实验：阴影

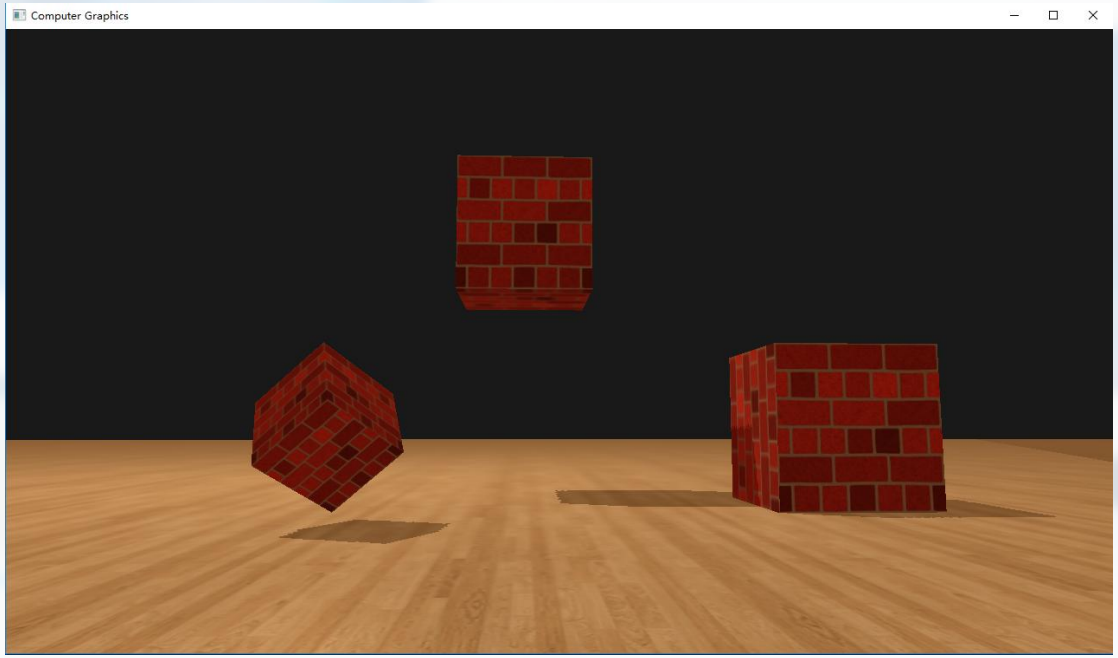
华中科技大学软件学院　万琳

提纲

1. 实验要求
2. 程序流程
3. 要点解析

1 实验要求

要求：基于shadow mapping实现实时动态阴影。



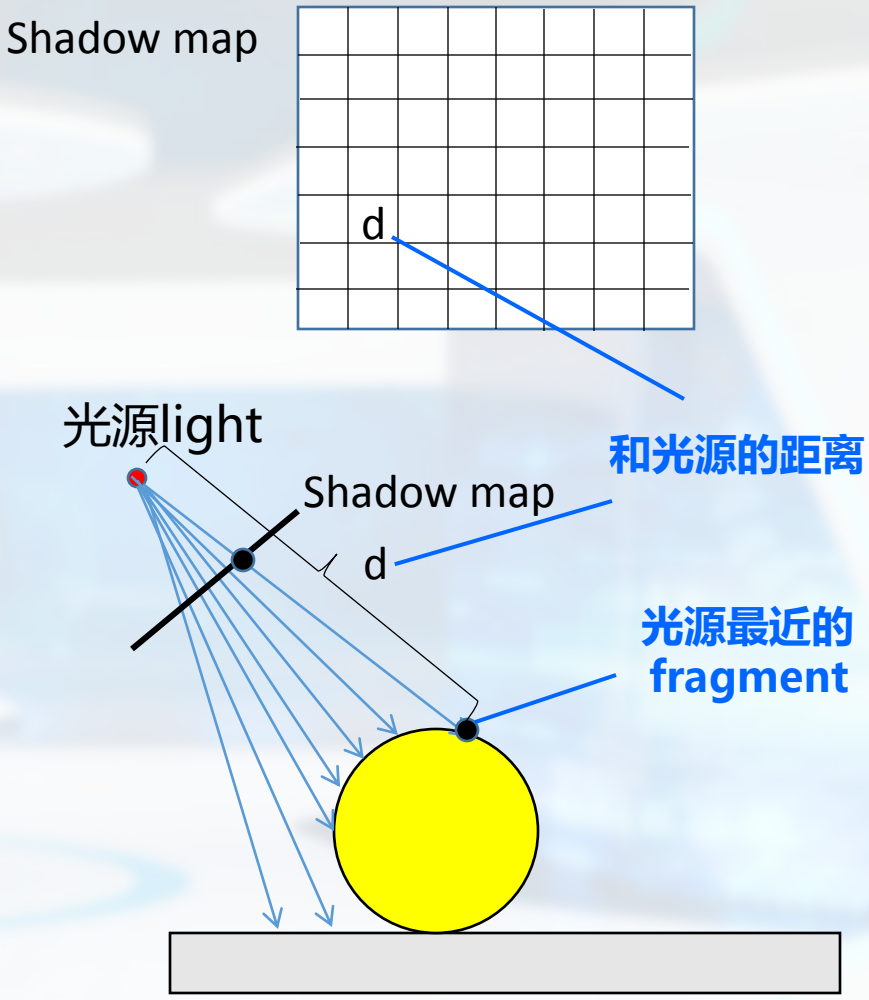
3 要点解析

➢问题分析

基于Shadow Mapping阴影计算算法回顾：

Step 1: 以光源为视点，或者说在光源坐标系下面对整个场景进行渲染，目的是要得到一副所有物体相对于光源的 depth map（也就是我们所说的shadow map），也就是这副图像中每个象素的值代表着场景里面离光源最近的 fragment 的深度值。

