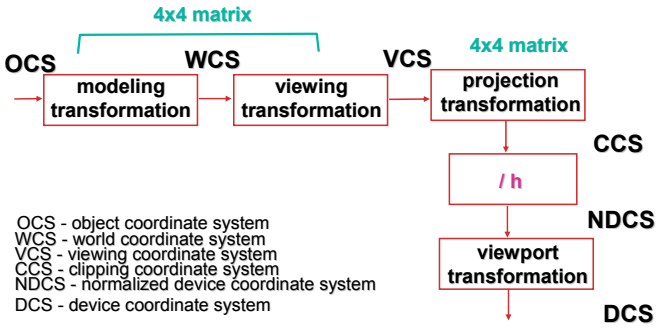
Projective Rendering Pipeline



Lines and Curves

Explicit

line $y = mx + b$

$$y = \frac{(y_2 - y_1)}{(x_2 - x_1)}(x - x_1) + y_1$$

circle $y = \pm\sqrt{r^2 - x^2}$

plane $z = Ex + Fy + G$

sphere $z = \pm\sqrt{r^2 - x^2 - y^2}$

Lines and Curves

Parametric

line

$$x(t) = x_0 + t(x_1 - x_0)$$
$$y(t) = y_0 + t(y_1 - y_0)$$
$$t \in [0,1]$$
$$P(t) = P_0 + t(P_1 - P_0)$$
$$P(t) = (1-t)P_0 + tP_1$$

not unique! e.g. could replace t by $5t$

circle

$$x(\theta) = r\cos(\theta)$$
$$y(\theta) = r\sin(\theta)$$
$$\theta \in [0, 2\pi]$$

plane $P(s,t) = P_0 + s(P_1 - P_0) + t(P_2 - P_0)$

Lines and Curves

Implicit

line $F(x,y) = (x - x_0)dy - (y - y_0)dx$ e.g. not unique: could replace $F(x,y)$ by $k \cdot F(x,y)$

$F(x,y) = 0$ **(x,y) is on line**

$F(x,y) > 0$ **(x,y) is below line**

$F(x,y) < 0$ **(x,y) is above line**

circle $F(x,y) = x^2 + y^2 - r^2$

$F(x,y) = 0$ **(x,y) is on circle**

$F(x,y) > 0$ **(x,y) is outside**

$F(x,y) < 0$ **(x,y) is inside**

plane $F(x,y,z) = Ax + By + Cz + D$
$= \langle A,B,C \rangle \cdot P + D$
$= \vec{N} \cdot P + D$
normal to plane

Polygons

Interactive graphics uses Polygons

- Can represent any surface *with arbitrary accuracy*
  - *Splines, mathematical functions, ...*
- simple, regular rendering algorithms
  - *embed well in hardware*

Polygons

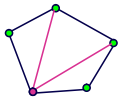
Basic Types

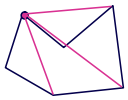
simple convex | **simple concave** | **non-simple (self-intersection)**

From Polygons to Triangles

University of British Columbia

- why? triangles are planar and convex
- simple convex polygons
  - *break into triangles, trivial*
  - `glBegin(GL_POLYGON) ... glEnd()`
- concave or non-simple polygons
  - *break into triangles, more effort*
  - `gluNewTess(), gluTessCallback(), ...`

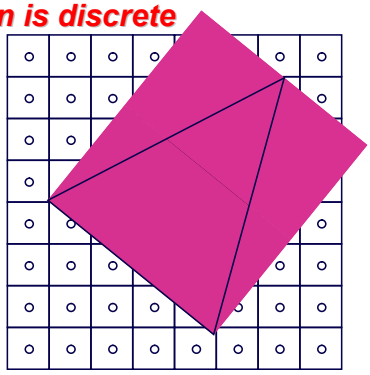




What is Scan Conversion? (a.k.a. Rasterization)

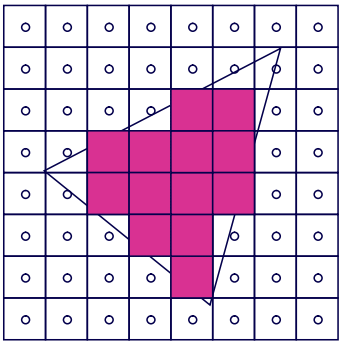
University of British Columbia

screen is discrete



University of British Columbia

one possible scan conversion

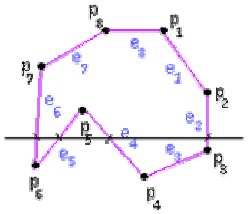


Scan Conversion

University of British Columbia

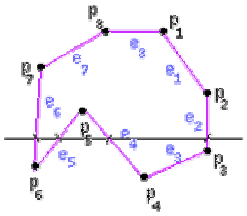
A General Algorithm

- intersect each scanline with all edges
- sort intersections in x
- calculate parity to determine in/out
- fill the 'in' pixels



