Real-time Shadows

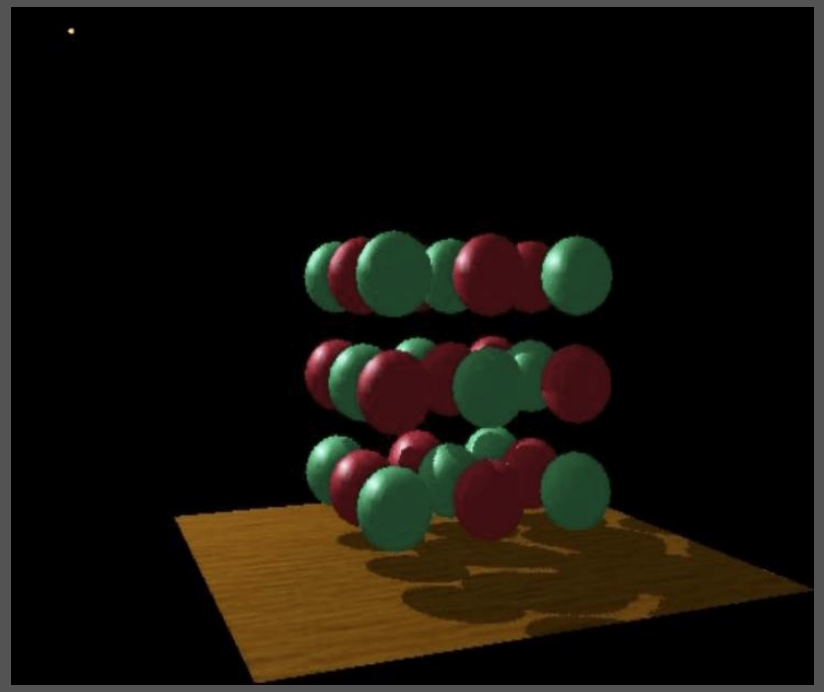
The Rendering Equation

$$L_o(p, w_o) = L_e(p, w_o) + \int_{H^2} f_r(p, w_i \rightarrow w_o) L_i(p, w_i) cos\theta_i d\omega_i$$

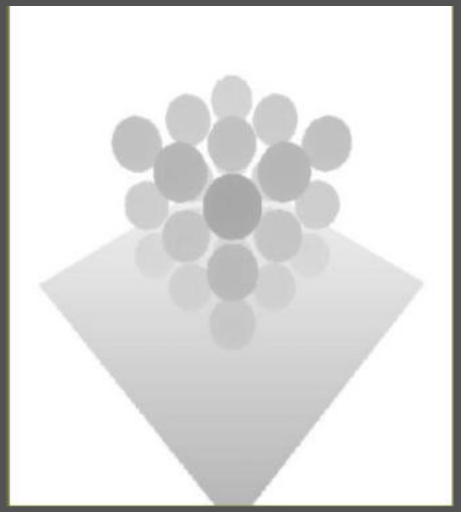
The Real-time Rendering Equation

$$L_o(p, w_o) = L_e(p, w_o) + \int_{\Omega^+} f_r(p, w_i, w_o) L_i(p, w_i) cos\theta_i V(p, w_i) d\omega_i$$

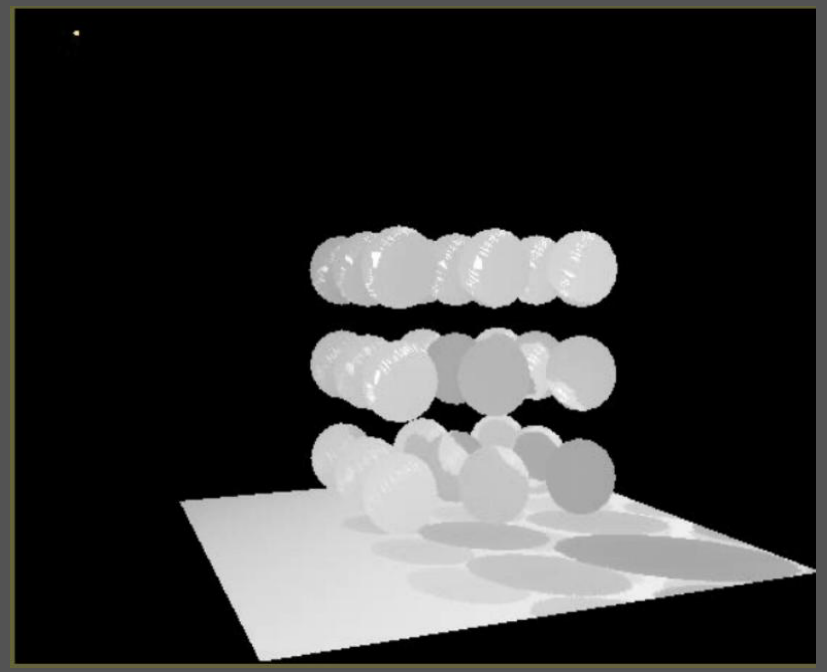
What is shadow map?



摄像机视角的彩色图



光源视角的灰度图



摄像机视角的灰度图

What is shadow map?

```
// Shadow pass
if (this.lights[l].entity.hasShadowMap == true) {
    for (let i = 0; i < this.shadowMeshes.length; i++) {
        this.shadowMeshes[i].draw(this.camera);
    }
}
// Camera pass
for (let i = 0; i < this.meshes.length; i++) {
    this.gl.useProgram(this.meshes[i].shader.program.glShaderProgram);
    this.gl.uniform3fv(this.meshes[i].shader.program.uniforms.uLightPos, this.lights[l].entity.lightPos);
    this.meshes[i].draw(this.camera);
}
```

How to use a shadow map?

我们在光源视角下渲染一张场景深度图，这张深度图保存了在光源视角下最前面的物体的深度，然后我们在渲染自己场景时，将在相机视角下的点转化到光源视角下（LookFromLight），再通过裁剪矩阵（Project）投影到视图上，此时我们可以知道场景中某一点在光源视角下应当对应那个位置的像素，通过比较之前得到的深度图对应位置像素的值，我们可以知道，当前视角下的这个点在光源位置来看，是否被遮挡住了？

如果遮挡住了，那么将V取为0，如果没有遮挡住，那么将V取为1，那么我们可以正确的得到阴影。

implementation details

因此，我们在实现细节上面临的主要问题就是，怎么把光源视角的深度图转换成摄像机视角的深度，然后再将两者的深度进行比较。即完成CalcLightMVP(translate, scale)函数。

Model Matrix

- 让顶点坐标从模型空间 → 世界空间
- 对顶点依次进行了：缩放→旋转→平移（注意：顺序问题，不能改变。）
  （矩阵的左乘/复合Transform的顺序很重要）

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & tx \\ 0 & 1 & 0 & ty \\ 0 & 0 & 1 & tz \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} kx & 0 & 0 & 0 \\ 0 & ky & 0 & 0 \\ 0 & 0 & kz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

