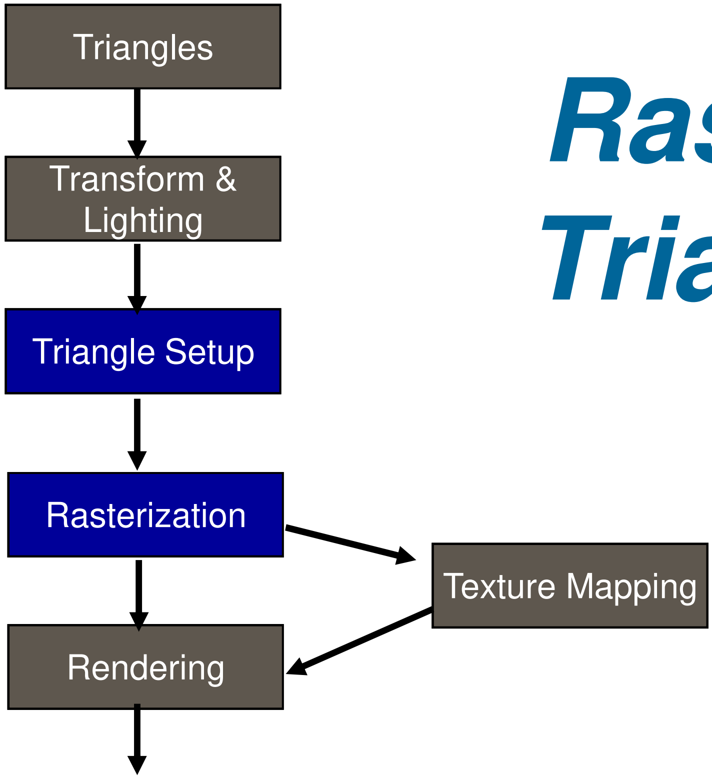
Computer Graphics

by Ruen-Rone Lee
ICL/ITRI

Wrap up from last Class

- Introduction to Computer Graphics
- Graphics Applications
- Graphics Display System
- Concept of Graphics Rendering Pipeline
- OpenGL Introduction

Part I:
Conventional 3D Graphics Pipeline



Rasterization and Triangle Setup

Primitives

Rasterization

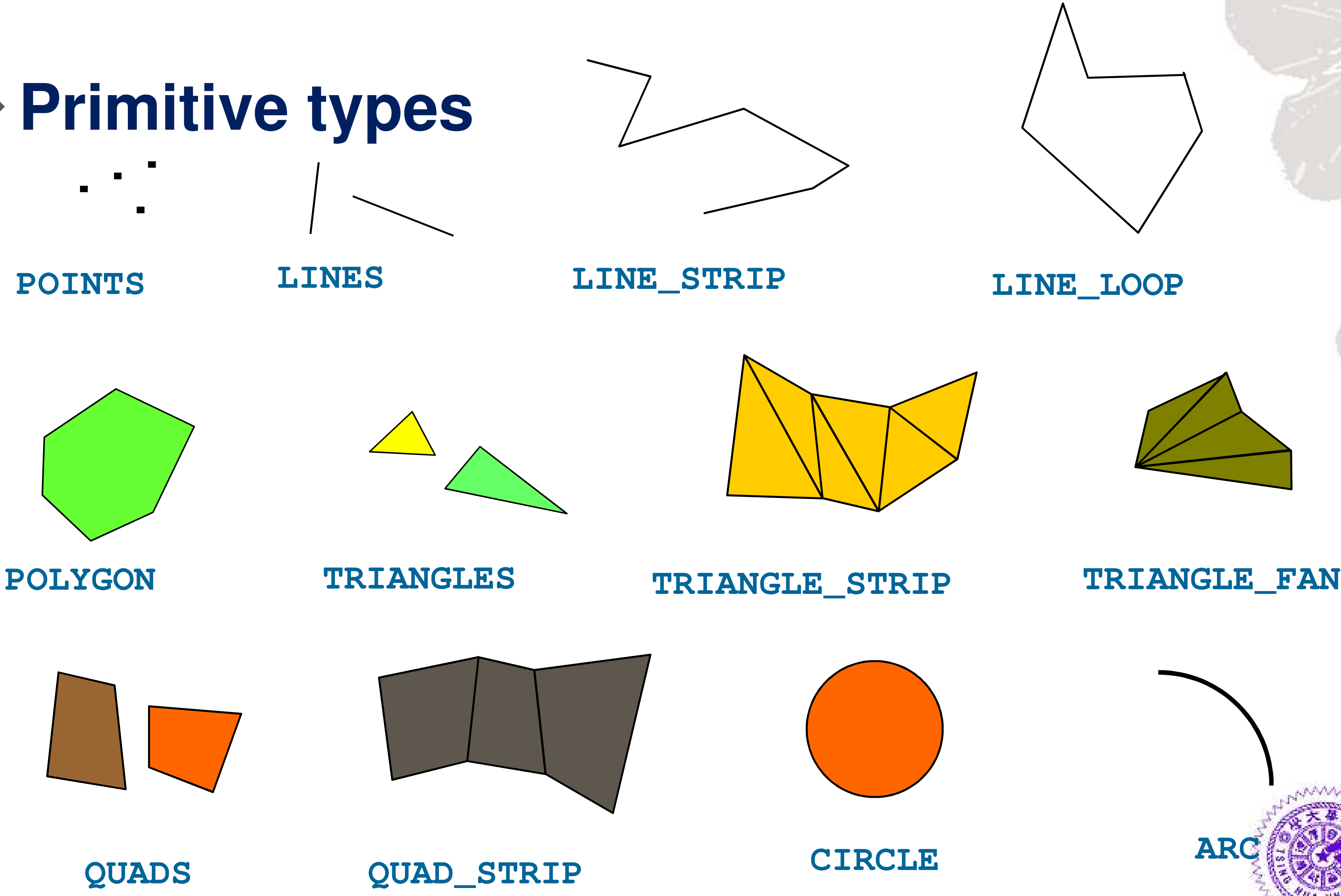
Triangle Setup

Primitives

Point
Line
Circle
Polygon
Triangles

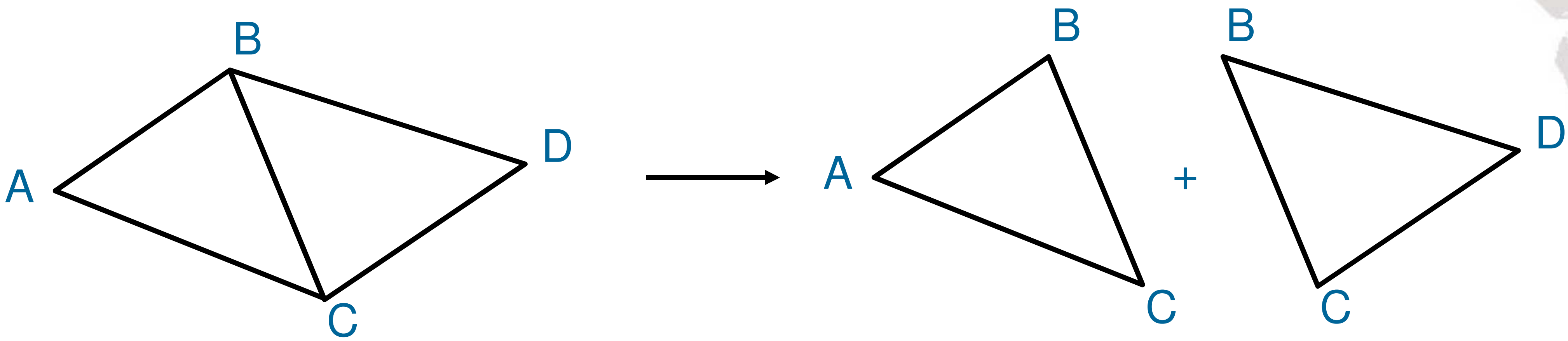
Primitives

- Primitive types



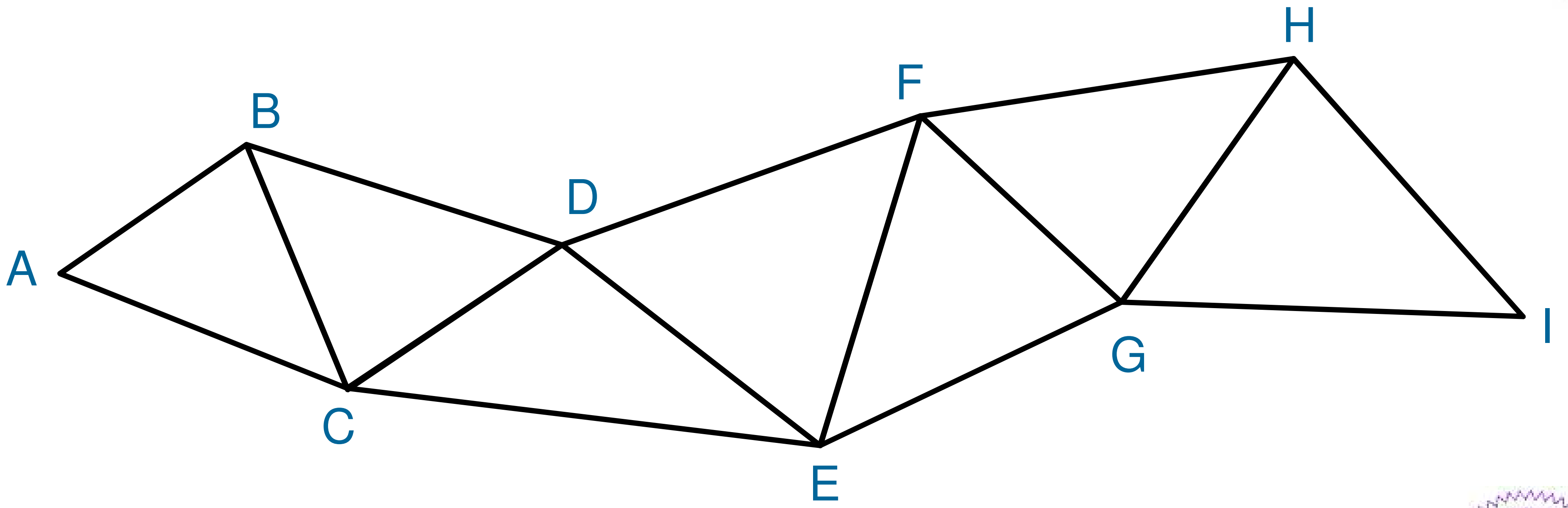
Triangles

- **Triangle List**
  - **A list of individually defined triangles**



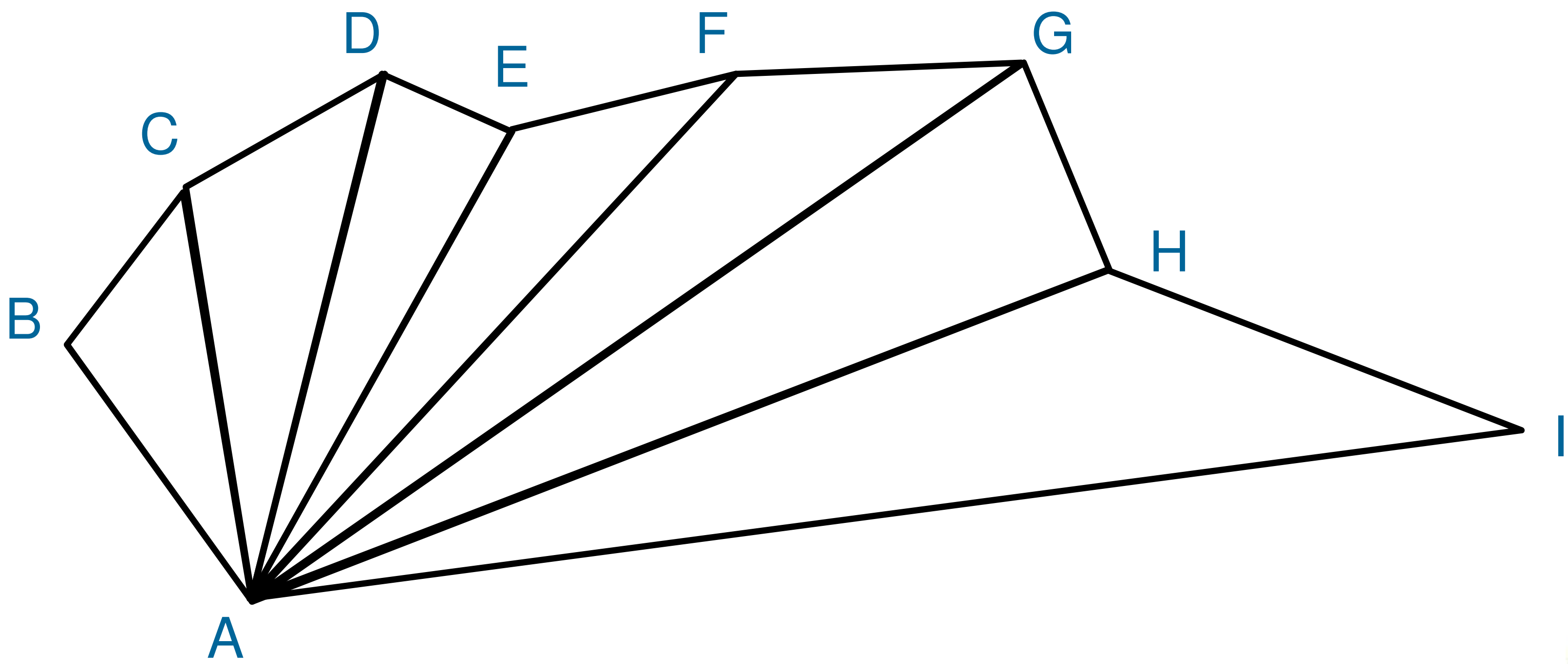
Triangles

- Triangle Strip
  - A new triangle is defined with one new vertex and two shared vertices from triangle defined immediate prior to it

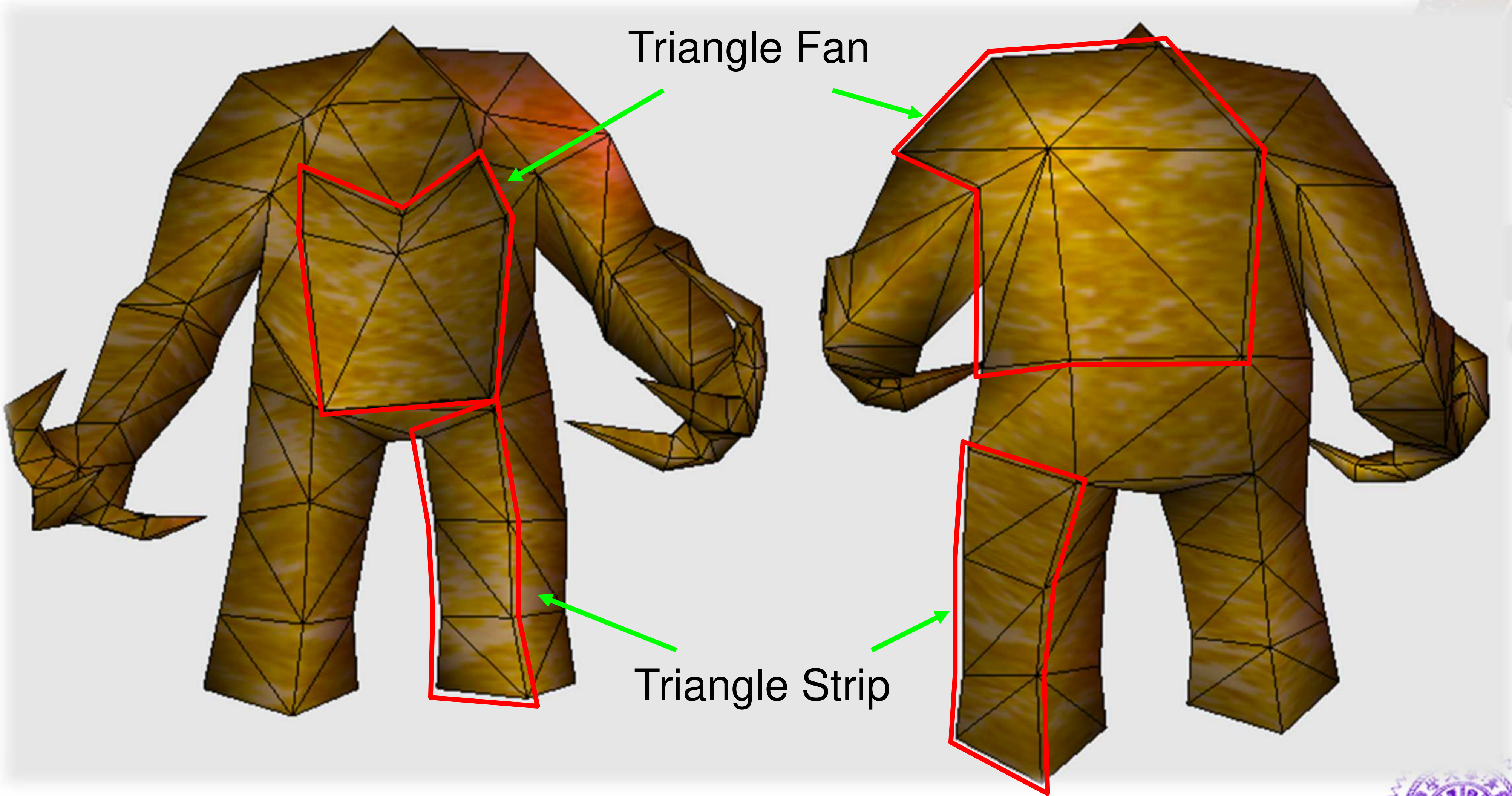


Triangles

- **Triangle Fan**
  - **Similar to triangle strip but with one common shared vertex for each triangle**

