CS130 : Computer Graphics

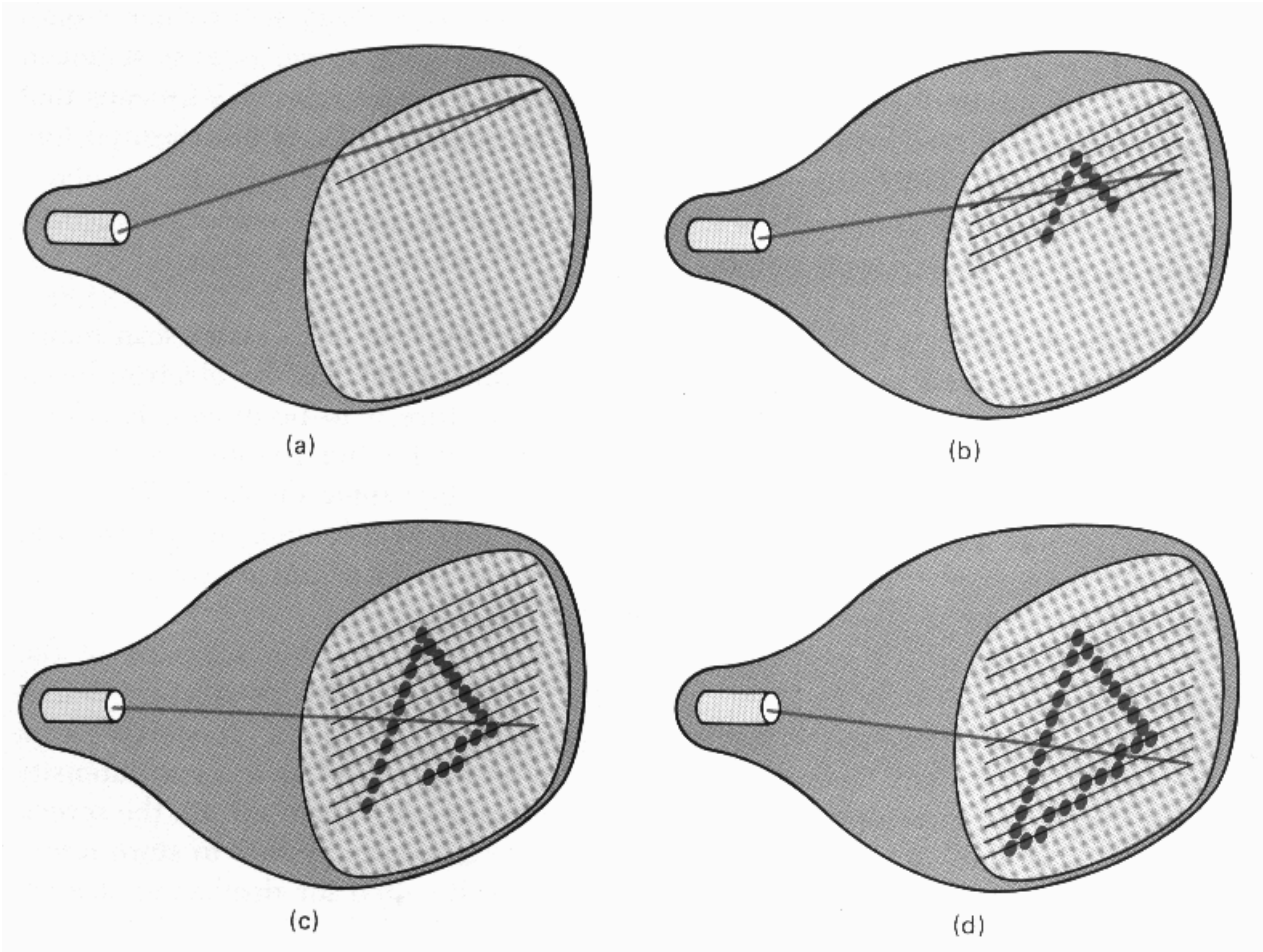
Lecture 2: Graphics Pipeline

Tamar Shinar
Computer Science & Engineering
UC Riverside

Raster Devices and Images

Raster Devices

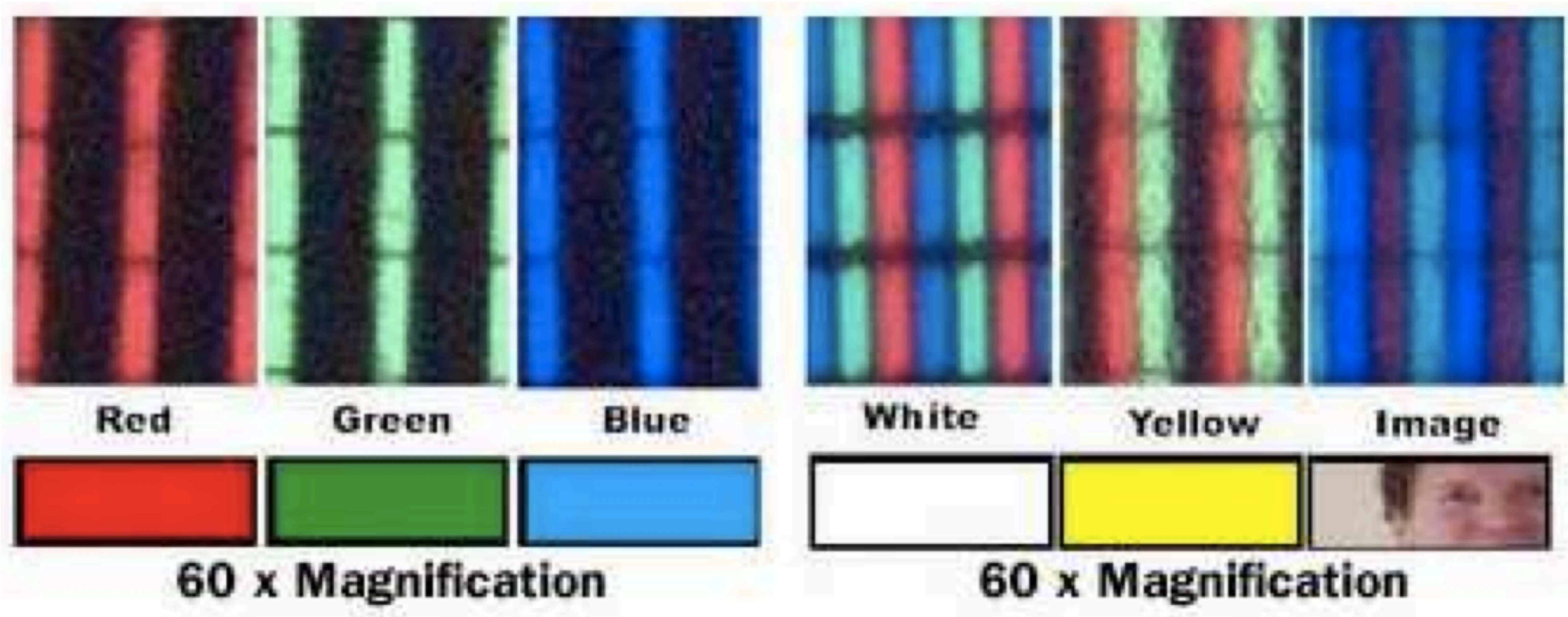
Raster Display



Hearn, Baker, Carithers

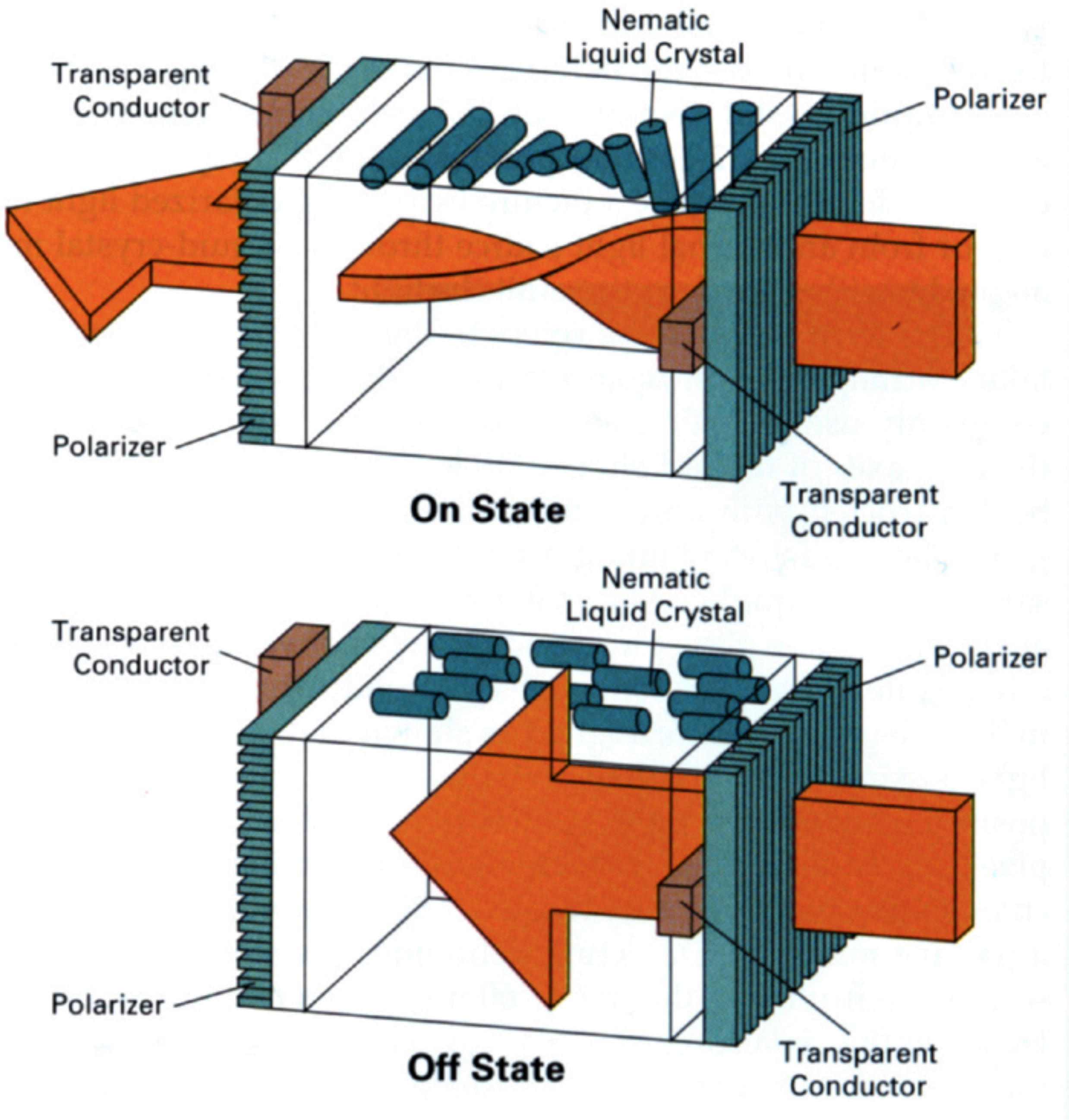
virtually all graphics system are **raster based,** meaning the image we see is a **raster of pixels**
or a rectangular array of pixels
Here a raster scan device display an image as a set of discrete points across each scanline

Raster Display

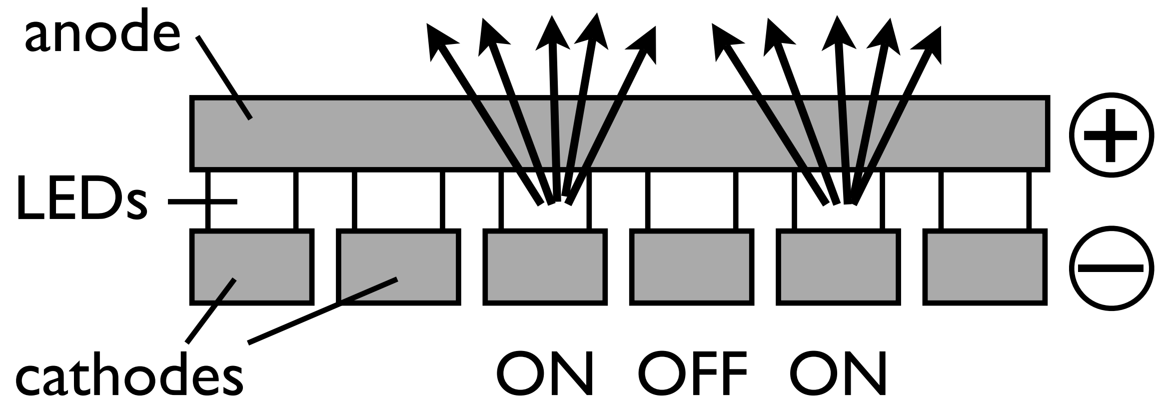


get different colors by mixing red, green, and blue
this is from an LCD monitor
printers are also raster-based. image is made out of points on a grid

Transmissive vs. Emissive Display



[H&B, Fig. 2-16]

