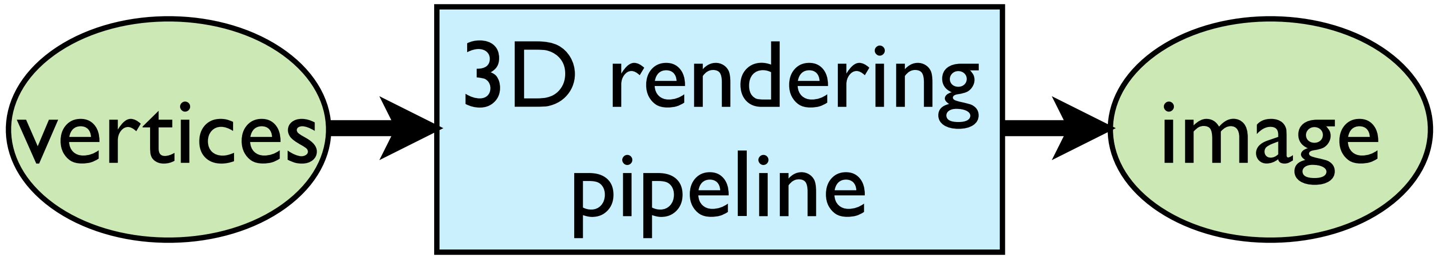
Graphics Pipeline

Rendering approaches

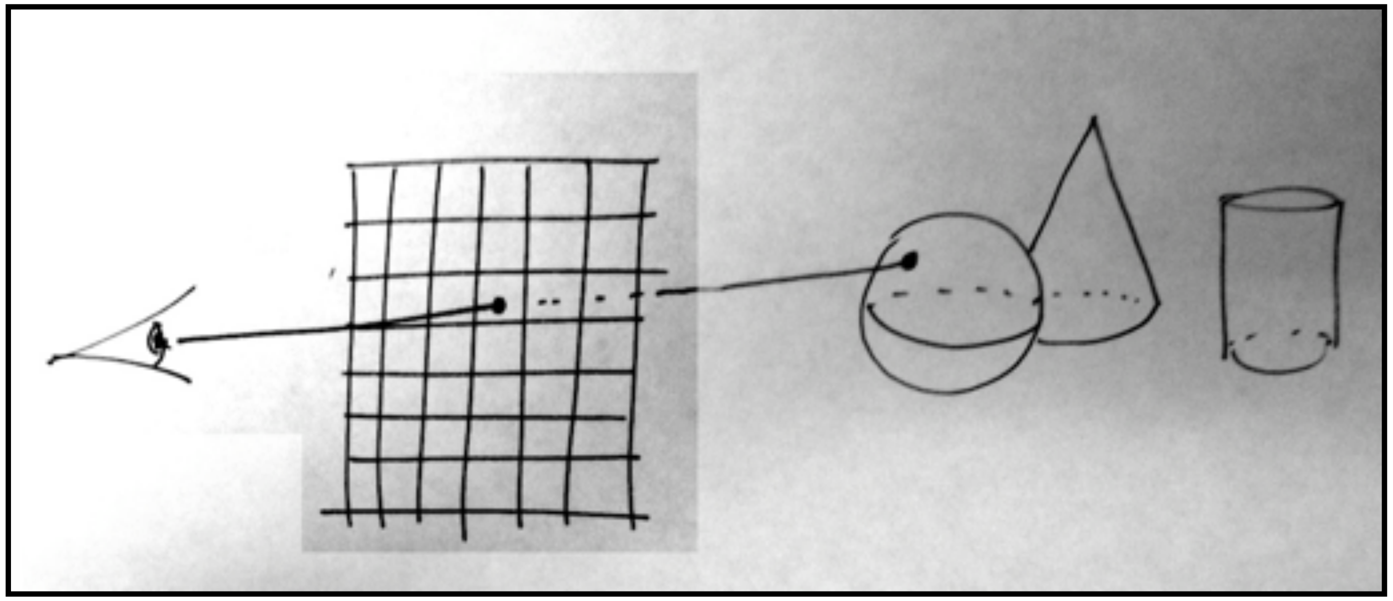
1. **object-oriented**

   foreach object ...