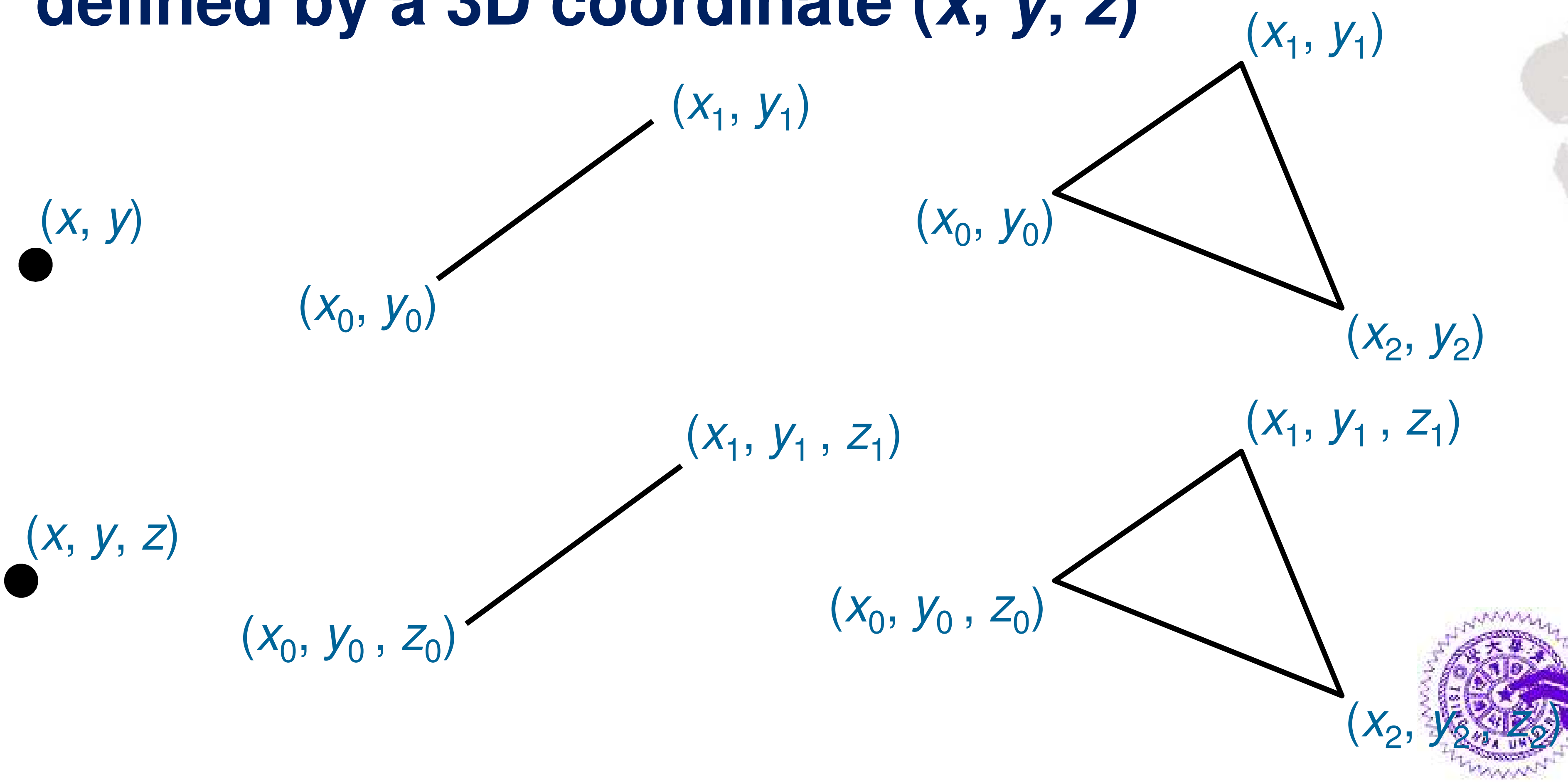


Application of Triangle Primitives



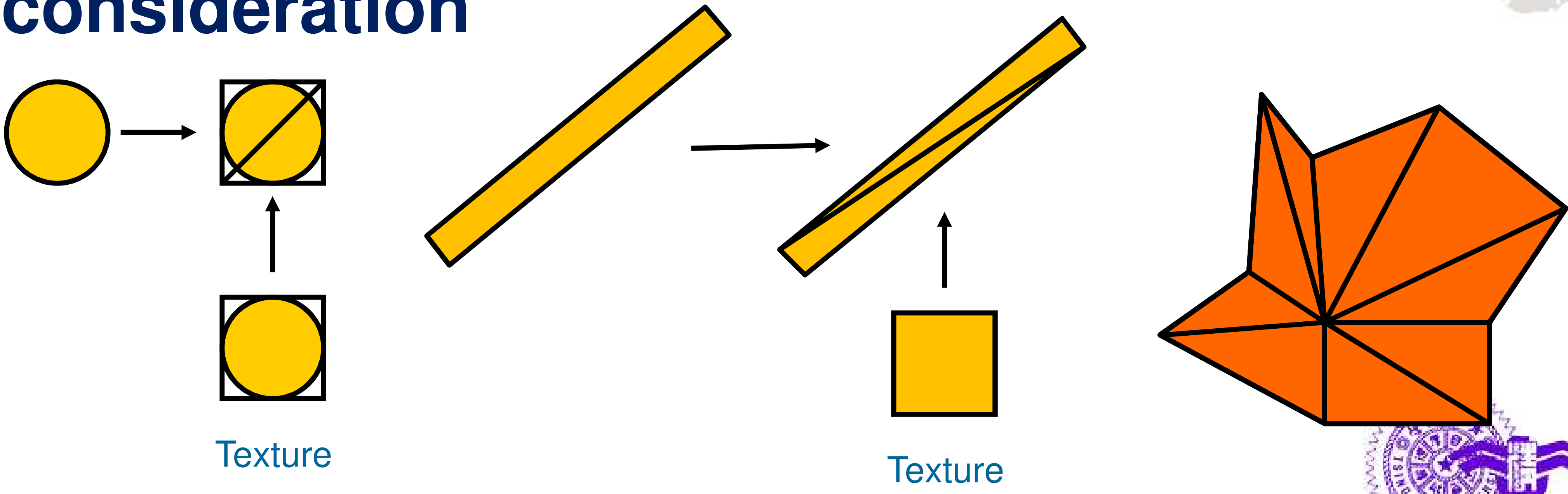
2D vs. 3D Primitives

- **Points / Lines / Triangles (Polygons)**
  - **Similar to 2D primitives except each vertex is defined by a 3D coordinate (*x*, *y*, *z*)**



Implementation Consideration

- **Points are sometimes implemented as sprite with size and texture defined.**
- **Points, lines or polygons can be converted into triangles for implementation consideration**



Basic 3D Modeling

Modeling using Primitives
Model Editing
Model Data

Modeling using Primitives

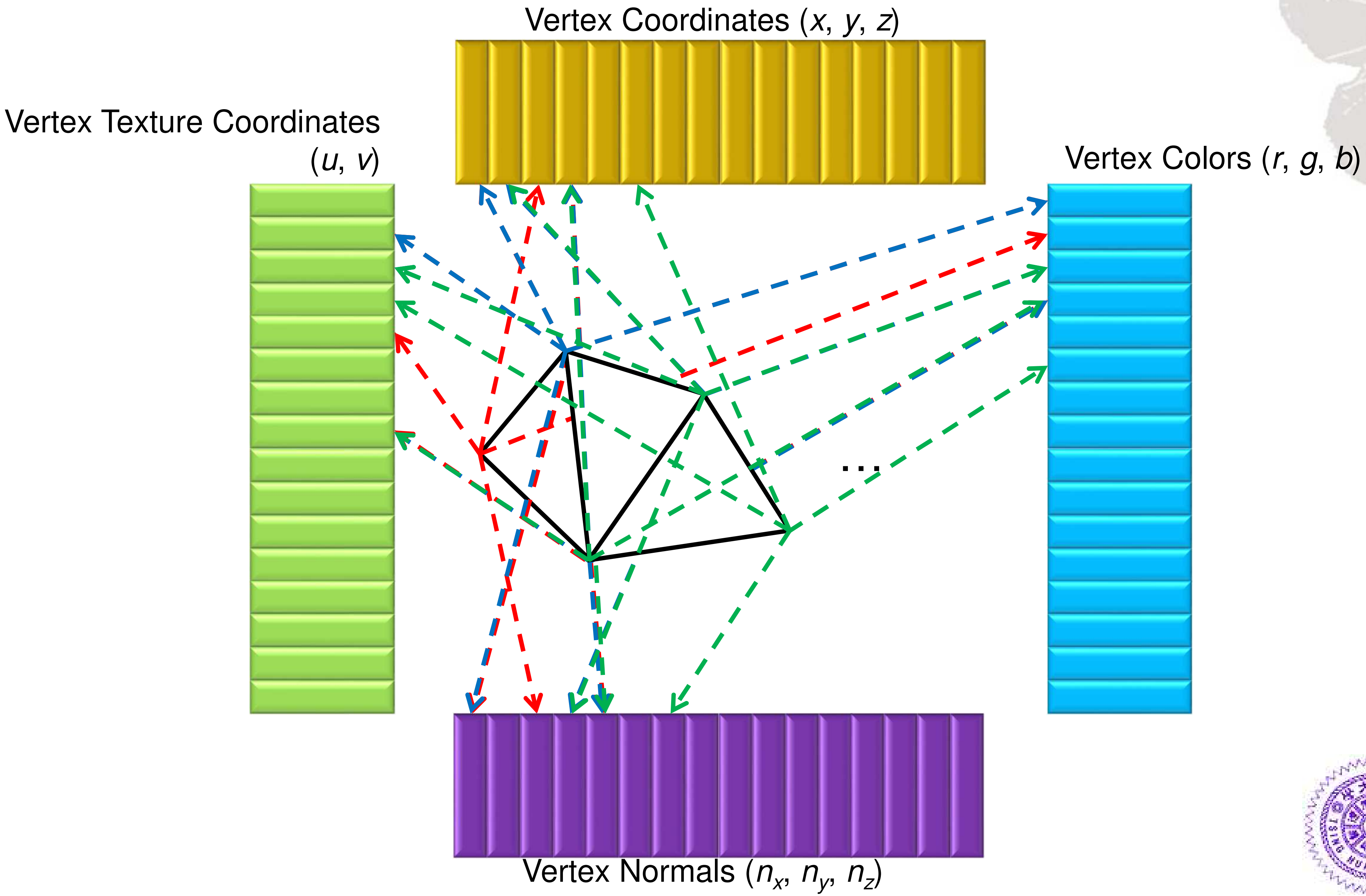
- **Using geometric primitives as the basic building components**

Getting started with
Google SketchUp

Modeling using Primitives

- Using primitives and the associated operations

Model Data



Vertex Coordinates (x, y, z)
Vertex Texture Coordinates (u, v)
Vertex Colors (r, g, b)
…
Vertex Normals (n_x, n_y, n_z)

Example 3D Model Data Format

- Wavefront 3D Graphics model description file with extension .obj
- Refer to http://en.wikipedia.org/wiki/Wavefront_.obj_file for detail file format
- Example
  - Model with colors (boxC.obj)
  - Model with normals (boxN.obj, boxN.mtl)
  - Model with textures (boxT.obj, boxT.mtl, tex1.tga, tex2.jpg)

Rasterization

Scan Converting Lines
Scan Converting Circles
Scan Converting Polygons

Scan Converting Lines

- Principles
  - The sequence of pixels should lie as close as possible to the ideal line
  - The scan converted line should be as straight as possible

Scan Converting Lines

- **Line Representations**
  - **Implicit Form**

    $$f(x, y) = ax + by + c = 0$$

  - **Explicit Form**

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

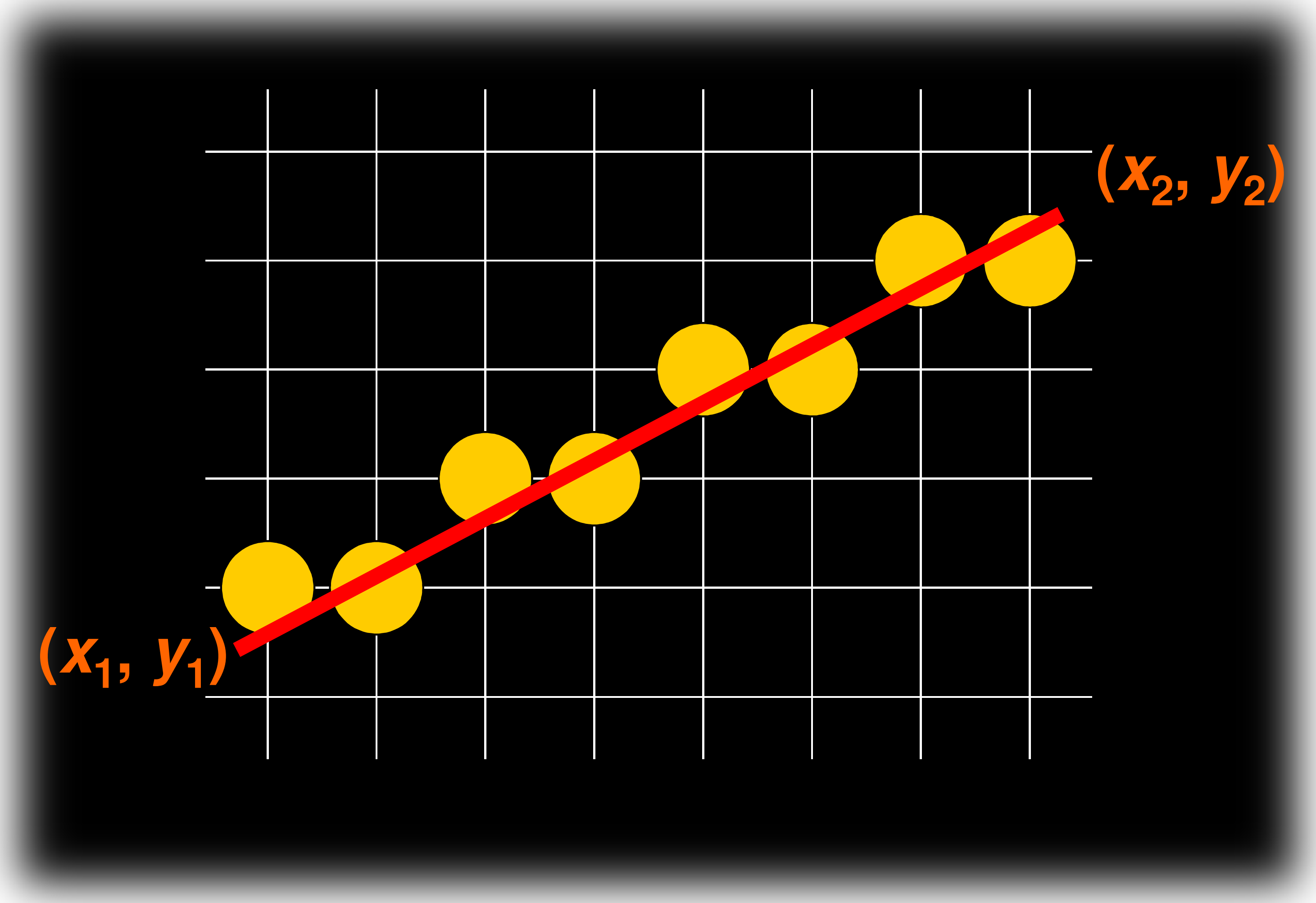
  - **Parametric Form**

    $$x = x(t) = a_0 t + b_0$$

    $$y = y(t) = a_1 t + b_1$$

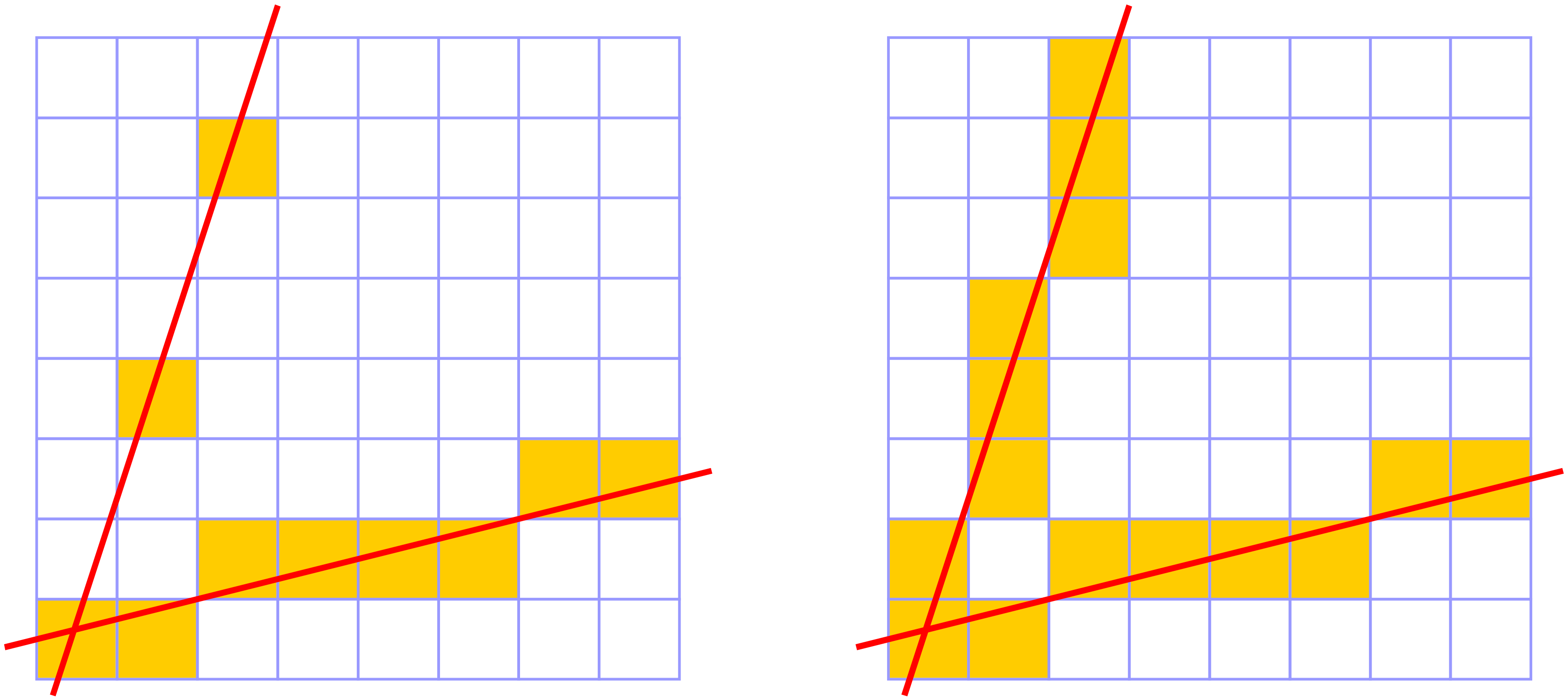
Scan Converting Lines

- Example: Draw from (x_1, y_1) to (x_2, y_2)



