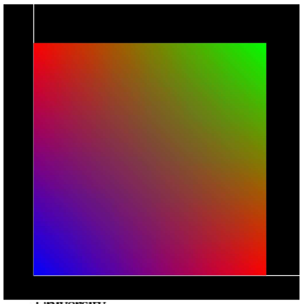
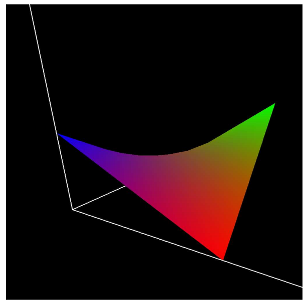
The SuperQuad



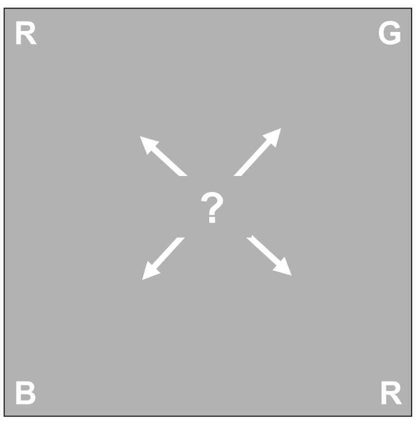
Oregon State University

Mike Bailey

mjb@cs.oregonstate.edu

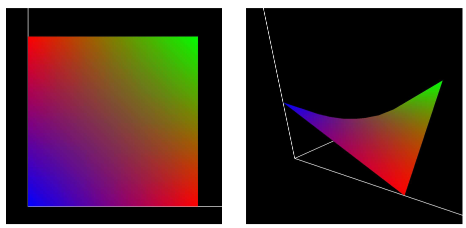


The scientific scenario: a quadrilateral representing continuous data needs to be displayed. Unfortunately, it is non-planar and the data values at the corner vertices map to four widely-varying colors.

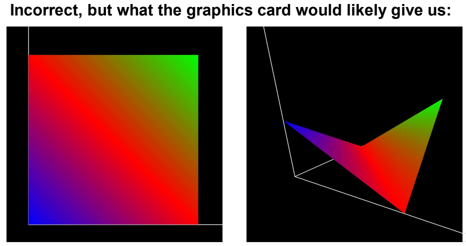


How can we correctly smooth both the internal positions and colors, regardless of how we cut (or the graphics system cuts) the quad into triangles?

Correct:

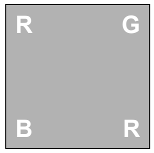


Incorrect, but what the graphics card would likely give us:



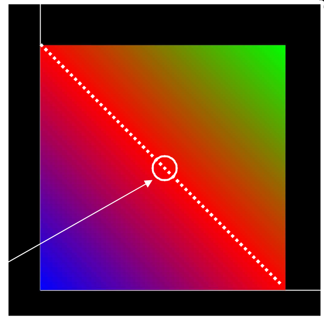
Introducing the *SuperQuad* Geometry Shader!

From a scientific perspective, shouldn't:



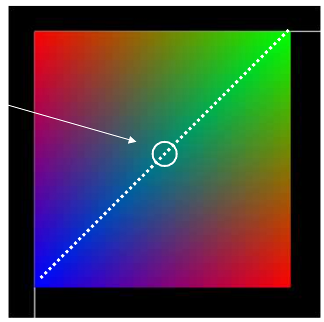
produce the exact same color interpolation regardless of which way the quad is triangularized for display?

