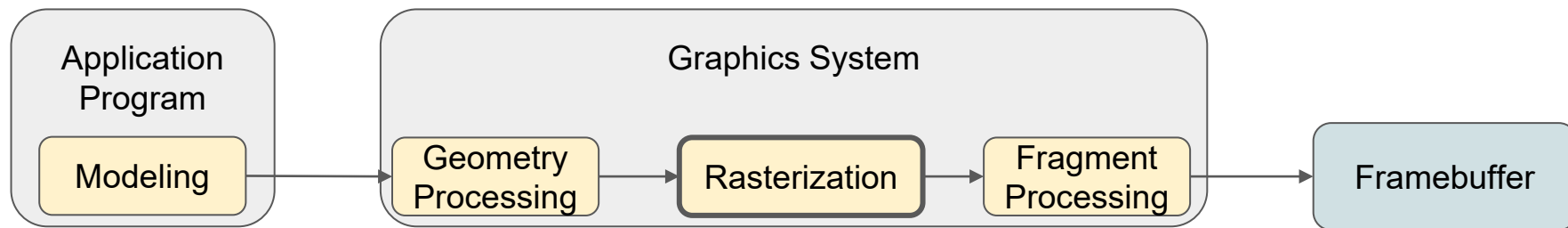


Rasterization and Fragment Processing

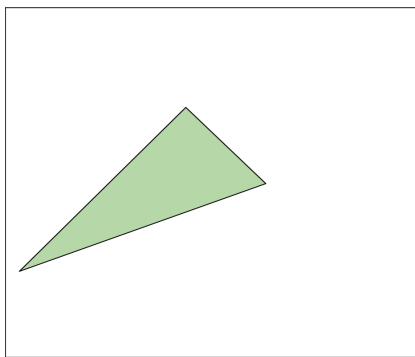
CS 425: Computer Graphics 1

Rasterization

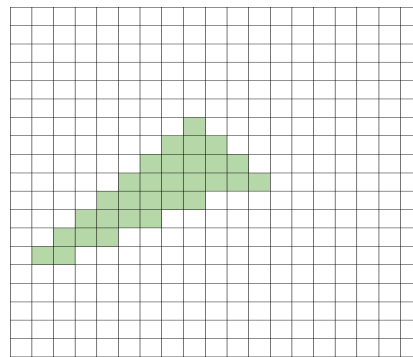


