Screen Space Ambient Occlusion

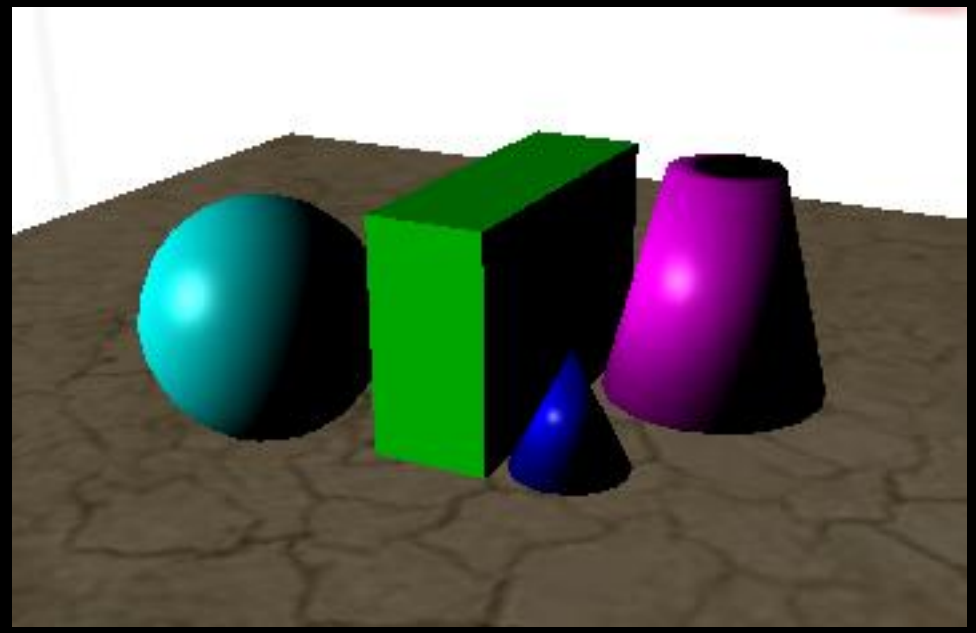
쿠재아이

김재경

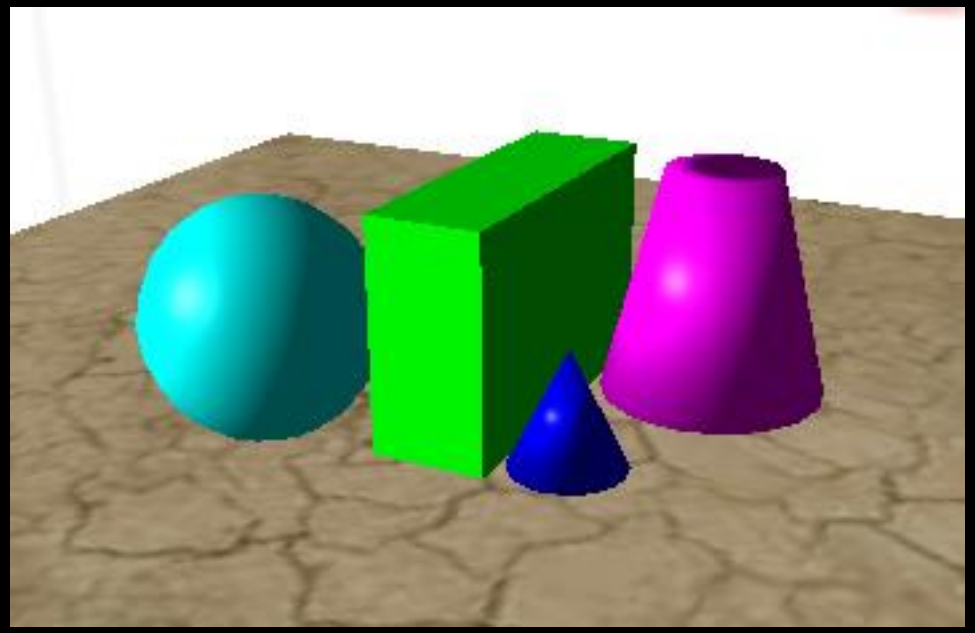
Ambient?