Scan Converting Lines

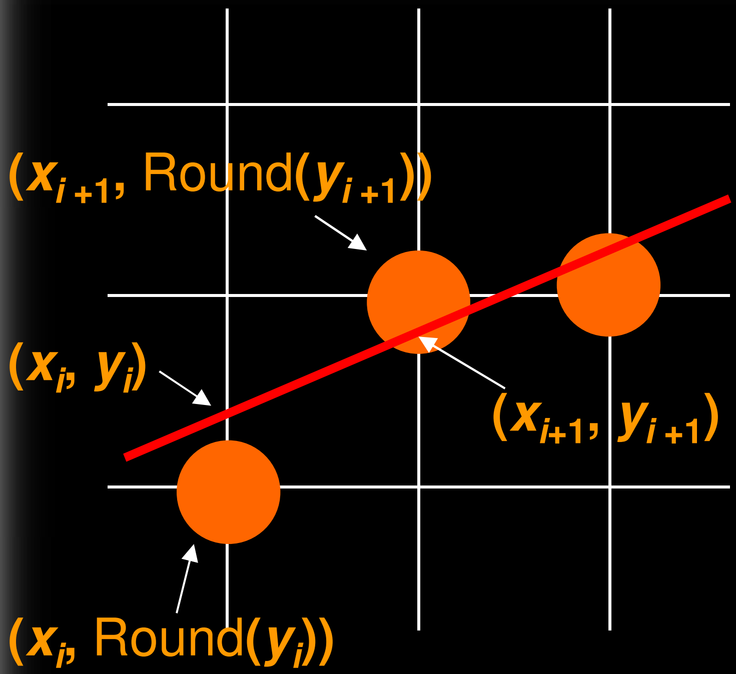
- **Quality Issues**



Gaps exist in slope > 1

Scan Converting Lines

- **Incremental Algorithm (DDA: Digital Differential Analyzer)**



$y = mx + B$, where $m = \Delta y/\Delta x$, and $0 \le m \le 1$

$y_i = mx_i + B$

draw pixel at $(x_i, \text{Round}(y_i))$

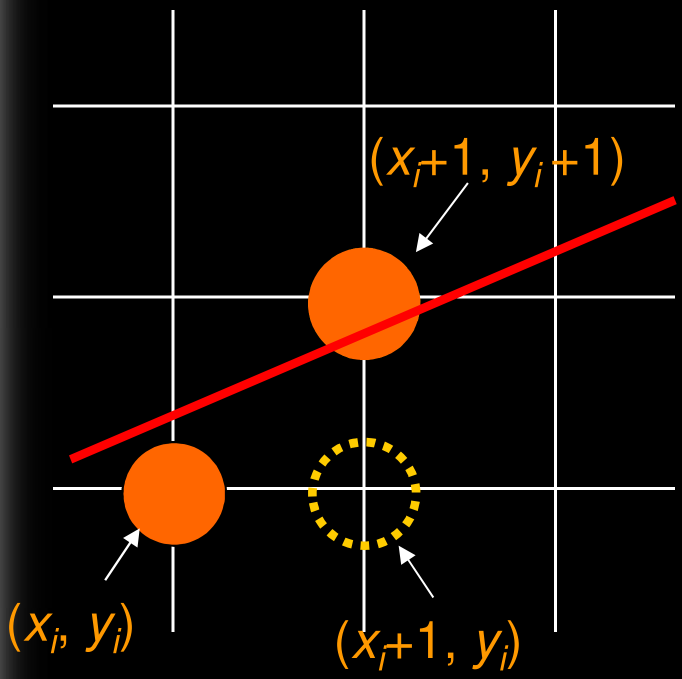
$x_{i+1} = x_i + 1$

$$
\begin{aligned}
y_{i+1} &= mx_{i+1} + B \\
&= m(x_i + 1) + B \\
&= y_i + m
\end{aligned}
$$

draw pixel at $(x_{i+1}, \text{Round}(y_{i+1}))$

Scan Converting Lines

- **Bresenham Algorithm**



$y = mx + B$, where $m = \Delta y/\Delta x$, and $0 \le m \le 1$

Current pixel : (x_i, y_i)

Next pixel : (x_i+1, y_i) or (x_i+1, y_i+1)

$d_1 = y - y_i = m(x_i+1) + B - y_i$

$d_2 = y_i + 1 - y = y_i + 1 - m(x_i+1) - B$

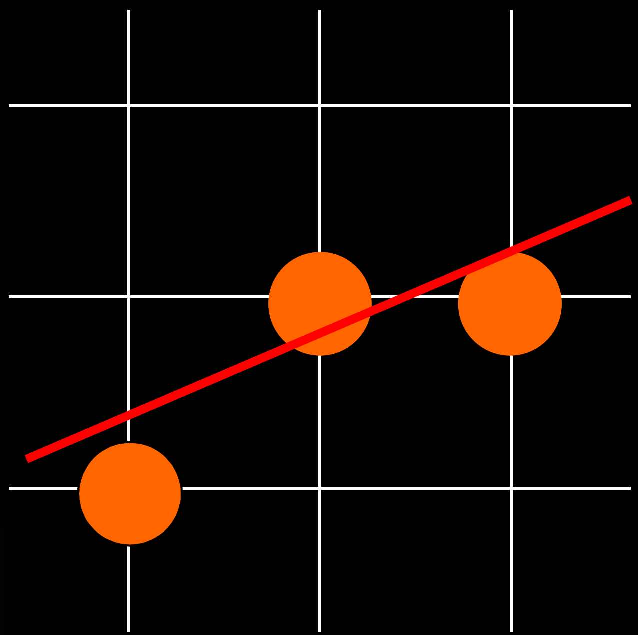
$d_1 - d_2 = 2m(x_i+1) - 2y_i + 2B - 1$

Let $P_i = \Delta x(d_1 - d_2) = 2\Delta y \cdot x_i - 2\Delta x \cdot y_i + C$
where $C = 2\Delta y + \Delta x(2B - 1)$

If $P_i < 0$ then choose (x_i+1, y_i)
otherwise, choose (x_i+1, y_i+1)

Scan Converting Lines

- **Incremental Computation of P_i**



$$P_{i+1} = 2\Delta y \cdot x_{i+1} - 2\Delta x \cdot y_{i+1} + C$$
$$= 2\Delta y \cdot (x_i+1) - 2\Delta x \cdot y_{i+1} + C$$
$$= P_i + 2\Delta y - 2\Delta x\,(y_{i+1} - y_i)$$

Incremental update :

When $P_i < 0$, $P_{i+1} = P_i + 2\Delta y$

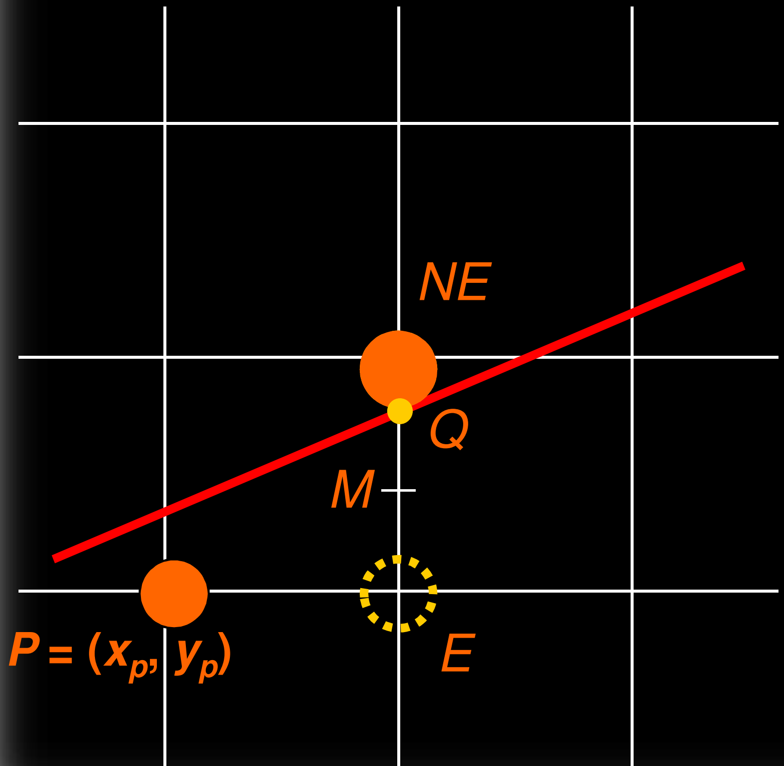
When $P_i \geq 0$, $P_{i+1} = P_i + 2\Delta y - 2\Delta x$

Initial Value :

$$P_1 = 2\Delta y - \Delta x$$

Scan Converting Lines

◆ Mid-Point Algorithm



$y = mx + B$, where $m = \Delta y/\Delta x$, and $0 \le m \le 1$

$F(x, y) = \Delta y \cdot x - \Delta x \cdot y + B \cdot \Delta x = 0$

let $a = \Delta y$, $b = -\Delta x$, and $c = B \cdot \Delta x$

Current pixel: (x_p, y_p)

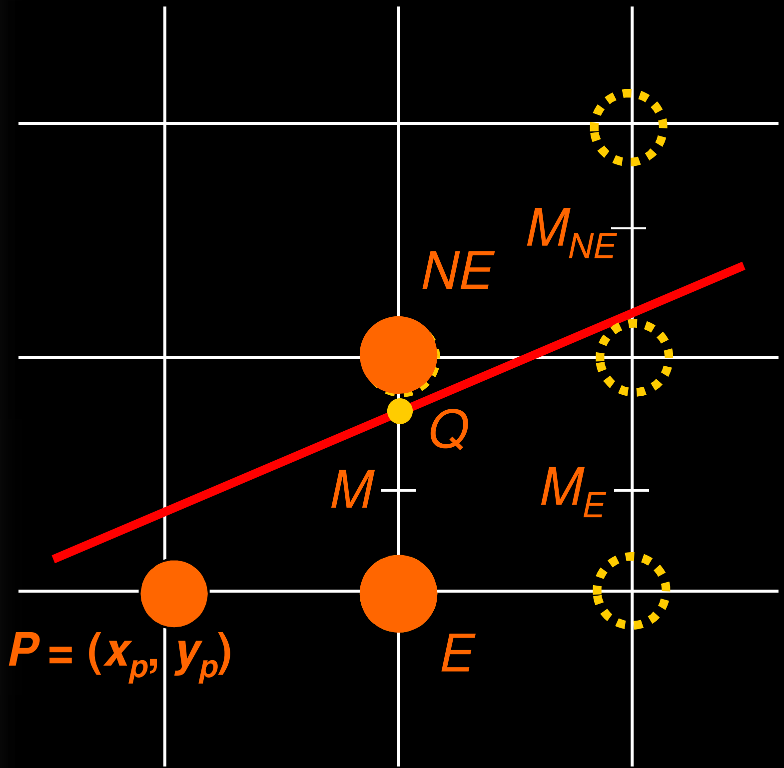
Let $d = F(M) = F(x_p+1, y_p+½)$

If $d > 0$, then choose pixel NE

If $d \le 0$, then choose pixel E

Scan Converting Lines

◆ Mid-Point Algorithm



$$d_{old} = F(x_p+1, y_p+½) = a(x_p+1) + b(y_p+½) + c$$

If E is chosen, then

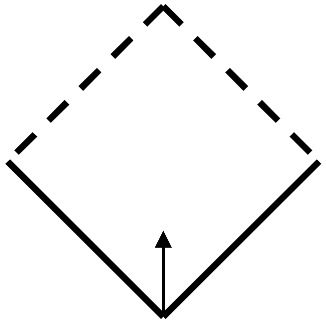
$$\begin{aligned} d_E &= F(x_p+2, y_p+½) \\ &= a(x_p+2) + b(y_p+½) + c \\ &= d_{old} + a \end{aligned}$$

If NE is chosen, then

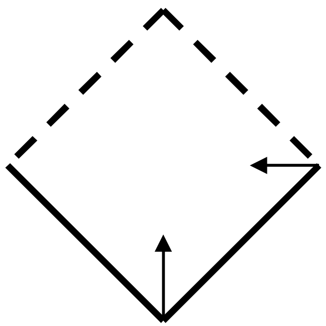
$$\begin{aligned} d_{NE} &= F(x_p+2, y_p+\tfrac{3}{2}) \\ &= a(x_p+2) + b(y_p+\tfrac{3}{2}) + c \\ &= d_{old} + a + b \end{aligned}$$

Scan Converting Lines

- Diamond Test Area
  - *x*-major lines (-1 ≤ slope ≤ 1)

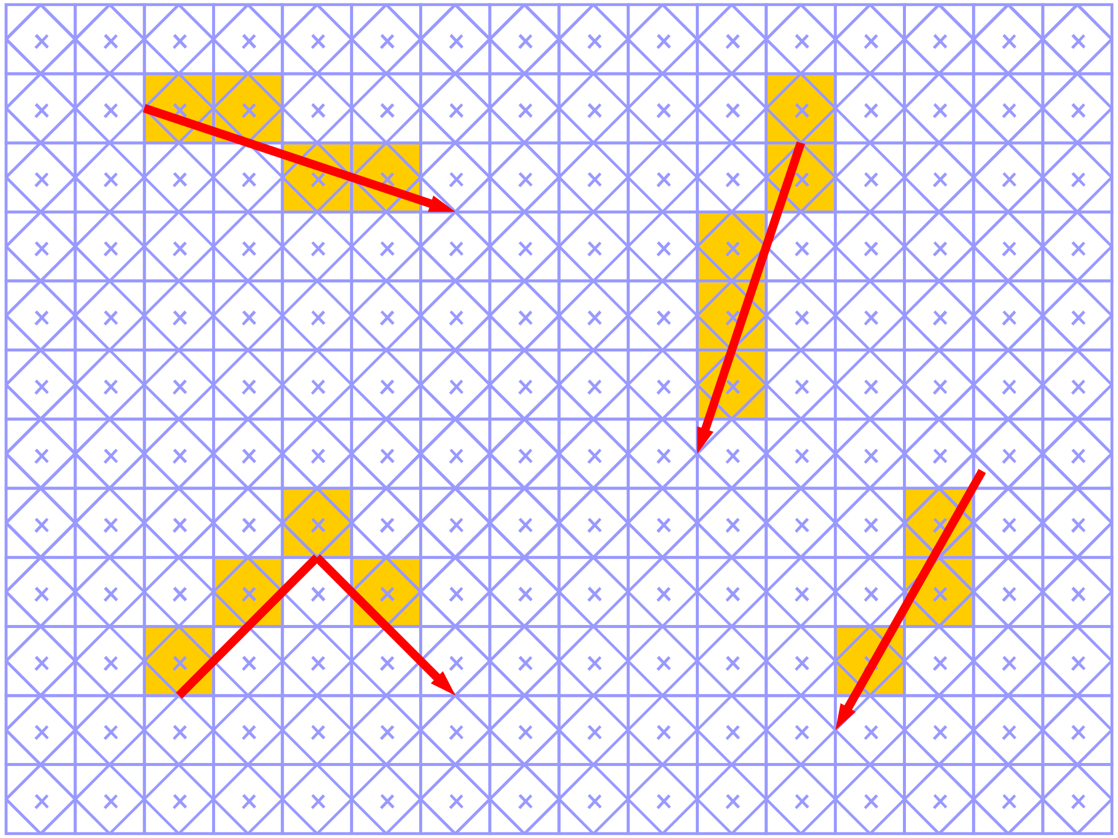


  - y-major lines (slope < -1 or slope > 1)



Scan Converting Lines

- **Diamond Check**



Issues on Scan Converting Lines

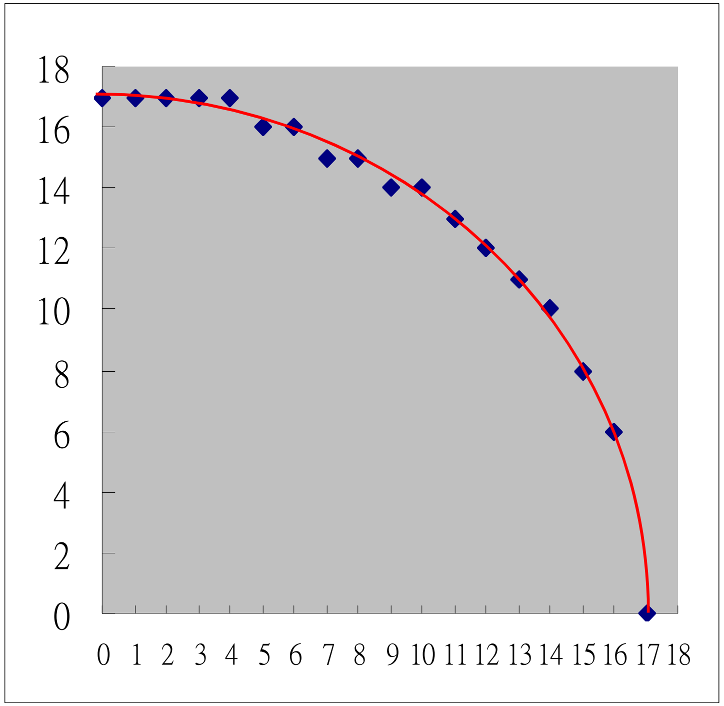
- Precision
  - Fixed-point (e.g. s.7.16)
  - Floating-point (e.g. s.[8].23)
- Endpoint Order
  - Left to right or right to left
- Aliasing
  - Coverage of a line