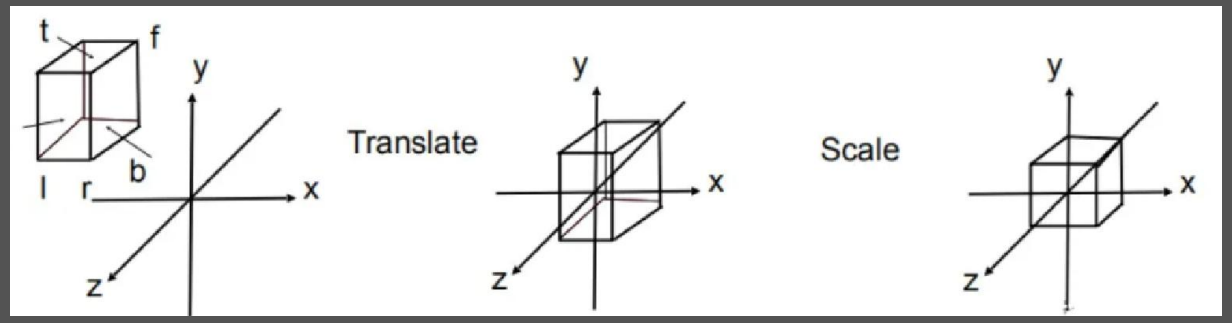
View Matrix

- 世界空间 → 视觉空间（以camera为中心的空间坐标系）
- 要想得到view矩阵，可以这样理解：把camera放在世界坐标的原点（逆过程），那么得到的矩阵的逆矩阵就是V矩阵

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & tx \\ 0 & 1 & 0 & ty \\ 0 & 0 & 1 & tz \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} kx & 0 & 0 & 0 \\ 0 & ky & 0 & 0 \\ 0 & 0 & kz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

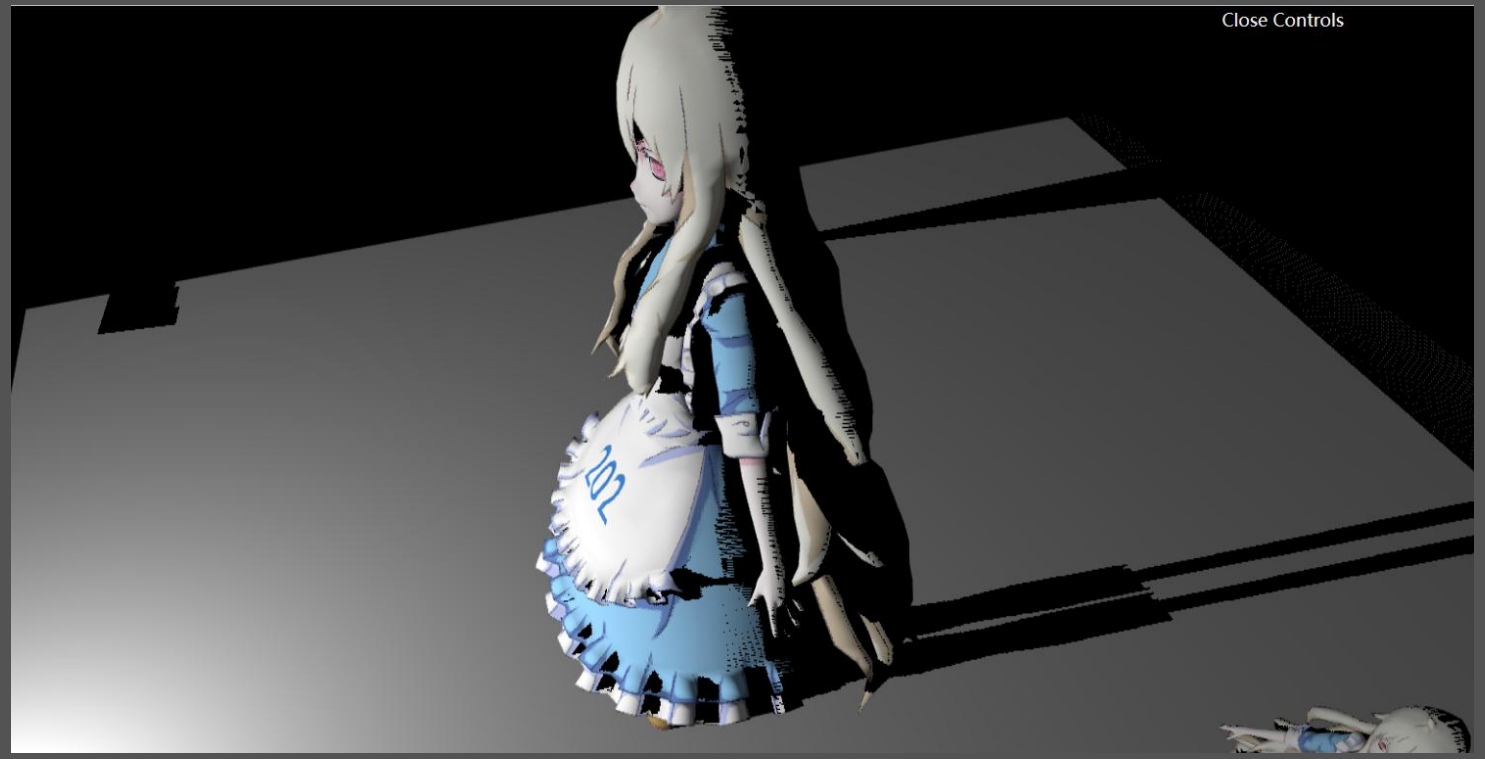
Projection Matrix

- 视觉空间→裁剪空间
- 对x，y，z分量进行缩放，用w分量做范围值。如果xyz都在w范围内，那么就是可见的
- 为了使在裁剪空间中深度是呈线性关系，投影变换采用正交投影
- 通俗的来说，我们将z轴舍弃，把屏幕上的像素点都挤进xy空间，分别进

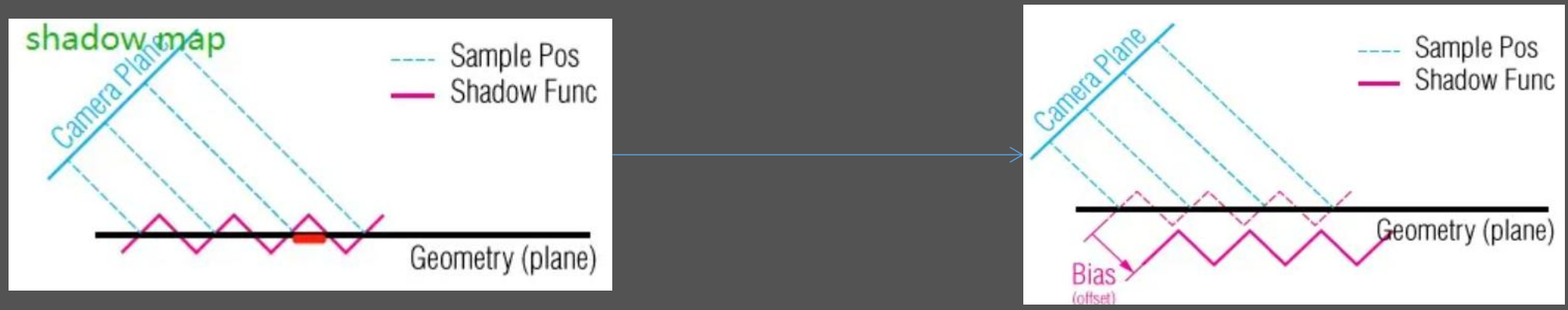


$$M_{ortho} = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2}{t-b} & 0 & 0 \\ 0 & 0 & \frac{2}{n-f} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -\frac{r+l}{2} \\ 0 & 1 & 0 & -\frac{t+b}{2} \\ 0 & 0 & 1 & -\frac{n+f}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Why does this problem arise?