- 주변 물체가 반사한 빛이 관찰자가 보는 물체에 반사된 빛

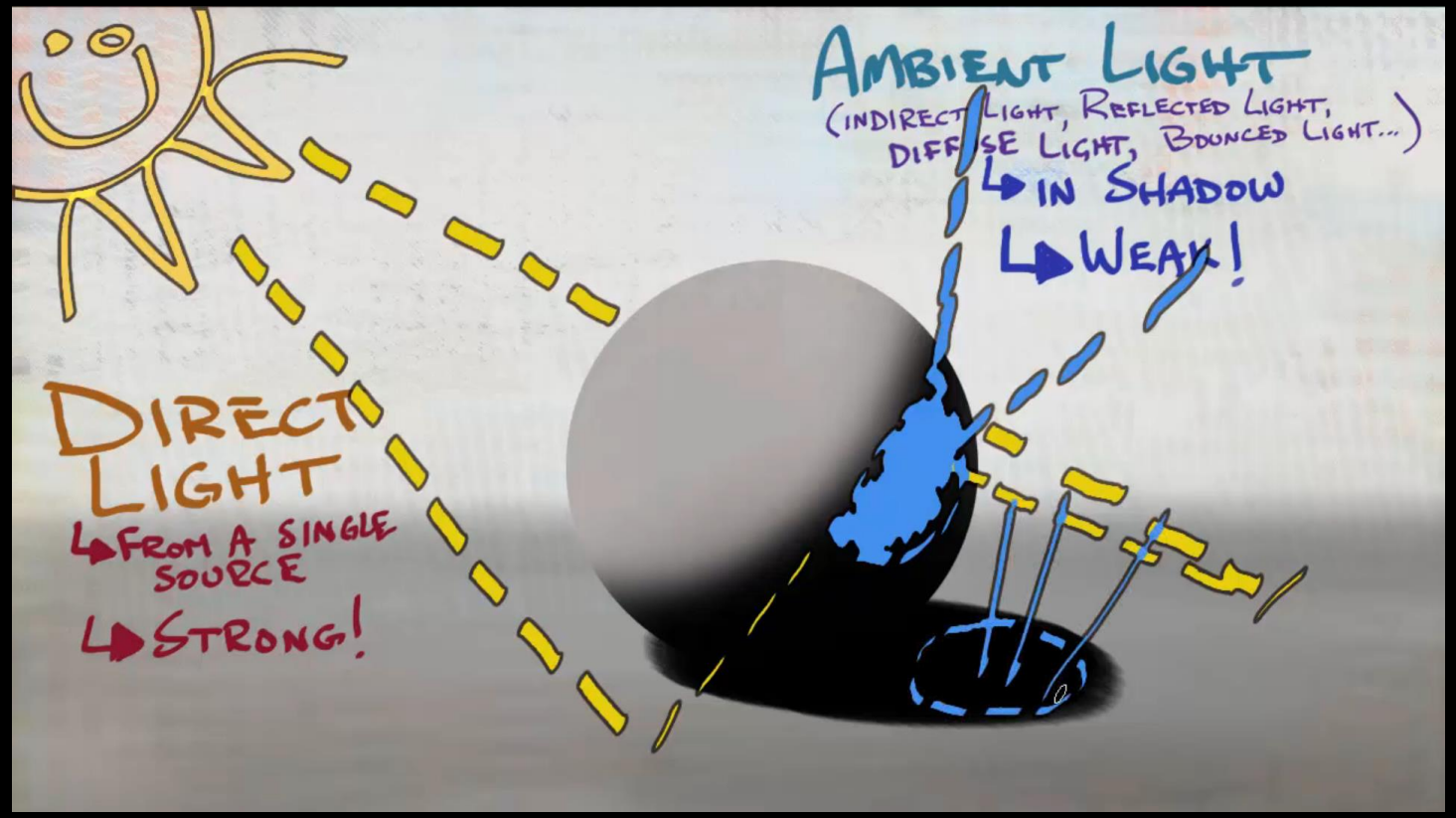
Ambient X



Ambient O



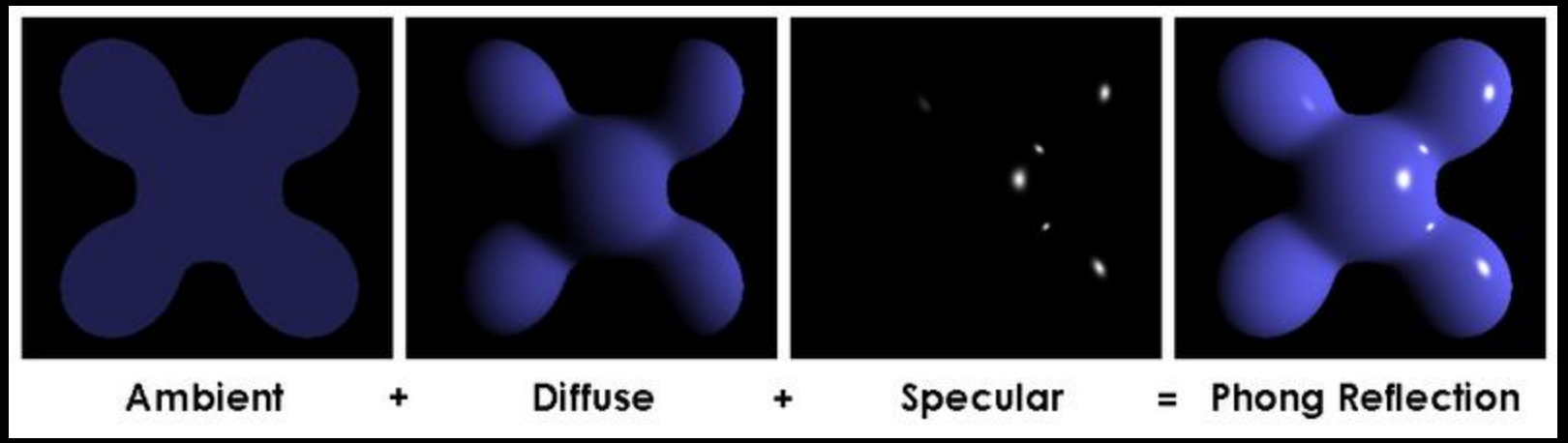
Ambient?



