The SuperQuad

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?

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

Correct: Incorrect
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.

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

\[ Q(s,t) = (1-s)(1-t)Q_0 + s(1-t)Q_1 + (1-s)tQ_2 + stQ_3; \]

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

```
superquad.glib
```

```
# OpenGL GLIB
Perspective 70
LookAt 0 0 3 0 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]
```
void main()
{
    gl_Position = gl_Vertex;
}

superquad.geom, II

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

in vec3 gColor;
void main()
{
    gl_FragColor = vec4(gColor, 1.);
}