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

"Render" these normal and depth values into a software framebuffer. Now, look for places in the framebuffer where the depths are about the same, but there is a discontinuity in the normal. Make that part of the scene darker.

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

```c
// create a framebuffer object and a depth texture object:
glGenFramebuffers(1, &NZFramebuffer);
glGenTextures(1, &NZTexture);
glGenTextures(1, &NZDepth);

// 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);
gTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
gTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
gTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
gTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);

// 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);
gTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
gTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
gTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
gTexParameteri(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 color buffer (to store normal and z):
gBindFramebuffer(GL_FRAMEBUFFER, NZFramebuffer);
```

First, Create a GPU Memory Framebuffer
SSAO is a Two-pass Algorithm:
Pass #1: Render the Surface Normals and Depths into a GPU Memory Framebuffer

GetNZ.vert

```cpp
#version 330 compatibility

// 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;
}
```

GetNZ.frag

```cpp
#version 330 compatibility

// note: in this pass, we are not rendering any colors, so no lighting info is necessary

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
}
```
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

**RenderWithNZ.vert**

```cpp
#version 330 compatibility
uniform mat4 uAnim;
uniform mat4 uModelView;
uniform mat4 uProj;
uniform float uLightX;
uniform float uLightY;
uniform float uLightZ;

out vec3 vNs;
out vec3 vLs;
out vec3 vEs;
out vec4 vP;

void main()
{
    vec3 LightPosition = vec3(uLightX, uLightY, uLightZ);
vec4 ECposition = uModelView * uAnim * gl_Vertex;
vec3 tnorm = normalize( mat3(uModelView*uAnim) * gl_Normal );
vNs = tnorm;
vLs = LightPosition - ECposition.xyz;
vEs = vec3( 0., 0., 0. ) - ECposition.xyz;
vP = uProj * uModelView * uAnim * gl_Vertex;  // need the projection matrix to put z values in the proper range
    gl_Position = vP;
}
```

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

**RenderWithNZ.frag, 1**

```cpp
#version 330 compatibility

uniform vec3 uColor;
uniform sampler2D uNZMap;
uniform int uSSAOOn;

in vec3 vNs;
in vec3 vLs;
in vec3 vEs;
in vec4 vP;

const float uDelta = 0.0050;
const float uZTol = 0.0010;
const float uNTol = -0.00001;

float Ds, Dt;  // change in s and t to get the next texel
int Nums = 15;  // must be an odd number
int Numt = 15;  // must be an odd number

const float MININTEN = 0.50;
const vec3 SPECULAR_COLOR = vec3(1., 1., 1.);
const float SHININESS = 8;
const float KA = 0.20;
const float KD = 0.60;
const float KS = (1.-KA-KD);
```
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

**RenderWithNZ.frag, II**

```cpp
float IsInACrevice( vec2 st )
{
    float inten = 1.;
    // get the normals and z's:
    vec4 rgba00 = texture(uNZMap, st );
    vec3 n00 = normalize(rgba00.rgb);
    float z00 = rgba00.a;
    for( int it = -Numt/2; it <= Numt/2; it++ )
    {
        vec2 newst;
        newst.t = st.t + Dt * float(it);
        for( int is = -Nums/2; is <= Nums/2; is++ )
        {
            newst.s = st.s + Ds * float(is);
            vec4 rgba = texture(uNZMap, newst);
            vec3 n = normalize(rgba.rgb);
            float z = rgba.a;
            // we are in a crevice if:
            // 1. the surrounding z values are approximately the same, and,
            // 2. the surrounding normals are approximately different
            if( abs(z00-z) < uZTol && dot( n00, n ) < uNTol )
                {
                    float i = 1. - MININTEN * float( abs(is) + abs(it) ) / float( Nums/2 + Numt/2 );
                    if( i < inten )
                        inten = i;  // keep the minimum
                }
        }
    }
    return inten;
}
```

**RenderWithNZ.frag, II**

**RenderWithNZ.frag, III**

```cpp
void main()
{
    vec2 numtexels = textureSize( uNZMap, 0 );
    Ds = 1. / numtexels.s;
    Dt = 1. / numtexels.t;
    vec3 normal = normalize(vNs);
    vec3 light = normalize(vLs);
    vec3 eye = normalize(vEs);

    // have to manually do homogenous division to make position in range of -1 to 1:
    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 as a texture lookup

    // get the normal and z at the uNZMap texture coords (st.s,st.t)
    float Inten = 1.;
    if( uSSAOOn != 0 )
        Inten = IsInACrevice( st );
    vec3 lighting = KA * uColor;
    float d = dot(normal,light);

    . . . Lighting . . .
    gl_FragColor = vec4( Inten * lighting, 1. );
}
```
#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;
}

#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 Colored Normal Vectors

DisplayNMap.vert

DisplayNMap.frag
#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;
}

How to Display the Grayscale Depth Values

DisplayZMap.vert

How to Display the Grayscale Depth Values

DisplayZMap.frag

How to Display the Grayscale Depth Values

DisplayZMap.frag