vertices → 3D rendering pipeline → image

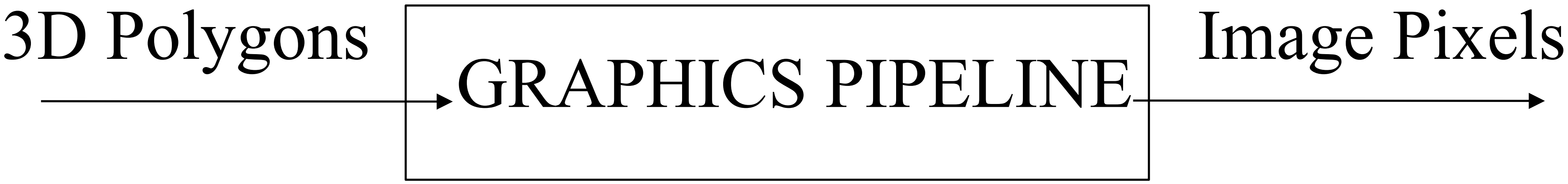
2. **image-oriented**

   foreach pixel ...



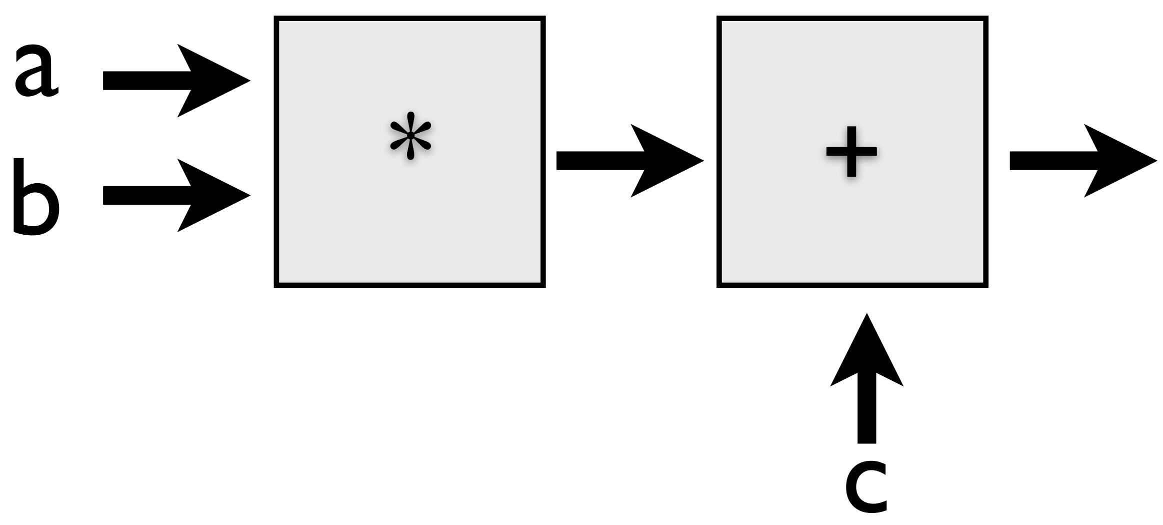
Z-buffer Rendering

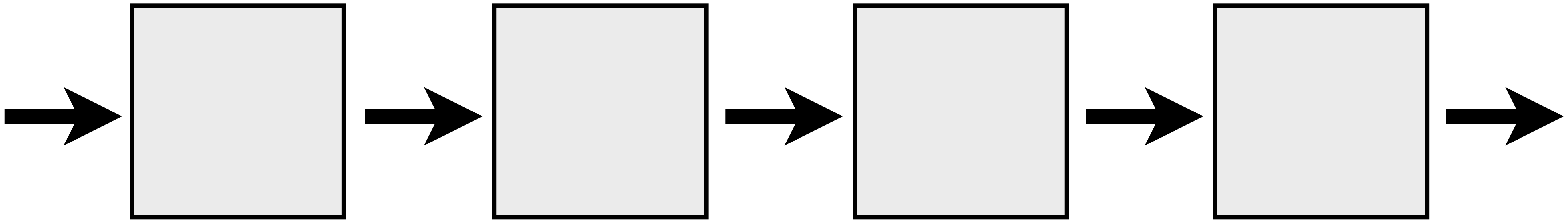
- Z-buffering is very common approach, also often accelerated with hardware
- OpenGL is based on this approach



