Shader Steps

In the global variables:

```
GLuint          SalmonDL;
float           Time;              // same as we used before
```

Near the top of the program:

```
//#include "setmaterial.cpp"
//#include "setlight.cpp"
//#include "osusphere.cpp"
//#include "osucone.cpp"
//#include "osutorus.cpp"
//#include "bmptotexture.cpp"
#include "loadobjfile.cpp"
//#include "keytime.cpp"
#include "glslprogram.cpp"
```

Right after those #includes:

```
GLSLProgram   Salmon;          // your VS+FS shader program name
```

In InitGraphics( ):

```
Salmon.Init( );
bool valid = Salmon.Create( "salmon.vert",  "salmon.frag" );
if( ! valid )
{
        fprintf( stderr, "Yuch!  The Salmon shader did not compile.\n" );

}
else
{
        fprintf( stderr, "Woo-Hoo!  The Salmon shader compiled.\n" );
}


Salmon.SetUniformVariable( "uKa", 0.1f );               // all 3 should add up to 1.0
Salmon.SetUniformVariable( "uKd",  ??? );
Salmon.SetUniformVariable( "uKs",  ??? );
Salmon.SetUniformVariable( "uShininess", ??? );        // whatever you like from P3
```

In InitLists( ):

```
SalmonDL = glGenLists( 1 );
glNewList( SalmonDL, GL_COMPILE );
    LoadObjFile( (char *) "salmon.obj" );
glEndList( );
```

In Display( ):

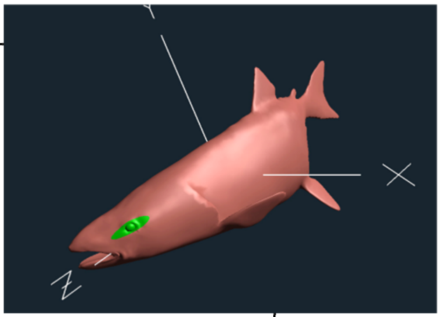
```
Salmon.Use( ); // turns the Salmon shader program on
                               // no more fixed-function – the shader Salmon now handles everything
                               // but the shader program just sits there idling until you draw something

float amp  = <<some function of time>>           // sine wave amplitude
float freq  = <<some function of time>>          // sine wave frequency
foat speed = <<some function of time >>          // oberall speed of movement
          . . .
Salmon.SetUniformVariable( "uTime", Time);                 // 0.-1., set in Animate( )
Salmon.SetUniformVariable( "uAmp",  amp );                 // keytimed perhaps?
Salmon.SetUniformVariable( "uSpeed",speed );               // keytimed perhaps?
Salmon.SetUniformVariable( "uFreq",  freq);                // keytimed perhaps?

glCallList( SalmonDL );  // now the shader program has vertices and fragments to work on

Salmon.UnUse( );                    // go back to fixed-function OpenGL
```

salmon.vert:



```
#version 330 compatibility
uniform float   uTime;
uniform float   uAmp;
uniform float   uSpeed;
uniform float   uFreq;

out  vec2      vST;          // texture coords
out  vec3      vN;           // surface normal vector
out  vec3      vL;           // vector from point to light
out  vec3      vE;           // vector from point to eye

const vec3     LIGHTPOS       = vec3(  10., 10., 5. );        // light position
const float     PI              = 3.14159265;
const float    TWOPI            = 2.*PI;
const float    LENGTH           = 5.;                         // salmon length

void  main( )
{
      vST = gl_MultiTexCoord0.st;
      vec3 vert = gl_Vertex.xyz;
       // which direction on the salmon will do the wriggling?
       // what multiplies time to get distance (wriggled)
       // what multiplies position to get how many wriggles we see?
      vert.? += uAmp * sin( TWOPI*( (???*uTime)+(???*vert.z/LENGTH) ) );


      // setup for the per-fragment lighting:
      vec4 ECposition = gl_ModelViewMatrix * vec4( vert, 1. );
      vN = normalize( gl_NormalMatrix * gl_Normal );  // surface normal vector
      vL = LIGHTPOS - ECposition.xyz;                 // vector from the point to the light position
      vE = vec3( 0., 0., 0. ) - ECposition.xyz;       // vector from the point to the eye position
      gl_Position = gl_ModelViewProjectionMatrix * vec4( vert, 1. );
}
```

salmon.frag:

```
#version 330 compatibility

uniform float   uKa, uKd, uKs;              // coefficients of each type of lighting
uniform float   uShininess;                 // specular exponent


in  vec2  vST;                              // texture coords of the current fragment
in  vec3  vN;                               // surface normal vector of the current fragment
in  vec3  vL;                               // vector from current fragment to the light
in  vec3  vE;                               // vector from current fragment to our eye

const float EYES                 = 0.80;                        // not correct!
const float EYET                 = 0.50;                        // not correct!
const float R                    = 0.03;                        // radius of salmon eye
const vec3 SALMONCOLOR           = vec3( 0.98, 0.50, 0.45 );    // "salmon" (r,g,b) color
const vec3 EYECOLOR              = vec3( 0., 1., 0. );          // color to make the eye
const vec3 SPECULARCOLOR         = vec3( 1., 1., 1. );

void
main( )
{
    vec3 myColor = SALMONCOLOR;         // color if not in the eye
    float ds = ?????;                   // s distance from current frag to salmon eye
    float dt  = ?????;                  // t distance from current frag to salmon eye
    if( <<we are within the eye circle>>)
    {
         myColor = EYECOLOR;
    }


     // now do the per-fragment lighting:

     vec3 Normal    = normalize(vN);
     vec3 Light      = normalize(vL);
     vec3 Eye        = normalize(vE);

     vec3 ambient = uKa * myColor;

     float d = max( dot(Normal,Light), 0. );      // only do diffuse if the light can see the point
     vec3 diffuse = uKd * d * myColor;

    float s = 0.;
    if( d > 0. )                                  // only do specular if the light can see the point
    {
        vec3 ref = normalize(  reflect( -Light, Normal )  );    // perfect reflection vector
        float cosphi = dot( Eye, ref );
        if( cosphi > 0. )
             s = pow( max( cosphi, 0. ), uShininess );
    }
    vec3 specular = uKs * s * SPECULARCOLOR.rgb;
    gl_FragColor = vec4( ambient + diffuse + specular,  1. );
}
```