**Screen Space Ambient Occlusion (SSAO)**

Mike Bailey
mjb@cs.oregonstate.edu

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License

**A Neat Global Illumination-ish Trick:**
Screen Space Ambient Occlusion (SSAO)

The idea is to imitate the darkness that appears in crevices that light has a hard time getting to.

**First, Create a GPU Memory Framebuffer**

```
// create a framebuffer object and a depth texture object:
glGenFramebuffers(1, &NZFramebuffer);
glGenTextures(1, &NZTexture);
glGenTextures(1, &NZDepth);
glBindFramebuffer(GL_FRAMEBUFFER, NZFramebuffer);
// create a texture that will be the framebuffer's color buffer (to store normal and z):
glBindTexture(GL_TEXTURE_2D, NZTexture);
gTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA32F, SSAO_WIDTH, SSAO_HEIGHT, 0, GL_RGBA, 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_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D, NZTexture, 0);
// create a texture that will be the framebuffer's depth buffer:
gBindTexture(GL_TEXTURE_2D, NZDepth);
gTexImage2D(GL_TEXTURE_2D, 0, GL_DEPTH_COMPONENT, SSAO_WIDTH, SSAO_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_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
glFramebufferTexture2D(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, GL_TEXTURE_2D, NZDepth, 0);
gBindFramebuffer(GL_FRAMEBUFFER, 0);
```

SSAO is a Two-pass Algorithm:

**Pass #1: Render the Surface Normals and Depths into a GPU Memory Framebuffer**

```
// create a fragment shader for a depth texture object
#include < fileInfo.frag >
#include < fileInfo.vert >
#include < fileInfo.geom >

// note: in this pass, we are not rendering any colors, so no lighting info is necessary
// however, transformation matrix info is necessary so that the scene is in the right orientation to get the normal and z

uniform mat4 uAnim;
uniform mat4 uModelView;
uniform mat4 uProj;

out vec3 vN;
out vec4 vP;

void main()
{
    vN = normalize( vec3( uAnim * vec4(gl_Normal, 0.) ) );
    // we want the normal in model coordinates, not world coords or eye coords
    vP = uProj * uModelView * uAnim * gl_Vertex;
    // we want the z in eye coordinates because we need to divide by the .w
    gl_Position = vP;
}
```

```
// create a fragment shader for a depth texture object
#include < fileInfo.frag >
#include < fileInfo.vert >
#include < fileInfo.geom >

// note: in this pass, we are not rendering any colors, so no lighting info is necessary
// however, transformation matrix info is necessary so that the scene is in the right orientation to get the normal and z

in vec3 vN;
in vec4 vP;

void main()
{
    vec3 N = normalize(vN);             // in the range 0. to +1.
    float Z = vP.z / vP.w;              // in the range -1. to +1.
    gl_FragColor = vec4(N, Z);         // this gets written into the uNZMap texture
}
```

The part of the scene should be darker because it is harder for ambient light to get down between objects.

**Pass #2: Use theFramebuffer**

```
// render nothing into the framebuffer because we only want the normal and depth
// make the ambient occlusion shader
#include < ambientOcclusion.frag >
```

```
// render nothing into the framebuffer because we only want the normal and depth
// make the ambient occlusion shader
#include < ambientOcclusion.frag >
```

This part of the scene should be darker because it is harder for ambient light to get down between objects.
Here is the Scene and What the Normal Vectors Look Like, Rendered as Colors

Here is the Scene and What the Z Values Look Like, Rendered as Grayscale

Pass #2: Render the 3D Scene and Adjust the Lighting Anywhere the SSAO Conditions are Right

Pass #2: Render the 3D Scene and Adjust the Lighting Anywhere the SSAO Conditions are Right

Pass #2: Render the 3D Scene and Adjust the Lighting Anywhere the SSAO Conditions are Right

Pass #2: Render the 3D Scene and Adjust the Lighting Anywhere the SSAO Conditions are Right

Pass #2: Render the 3D Scene and Adjust the Lighting Anywhere the SSAO Conditions are Right
How to Display the Colored Normal Vectors

**DisplayNMap.vert**

#version 330 compatibility
uniform mat4 uModel;
uniform mat4 uView;
uniform mat4 uProj;
out vec4 vP;
void
main()
{
    vP = uProj * uView * uModel * gl_Vertex;
    gl_Position = vP;
}

**DisplayNMap.frag**

#version 330 compatibility
uniform sampler2D uNZMap;
in vec4 vP;
void
main()
{
    vec2 st = vP.xy / vP.w; // in the range -1. to 1.
    st = vec2(0.5,0.5) * ( st + vec2(1.,1.) );
    // now in the range 0. to 1. so can use it for a texture lookup
    // get the normal at the uNZMap texture coords (st.s,st.t)
    vec3 n = texture( uNZMap, st ).rgb; // in the range -1. to +1.
    n = abs(n); // in the range 0. to +1.
    gl_FragColor = vec4( n, 1. ); // display normal components as rgb
}

How to Display the Grayscale Depth Values

**DisplayZMap.vert**

#version 330 compatibility
uniform mat4 uModel;
uniform mat4 uView;
uniform mat4 uProj;
out vec4 vP;
void
main()
{
    vP = uProj * uView * uModel * gl_Vertex;
    gl_Position = vP;
}

**DisplayZMap.frag**

#version 330 compatibility
uniform sampler2D uNZMap;
in vec4 vP;
void
main()
{
    vec2 st = vP.xy / vP.w; // in the range -1. to 1.
    st = vec2(0.5,0.5) * ( st + vec2(1.,1.) );
    // now in the range 0. to 1. so can use it for a texture lookup
    // get the normal at the uNZMap texture coords (st.s,st.t)
    float Z = texture( uNZMap, st ).a;
    Z = 0.5 * ( Z + 1. ); // now in the range 0. to 1.
    gl_FragColor = vec4( Z, Z, Z, 1. ); // greyscale
}