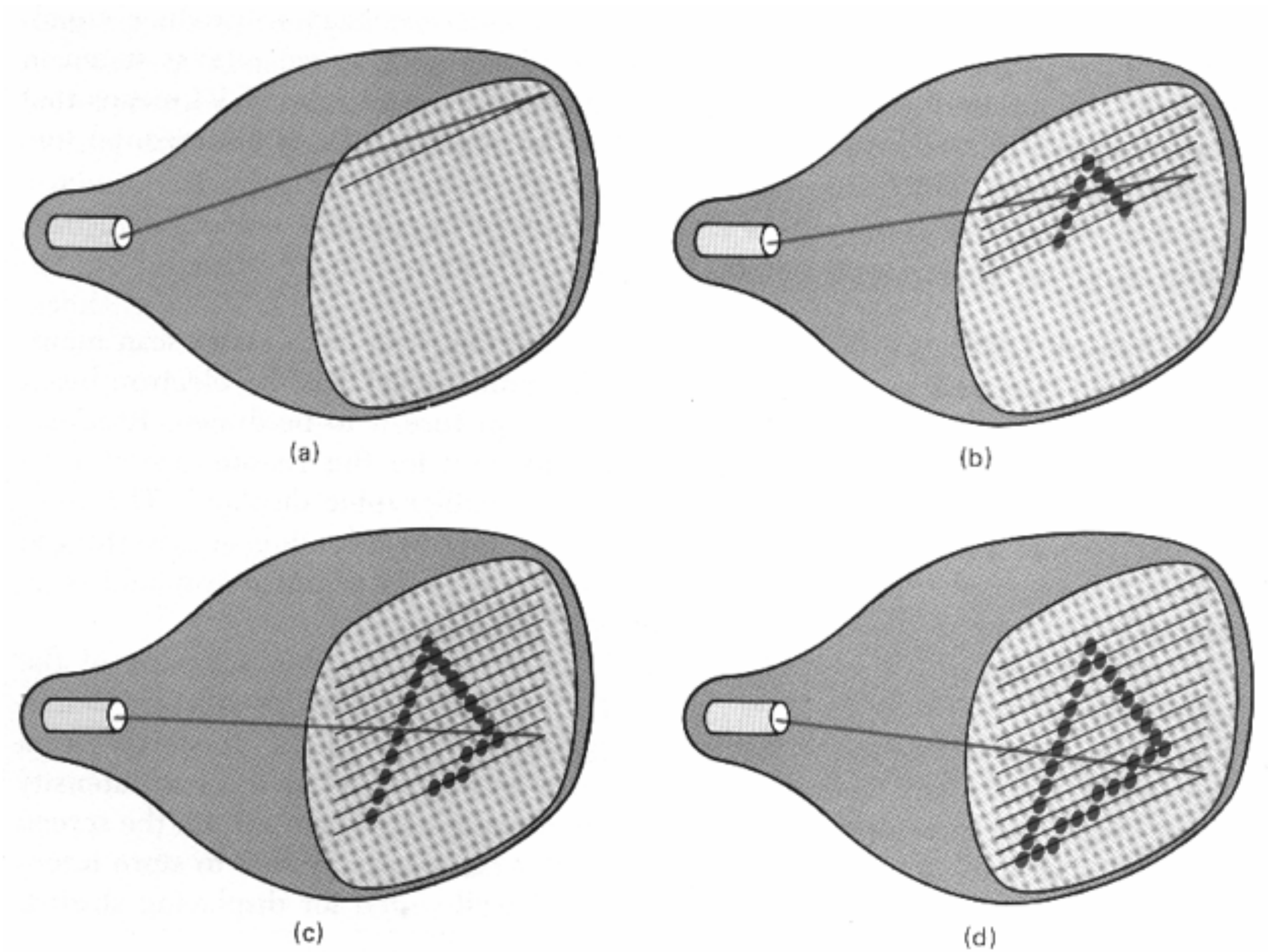
# CS130 : Computer Graphics

Tamar Shinar
Computer Science & Engineering
UC Riverside

# Raster Devices and Images

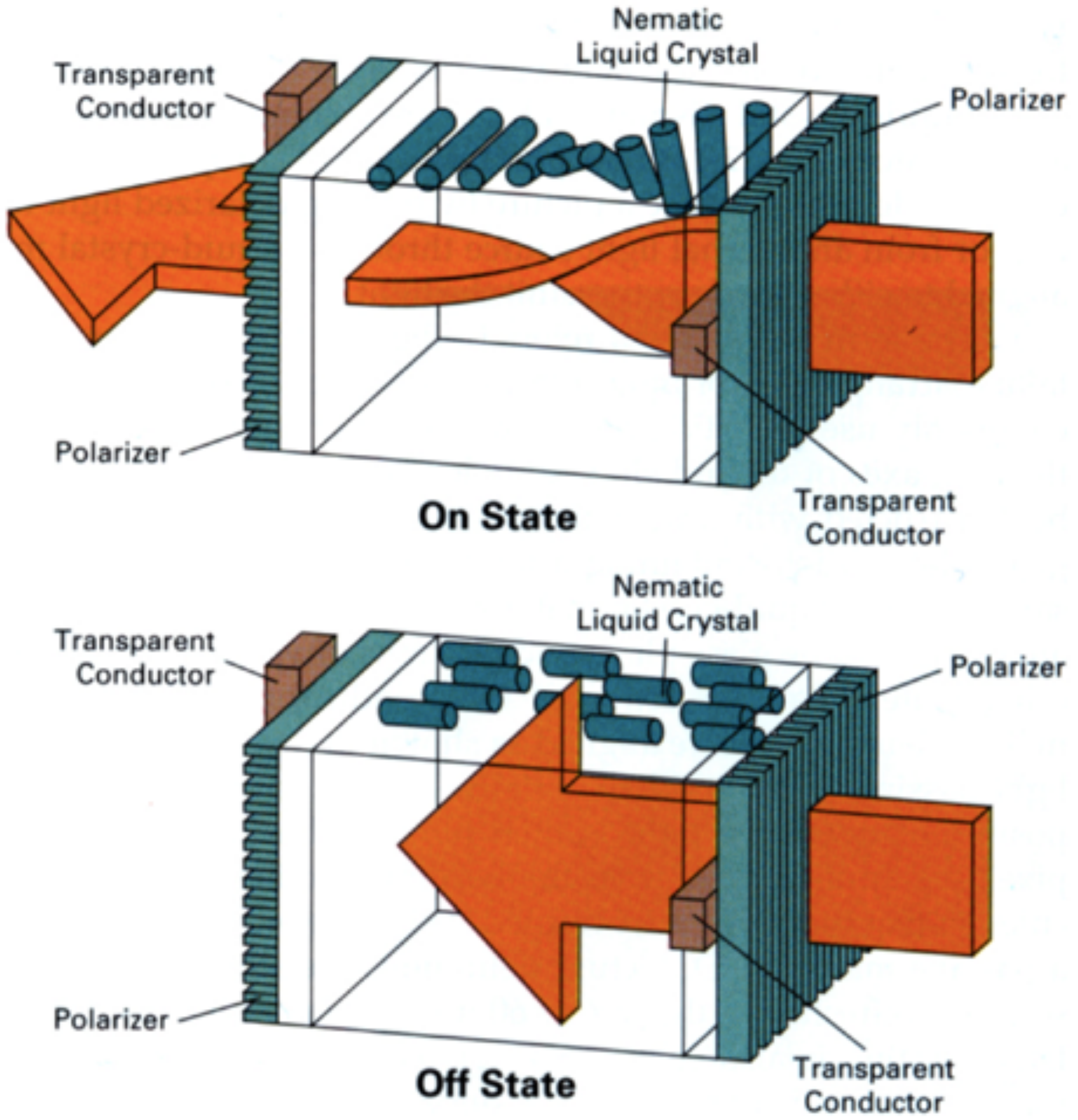
# Raster Devices

# Raster Display



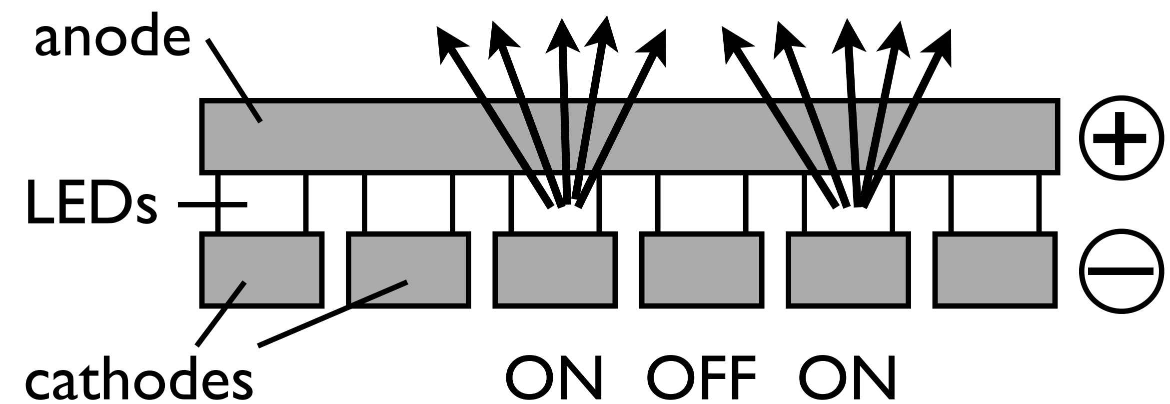

Hearn, Baker, Carithers

# Transmissive vs. Emissive Display



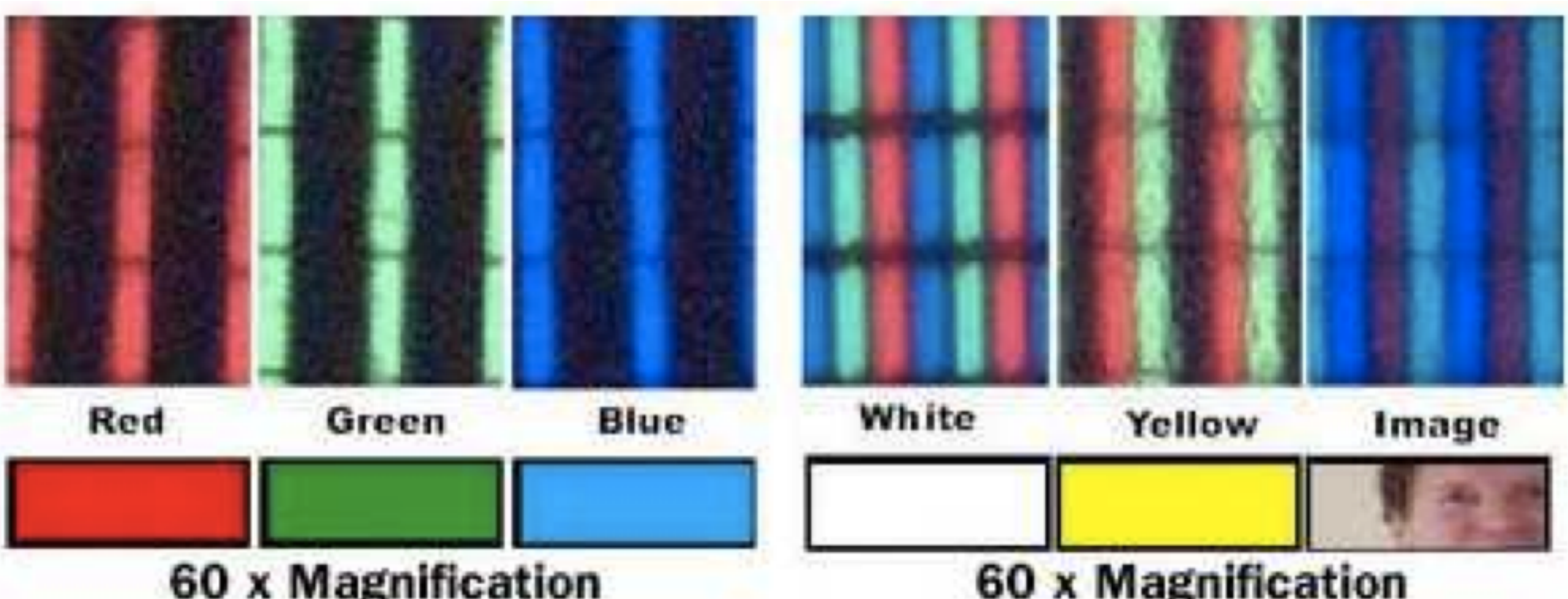

[H&B, Fig. 2-16]