https://www.youtube.com/watch?v=7fLV5ezO64w

Ambient?

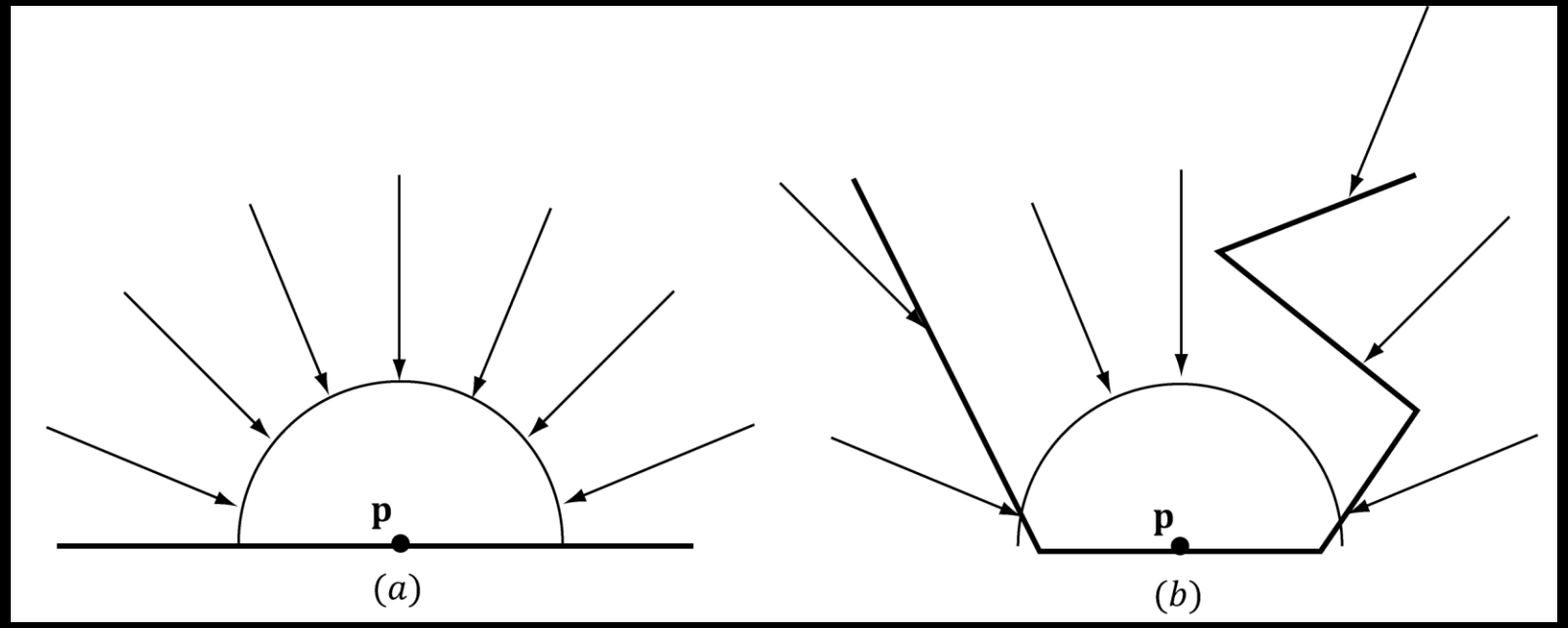
- Phong Shading에서는 계산이 복잡하다는 이유로 상수로 사용



$$C_a = A_L \otimes m_d$$

Ambient Occlusion?

