Ambient Occlusion

CMPM164

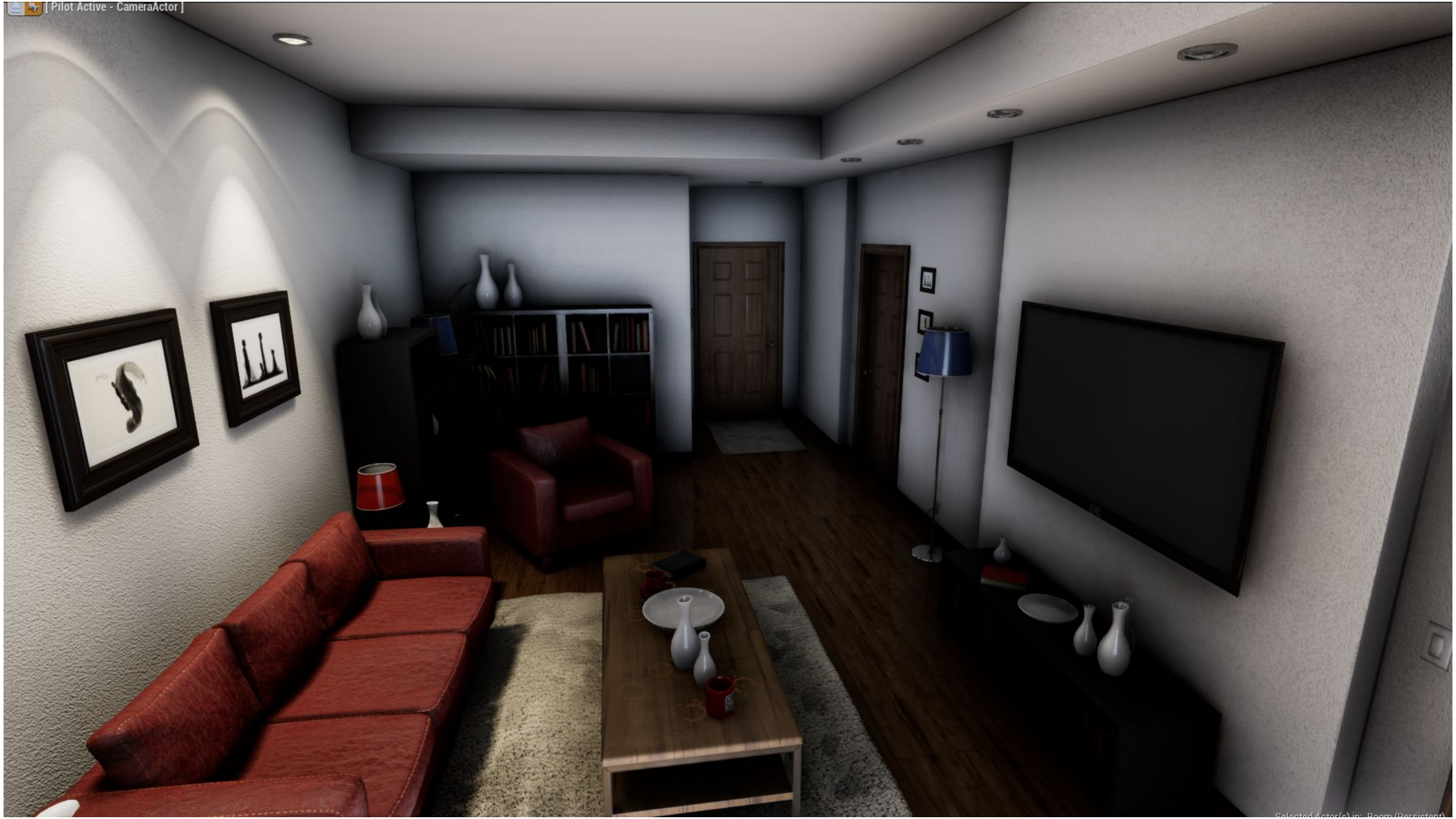
Ambient Occlusion types include:

Screen Space Ambient Occlusion (SSAO) (fast and decent quality)

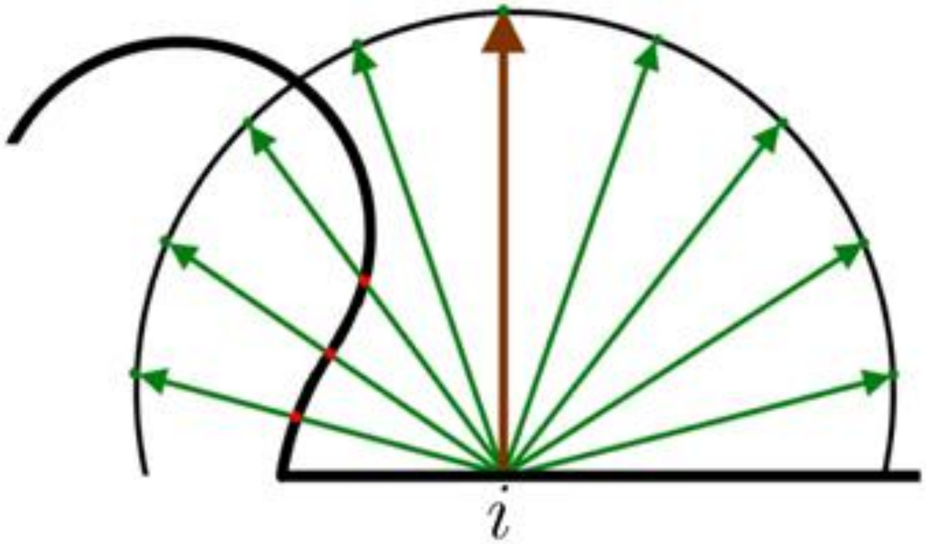
Horizon Based Ambient Occlusion (HBAO) (slower but better quality)

