Hyperbolic Geometry for Visualization

Mike Bailey
mjb@cs.oregonstate.edu
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
Usual Zooming in Euclidean Space

123,101 line strips
446,585 points
Zooming in Polar Hyperbolic Space
Polar Hyperbolic Equations

Overall theme: something divided by something else a little bigger

Because

\[ R' = \frac{R}{R + K} \]

then:

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

\[ \lim_{K \to \infty} R' = 0 \]
Polar Hyperbolic Equations Don’t Actually Need to use Trig

\[ R = \sqrt{X^2 + Y^2} \]

\[ \Theta = \tan^{-1}\left(\frac{Y}{X}\right) \]

\[ R' = \frac{R}{R + K} \]

\[ X' = R' \cos \Theta = \frac{R}{R + K} \times \frac{X}{R} = \frac{X}{R + K} \]

\[ Y' = R' \sin \Theta = \frac{R}{R + K} \times \frac{Y}{R} = \frac{Y}{R + K} \]
Cartesian Hyperbolic Equations – Treat X and Y Independently

\[
X' = \frac{X}{R + K} \\
Y' = \frac{Y}{R + K}
\]

- Polar

- 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
The Problem with T-Intersections
The Problem with T-Intersections

Your code computes the hyperbolic transformation here and here, and OpenGL draws a straight line between them. But, this point had its hyperbolic transformation computed separately, and doesn’t match up with the straight line.

This kind of situation is called a T-intersection, and crops up all the time in computer graphics, even though we don’t want it to. 😞
A Solution to the T-Intersection Problem

Break this line up into several (many?) sub-pieces, and perform the Hyperbolic Transformation on each intermediate point.

\[ P(t) = (1-t)P_0 + tP_1 \]

\[ t = 0., .01, .02, .03, \ldots \]

This makes that straight line into a curve, as it should be. But, how many line segments should we use?
A More Elegant Approach is to Recursively Subdivide

void
DrawHyperbolicLine( P₀, P₁ )
{
    Compute point \( A = \frac{P₀ + P₁}{2} \).
    Convert point A to Hyperbolic Coordinates, calling it \( A' \).
    Convert \( P₀ \) and \( P₁ \) to Hyperbolic Coordinates \( P₀', P₁' \).
    Compute point \( B' = \frac{P₀' + P₁'}{2} \).
    Compare \( A' \) and \( B' \).
    if( they are “close enough”) {
        Draw the line \( P₀'-P₁' \).
    } else {
        DrawHyperbolicLine( \( P₀ \), \( A \) );
        DrawHyperbolicLine( \( A \), \( P₁ \) );
    }
}

Subdividing to render a curve or surface correctly is a recurring theme in computer graphics.
Hyperbolic Corvallis (Streets, Buildings, Parks)

Kelley Engineering Center