- 표면의 한 점 p가 받는 간접광(Ambient)의 양은 p를 중심으로 한 반구로 들어오는 빛이 가려진(Occlusion) 정도에 비례한다

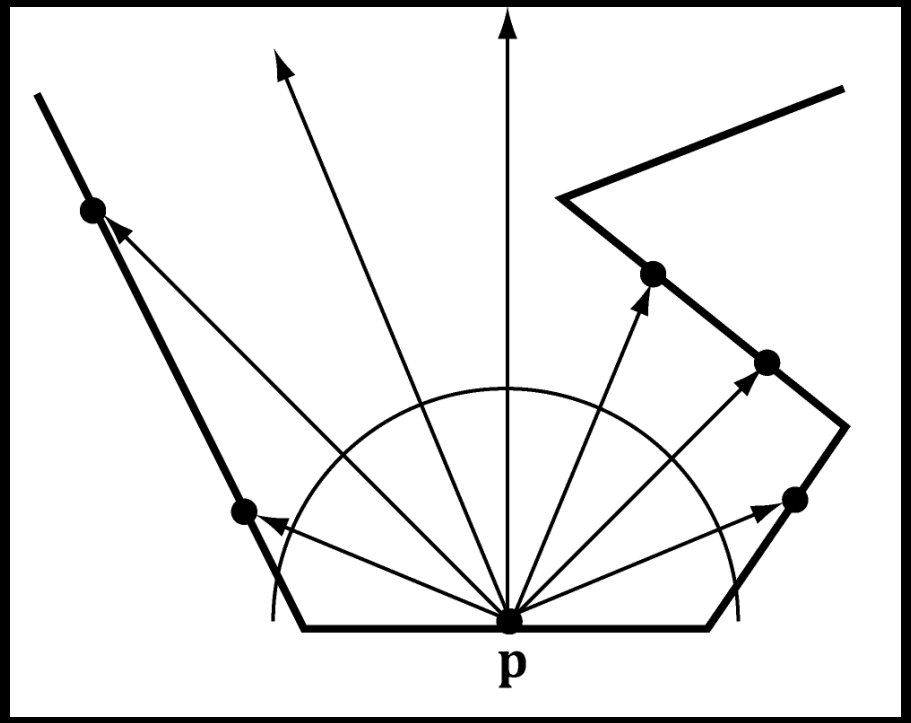


(a)는 차폐가 적다

(b)는 차폐가 많다

Ambient Occlusion?

- 책에서는 Ray Casting으로 차폐도를 추정함
- P 중심의 반구 전반에 무작위로 광선을 쏘아서 메시와의 교차를 판정



N개의 광선을 쏘고 h개가 메시에 교차했을 때의 차폐도

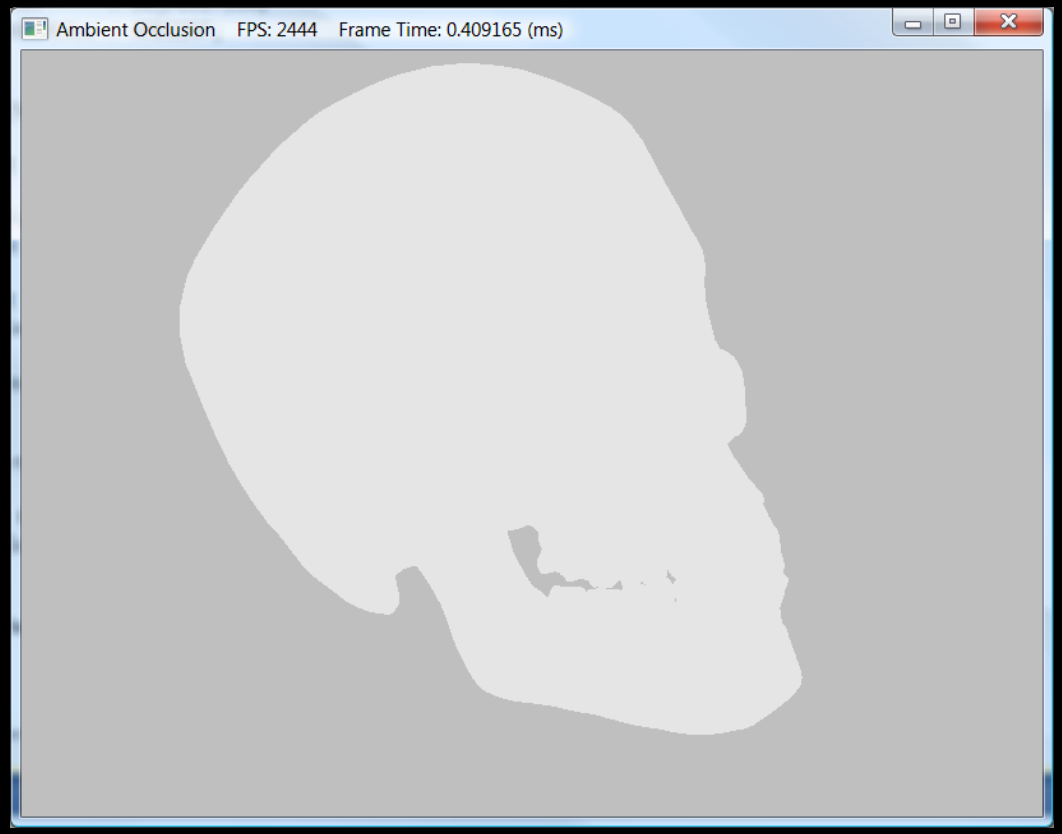
$$\text{차폐도} = \frac{h}{N} \in [0,1]$$

Screen Space Ambient Occlusion?

