A Parallel Algorithm for Polygon Rasterization

Zhongchuan Zhang
10/29/2009
Introduction

• Fast rendering of 3D Z-buffered linearly interpolated polygons
  – 3D transformation, projection and light calculation of the vertices
  – **Rasterization of the polygon into a frame buffer**

• This paper deals with one aspect of the latter problem: the computation of the boundaries of a polygon
Introduction

• Ways to compute the edges of polygons
  – Compute edges by a line interpolation algorithm
    • Not so convenient for frame buffers
  – Use a parallel multiplier tree to simultaneously compute a linear edge function
    • Highly parallel
    • Require dedicated logic for each pixel
  – Use a linear function to define polygon edges and can use conventional DRAM and VRAM technology
The edge function

• Definition
  \[ E(x,y) = (x-X)dY - (y-Y)dX \]
  \[ E(x,y) > 0, \text{ if } (x,y) \text{ is to the “right” side} \]
  \[ E(x,y) > 0, \text{ if } (x,y) \text{ is exactly on the line} \]
  \[ E(x,y) > 0, \text{ if } (x,y) \text{ is to the “left” side} \]

• Linear, can be computed incrementally in the same way as color and Z values

• Convenient for rasterization
  \[ E(x+1,y) = E(x,y) + dY \]
  \[ E(x,y+1) = E(x,y) - dX \]
The edge function

- It can be computed parallel for all pixels in the frame buffer
- Define polygons using boolean combinations of edges
- Used as a "stencil" that allows a pixel to be modified only if it is interior to the polygon

Triangle formed by union of right sides of AB, BC and CA
Incremental classification of points around a convex polygon

- Vertices \((X_i, Y_i)\) \(0<i<=N\), \((X_0,Y_0)=(X_N,Y_N)\)
- Initial edge function at starting point \((X_s,Y_s)\)
  \[
  dX_i = X_i - X[i-1] \\
  dY_i = Y_i - Y[i-1] \\
  E_i(X_s,Y_s) = (X_s-X_i)dY_i - (Y_s-Y_i)dX_i
  \]
- Make increment
  \[
  E_i(x+1,y) = E_i(x,y) + dY_i \\
  E_i(x-1,y) = E_i(x,y) - dY_i \\
  E_i(x,y+1) = E_i(x,y) - dX_i \\
  E_i(x,y-1) = E_i(x,y) + dX_i
  \]
- Interior to the polygon
  \(E_i>=0\) for all \(i : 0<i<N\) (using a tie break rule)
Traversing the polygon

Traversing the bounding box

A more efficient traversal algorithm

Traversal algorithm may have to search for edge

Proceeding outward from center line
Clipping

• View left and right clipping as additional polygon edges
• Use top clip boundary to control the starting point
• Use bottom clip boundary to control the last scan line
Sub-pixel accuracy of vertices

- Vertices are in floating point format after 3D transformation and projection
- Rounding the X and Y floating point ordinates can leave gaps
- Perform the interpolator setup computation in float point and convert to fixed point at the end

\[
\begin{align*}
\text{dXi} &= X_i - X_{[i-1]} \\
\text{dYi} &= Y_i - Y_{[i-1]} \\
E_i(X_s,Y_s) &= (X_s - X_i)\text{dYi} - (Y_s - Y_i)\text{dXi} \\
\text{dXi}' &= \text{FIX}(\text{dXi}) \\
\text{dYi}' &= \text{FIX}(\text{dYi}) \\
E_i' &= \text{FIX}(E_i)
\end{align*}
\]
Parallel implementation

• The edge function is linear, so
  \[ E(x+L,y) = E(x) + L \Delta y \]
  L-distance from a given point \((x,y)\)

• Allows a group of interpolators to simultaneously compute the edge function of an adjacent block (L pixels wide) in a single cycle
Future work and extensions

• Compute higher order edge function and compute complex shapes
• Perform the interpolation of the edge function in a float point like manner
• Use the edge function to anti-alias edges
• Have the lookup table produce a crude sub-pixel resolution bitmap for each edge
Conclusion

• Can be computed in parallel and used with common refresh buffer word organization
• Can be computed with hardware similar to that required to interpolate color and Z values for 3D solids
• Maintains the sub-pixel accuracy of vertices
Thank you!