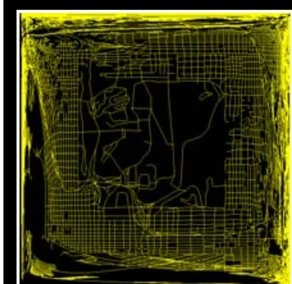
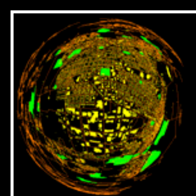
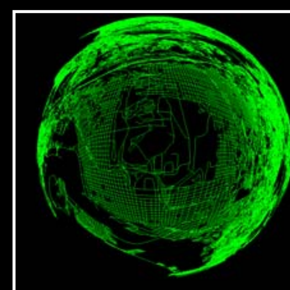
Using Vertex Shaders for Hyperbolic Geometry

Mike Bailey

Oregon State University

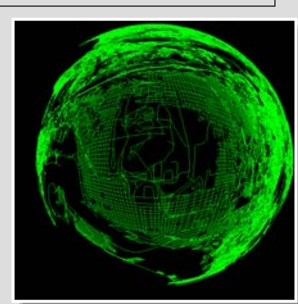






Zooming and Panning Around a Complex 2D Display

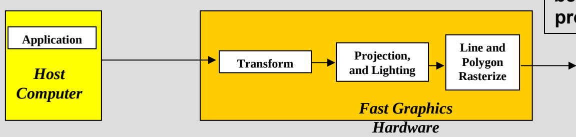
- Standard (Euclidean) geometry zooming forces much of the information off the screen
- This eliminates the context from the zoomed-in display
- This problem can be solved with *hyperbolic methods* if we are willing to give up Euclidean geometry
- At one time, this would have also meant severely giving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



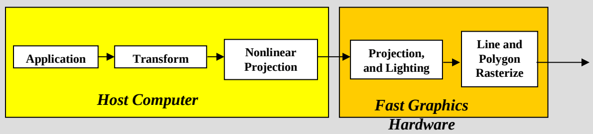
The Problem with Nonlinear Projections in Stock OpenGL

Transforming the coordinates to center the ROI must come *before* the non-linear projection equations

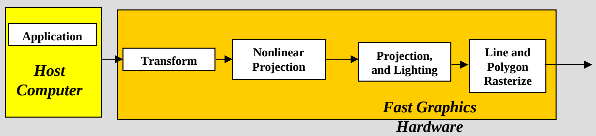
Graphics application with linear projections:



Graphics application with non-linear projections:



GPU-programmable interactive graphics application with non-linear projections:

