Scan Conversion

Projective Rendering Pipeline

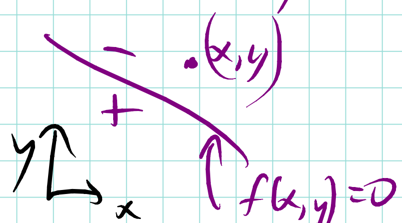
- OCS → **modeling transformation** → WCS → **viewing transformation** → VCS → **projection transformation** → CCS → **/ h** → NDCS → **viewport transformation** → DCS
- 4x4 matrix (modeling and viewing transformations)
- 4x4 matrix (projection transformation)

OCS - object coordinate system
WCS - world coordinate system
VCS - viewing coordinate system
CCS - clipping coordinate system
NDCS - normalized device coordinate system
DCS - device coordinate system

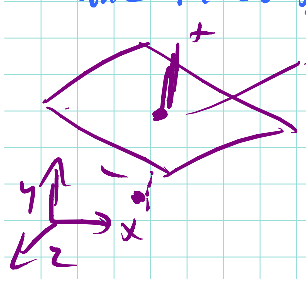
① Implicit

Line in 2D space

$0 = mx + b - y$



Plane in 3D space



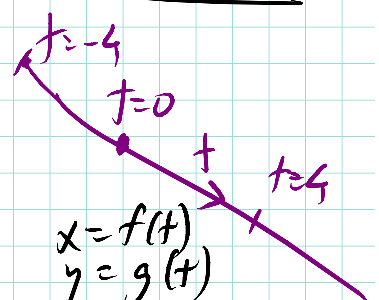
$f(x,y,z)=0$ on the plane

② Explicit

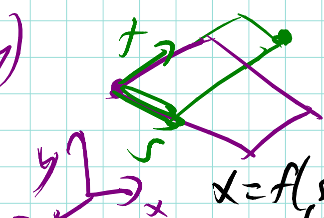
$y = f(x)$
$= mx + b$

$z = f(x,y)$

③ Parametric



$x = f(t)$
$y = g(t)$



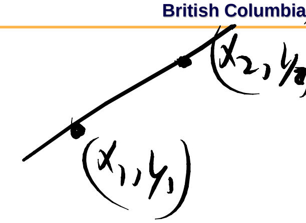
$x = f(s,t)$
$y = g(s,t)$
$z = h(s,t)$

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

comes from $x^2 + y^2 + z^2 = r^2$

Lines and Curves

University of British Columbia

Parametric

line

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

Same

"basis functions"

P_0 (x_0, y_0) $t=0$

P_1 (x_1, y_1) $t=1$

$p(t)$ $\begin{bmatrix} x(t) \\ y(t) \end{bmatrix}$

circle

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

r θ

plane $P(s,t) = P_0 + s(P_1 - P_0) + t(P_2 - P_0)$
where P_0, P_1, P_2 are pts on plane

Lines and Curves

University of British Columbia

Implicit

$F(x,y) = ax + by + c$

line $\cancel{F(x,y) = (x - x_0)dy - (y - y_0)dx}$

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

$F(x,y) > 0$ **(x,y) is ~~below line~~** is on one side

$F(x,y) < 0$ **(x,y) is ~~above line~~** is on the other

dx dy (x_0, y_0)

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

$x^2 + y^2 = r^2$

$0 = x^2 + y^2 - r^2$

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

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

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

plane: $\boxed{F(x,y,z) = Ax + By + Cz + D = N \cdot P + D}$

© Michiel van de Panne

Polygons (really triangles!)

University of British Columbia

Interactive graphics uses Polygons

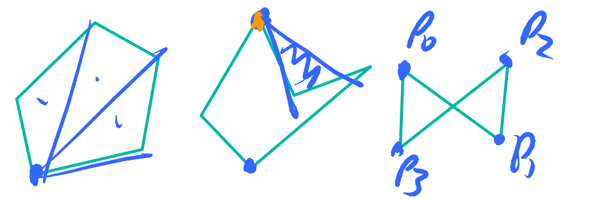
- Can represent any surface *with arbitrary accuracy*
  - *Splines, mathematical functions, ...*
- simple, regular rendering algorithms
  - *embed well in hardware*

© Michiel van de Panne

Polygons

University of British Columbia

Basic Types



simple convex

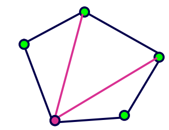
simple concave

non-simple (self-intersection)

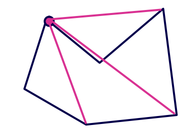
From Polygons to Triangles

University of British Columbia

- why? triangles are planar and convex



- simple convex polygons
  - *break into triangles, trivial*
  - `glBegin(GL_POLYGON) ... glEnd()`