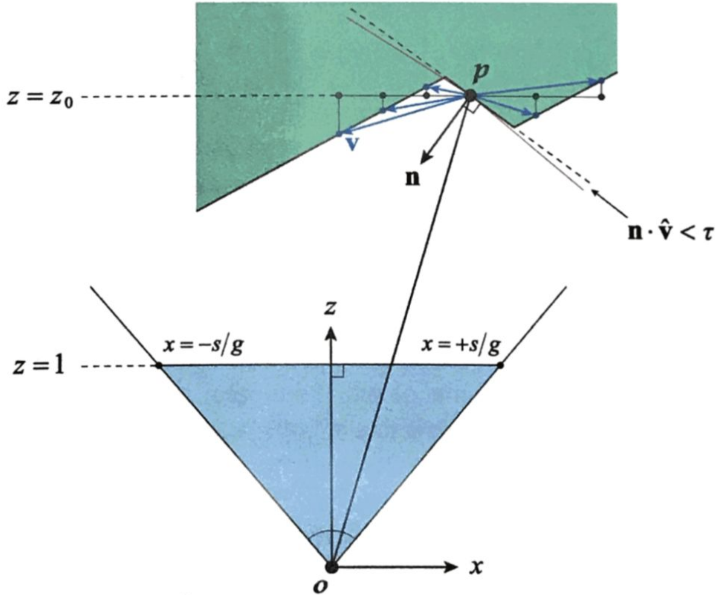
HBAO+ (even slower but even better quality)

Voxel Ambient Occlusion (VXAO) (a lot slower and a lot better quality)


[Pilot Active - CameraActor]

A pixel shader that runs this:





$$f(\mathbf{v}) = 1 - \sqrt{1 - \max(\mathbf{n}\cdot\hat{\mathbf{v}} - \tau, 0)^2}.$$

$$w(\mathbf{v}) = \operatorname{sat}\left(1 - \frac{\mathbf{n}\cdot\mathbf{v}}{d_{\max}}\right),$$