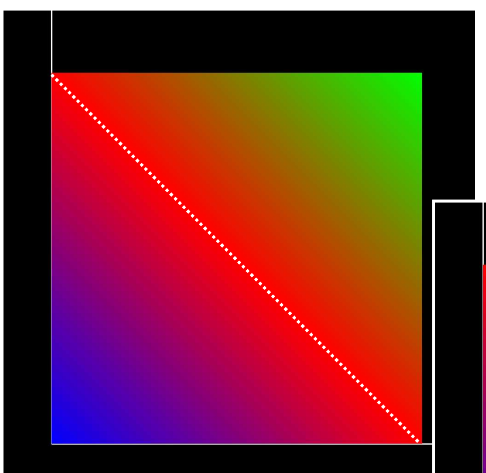
Incorrect



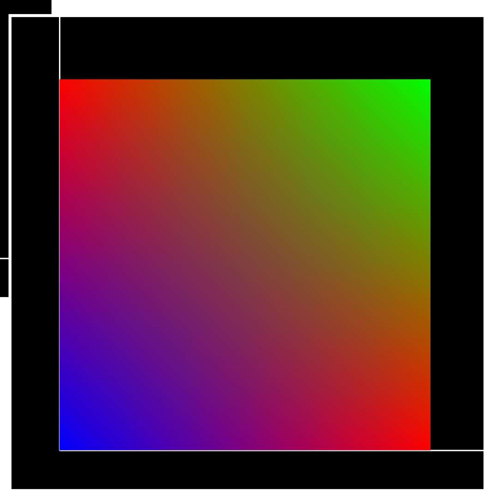
And, shouldn't the color in the middle of the quad be some combination of all 4 corner colors, not just 2 of them?

Incorrect

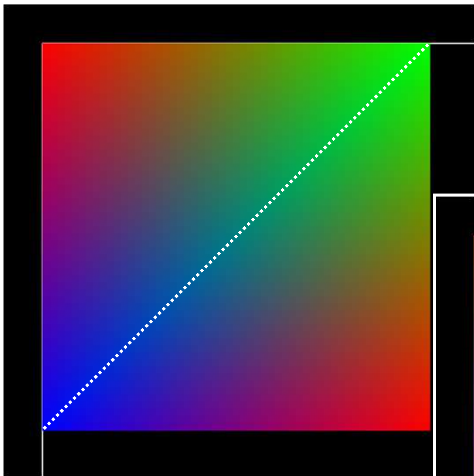




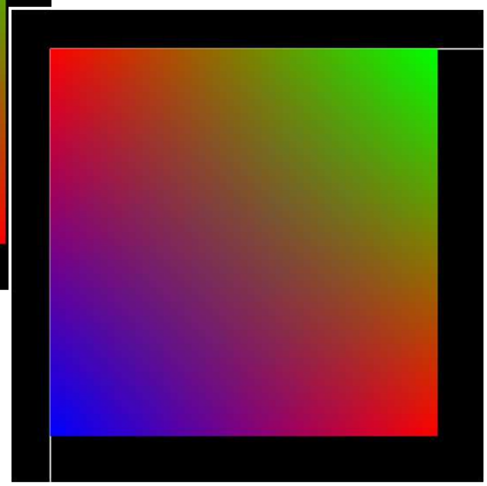
Incorrect



Correct



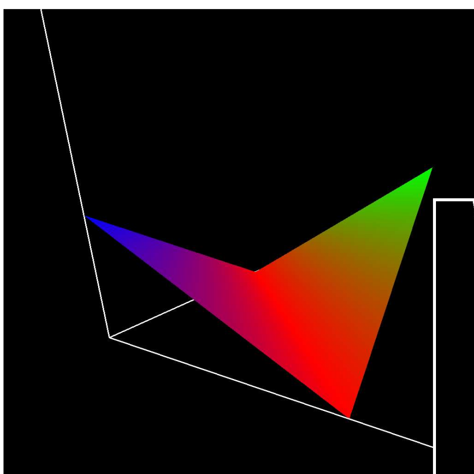
Incorrect



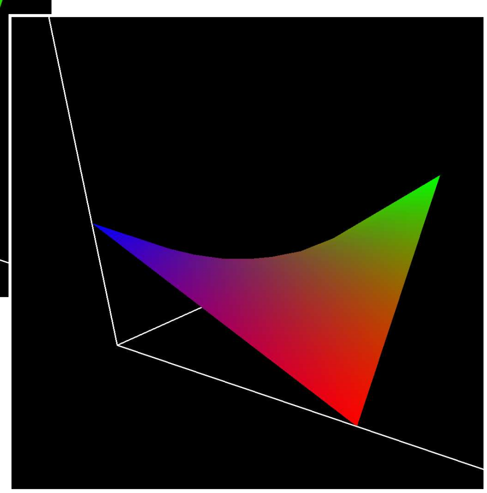
Oregon State University
Computer Graphics

Correct

The same idea applies to how the quad is drawn. The direction of triangularizing shouldn't matter and the center position should be some combination of all 4 corner positions, not just 2 of them.