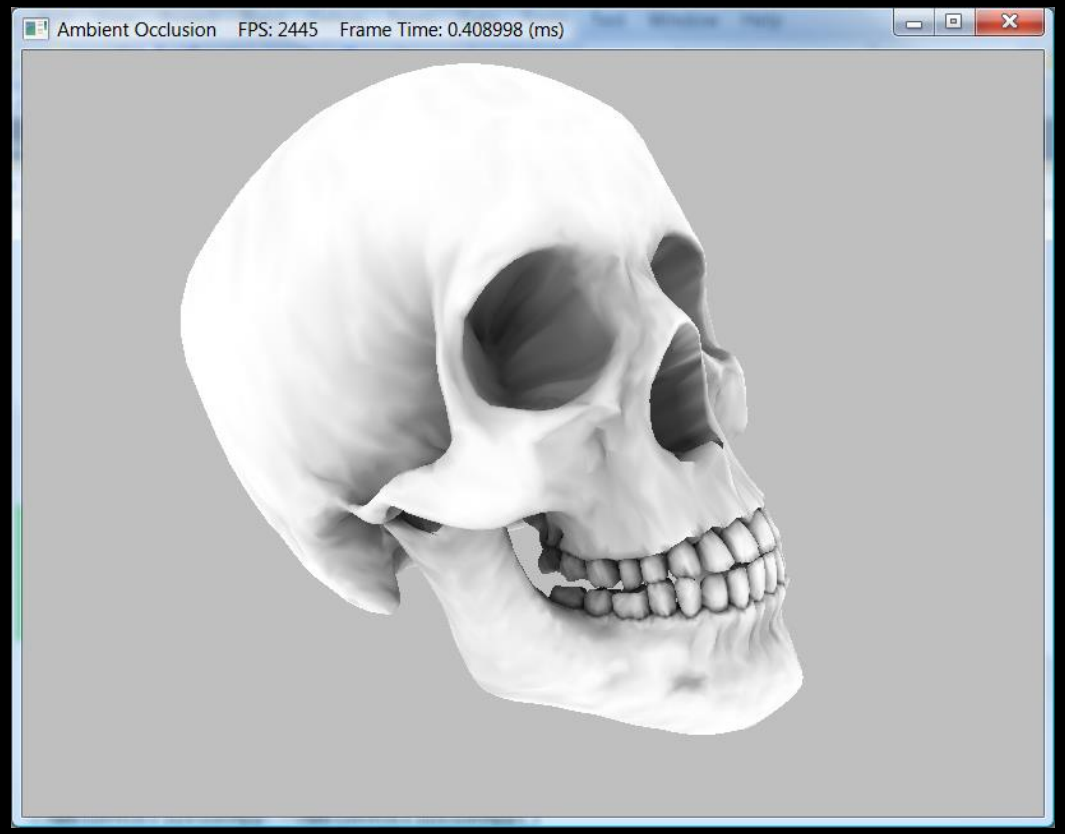
- 차폐도를 정점별로 계산하면 오래 걸리니 Screen Space에서 계산
- 좀 더 사실적인 그림자를 표현할 수 있다

Screen Space Ambient Occlusion?