Rasterization

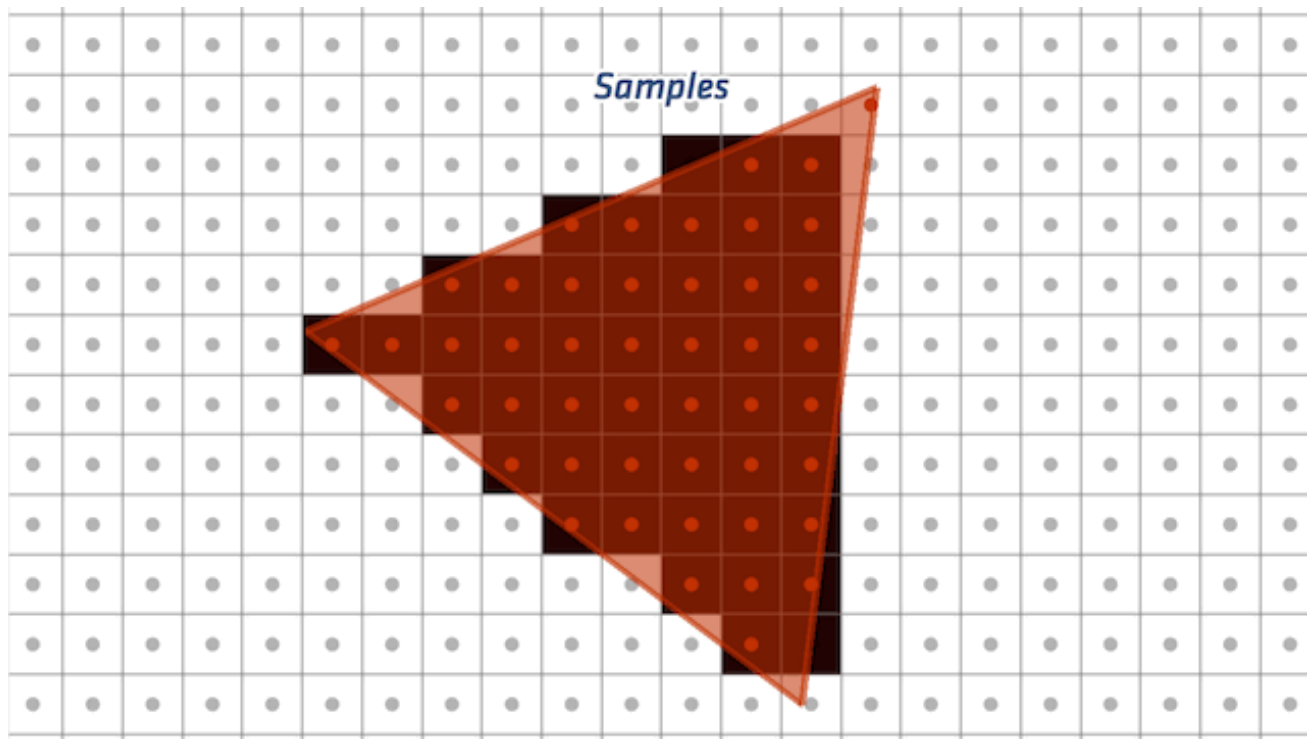
- Process of converting the vertices that are output from the clipping stage to fragments.
- Fragments are potential pixels.



