Hyperbolic Geometry for Visualization

Mike Bailey
mjb@cs.oregonstate.edu
Oregon State University

Usual Zooming in Euclidean Space

123,101 line strips
446,585 points

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

Zooming in Polar Hyperbolic Space

Polar Hyperbolic Equations

Overall theme: something divided by something a little bigger

Because $R' = \frac{R}{R+K}$ then:

$\lim_{K \to 0} \frac{R}{R+K} = 1$

$\lim_{K \to \infty} \frac{R}{R+K} = 0$

$X' = R'\cos\Theta' = \frac{R\cos\Theta}{R+K}$

$Y' = R'\sin\Theta' = \frac{R\sin\Theta}{R+K}$

Polar Hyperbolic Equations Don’t Actually Need to use Trig

$R = \sqrt{X^2 + Y^2}$

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

$X' = \frac{R}{R+K} \times X = \frac{X}{R+K}$

$Y' = \frac{R}{R+K} \times Y = \frac{Y}{R+K}$

Coordinates moved to outer edge when $K = 0$

Coordinates moved to center when $R + K$
Cartesian Hyperbolic Equations – Treat X and Y Independently

\[
\begin{aligned}
Polar: & \quad X' = \frac{X}{R + K} \\
& \quad Y' = \frac{Y}{R + K} \\
Cartesian: & \quad X' = \frac{X}{\sqrt{X^2 + K^2}} \\
& \quad Y' = \frac{Y}{\sqrt{Y^2 + K^2}}
\end{aligned}
\]

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

The Problem with T-Intersections

Your code computes the hyperbolic transformation here and here, and OpenGL draws a straight line between them. But, the 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.

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.

\[ P(t) = (1 - t)P_i + tP_j \]

1 = 0., .01, .02, .03, …

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

```cpp
void DrawHyperbolicLine( P0, P1 )
{
    Compute point A to Hyperbolic Coordinates, calling it A'
    Convert P0 and P1 to Hyperbolic Coordinates P0', P1'
    Compute point B' = (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 correctly is a recurring theme in computer graphics.