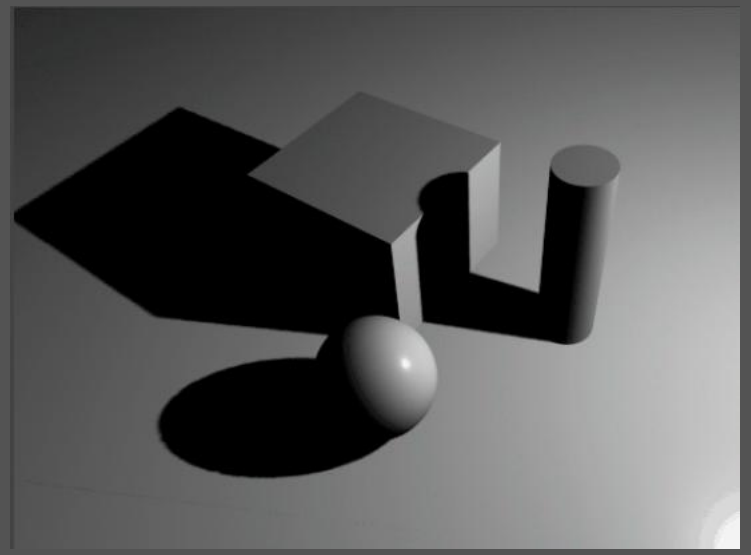


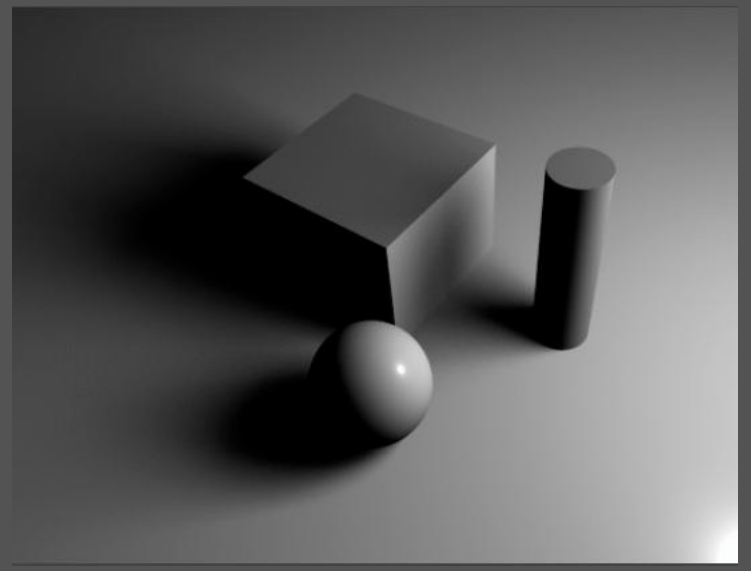
Why does this problem arise?