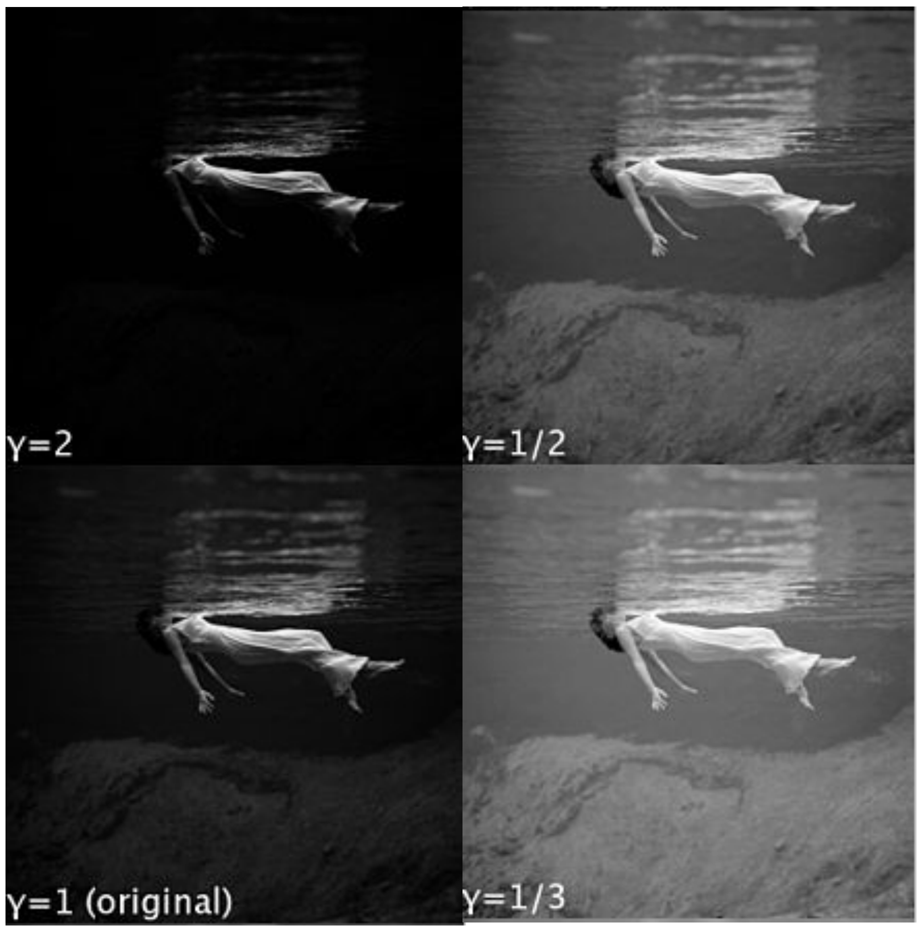


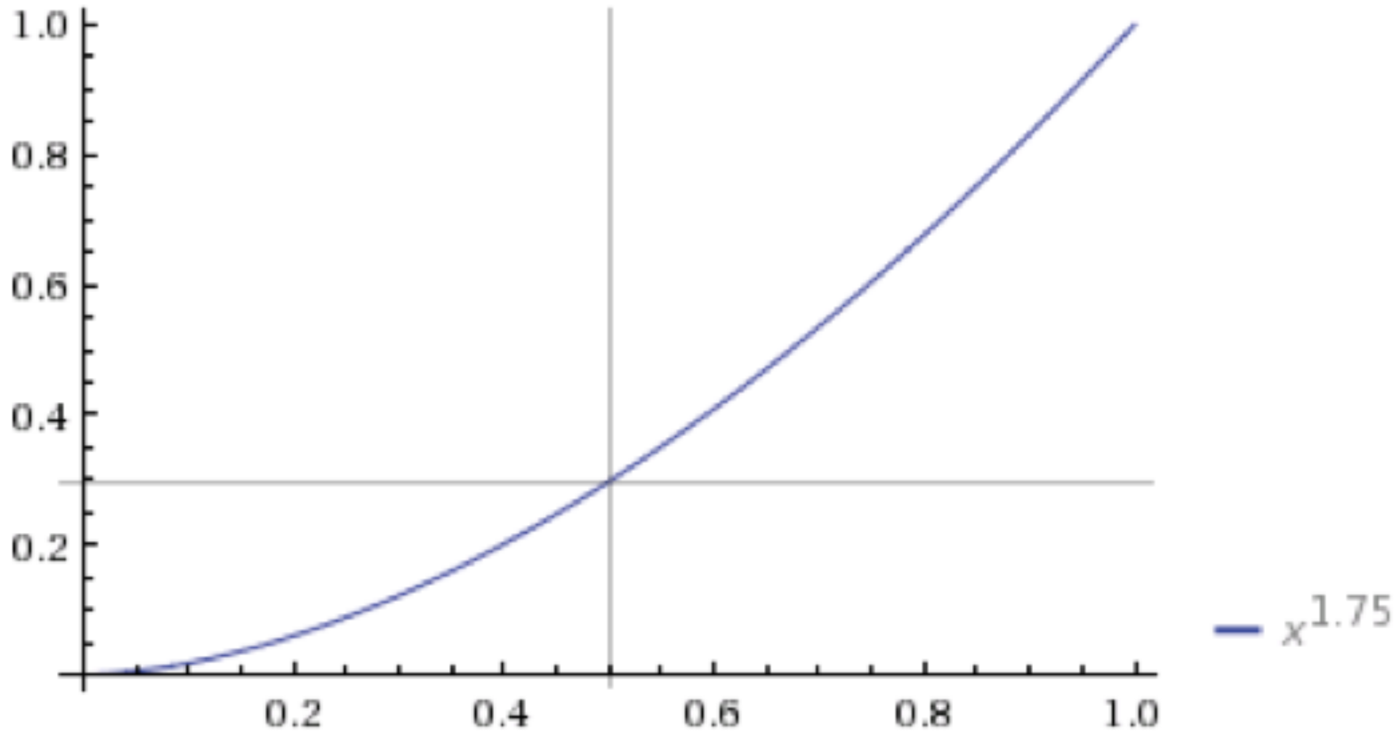
(LEFT)In the **off state** the front polarizer blocks all the light that passes the back polarizer in the **on state** the liquid crystal rotates the polarization of the light so it can pass through the front polarizer

(RIGHT) LED display

Monitor Gamma

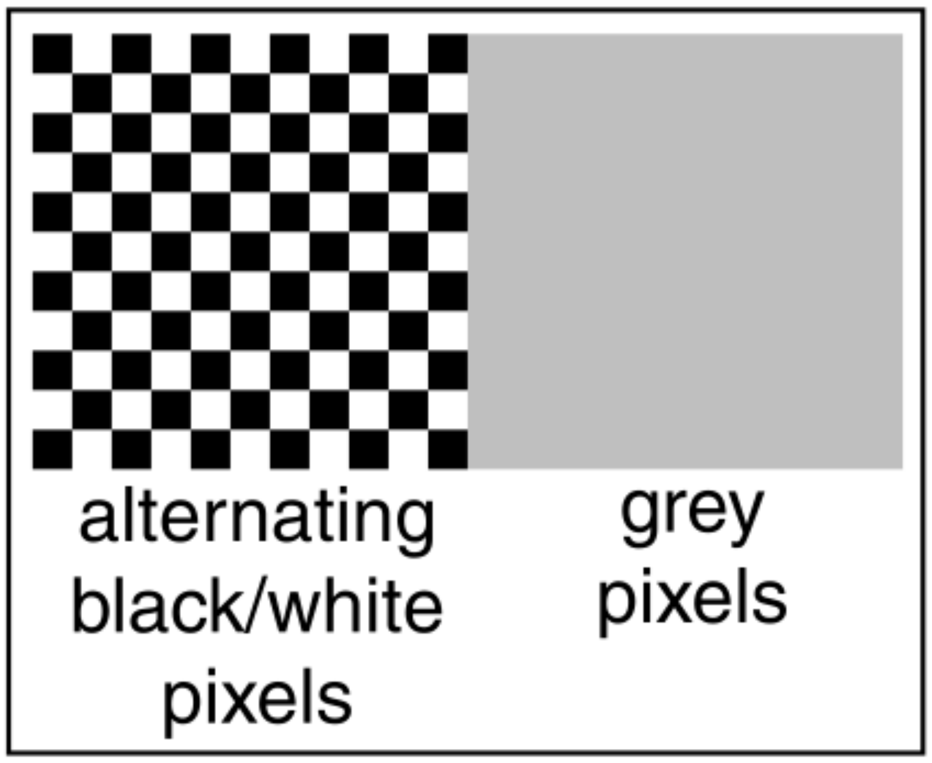
displayed intensity = (max intensity) a^{γ}

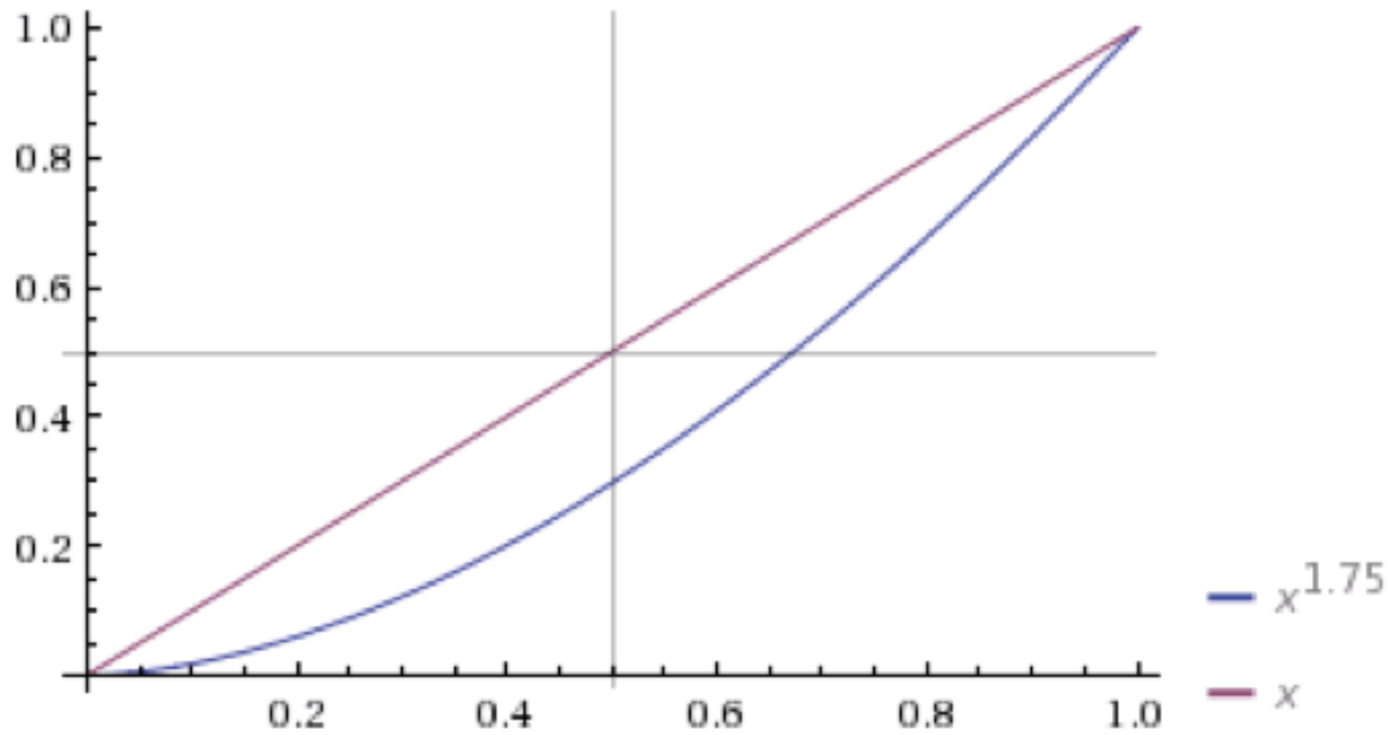




Gamma Correction

displayed intensity = (max intensity) a^γ

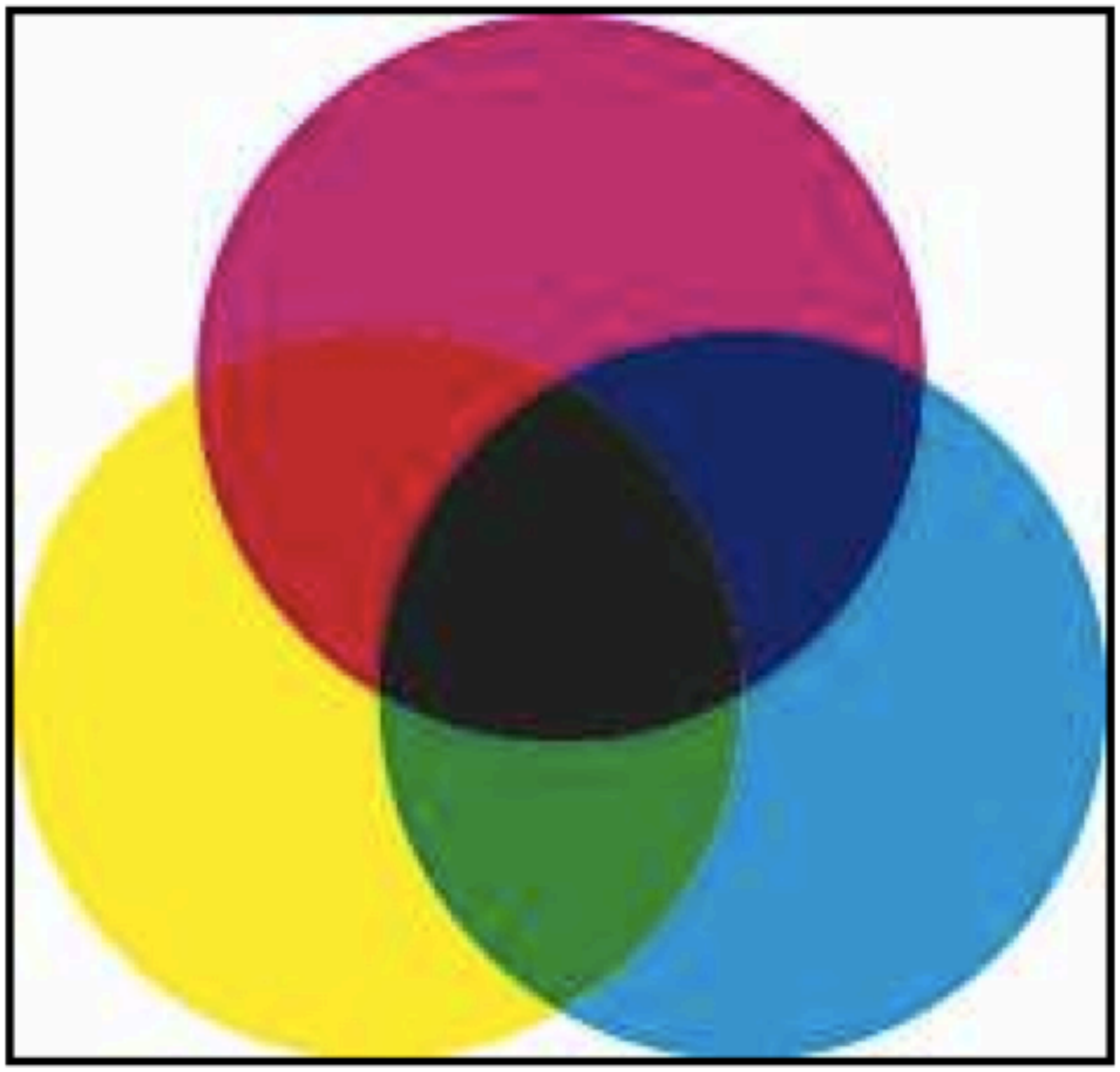




find gamma, so that you can give the monitor a^{1/\gamma}
- find a such that a^{\gamma} = .5 through checkboard test and solve for gamma