LCD




LED

# Raster Display




red, green, blue subpixels

# What is an image?

**Continuous image**

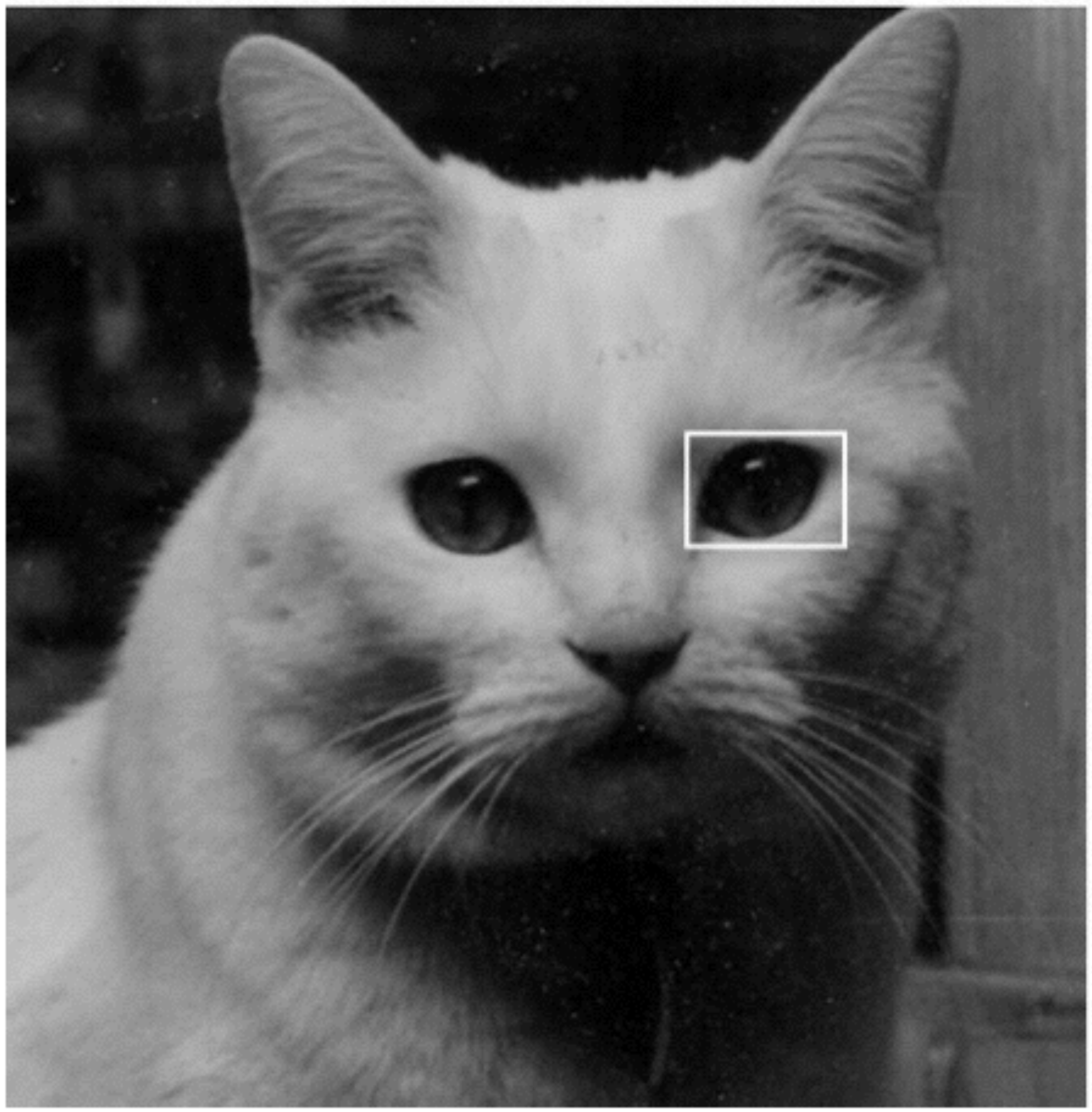
$$I : R \to V$$

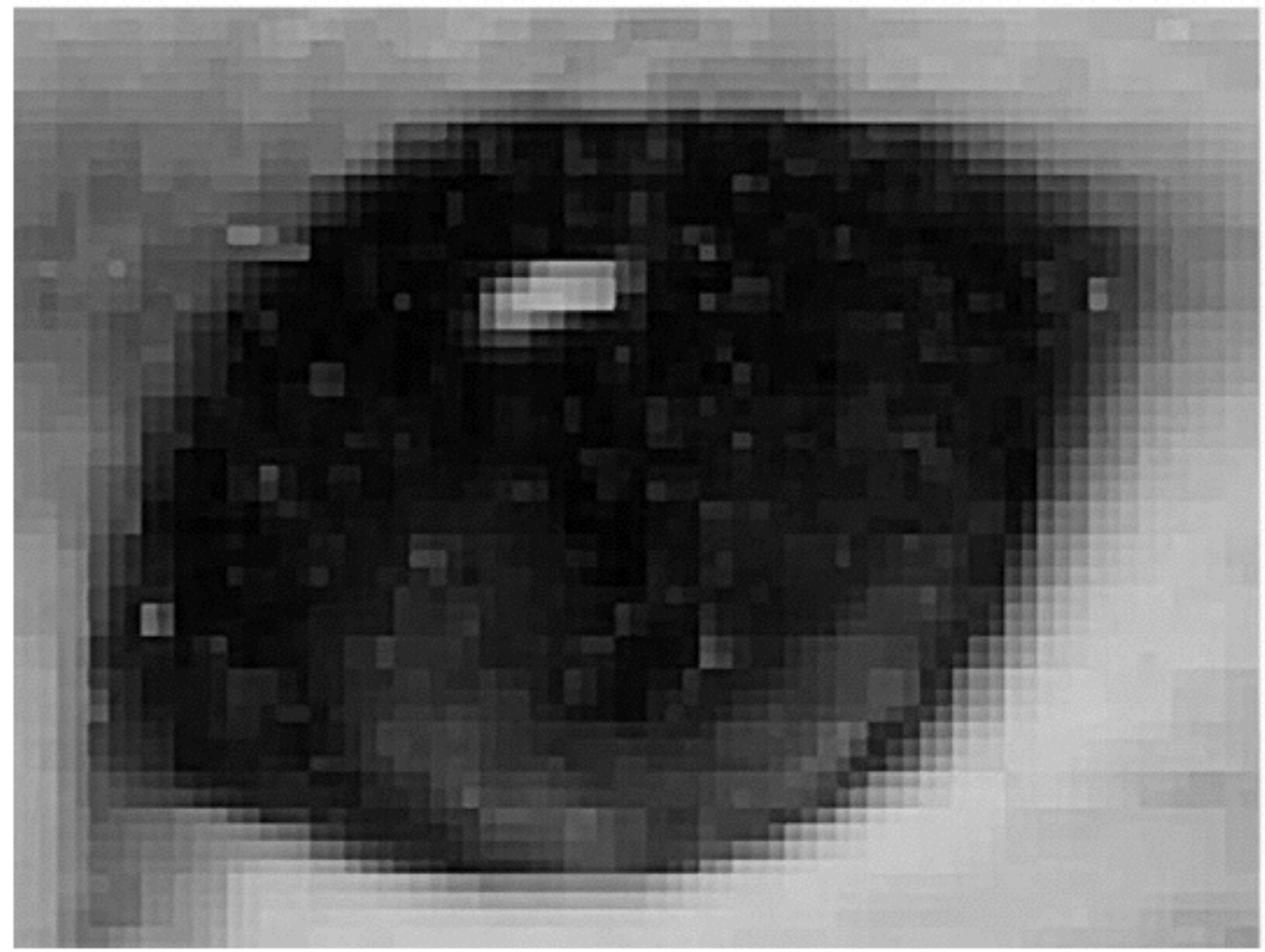
$$R \subset \mathbb{R}^2$$

$$V = \mathbb{R}^+ \quad \text{(grayscale)}$$

$$V = (\mathbb{R}^+)^3 \quad \text{(color)}$$

# Raster Image





A **raster image** is 2D array storing pixel values at each pixel

# What is an image?

**Raster image**

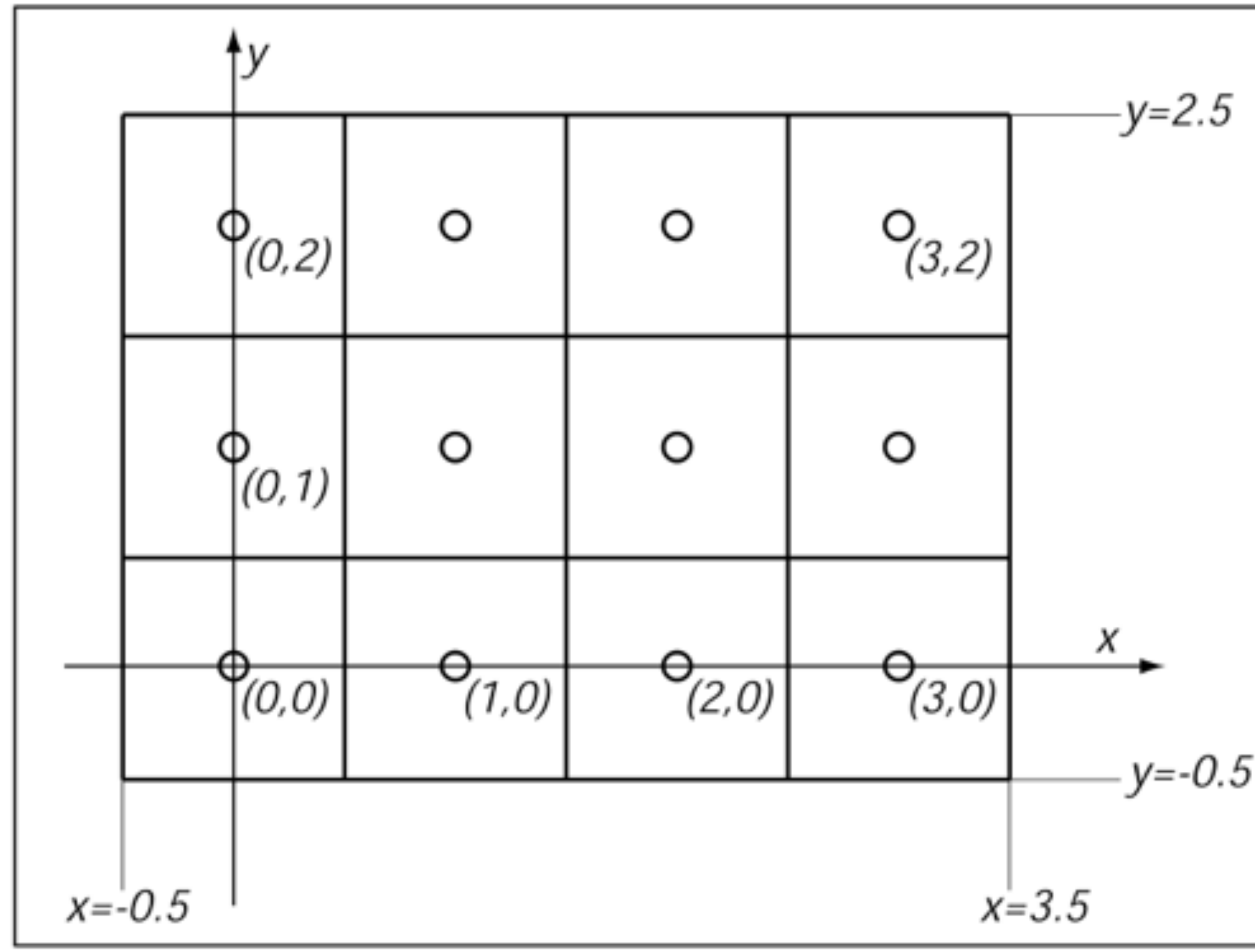
$$I : R \to V$$

$$R \subset \mathbb{Z}^2$$

$$V = \mathbb{R}^+ \quad \text{(grayscale)}$$

$$V = (\mathbb{R}^+)^3 \quad \text{(color)}$$

Each pixel value represents the **average color** of the image over that pixel's area.




$$[-0.5, n_x - 0.5] \times [-0.5, n_y - 0.5]$$

$n_x$ = number of columns

$n_y$ = number of rows

# What is an image?

**Raster image**

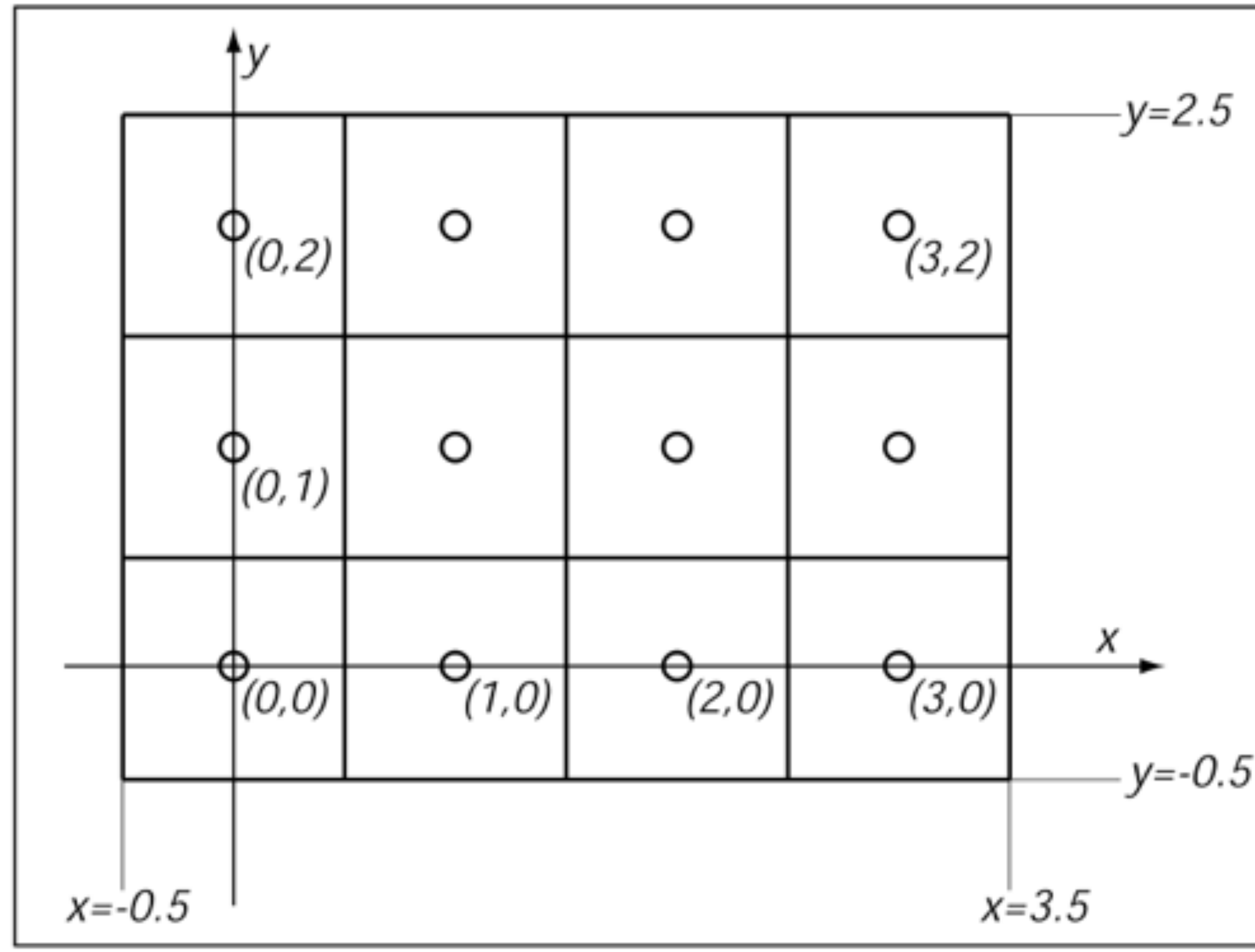
$$I : R \to V$$

$$R \subset \mathbb{Z}^2$$

$$V = [0, 1] \quad \text{(grayscale)}$$

$$V = [0, 1]^3 \quad \text{(color)}$$

Each pixel value represents the **average color** of the image over that pixel's area.




