Project 7

Alan Neads

15th March, 2017

bezier.glib

```
##OpenGL GLIB

Perspective 70
LookAt 2 -1 1 0 2 -1 0 0 1

Vertex mtriangle.vert
TessControl mtriangle.tcs
TessEvaluation mtriangle.tes
Geometry mtriangle.geom
Fragment mtriangle.frag

Program MidTriangle \
                uOuter01 <0.5 5. 50.>           \
                uOuter12 <0.5 5. 50.>           \
                uOuter20 <0.5 5. 50.>           \
                uInner <0.5 5. 20.>                     \
                uZ01 <-2. 0. 2.>                        \
                uZ12 <-2. 0. 2.>                        \
                uZ20 <-2. 0. 2.>                        \
                uAdaptToZs <false>                      \
                uShrink <0. 0.9 1.>                     \
                uSurfaceColor {1. 1. 0.3 1.}\
                uSpecularColor {1. 1. 1. 1.}\
                uKa <0. 0.1 1.0>                        \
                uKd <0. 0.7 1.0>                        \
                uKs <0. 0.2 1.0>                        \
                uShininess <3. 10. 1000.>       \
                uLightX <-10. 0. 10.>           \
                uLightY <-10. 8. 10.>           \
                uLightZ <-10. 8. 10.>

Color 1. .5 0.
NumPatchVertices 3
glBegin gl_patches
        glVertex 0. 0. 0.
        glVertex 2. 0. 0.
        glVertex 0. 2. 0.
glEnd
```

mtriangle.vert

```
void main()
{
	gl_Position = gl_Vertex;
}
```

mtriangle.frag

```
#version 400 compatibility

in vec3 gNs;
in vec3 gLs;
in vec3 gEs;

uniform float uKa, uKd, uKs;
uniform float uShininess;

uniform vec4 uSurfaceColor;
uniform vec4 uSpecularColor;

void main()
{
	vec3 norm = normalize(gNs);
	vec3 light = normalize(gLs);
	vec3 eye = normalize(gEs);

	float dotprod = dot( norm, light );

	vec3 amb = uKa * uSurfaceColor.rgb;
	vec3 diff = uKd * uSurfaceColor.rgb;
	vec3 spec = uKs * uSpecularColor.rgb;

	if ( dotprod > 0.0 ) {
		spec *= pow( max(dot(eye,normalize(2.0 * norm * dot(norm, light) - light)), 0.0),
uShininess );
	} else {
		spec = vec3(0.0, 0.0, 0.0);
	}

	gl_FragColor.rgb = amb + diff + spec;
	gl_FragColor.a = 1.0;
}
```

mtriangle.tcs

```
#version 400 compatibility
#extension GL_ARB_tessellation_shader : enable

uniform float uZ01, uZ12, uZ20;
uniform int uOuter01, uOuter12, uOuter20, uInner;
uniform bool uAdaptToZs;

layout( vertices = 3 ) out;

void main()
{
	gl_out[ gl_InvocationID ].gl_Position = gl_in[ gl_InvocationID ].gl_Position;

	if( uAdaptToZs ) {
		gl_TessLevelOuter[0] = float(uOuter12) + abs(uZ12 * 3) * float(uOuter12);
		gl_TessLevelOuter[1] = float(uOuter20) + abs(uZ20 * 3) * float(uOuter20);
		gl_TessLevelOuter[2] = float(uOuter01) + abs(uZ01 * 3) * float(uOuter01);
		gl_TessLevelInner[0] = gl_TessLevelInner[1] = float(uInner);
	} else {
		gl_TessLevelOuter[0] = float(uOuter12);
		gl_TessLevelOuter[1] = float(uOuter20);
		gl_TessLevelOuter[2] = float(uOuter01);
		gl_TessLevelInner[0] = gl_TessLevelInner[1] = float(uInner);
	}
}
```