Color representation

additive

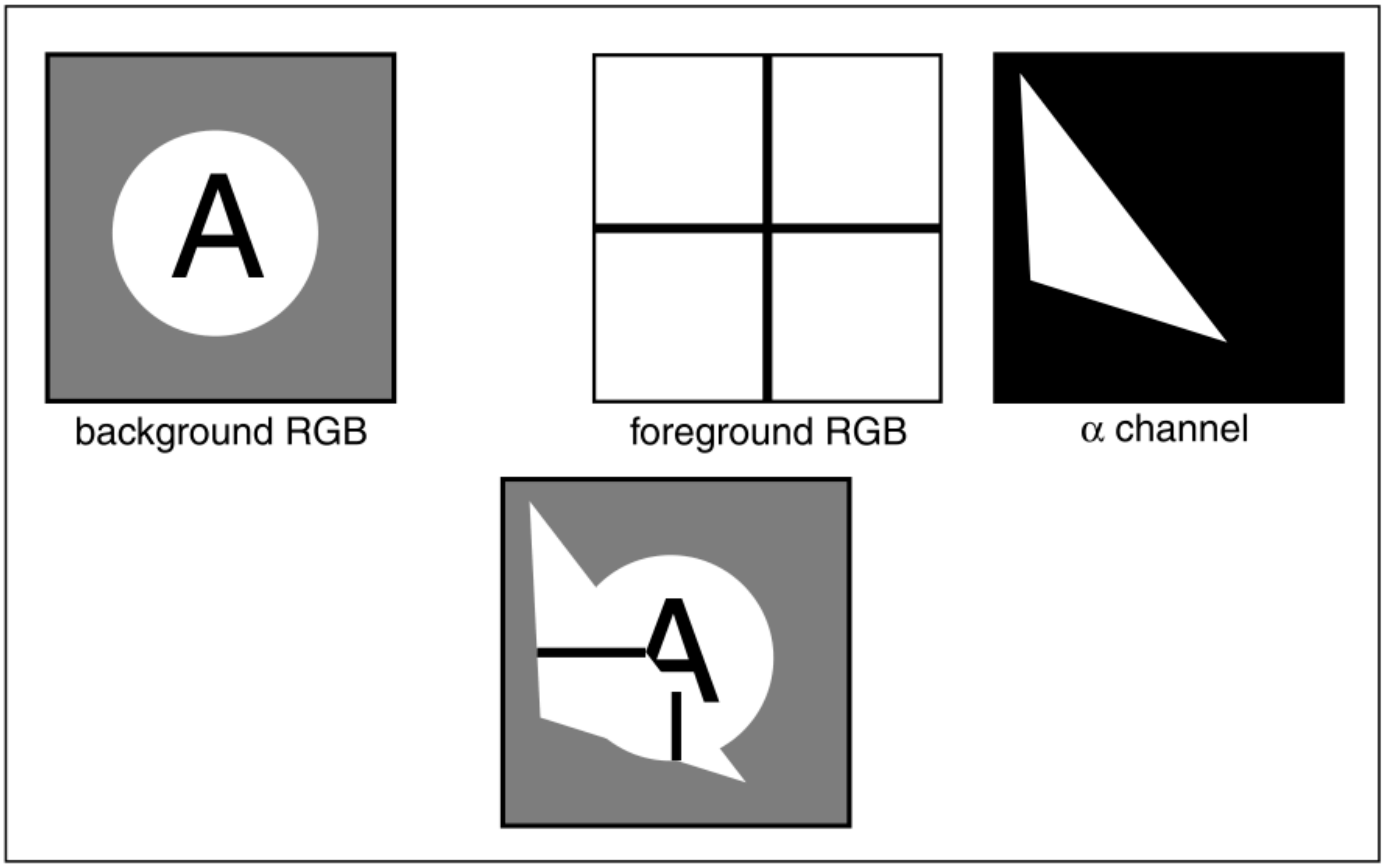


subtractive

additive color – Primary colors are red, green, blue. form a color by adding these. CRTs, projectors, LCD displays, positive film
subtractive color – form a color by filtering white light with cyan, magenta, and yellow filters printing, negative film

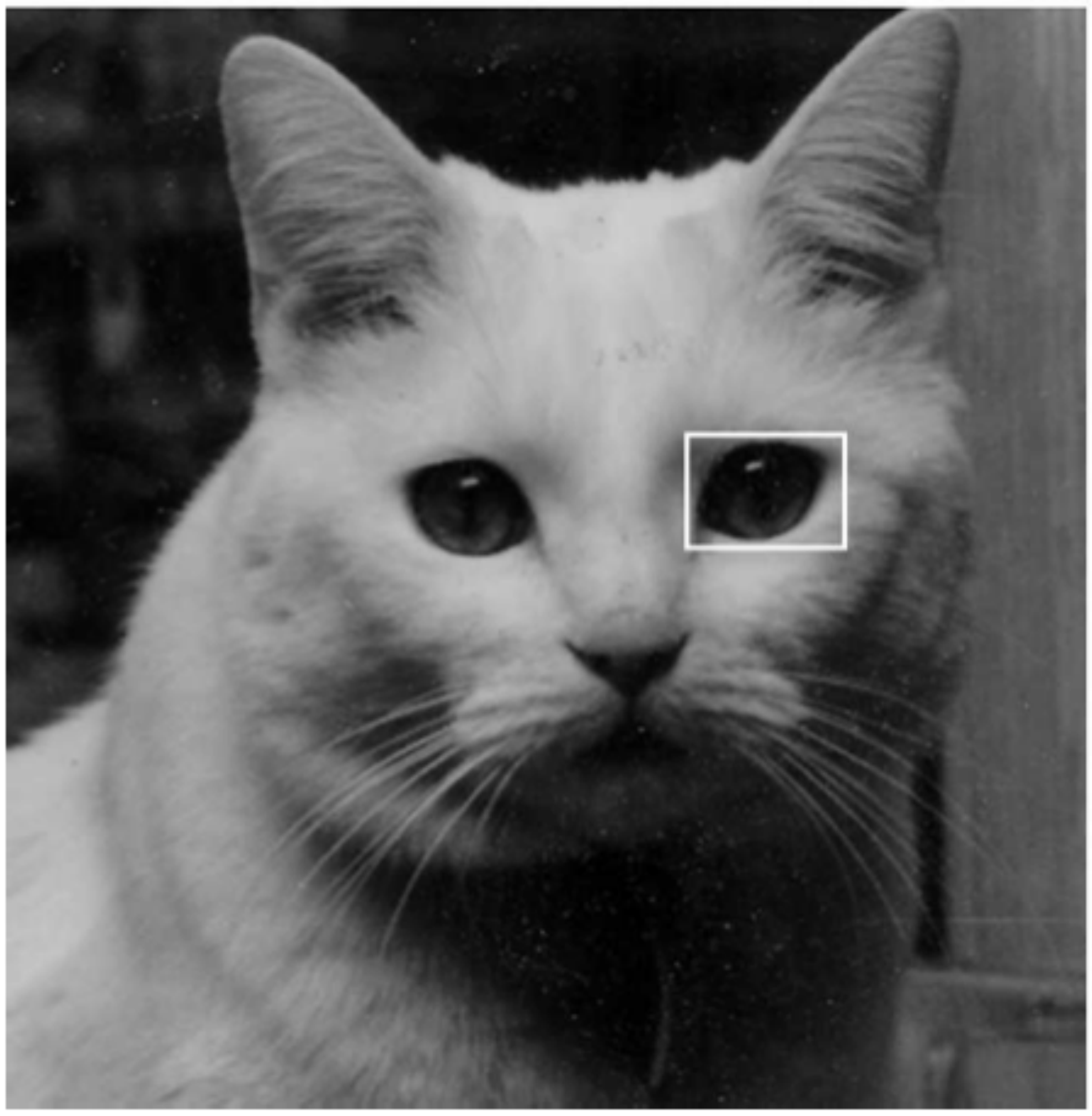
Alpha Channel

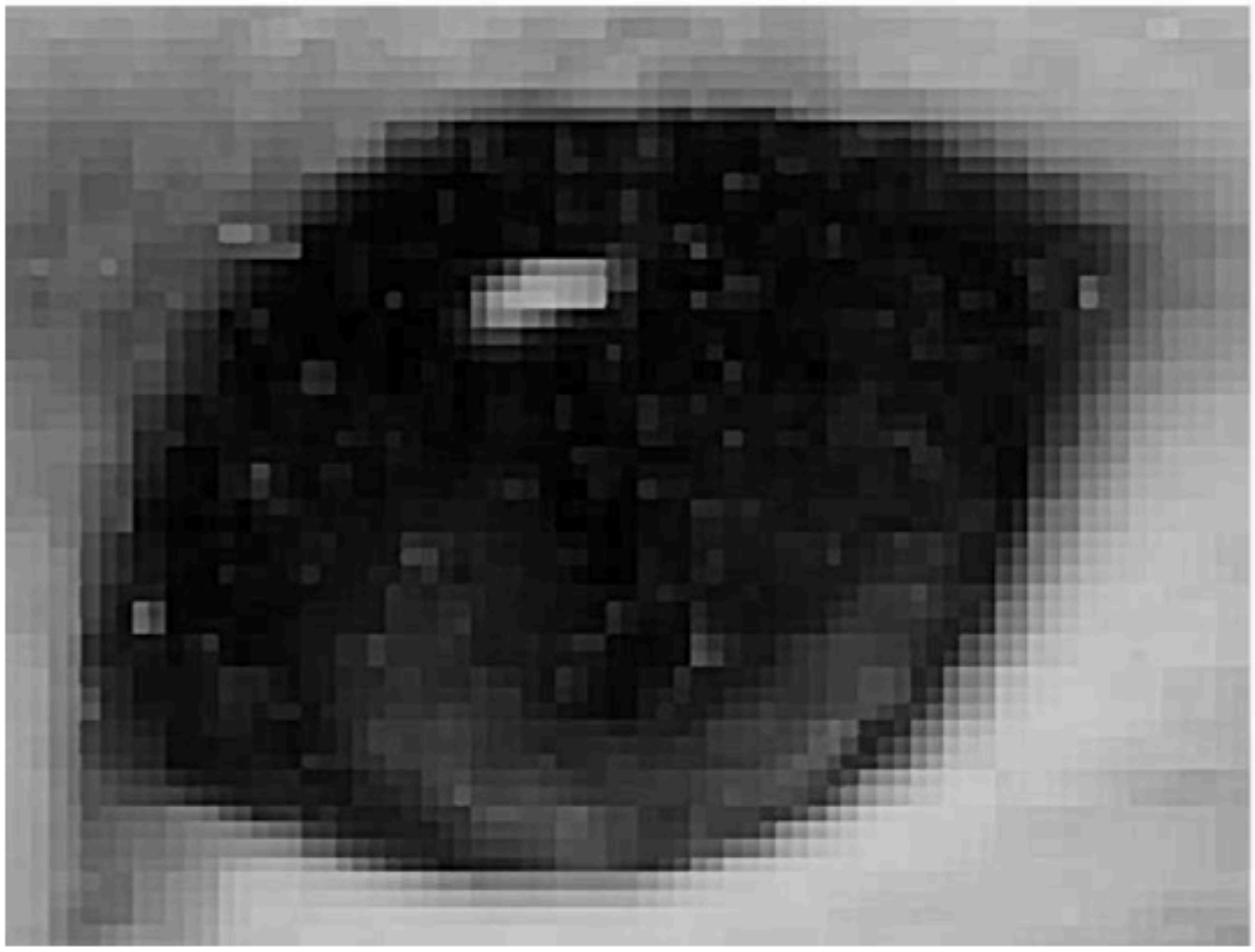
$$\mathbf{c} = \alpha \mathbf{c}_f + (1 - \alpha)\mathbf{c}_b$$



Compositing: two different interpretations: **pixel coverage** (fraction of pixel covered) and **blending**

Raster Image





A raster image is 2D array storing pixel values at each pixel (picture element)
3 numbers for color
alternative: vector image -- essentially a set of instructions for rendering an image

What is an image?

