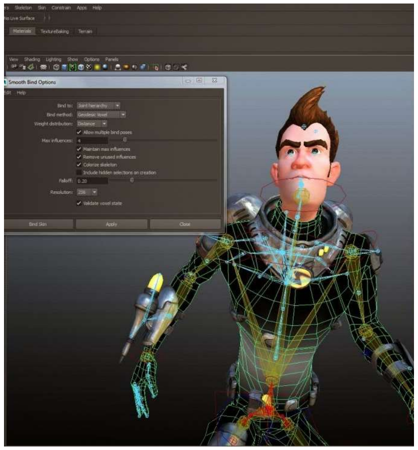
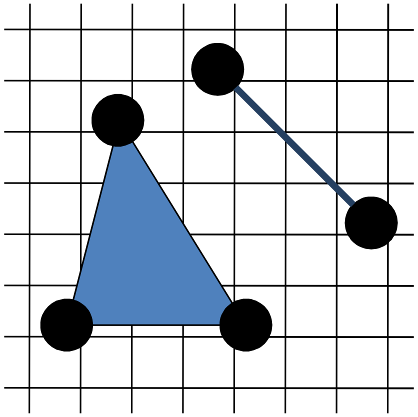
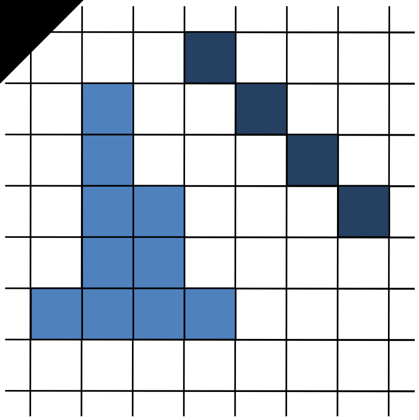


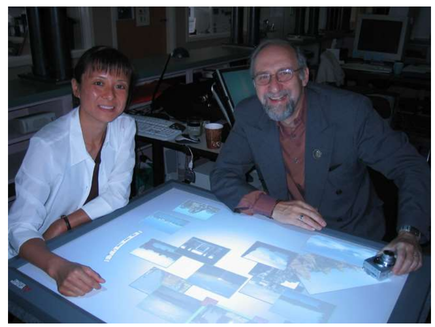
Rasterisation

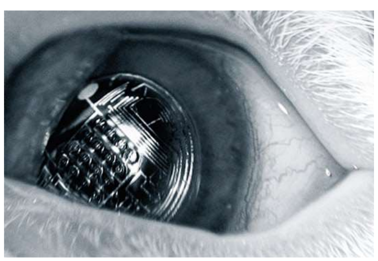




Rasterization

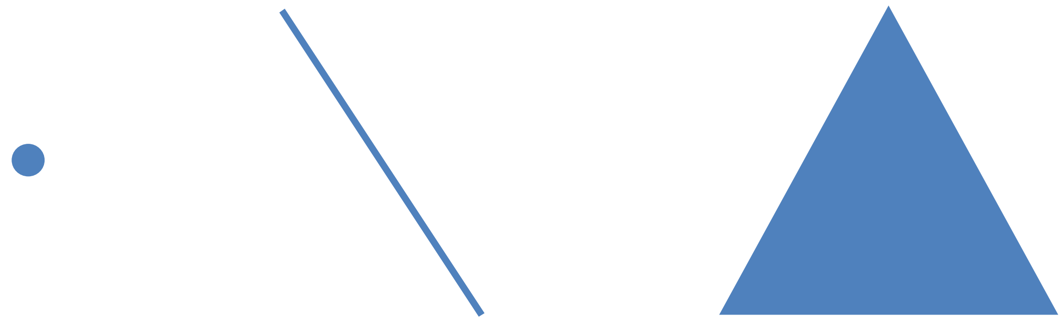
- Want to do vector graphics on a raster device

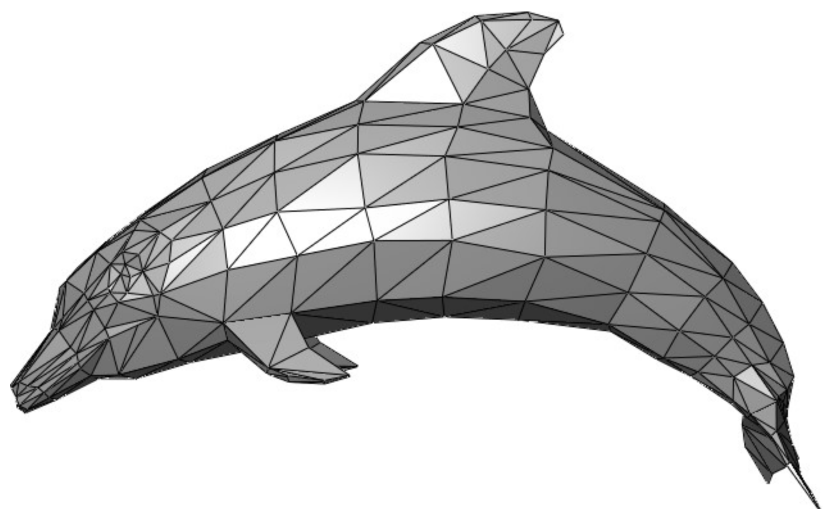






Rasterization Primitives

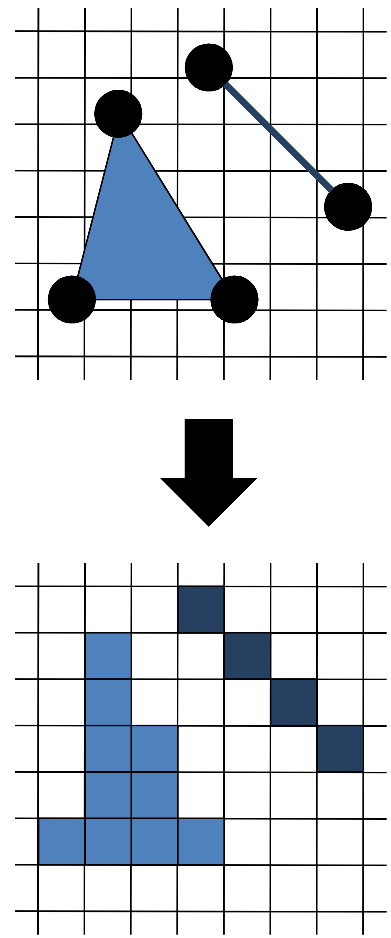
- 2D/3D
  - Point, line
  - Triangle





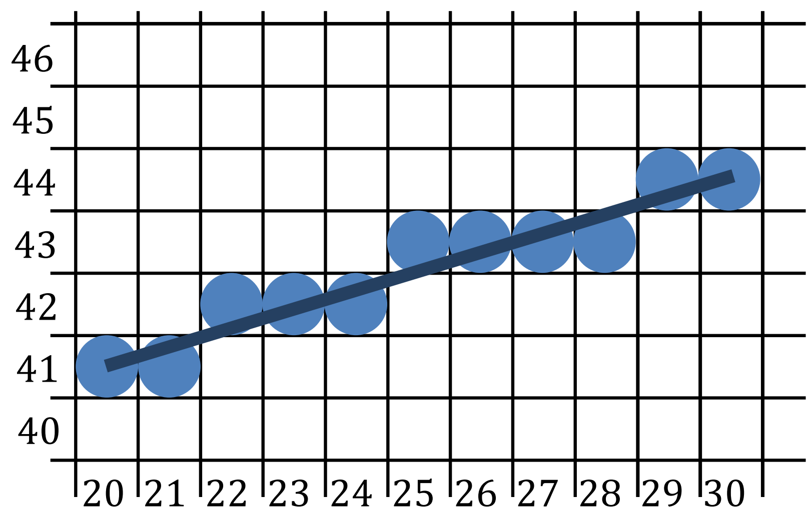
Rasterization

- Converts
  - Primitives
  - With floating point vertices



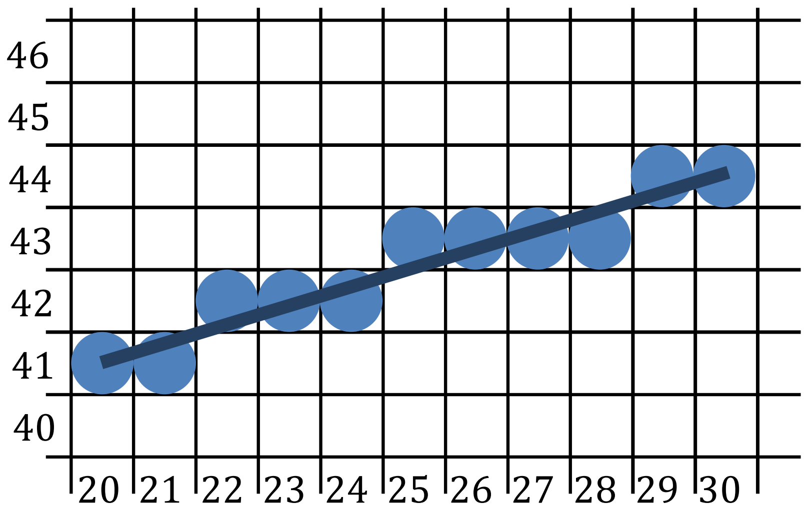
- Into
  - Pixels
  - With integer coordinates

Rasterization of Lines



Drawing Lines

- Line is a series of pixel positions
- Intermediate discrete pixel positions calculated
- Staircase effect, “jaggies” (aliasing)