$$[-0.5, n_x - 0.5] \times [-0.5, n_y - 0.5]$$

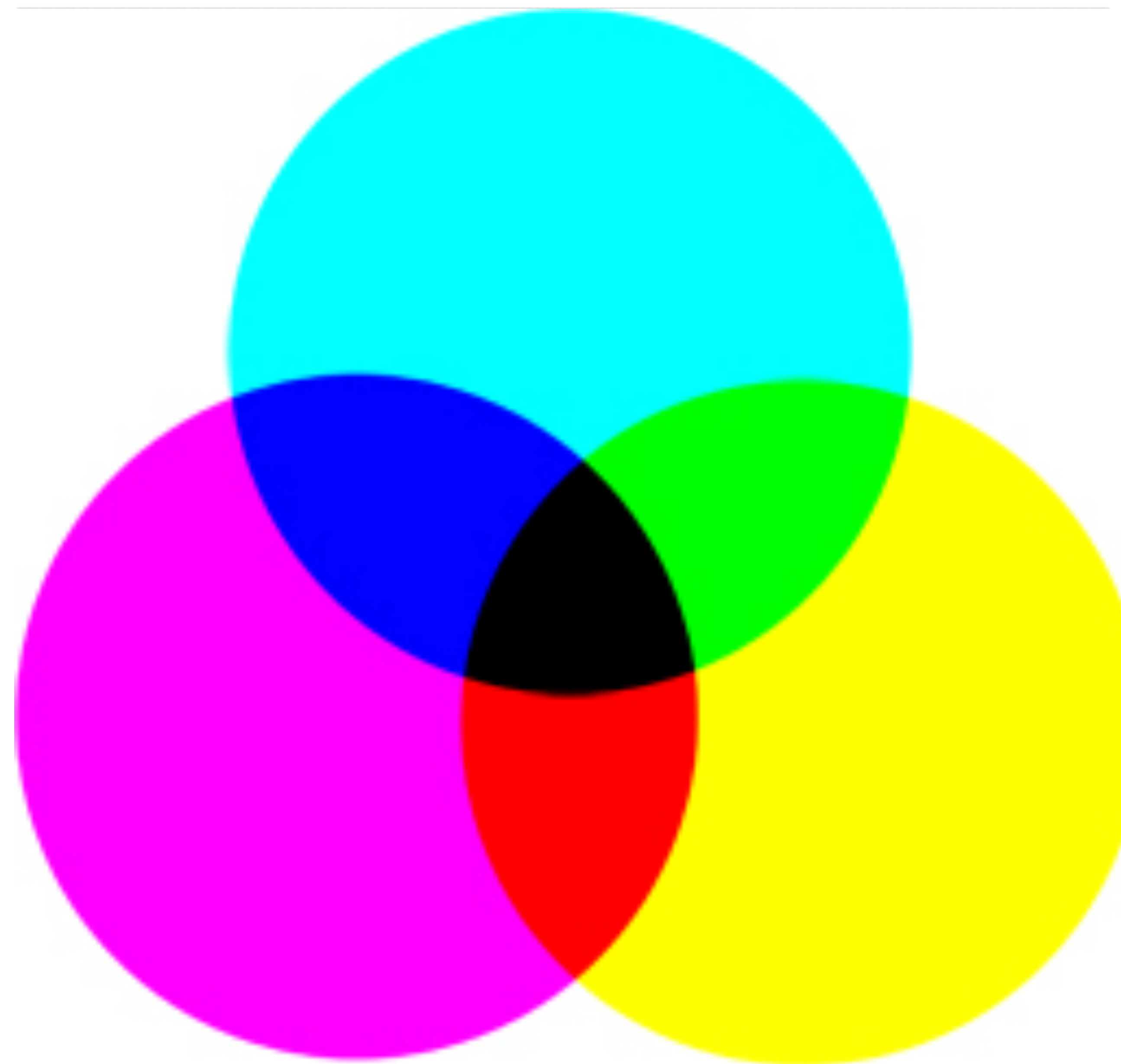
$n_x$ = number of columns

$n_y$ = number of rows

# Color Representation

*additive*

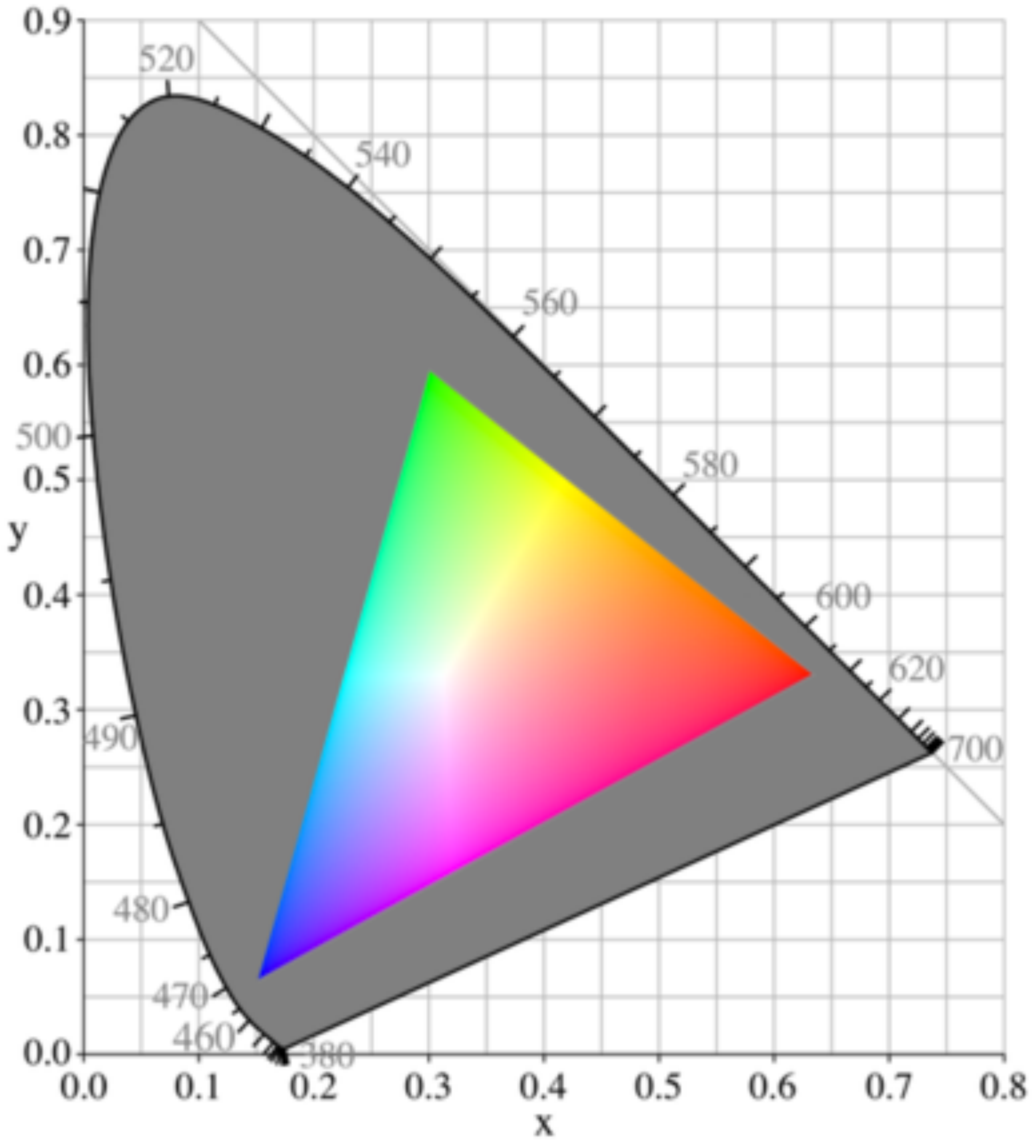
RGB



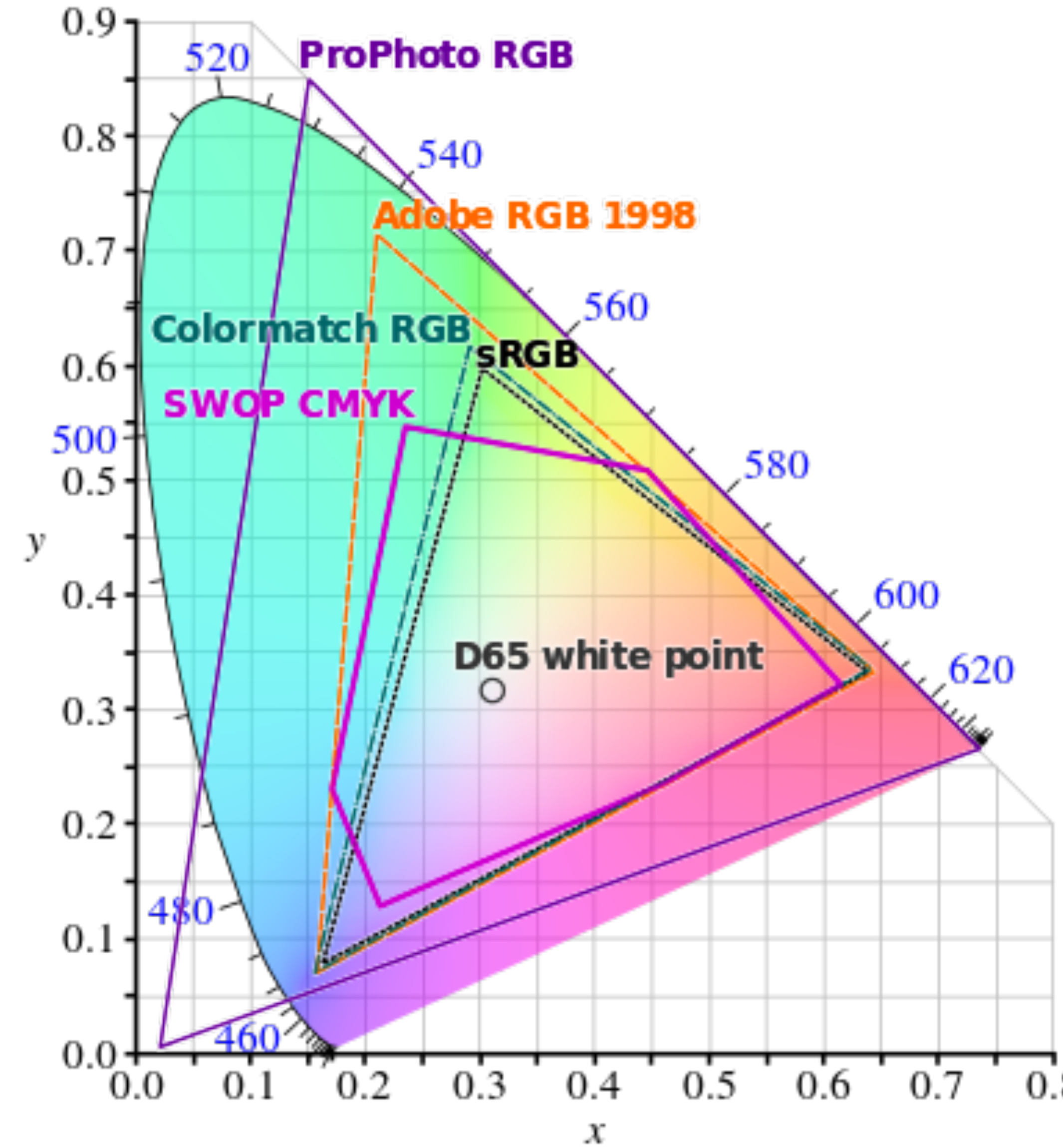
*subtractive*

CMYK

# Color Representation




sRGB color triangle




comparison of color gamuts

[wikipedia]

# Bit depth - defined by device standards

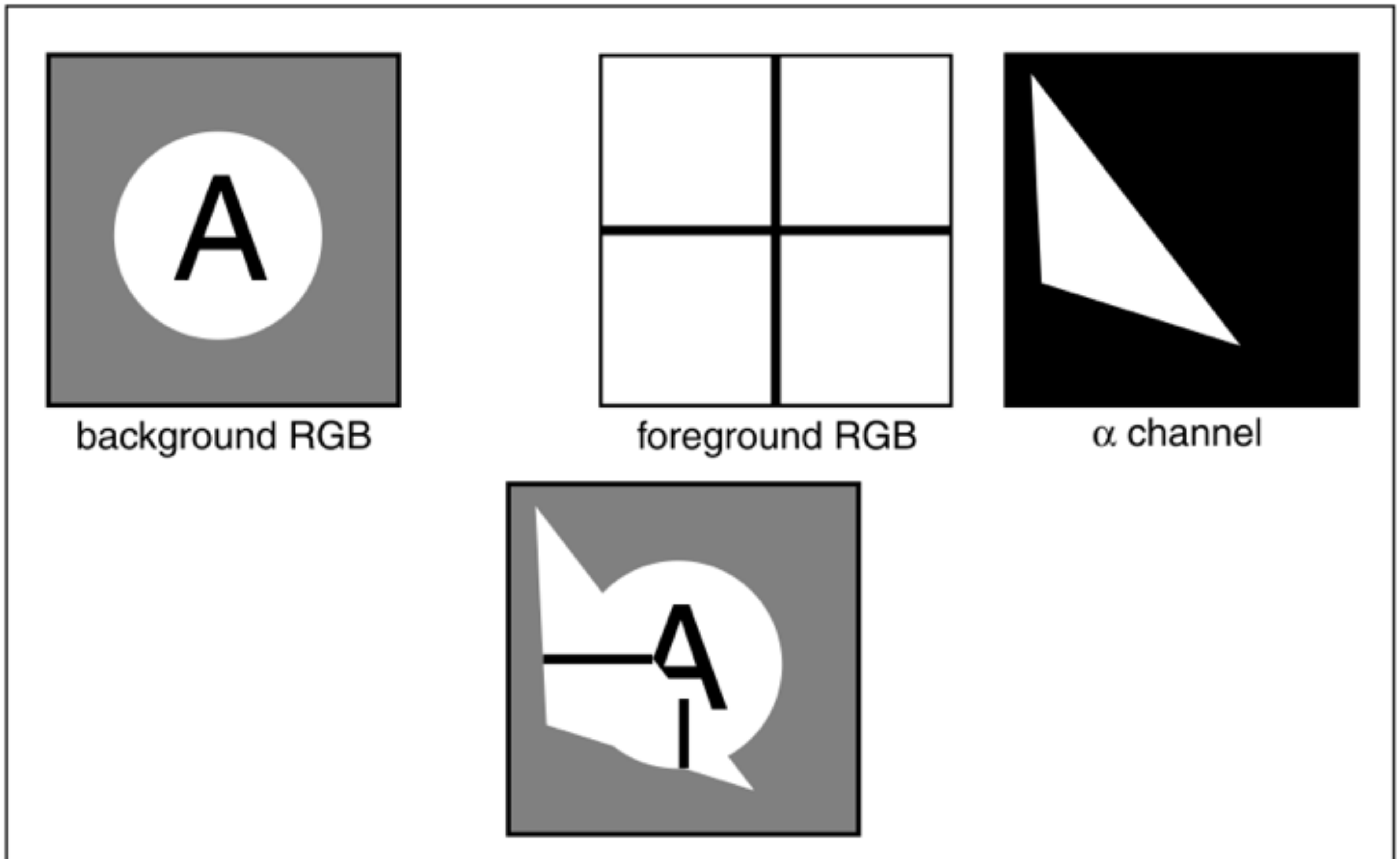
| Bit-Depth | Number of Colors |
|---|---|
| 1 | 2<br>(monochrome) |
| 2 | 4<br>(CGA) |
| 4 | 16<br>(EGA) |
| 8 | 256<br>(VGA) |
| 16 | 65,536<br>(High Color, XGA) |
| 24 | 16,777,216<br>(True Color, SVGA) |
| 32 | 16,777,216<br>(True Color + Alpha Channel) |