Continuous image

$$I : R \to V$$

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

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

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

What is an image?

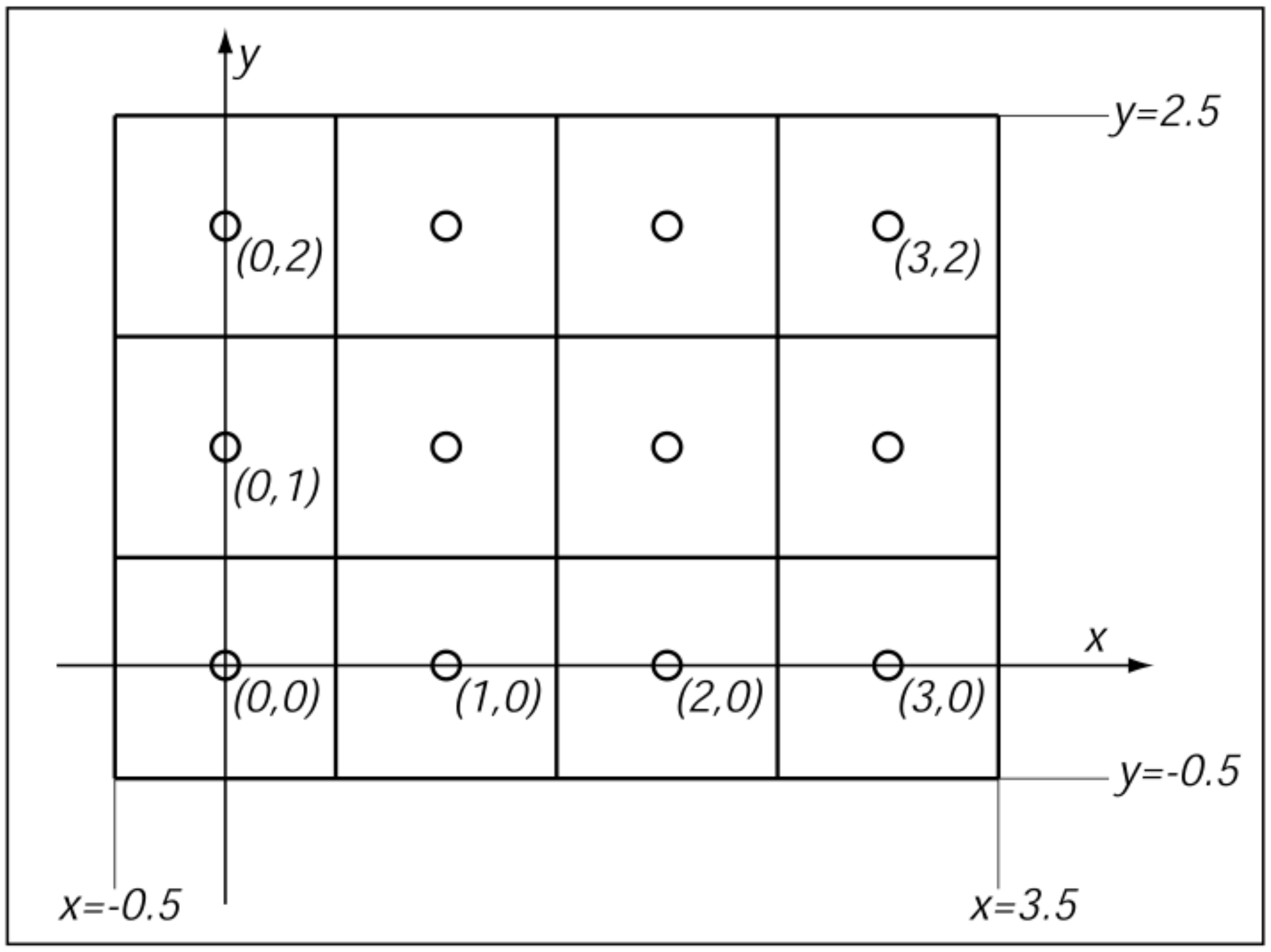
Sampled image

$$I : R \to V$$

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

$V = [0, 1]$ (grayscale)

$V = [0, 1]^3$ (color)



Bit depth - defined by device standards

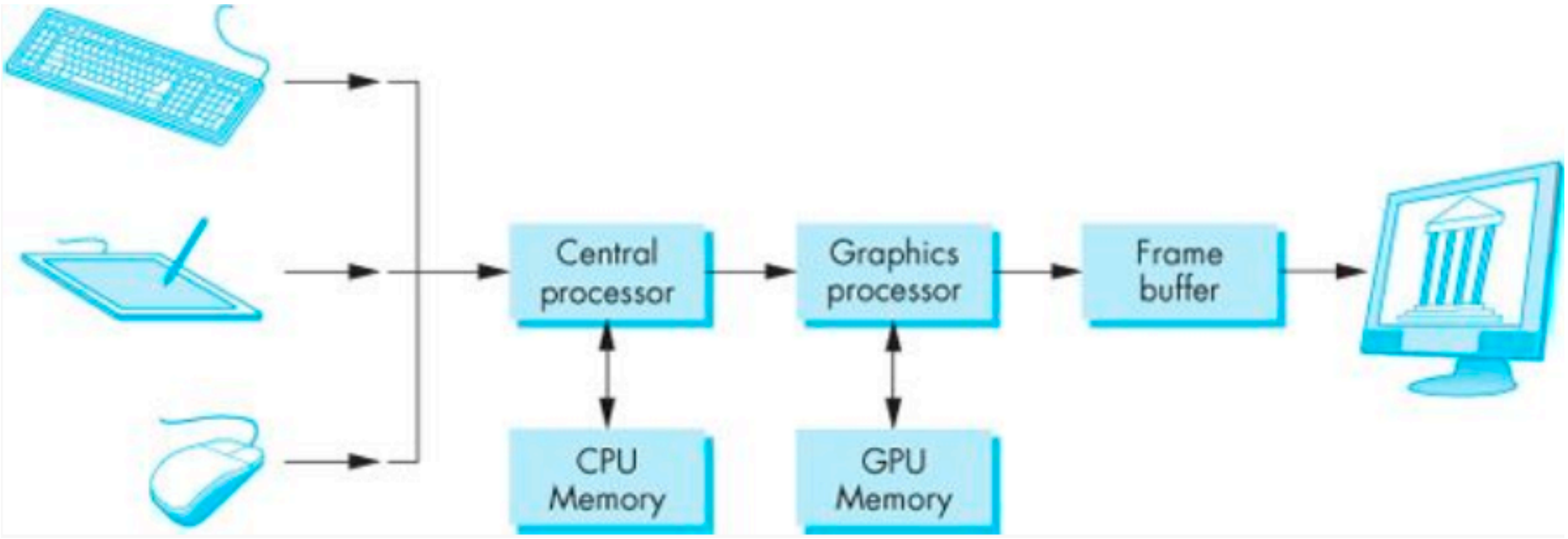
Bit-Depth	Number of Colors
1	2 (monochrome)
2	4 (CGA)
4	16 (EGA)
8	256 (VGA)
16	65,536 (High Color, XGA)
24	16,777,216 (True Color, SVGA)
32	16,777,216 (True Color + Alpha Channel)

(Note alpha)

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

Graphics Pipeline

Modern graphics system



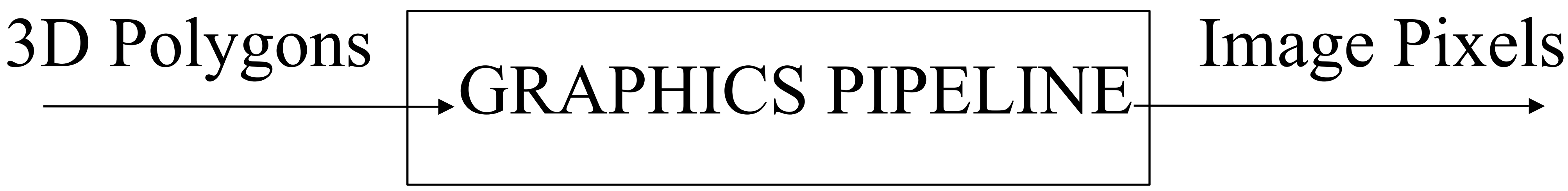
[Angel and Shreiner]

the pixels are stored in a location in memory call the **frame buffer**
frame buffer resolution determines the details in the image
- e.g., 24 bit color “full color”
- high dynamic range or HDR use 12 or more bits for each color

frame buffer = color buffers + other buffer

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

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?

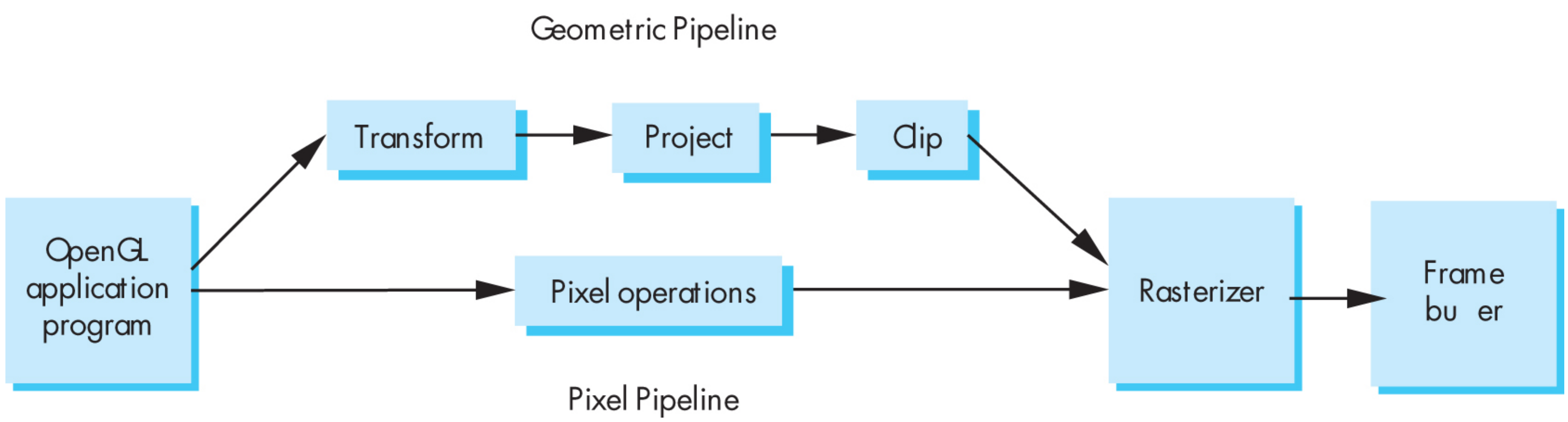
➡**small set - supported by hardware**

- lots of primitives - convenient for user

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

Other geometric shapes will be built out of these

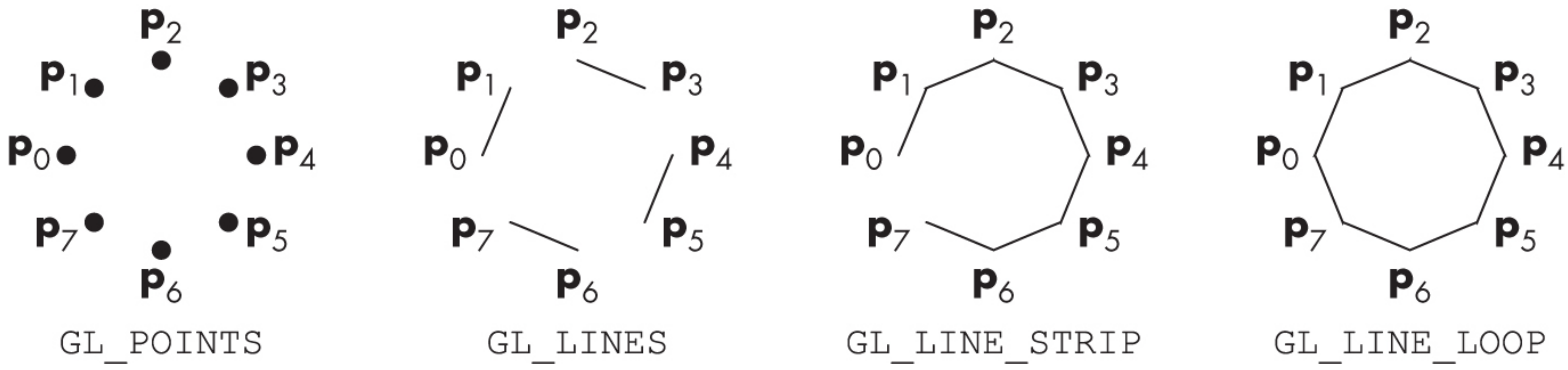
Two classes of primitives



[Angel and Shreiner]