Scan Converting Circles

- **Method I**

  $x^2 + y^2 = R^2$

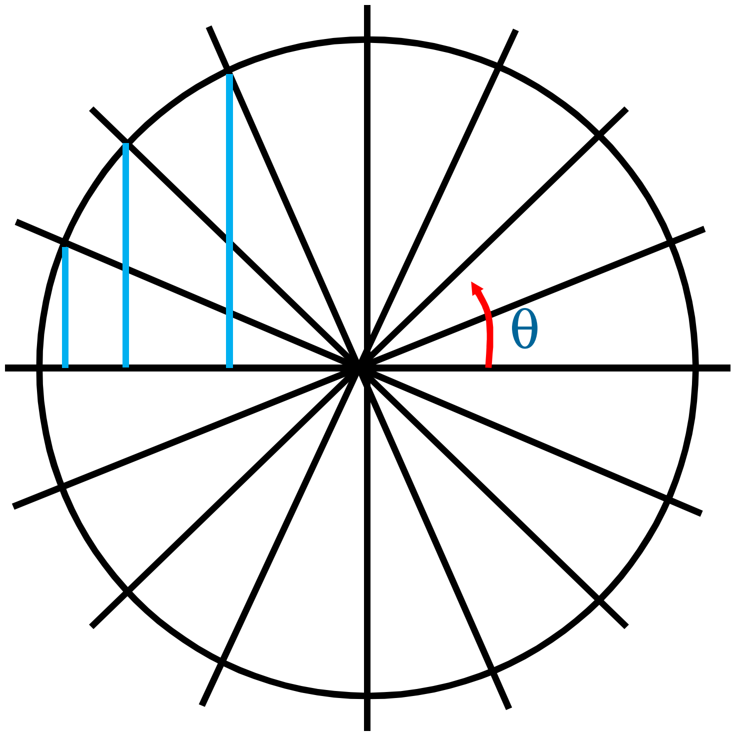
  $y = \pm\, \mathrm{sqrt}(R^2 - x^2),\ x = 0, 1, \ldots, R$



Scan Converting Circles

- **Method II**

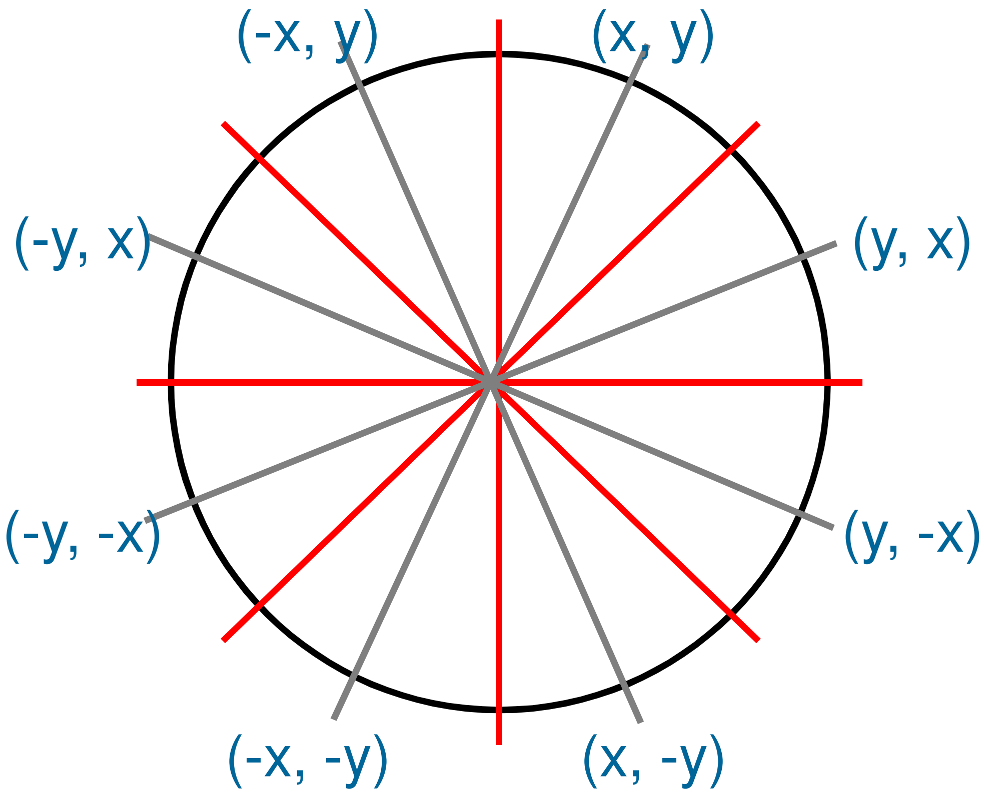
  $(\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{r}\cos\theta, \boldsymbol{r}\sin\theta), \theta = 0^\circ, \ldots, 360^\circ$



Scan Converting Circles

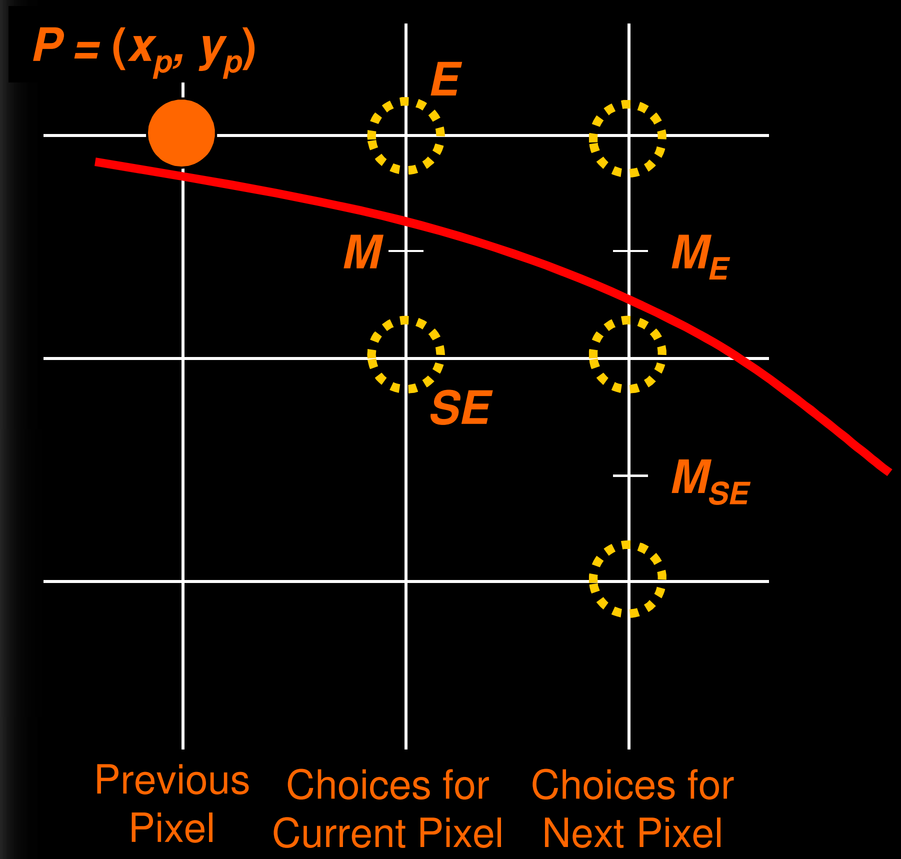
- **Method III**

  Eight-Way Symmetry



Scan Converting Circles

◆ Midpoint Circle Algorithm



Let $F(x, y) = x^2 + y^2 - R^2$

Current Pixel: (x_p, y_p)

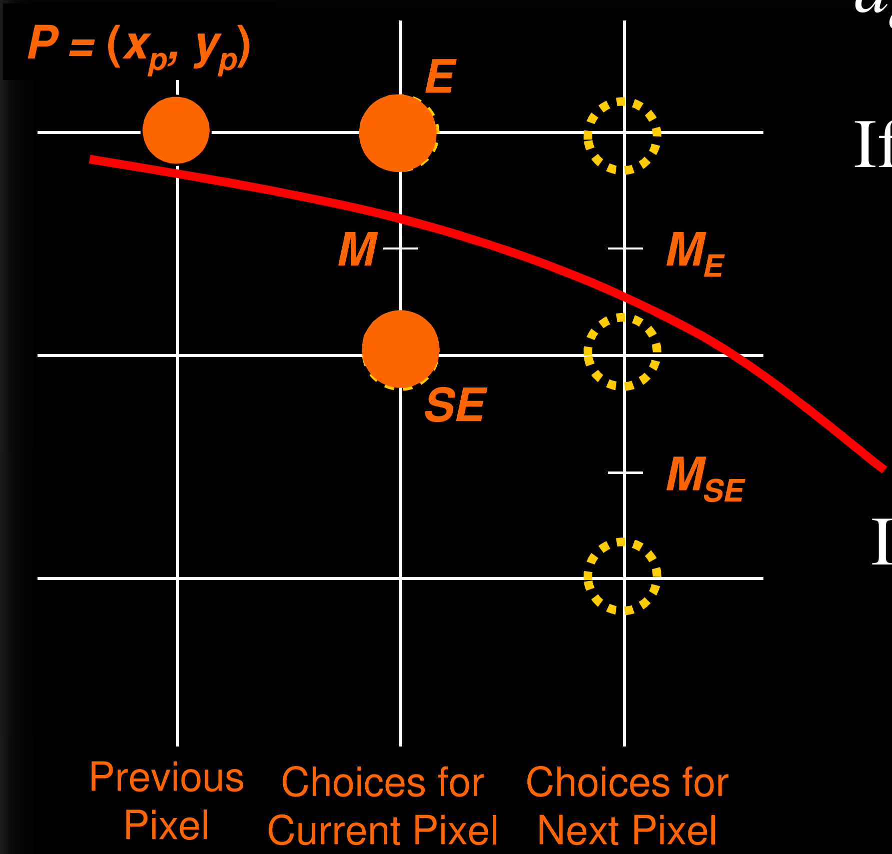
Let $d = F(M) = F(x_p+1, y_p-½)$

If $d \geq 0$, then choose pixel SE

If $d < 0$, then choose pixel E

Scan Converting Circles

◆ Midpoint Circle Algorithm



$$d_{old} = F(x_p+1, y_p-\tfrac{1}{2}) = (x_p+1)^2 + (y_p-\tfrac{1}{2})^2 - R^2$$

If E is chosen, then

$$\begin{aligned} d_E &= F(x_p+2, y_p-\tfrac{1}{2}) \\ &= (x_p+2)^2 + (y_p-\tfrac{1}{2})^2 - R^2 \\ &= d_{old} + (2x_p + 3) \end{aligned}$$

If SE is chosen, then

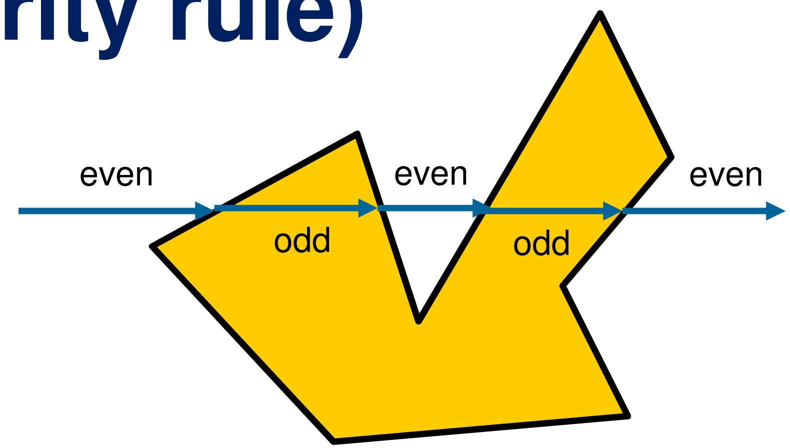
$$\begin{aligned} d_{SE} &= F(x_p+2, y_p-\tfrac{3}{2}) \\ &= (x_p+2)^2 + (y_p-\tfrac{3}{2})^2 - R^2 \\ &= d_{old} + (2x_p - 2y_p + 5) \end{aligned}$$

Rectangle Fill

- Determine which pixels to fill
  - Taking successive scan lines that intersect the rectangle
  - Filling in spans of adjacent pixels that lie inside the rectangle from left to right

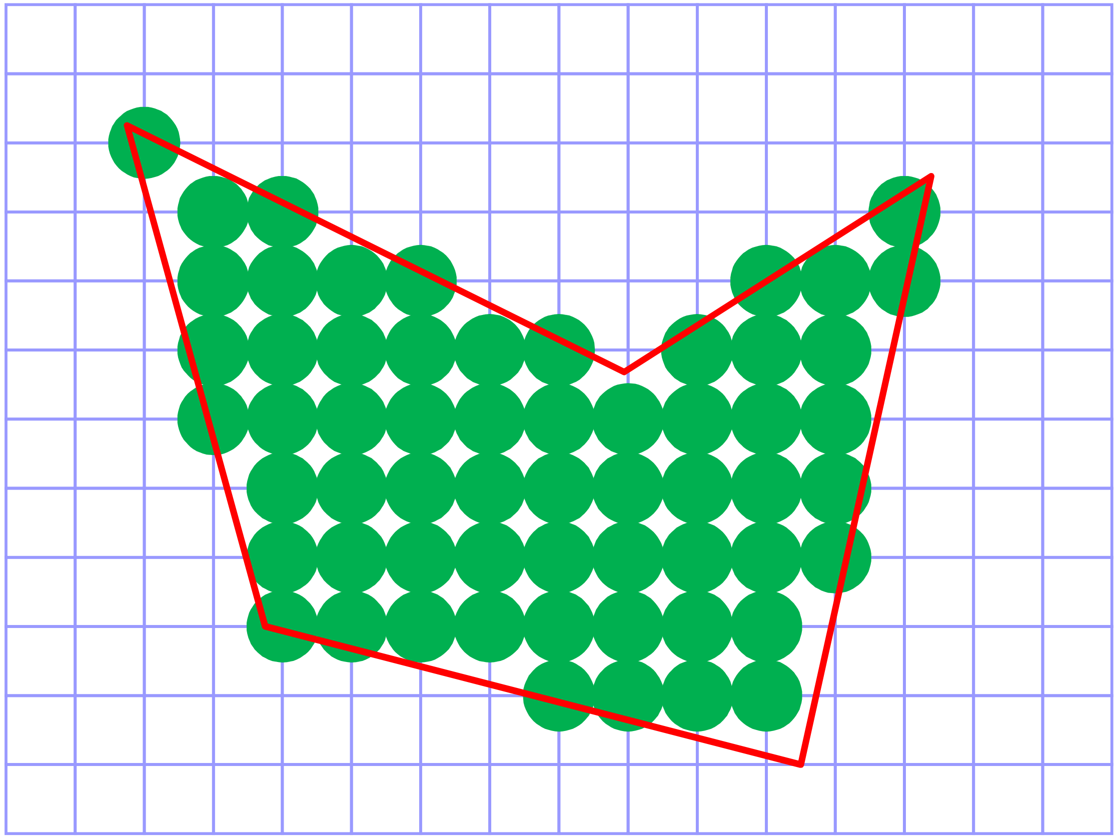
Polygon Fill

- Find the intersections of the scan line with all edges of the polygon
- Sort the intersections by increasing the x coordinate
- Fill in all pixels between pairs of intersections that lie inside the polygon (using odd-parity rule)