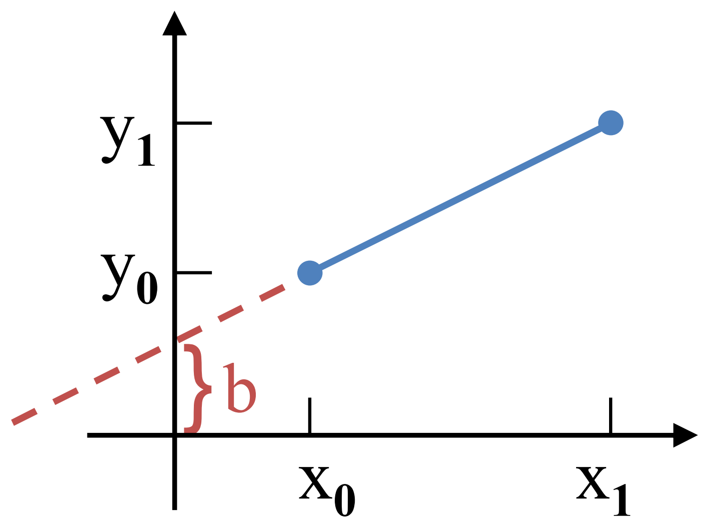


Line-Drawing Algorithms

- Line equation: $y = \mathrm{m} \cdot x + \mathrm{b}$
- Line path between two points:

$$m = \frac{y_1 - y_0}{x_1 - x_0}$$

$$b = y_0 - m \cdot x_0$$



Example

$(x_0, y_0) = (20, 41)$
$(x_1, y_1) = (30, 44)$

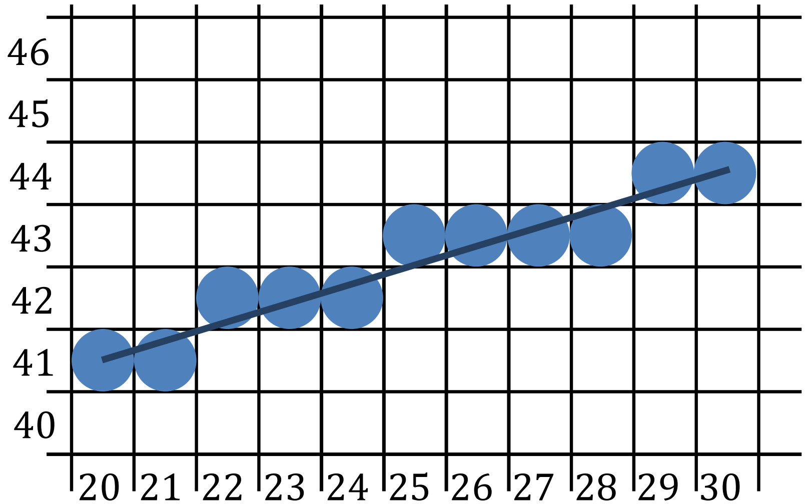
$$m = \frac{44-}{30-20} = \frac{3}{10}$$

$$b = 41 - \frac{3}{10} \cdot 20 = 35$$

$$y = \frac{3}{10} \cdot x + 35$$

x	y
21	$\frac{413}{10} \approx 41$
22	42
23	42
24	42
25	43
26	43
27	43
28	43
29	44
30	44

Example



x	y
21	$\frac{413}{10} \approx 41$
22	42
23	42
24	42
25	43
26	43
27	43
28	43
29	44
30	44

Example 2

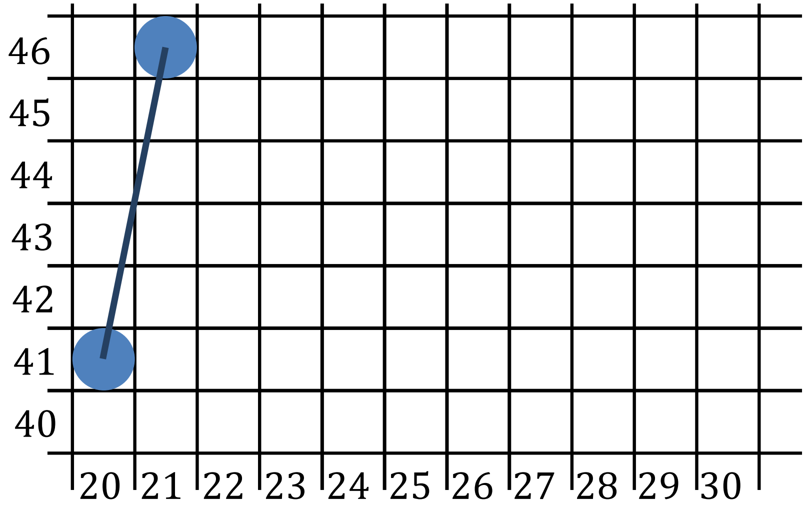
$(x_0,y_0) = (20,41)$
$(x_1,y_1) = (21,46)$

x	y
21	46

$$m = \frac{46-41}{21-20} = \frac{5}{1} = 5$$

$$b = 41 - 5 \cdot 20 = -59$$

$$y = 5 \cdot x - 59$$

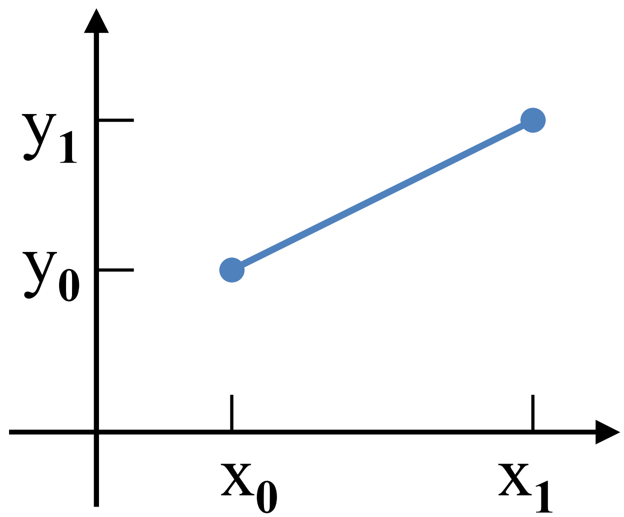


Résumé

- Quality
  - Works for some cases
    - If m < 1
- Performance
  - Division()
  - Round()
  - Floating point operation

DDA Line-Drawing Algorithm

- DDA (digital differential analyzer)
- Define $x_1 > x_0$ otherwise switch points
- $\Delta x = x_1 - x_0$
- $\Delta y = y_1 - y_0$
- Check if $|m| < 1$
  - Iterate along x
  - Otherwise iterate along y

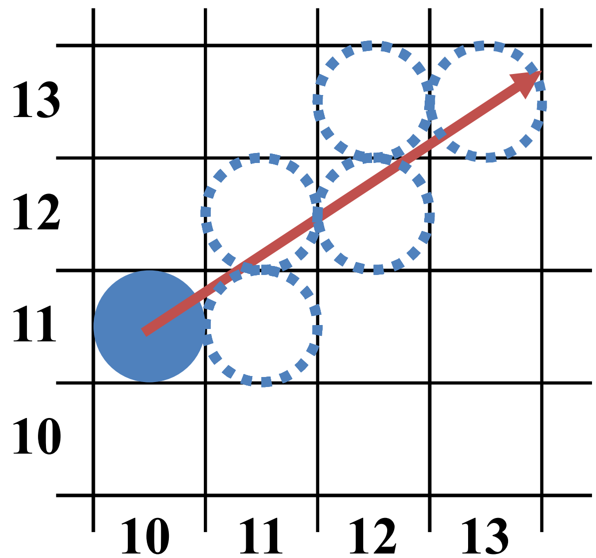


Résumé

- Quality
  - Works
- Performance
  - Division()
  - Round()
  - Floating point operation

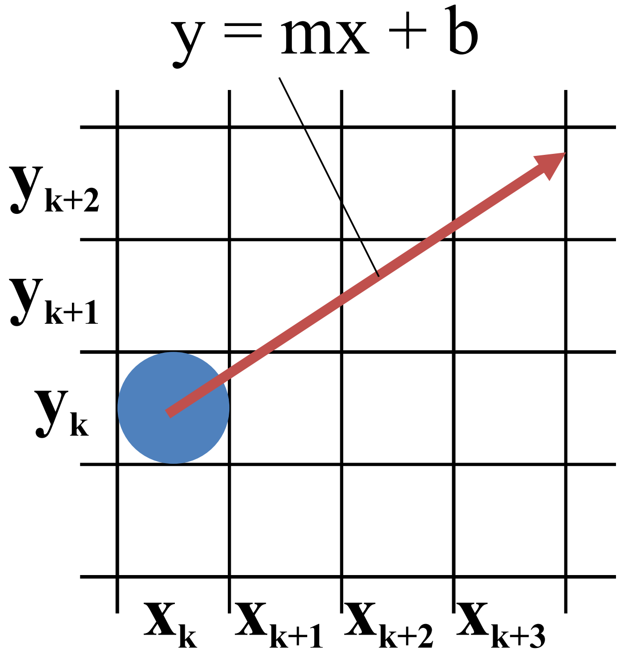
Bresenham's Line Algorithm

- Faster than simple DDA
  - Incremental integer calculations
  - Each step decision if draw upper or lower pixel
  - We analyze $|m| < 1$ case, others analog to DDA