mtriangle.tes

```
#version 400 compatibility
#extension GL_ARB_tessellation_shader : enable
layout( triangles, equal_spacing, ccw) in;
uniform float uZ01, uZ12, uZ20;
out vec3 teNormal;
out vec3 teECposition;
void main()
{
	vec4 p0 = gl_in[0].gl_Position;
	vec4 p1 = gl_in[1].gl_Position;
	vec4 p2 = gl_in[2].gl_Position;
	vec4 p3 = gl_in[3].gl_Position;
	vec4 p01 = vec4( (p0.x + p1.x)/2.0, (p0.y + p1.y)/2.0, uZ01, 1.0 );
	vec4 p12 = vec4( (p2.x + p1.x)/2.0, (p2.y + p1.y)/2.0, uZ12, 1.0 );
	vec4 p20 = vec4( (p0.x + p2.x)/2.0, (p0.y + p2.y)/2.0, uZ20, 1.0 );
	float u = gl_TessCoord.x;
	float v = gl_TessCoord.y;
	float w = gl_TessCoord.z;
	float b0 = pow( u, 2.0 );
	float b1 = pow( v, 2.0 );
	float b2 = pow( 1.0 - u - v, 2.0 );
	float b01 = 2.0 * u * v;
	float b12 = 2.0 * v * (v - u * v - pow(v, 2.0));
	float b20 = 2 * (u - pow(u, 2.0) - u * v);
	float db0du = 2.*u;
	float db0dv = 0.;
	float db1du = 0.;
	float db1dv = 2.*v;
	float db2du = -2.*(1.-u-v);
	float db2dv = -2.*(1.-u-v);
	float db01du = 2.*v;
	float db01dv = 2.*u;
	float db12du = -2.*v;
	float db12dv = 2.*(1.-u-2.*v);
	float db20du = 2.*(1.-2.*u-v);
	float db20dv = -2.*u;

	teECposition = ( gl_ModelViewMatrix * ( b0*p0 + b01*p01 + b1*p1 + b12*p12 + b2*p2 +
b20*p20 ) ).xyz;
	gl_Position = vec4( teECposition, 1. );
	vec4 dpdu = db0du*p0 + db01du*p01 + db1du*p1 + db12du*p12 + db2du*p2 +
db20du*p20;
	vec4 dpdv = db0dv*p0 + db01dv*p01 + db1dv*p1 + db12dv*p12 + db2dv*p2 + db20dv*p20;
	teNormal = gl_NormalMatrix * normalize( cross( dpdu.xyz, dpdv.xyz ) );
}
```

mtriangle.geom

```
#version 400 compatibility
#extension GL_EXT_gpu_shader4: enable
#extension GL_EXT_geometry_shader4: enable

layout( triangles ) in;
layout( triangle_strip, max_vertices=32 ) out;

uniform float uShrink;
uniform float uLightX, uLightY, uLightZ;

in vec3 teECposition[3];
in vec3 teNormal[3];

out vec3 gNs;
out vec3 gLs;
out vec3 gEs;

vec3 LightPos = vec3( uLightX, uLightY, uLightZ );
vec3 V[3];
vec3 CG;

void ProduceVertex( int v )
{
	gNs = teNormal[v];
	gLs = LightPos - teECposition[v];
	gEs = vec3(0.,0.,0.) - teECposition[v];
	gl_Position = gl_ProjectionMatrix * vec4( CG + uShrink * ( V[v] - CG ), 1. );
	EmitVertex();
}

void main()
{
	V[0] = gl_PositionIn[0].xyz;
	V[1] = gl_PositionIn[1].xyz;
	V[2] = gl_PositionIn[2].xyz;

	CG = ( V[0] + V[1] + V[2] ) / 3.;

	ProduceVertex( 0 );
	ProduceVertex( 1 );
	ProduceVertex( 2 );
}
```

Deconstruction

To create the bezier triangle surface I started with the basis provided in the project description, and substituted in the expected values. For the tessellation control shader I had to design the function to automatically adapt to the changes in the Z points on the triangle, thus I opted to make it such that as the value increased, it added an increased tessellation value to the outer edges appropriately. This makes it so that as the Z point increases or decreases away from zero it increases the tessellation, however I ran into an issue with when the values were zero it would simply not render anything. I realized this was due to the Z values being zero, thus indicating there should be no tessellation along the edge. I solved this by adding the current edge tessellation value back into the function.

To determine the tessellation vertex positions it was just a modification of the examples in the tessellation handout. It was a matter of finding the contribution from each point at the vertex and displacing the vertex by the appropriate amount. The position is determined by computing where the point falls on the surface in UV coordinate space, then computing the bezier curve value at that point, reconstructing the position with the displacement and then creating the vertex with a reconstructed normal. This produces the curvature effect shown in the images below.

Lastly was the lighting, which is once again the same as previous submissions. It computes the contributions from the ambient, diffuse, and specular lighting. It offers customizable surface color and specular color. You can modify the contribution of each of the components into the final fragment color.

Images