University of British Columbia

- works for arbitrary polygons
- efficiency improvement:
  - *exploit row-to-row coherence using "edge table"*



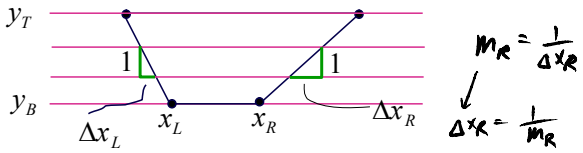
Edge Walking

University of British Columbia

past graphics hardware

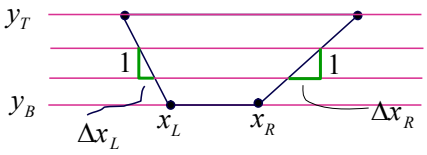
- exploit continuous L and R edges on trapezoid

scanTrapezoid($x_L, x_R, y_B, y_T, \Delta x_L, \Delta x_R$)



University of British Columbia

```
for (y=yB; y<=yT; y++) {
  for (x=xL; x<=xR; x++)
    setPixel(x,y);
  xL += DxL;
  xR += DxR;
}
```



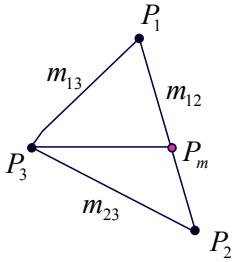
Edge Walking Triangles

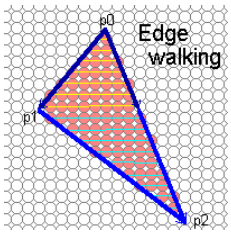
University of British Columbia

- split triangles into two regions with continuous left and right edges

scanTrapezoid($x_3, x_m, y_3, y_1, \frac{1}{m_{13}}, \frac{1}{m_{12}}$)

scanTrapezoid($x_2, x_2, y_2, y_3, \frac{1}{m_{23}}, \frac{1}{m_{12}}$)



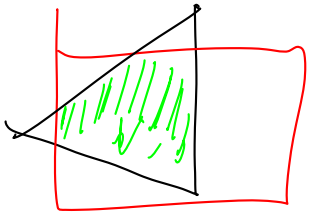


Edge Walking Triangles

University of British Columbia

Issues

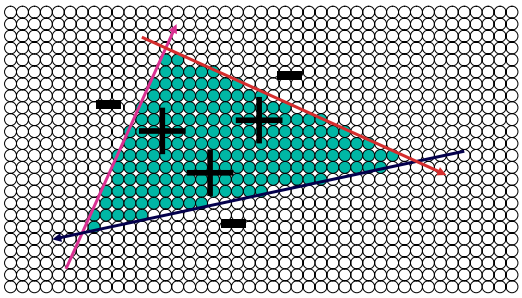
- many applications have small triangles
  - *setup cost is non-trivial*
- clipping triangles produces non-triangles



Modern Rasterization

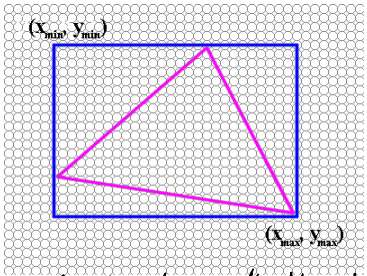
University of British Columbia

Define a triangle as follows:



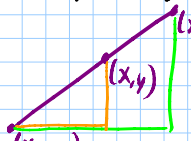
Using Edge Equations

University of British Columbia



→ clip to window by clipping the bbox to the window

Implicit Equation for a 2D line



general form: $Ax + By + C = 0$

$$\frac{y_1 - y_0}{x_1 - x_0} = \frac{y - y_0}{x - x_0}$$

$$(y_1 - y_0)(x - x_0) = (y - y_0)(x_1 - x_0)$$

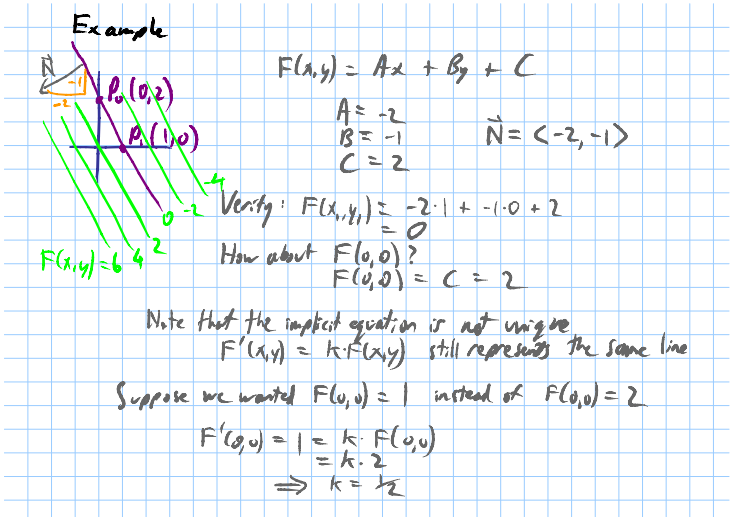
$$x(y_1 - y_0) - x_0(y_1 - y_0) + y(x_0 - x_1) - y_0(x_0 - x_1) = 0$$

$$x(y_1 - y_0) + y(x_0 - x_1) - x_0 y_1 + x_0 y_0 - x_0 y_0 + x_1 y_0 = 0$$

$$\Rightarrow F(x,y) = Ax + By + C$$

$A = y_1 - y_0$
$B = x_0 - x_1$
$C = x_1 y_0 - x_0 y_1$

Note: can think of $\langle A, B \rangle$ as a normal $\vec{N}$ to the line



Edge Equations

- So…we can find edge equation from two verts.
- Given $\mathbf{P}_0$, $\mathbf{P}_1$, $\mathbf{P}_2$, what are our three edges?

  How do we make sure the half-spaces defined by the edge equations all share the same sign on the interior of the triangle?
- A: Be consistent (Ex: $[\mathbf{P}_0\,\mathbf{P}_1]$, $[\mathbf{P}_1\,\mathbf{P}_2]$, $[\mathbf{P}_2\,\mathbf{P}_0]$)

  How do we make sure that sign is positive?
- ~~A: Test, and flip if needed ($A = -A$, $B = -B$, $C = -C$)~~

  Consistently walking clockwise around the Δ and using $F(x,y)$ as developed previously will result in the Δ interior being positive for all 3 line equations

Edge Equations: Code

Basic structure of code:

- Setup: compute edge equations, bounding box
- (Outer loop) For each scanline in bounding box...
- (Inner loop) …check each pixel on scanline, evaluating edge equations and drawing the pixel if all three are positive

Edge Equations: Code


```
findBoundingBox(&xmin, &xmax, &ymin, &ymax);
setupEdges (&a0,&b0,&c0,&a1,&b1,&c1,&a2,&b2,&c2);

for (int y = yMin; y <= yMax; y++) {
      for (int x = xMin; x <= xMax; x++) {
            float e0 = a0*x + b0*y + c0;
            float e1 = a1*x + b1*y + c1;
            float e2 = a2*x + b2*y + c2;
            if (e0 > 0 && e1 > 0 && e2 > 0)
                  Image[x][y] = TriangleColor;
      }
}
```

Edge Equations: Code


```
  // more efficient inner loop
for (int y = yMin; y <= yMax; y++) {
      float e0 = a0*xMin + b0*y + c0;
      float e1 = a1*xMin + b1*y + c1;
      float e2 = a2*xMin + b2*y + c2;
      for (int x = xMin; x <= xMax; x++) {
            if (e0 > 0 && e1 > 0 && e2 > 0)
                  Image[x][y] = TriangleColor;
            e0 += a0;    e1+= a1;     e2 += a2;
      }
}
```

Can pull much of this out of loop for further efficiency

Triangle Rasterization Issues

Exactly which pixels should be lit?

A: Those pixels inside the triangle edges

What about pixels exactly on the edge?