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

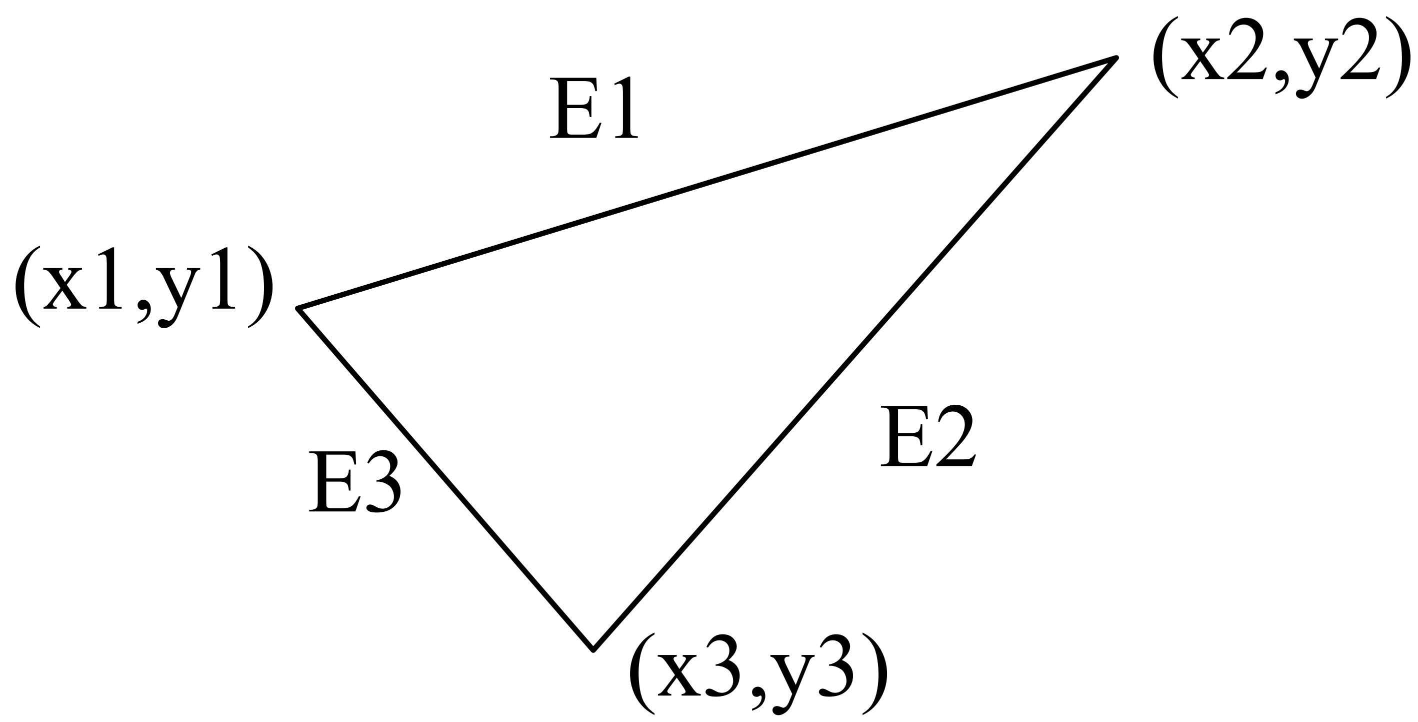
Point and line segment types



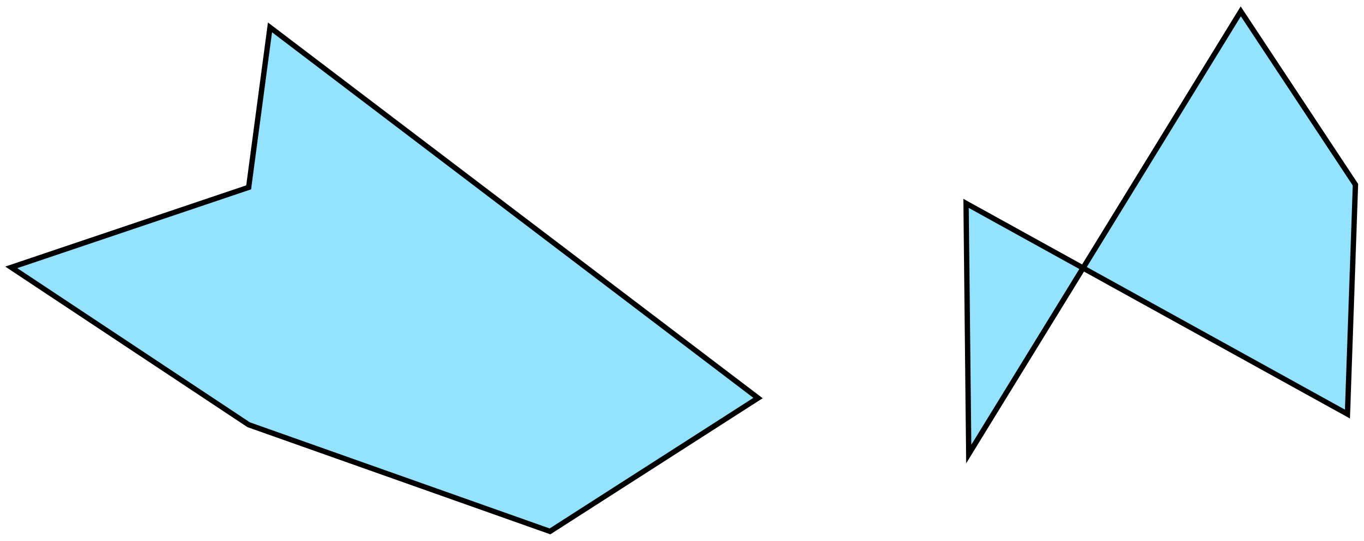
[Angel and Shreiner]

Polygons

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

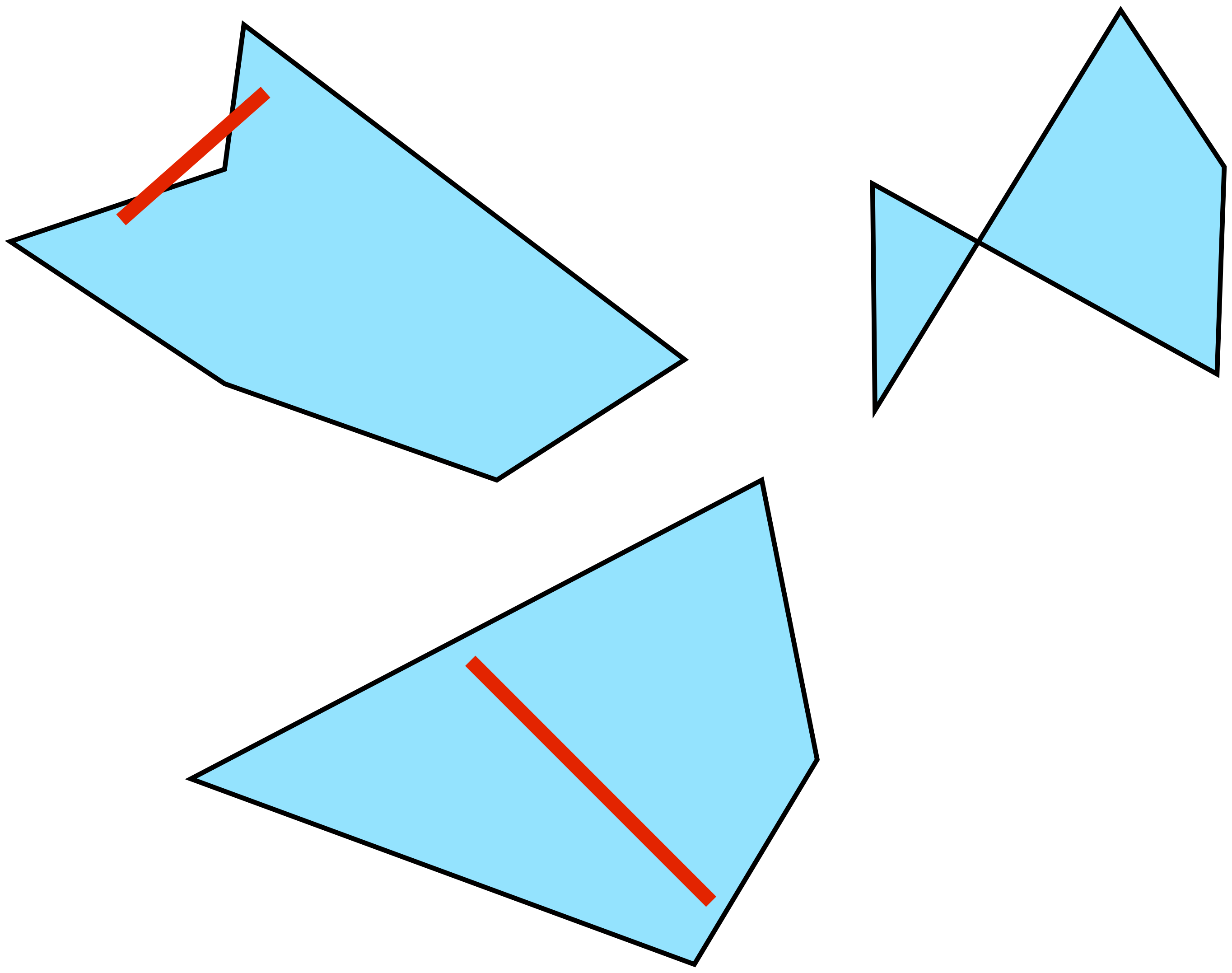


Valid polygons



- Simple
- Convex
- Flat

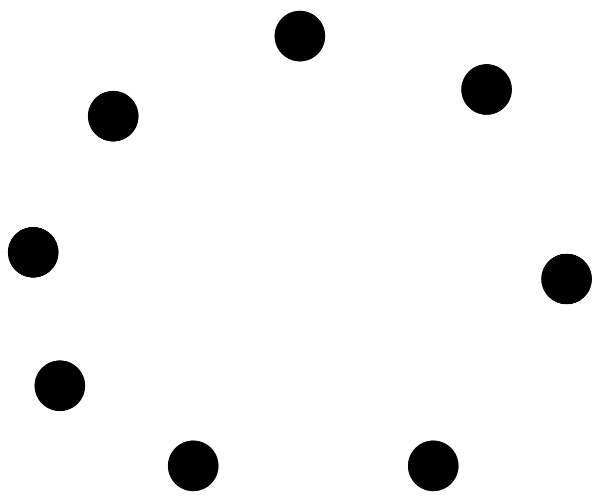
Valid polygons



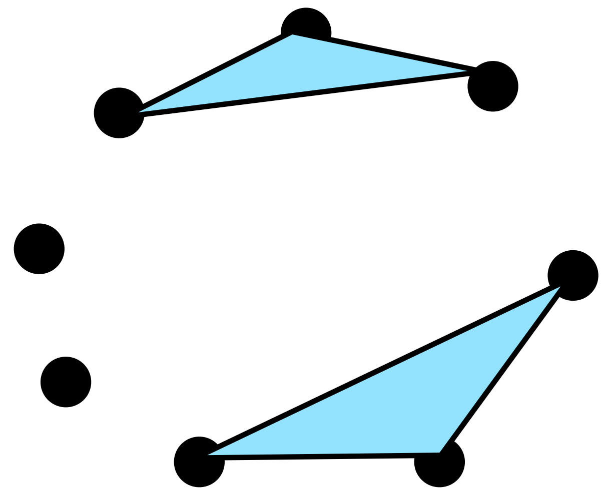
- Simple
- Convex
- Flat

OpenGL polygons

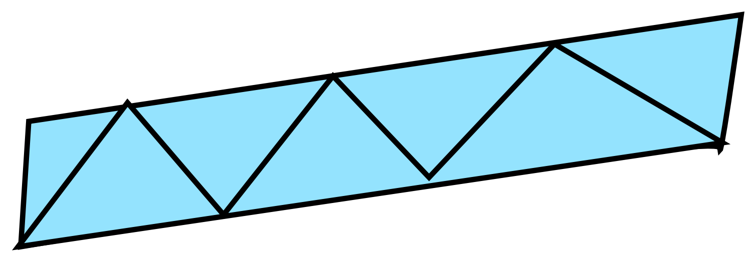
- Only triangles are supported (in latest versions)