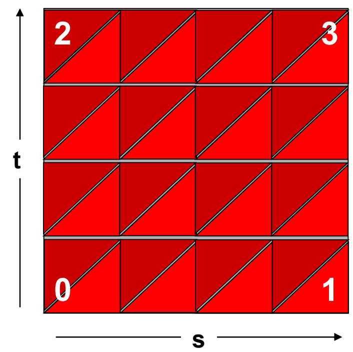
Incorrect



Oregon State University
Computer Graphics

Correct

Solution: Use bilinear interpolation to break the super-quad into sub-triangles



For any quantity, Q, defined at the 4 vertices,
Q can be interpolated into the interior with:

$$Q(s,t) = (1.-s)*(1.-t)*Q_0 + s*(1.-t)*Q_1 + (1.-s)*t*Q_2 + s*t*Q_3;$$

$$0. \le s,t \le 1.$$

superquad.glib

```
##OpenGL GLIB
Perspective 70
LookAt  0 0 3   0 0 0   0 1 0

Vertex      superquad.vert
Geometry superquad.geom
Fragment superquad.frag
Program   SuperQuad  uNum <1 1 6>

Color 1 0 0
LinesAdjacency  [0. 0. 0.5]   [1. 0. 0.]   [0. 1. 0.]   [1. 1. 0.5]
```

superquad.vert

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


superquad.geom, I

```
#version 330 compatibility
#extension GL_EXT_gpu_shader4: enable
#extension GL_EXT_geometry_shader4: enable
layout( lines_adjacency )  in;
layout( triangle_strip, max_vertices=200 )  out;

uniform int uNum;
out vec3    gColor;

const vec3 ColorIn0 = vec3( 1., 0., 0. );
const vec3 ColorIn1 = vec3( 0., 0., 1. );
const vec3 ColorIn2 = vec3( 0., 1., 0. );
const vec3 ColorIn3 = vec3( 1., 0., 0. );

void ProcessPoint( float s, float t )
{
		float oms = 1. - s;
		float omt  = 1. - t;
		gColor =          oms*omt*ColorIn0 +    s*omt*ColorIn1
		                  + oms*t*      ColorIn2    + s*t*      ColorIn3;
		vec4 xyzw =     oms*omt*gl_PositionIn[0]    + s*omt*gl_PositionIn[1]
		                  + oms*t*      gl_PositionIn[2]    + s*t*   gl_PositionIn[3];
		gl_Position = gl_ModelViewProjectionMatrix * xyzw;
		EmitVertex();
}
```

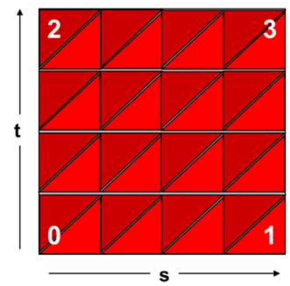
superquad.geom, II

11

```
void
main( )
{
	int nums = uNum;
	int numt = nums;
	float ds = 1. / float(nums);
	float dt = ds;
	float tbot = 0.;
	for( int it = 0; it < numt; it++ )
	{
		float ttop = tbot + dt;

		float s = 0.;
		for( int is = 0; is <= nums; is++ )
		{
			ProcessPoint( s, tbot );
			ProcessPoint( s, ttop );
			s += ds;
		}

		EndPrimitive();
		tbot = ttop;
	}
}
```



superquad.frag

12

```
in vec3 gColor;

void
main( )
{
	gl_FragColor = vec4( gColor, 1. );
}
```

Oregon State University Computer Graphics

mjb – December 4, 2022