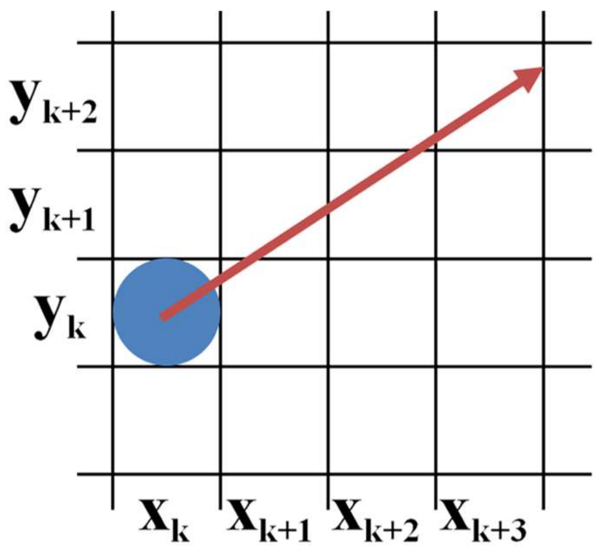


$(x_0, y_0) = (10, 11)$
$(x_1, y_1) = (13, 13)$

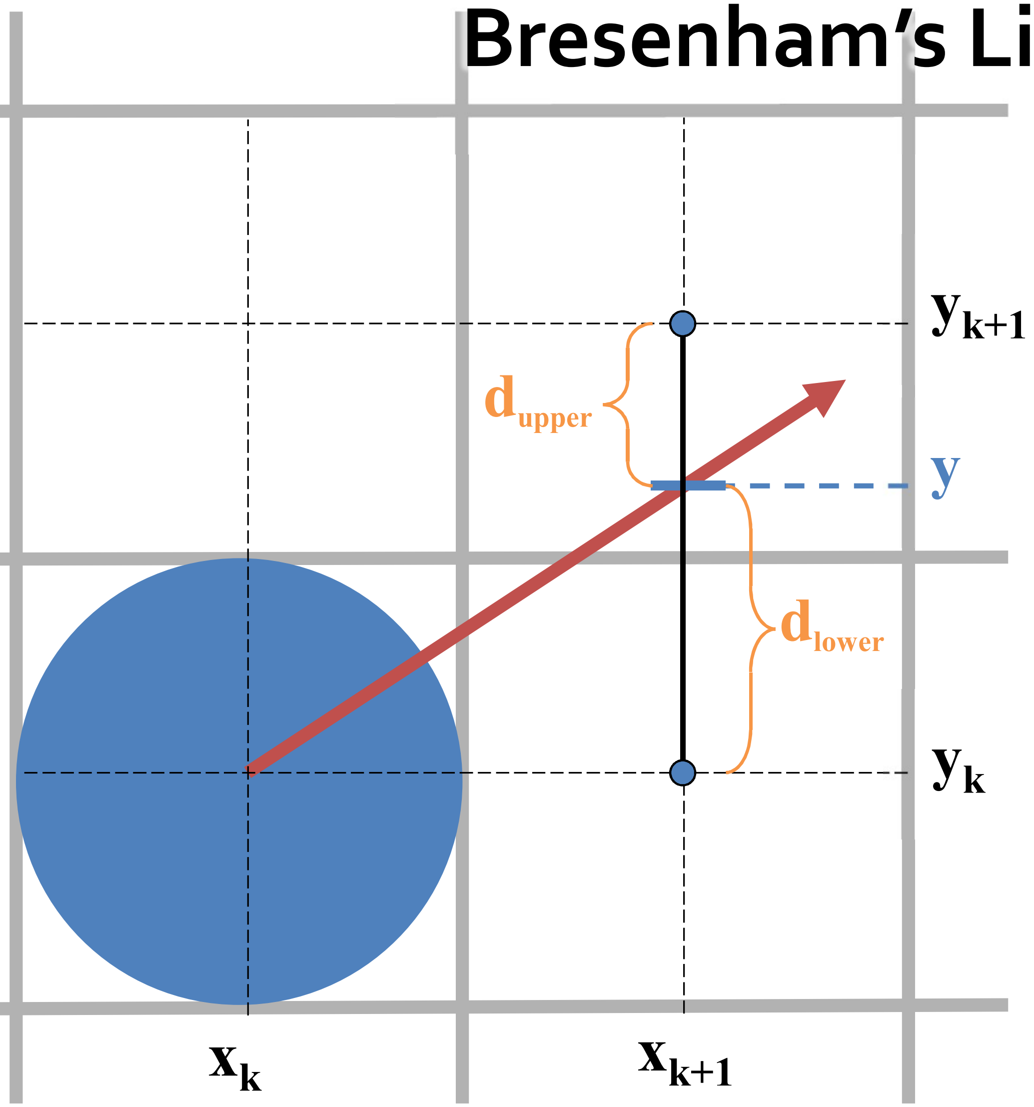
Bresenham's Line Algorithm



Bresenham's Line Algorithm



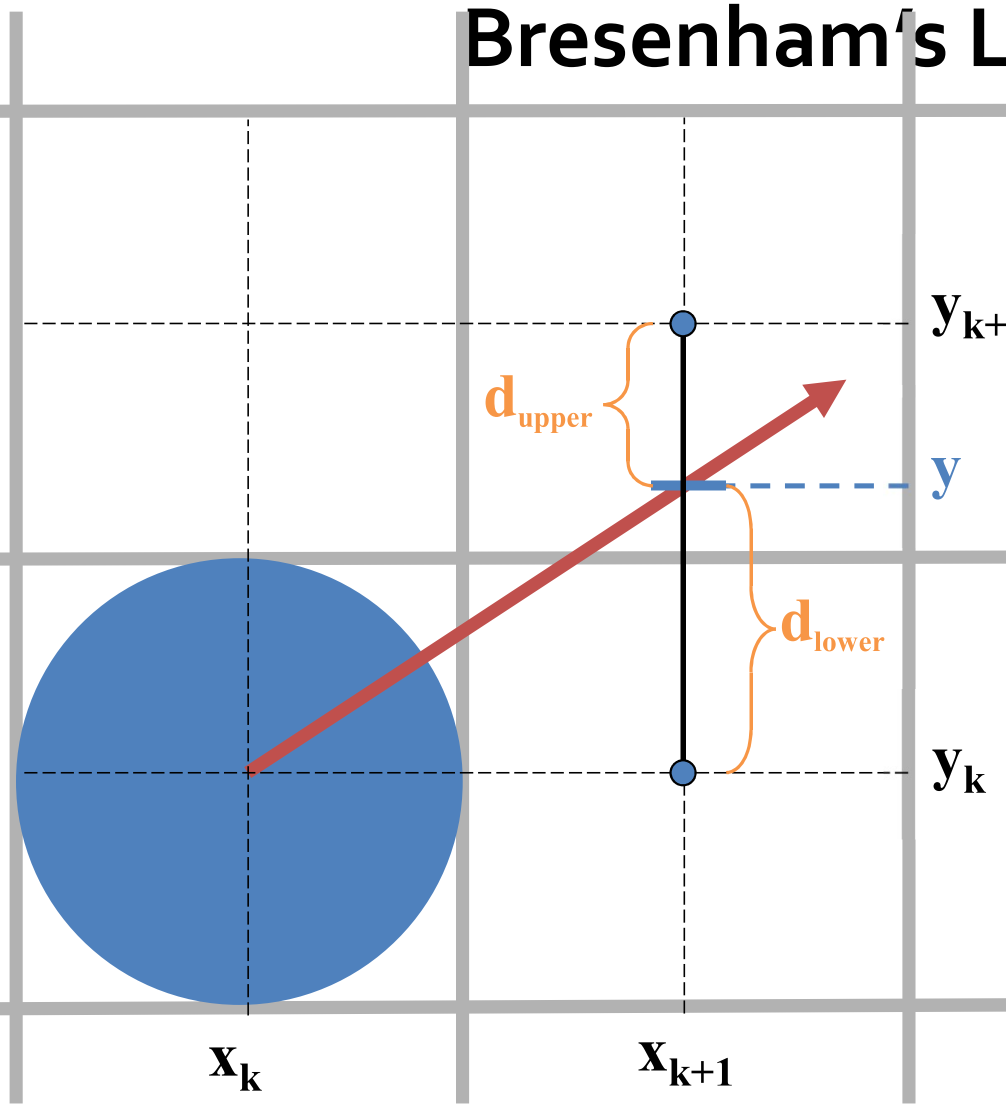
Bresenham's Line Algorithm (1/4)



- $y = m\cdot(x_k + 1) + b$
- $d_{lower} = y - y_k =$
  $m\cdot(x_k + 1) + b - y_k$
- $d_{upper} = (y_k + 1) - y =$
  $y_k + 1 - m\cdot(x_k + 1) - b$

  $d_{lower} - d_{upper} =$
  $2m\cdot(x_k + 1) - 2y_k + 2b - 1 =$
  $2\Delta y/\Delta x\cdot(x_k + 1) - 2y_k + 2b - 1$

Bresenham's Line Algorithm (2/4)