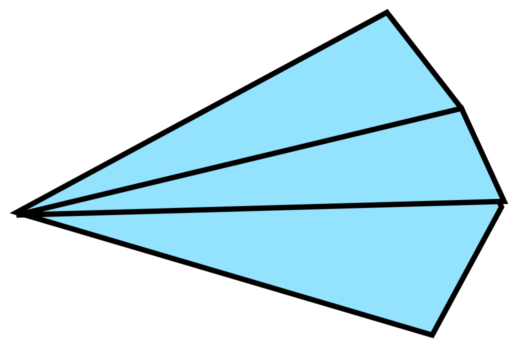
GL_POINTS



GL_TRIANGLES

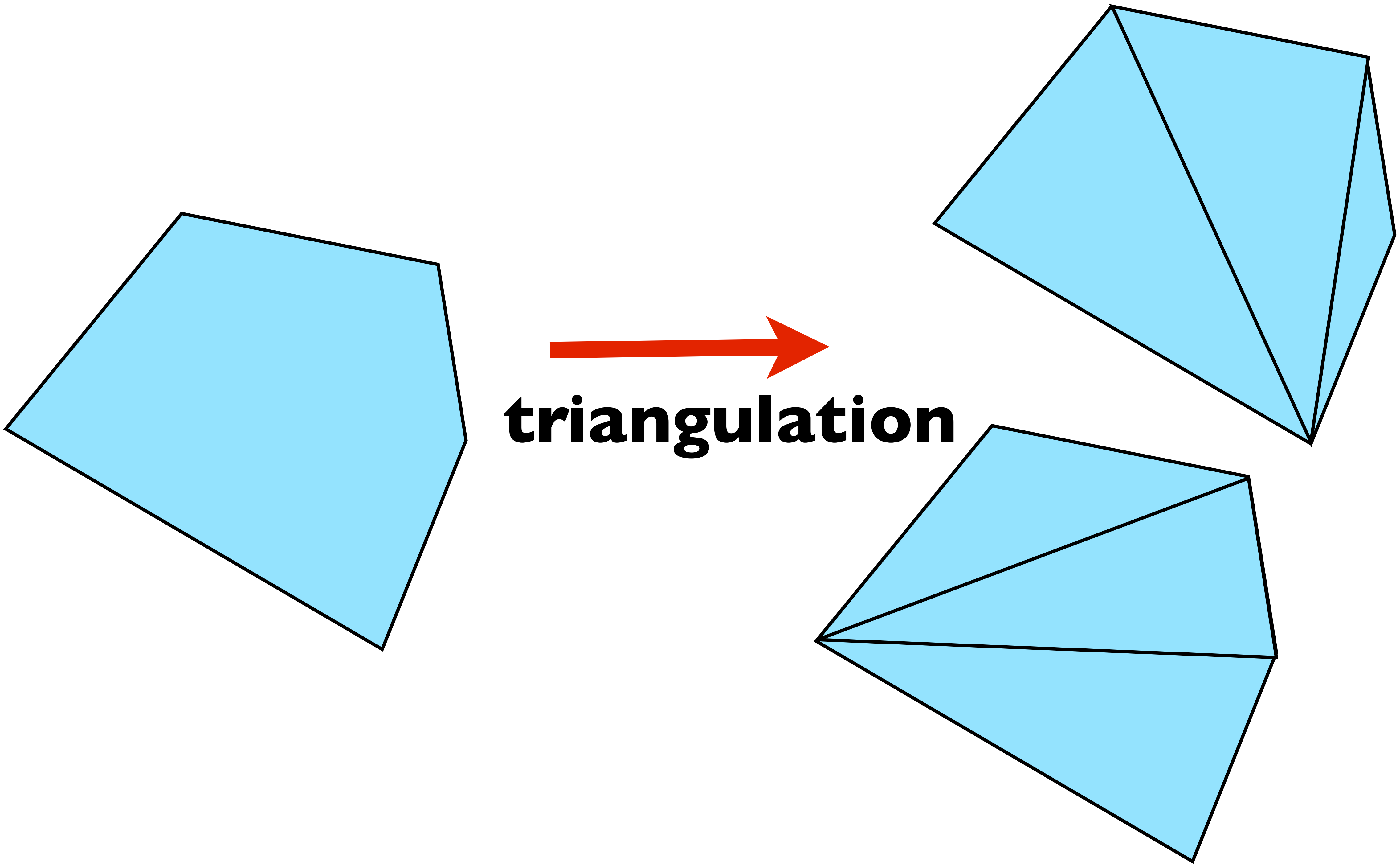


GL_TRIANGLE_STRIP



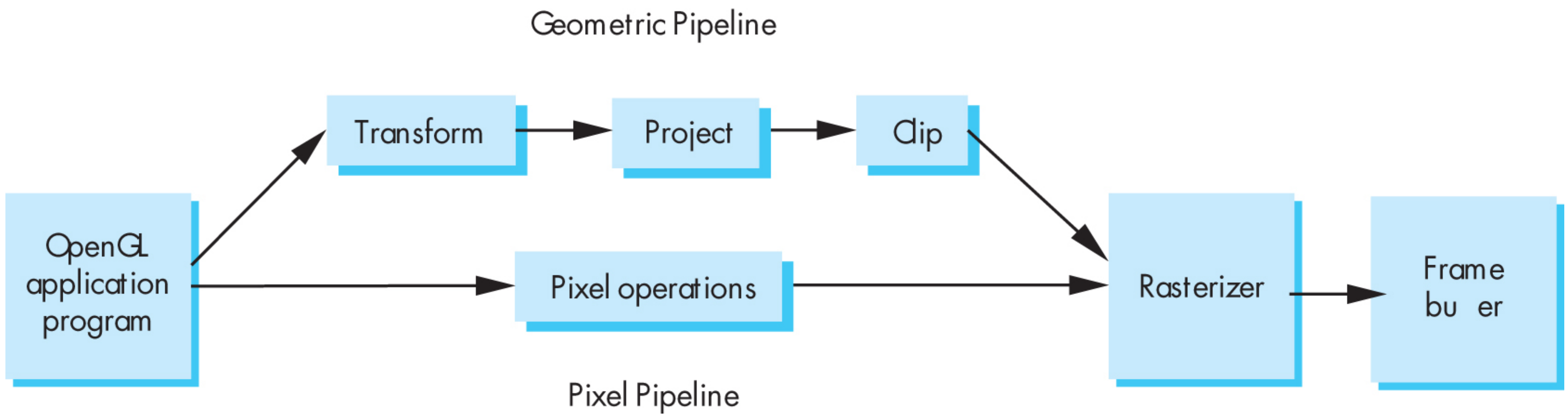
GL_TRIANGLE_FAN

Other polygons



triangulation
as long as triangles are not **collinear**, they will be **simple**, **flat,** and **convex -- easy to render**

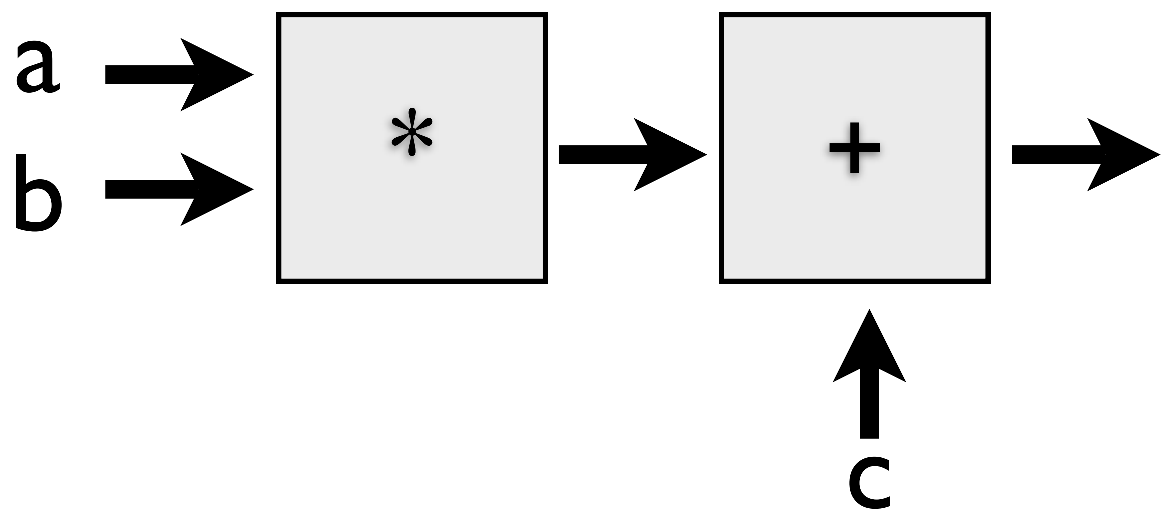
Graphics Pipeline

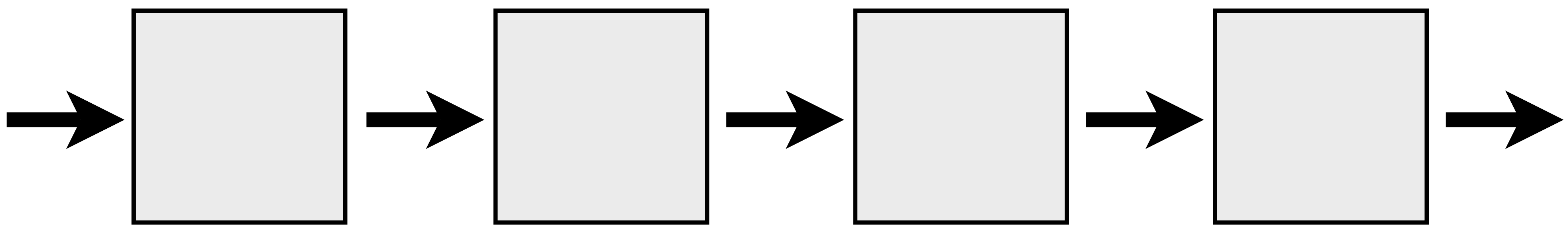


[Angel and Shreiner]

Pipelining operations

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



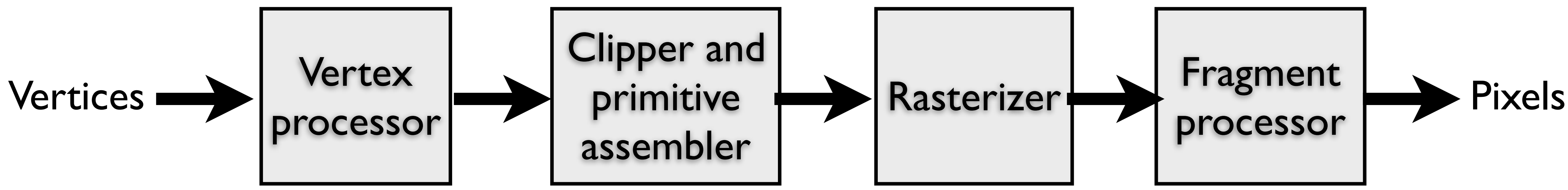


By pipelining the arithmetic operation, the **throughput**, or rate at which data flows through the system, has been **doubled**

If the pipeline had more boxes, the **latency, or time it takes one datum to pass through the system**, would be higher

throughput and latency must be balanced

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

the pipeline is best when we are doing the same operations on many data sets
 -- good for computer graphics!! where we process larges sets of vertices and pixels in the same manner

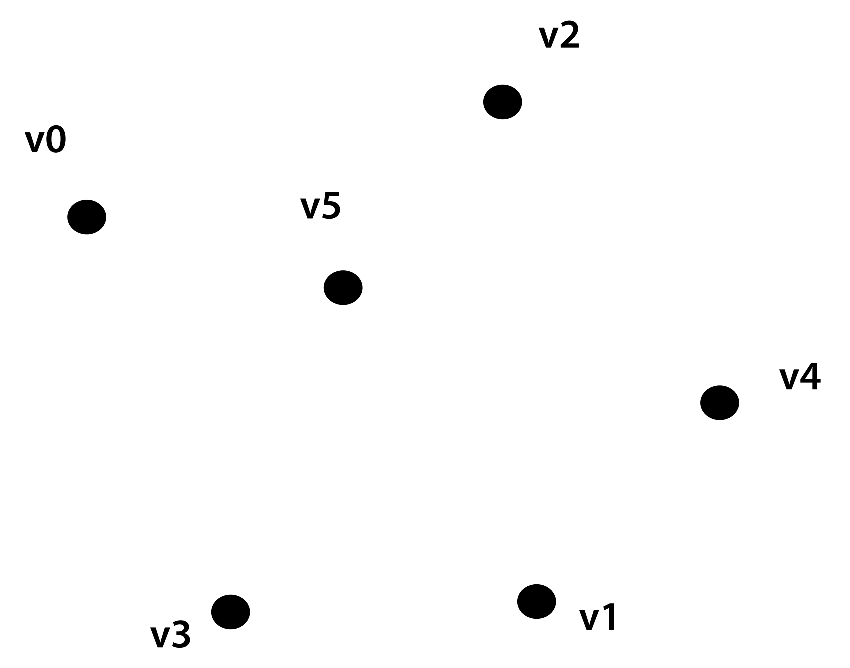
1. **Geometry**: objects – made of primitives – made of vertices
2. **Vertex processing:** coordinate transformations and color
3. **Clipping and primitive assembly:** use clipping volume. must be primitive by primitive rather than vertex by vertex. therefore vertices must be assembled into primitives before clipping can take place. Output is a set of primitives.
4. **Rasterization:** primitives are still in terms of vertices -- must be converted to pixels. E.g., for a triangle specificied by 3 vertices, the rasterizer must figure out which pixels in the frame buffer fill the triangle. Output is a set of **fragments for each primitive**. A fragment is like a **potential pixel.** Fragments can carry depth information used to figure out if they lie behind other fragments for a given pixel.
5. **Fragment processing:** update pixels in the frame buffer. some fragments may not be visible. texture mapping and bump mapping. blending.