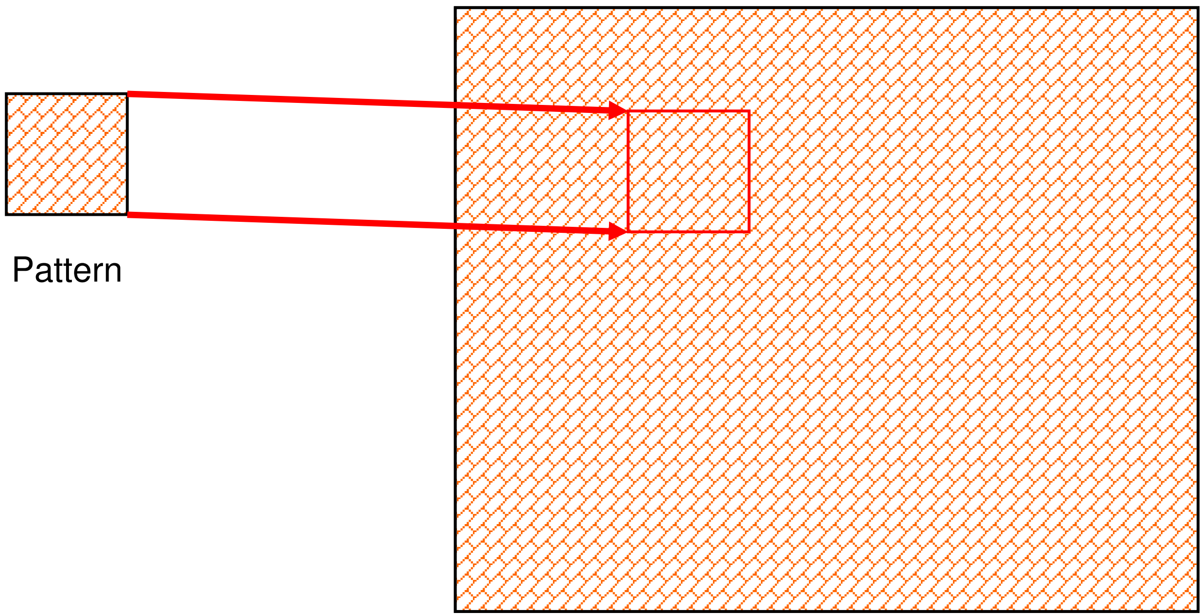
Polygon Fill

- Example



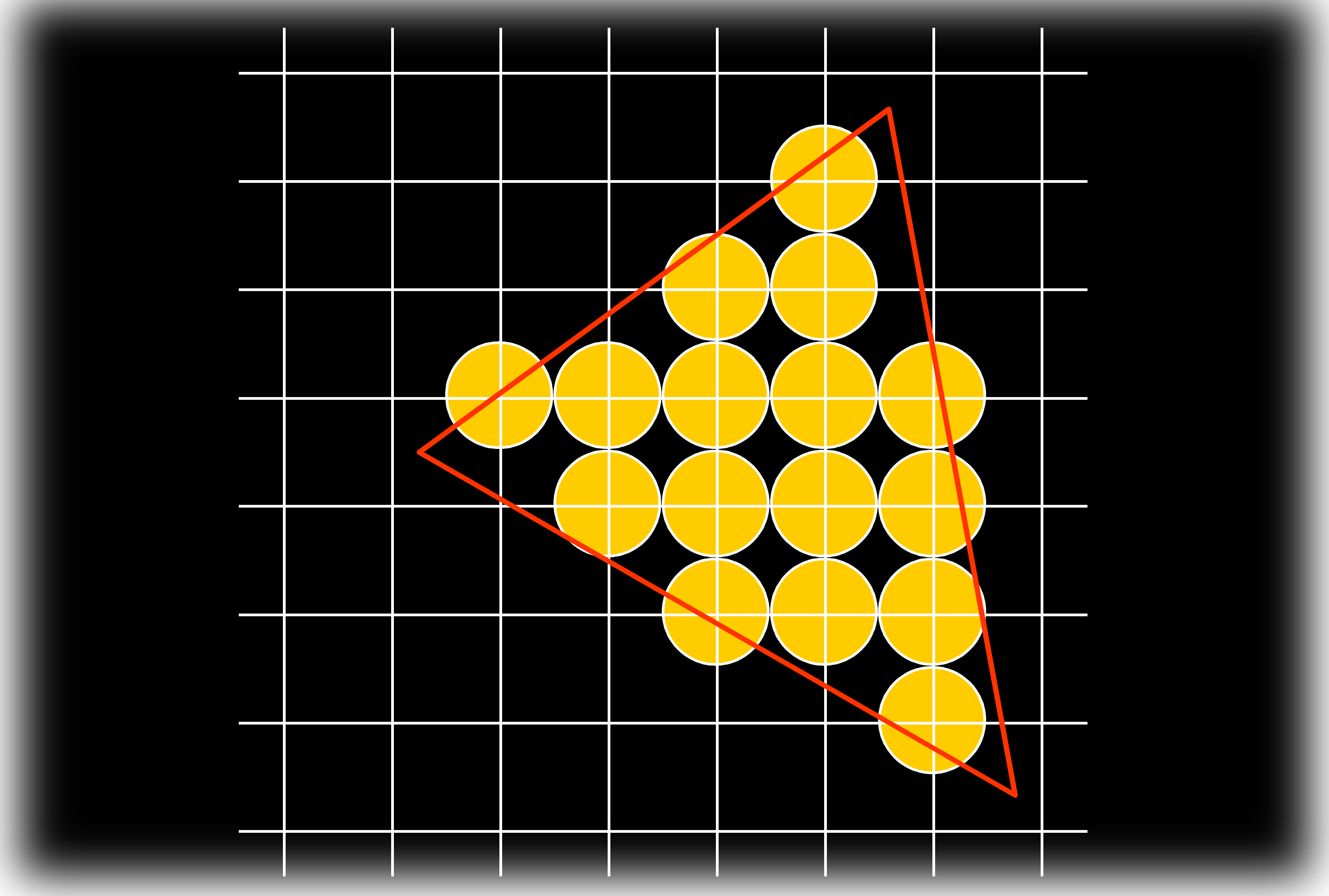
Pattern Fill

- **A pattern map is retrieved and mapped onto the destination primitive**



Triangle Fill

- Generate Interior Pixels of a Triangle



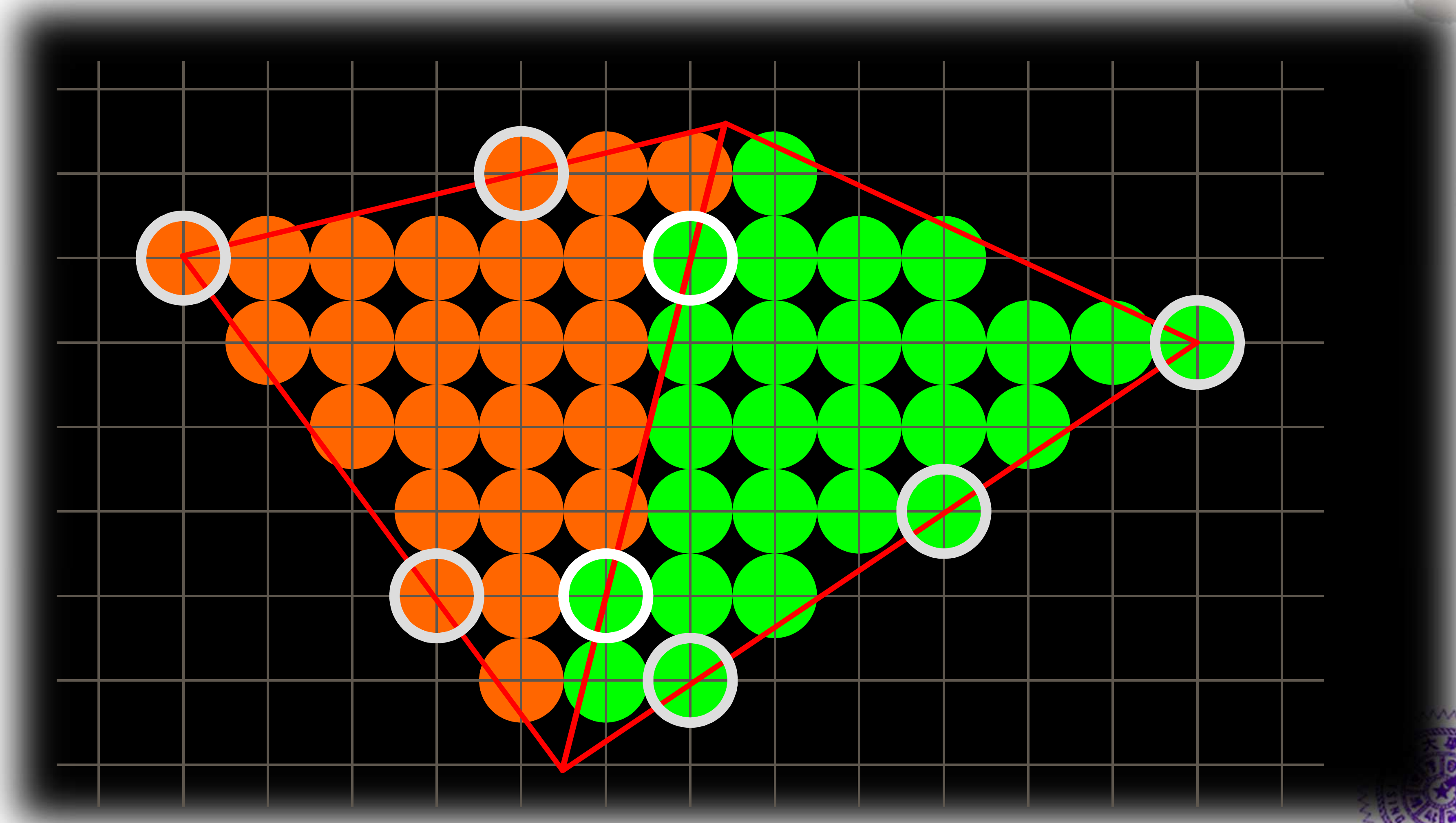
Triangle Rasterization

Triangle Rasterization

Triangle Rasterization Rule

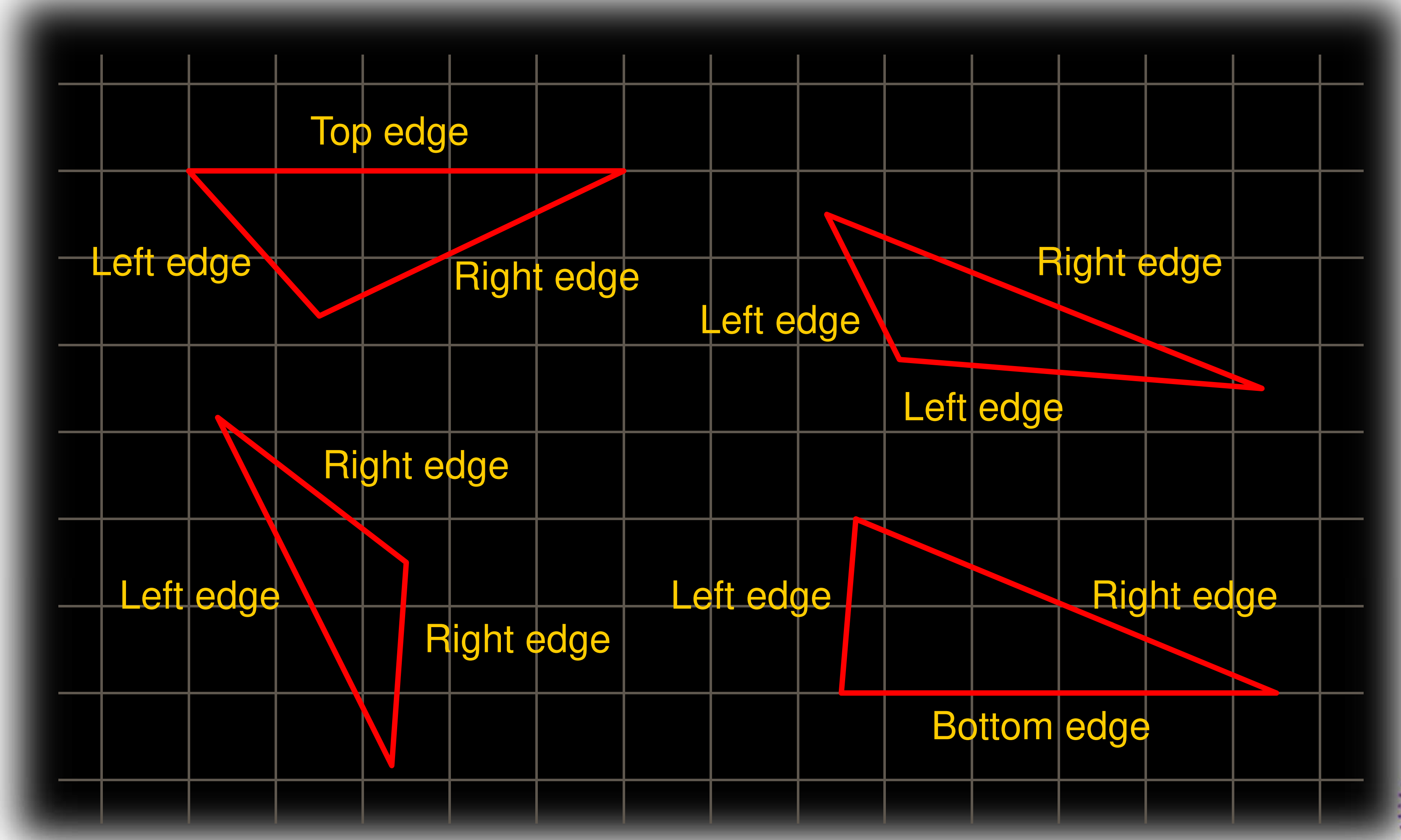
Triangle Rasterization

- Scan Line Conversion



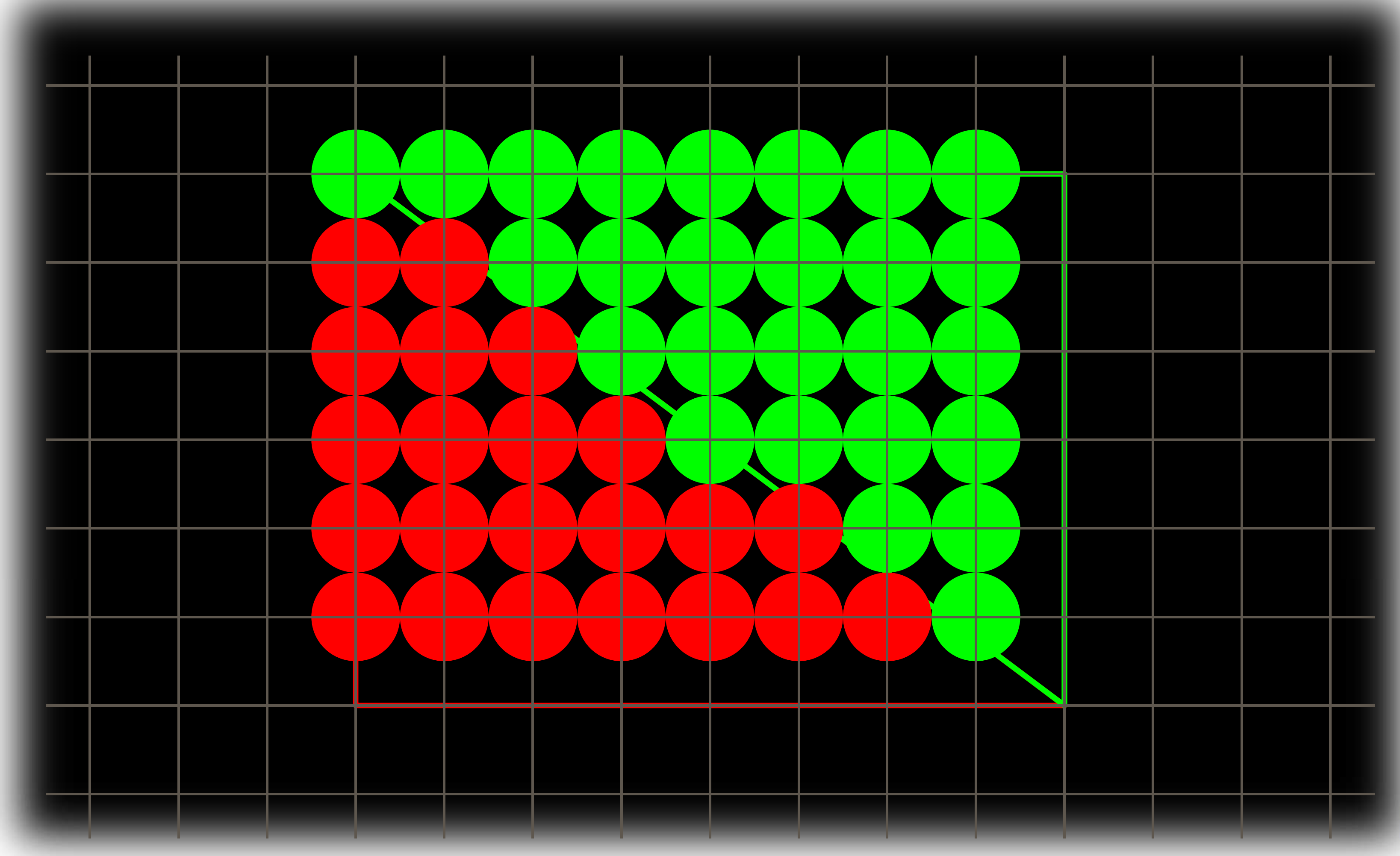
Triangle Rasterization Rule

- **Top-Left Filling Convention**



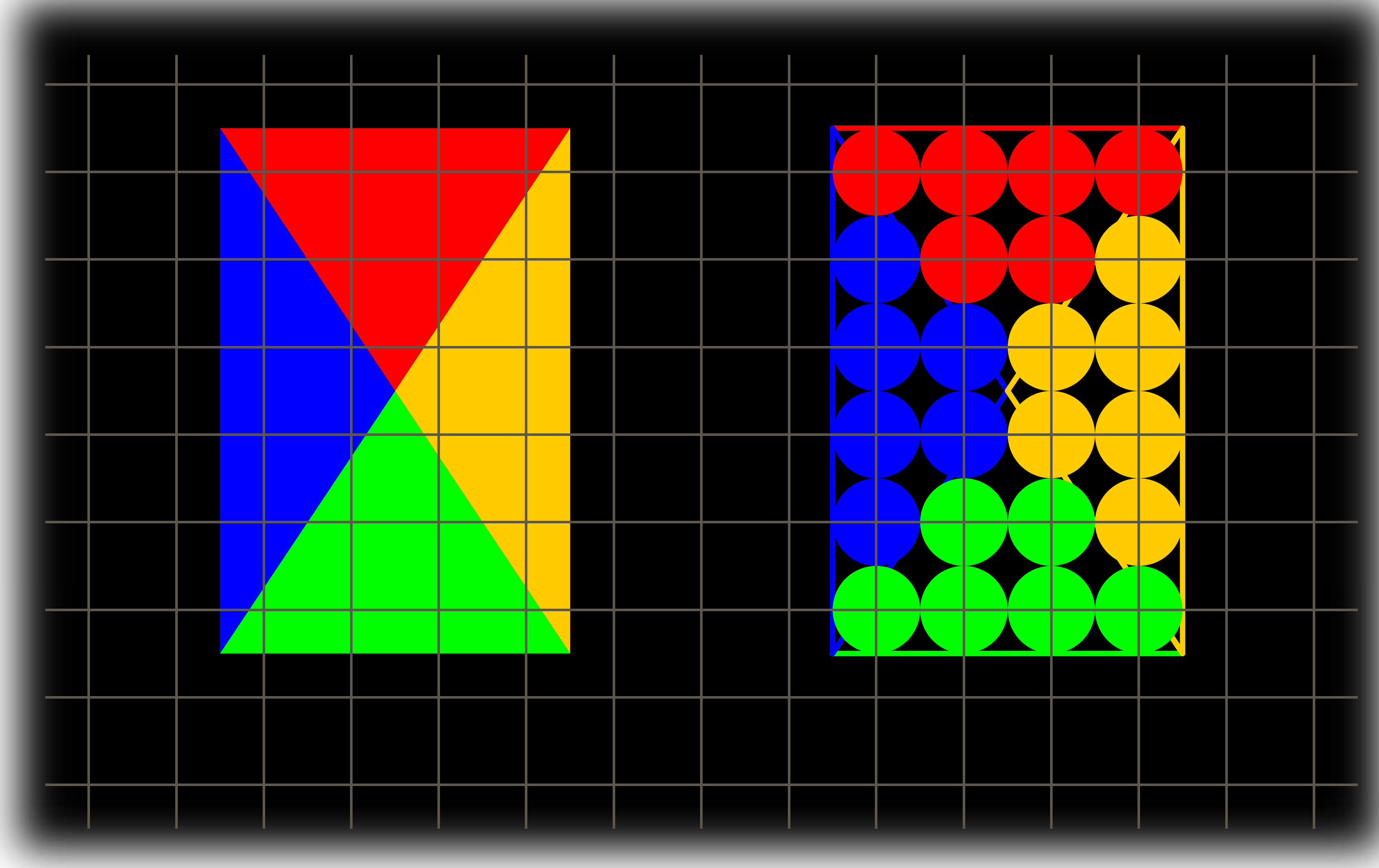
Triangle Rasterization Rule

- Top-Left Filling Convention



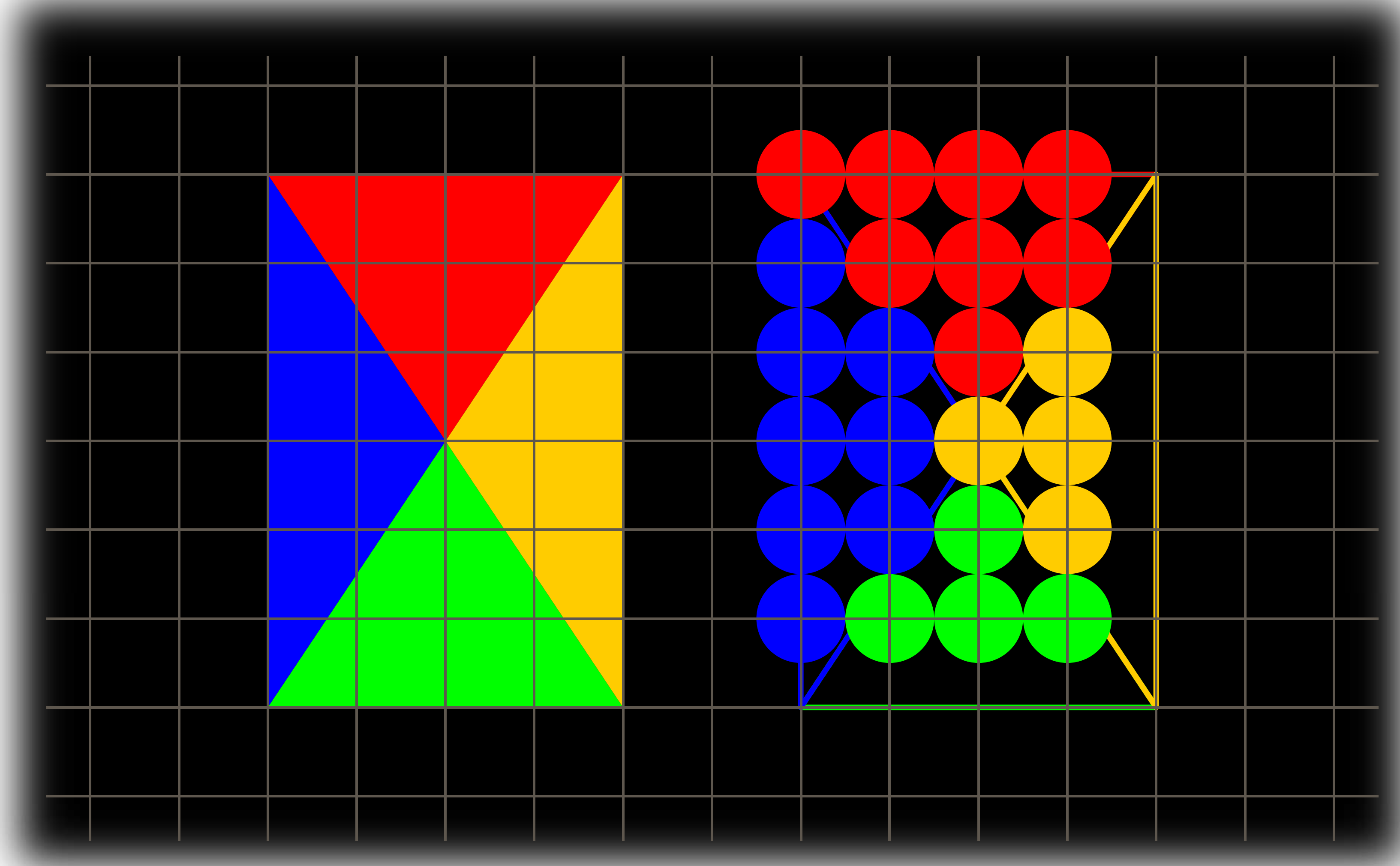
Triangle Rasterization Rule

- **Top-Left Filling Convention**



Triangle Rasterization Rule

- Top-Left Filling Convention



Triangle Rasterization Rule

- Sorting is required to scan covert from top to bottom
- For each scan line, scan convert from left to right
- Mid-point check for not over shooting and switch to right slope