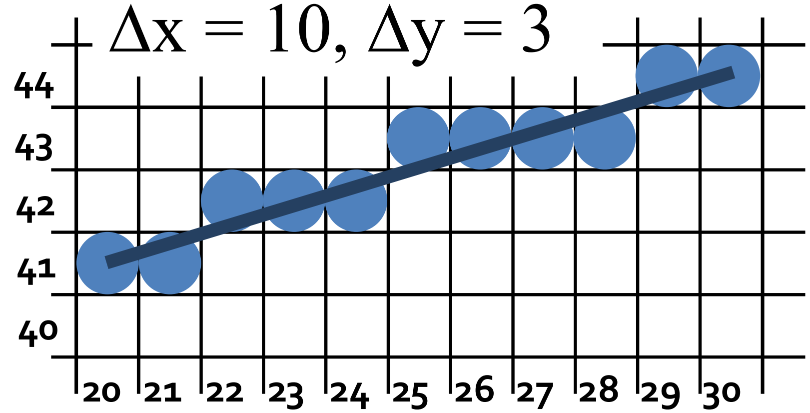


- $d_{lower} - d_{upper} =$ $2\Delta y/\Delta x \cdot (x_k + 1) - 2y_k + 2b - 1$
- Multiply with Δx to avoid division $p_k = \Delta x \cdot (d_{lower} - d_{upper})$
- p_k same sign as $(d_{lower} - d_{upper})$
- Use as decision parameter
- Make iterative

Bresenham: Example

k	p_k	(x_{k+1}, y_{k+1})
		$\boxed{(20, 41)}$
0	$\boxed{-4}$	$\boxed{(21, 41)}$
1	$\boxed{2}$	$\boxed{(22, 42)}$
2	$\boxed{-12}$	$(23, 42)$
3	-6	$(24, 42)$
4	0	$(25, 43)$
5	-14	$(26, 43)$
6	-8	$(27, 43)$
7	-2	$(28, 43)$
8	4	$(29, 44)$
9	-10	$\boxed{(30, 44)}$

$\Delta x = 10, \Delta y = 3$




```
p0 = 2Δy - Δx
For(Δx - 1 steps) {
  If(pk <= 0) {
    draw pixel(xk+1, yk);
    pk+1 = pk + 2Δy; }
  else {
    draw pixel(xk+1, yk+1);
    pk+1= pk + 2Δy - 2Δx; }
}
```

Résumé

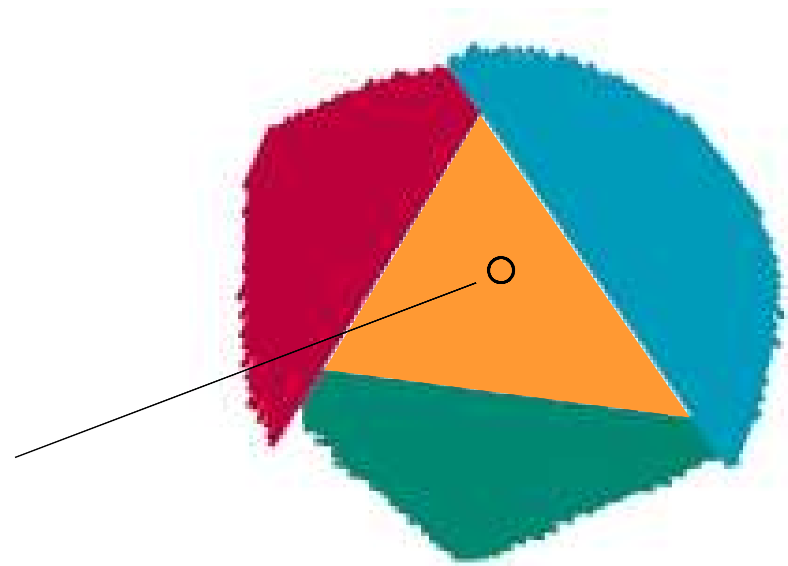
- Quality
  - Works
- Performance
  - No division()
  - No round()
  - No floating point operation
- Idea
  - Adaptable to circles, other curves
  - Narrow decision to simple/binary decision

Flood-Fill Algorithm

- Pixel filling of area
  - Start from interior point
  - "Flood" internal region
- Works for arbitrary shapes

Flood-Fill: Boundary and Seed Point

- Area must be distinguishable from boundaries
- Example
  - Area defined within multiple color boundaries



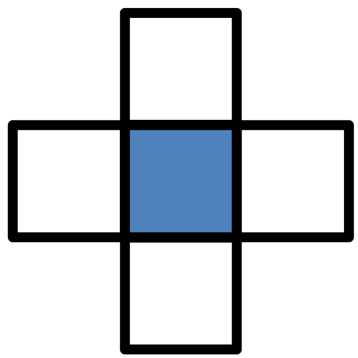
seed point

Simple Flood-Fill Algorithm

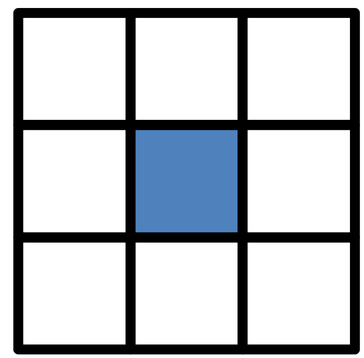
```
void floodFill4(x, y, newColor, oldColor) {
  color = getPixel(x, y);
  if (color == oldColor) {
    drawPixel (x, y, newColor);
    floodFill4 (x-1, y, newColor, oldColor); // left
    floodFill4 (x, y+1, newColor, oldColor); // up
    floodFill4 (x+1, y, newColor, oldColor); // right
    floodFill4 (x, y-1, newColor, oldColor); // down
  }
}
```

Flood-Fill: Who is my Neighbour?

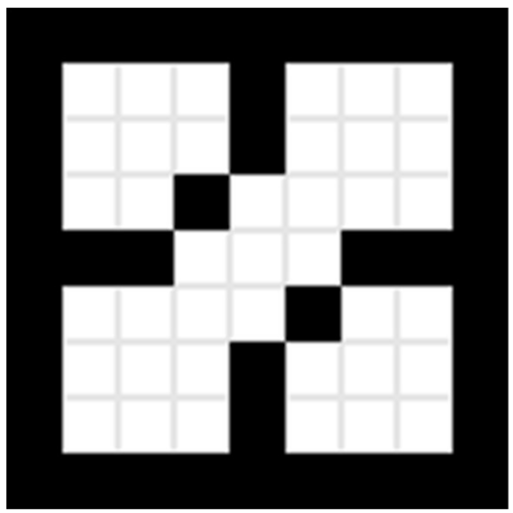
- ***4-connected*** means, that a connection is only valid in these 4 directions



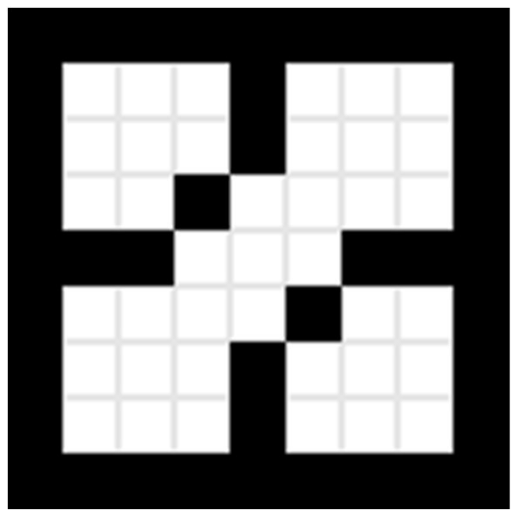
- ***8-connected*** means, that a connection is valid in these 8 directions