把阴影函数的判定调整更加“宽松”时，即当我们的光源采样的深度显著大于第一次采样的深度时，才判定产生阴影。

Percentage Closer Filtering

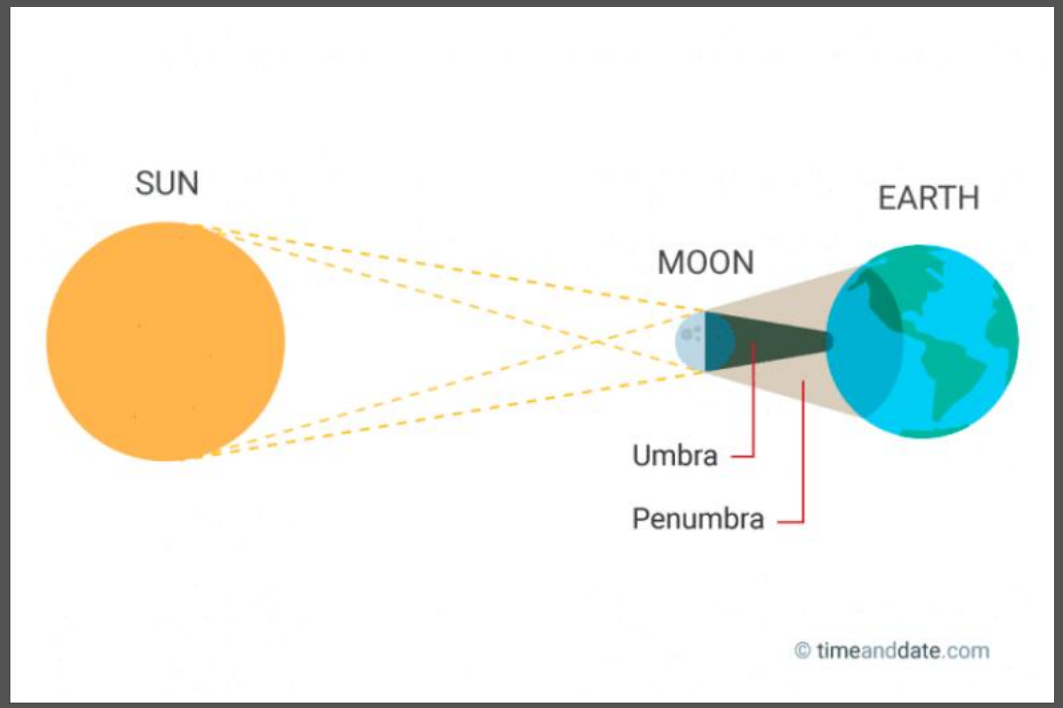




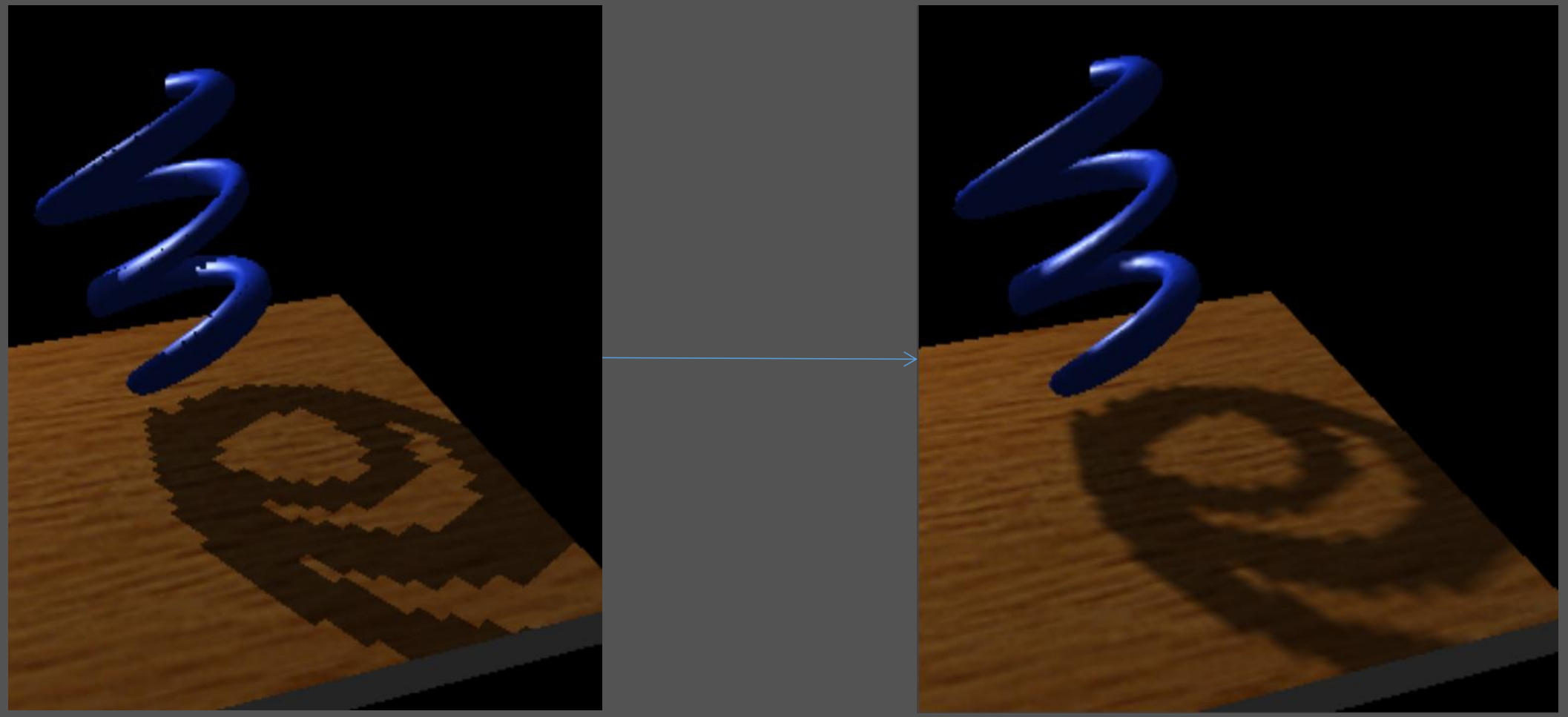
Which is more in line with the real world?

Percentage Closer Filtering

在现实世界中，是不存在单一的点光源的，面光源的存在会使我们的阴影在地面上时会从近自远有一定的变化，而要解决这个问题，如果我们直接根据对阴影边缘进行均值滤波结果会怎么样呢？