$$H(\mathbf{v}) = w(\mathbf{v})\, f(\mathbf{v}).$$

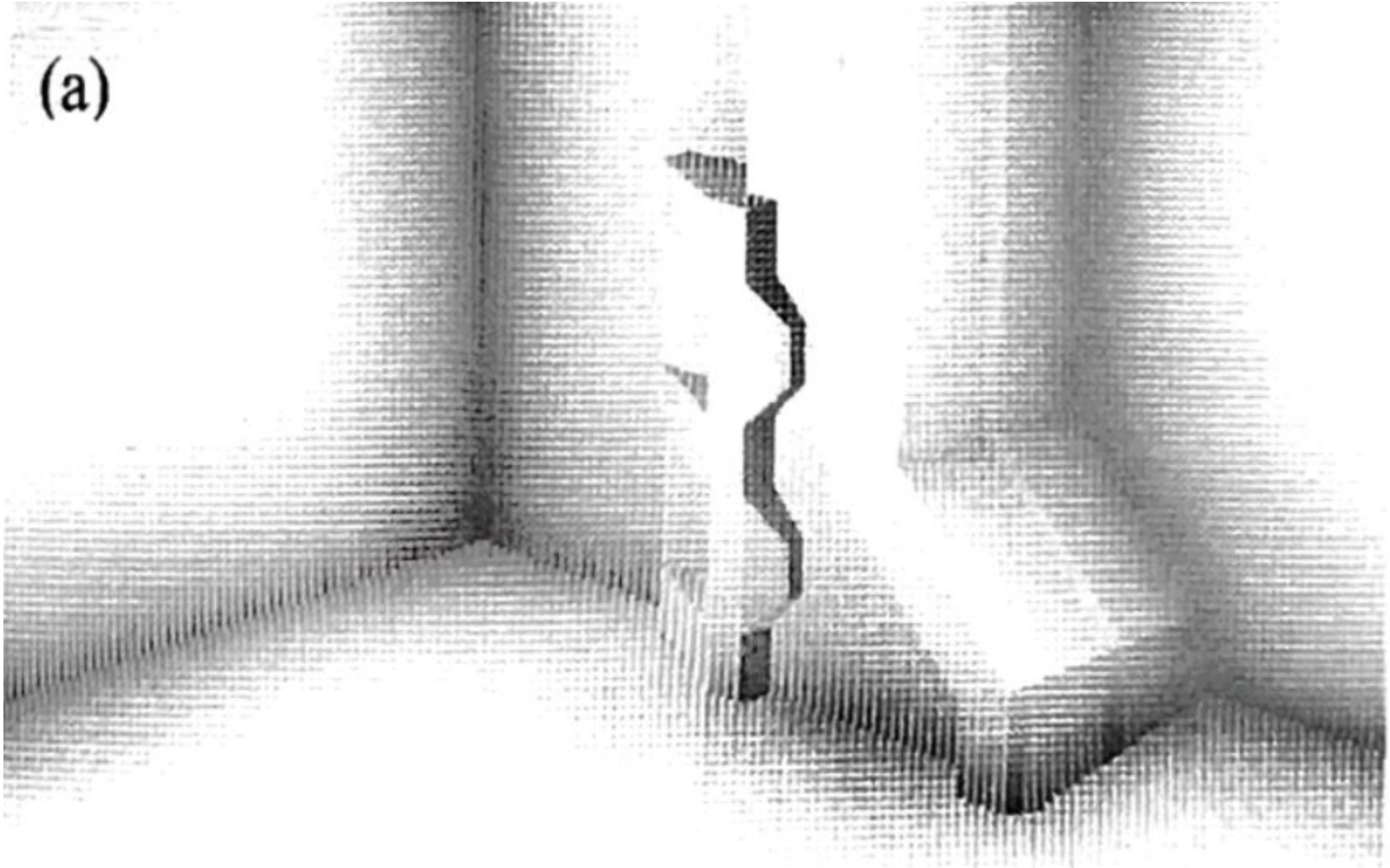
$$A = 1 - \sigma\left[\frac{\sum_{i=1}^{n} H(\mathbf{v}_i)}{\sum_{i=1}^{n} w(\mathbf{v}_i)}\right],$$

```
uniform TextureRect    structureBuffer;
uniform Texture2D      rotationTexture;
uniform float          vectorScale, intensity;

float CalculateAmbientOcclusion(float2 pixelCoord)
{
    const float kTangentTau = 0.03125;

    // These are the offset vectors used for the four samples.
    const float dx[4] = {0.1, 0.0, -0.3, 0.0};
    const float dy[4] = {0.0, 0.2, 0.0, -0.4};

    // Sample the structure buffer at the central pixel.
    float4 structure = texture(structureBuffer, pixelCoord);
    float z0 = structure.z + structure.w;

    // Calculate the normal vector.
    float scale = vectorScale * z0;
    float3 normal = normalize(float3(structure.xy, -scale));
    scale = 1.0 / scale;

    // Fetch a cos/sin pair from the 4x4 rotation texture.
    float2 rot = texture(rotationTexture, pixelCoord * 0.25).xy;
    float occlusion = 0.0;
    float weight = 0.0;

    for (int i = 0; i < 4; i++)
    {
        float3    v;

        // Calculate the rotated offset vector for this sample.
        v.x = rot.x * dx[i] - rot.y * dy[i];
        v.y = rot.y * dx[i] + rot.x * dy[i];

        // Fetch the depth from the structure buffer at the offset location.
        float2 depth = texture(structureBuffer, (pixelCoord + v.xy * scale)).zw;
        v.z = depth.x + depth.y - z0;

        // Calculate w(v) and f(v), and accumulate H(v) = w(v)f(v).
        float d = dot(normal, v);
        float w = saturate(1.0 - d * 0.5);
        float c = saturate(d * rsqrt(dot(v, v)) - kTangentTau);
```



