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)
A pixel shader that runs this:
\[ f(v) = 1 - \sqrt{1 - \max(n \cdot \hat{v} - \tau, 0)^2}. \]

\[ w(v) = \text{sat} \left( 1 - \frac{n \cdot v}{d_{\max}} \right), \]

\[ H(v) = w(v) f(v). \]

\[ A = 1 - \sigma \left[ \frac{\sum_{i=1}^{n} H(v_i)}{\sum_{i=1}^{n} w(v_i)} \right], \]
uniform Texture2D structureBuffer;
uniform Texture2D rotationTexture;
uniform float vectorScale, intensity;

float CalculateAmbientOcclusion(float2 pixelCoord)
{
    const float kTangentTau = 0.03115;

    // 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, 0.3, 0.1, -0.1);
    // Sample the structure buffer at the central pixel.
    float4 texture = texture(structureBuffer, pixelCoord);
    float4 depth = texture.z + structure.w;

    // Calculate the normal vector.
    float scale = vectorScale * 2.5f;
    float3 normal = normalize((float3(structure.xy, -scale)));
    scale = 1.0f / scale;

    // Fetch a cosine pair from the 4x rotation texture.
    float2 rotx = texture(rotationTexture, pixelCoord * 0.25).xy;
    float occlusion = 0.0f;
    float weight = 0.0f;

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

        // Calculate the rotated offset vector for this sample.
        v.x = rotx.x * dx[i] - rot.x * 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 - 2.0f;

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

        occlusion += w * c;
        weight += w;
    }

    // Return the ambient light factor.
    return (1.0f - occlusion * intensity / max(weight, 0.00001));
}
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, accumulate 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);
}
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
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
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

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

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 $W_0$ represented as four 2D filters. The rest of the layers are standard neural network layers. Bottom: the four filter images extracted from $W_0$. 
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
Ray Traced Ambient Occlusion


Alexey Panteleev, “VXAO: Voxel Ambient Occlusion”, 2016


NVIDIA, “RTX Coffee Break: Ray Traced Ambient Occlusion (4:17 minutes)”, July 13, 2018