- Draw them: order of triangles matters (it shouldn't)
- Don't draw them: gaps possible between triangles

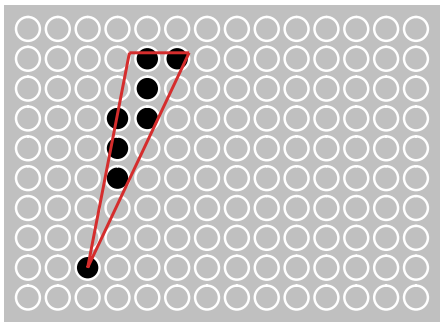
We need a consistent (if arbitrary) rule

- Example: draw pixels on left or top edge, but not on right or bottom edge

Triangle Rasterization Issues

University of British Columbia

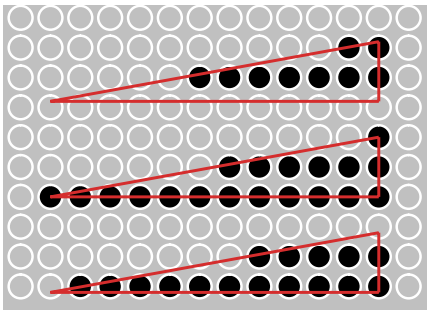
Sliver



Triangle Rasterization Issues

University of British Columbia

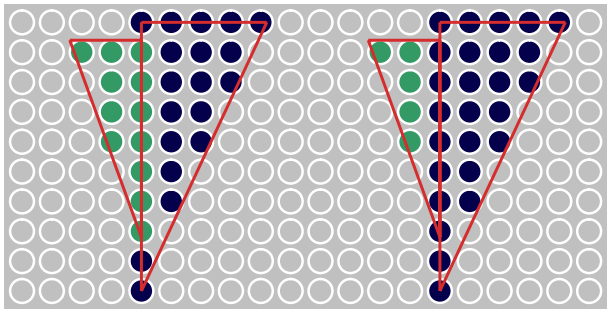
Moving Slivers



Triangle Rasterization Issues

University of British Columbia

Shared Edge Ordering



Interpolation During Scan Conversion

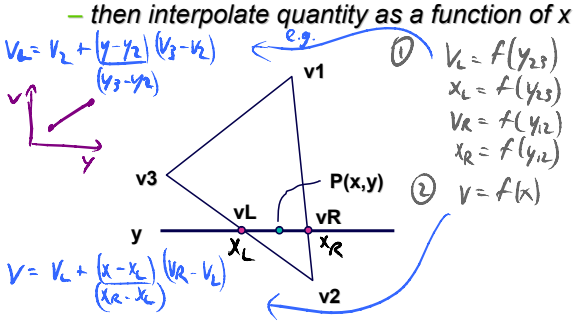
University of British Columbia

- interpolate between vertices: (demo)
  - *z*
  - *r,g,b colour components*
  - *u,v texture coordinates*
  - N_x, N_y, N_z *surface normals*
- three equivalent methods (for triangles)
  1. *bilinear interpolation*
  2. *plane equation*
  3. *barycentric coordinates*

1. Bilinear Interpolation

University of British Columbia

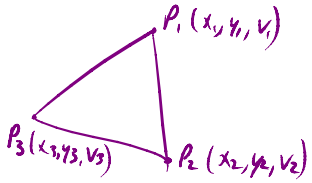
- interpolate quantity along LH and RH edges, as a function of y
  - *then interpolate quantity as a function of x*



2. Plane Equation

University of British Columbia

- v = Ax + By + C



Plane eqn: $Ax + By + Cz + D = 0$

$\Rightarrow Ax + By + Cv + D = 0$

$v = -\frac{A}{C}x - \frac{B}{C}y - \frac{D}{C}$

Computing A, B, C:

$\vec{N} = \langle A, B, C \rangle = (P_2 - P_3) \times (P_1 - P_3)$

$N \cdot P + D = 0$

$D = -N \cdot P$

choose any P:

$D = -N \cdot P_1$

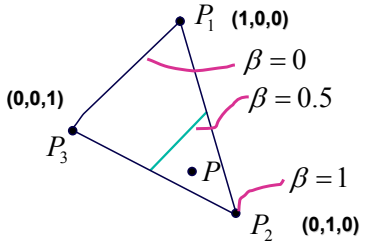
3. Barycentric Coordinates

University of British Columbia

- ***weighted combination of vertices***

$$\begin{cases} P = \alpha \cdot P_1 + \beta \cdot P_2 + \gamma \cdot P_3 \\ \alpha + \beta + \gamma = 1 \\ 0 \le \alpha, \beta, \gamma \le 1 \end{cases}$$

"convex combination of points"



Barycentric Coordinates

University of British Columbia

- once computed, use to interpolate any # of parameters from their vertex values

$$z = \alpha \cdot z_1 + \beta \cdot z_2 + \gamma \cdot z_3$$
$$r = \alpha \cdot r_1 + \beta \cdot r_2 + \gamma \cdot r_3$$
$$g = \alpha \cdot g_1 + \beta \cdot g_2 + \gamma \cdot g_3$$

etc.