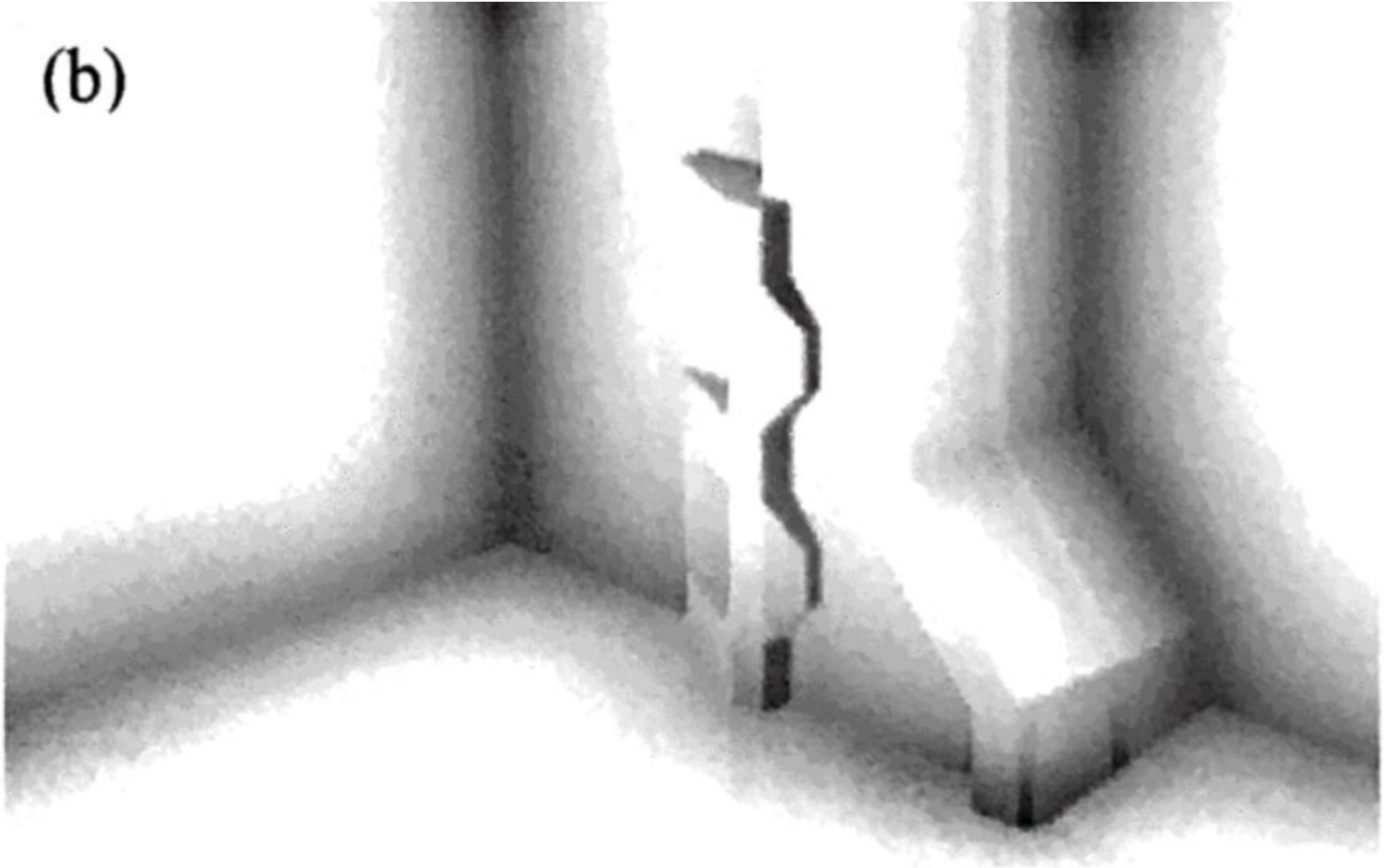
```
        occlusion += w - w * sqrt(1.0 - c * c);
        weight += w;
    }

    // Return the ambient light factor.
    return (1.0 - occlusion * intensity / max(weight, 0.0001));
}
```

(a)



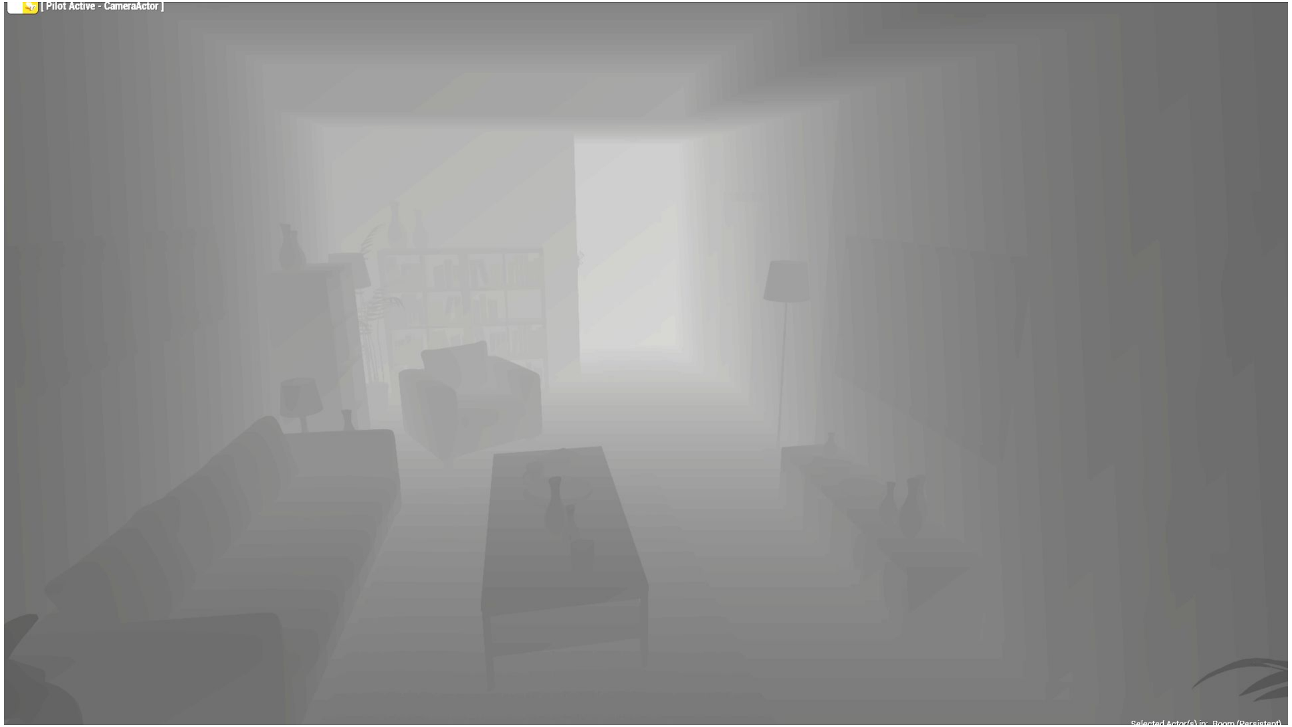
```
uniform TextureRect    structureBuffer;
uniform TextureRect    occlusionBuffer;

float BlurAmbientOcclusion(float2 pixelCoord)
{
   const float kDepthDelta = 0.0078125;

   // Use depth and gradient to calculate a valid range for the blur samples.
   float4 structure = texture(structureBuffer, pixelCoord);
   float range = (max(abs(structure.x), abs(structure.y)) + kDepthDelta) * 1.5;
   float z0 = structure.z + structure.w;

   float2 sample = float2(0.0, 1.0); float3 occlusion = float3(0.0, 0.0, 0.0);
   for (int j = 0; j < 2; j++)
   {
      float y = float(j * 2) - 0.5;
      for (int i = 0; i < 2; i++)
      {
         float x = float(i * 2) - 0.5;

         // Fetch a filtered sample and accumulate.
         float2 sampleCoord = pixelCoord + float2(x, y);
         sample.x = texture(occlusionBuffer, sampleCoord).x;
         occlusion.z += sample.x;

         // If depth at sample is in range, acculumate the occlusion value.
         float2 depth = texture(structureBuffer, sampleCoord).zw;
         if (abs(depth.x + depth.y - z0) < range) occlusion.xy += sample;
      }
   }

  // Divide the accumulated occlusion value by the number of samples that passed.
  return ((occlusion.y > 0.0) ? occlusion.x / occlusion.y : occlusion.z * 0.25);
}
```