*(Note alpha)*

(Humans can perceive ~10,000,000 colors)

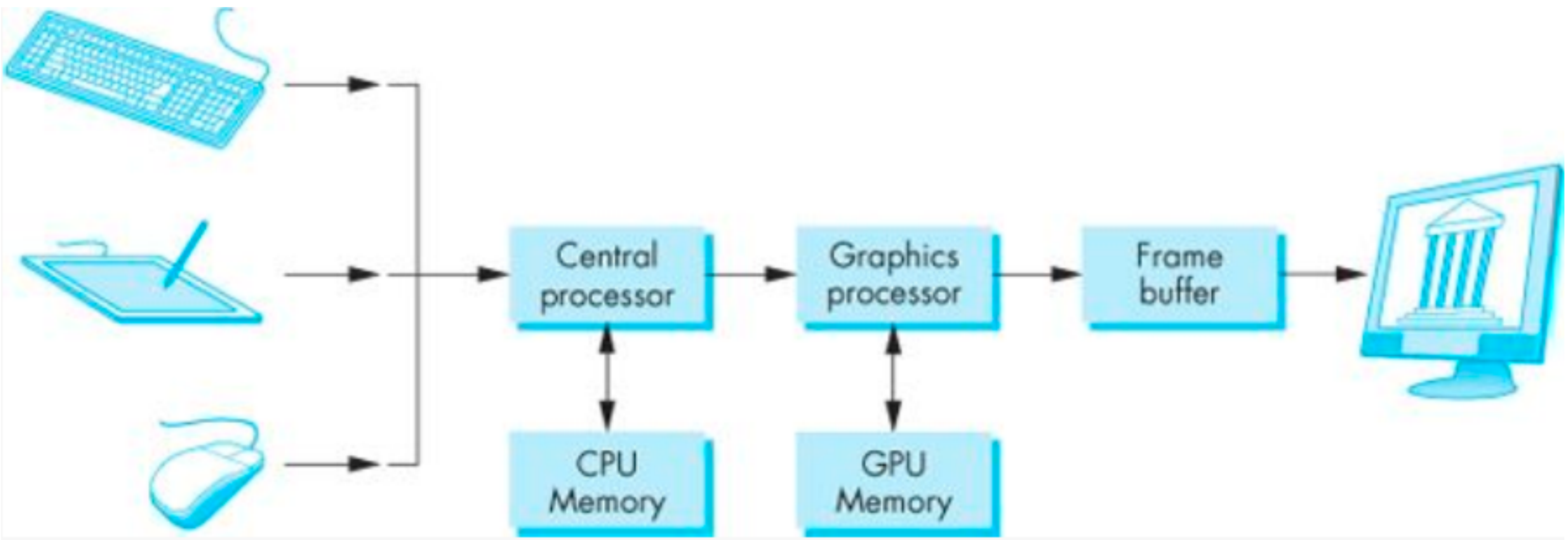
# Alpha Channel

$$\mathbf{c} = \alpha \mathbf{c}_f + (1 - \alpha)\mathbf{c}_b$$



# Graphics Pipeline

# Modern graphics system




[Angel and Shreiner]

# Z-buffer Rendering

- Z-buffering is very common approach, also often accelerated with hardware
- OpenGL is based on this approach

# Choice of primitives

- Which primitives should an API contain?
  - small set - supported by hardware, *or*
  - lots of primitives - convenient for user

# Choice of primitives

- Which primitives should an API contain?

➡ **small set - supported by hardware**

  - lots of primitives - convenient for user

# Choice of primitives

- Which primitives should an API contain?

➡ **small set - supported by hardware**

  - lots of primitives - convenient for user

Performance is in **10s millions polygons/sec**
**portability, hardware support** key

# Choice of primitives

- Which primitives should an API contain?

➡ **small set - supported by hardware**

  - lots of primitives - convenient for user

GPUs are optimized for **points**, **lines**, and **triangles**

# Choice of primitives

- Which primitives should an API contain?

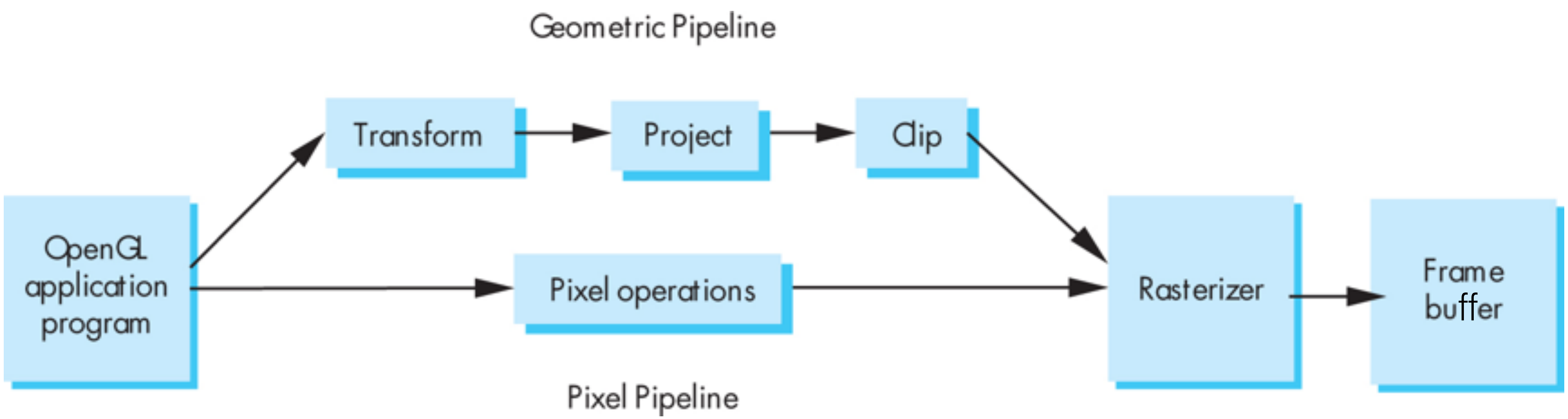
➡ **small set - supported by hardware**

- lots of primitives - convenient for user

GPUs are optimized for **points**, **lines**, and **triangles**

**Other geometric shapes** will be built out of these

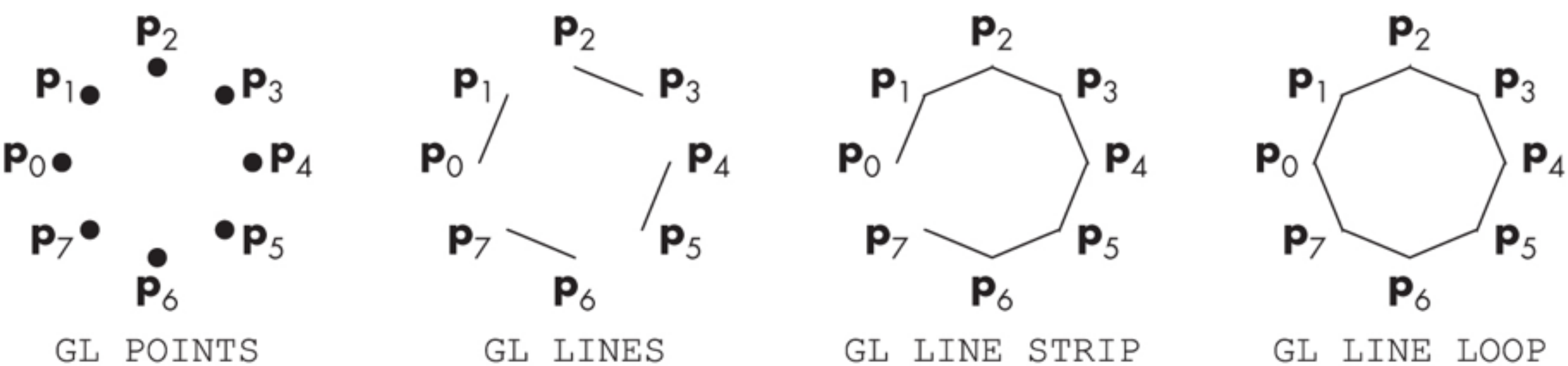
# Two classes of primitives




[Angel and Shreiner]

**Geometric** : points, lines, polygons
**Image** : arrays of pixels

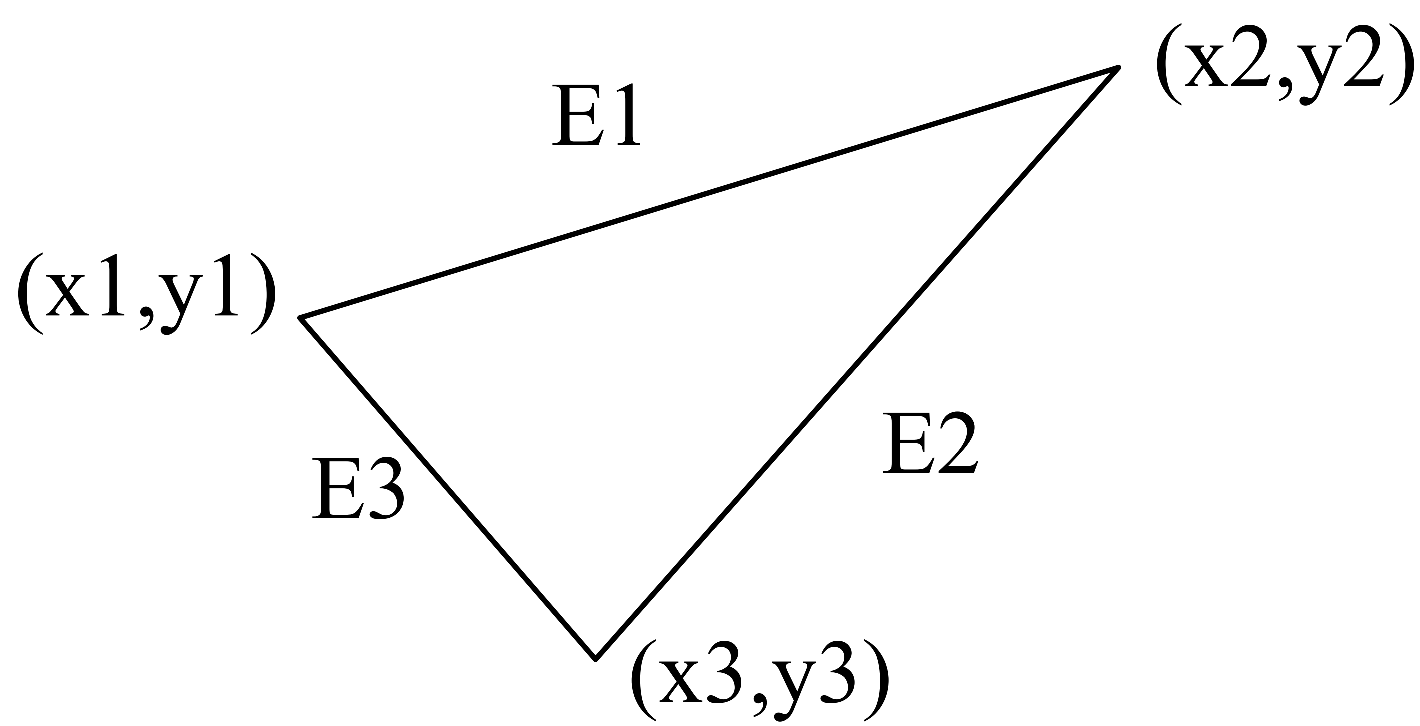
# Point and line segment types



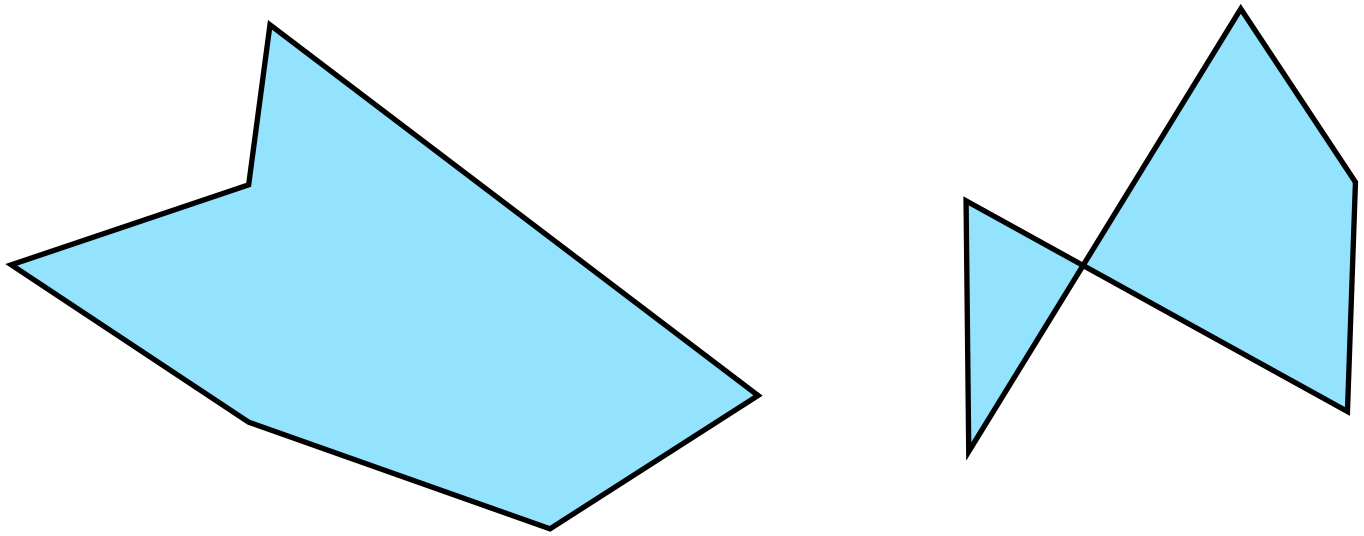

[Angel and Shreiner]