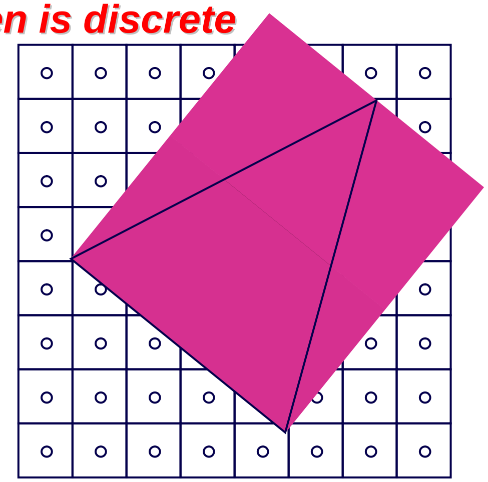


- concave or non-simple polygons
  - *break into triangles, more effort*
  - `gluNewTess(), gluTessCallback(), ...`

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

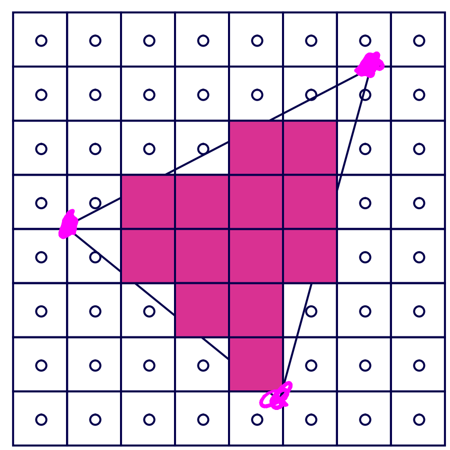
University of British Columbia

screen is discrete



University of British Columbia

one possible scan conversion



Scan Conversion

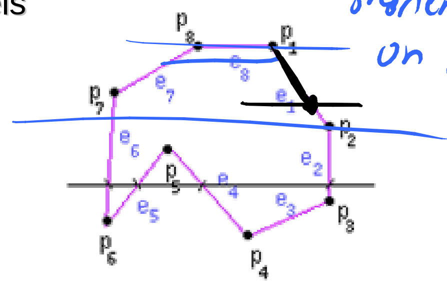
University of British Columbia

A General Algorithm

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

- ignore horizontal edges
- shorten each edge on one side

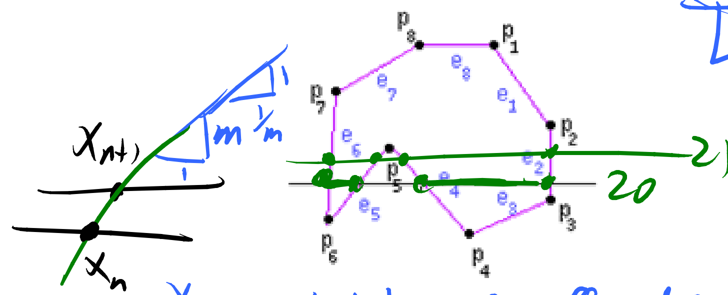
1, 2, 3, 4, 5, 6



University of British Columbia

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

for bookkeeping of when edges start/stop



use $x_{n+1} = x_n + 1/m$ for efficient intersection update

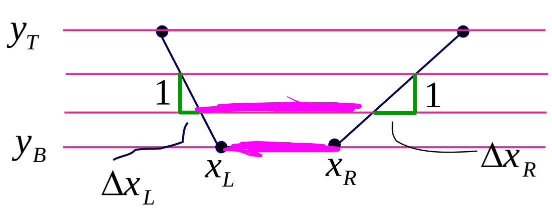
Edge Walking

University of British Columbia

past graphics hardware

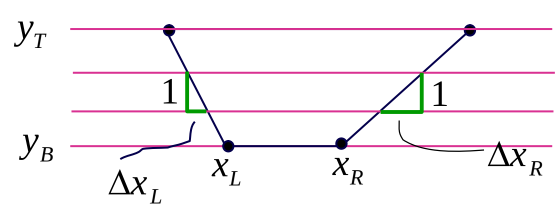
- exploit continuous L and R edges on trapezoid