(b)

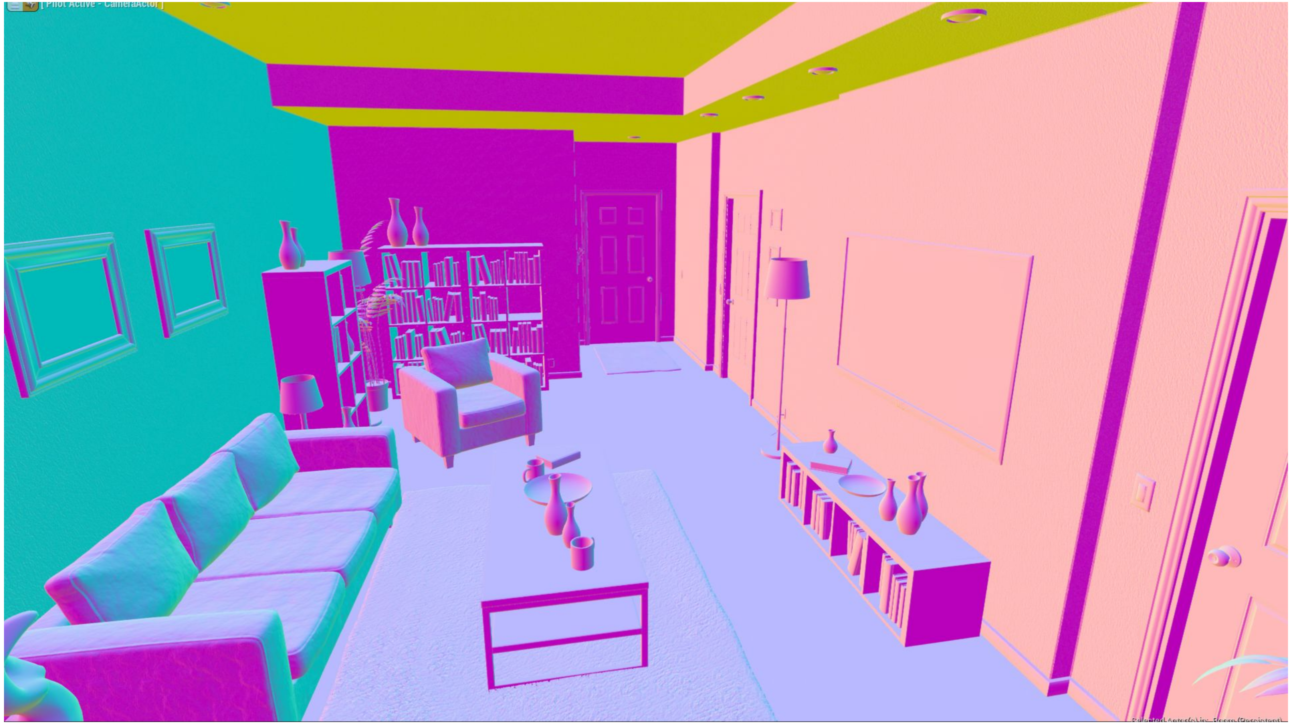


Place geometry into the scene

Calculate depth of the scene



Calculate normal values



The occlusion buffer

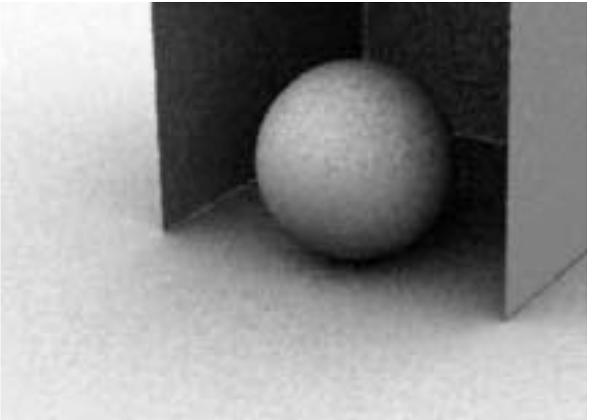
Combine with the rest of the buffers

Post process effects

The Evolution of Ambient Occlusion

The First Ambient Occlusion

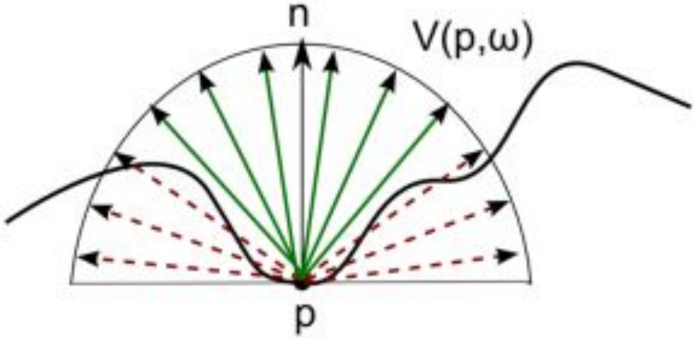
- In the real world, lights bounce around objects
- Simulate global illumination
- Shoots out rays from each point
- Too much noise and takes too long
- Needs a fast approximation for real-time rendering



Original model

With ambient occlusion

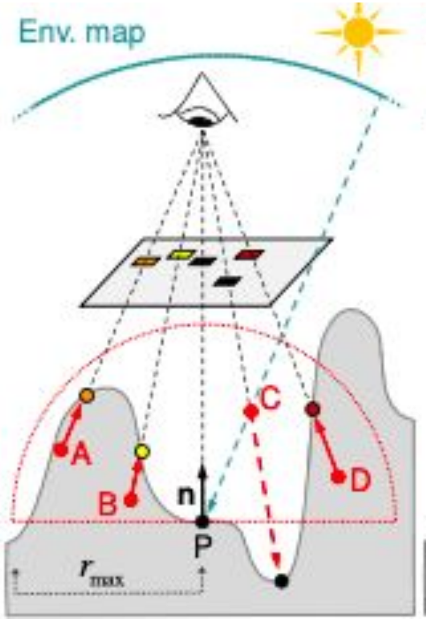
Extracted ambient occlusion map

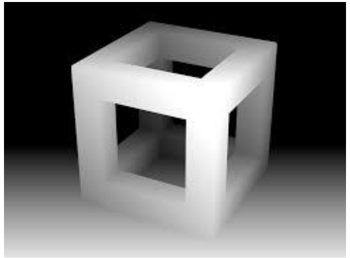


$$AO(\mathbf{p},\mathbf{n}) = \frac{1}{\pi}\int_{\Omega} V(\mathbf{p},\omega)\,\mathbf{n}\cdot\omega\, d\omega,$$

SSAO (Screen Space Ambient Occlusion) [Bavoil, 2008]

- Screen Space, independent of the complexity of the scene
- How much of the random sample points are occluded, using the depth map
- Blur to eliminate noise





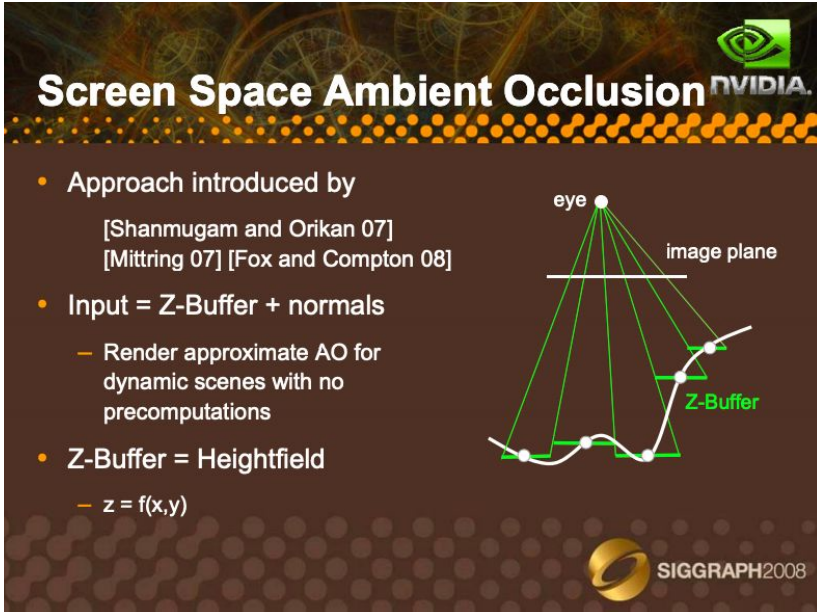
low sample 'banding'