SSAO X



SSAO O

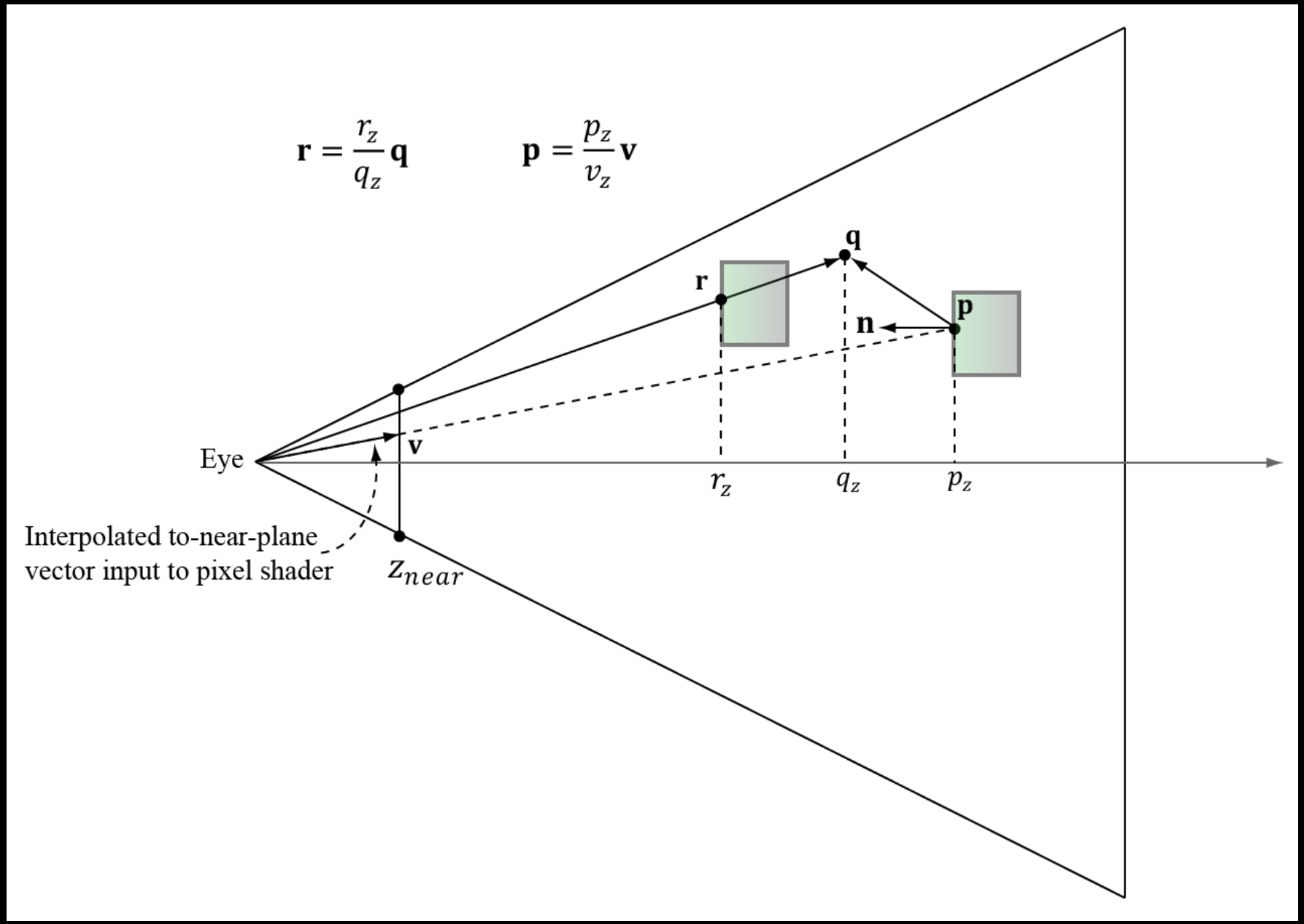


구현

```
VertexOut VS(VertexIn vin)
{
    VertexOut vout = (VertexOut)0.0f;

    // Fetch the material data.
    MaterialData matData = gMaterialData[gMaterialIndex];

    // Assumes nonuniform scaling; otherwise, need to use inverse-transpose of world matrix.
    vout.NormalW = mul(vin.NormalL, (float3x3)gWorld);
    vout.TangentW = mul(vin.TangentU, (float3x3)gWorld);

    // Transform to homogeneous clip space.
    float4 posW = mul(float4(vin.PosL, 1.0f), gWorld);
    vout.PosH = mul(posW, gViewProj);

    // Output vertex attributes for interpolation across triangle.
    float4 texC = mul(float4(vin.TexC, 0.0f, 1.0f), gTexTransform);
    vout.TexC = mul(texC, matData.MatTransform).xy;

    return vout;
}
```

구현

```
float4 PS(VertexOut pin) : SV_Target
{
    // Fetch the material data.
    MaterialData matData = gMaterialData[gMaterialIndex];
    float4 diffuseAlbedo = matData.DiffuseAlbedo;
    uint diffuseMapIndex = matData.DiffuseMapIndex;
    uint normalMapIndex = matData.NormalMapIndex;

    // Dynamically look up the texture in the array.
    diffuseAlbedo *= gTextureMaps[diffuseMapIndex].Sample(gsamAnisotropicWrap, pin.TexC);

#ifdef ALPHA_TEST
    // Discard pixel if texture alpha < 0.1.  We do this test as soon
    // as possible in the shader so that we can potentially exit the
    // shader early, thereby skipping the rest of the shader code.
    clip(diffuseAlbedo.a - 0.1f);
#endif

    // Interpolating normal can unnormalize it, so renormalize it.
    pin.NormalW = normalize(pin.NormalW);

    // NOTE: We use interpolated vertex normal for SSAO.

    // Write normal in view space coordinates
    float3 normalV = mul(pin.NormalW, (float3x3)gView);
    return float4(normalV, 0.0f);
}
```


