深度值减去一个偏移量即可。

```
float bias = max(0.05 * (1.0 - dot(normal, lightDir)), 0.005);
```

3. 采样过多：让超出深度贴图的坐标的深度范围是1.0，即这些部分并不会处于阴影中，防止默认将超出深度贴图坐标的部分全部设为阴影导致不真实的效果，同时，将比远平面还远的部分也设置成为没有阴影的。

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_BORDER);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_BORDER);
float borderColor[] = { 1.0, 1.0, 1.0, 1.0 };
glTexParameterfv(GL_TEXTURE_2D, GL_TEXTURE_BORDER_COLOR, borderColor);
```

4. PCF：不仅仅对于该像素的纹理进行采样，同时对于附近的都进行采样，再取平均，这样就会出现处于0-1中间的阴影值，有过渡的效果

```
float shadow = 0.0;
vec2 texelSize = 1.0 / textureSize(shadowMap, 0);
for(int x = -1; x <= 1; ++x)
{
    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;
    }
}
shadow /= 9.0;
```