Flood-Fill: Connectedness

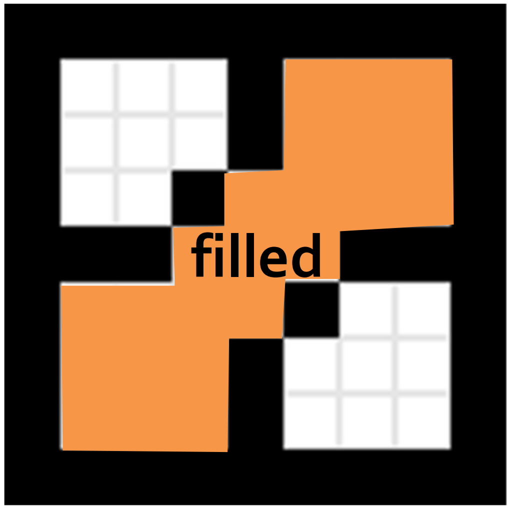


4-connected

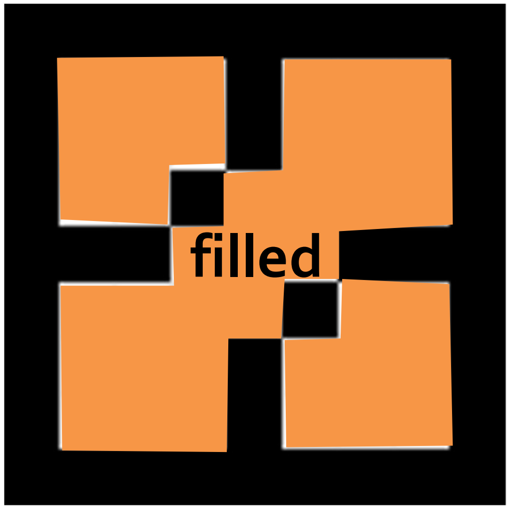


8-connected

Flood-Fill: Connectedness



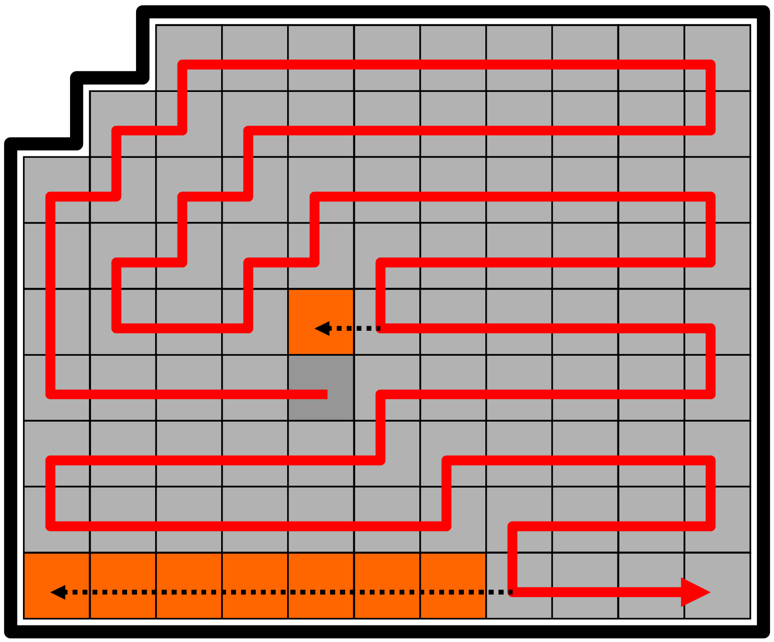
4-connected

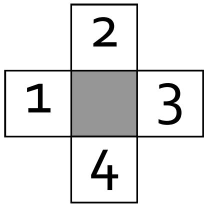


8-connected

Bad Behavior of Simple Flood-Fill

- Danger of stack overflow (recursion!)
- Memory inefficient

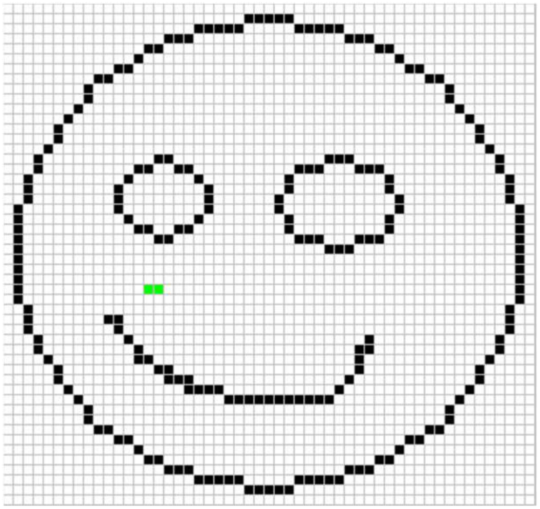




recursion sequence

Span Flood-Fill Algorithm

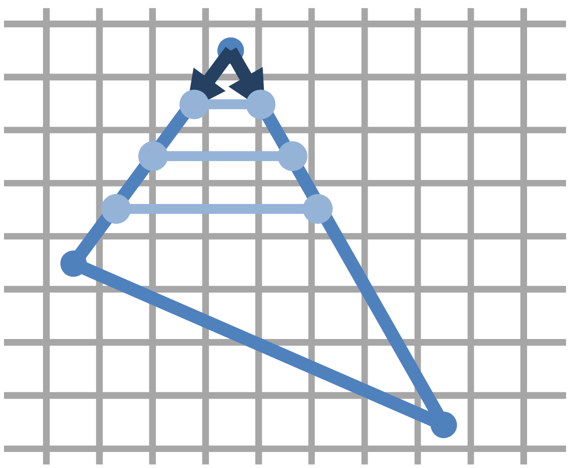
- Incremental horizontal fill (e.x.: left to right)
- Recursive vertical fill (e.x.: first up then down)
- More efficent methods possible, but more complex



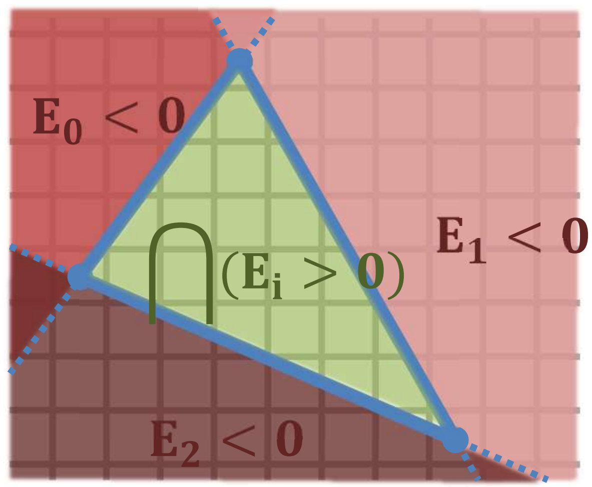
Triangle Rasterization

Direct rendering of filled triangles

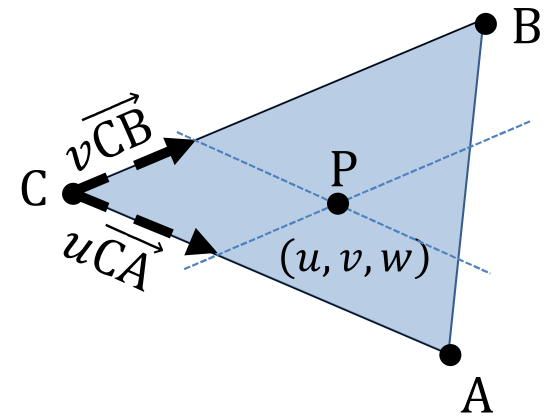
Scan Converting a Triangle



Edge Walking



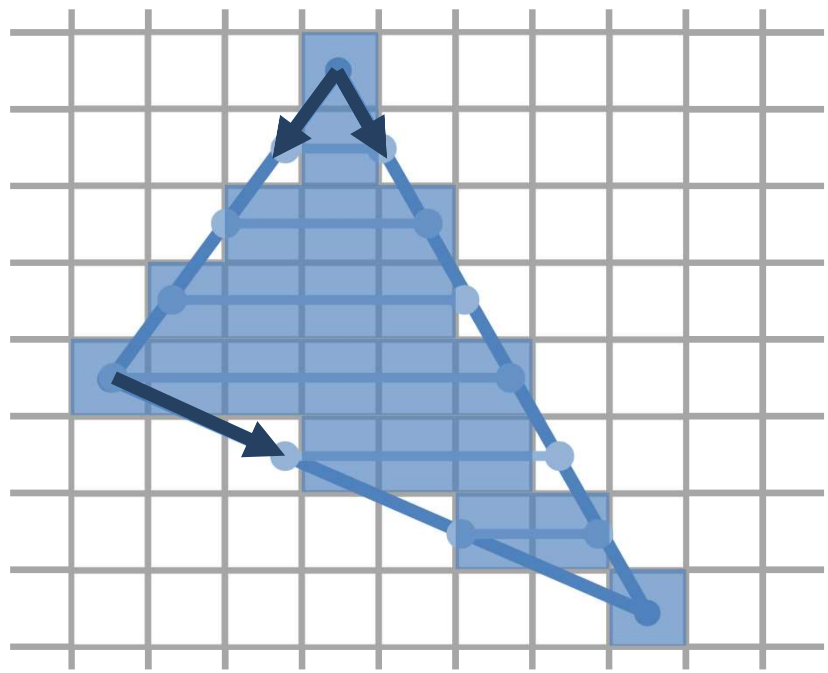
Edge Equations



Barycentric Coordinates

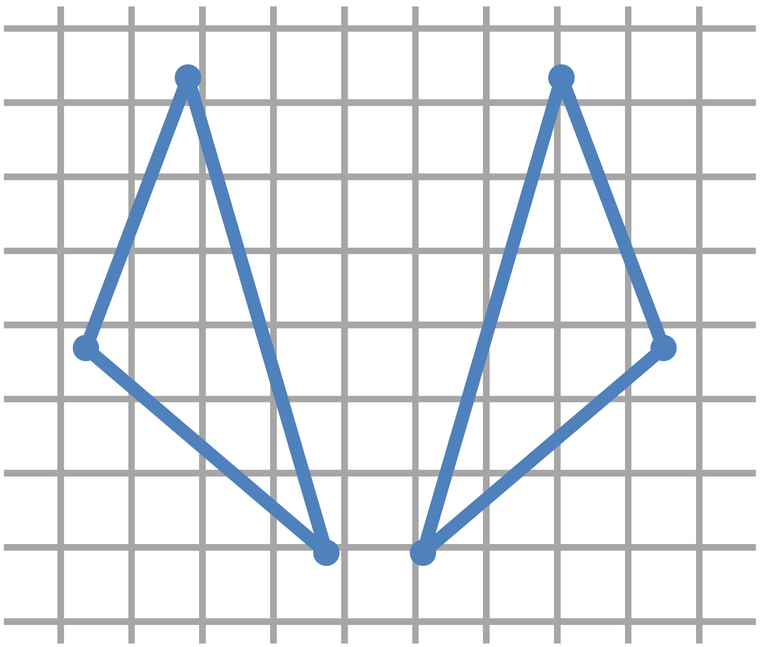
Edge Walking

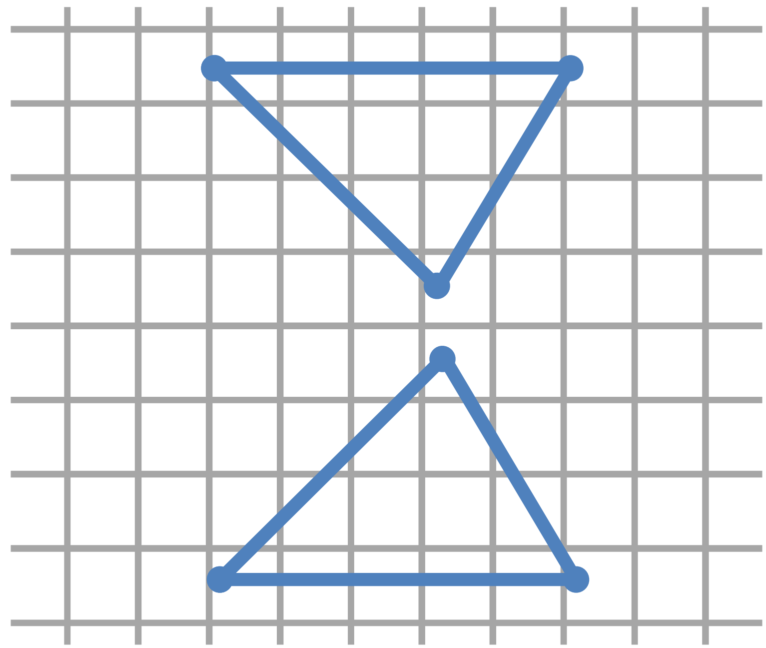
1. Sort vertices in y
2. Walk down edges from extremal y-point
3. Compute spans
4. Switch in 3rd edge
5. Repeat 2 and 3
   until lowest point