Percentage Closer Filtering



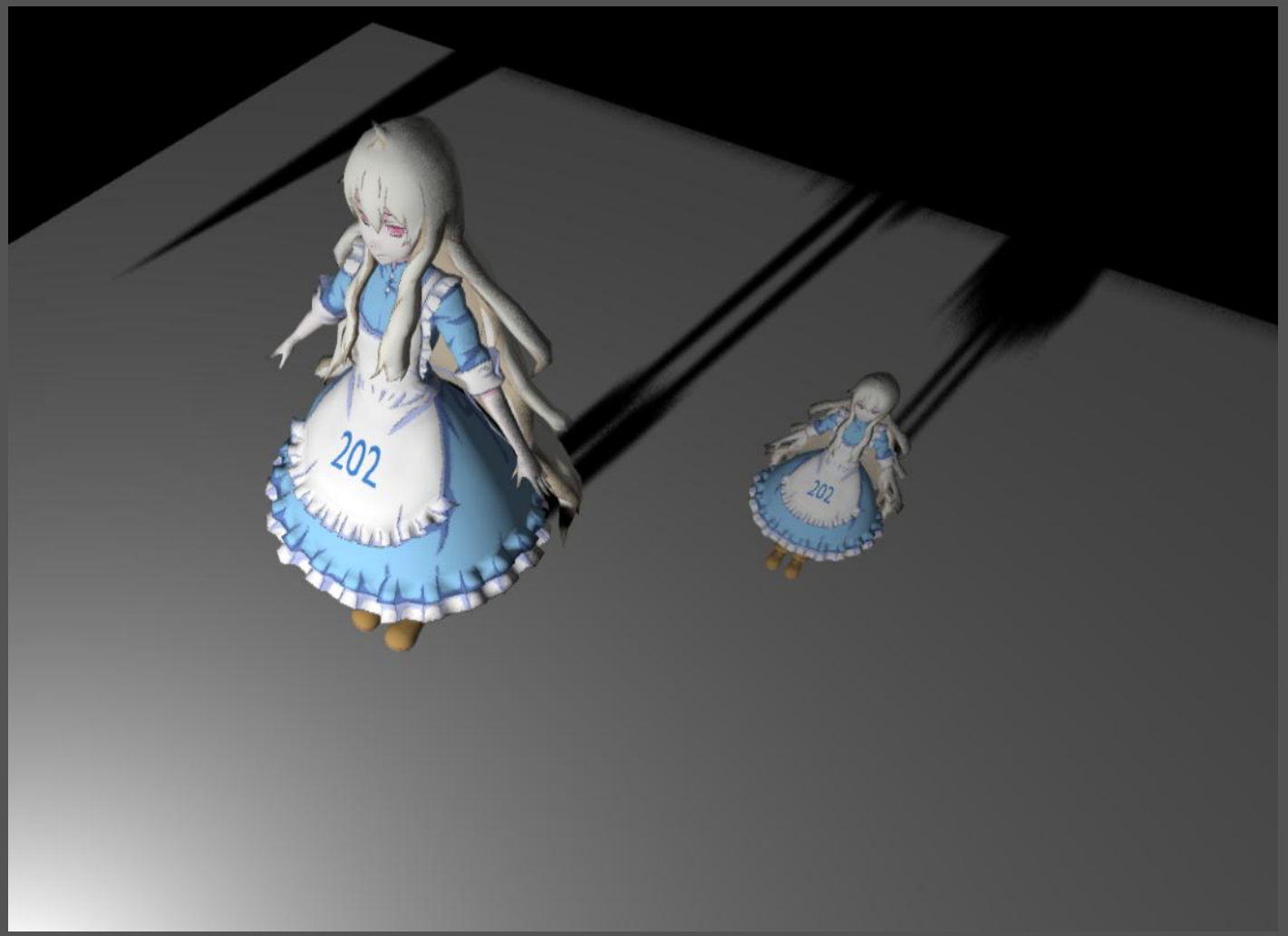
显然，与我们想要得到的效果相差甚远！

Solution

[link]1987,Rendering antialiased shadows with depth maps

- 对屏幕上所有像素点进行随机采样
- 将采样到的点进行均值滤波
- 最后得出来值为可见强度（即V）

Percentage Closer Filtering



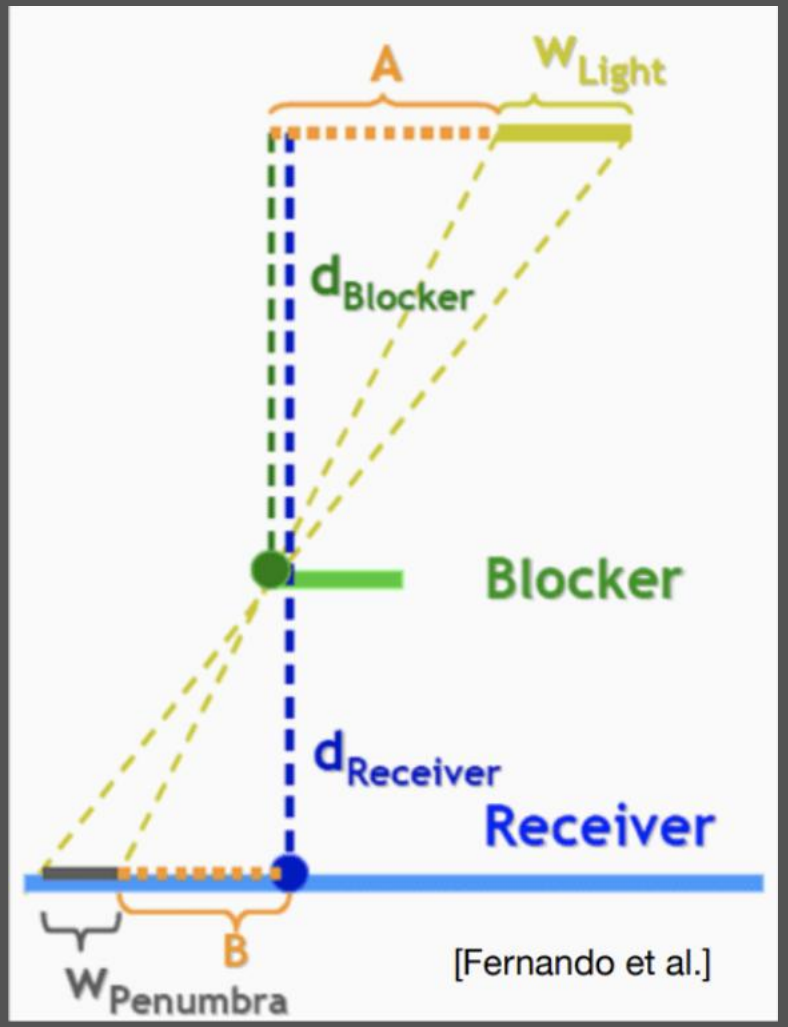
Percentage Closer Soft Shadows

在现实生活中，阴影也不总是软阴影，笔头的阴影与笔筒的阴影是明显不同的，简而言之，阴影离我们的遮挡物越远越模糊。

The cause of this problem

- 根据对现实生活的观察，阴影离遮挡物越远越模糊，即阴影离遮挡物越远那我们的采样范围就越小，反之同理。
- 遮挡物、阴影、物体之间的关系可以反应为右图，可得出以下公式：

$$w_{Penumbra} = (d_{Receiver} - d_{Blocker}) \cdot w_{Light} / d_{Blocker}$$

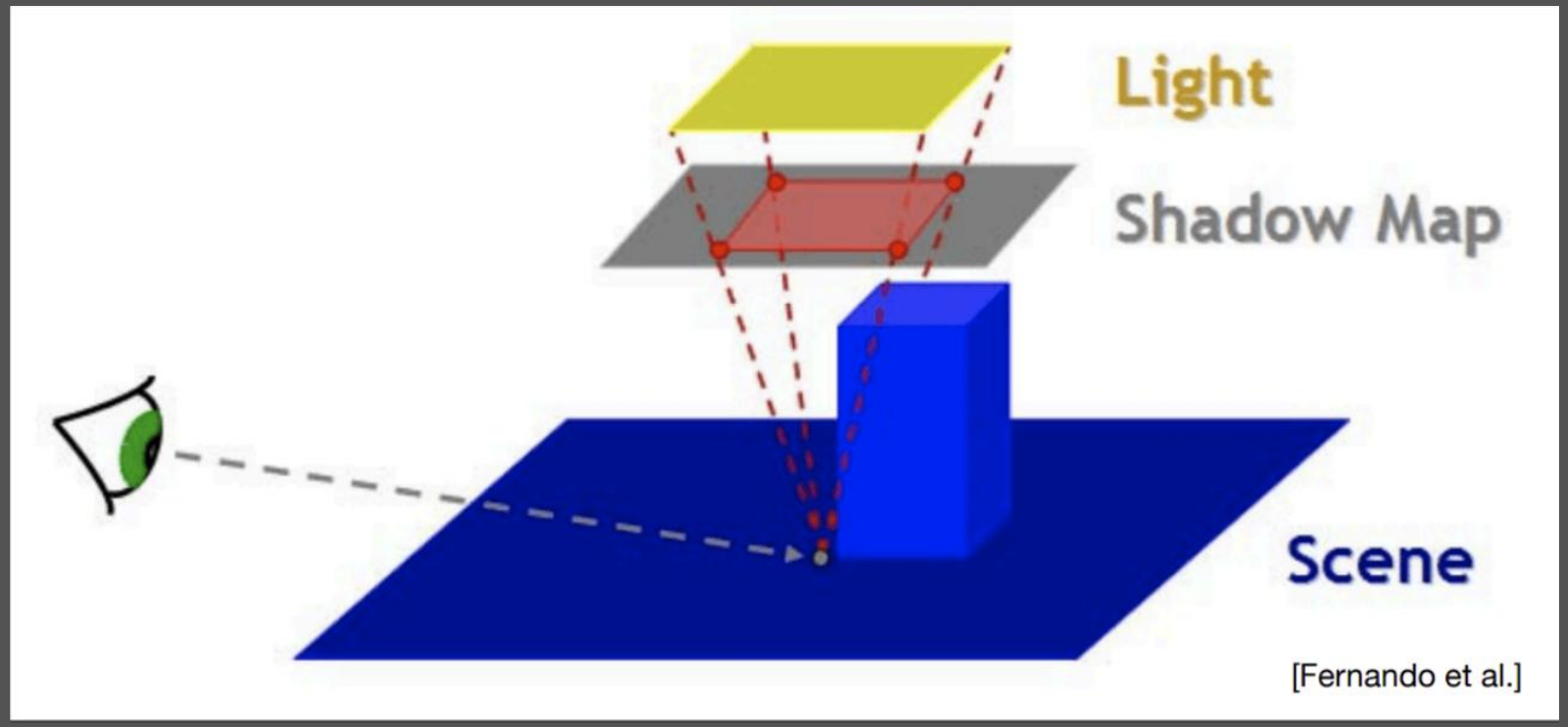


How to get radius?

$$w_{Penumbra} = (d_{Receiver} - d_{Blocker}) \cdot w_{Light} / d_{Blocker}$$

- 在上述公式中，receiver的深度是已知的，light的大小是我们设定的，那么剩下的是遮挡物blocker的深度了，为了结果更准确，我们在一个范围里计算这个点遮挡物的平均深度，那么我们又有了一个新的问题，这个范围取多少呢？能否根据相关联的数据计算出合适的范围？

How to get radius?