$\mathbf{r} = \frac{r_z}{q_z}\mathbf{q}$
$\mathbf{p} = \frac{p_z}{v_z}\mathbf{v}$
q
r
p
n
v
Eye
r_z
q_z
p_z
Interpolated to-near-plane
vector input to pixel shader
z_{near}

구현

```
static const float2 gTexCoords[6] =
{
    float2(0.0f, 1.0f),
    float2(0.0f, 0.0f),
    float2(1.0f, 0.0f),
    float2(0.0f, 1.0f),
    float2(1.0f, 0.0f),
    float2(1.0f, 1.0f)
};
```

```
VertexOut VS(uint vid : SV_VertexID)
{
    VertexOut vout;

    vout.TexC = gTexCoords[vid];

    // Quad covering screen in NDC space.
    vout.PosH = float4(2.0f*vout.TexC.x - 1.0f, 1.0f - 2.0f*vout.TexC.y, 0.0f, 1.0f);

    // Transform quad corners to view space near plane.
    float4 ph = mul(vout.PosH, gInvProj);
    vout.PosV = ph.xyz / ph.w;

    return vout;
}
```

구현

```
float4 PS(VertexOut pin) : SV_Target
{
    // p -- the point we are computing the ambient occlusion for.
    // n -- normal vector at p.
    // q -- a random offset from p.
    // r -- a potential occluder that might occlude p.

    // Get viewspace normal and z-coord of this pixel.
    float3 n = normalize(gNormalMap.SampleLevel(gsamPointClamp, pin.TexC, 0.0f).xyz);
    float pz = gDepthMap.SampleLevel(gsamDepthMap, pin.TexC, 0.0f).r;
    pz = NdcDepthToViewDepth(pz);

    //
    // Reconstruct full view space position (x,y,z).
    // Find t such that p = t*pin.PosV.
    // p.z = t*pin.PosV.z
    // t = p.z / pin.PosV.z
    //
    float3 p = (pz/pin.PosV.z)*pin.PosV;

    // Extract random vector and map from [0,1] --> [-1, +1].
    float3 randVec = 2.0f*gRandomVecMap.SampleLevel(gsamLinearWrap, 4.0f*pin.TexC, 0.0f).rgb - 1.0f;
```