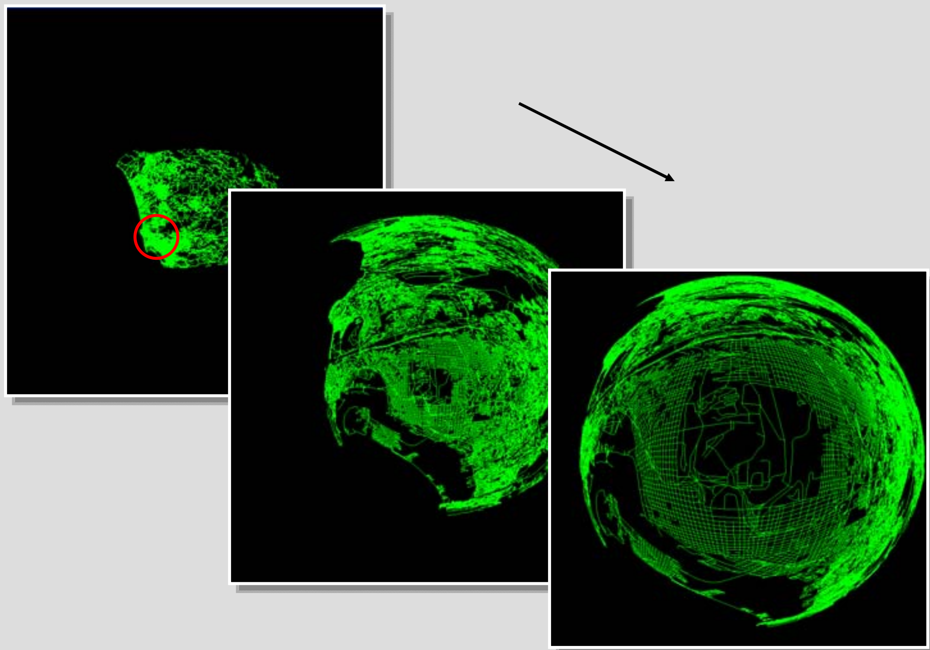


mjb – December 3, 2007

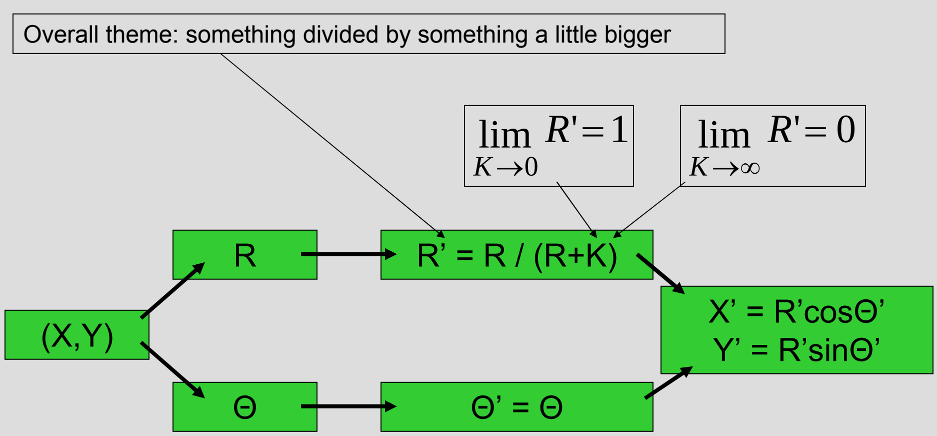
Zooming in Euclidean Space

123,101 line strips
446,585 points

Zooming in Polar Hyperbolic Space



Polar Hyperbolic Equations



mjb – December 3, 2007

Polar Hyperbolic Equations

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

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

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

Coordinates moved to outer edge when K = 0

Coordinates moved to center when K = ∞

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

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

Cartesian Hyperbolic Equations

Polar

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

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

Cartesian

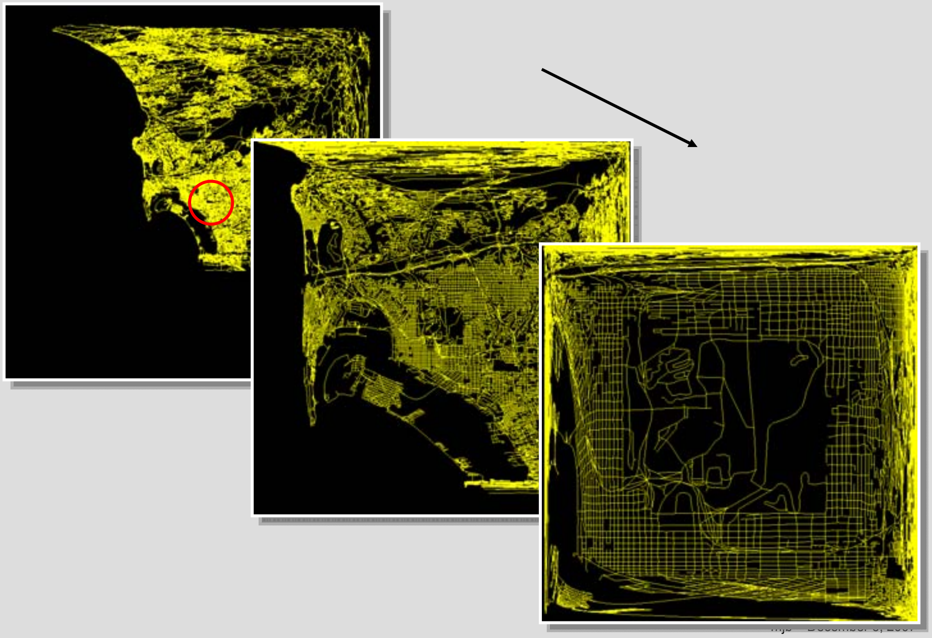
$$X' = \frac{X}{\sqrt{X^2 + K^2}}$$

$$Y' = \frac{Y}{\sqrt{Y^2 + K^2}}$$

Coordinates moved to outer edge when K = 0