- 这张图给了我们一种方案，我们可以从shading point连接到光源，其经过Shadow Map时所截取的面积，就是我们用来求平均深度的采样面积，也就是离光源越近，我们所采样的范围就越大。

Get blocker’s depth

```
//phongFragment.glsl
float findBlocker(sampler2D shadowMap, vec2 uv, float zReceiver) {
  int blockerNum = 0;
  float blockDepth = 0.;
  float posZFromLight = vPositionFromLight.z;
  float searchRadius = LIGHT_SIZE_UV * (posZFromLight - NEAR_PLANE) / posZFromLight;
  poissonDiskSamples(uv);
  for(int i = 0; i < NUM_SAMPLES; i++){
    float shadowDepth = unpack(texture2D(shadowMap, uv + poissonDisk[i] * searchRadius));
    if(zReceiver > shadowDepth){
      blockerNum++;
      blockDepth += shadowDepth;
    }
  }

  if(blockerNum == 0)
    return -1.;
  else
    return blockDepth / float(blockerNum);
}
```

PCSS

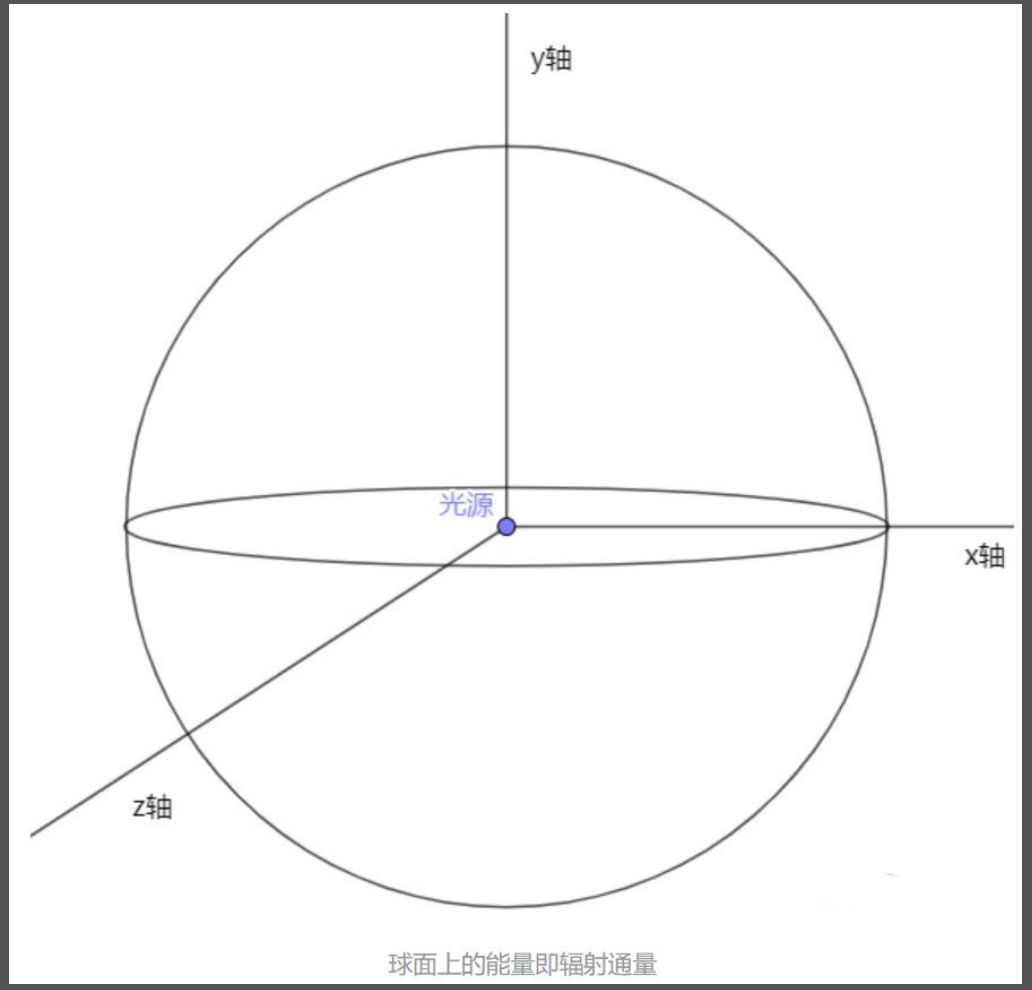
```
//phongFragment.glsl

float PCSS(sampler2D shadowMap, vec4 coords, float biasC){
  float zReceiver = coords.z;

  // STEP 1: avgblocker depth
  float avgBlockerDepth = findBlocker(shadowMap, coords.xy, zReceiver);

  if(avgBlockerDepth < -EPS)
    return 1.0;

  // STEP 2: penumbra size
  float penumbra = (zReceiver - avgBlockerDepth) * LIGHT_SIZE_UV / avgBlockerDepth;
  float filterRadiusUV = penumbra;

  // STEP 3: filtering
  return PCF(shadowMap, coords, biasC, filterRadiusUV);
}
```

PCSS

```
float PCSS(sampler2D shadowMap, vec4 coords,float biasC){

  // STEP 1: avgblocker depth
  float dReceiver=coords.z;

  float dBlocker=findBlocker(shadowMap,coords.xy,dReceiver);
  if(dBlocker<-EPS)
    return 1.0;
  // STEP 2: penumbra size
  float penumbra=(dReceiver-dBlocker)/dBlocker*LIGHT_SIZE_UV;
  // STEP 3: filtering

  return PCF(shadowMap,coords,biasC,penumbra);

}
```