Possible Cases

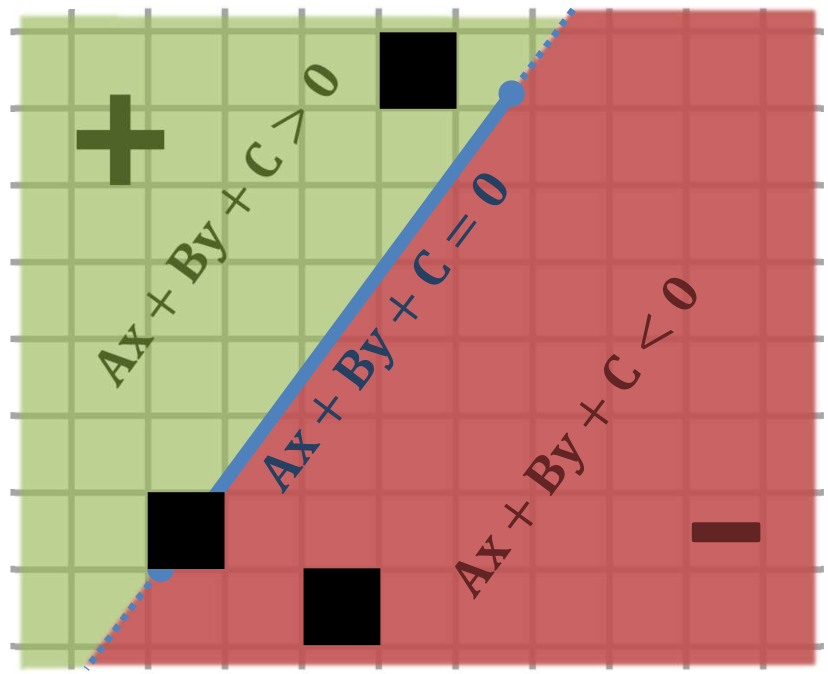
- Left or right y middle point
- 2 highest/lowest points





Edge Equations

- Defines positive/negative half-spaces
- Reverse spaces by multiplication by -1
- $E(x, y) = Ax + By + C$
- Value for pixels?
  - $E\left(P_x, P_y\right)$



Given 2 points $\begin{pmatrix} x_0 \\ y_0 \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \end{pmatrix}$, compute A,B,C

1. Setup equation system

$$Ax_0 + By_0 + C = 0 \quad Ax_1 + By_1 + C = 0$$

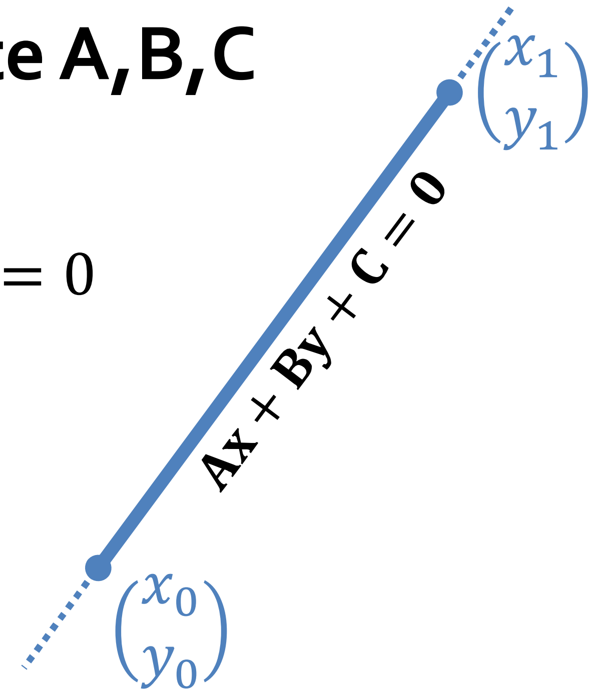
2. Matrix representation

$$\begin{bmatrix} x_0 & y_0 \\ x_1 & y_1 \end{bmatrix} \begin{bmatrix} A \\ B \end{bmatrix} + \begin{bmatrix} C \\ C \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \leftrightarrow \begin{bmatrix} x_0 & y_0 \\ x_1 & y_1 \end{bmatrix} \begin{bmatrix} A \\ B \end{bmatrix} = -C \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

3. Solve

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

4. Choose C

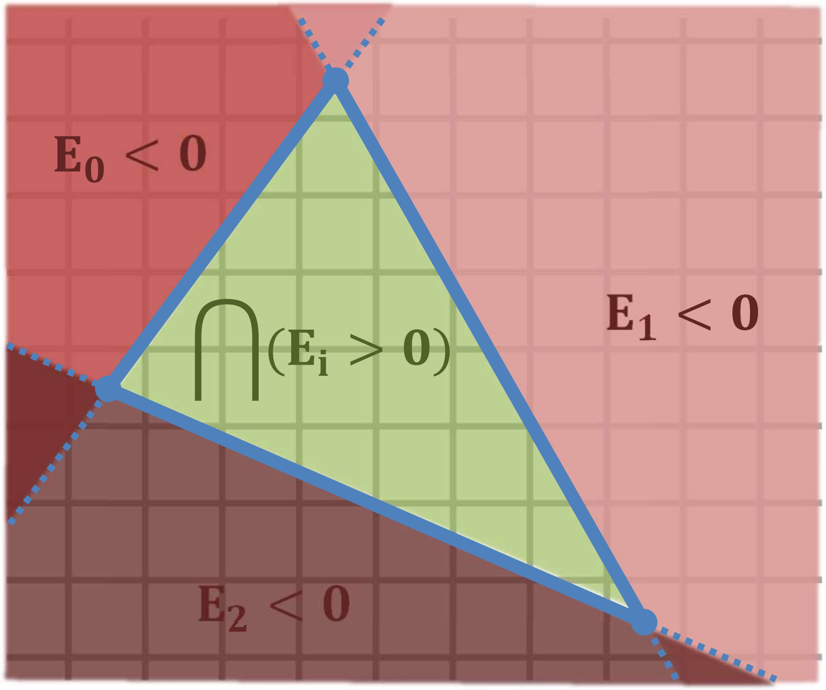


Edge Equations for the Triangle

$$E_0(x, y) = A_0 x + B_0 y + C_0$$

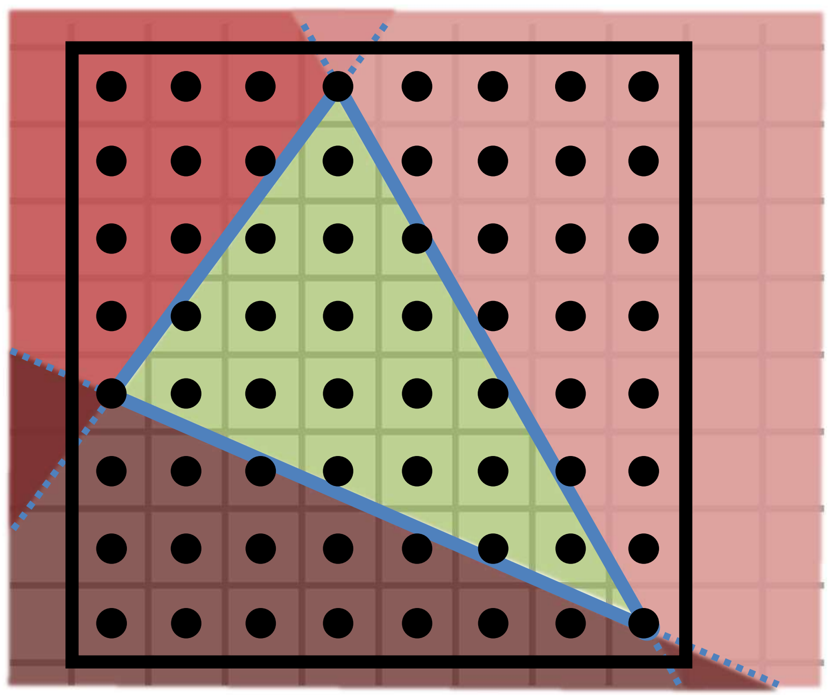
$$E_1(x, y) = A_1 x + B_1 y + C_1$$

$$E_2(x, y) = A_2 x + B_2 y + C_2$$



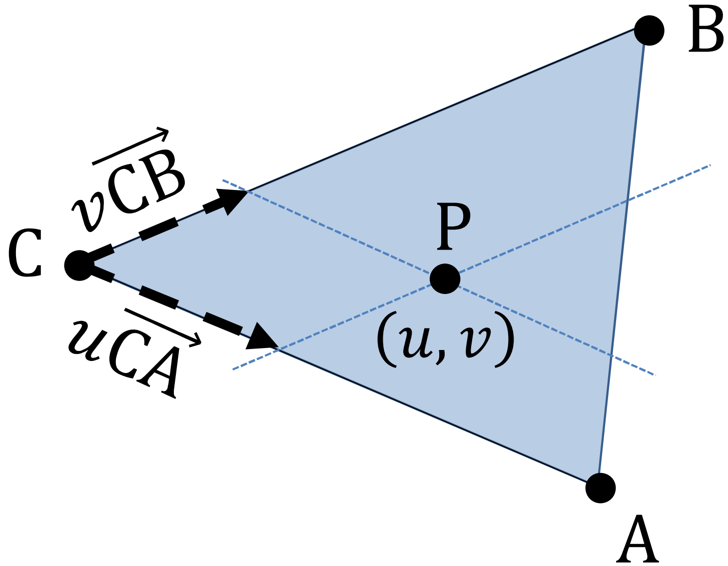
Testing Pixels

- Find bounding box
- Test $\bigcap(\mathbf{E_i} > \mathbf{0})$ for each pixel
- Happy?