Coordinates moved to center when K = ∞

Zooming in Cartesian Hyperbolic Space

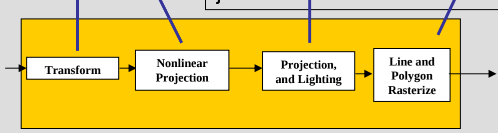


hyper.vert

```
uniform float       Polar;
uniform float       K;
uniform float       TransX;
uniform float       TransY;
varying vec4        Color;

void
main( void )
{
	Color = gl_Color;

	vec2 pos = ( gl_ModelViewMatrix * gl_Vertex ).xy;
	pos += vec2( TransX, TransY );
	float r = length( pos.xyz );

	vec4 pos2 = vec4( 0., 0., -5., 1. );

	if( Polar != 0. )
		pos2.xy = pos / ( r + K );
	else
		pos2.xy = pos * inversesqrt(  pos*pos + K*K );

	gl_Position = gl_ProjectionMatrix * pos2;
}
```

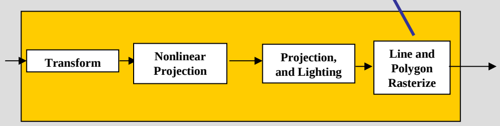
Note: try to avoid “if” statements whenever possible

Note: “swizzling”

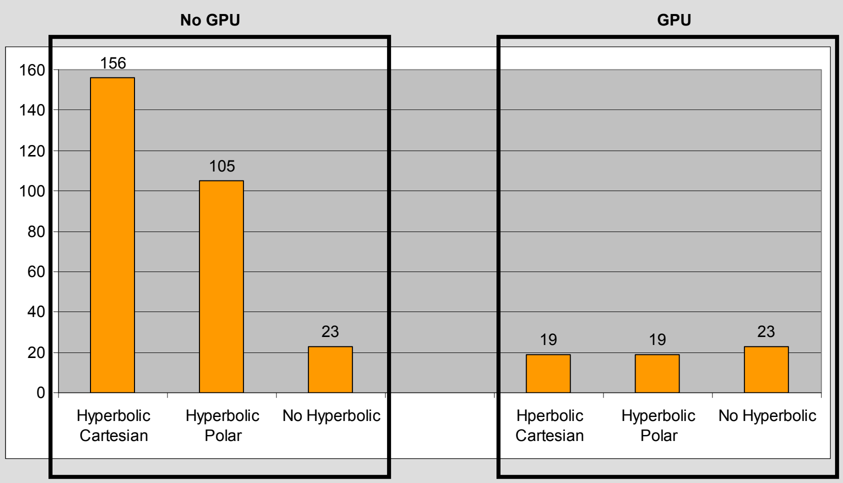


hyper.frag

```
varying vec4  Color;

void
main( void )
{
        gl_FragColor = Color;
}
```

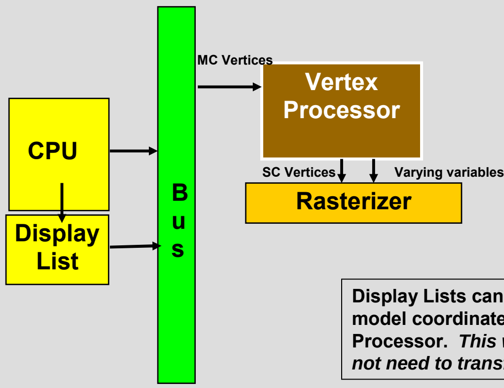


Display Timing (msec/update)