由于这个部分我们感兴趣的只是象素的深度值，所以可以把所有的光照计算关掉。

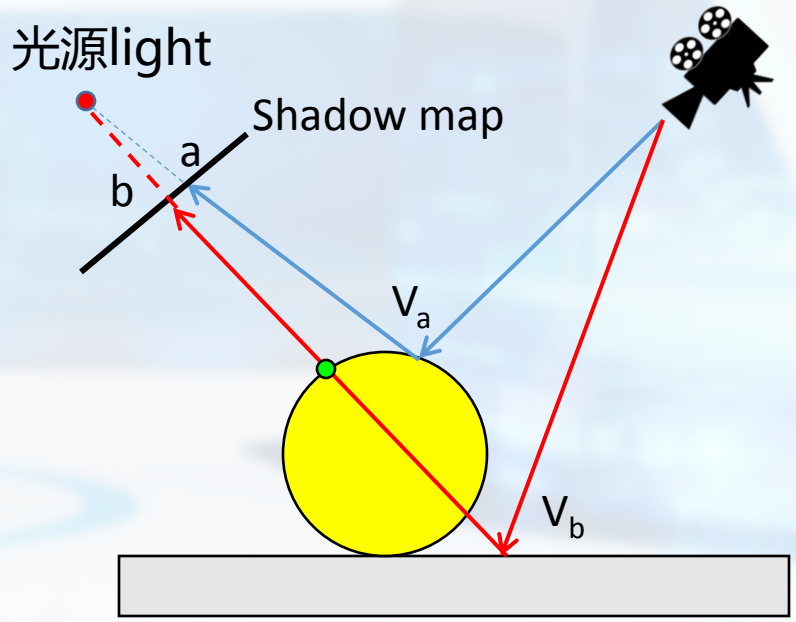


3 要点解析

➢问题分析

基于Shadow Mapping阴影计算算法回顾：

Step 2：将视点恢复到原来的正常位置，渲染整个场景，对每个象素计算它和光源的距离，然后将这个值和 depth map（shadow map）中相应的值比较，以确定这个象素点是否处在阴影当中。然后根据比较的结果，对 shadowed fragment（有阴影的片元）和 lighted fragment（有光照的片元）分别进行不同的光照计算，这样就可以得到阴影的效果了。