Path-tracing

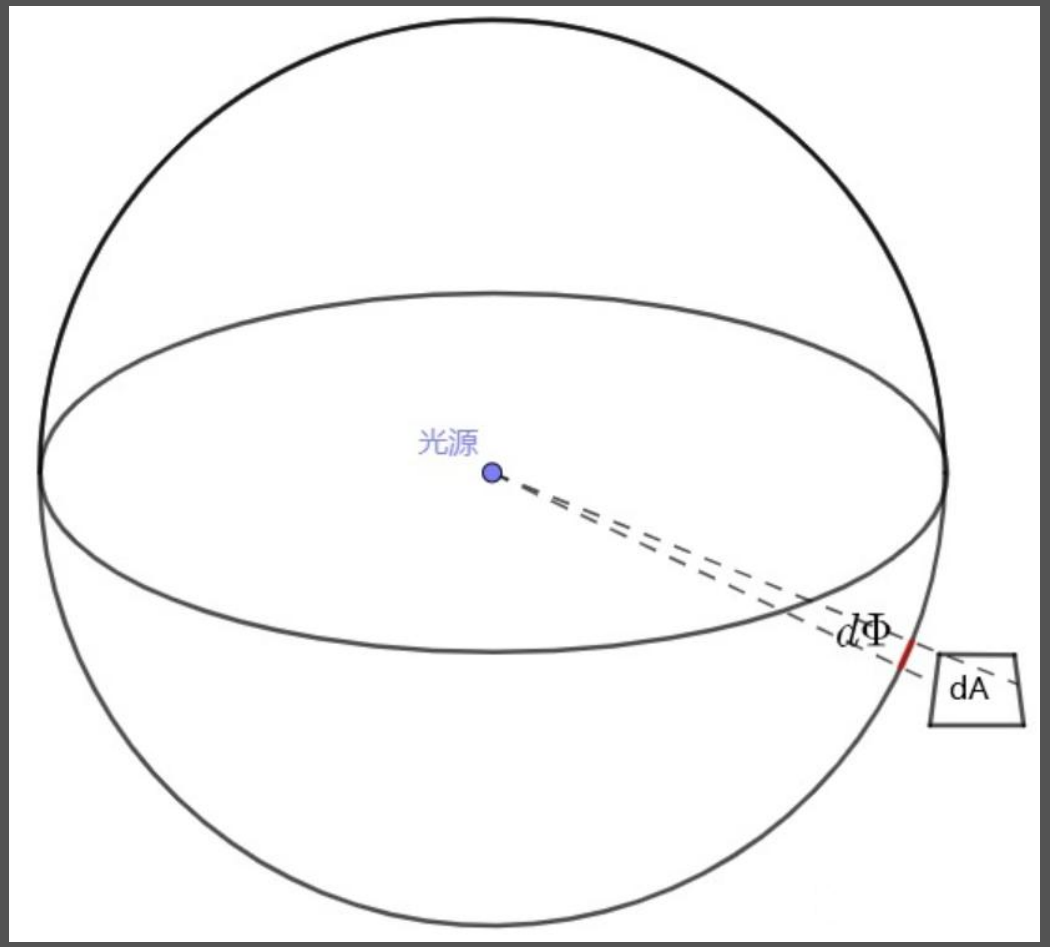
Radiant flux



辐射通量指的就是单位时间内的发射、反射、传播或吸收的辐射能。这就好比速度指的是单位时间内运动的距离。那么我们只需要取得一个极短时间（dt）内接收到的辐射能（dQ），即可算出辐射通量的值。公式如下：

$$\varphi \equiv \frac{dQ}{dt}$$

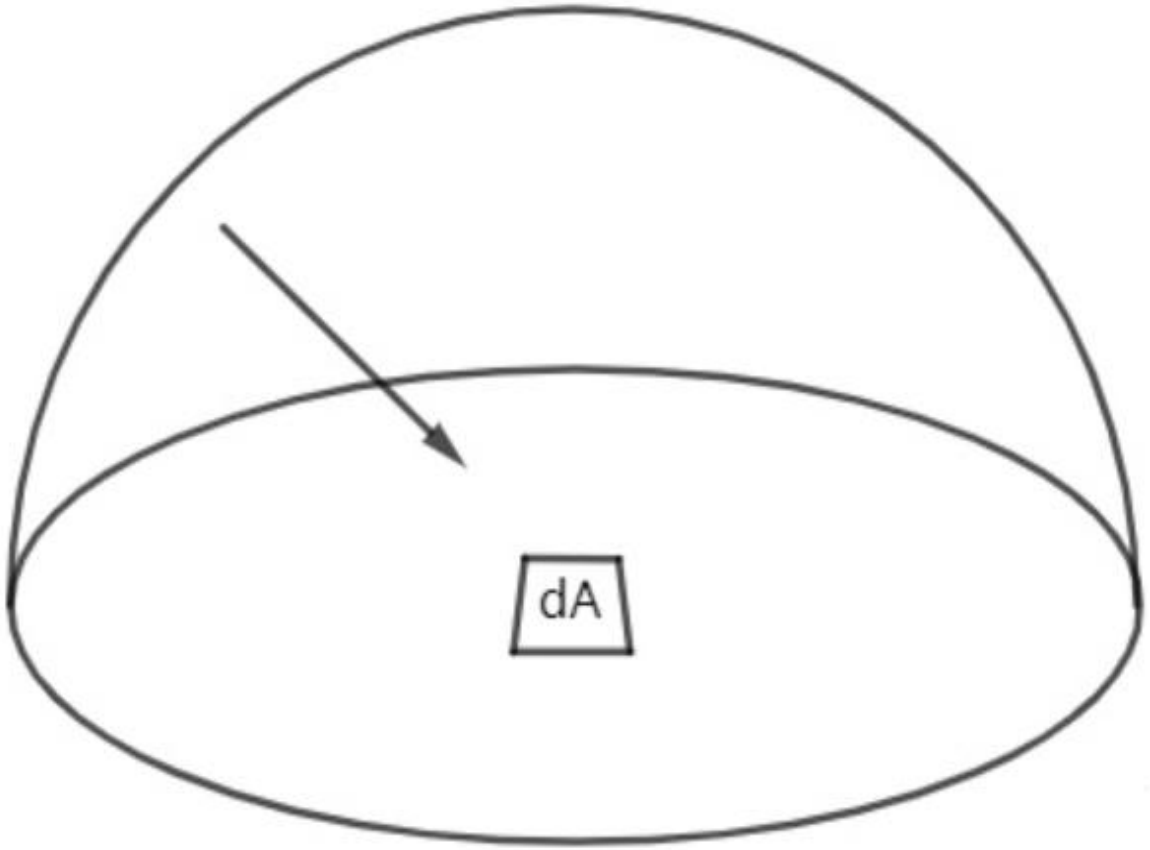
Irradiance



在极短时间内整个面积接收到的能量即为 $d\varphi$（图中红色弧线），那么单位面积接收到的辐射通量即为辐照度，常用 E 表示，单位为：W/m²:

