Project 3

Alan Neads

7th February, 2017

neadsa_noisy_dots.glib

```
##OpenGL GLIB
LookAt  5 0 5  0 0 0  0 1 0
Perspective 70

Vertex   neadsa_noisy_dots.vert
Fragment neadsa_noisy_dots.frag
Program  neadsa_noisy_dots \
        uAd <.01 .1 .5> \
        uBd <.01 .15 .5> \
        uNoiseAmp <0.0 1.0 10.0> \
        uNoiseFreq <0.0 1.0 10.0> \
        uAlpha <0.0 1.0 1.0> \
        uTol <0.001 0.01 1.0> \
        uUseLighting <true> \
        uUseChromaDepth <false> \
        uChromaRed <-10.0 -6.7, 0.0> \
        uChromaBlue <-20.0 -9.4, 0.0>

Color 1.0 1.0 1.0
Sphere 3

MessageBox This implements Alpha and ChromaDepth extra credit
MessageBox Please note that lighting and ChromaDepth are mutually exclusive options
```

neadsa_noisy_dots.vert

```
out vec4 Color;
out vec3 vPosition;
out float depth;

out vec3 modelVec;
out vec3 modelNormalVec;

void main( void )
{
	Color = gl_Color;
	gl_TexCoord[0] = gl_MultiTexCoord0;
	gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
	vPosition = gl_Vertex.xyz;

	depth = (gl_ModelViewMatrix * gl_Vertex).z;

	modelVec = vec3(gl_ModelViewMatrix * gl_Vertex);
	modelNormalVec = normalize(gl_NormalMatrix * gl_Normal);
}
```

neadsa_noisy_dots.frag

```
in vec4 Color;
in vec3 vPosition;
in float depth;

in vec3 modelVec;
in vec3 modelNormalVec;

uniform sampler3D Noise3;

uniform float uAd;
uniform float uBd;

uniform float uNoiseAmp;
uniform float uNoiseFreq;

uniform float uAlpha;

uniform float uTol;

uniform bool uUseLighting;

uniform bool uUseChromaDepth;
uniform float uChromaRed;
uniform float uChromaBlue;

const vec3 lightPos = vec3( -3.0, 0.0, 3.0 );
const vec3 ambientColor = vec3( 0.0, 0.0, 0.0 );
const vec3 specularColor = vec3( 0.4, 0.4, 0.4 );
const float specularShininess = 1000.0;

vec3 Rainbow( float t )
{
	t = clamp( t, 0., 1. );

	float r = 1.;
	float g = 0.0;
	float b = 1.  -  6. * ( t - (5./6.) );

	if( t <= (5./6.) )
	{
		r = 6. * ( t - (4./6.) );
		g = 0.;
		b = 1.;
	}
```

```
	if( t <= (4./6.) )
	{
			r = 0.;
			g = 1.  -  6. * ( t - (3./6.) );
			b = 1.;
	}

	if( t <= (3./6.) )
	{
			r = 0.;
			g = 1.;
			b = 6. * ( t - (2./6.) );
	}

	if( t <= (2./6.) )
	{
			r = 1.  -  6. * ( t - (1./6.) );
			g = 1.;
			b = 0.;
	}

	if( t <= (1./6.) )
	{
			r = 1.;
			g = 6. * t;
	}

	return vec3( r, g, b );
}

void main( void )
{
	vec4 nv = texture3D(Noise3, vPosition * uNoiseFreq);
	float sum = nv[0] + nv[1] + nv[2] + nv[3];
	float noise = ((sum - 1.0) / 2.0) - 0.5;
	noise *= uNoiseAmp;

	float up = 2 * gl_TexCoord[0].s;
	float vp = gl_TexCoord[0].t;

	float numinu = floor(up / uAd);
	float numinv = floor(vp / uBd);

	float uc = numinu * uAd + uAd / 2.0;
	float vc = numinv * uBd + uBd / 2.0;
```

```
	float du = up - uc;
	float dv = vp - vc;

	float oldrad = sqrt( du*du + dv*dv );
	float newrad = oldrad + noise;
	float ratio = newrad / oldrad;

	du *= ratio;
	dv *= ratio;

	float d = (du * 2.0 / uAd) * (du * 2.0 / uAd) + (dv * 2.0 / uBd) * (dv * 2.0 / uBd);
	float step = smoothstep( 0.0, uTol, 1.0 - d );

	vec4 ellipseColor = vec4( 1.0, 0.5, 0.0, 1.0 );
	if ( uUseChromaDepth ) {
		float t = (2.0 / 3.0) * (depth - uChromaRed) / (uChromaBlue - uChromaRed);
		t = clamp( t, 0.0, 2.0 / 3.0 );
		ellipseColor.xyz = Rainbow(t);
	}

	gl_FragColor = mix( Color, ellipseColor, step );

	if ( step < 1.0 ) {
		if ( uAlpha == 0.0 ) {
			discard;
		} else {
			gl_FragColor.a = uAlpha;
		}
	}

	if ( uUseLighting && !uUseChromaDepth ) {
		vec3 L = normalize( lightPos - modelVec );
		vec3 E = normalize(-modelVec);
		vec3 R = normalize(-reflect(L,modelNormalVec));

		vec3 diffuse = gl_FragColor.xyz * max(dot(modelNormalVec,L), 0.0);
		diffuse = clamp( diffuse, 0.0, 1.0 );

		vec3 specular = specularColor * pow(max(dot(R,E), 0.0), specularShininess);
		specular = clamp( specular, 0.0, 1.0 );

		gl_FragColor.xyz = ambientColor + diffuse + specular;
		gl_FragColor.xyz = clamp( gl_FragColor.xyz, 0.0, 1.0 );
	}
}
```

Deconstruction

I took my implementation of noisy ellipses from project one and two, modifying it to work within OpenGL's Shader Language instead. I setup the required variables, implementing alpha and chrome depth support. The noisy ellipses with modifiable frequency and amplitude is easily accomplished, the frequency multiplies the input position into the noise function, while the amplitude multiplies the resulting sum of the output. This is then added to modify the ellipse by adjusting the boundary of the ellipse by adjusting the radius of the ellipse, then using that ratio to modify our input positions. We then compute the distance from the center of the ellipse and coloring appropriately. We opt to smoothstep the result using the width of the uTol, then using the output of smoothstep to determine which color to either blend, if uTol is non-zero, or simply pick one or the other of the two colors. To implement chromadepth we simply pass the depth value from the vertex shader, multiplying it by appropriate distances and then passing it into the rainbow function. We then opt to use this color instead of the predetermined beaver orange for our ellipse color. Then for lighting I opted to implement Phong lighting, utilizing the OpenGL site as a reference. Essentially you take the position of the light, versus the viewing angle, if the light is capable of being reflected into the viewer's eye we opt to create a reflective patch called specular light. Everything else is computed via diffuse lighting compared to the surface normal against the light source, then we add the ambient light to that value. However, lighting is mutually exclusive with chromadepth, as it ruins the effect and also shades interior surfaces as black, as they are not within the possible reflection angles.

Images