Attribute Derivation inside a Triangle

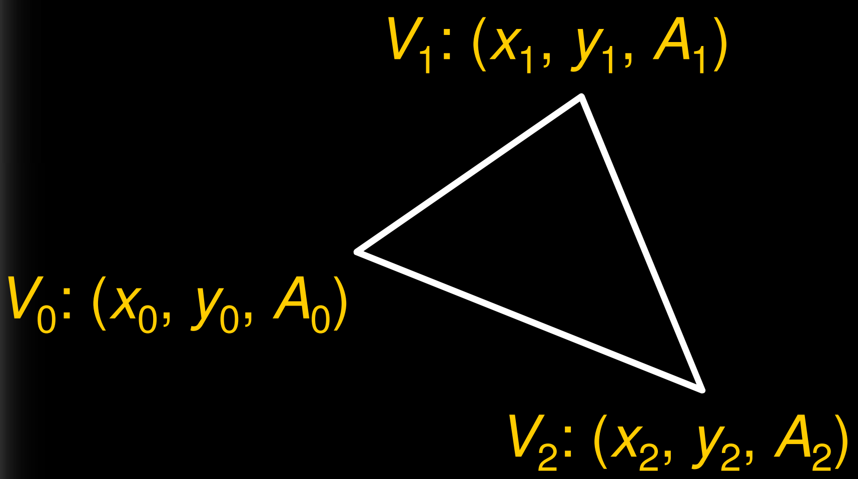
Color

Depth

Texture Coordinates

Attribute Derivation

- **Plane Equation**



$$f(x, y, A) = ax + by + cA + d = 0$$

$$(a, b, c) = \overrightarrow{V_0V_1} \times \overrightarrow{V_0V_2}$$

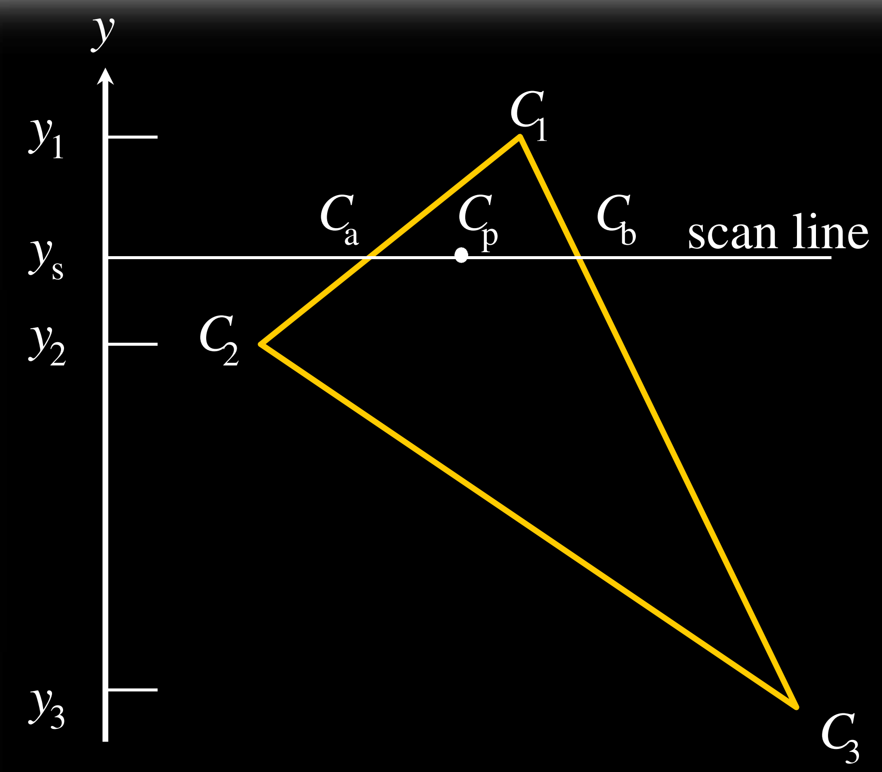
d is a constant

$$A = f(x, y)$$
$$= -(a/c)x - (b/c)y - (d/c)$$

$$A_i = f(x_i, y_i)$$

Attribute Derivation

- **Interpolation**



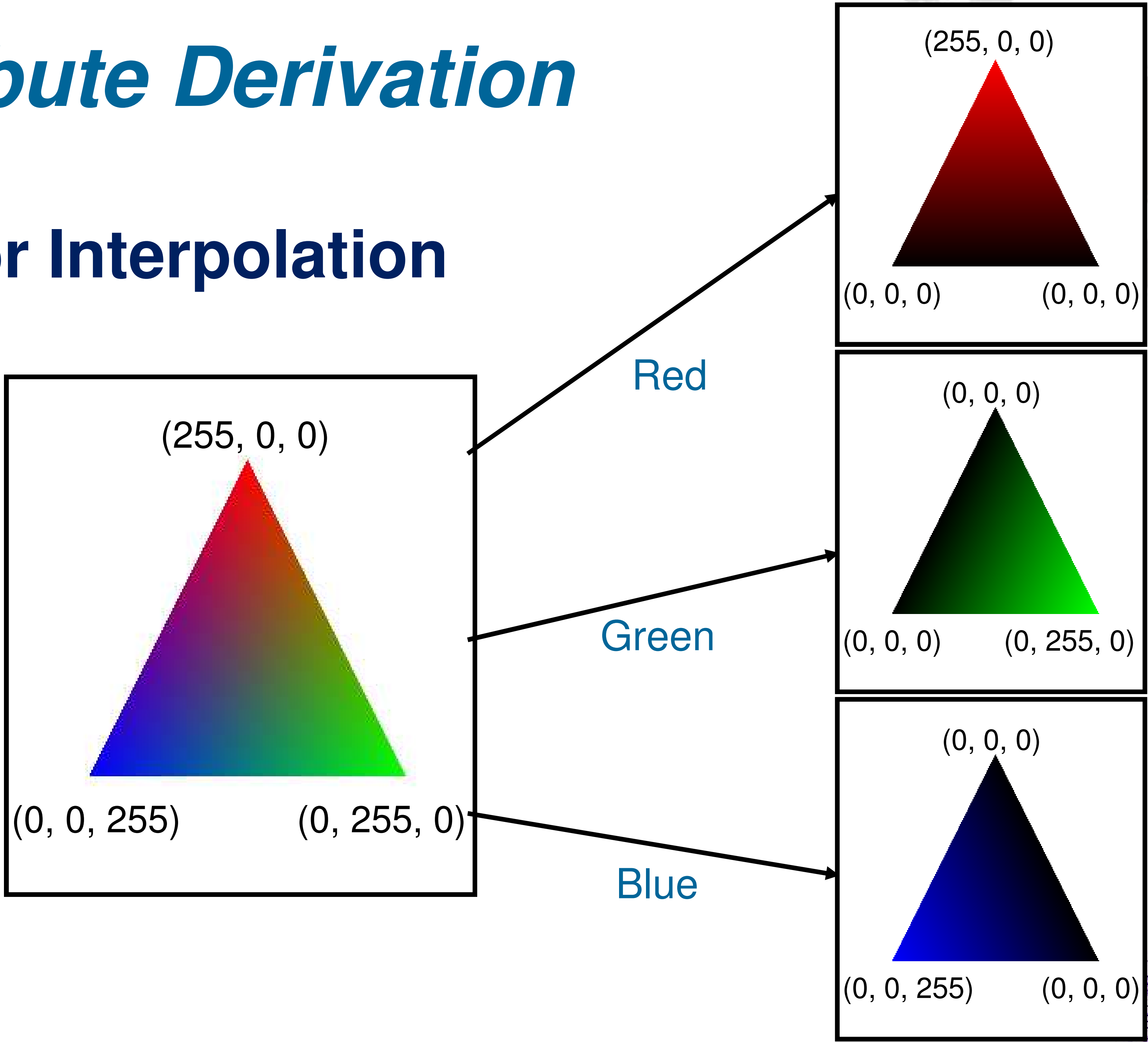
$$C_a = C_1 - (C_1 - C_2)\frac{y_1 - y_s}{y_1 - y_2}$$

$$C_b = C_1 - (C_1 - C_3)\frac{y_1 - y_s}{y_1 - y_3}$$

$$C_p = C_a - (C_a - C_b)\frac{x_a - x_p}{x_a - x_b}$$

Attribute Derivation

- **Color Interpolation**

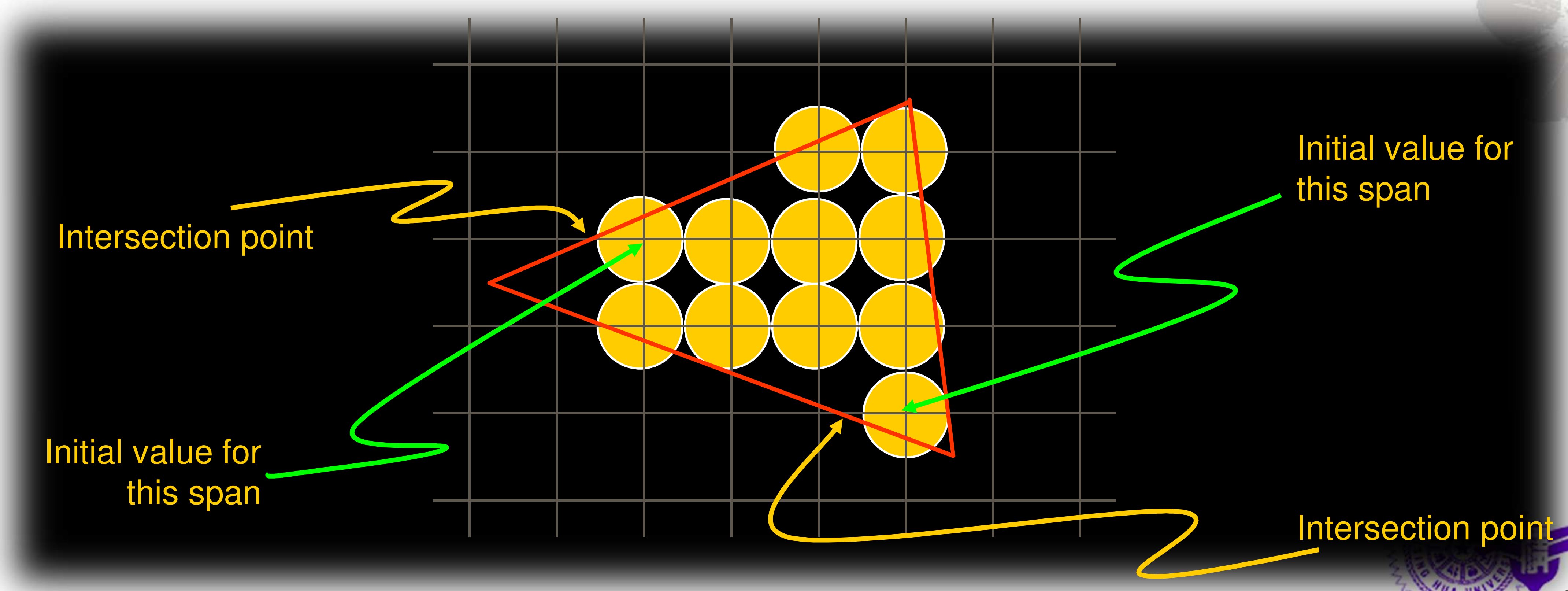


Attribute Derivation

- **Advantage of Interpolation**
  - **Can apply the concept of DDA to reduce computation cost**

Attribute Derivation

- Disadvantage of Interpolation using DDA
  - Need to derive the initial value for each span
  - Precision error will accumulate