Clipped object in vertex representation



Fragments of the rasterized object



Rasterization: Lines

DDA algorithm: Idea

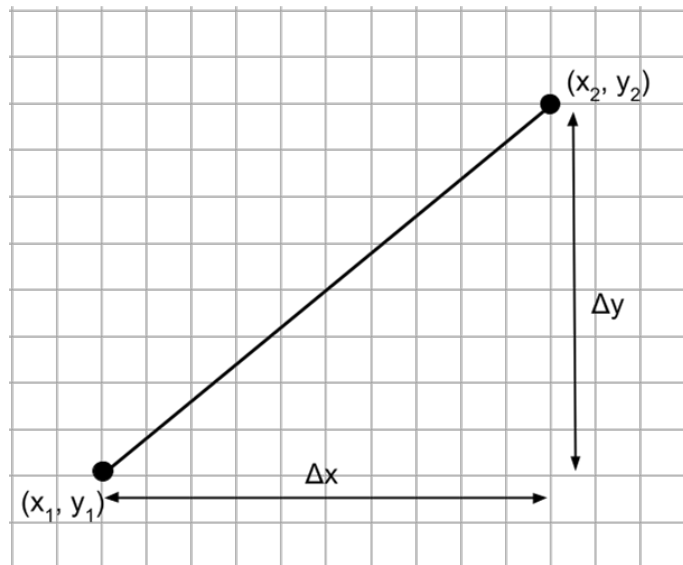
$$y = mx + h$$

$$y_{i+1} = m (x_i + \Delta x) + h$$

$$y_{i+1} = mx_i + h + m\Delta x = y_i + m\Delta x$$

If we set $\Delta x = 1$, then

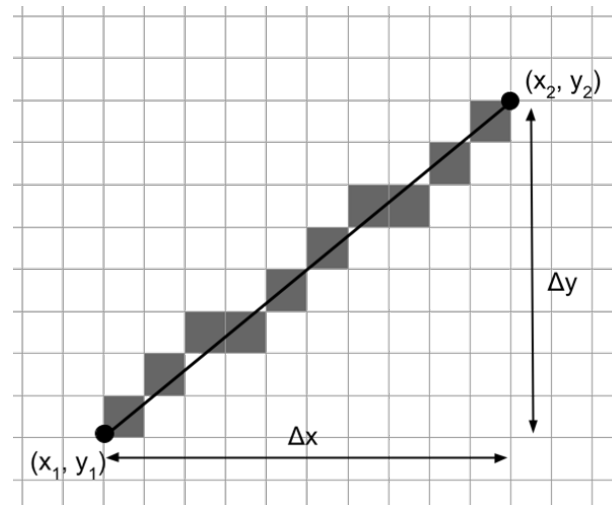
$$y_{i+1} = y_i + m$$



Rasterization: Lines

DDA algorithm

```
for (ix = x1; ix <= x2; ++ix) {  
    y += m;  
    writePixel(x, round(y), line_color);  
}
```



Rasterization: Lines

DDA algorithm

- Assumes $0 \leq m \leq 1$
- For each x , find best y .
- For slopes greater than 1 we can swap the roles of x and y .
- Approximating m by rounding it over many iterations can induce error and result in the rasterized line being off the actual line.
- Has floating point calculations and rounding function which are computationally expensive.

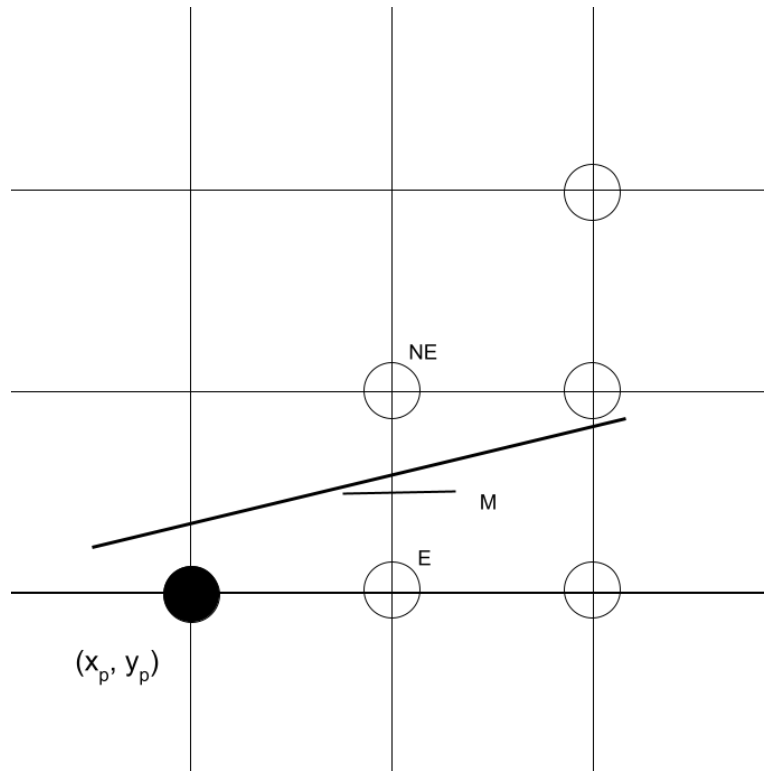
Rasterization: Lines

Bresenham's Algorithm:

We assume a line with a slope m such that: $0 \leq m \leq 1$

$$y = mx + h$$

$$y = (dy / dx) x + h$$



Rasterization: Bresenham's Algorithm

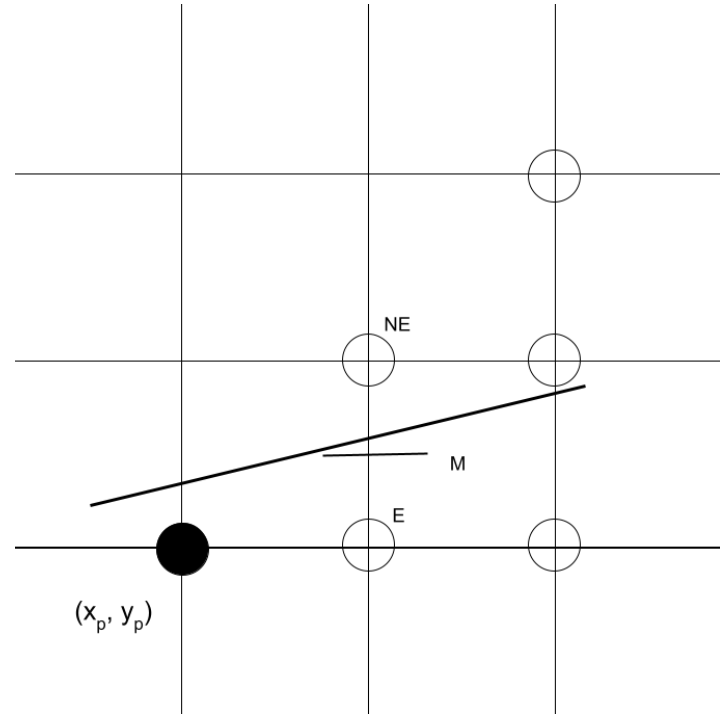
In the implicit form,

$$F(x, y) = x.dy - y.dx + h.dx = 0$$

$$\text{Or, } F(x, y) = A.x + B.y + C = 0$$

$$A = dy, B = -dx, C = h.dx$$

$F(x, y) = 0$ on the line, < 0 above the line, and > 0 below the line



Rasterization: Bresenham's Algorithm

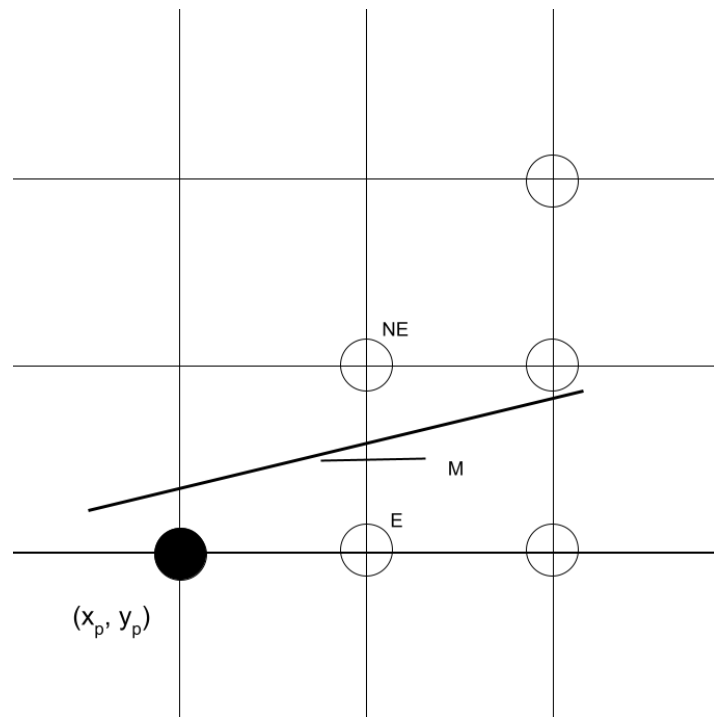
At the current iteration, the pixel at (x_p, y_p) is chosen. Now how to choose the next pixel?

The choice is between pixels E and NE.