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



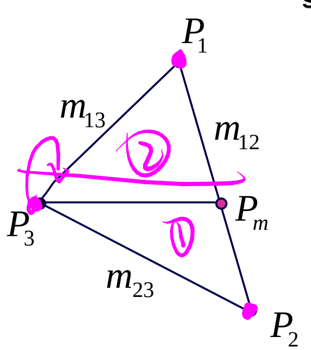
University of British Columbia

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



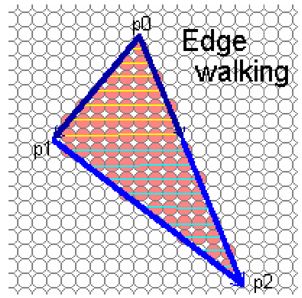
Edge Walking Triangles

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



scanTrapezoid($x_3, x_m, y_3, y_1, \frac{1}{m_{13}}, \frac{1}{m_{12}}$ **)** 2

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



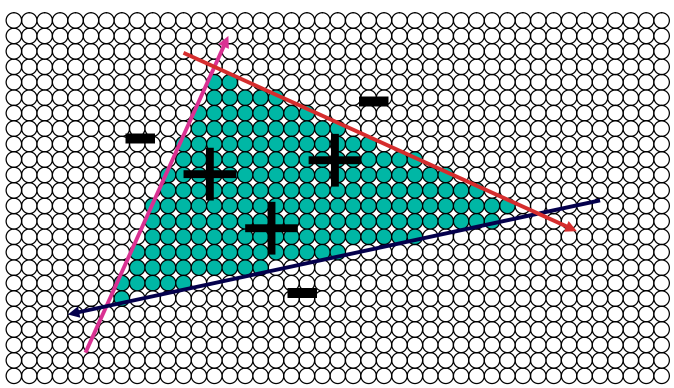
Edge Walking Triangles

Issues

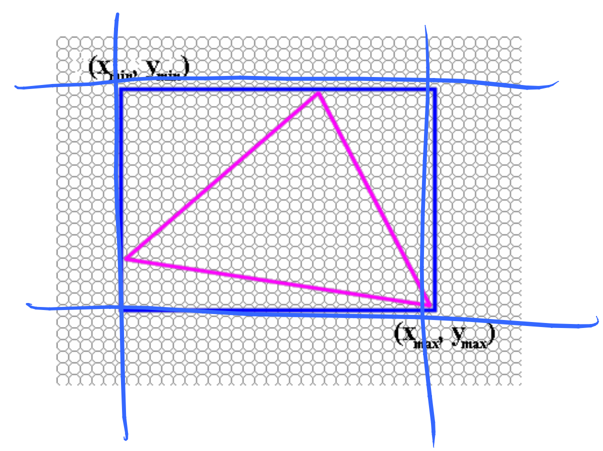
- many applications have small triangles
  - *setup cost is non-trivial*
- clipping triangles produces non-triangles

Modern Rasterization

Define a triangle as follows:



Using Edge Equations



Computing Edge Equations

University of British Columbia

Computing A,B,C from $(x_1, y_1), (x_2, y_2)$

$Ax_1 + By_1 + C = 0$

$Ax_2 + By_2 + C = 0$

$Ax_3 + By_3 + C = 1$

- ~~two~~ three equations, three unknowns
- ~~solve for A & B in terms of C~~

Computing Edge Equations

University of British Columbia

$$\begin{bmatrix} x_0 & y_0 \\ x_1 & y_1 \end{bmatrix} \begin{bmatrix} A \\ B \end{bmatrix} = -C \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} A \\ B \end{bmatrix} = \frac{-C}{x_0 y_1 - x_1 y_0} \begin{bmatrix} y_1 - y_0 \\ x_1 - x_0 \end{bmatrix}$$

- choose $C = x_0 y_1 - x_1 y_0$ for convenience
- Then $A = y_0 - y_1$ and $B = x_1 - x_0$

Edge Equations

University of British Columbia

- 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: $[\mathbf{P_0 P_1}], [\mathbf{P_1 P_2}], [\mathbf{P_2 P_0}]$)

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

Edge Equations: Code

University of British Columbia

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

University of British Columbia

```
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

University of British Columbia