random rotation = noise

+ blur = acceptable

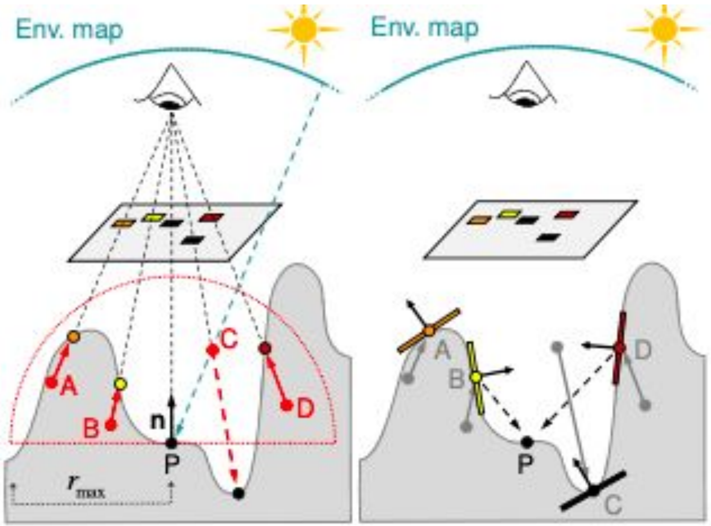
HBAO (Horizon-Based Ambient Occlusion) [Bavoil, 2008]

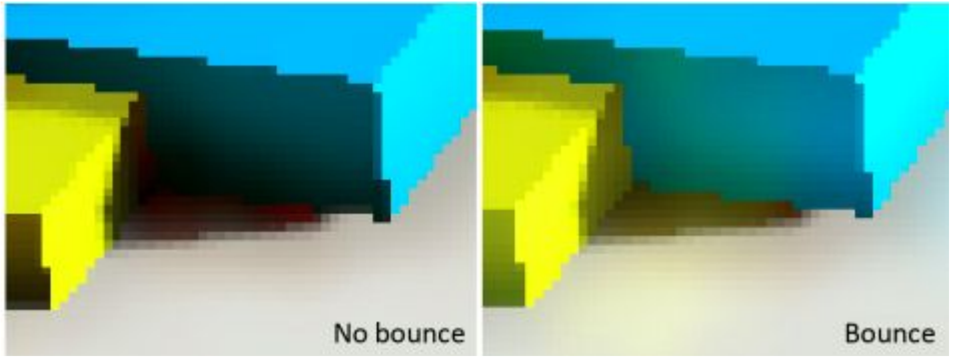
- Screen Space
- Choose a direction. Trace the height to approximate occlusion.
- Half resolution due to its slow computation and artifacts.
- HBAO+ overcomes these issues.



SSDO (Screen Space Directional Occlusion) [Ritschel, 2009]

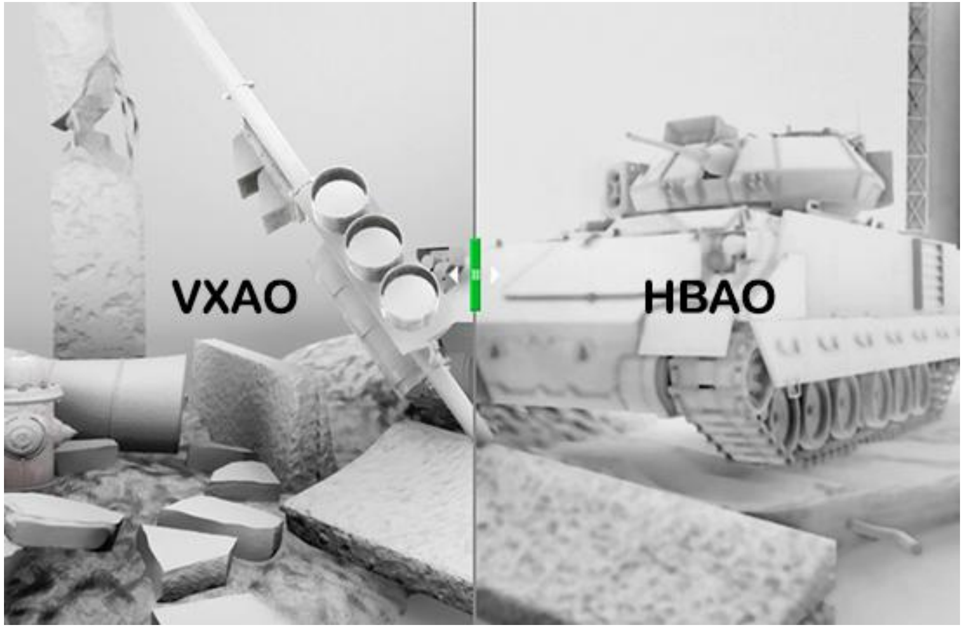
- Screen Space
- One bounce of indirect illumination





VXAO (Voxel Ambient Occlusion) [Penmatsa, 2012]

- Voxel Space (like a Spatial Partitioning acceleration structure)
- Divide the scene into small voxels