3 要点解析

➢问题分析

FBO

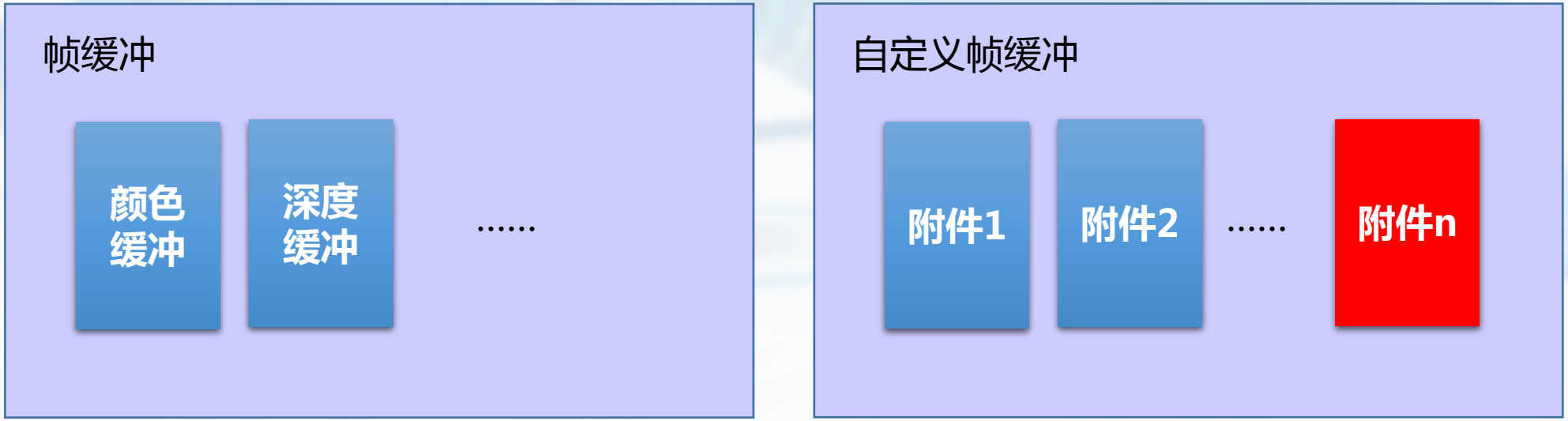
阴影

3 要点解析

➢问题一：FBO

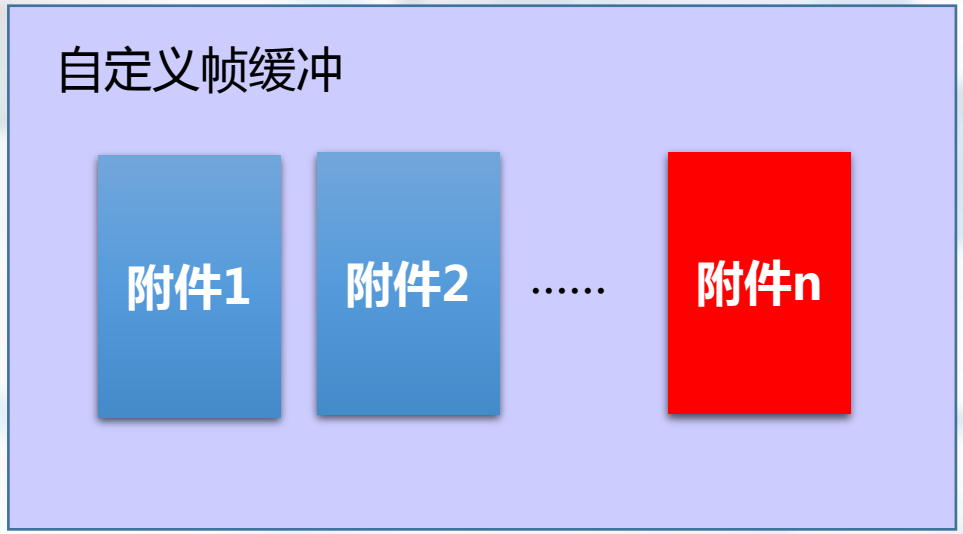
Frame Buffer Obiect（简称FBO）

创建窗口的过程中形成的颜色缓冲、深度缓冲等，外加自己生成这些缓冲的组合被称为帧缓冲。



3 要点解析

➢问题一：FBO



一个完整的帧缓冲需要满足以下条件：

◆ 附加至少一个缓冲（颜色、深度或模板缓冲）

◆ 至少有一个颜色附件（Attachment）

◆所有的附件都必须是完整的（保留了内存）

◆ 每个缓冲都应该有相同的样本数

3 要点解析

➢问题一：FBO

使用FBO的流程：

3 要点解析

➢问题一：FBO

使用FBO的流程：

Step1

创建和绑定

```
//创建并绑定一个帧缓冲
unsigned int fbo;
glGenFramebuffers(1, &fbo);
glBindFramebuffer(GL_FRAMEBUFFER, fbo);
//检查帧缓冲的完整性
if(glCheckFramebufferStatus(GL_FRAMEBUFFER) ==
GL_FRAMEBUFFER_COMPLETE)
{   ……
}
```

3 要点解析

➢问题一：FBO

使用FBO的流程：

使用

3 要点解析

➢问题一：FBO

使用FBO的流程：

Step3

善后工作

//之后所有的渲染操作将会渲染到当前绑定帧缓冲的附件中，如果我们需要再次激活默认帧缓冲，将它绑定到0

```
glBindFramebuffer(GL_FRAMEBUFFER, 0);
```

//在完成所有的帧缓冲操作之后，不要忘记删除这个帧缓冲对象

```
glDeleteFramebuffers(1, &fbo);
```

3 要点解析

➢问题一：FBO

纹理附件：

3 要点解析

➢问题一：FBO

纹理附件：

创建纹理

为帧缓冲创建一个纹理和创建一个普通的纹理差不多：

```
unsigned int texture;
glGenTextures(1, &texture);
glBindTexture(GL_TEXTURE_2D, texture);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, 800, 600, 0, GL_RGB,
GL_UNSIGNED_BYTE, NULL);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
```

3 要点解析

➢问题一：FBO

纹理附件：

附加到帧缓冲

现在我们已经创建好一个纹理了，要做的最后一件事就是将它附加到帧缓冲上了：

```
glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D, texture, 0);
```