Graphics Pipeline

(slides courtesy K. Fatahalian)

Vertex processing

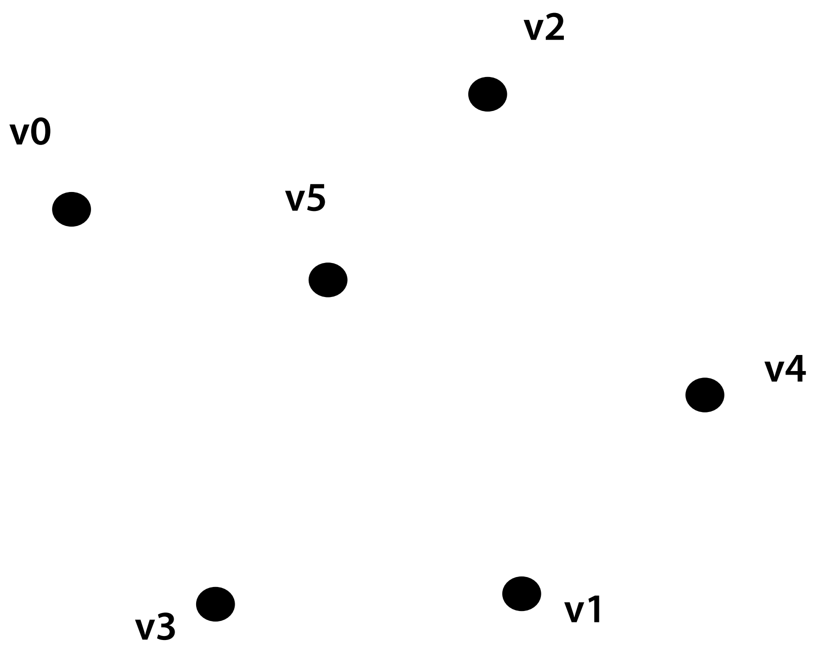
Vertices are transformed into "screen space"



Vertices

Vertex processing

Vertices are transformed into "screen space"

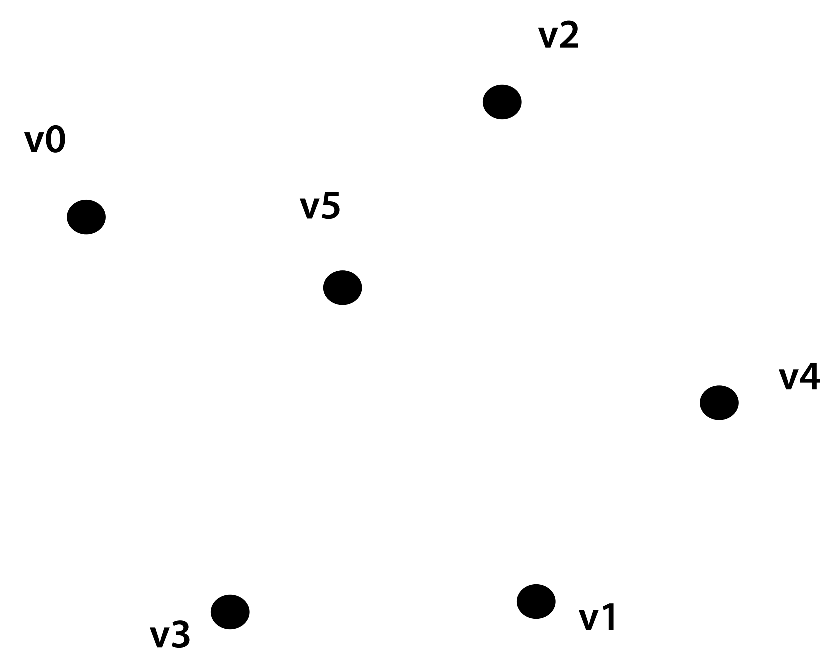


Vertices

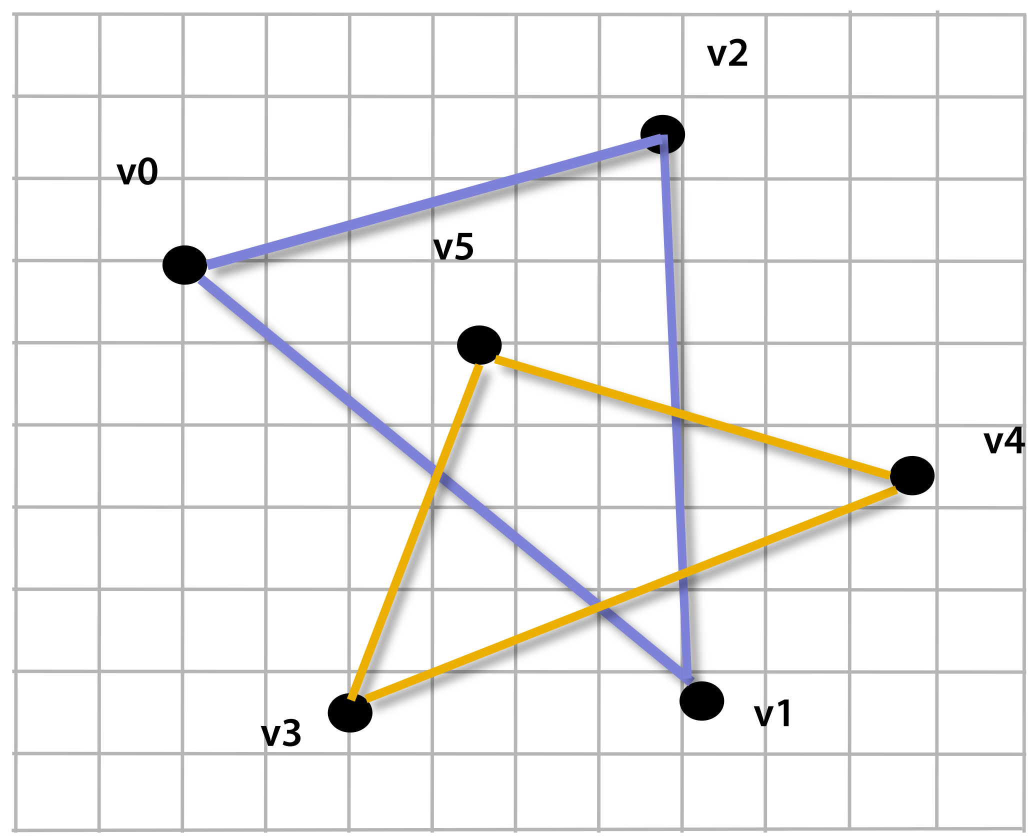
EACH VERTEX IS TRANSFORMED INDEPENDENTLY

Primitive processing

Then organized into primitives that are clipped and culled…



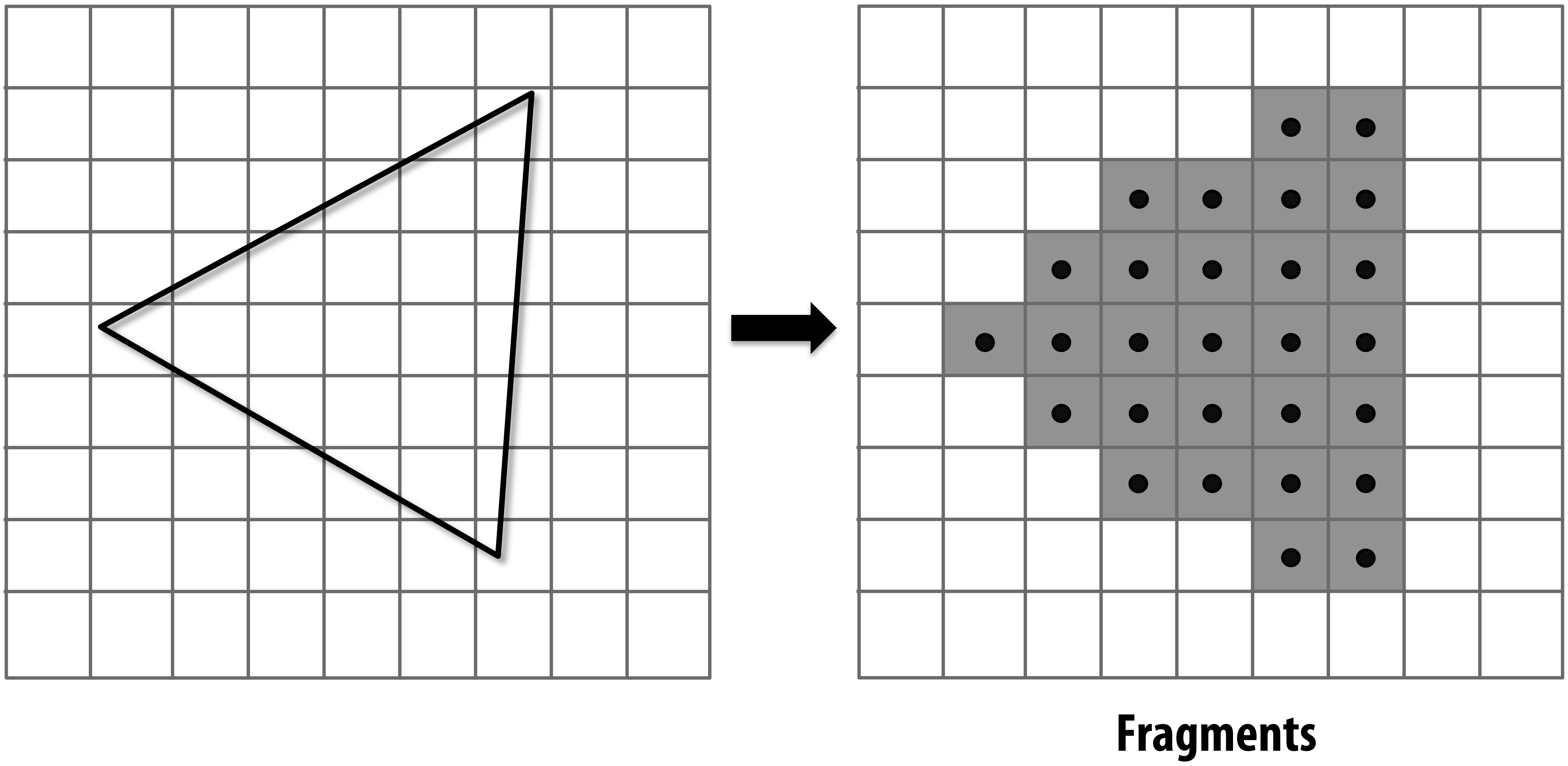
Vertices



Primitives (triangles)

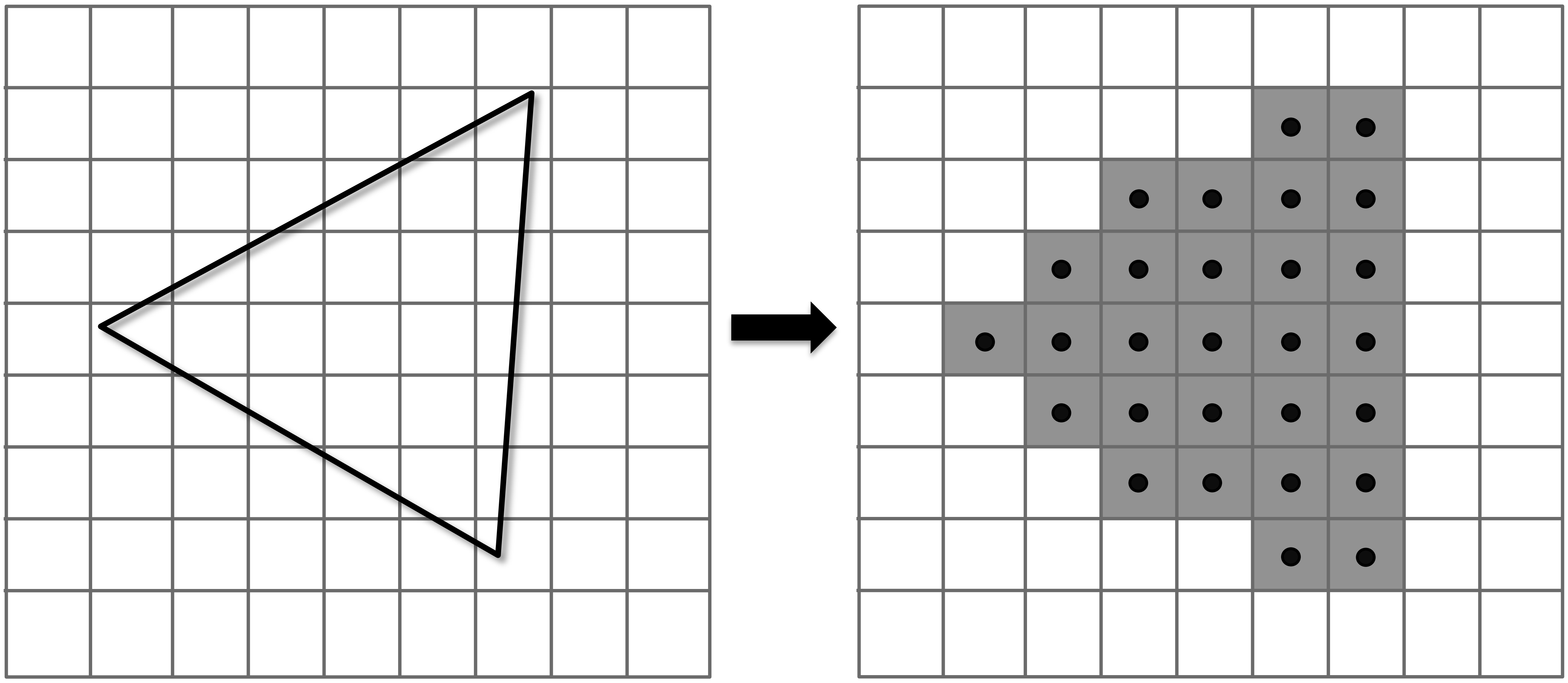
Rasterization

Primitives are rasterized into "pixel fragments"



