Using Vertex Shaders for Hyperbolic Geometry

Zooming and Panning Around a Complex 2D Display

- Standard (Euclidean) geometry zooming forces much of the information off the screen
- This eliminates the context from the zoomed-in display
- This problem can be solved with hyperbolic methods if we are willing to give up Euclidean geometry
- At one time, this would have also meant severely giving up graphics performance, but not now

Zooming in Euclidean Space

Zooming in Polar Hyperbolic Space
**Polar Hyperbolic Equations**

Overall theme: something divided by something a little bigger

- \( R' = R / (R+K) \)
- \( R = \sqrt{X^2 + Y^2} \)
- \( \Theta = \tan^{-1} \left( \frac{Y}{X} \right) \)
- \( X' = R' \cos \Theta' \)
- \( Y' = R' \sin \Theta' \)

\[ \lim_{K \to 0} R' = 1 \]
\[ \lim_{K \to \infty} R' = 0 \]

Coordinates moved to outer edge when \( K = 0 \)
Coordinates moved to center when \( K = \infty \)

**Cartesian Hyperbolic Equations**

- Polar
  - \( X' = \frac{X}{R+K} \)
  - \( Y' = \frac{Y}{R+K} \)

- Cartesian
  - \( X' = \frac{X}{\sqrt{X^2 + K^2}} \)
  - \( Y' = \frac{Y}{\sqrt{Y^2 + K^2}} \)

Coordinates moved to outer edge when \( K = 0 \)
Coordinates moved to center when \( K = \infty \)

**Zooming in Cartesian Hyperbolic Space**
#version 330 compatibility
uniform bool uPolar;
uniform float uK;
uniform float uTransX;
uniform float uTransY;
out vec3 vColor;

void main()
{
    vColor = gl_Color.rgb;
    vec2 pos = (gl_ModelViewMatrix * gl_Vertex).xy;
    pos += vec2(uTransX, uTransY);
    float r = length(pos.xyz);
    vec4 pos2 = vec4(0., 0., -5., 1.);
    if (uPolar)
    {
        pos2.xy = pos / (r + uK);
    }
    else
    {
        pos2.xy = pos / (pos*pos + uK*uK);
    }
    gl_Position = gl_ProjectionMatrix * pos2;
}

#version 330 compatibility
in vec3 vColor;

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

Data courtesy of the Corvallis Fire Department