Compute $F(M)$, $F(M)$ is the decision value 'd'

$$d = A(x_p + 1) + B(y_p + \frac{1}{2}) + C$$

If $d > 0$, choose NE, otherwise choose E

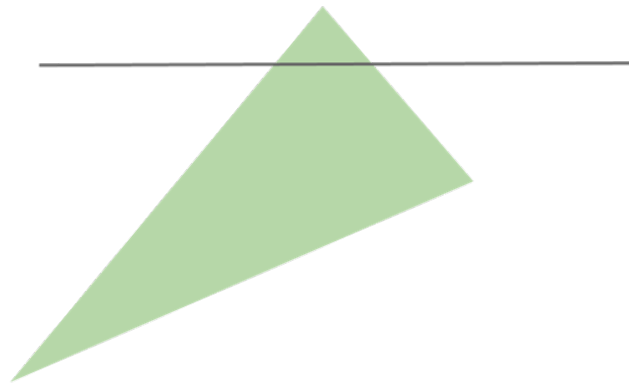


Rasterization: Filling

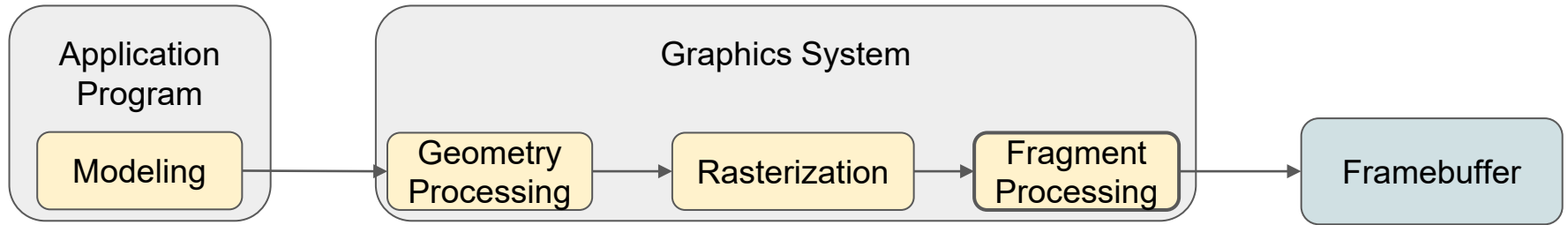
Primitive assembly information is utilized for filling.

Inside-out testing is done to determine what pixels are part of the polygon.

- Crossing or odd-even test.



Hidden Surface Removal

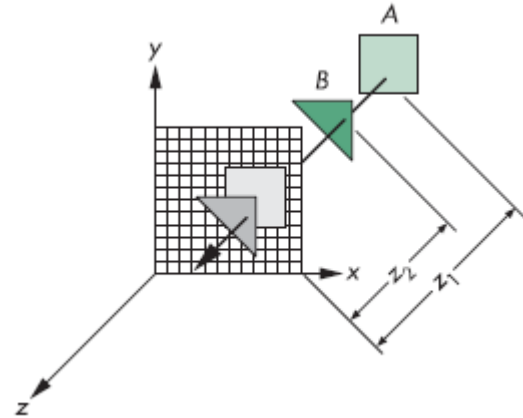


Hidden Surface Removal: The z-Buffer algorithm

A separate buffer to hold the depth information.

Initialized to maximum depth value from center of projection. Color buffer to the background color.

Iteratively rasterize polygons and simultaneously fill the z-buffer.

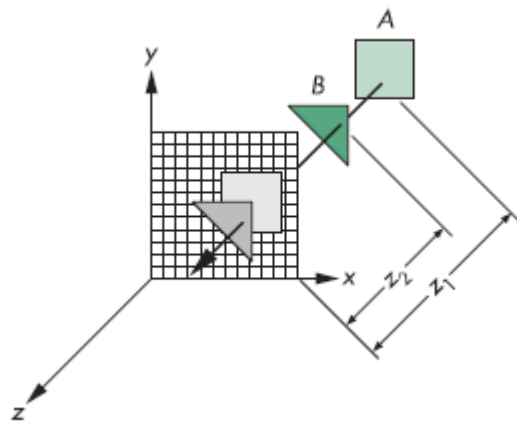


Hidden Surface Removal: The z-Buffer algorithm

Compare depth of incoming fragment with value in z-buffer.