Rasterization

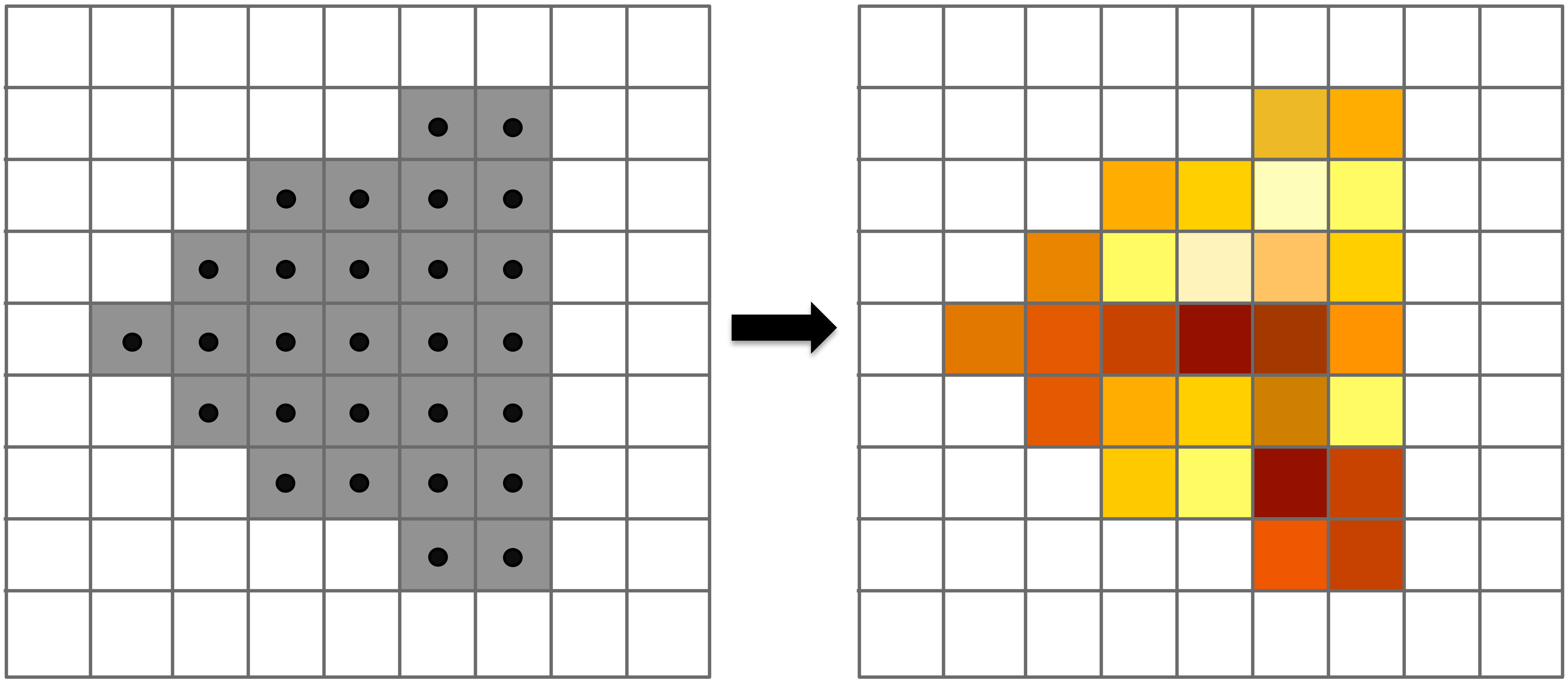
Primitives are rasterized into "pixel fragments"



EACH PRIMITIVE IS RASTERIZED INDEPENDENTLY

Fragment processing

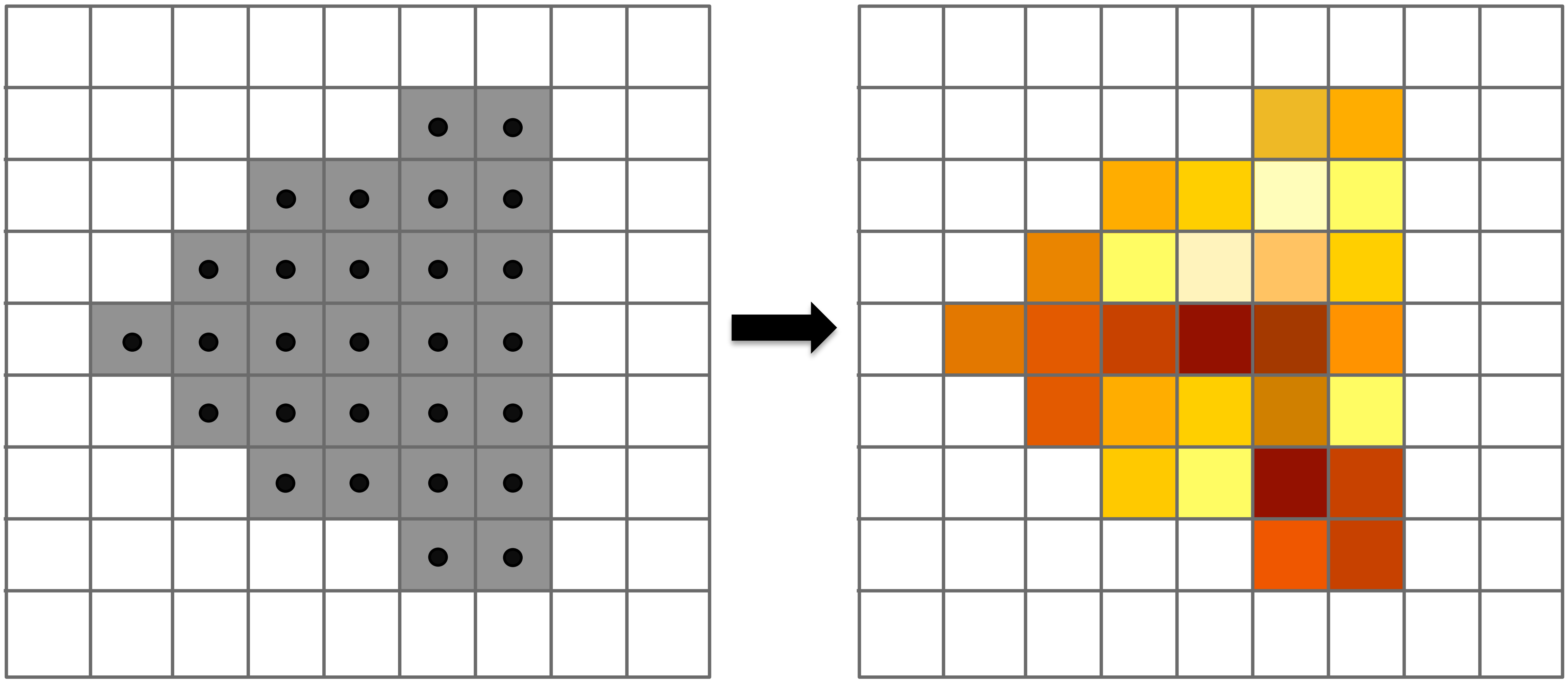
Fragments are shaded to compute a color at each pixel



Shaded fragments

Fragment processing

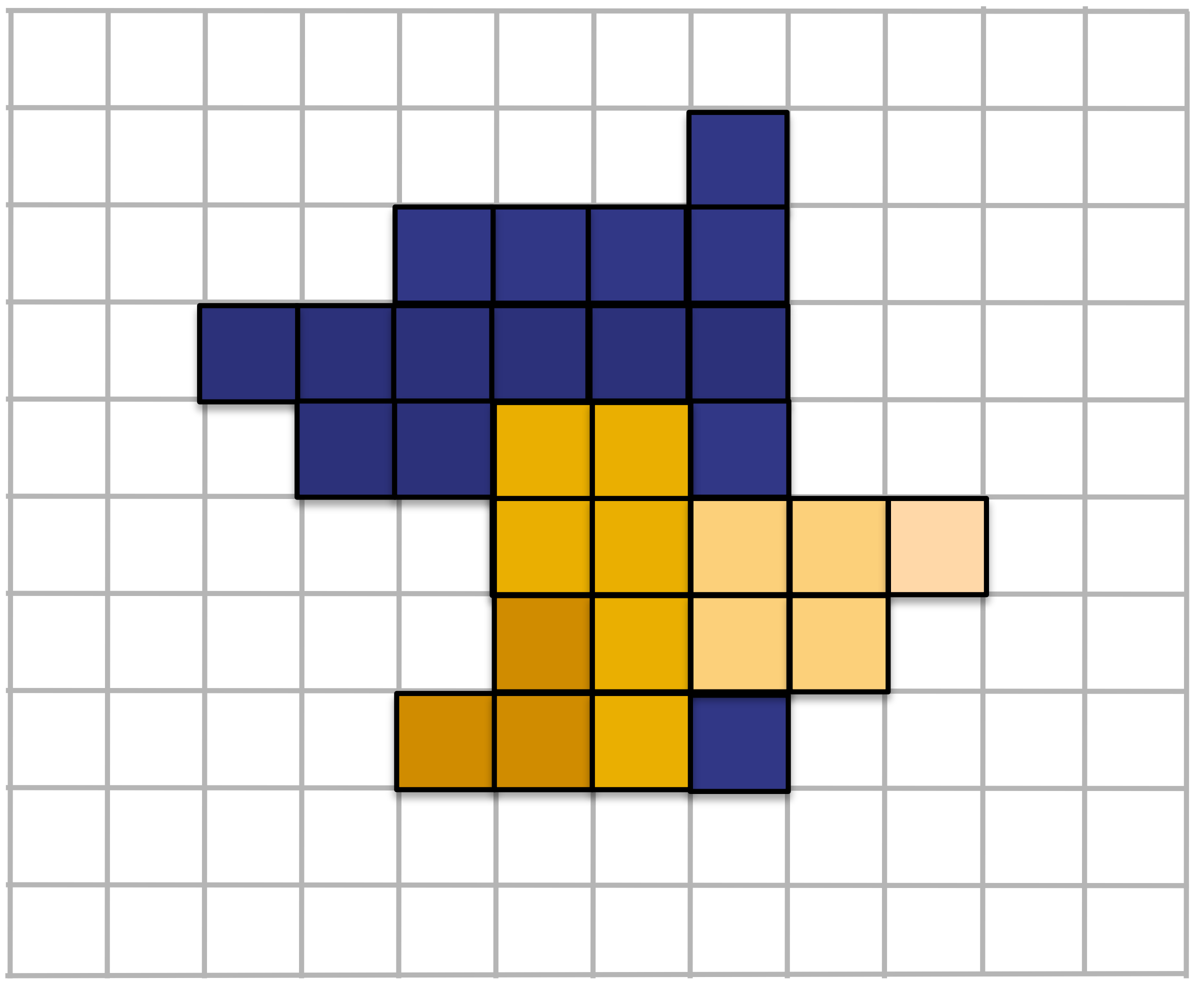
Fragments are shaded to compute a color at each pixel



EACH FRAGMENT IS PROCESSED INDEPENDENTLY

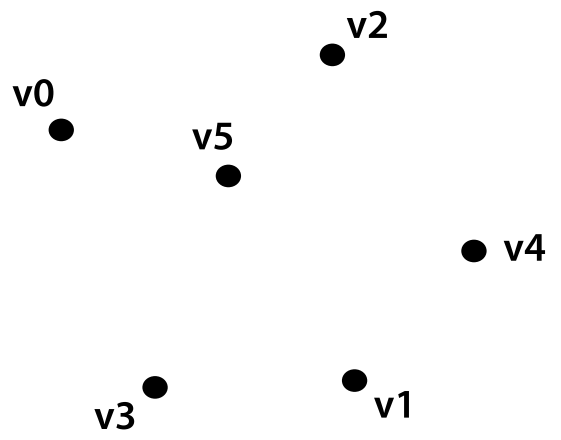
Pixel operations

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

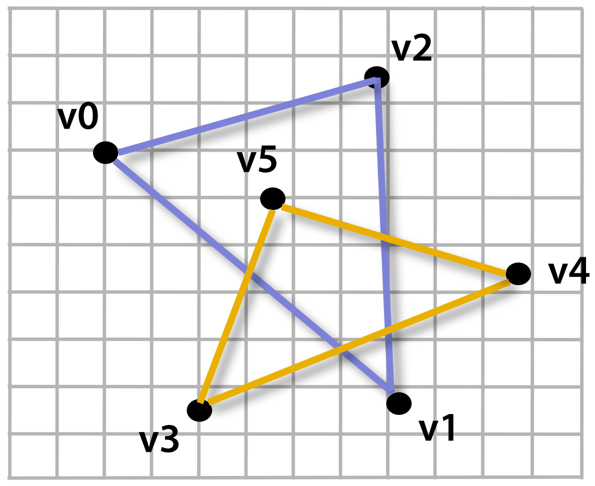


Pixels

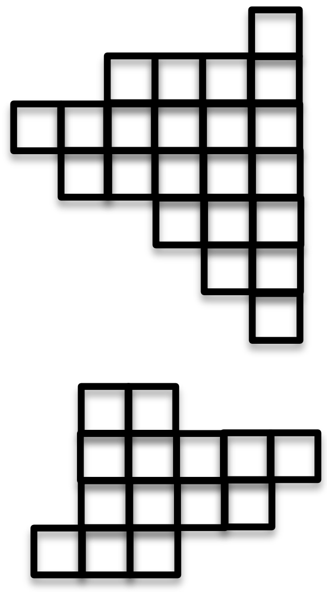
Pipeline entities



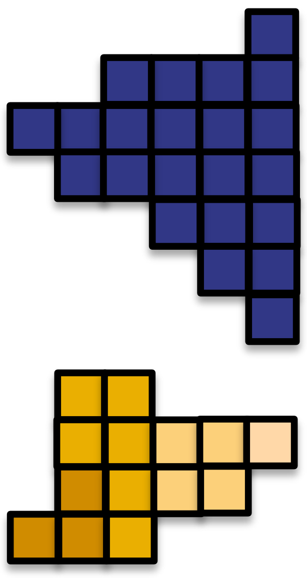
Vertices



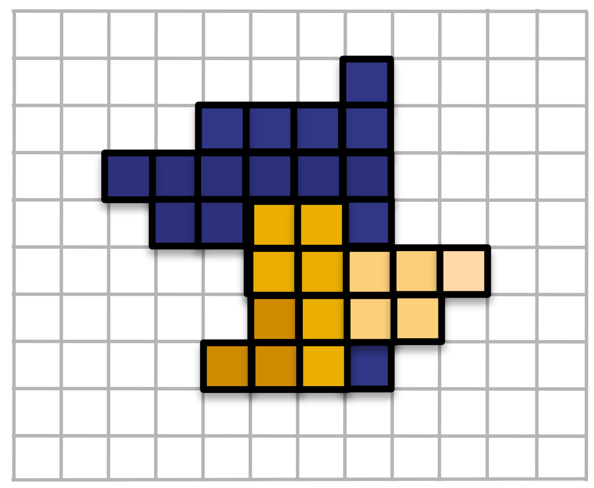
Primitives



Fragments



Fragments (shaded)



Pixels

Graphics pipeline