# Polygons

- Multi-sided planar element composed of edges and vertices.
- Vertices (singular: vertex) are represented by points
- Edges connect vertices as line segments

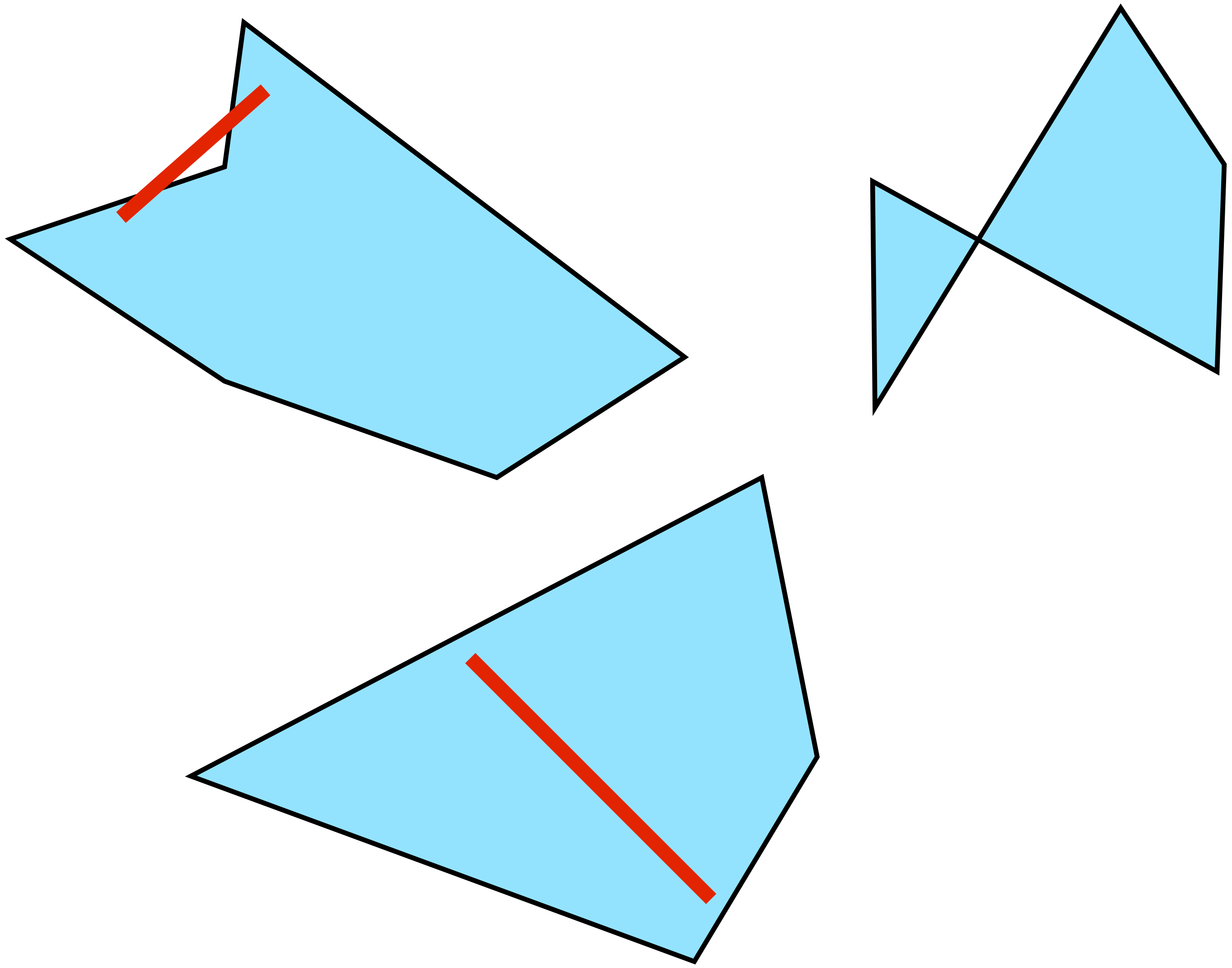


# Valid polygons



- Simple
- Convex
- Flat

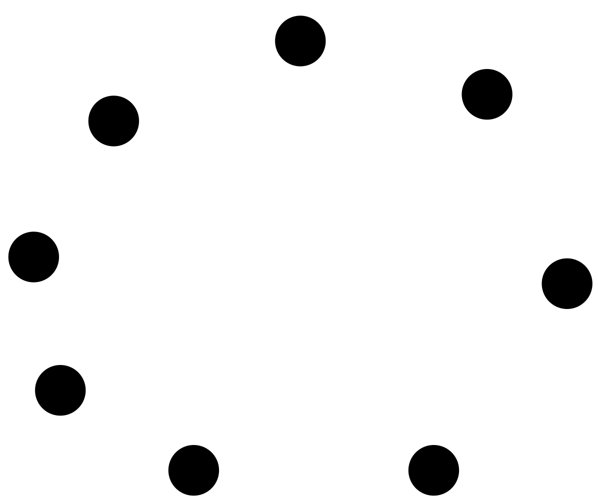
# Valid polygons



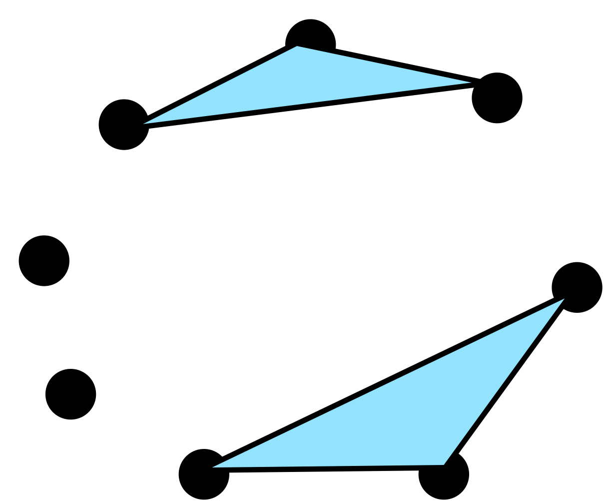
- Simple
- Convex
- Flat

# OpenGL polygons

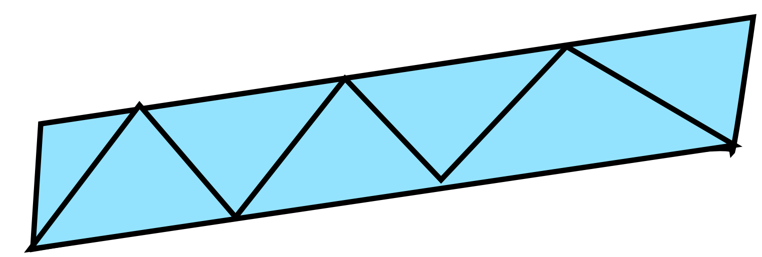
- Only triangles are supported (in latest versions)