$$E \equiv \frac{d\varphi}{dA}$$

Radiance



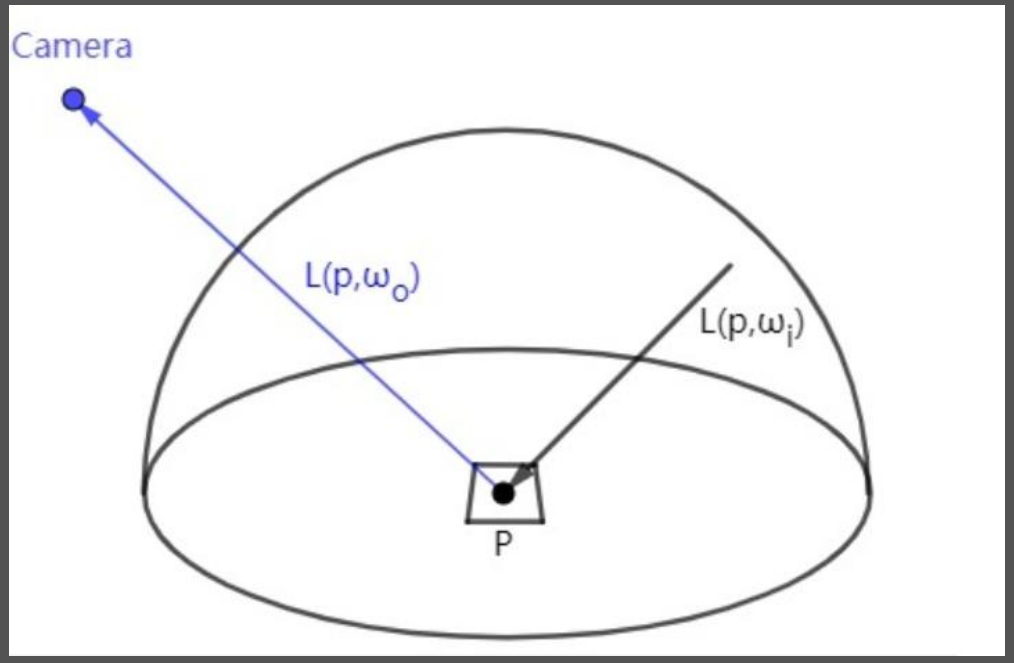
某个方向的入射光

辐射率的指的就是单位面积下接收到的来自单位立体角的辐射通量，常用 L 来表示，其单位为：W/sr · m²。

假设一个微小的平面 dA 接收到来自某个微分立体角 dω 的辐射通量为 ，θ为法线与入射光线的夹角，那么辐射率即为：

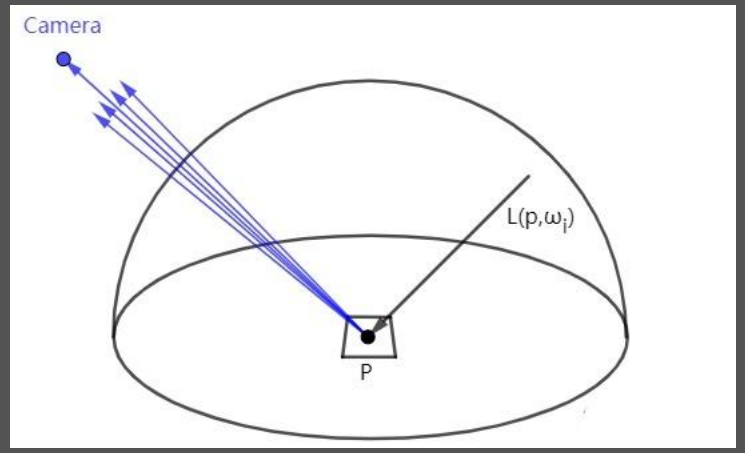
$$L(p,\omega) = \frac{d^2\varphi(p,\omega)}{d\omega dA cos\theta}$$

BRDF

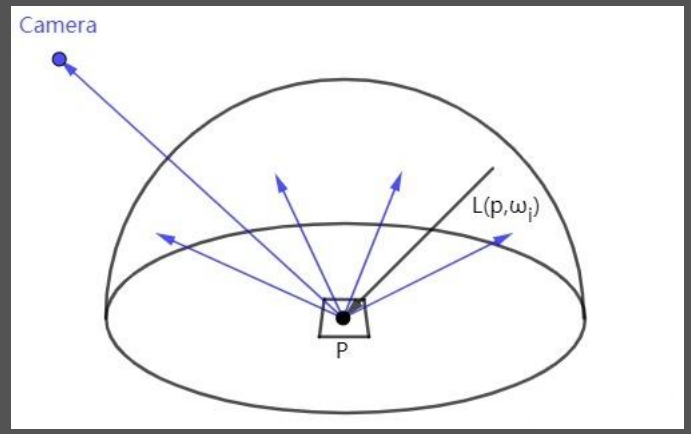


设入射光为 $L(p, w_i)$那么P点的辐照度为 $dE(p, w_i) = L(p, w_i)n \cdot w_i dw_i$ ，那么我们只需要知道有多少比例的辐射通量会被反射到w_i 方向上，就可以求$L(p, w_i)$的值了。

BRDF



Specular

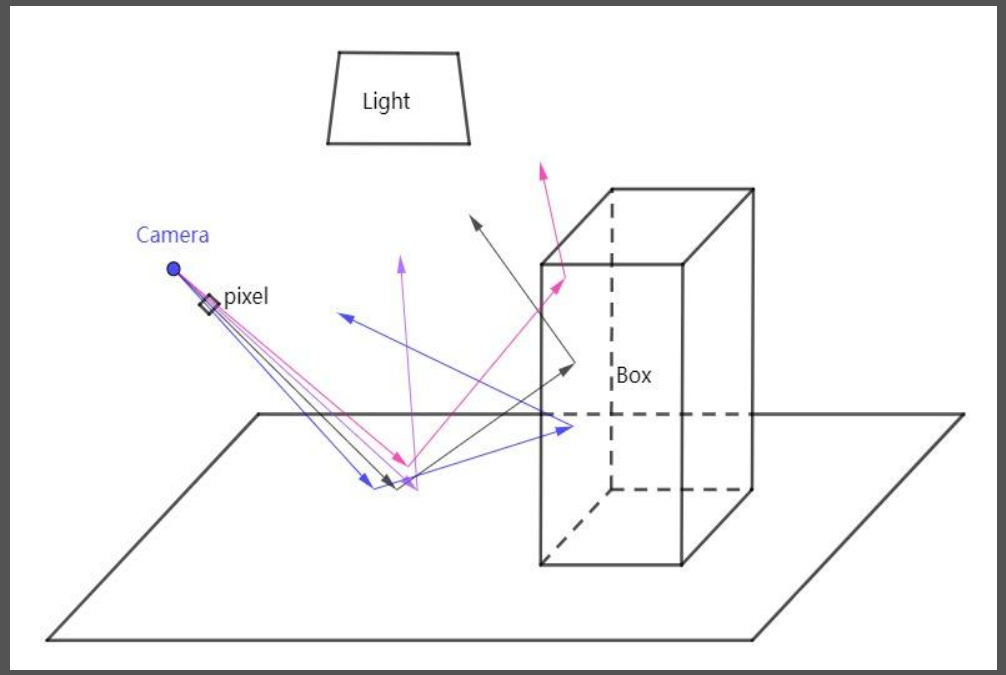


Diffuse

BRDF决定了物体的材质,BRDF我们常用$f_r(p, w_i, w_o)$来表示，同时我要对P点接收到的所有光都要计算往Camera方向的分布值，然后把它们积分起来，公式如下：

$$L_o(p, w_o) = L_e(p, w_o) + \int f_r(p, w_i, w_o) L_i(p, w_i) n \cdot w_i dw_i$$

Path Tracing




```
float generationRay(vec3 camera, vec3 pixel)
{
   float pixel_radiance = 0.0;//初始化像素受到的辐射率
   for(int i=0; i < count; i++)
   {
      vec3 pos = random() by pdf;//根据pdf随机像素内的一个位置
      ray r = ray(camera, pos-camera);//从相机往pos发射射线
      if(r hit object at p)//如果射线打到物体上，交点为p
      {
         //利用shade计算p点camera方向的radiance
         pixel_radiance += (1/count) * shade(p, camera-pos);
      }
   }
   return pixel_radiance;
}
```

Russian roulette

如何解决无线递归问题?

- 设置深度
- 俄罗斯轮盘赌

```
float shade(vec3 p, vec3 wo)
{
    float prob = 0.6;// 随便指定一个概率值
    float num = random(0,1);// 随机获取一个0-1的数值
    if(num > prob)
    {
        // 1-prob 的概率不发射光线
        return 0.0;
    }
    vec3 wi = random() by pdf;
    ray r = ray(p, wi);
    if(r hit light)
        return fr(p, wi, wo) * li * dot(n, wi) / pdf(wi) / prob;// 记得要除以发射光线的概率prob
    else if(r hit object at o)
        return fr(p, wi, wo) * shade(o, -wi) * dot(n, wi) / pdf(wi) / prob;
}
```

Next...

- 实时环境光照?
- 实时全局光照?
- 实时光线追踪?
- 利用神经网络来推断光线追踪?