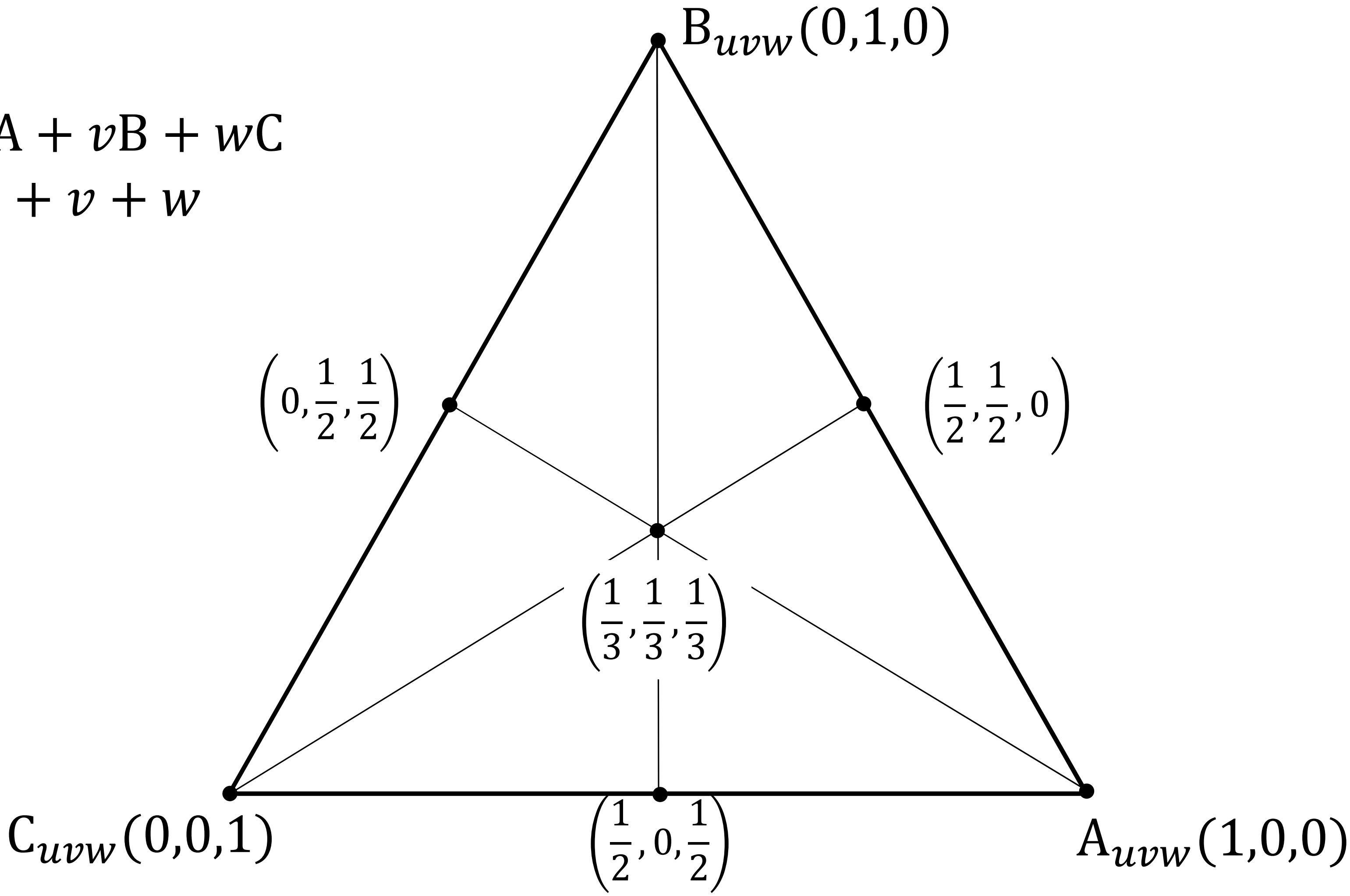
Barycentric Coordinates of P

- Define $\mathrm{P} = \mathrm{C} + u\overrightarrow{\mathrm{CA}} + v\overrightarrow{\mathrm{CB}}$
  $= u\mathrm{A} + v\mathrm{B} + (1 - u - v)\mathrm{C}$
  $= u\mathrm{A} + v\mathrm{B} + w\mathrm{C}$ with $1 = u + v + w$
- Triangle can also be 3d



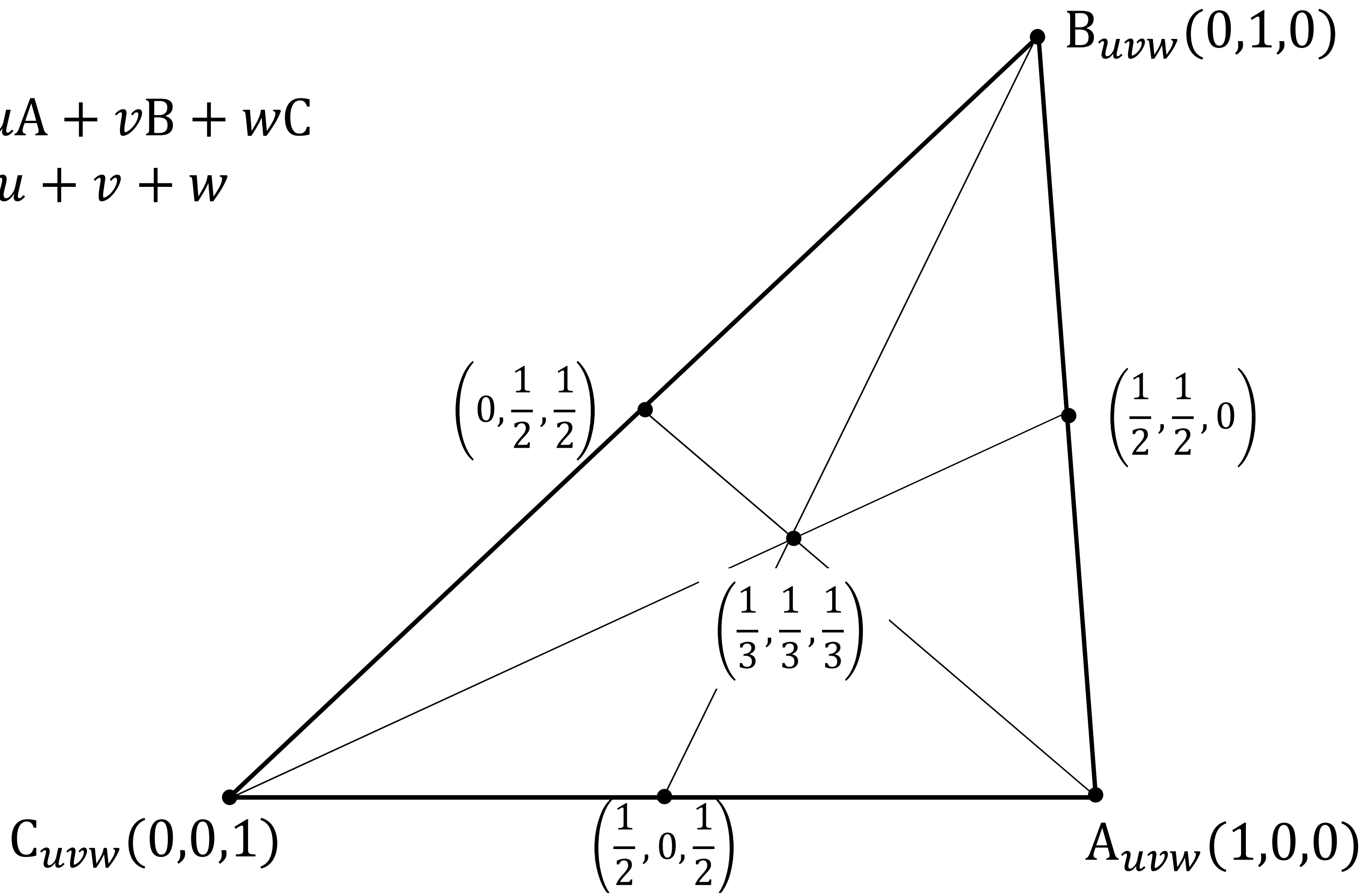
BC – Special Points

$$P = uA + vB + wC$$
$$1 = u + v + w$$



Barycentric Coordinates – Invariance

$$P = uA + vB + wC$$
$$1 = u + v + w$$



BC – Inside Triangle Test

- Also outside triangle
- In triangle if (u, v, w) all same sign
  - For CCW $(u, v, w) \geq 0$