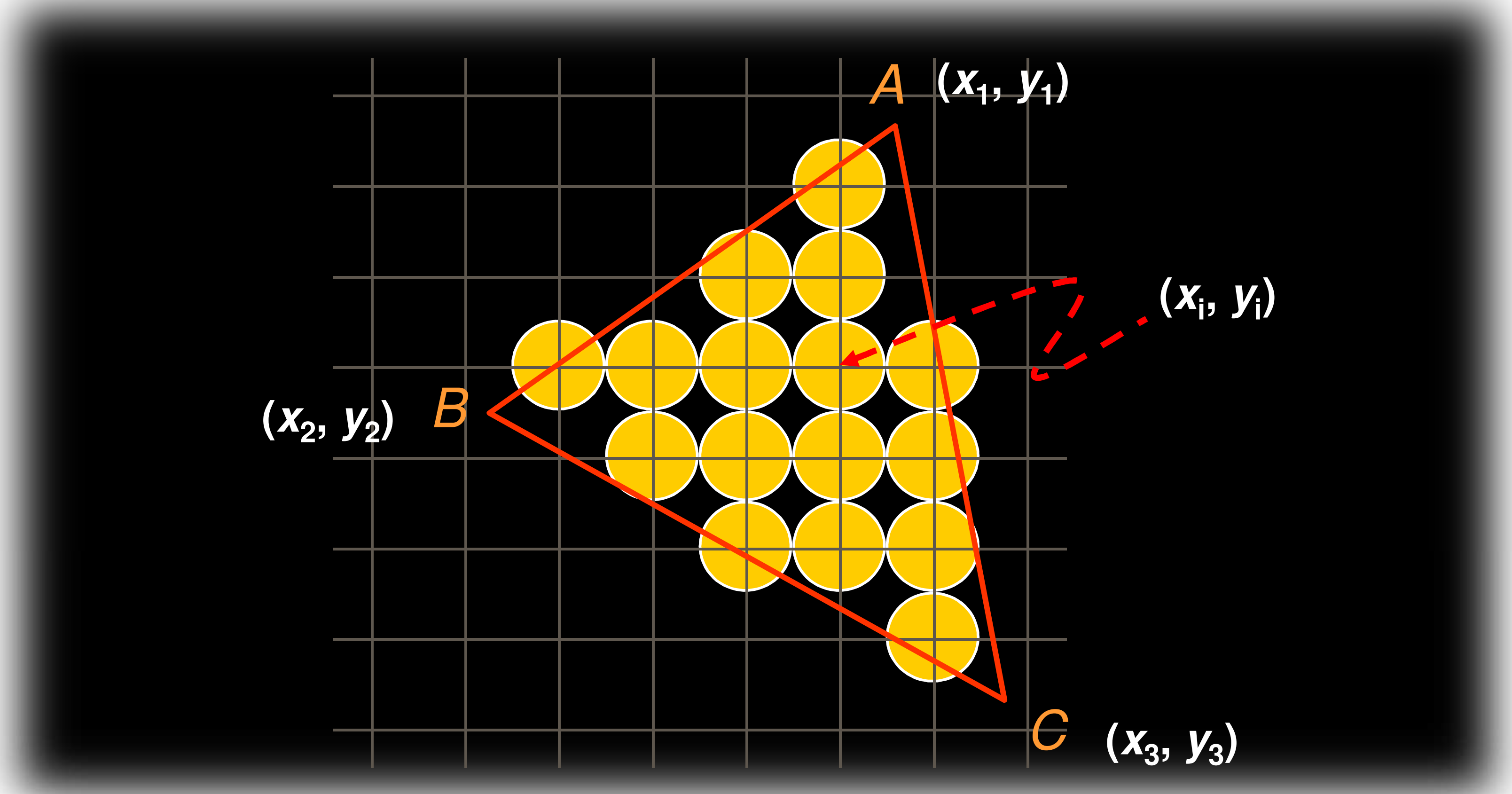


Setup for Coordinate Rasterization

Screen Coordinates

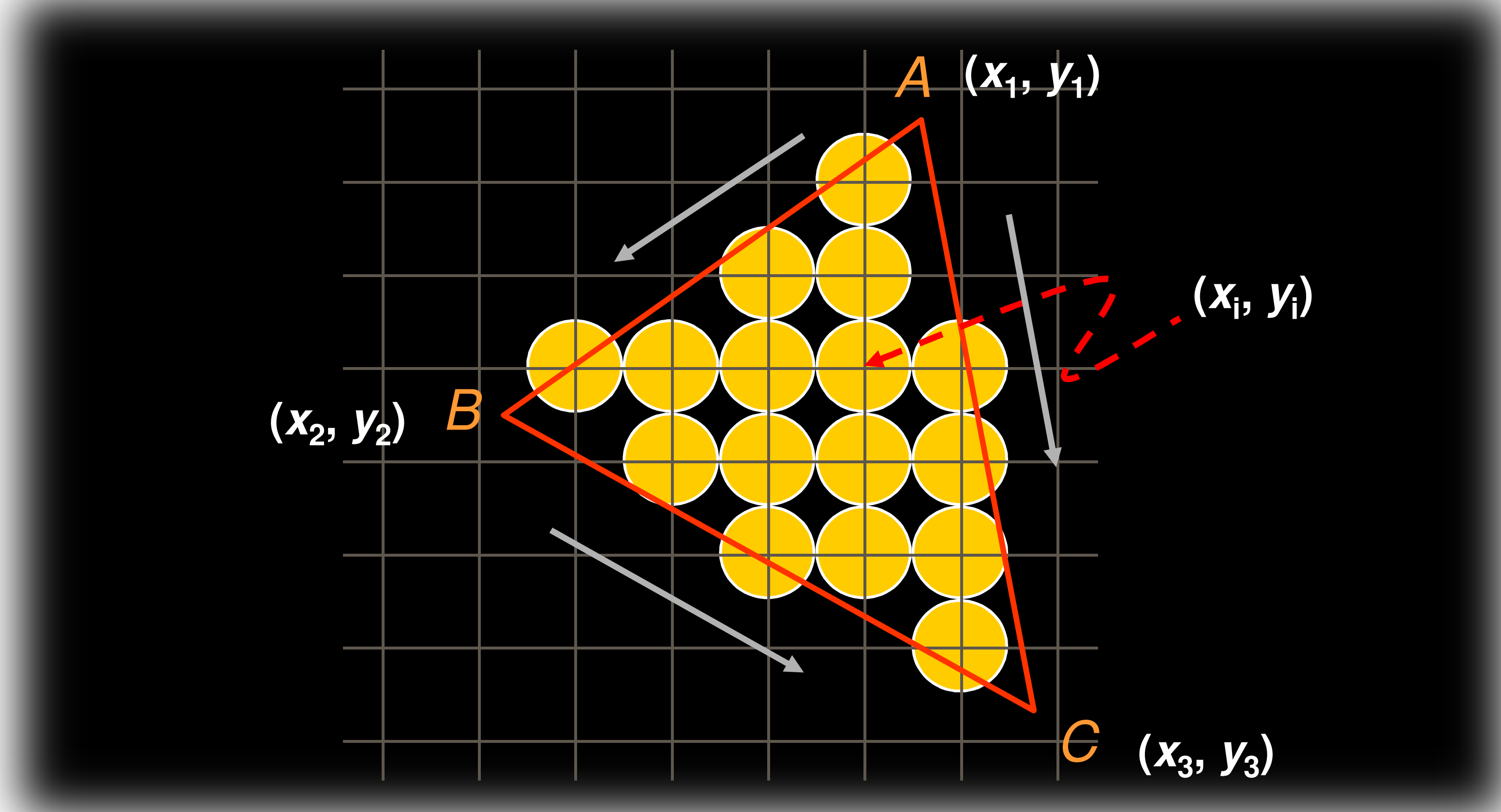
Screen Coordinate Setup

- **Initial setup for triangle rasterization**
  - **Use to generate digitized (x, y) coordinates inside a triangle**



Screen Coordinate Setup

- **Initial setup for triangle rasterization**
  - **Calculate gradients, $\Delta x/\Delta y$ for lines *AB*, *BC*, and *AC***



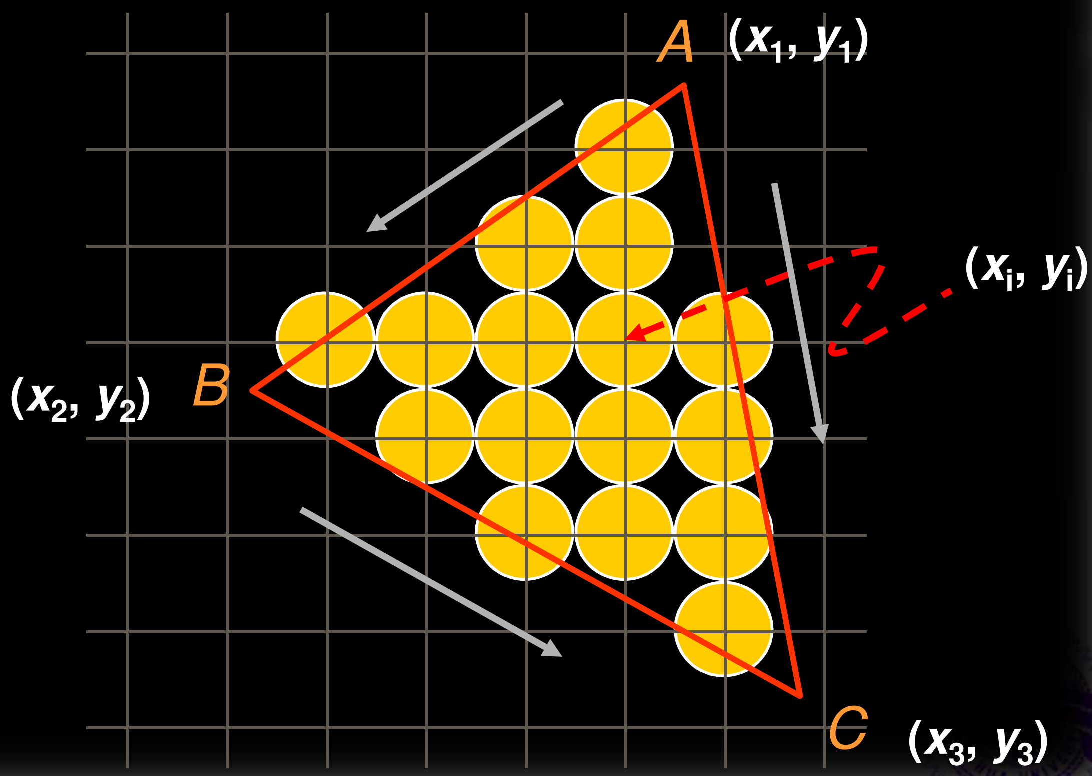
Screen Coordinate Setup

- **Initial setup for triangle rasterization**
  - **Calculate gradients, $\Delta x/\Delta y$ for lines *AB*, *BC*, and *AC***

$$\text{Line AB}: \Delta x / \Delta y = \frac{x_2 - x_1}{y_2 - y_1}$$

$$\text{Line BC}: \Delta x / \Delta y = \frac{x_3 - x_2}{y_3 - y_2}$$

$$\text{Line AC}: \Delta x / \Delta y = \frac{x_3 - x_1}{y_3 - y_1}$$



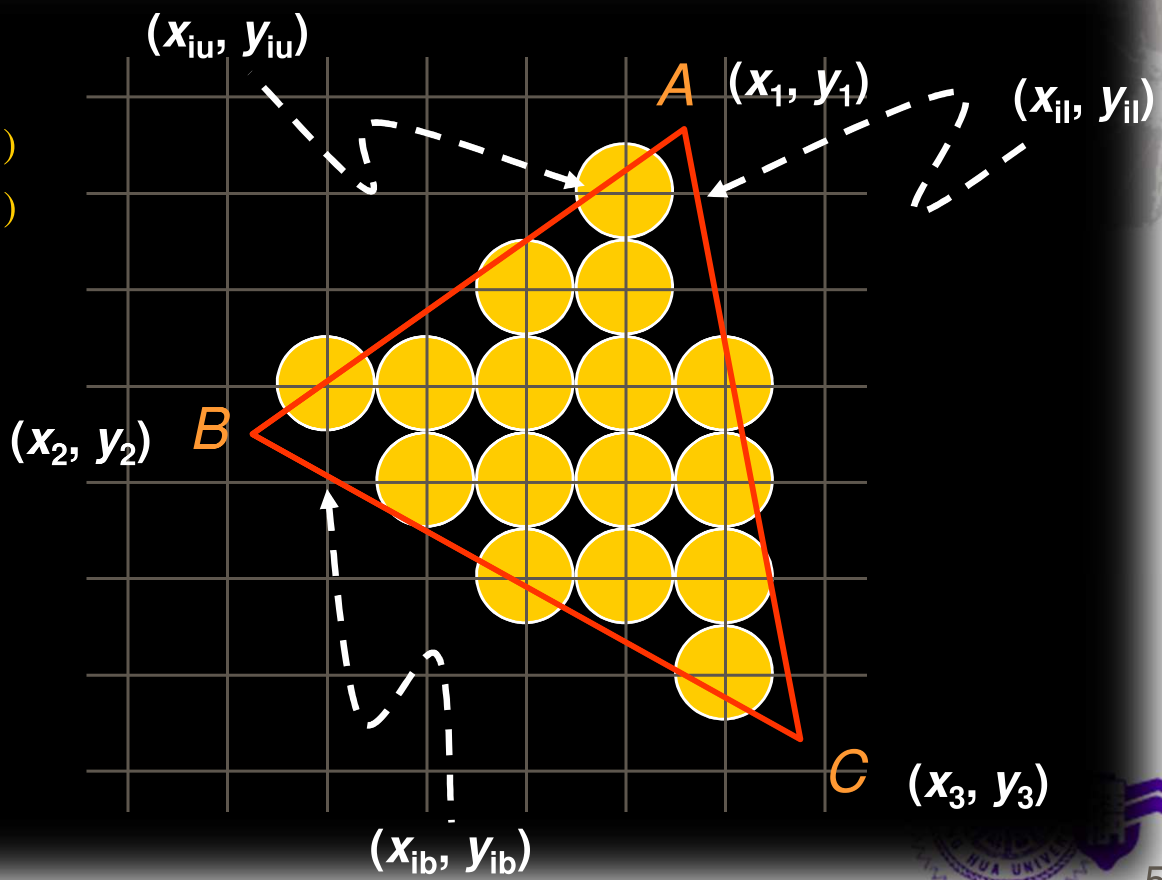
Screen Coordinate Setup

- **Initial setup for triangle rasterization**
  - **Y intersections for the line segments**

$$x_{iu} = x_1 + \frac{x_2 - x_1}{y_2 - y_1}(y_{iu} - y_1),\; y_{iu} = \begin{cases} y_1 & \text{if } y_1 = \text{int}(y_1) \\ \text{int}(y_1) + 1 & \text{if } y_1 \neq \text{int}(y_1) \end{cases}$$

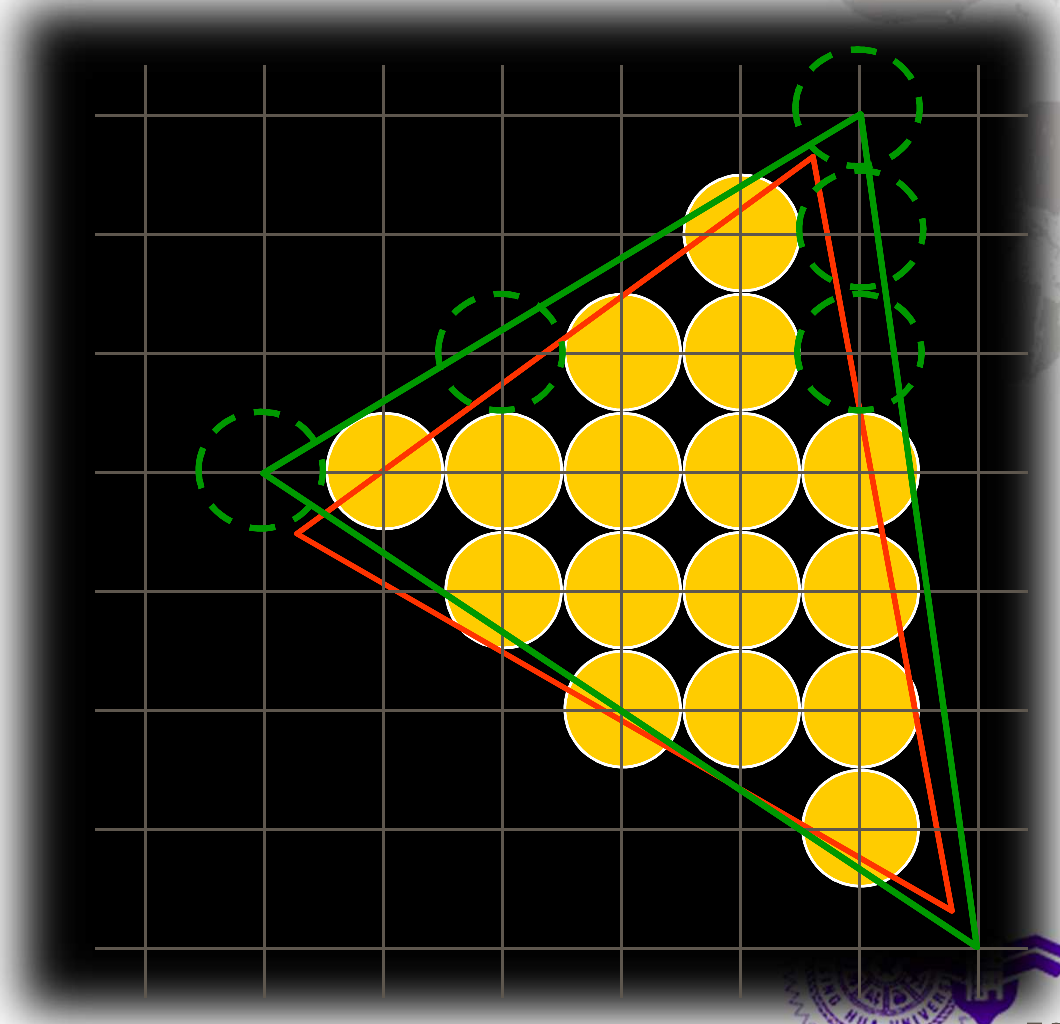
$$x_{ib} = x_2 + \frac{x_3 - x_2}{y_3 - y_2}(y_{ib} - y_2),\; y_{ib} = \begin{cases} y_2 & \text{if } y_2 = \text{int}(y_2) \\ \text{int}(y_2) + 1 & \text{if } y_2 \neq \text{int}(y_2) \end{cases}$$

$$x_{il} = x_1 + \frac{x_3 - x_1}{y_3 - y_1}(y_{il} - y_1),\; y_{il} = \begin{cases} y_1 & \text{if } y_1 = \text{int}(y_1) \\ \text{int}(y_1) + 1 & \text{if } y_1 \neq \text{int}(y_1) \end{cases}$$



Screen Coordinate Setup

- Adjustment to the vertex position might cause
  - Distortion
  - Draw different pixels than the pixels without vertex adjustment