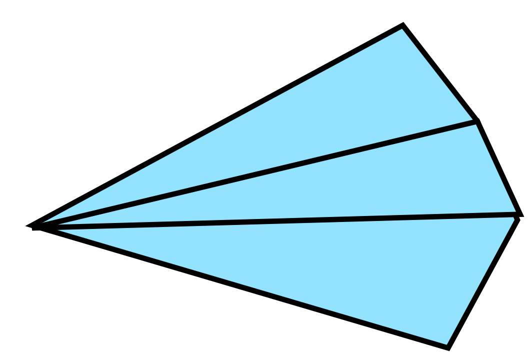
GL_POINTS



GL_TRIANGLES

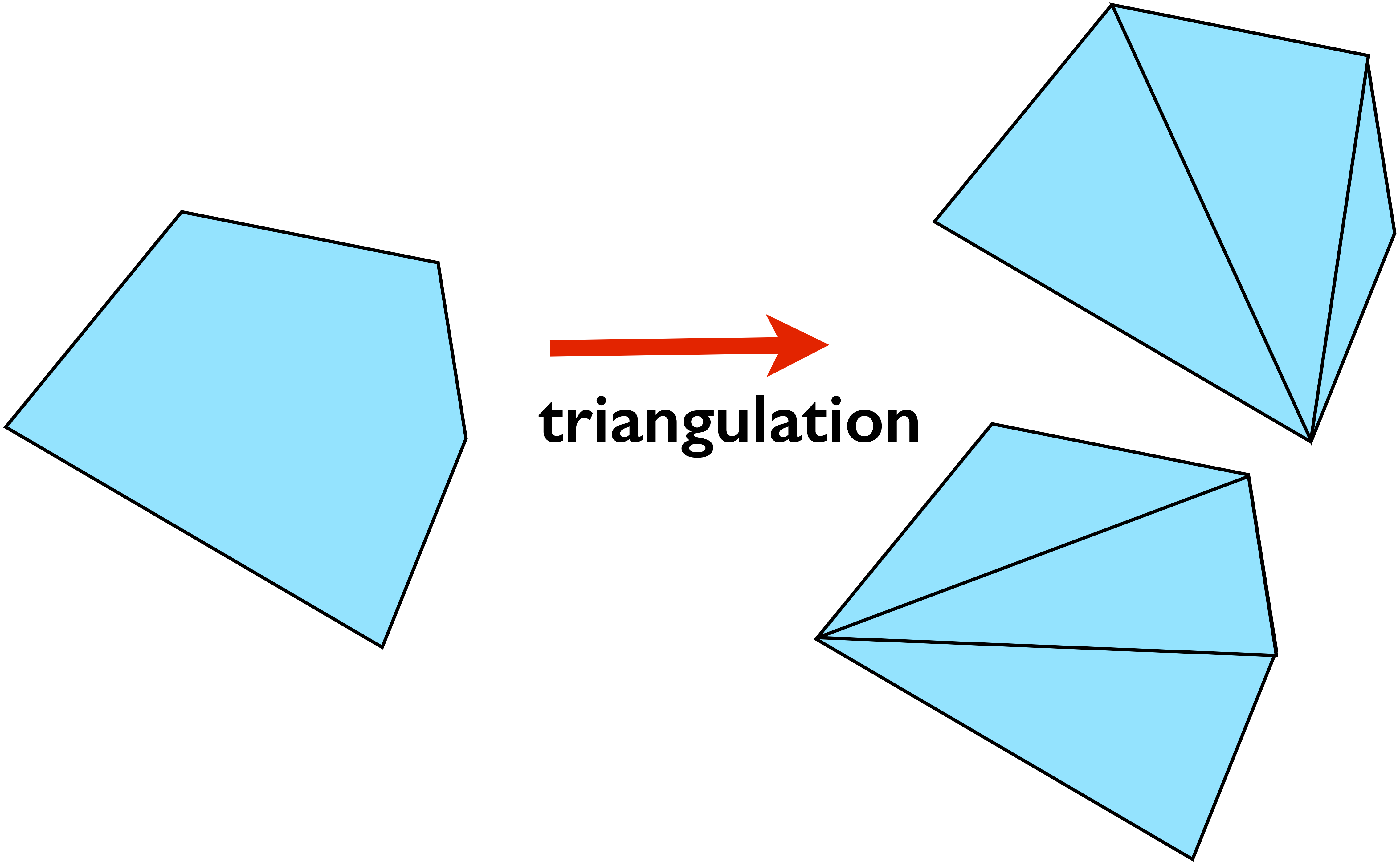


GL_TRIANGLE_STRIP

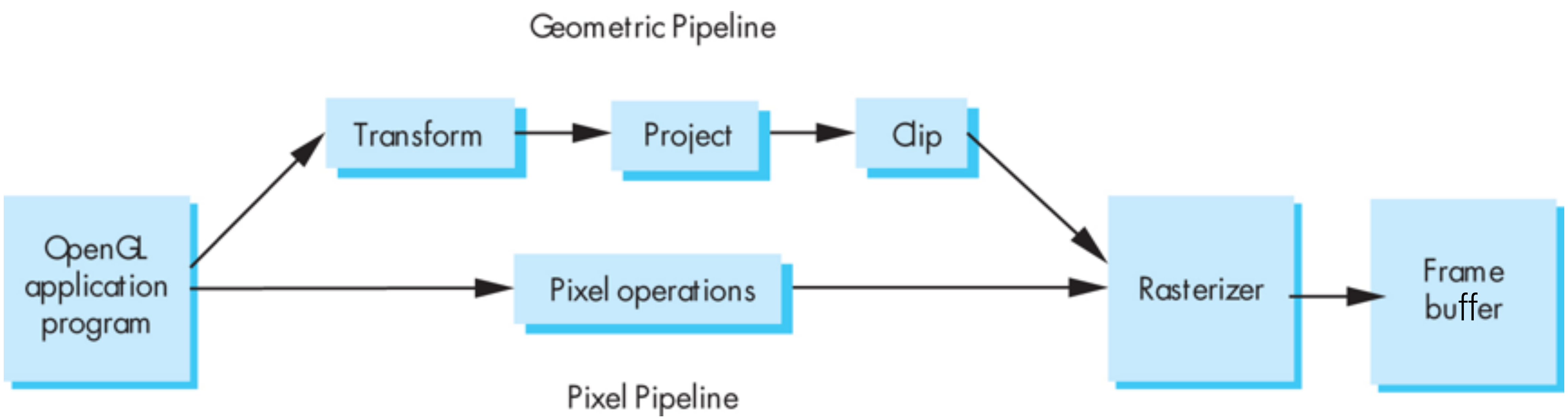


GL_TRIANGLE_FAN

# Other polygons



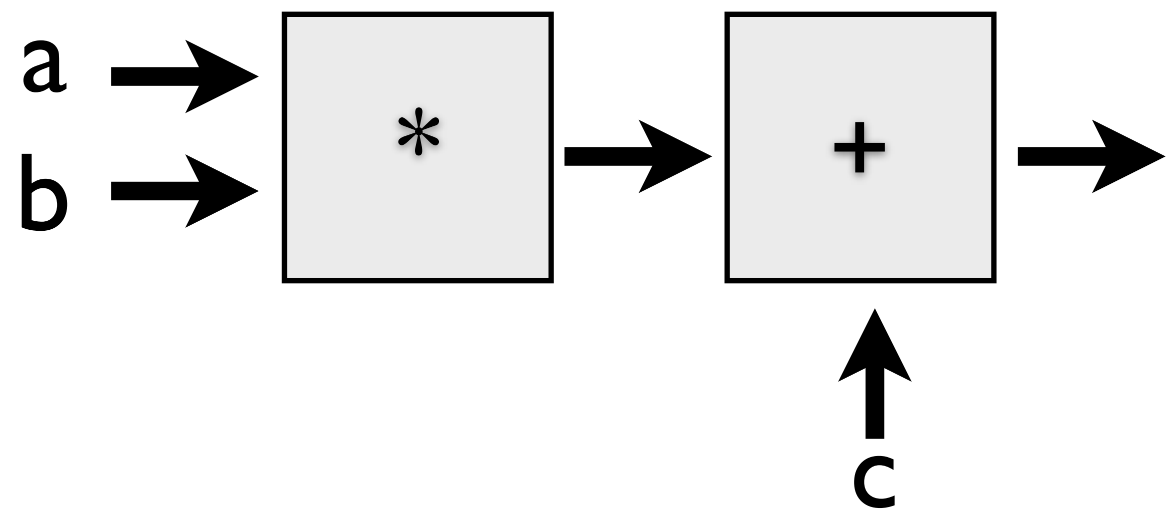
# Graphics Pipeline

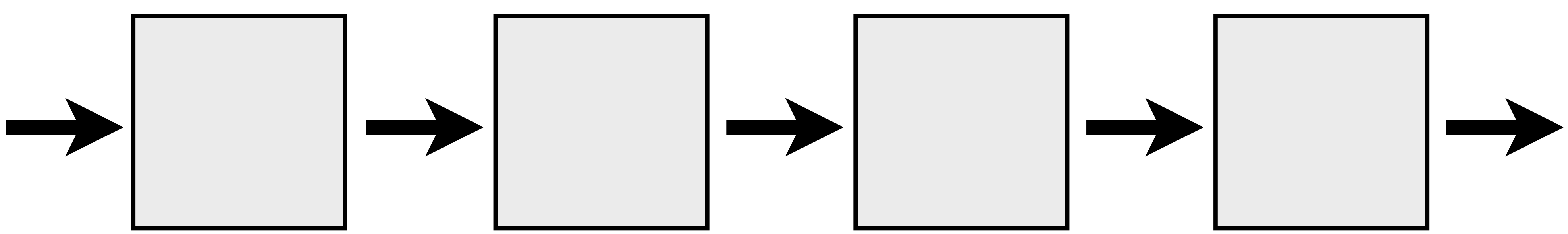



[Angel and Shreiner]

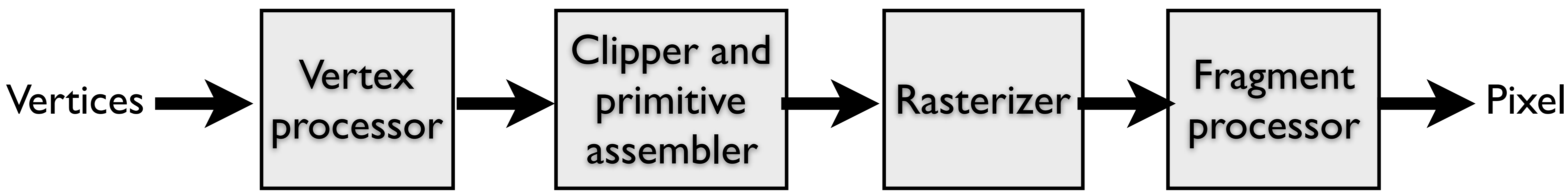
# Pipelining operations

An arithmetic pipeline that computes c+(a*b)



# 3D graphics pipeline



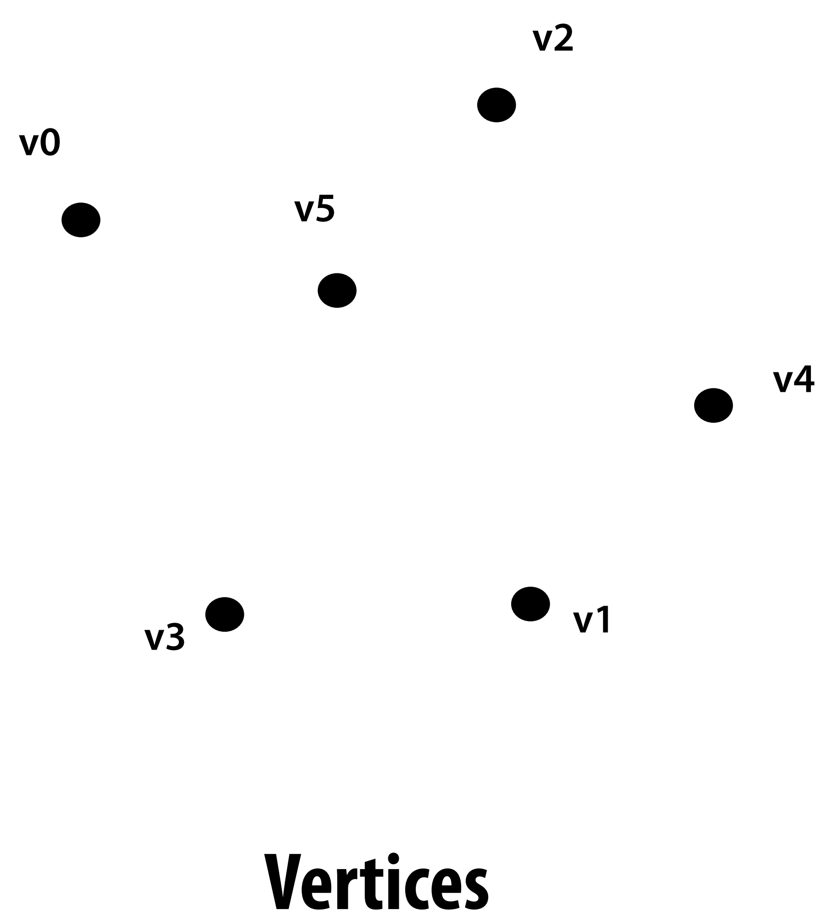

**Geometry**: primitives – made of vertices
**Vertex processing:** coordinate transformations and color
**Clipping and primitive assembly:** output is a set of primitives
**Rasterization:** output is a set of fragments for each primitive
**Fragment processing:** update pixels in the frame buffer

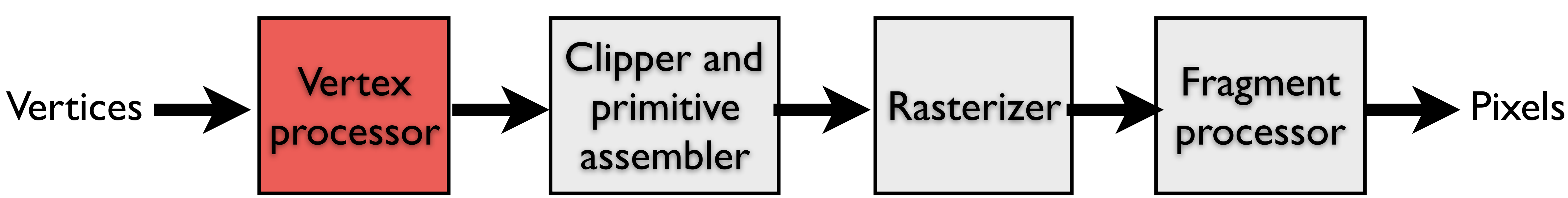
# Graphics Pipeline

(slides courtesy K. Fatahalian)

# Vertex processing

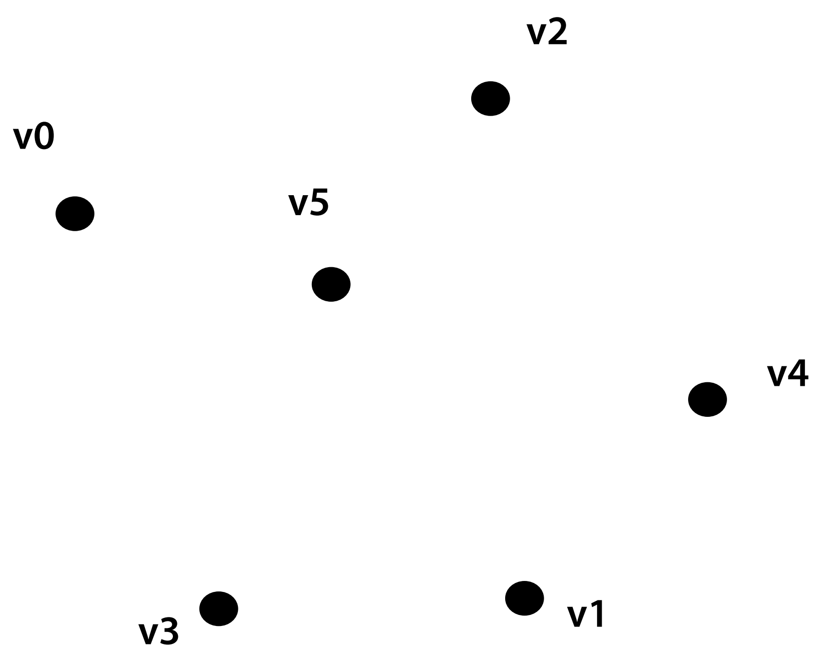
## Vertices are transformed into "screen space"