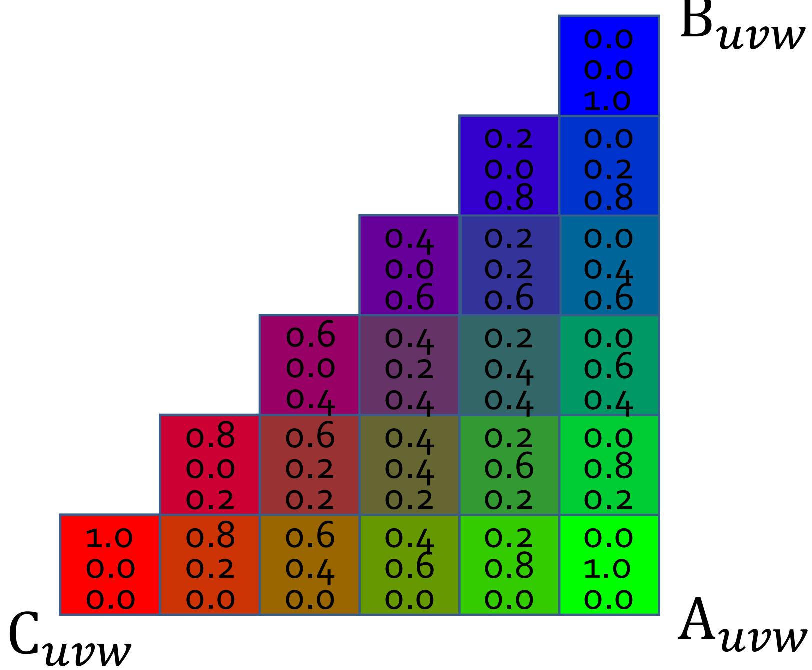
1.2 -1.4 1.2	1.0 -1.2 1.2	0.8 -1.0 1.2	0.6 -0.8 1.2	0.4 -0.6 1.2	0.2 -0.4 1.2	0.0 -0.2 1.2	-0.2 0.0 1.2
1.2 -1.2 1.0	1.0 -1.0 1.0	0.8 -0.8 1.0	0.6 -0.6 1.0	0.4 -0.4 1.0	0.2 -0.2 1.0	0.0 0.0 1.0	-0.2 0.2 1.0
1.2 -1.0 0.8	1.0 -0.8 0.8	0.8 -0.6 0.8	0.6 -0.4 0.8	0.4 -0.2 0.8	0.2 0.0 0.8	0.0 0.2 0.8	-0.2 0.4 0.8
1.2 -0.8 0.6	1.0 -0.6 0.6	0.8 -0.4 0.6	0.6 -0.2 0.6	0.4 0.0 0.6	0.2 0.2 0.6	0.0 0.4 0.6	-0.2 0.6 0.6
1.2 -0.6 0.4	1.0 -0.4 0.4	0.8 -0.2 0.4	0.6 0.0 0.4	0.4 0.2 0.4	0.2 0.4 0.4	0.0 0.6 0.4	-0.2 0.8 0.4
1.2 -0.4 0.2	1.0 -0.2 0.2	0.8 0.0 0.2	0.6 0.2 0.2	0.4 0.4 0.2	0.2 0.6 0.2	0.0 0.8 0.2	-0.2 1.0 0.2
1.2 -0.2 0.0	1.0 0.0 0.0	0.8 0.2 0.0	0.6 0.4 0.0	0.4 0.6 0.0	0.2 0.8 0.0	0.0 1.0 0.0	-0.2 1.2 0.0
1.2 0.0 -0.2	1.0 0.2 -0.2	0.8 0.4 -0.2	0.6 0.6 -0.2	0.4 0.8 -0.2	0.2 1.0 -0.2	0.0 1.2 -0.2	-0.2 1.4 -0.2

Interpolation with Barycentric Coordinates

- Interpolate per point attributes over the triangle
  - Colors
  - z-values
  - texture coordinates
  - …
- Given barycentric coordinates $\mathrm{P} = u\mathrm{A} + v\mathrm{B} + w\mathrm{C}$
- Attribute value for a point P on a triangle
- $\mathrm{P}_{attrib} = u\mathrm{A}_{attrib} + v\mathrm{B}_{attrib} + w\mathrm{C}_{attrib}$

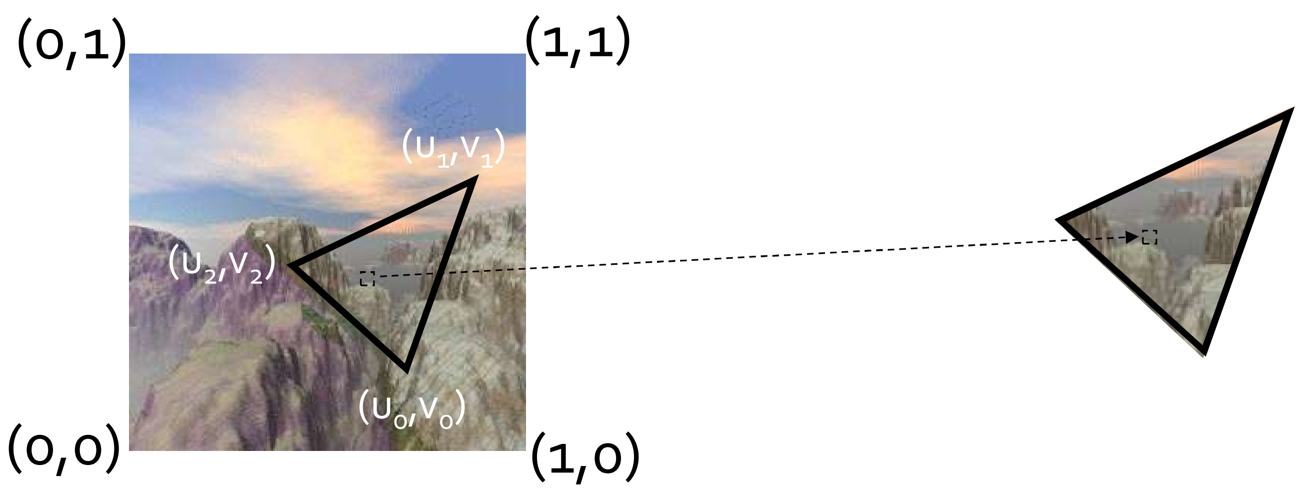
Barycentric Coordinates – Color Interpolation

- $\mathrm{P} = u\mathrm{A} + v\mathrm{B} + w\mathrm{C}$
- $\mathrm{P} = u\langle Green\rangle + v\langle Blue\rangle + w\langle Red\rangle$
- A.k.a. Gouraud interpolation



Barycentric Coordinates – Texture Interpolation

- $\mathrm{P} = u\mathrm{A} + v\mathrm{B} + w\mathrm{C}$
- $\mathrm{P}(u, v) = u\mathrm{A}_{(u_0,v_0)} + v\mathrm{B}_{(u_1,v_1)} + w\mathrm{C}_{(u_2,v_2)}$

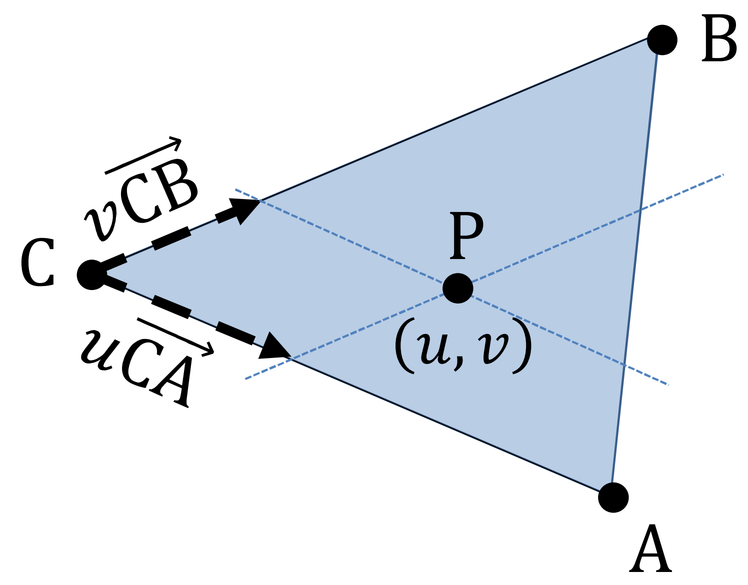


Barycentric Coordinates of P (2D)

$$P = C + u\overrightarrow{CA} + v\overrightarrow{CB}$$

$$\begin{pmatrix}\overrightarrow{CA} & \overrightarrow{CB}\end{pmatrix}\begin{pmatrix}u \\ v\end{pmatrix} = P - C$$

$$\begin{pmatrix}A - C & B - C\end{pmatrix}\begin{pmatrix}u \\ v\end{pmatrix} = P - C$$



Barycentric Coordinates of P (2D)

- Cramer's Rule

$$\begin{pmatrix} u \\ v \end{pmatrix} = \frac{1}{|\mathrm{A}-\mathrm{C} \quad \mathrm{B}-\mathrm{C}|} \begin{pmatrix} |\mathrm{P}-\mathrm{C} \quad \mathrm{B}-\mathrm{C}| \\ |\mathrm{A}-\mathrm{C} \quad \mathrm{P}-\mathrm{C}| \end{pmatrix}$$

- Point is inside triangle iff (means if and only if)

$$u \geq 0 \ \cap v \geq 0 \cap (u+v) \leq 1$$