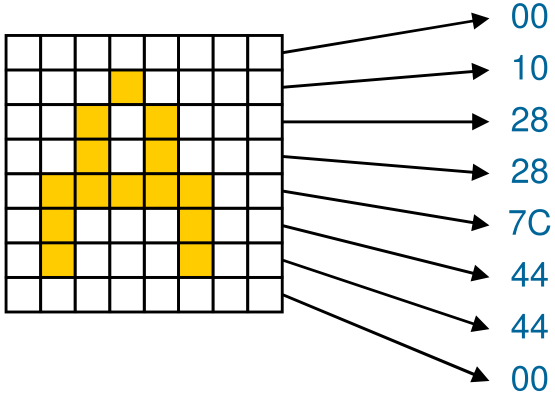


Other 2D Graphics Operations

Text Display
Block Copy
Raster Operations

2D Text

- Use rectangle fill with character bitmap pattern

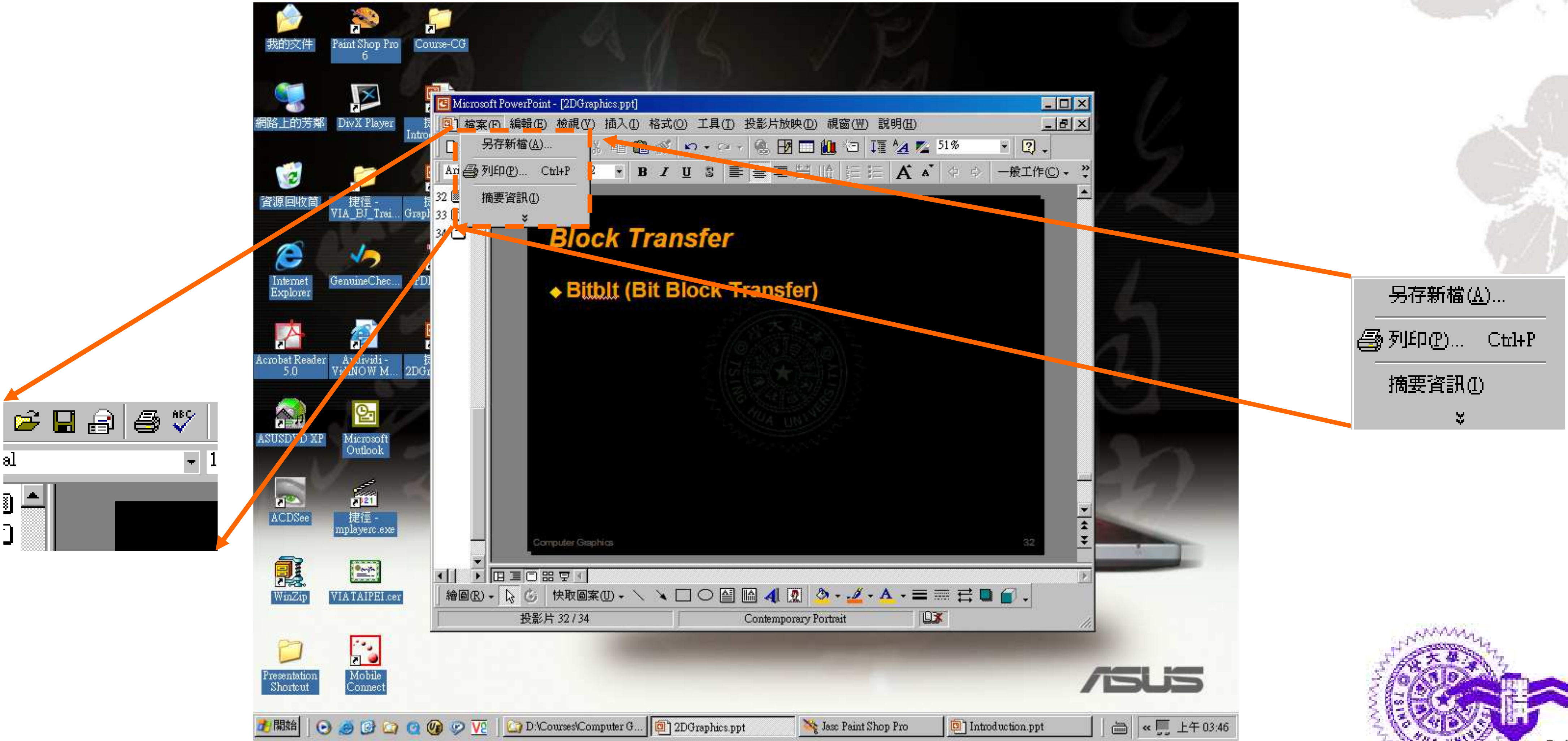


ABCDEFGHIJKLMN
OPQRSTUVWXYZ
abcdefghijklmnopqr
stuvwxyz

- Attributes
  - Font (Time Roman, `Courier`, Arial)
  - Appearance (Normal, **Bold**, *Italic*, Underline)
  - Size (points)
  - Constant/Proportional Spacing

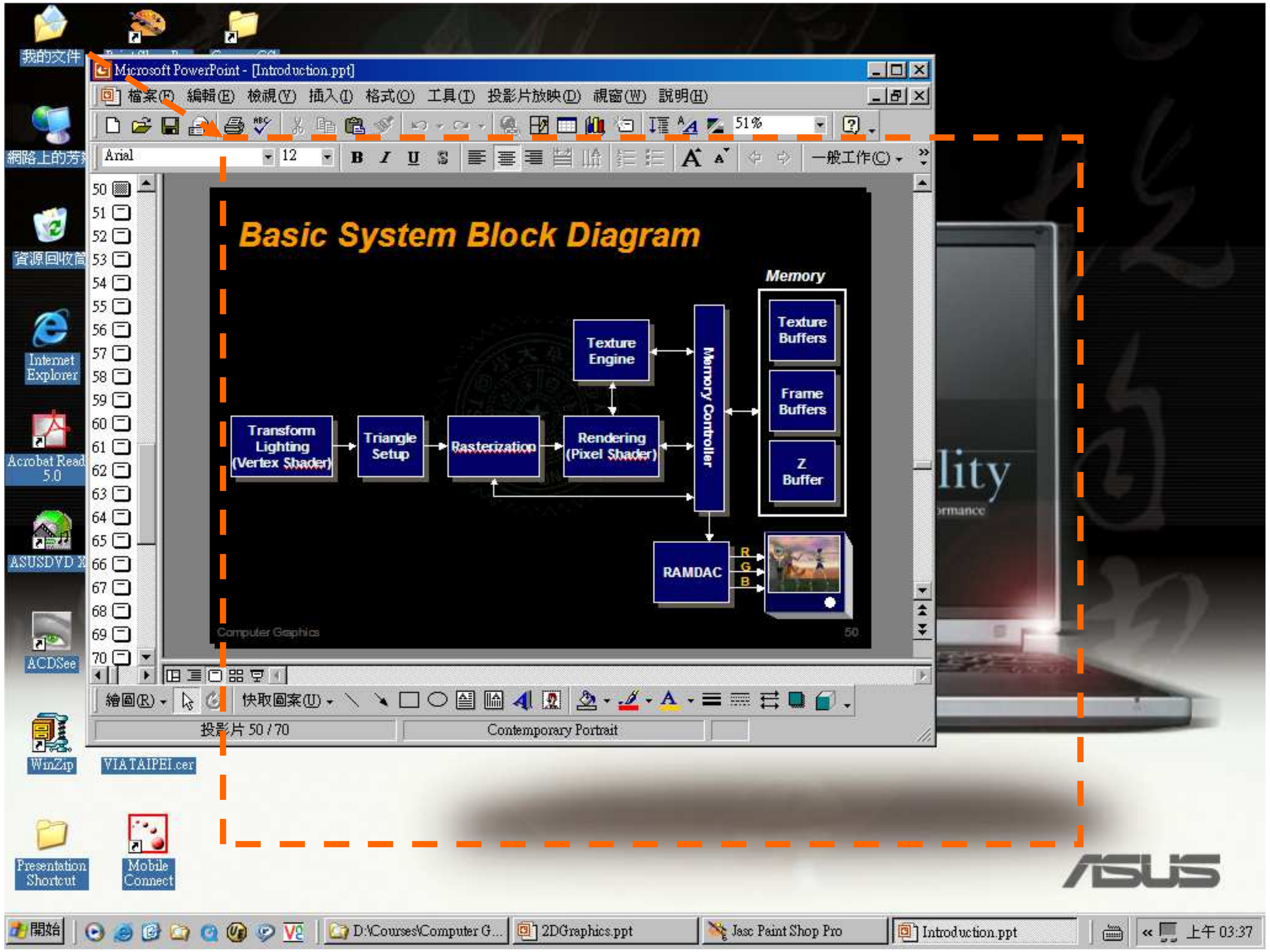
Block Transfer

- Bitblt (Bit Block Transfer)



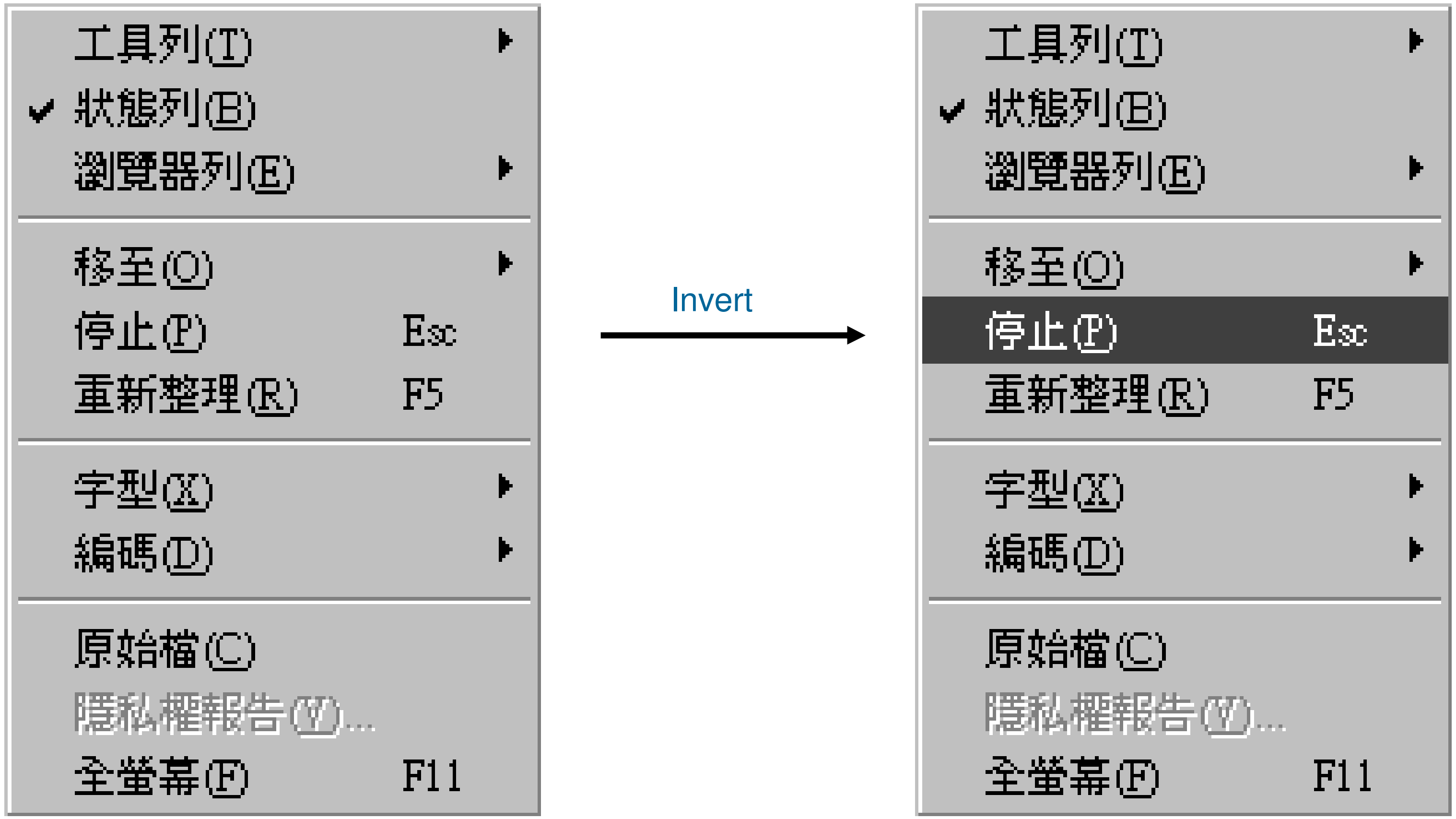
Block Transfer

- Bitblt (Bit Block Transfer)



Raster Operations

- **ROP3 (Source, Pattern, Destination)**



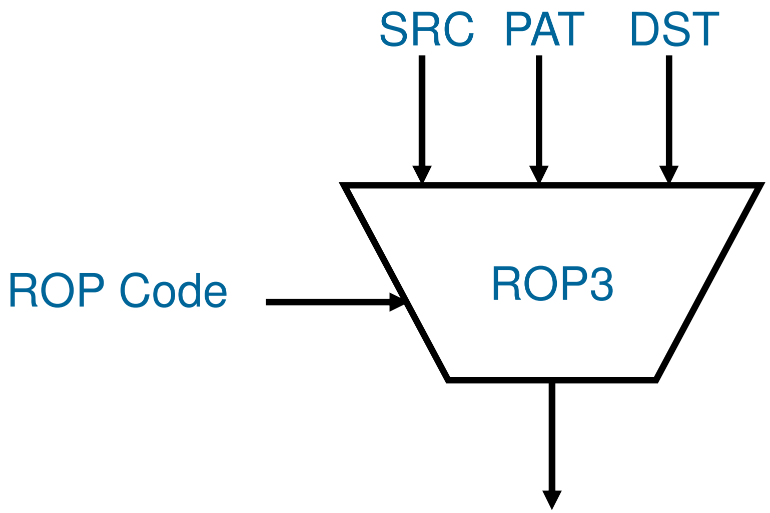
Raster Operations

- ROP3 (Source, Pattern, Destination)

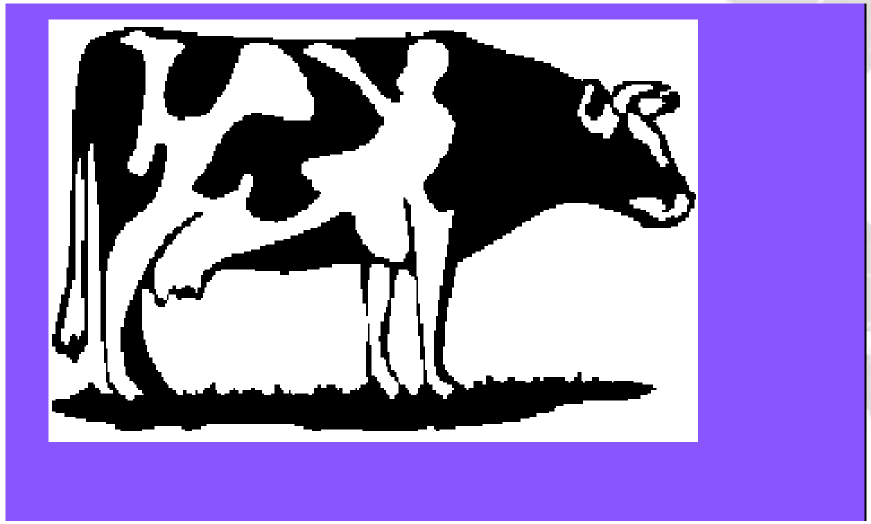
Brush "P"	1	1	1	1	0	0	0	0	Code in Hex	Name	Boolean Equation (C style)
Source "S"	1	1	0	0	1	1	0	0			
Background "D"	1	0	1	0	1	0	1	0			
	0	0	0	0	0	0	0	0	00	BLACKNESS	0
	0	0	0	1	0	0	0	1	11	NOTSRCERASE	~(D|S)
	0	0	1	1	0	0	1	1	33	NOTSRCCOPY	~S
	0	1	0	0	0	1	0	0	44	SRCERASE	S&~D
	0	1	0	1	0	1	0	1	55	DSTINVERT	~D
	0	1	0	1	1	0	1	0	5A	PATINVERT	D^P
	0	1	1	0	0	1	1	0	66	SRCINVERT	D^S
Result	1	0	0	0	1	0	0	0	88	SRCAND	D&S
	1	0	1	1	1	0	1	1	BB	MERGEPAINT	D|~S
	1	1	0	0	0	0	0	0	C0	MERGECOPY	P&S
	1	1	0	0	1	1	0	0	CC	SRCCOPY	>S
	1	1	1	0	1	1	1	0	EE	SRCPAINT	D|S
	1	1	1	1	0	0	0	0	F0	PATCOPY	P
	1	1	1	1	1	0	1	1	FB	PATPAINT	D|P|~S
	1	1	1	1	1	1	1	1	FF	WHITENESS	1

Raster Operations

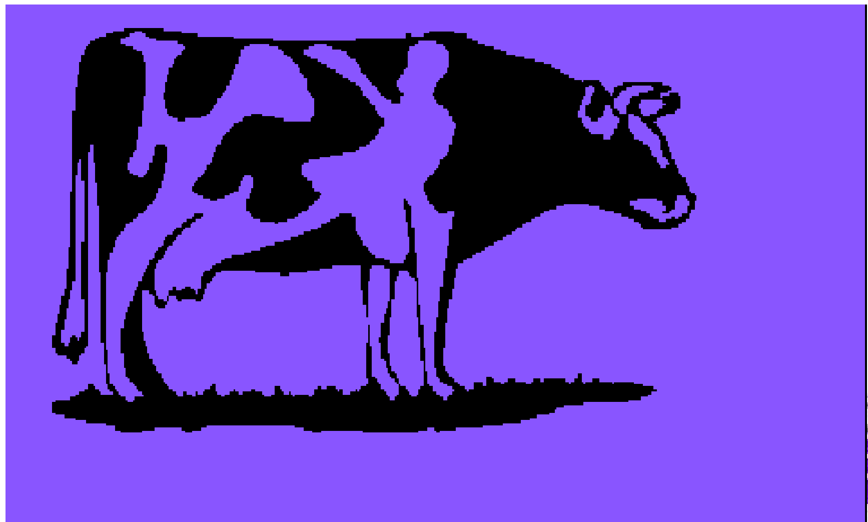
- **ROP3 (Source, Pattern, Destination)**



SRC Only



SRC & DST



Q&A