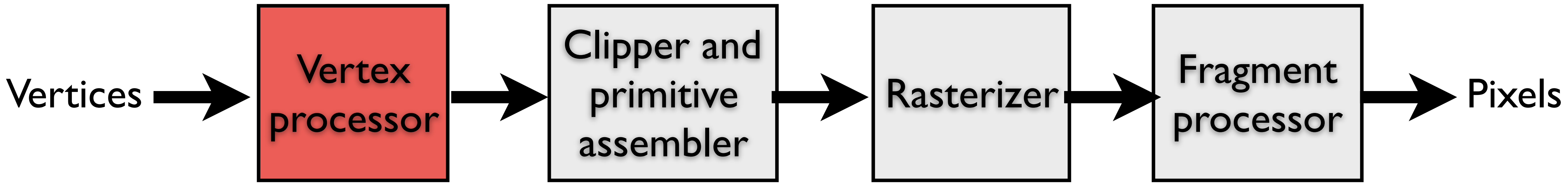
# Vertex processing

## Vertices are transformed into "screen space"



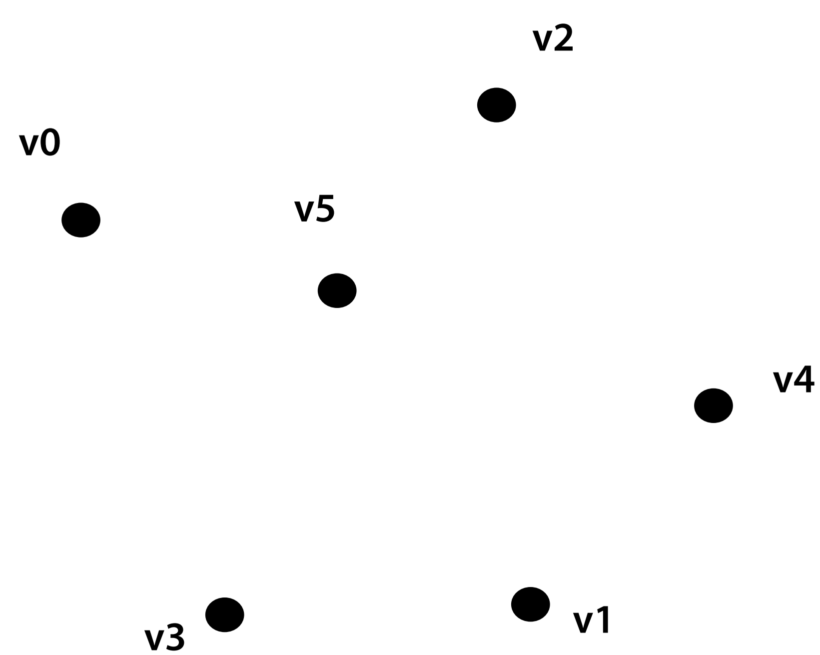

**EACH VERTEX IS TRANSFORMED INDEPENDENTLY**

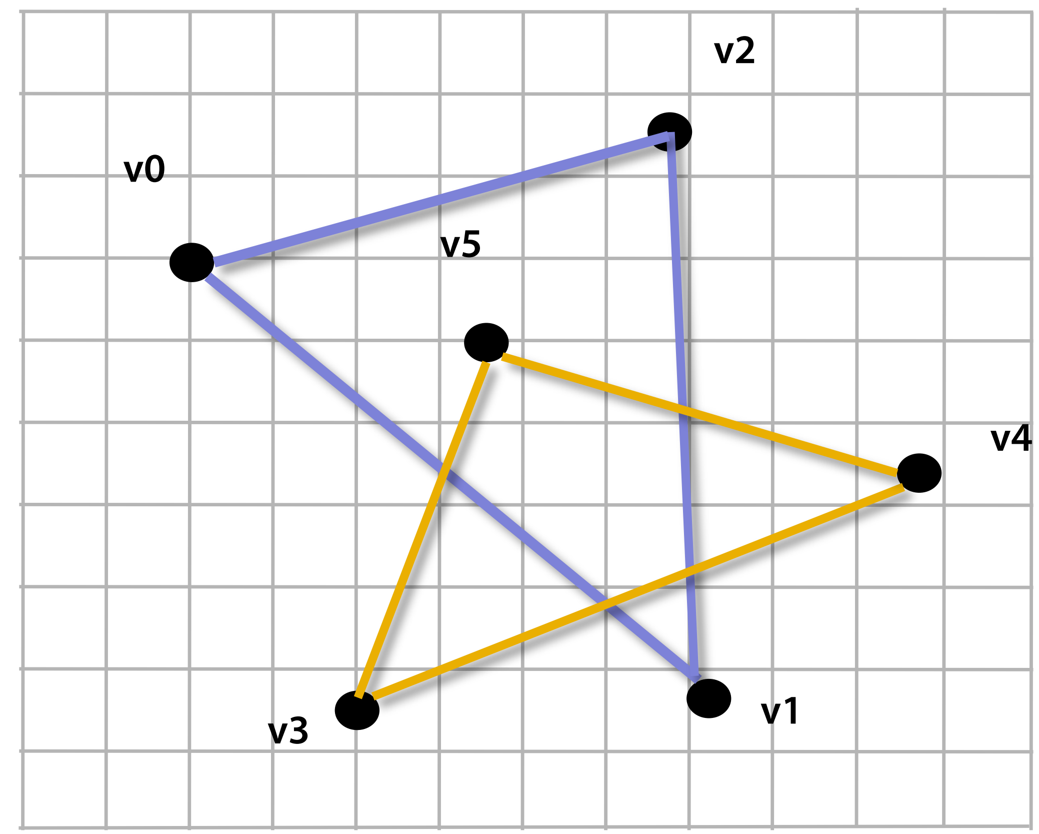
**Vertices**



# Primitive processing

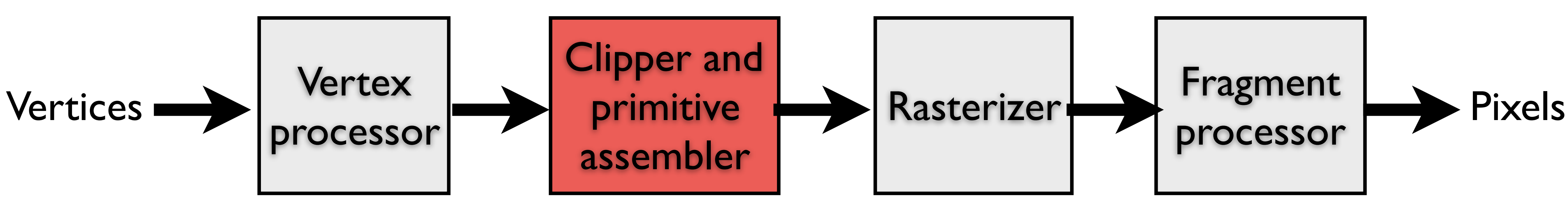
## Then organized into primitives that are clipped and culled…