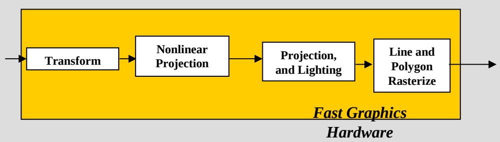


123,101 line strips
446,585 points

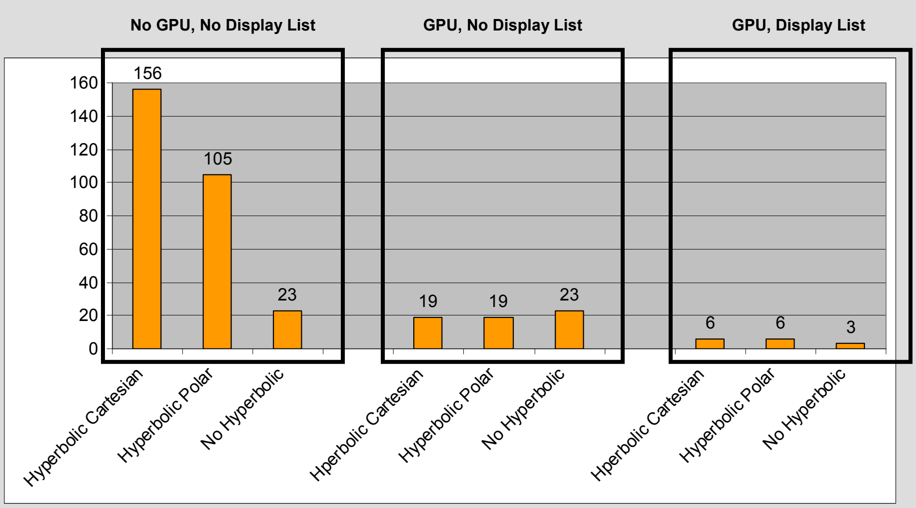
What if Display Lists are used?



Display Lists can be used to quickly stream model coordinate vertices into the Vertex Processor. ***This will only work if the CPU does not need to transform the vertices first.***



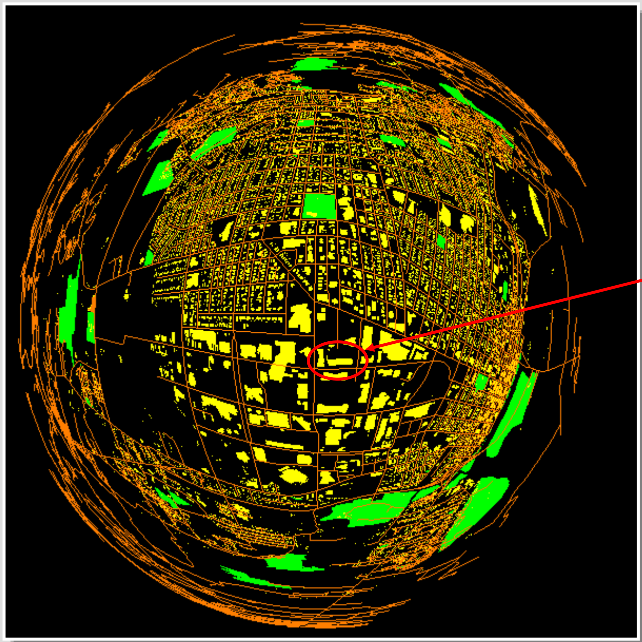
Display Timing (msec/update)



123,101 line strips
446,585 points

Note: can never have a category of "No GPU, Display List"

Corvallis Streets, Buildings, Parks



Kelley Engineering Center

mjb – December 3, 2007

Can also use ModelView Scaling in Place of K

$$R' = \frac{sR}{sR + K} = \frac{R}{R + \frac{K}{s}}$$

Therefore, increasing *K* and decreasing *s* accomplish the same thing.

Also, ModelView translation can be used in place of explicit translation factors

mjb – December 3, 2007