3 要点解析

- 问题一：FBO

  渲染缓冲对象附件：

3 要点解析

➢问题分析

FBO

阴影

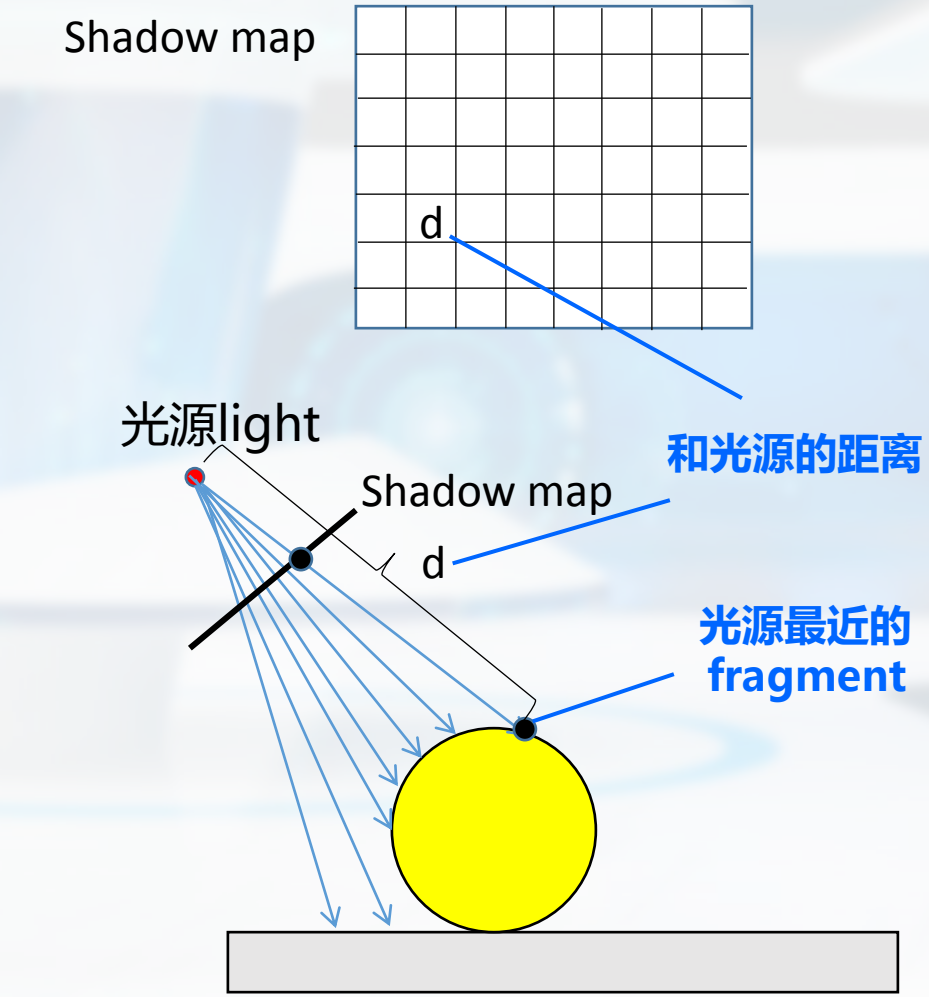
3 要点解析

➢问题二：阴影

3 要点解析

➢问题二：阴影

生成ShadowMap



3 要点解析

➢问题二：阴影

生成ShadowMap

首先，我们要为渲染的深度贴图创建一个帧缓冲对象：

```
// 设置离屏渲染帧缓冲(生成深度贴图)
GLuint depthMapFBO;
glGenFramebuffers(1, &depthMapFBO);
```

然后，创建一个2D纹理，提供给帧缓冲的深度缓冲使用：

```
GLuint 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_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
```