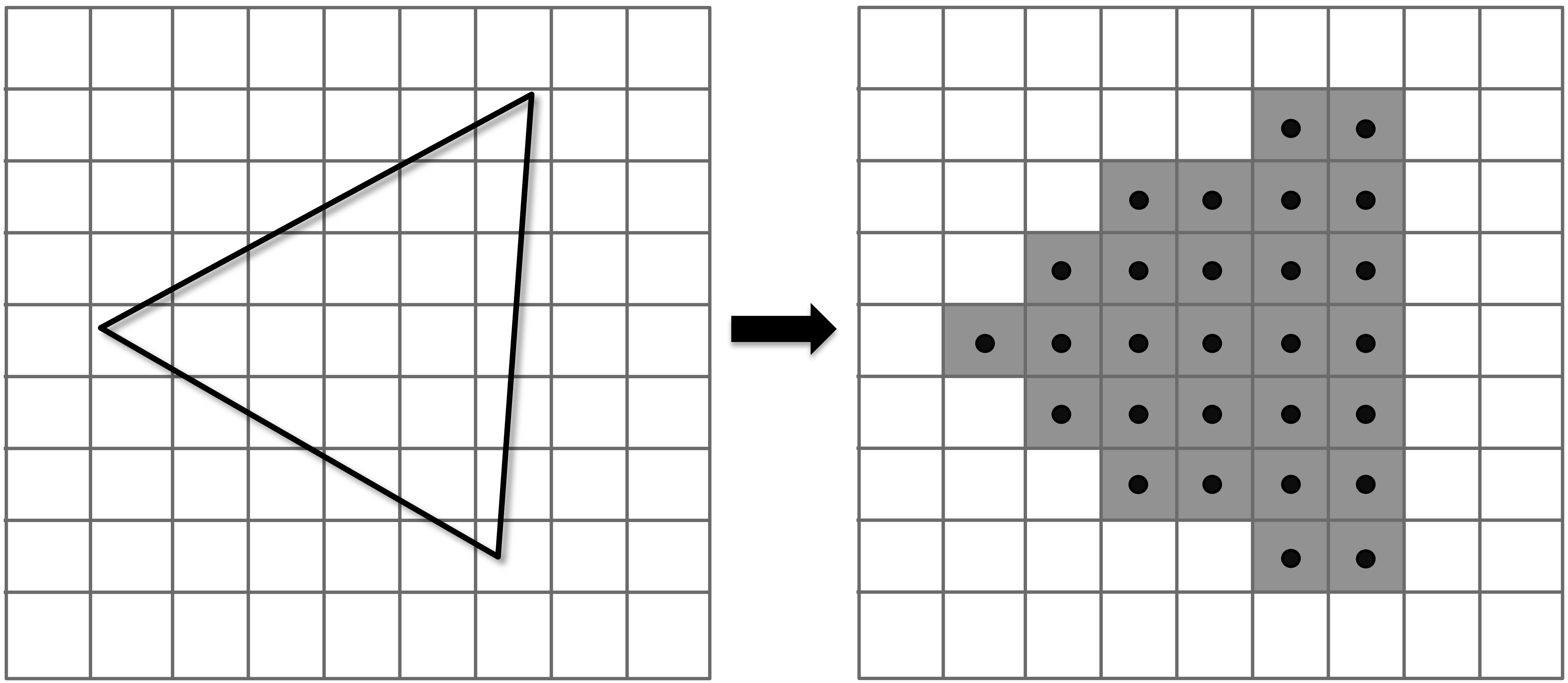

**Vertices**

**Primitives (triangles)**

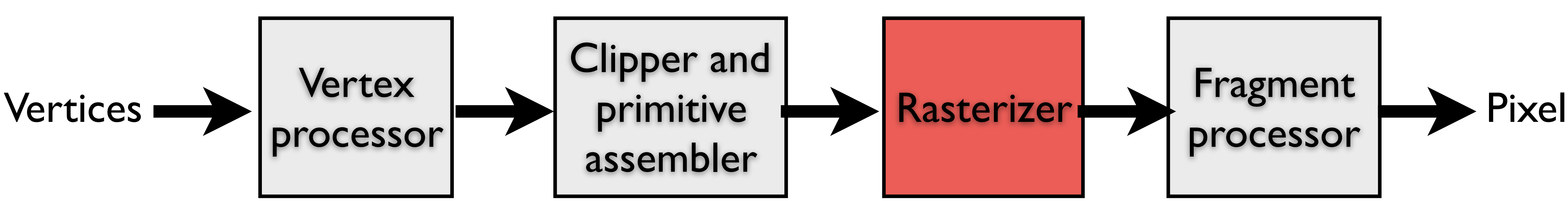


# Rasterization

## Primitives are rasterized into "pixel fragments"

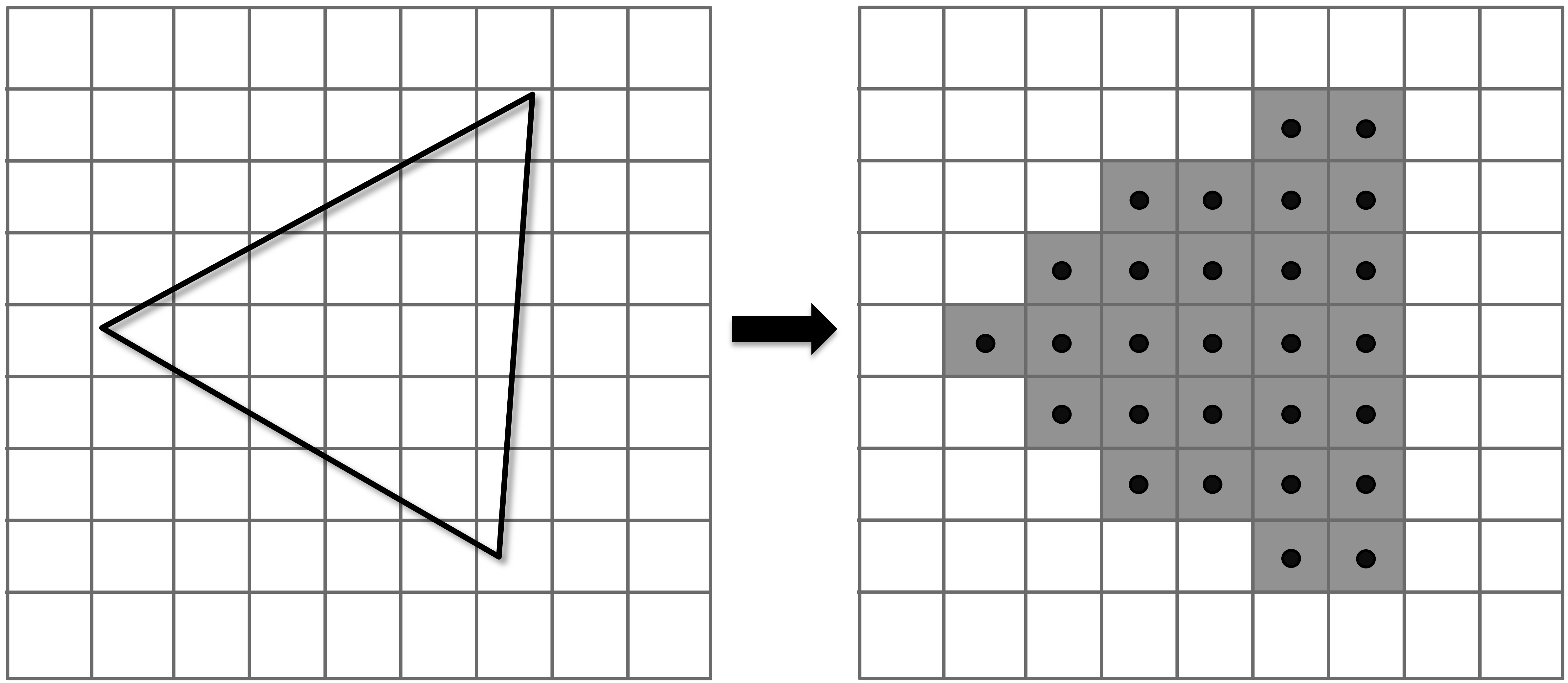


Fragments

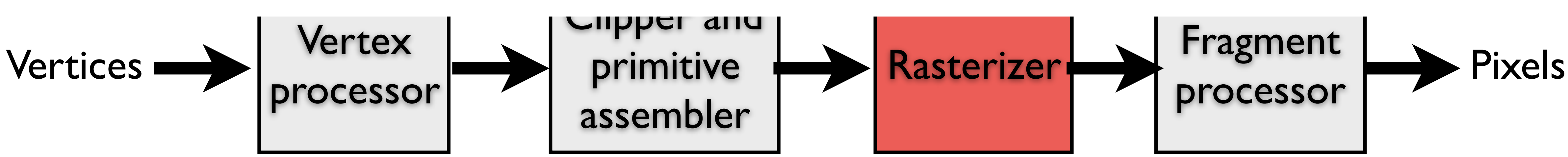


# Rasterization

## Primitives are rasterized into "pixel fragments"

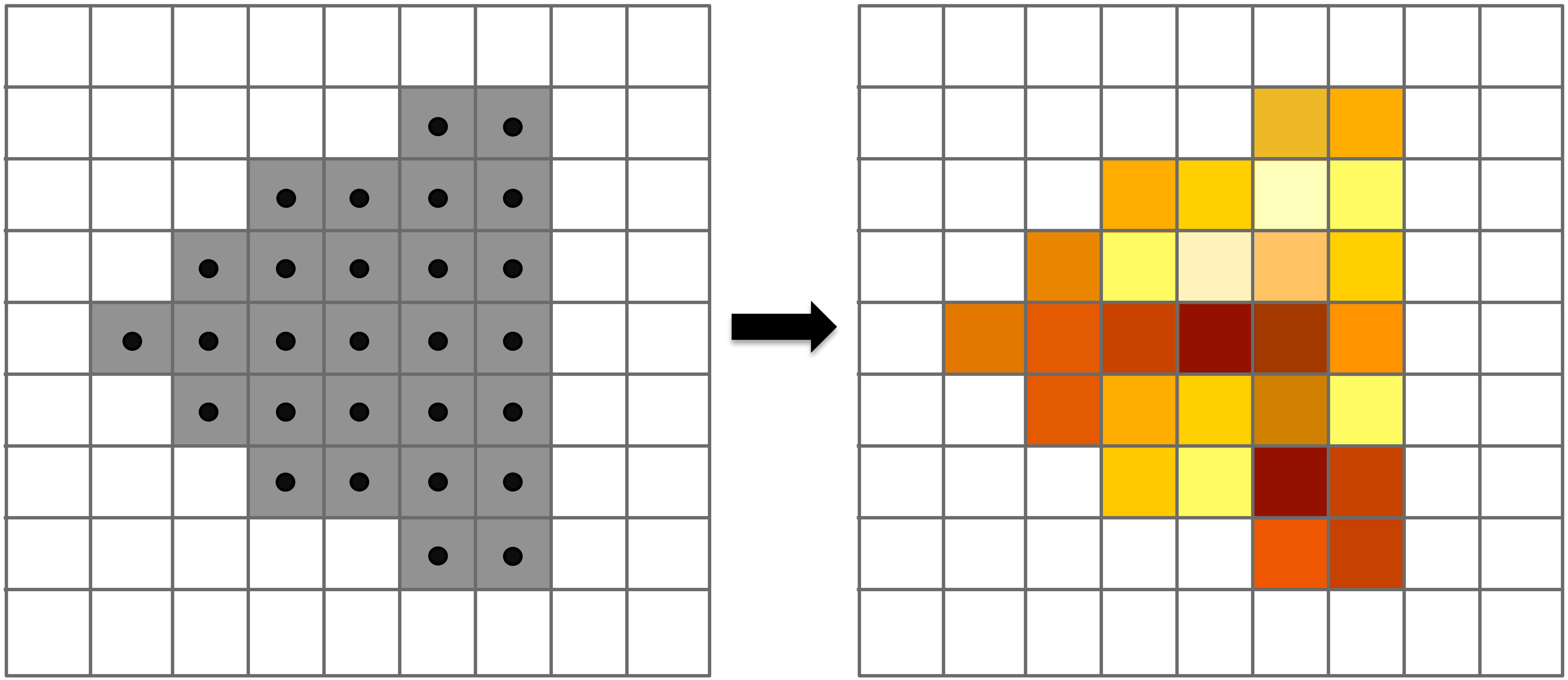


**EACH PRIMITIVE IS RASTERIZED INDEPENDENTLY**

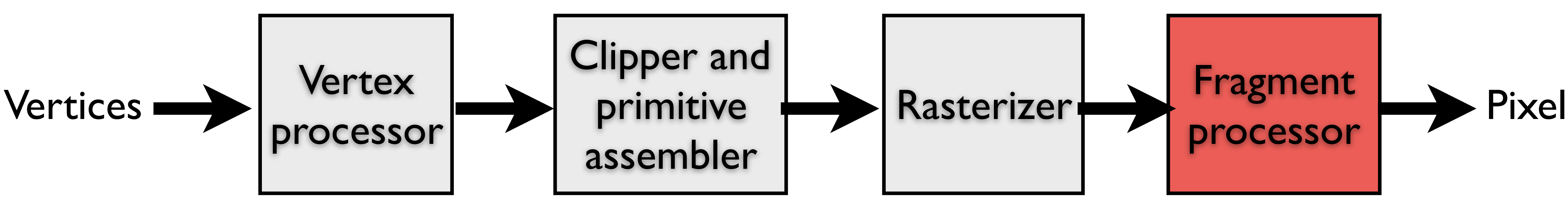


# Fragment processing

## Fragments are shaded to compute a color at each pixel

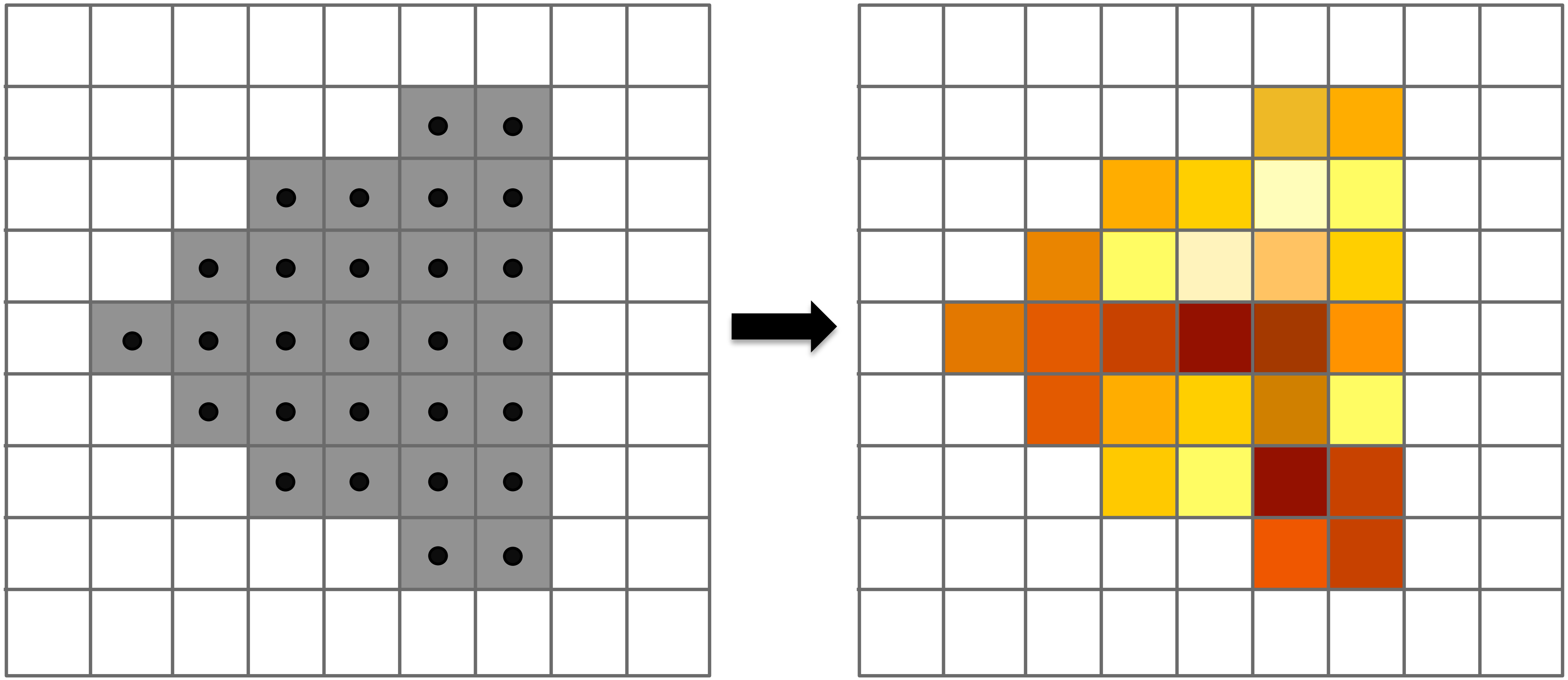


**Shaded fragments**

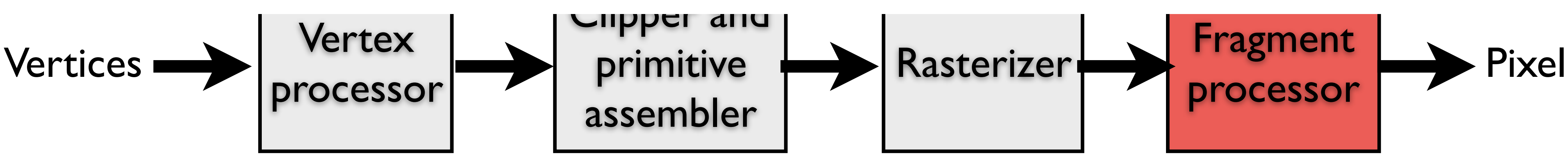


# Fragment processing

## Fragments are shaded to compute a color at each pixel

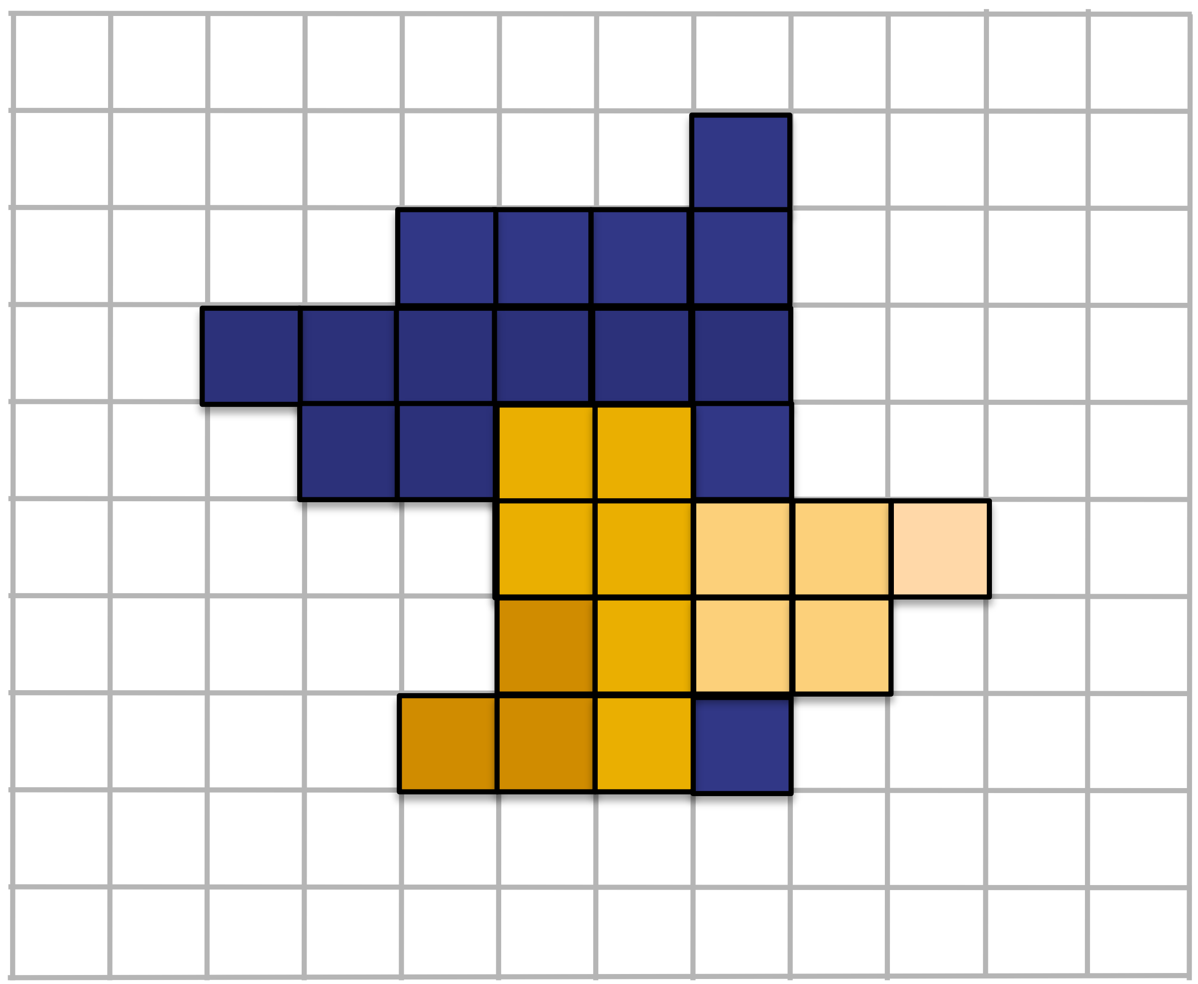


**EACH FRAGMENT IS PROCESSED INDEPENDENTLY**



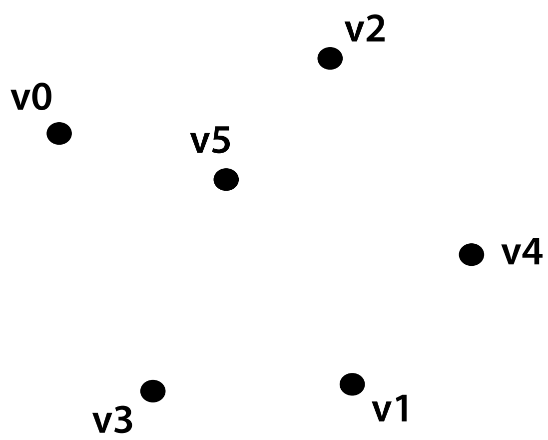
# Pixel operations

## Fragments are blended into the frame buffer at their pixel locations (z-buffer determines visibility)

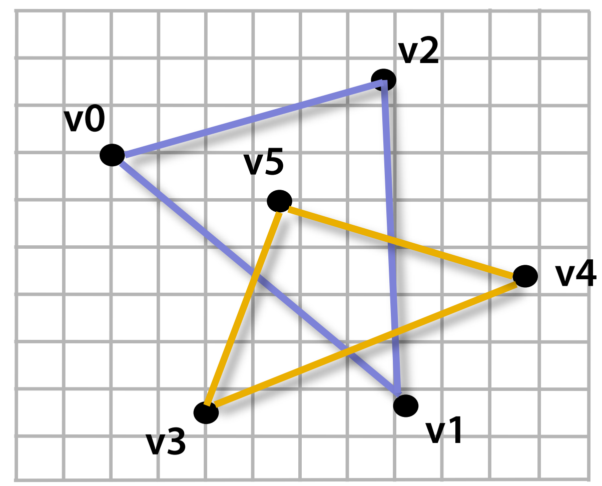


**Pixels**

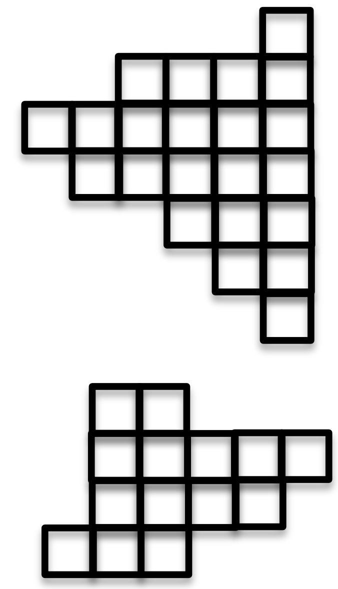
# Pipeline entities



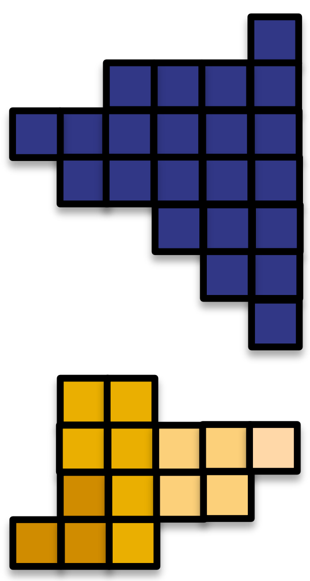

**Vertices**



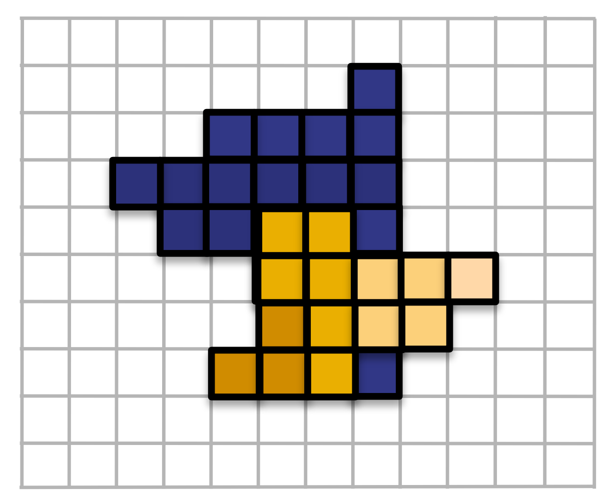

**Primitives**



**Fragments**



**Fragments (shaded)**



**Pixels**

# Graphics pipeline