```
  // 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;
      }
}
```


Triangle Rasterization Issues

University of British Columbia

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

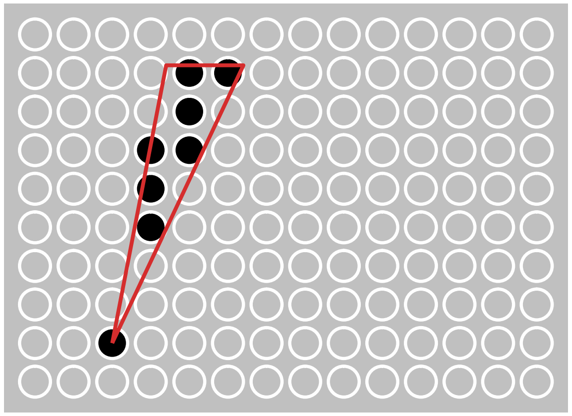
We need a consistent (if arbitrary) rule

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

Triangle Rasterization Issues

University of British Columbia

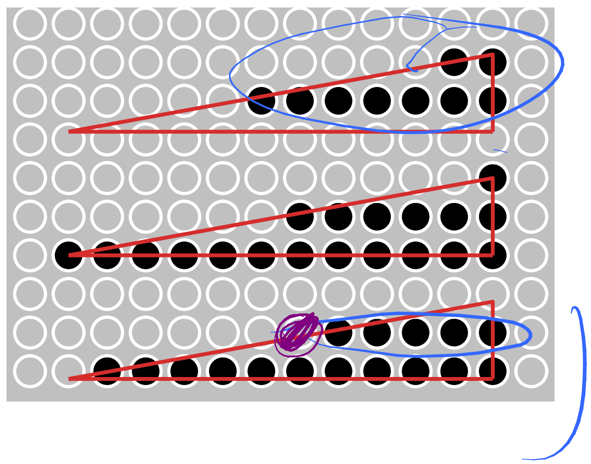
Sliver



Triangle Rasterization Issues

University of British Columbia

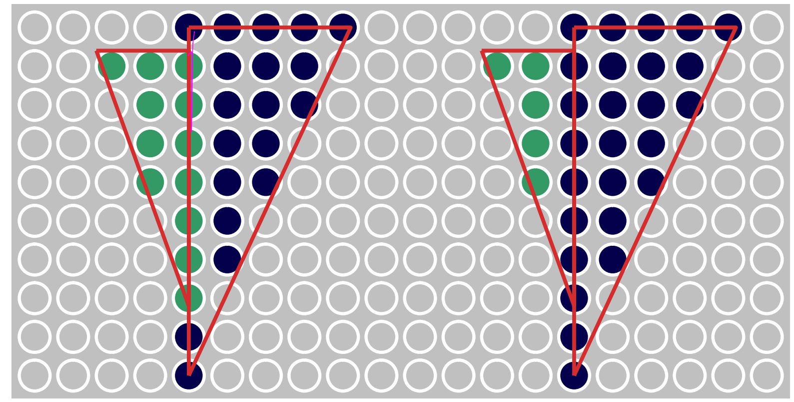
Moving Slivers



Triangle Rasterization Issues

University of British Columbia

Shared Edge Ordering



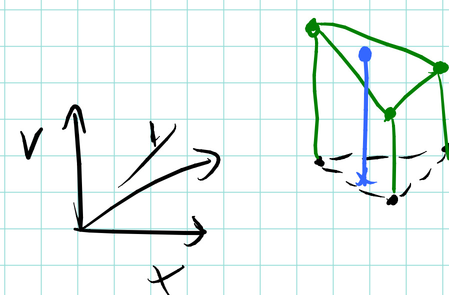
Interpolation During Scan Conversion

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*

"v"

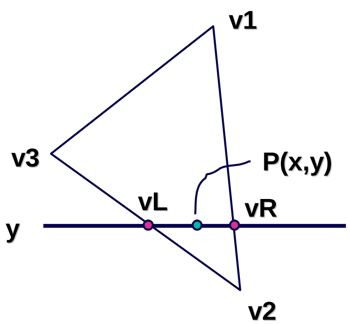
- three equivalent ways of viewing this (for triangles)
  1. *bilinear interpolation*
  2. *plane equation*
  3. *barycentric coordinates*



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

- $v = A'x + B'y + C'$

Plane equation (3D): $Ax + By + Cv + D = 0$

$\Rightarrow$ solve for v:

$$v = \frac{-Ax - By - D}{C}$$

Building Plane Equations

$Ax + By + Cz + D = 0$

$\langle A, B, C\rangle \cdot \langle x, y, z\rangle + D = 0$

$\vec{N} \cdot \vec{P} + D = 0$

(1) compute A, B, C by computing a normal

e.g. $\vec{N} = (P_3 - P_1) \times (P_2 - P_1)$

(2) compute $D = -N \cdot P$ for any point P on triangle

eg. $D = -N \cdot P_1$

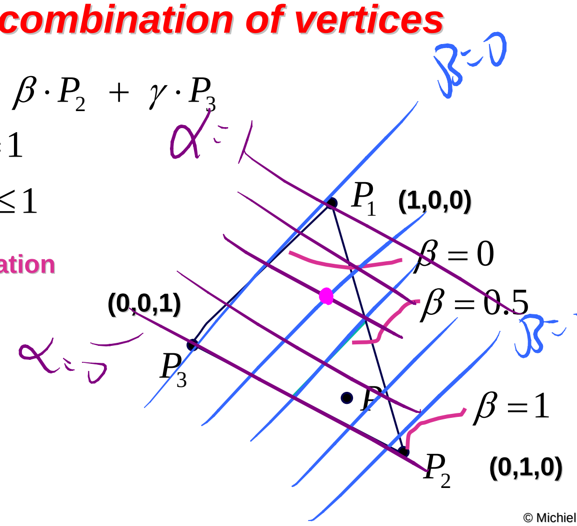
3. Barycentric Coordinates

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.

Computing Barycentric Coords

University of British Columbia

© Michiel van de Panne