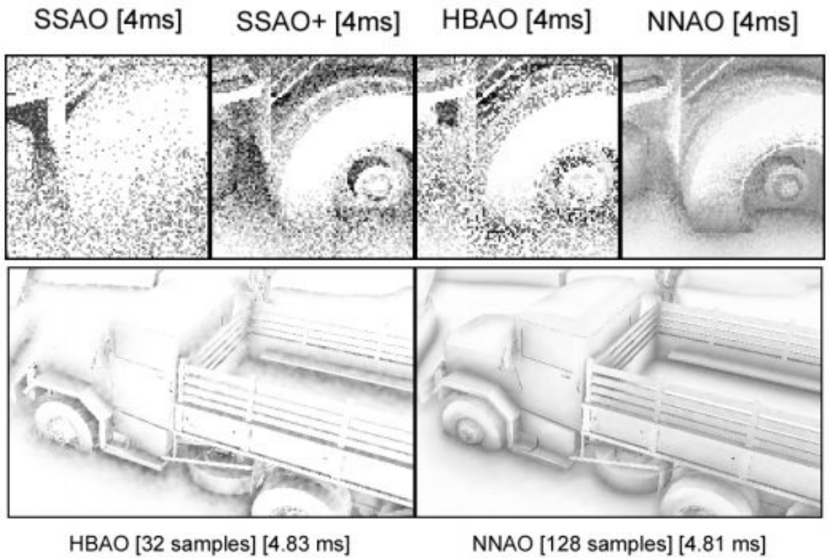


NNAO (Neural Network Ambient Occlusion) [Holden, 2016]

- Learns SSAO
- Fast => More sample => More accurate

Algorithm	Sample Count	Runtime (ms)	Error (mse)
SSAO	4	1.20	1.765
SSAO	8	1.43	1.558
SSAO	16	14.71	1.539
SSAO+	4	1.16	0.974
SSAO+	8	1.29	0.818
SSAO+	16	14.46	0.811
HBAO	16	3.53	0.965
HBAO	32	4.83	0.709
HBAO	64	8.50	0.666
NNAO	64	4.17	0.510
NNAO	128	4.81	0.486
NNAO	256	6.87	0.477

Table 1: *Numerical comparison between our method and others.*



NNAO (Neural Network Ambient Occlusion) [Holden, 2016]

- 500,000 independent pixels as a dataset (5 scenes x 150 vp x 1024 px)
- Learns the difference of normals and depth between nearby pixels.
- The trained model is too large to store in a shader.
- Some matrix operation for compression.

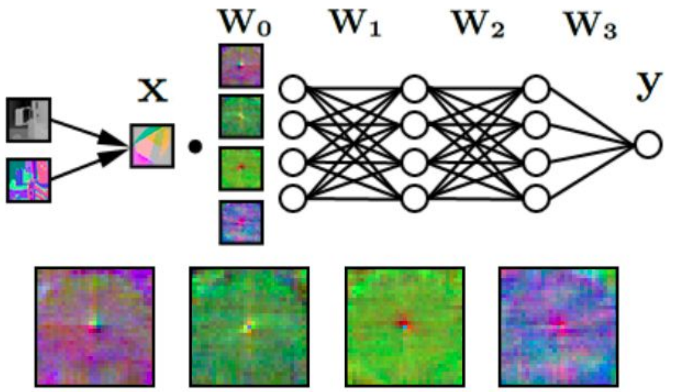


Figure 2: *Top: overview of our neural network. On the first layer four independent dot products are performed between the input and* $\mathbf{W}_0$ *represented as four 2D filters. The rest of the layers are standard neural network layers. Bottom: the four filter images extracted from* $\mathbf{W}_0$.

Back To Ray Tracing! (2018)

- SSAO is just an easy approximation
- SSAO can’t handle off-screen occlusion
- SSAO blurs the image to eliminate noise


Ambient Occlusion Ground Truth

Screen Space AO

Ambient Occlusion 2spp Denoised

Ambient Occlusion with 2spp


Ray Traced
Ambient Occlusion

References

Louis Bavoil, Miguel Sainz, “Screen Space Ambient Occlusion”, September 2008.

Louis Bavoil, Miguel Sainz, “Image-Space Horizon-Based Ambient Occlusion”, SIGGRAPH 2008, Talk Program, August 2008.

Tobias Ritschel, Thorsten Grosch, and Hans-Peter Seidel. Approximating dynamic global illumination in image space. In Proceedings of ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games 2009, pages 75–82, 2009.

Alexey Panteleev, “VXAO: Voxel Ambient Occlusion”, 2016
https://developer.nvidia.com/vxao-voxel-ambient-occlusion

Daniel Holden, Jun Saito, and Taku Komura. 2016. Neural Network Ambient Occlusion. In SIGGRAPH ASIA 2016 Technical Briefs (SA ’16). ACM, New York, NY, USA, Article 9, 4 pages. https://doi.org/10.1145/3005358.3005387

NVIDIA, “RTX Coffee Break: Ray Traced Ambient Occlusion (4:17 minutes)”, July 13, 2018
https://news.developer.nvidia.com/rtx-coffee-break-ray-traced-ambient-occlusion-417-minutes/