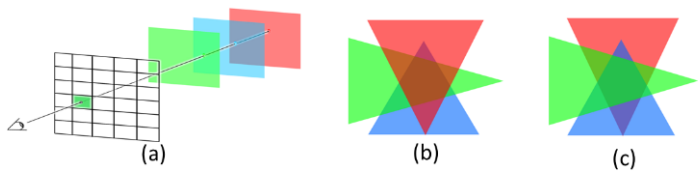
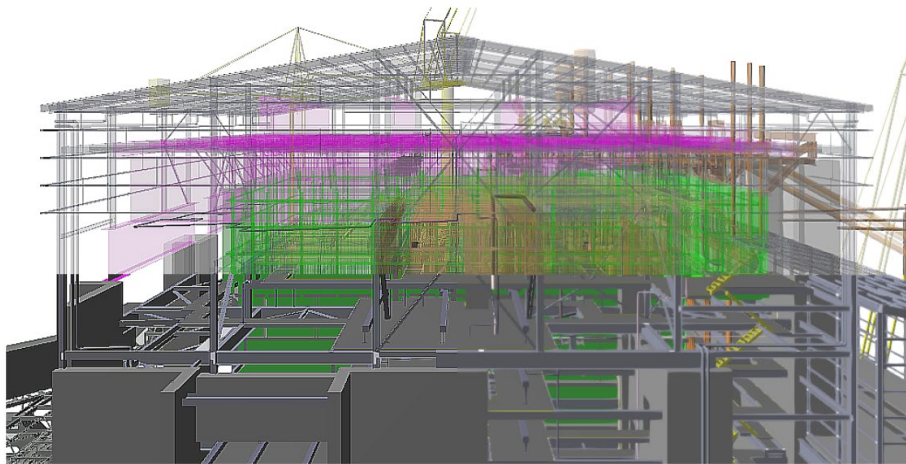
$\text{Depth}_{\text{new}} > \text{Depth}_{\text{z-buffer}}$ we have already rasterized a fragment that is closer to the viewer.

Otherwise the incoming fragment is placed in the color buffer and the z-buffer is updated.



Transparency

- Important to denote relationships among objects in a scene.
- One of the five major challenges in interactive rendering [Andersson,



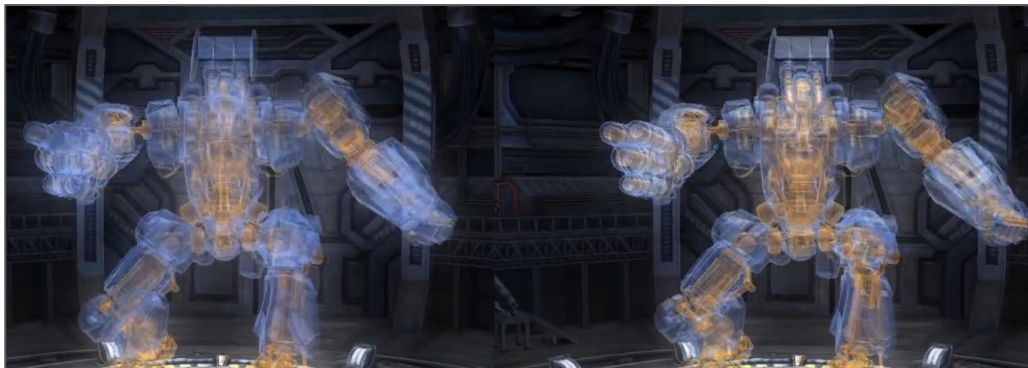
If not all fragments are opaque,
how can we blend them?

[Maule et al., 2012]

Transparency

- Blend fragment color and opacity such that the resulting pixel color is:
$$\mathbf{c} = \mathbf{c}_n + (1 - \alpha_n) \dots [\mathbf{c}_2 + (1 - \alpha_2)[\mathbf{c}_2 + (1 - \alpha_1)\mathbf{c}_0]]$$
- Order dependent: final pixel color depends of the fragment color.