구현

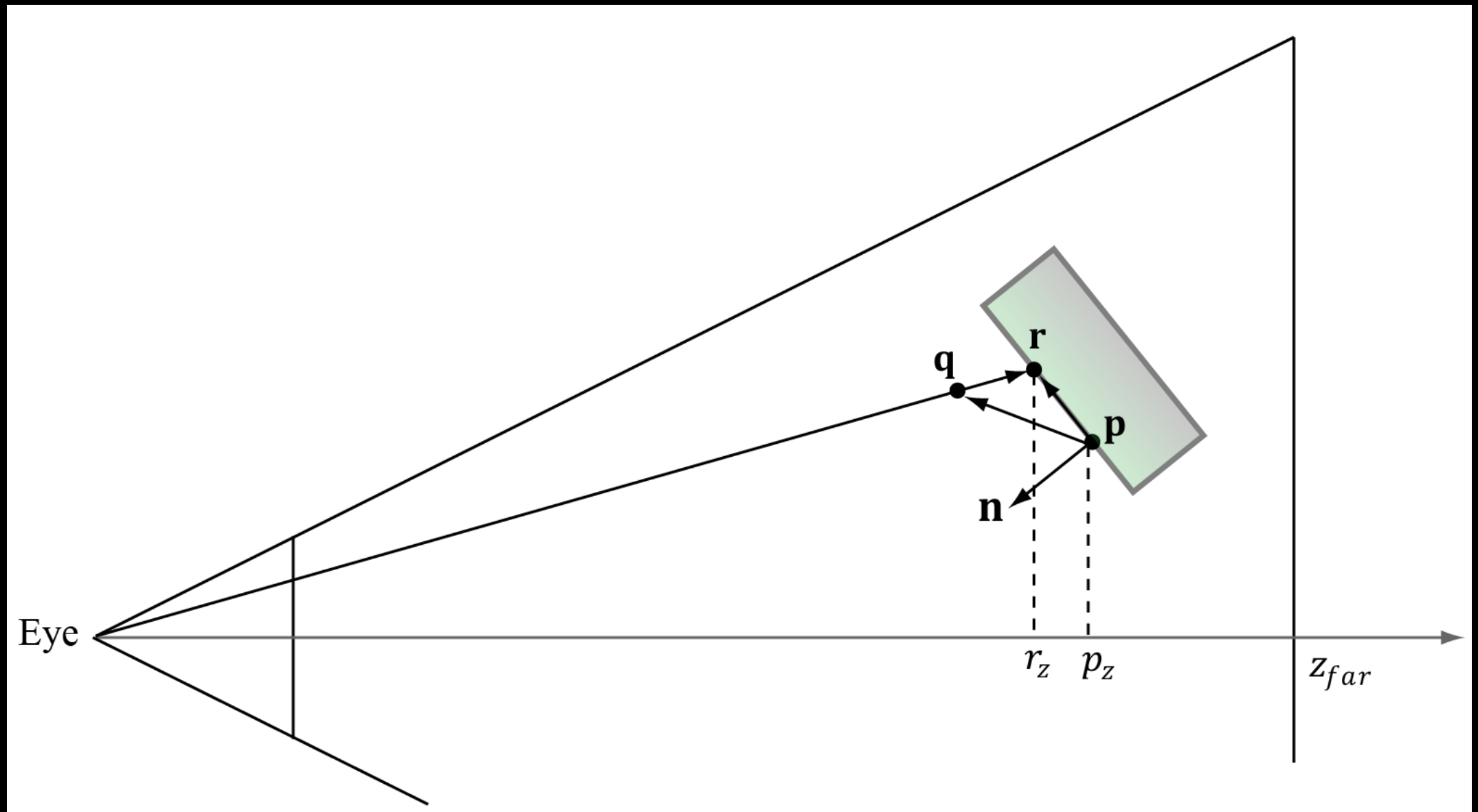
```
float occlusionSum = 0.0f;

// Sample neighboring points about p in the hemisphere oriented by n.
for(int i = 0; i < gSampleCount; ++i)
{
    // Are offset vectors are fixed and uniformly distributed (so that our offset vectors
    // do not clump in the same direction).  If we reflect them about a random vector
    // then we get a random uniform distribution of offset vectors.
    float3 offset = reflect(gOffsetVectors[i].xyz, randVec);

    // Flip offset vector if it is behind the plane defined by (p, n).
    float flip = sign( dot(offset, n) );

    // Sample a point near p within the occlusion radius.
    float3 q = p + flip * gOcclusionRadius * offset;

    // Project q and generate projective tex-coords.
    float4 projQ = mul(float4(q, 1.0f), gProjTex);
    projQ /= projQ.w;
```

구현

```
float rz = gDepthMap.SampleLevel(gsamDepthMap, projQ.xy, 0.0f).r;
rz = NdcDepthToViewDepth(rz);

// Reconstruct full view space position r = (rx,ry,rz).  We know r
// lies on the ray of q, so there exists a t such that r = t*q.
// r.z = t*q.z ==> t = r.z / q.z

float3 r = (rz / q.z) * q;

//
// Test whether r occludes p.
//   * The product dot(n, normalize(r - p)) measures how much in front
//     of the plane(p,n) the occluder point r is.  The more in front it is, the
//     more occlusion weight we give it.  This also prevents self shadowing where
//     a point r on an angled plane (p,n) could give a false occlusion since they
//     have different depth values with respect to the eye.
//   * The weight of the occlusion is scaled based on how far the occluder is from
//     the point we are computing the occlusion of.  If the occluder r is far away
//     from p, then it does not occlude it.
//
```

구현

```
    float distZ = p.z - r.z;
    float dp = max(dot(n, normalize(r - p)), 0.0f);

    float occlusion = dp*OcclusionFunction(distZ);

    occlusionSum += occlusion;
}

occlusionSum /= gSampleCount;

float access = 1.0f - occlusionSum;

// Sharpen the contrast of the SSAO map to make the SSAO affect more dramatic.
return saturate(pow(access, 6.0f));
```


Eye
q
r
p
n
r_z
p_z
z_{far}

구현

```
float NdcDepthToViewDepth(float z_ndc)
{
    // z_ndc = A + B/viewZ, where gProj[2,2]=A and gProj[3,2]=B.
    float viewZ = gProj[3][2] / (z_ndc - gProj[2][2]);
    return viewZ;
}
```

OcclusionFunction

```
float occlusion = 0.0f;
if(distZ > gSurfaceEpsilon)
{
    float fadeLength = gOcclusionFadeEnd - gOcclusionFadeStart;

    // Linearly decrease occlusion from 1 to 0 as distZ goes
    // from gOcclusionFadeStart to gOcclusionFadeEnd.
    occlusion = saturate( (gOcclusionFadeEnd-distZ)/fadeLength );
}

return occlusion;
```