3 要点解析

➢问题二：阴影

生成 ShadowMap

渲染深度贴图：

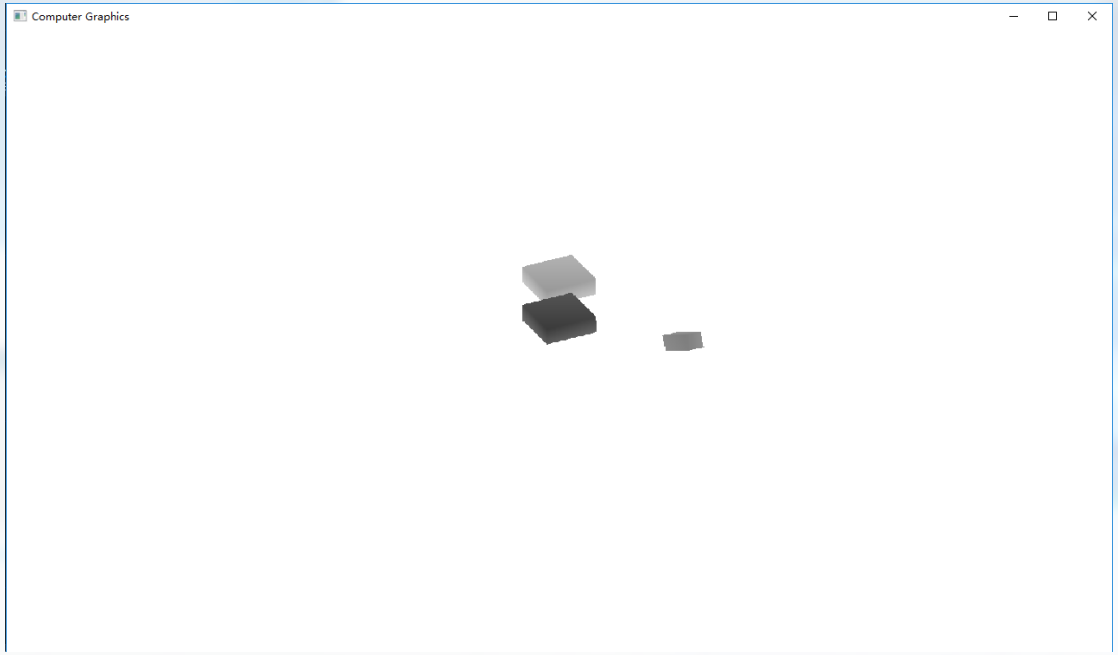
```
// 渲染阴影深度贴图
glBindFramebuffer(GL_FRAMEBUFFER, depthMapFBO);
    glViewport(0, 0, SHADOW_WIDTH, SHADOW_HEIGHT);
    glClear(GL_DEPTH_BUFFER_BIT);
    shadowMap_shader.Use();
    shadowMap_shader.SetMat4("lightPV", lightPV);
    DrawScene(shadowMap_shader, current_frame);
glBindFramebuffer(GL_FRAMEBUFFER, 0);
```

3 要点解析

➢问题二：阴影

生成ShadowMap

深度贴图的渲染结果：

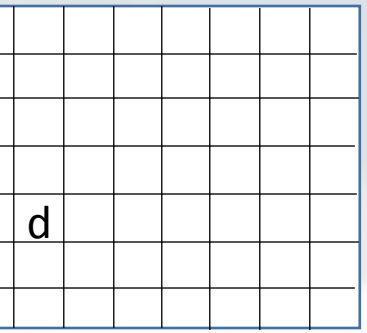


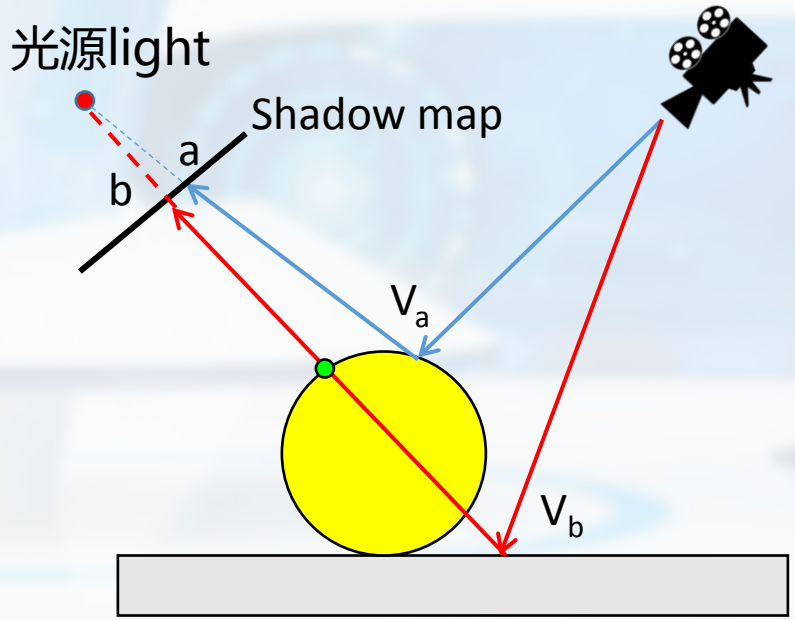
3 要点解析

➢问题二：阴影

阴影计算

Shadow map





3 要点解析

➢问题二：阴影

阴影计算


谢谢
软件学院　万琳