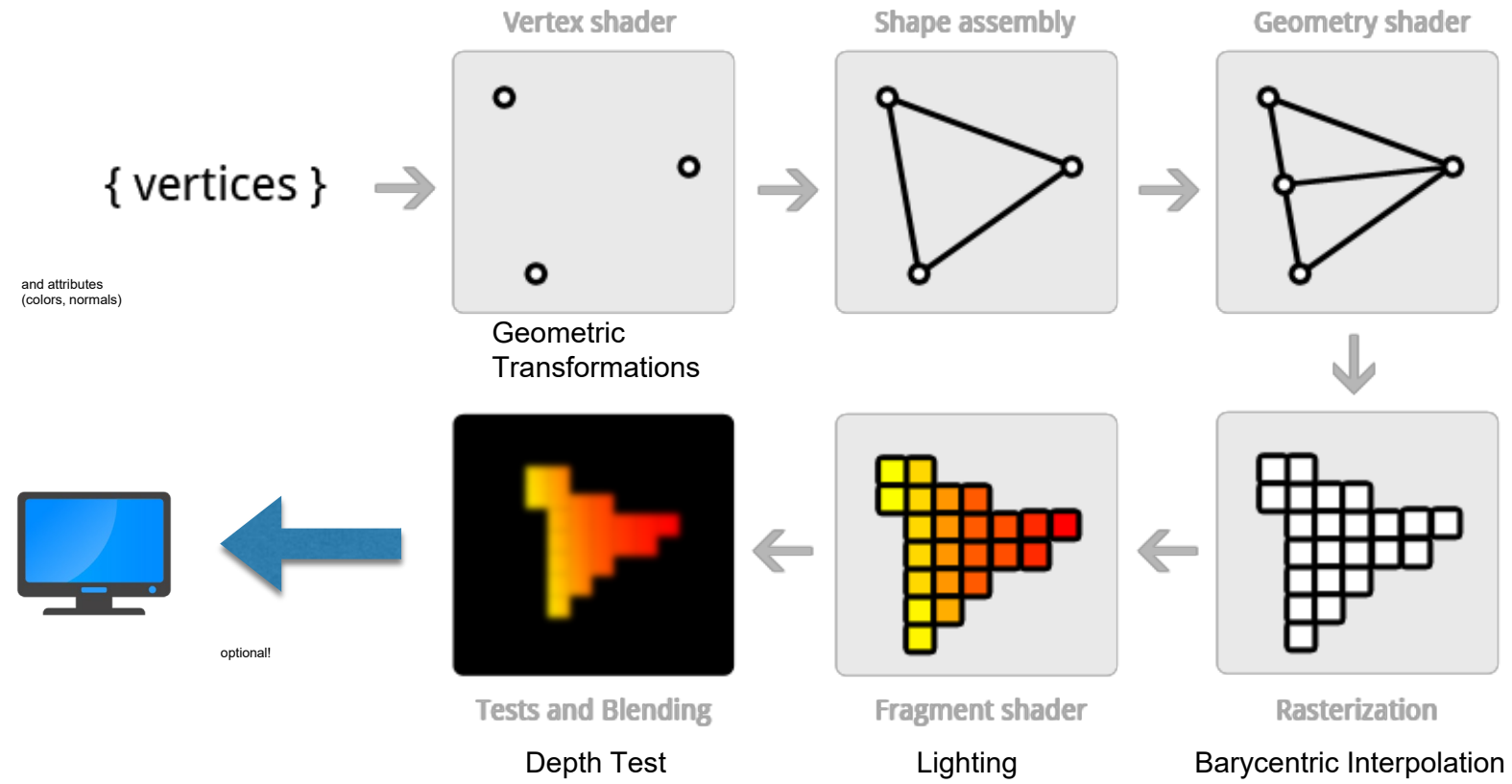
Incorrect result as fragments are generated and blended in 'random' order.



Correct result, sorting fragments.

ATI Tech Demo

OpenGL pipeline



Vertex Shader Fragment Shader + Tests