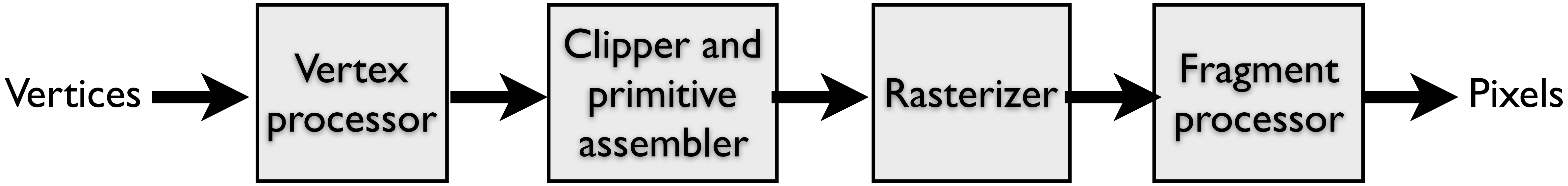
Pipelining operations

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





3D graphics pipeline



Geometry: objects – made of 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

3D graphics pipeline

- optimized for drawing 3D triangles with shared vertices
- map 3D vertex locations to 2D screen locations
- shade triangles and draw them in back to front order using a z-buffer
- speed depends on # of triangles
- most operations on vertices can be represented using a 4D coordinate space - 3D position + homogeneous coordinate for perspective viewing
    - 4x4 matrices and 4-vectors

Primitives and Attributes

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

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

Choice of primitives

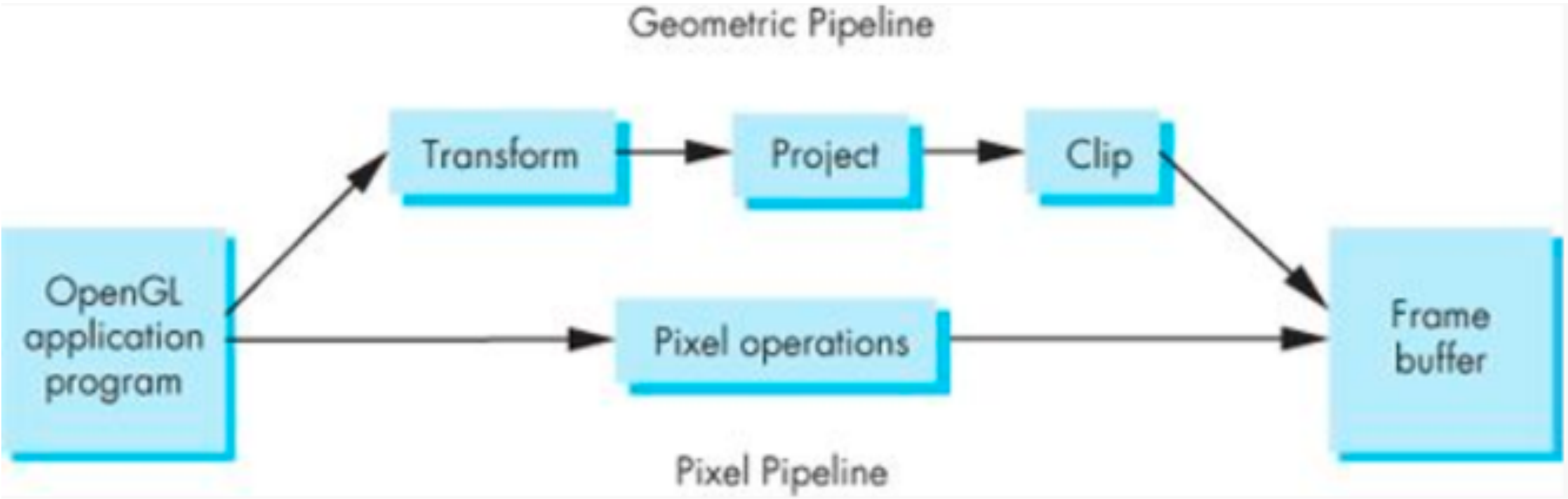
- Which primitives should an API contain?

➡ small set - supported by hardware

  - lots of primitives - convenient for user

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

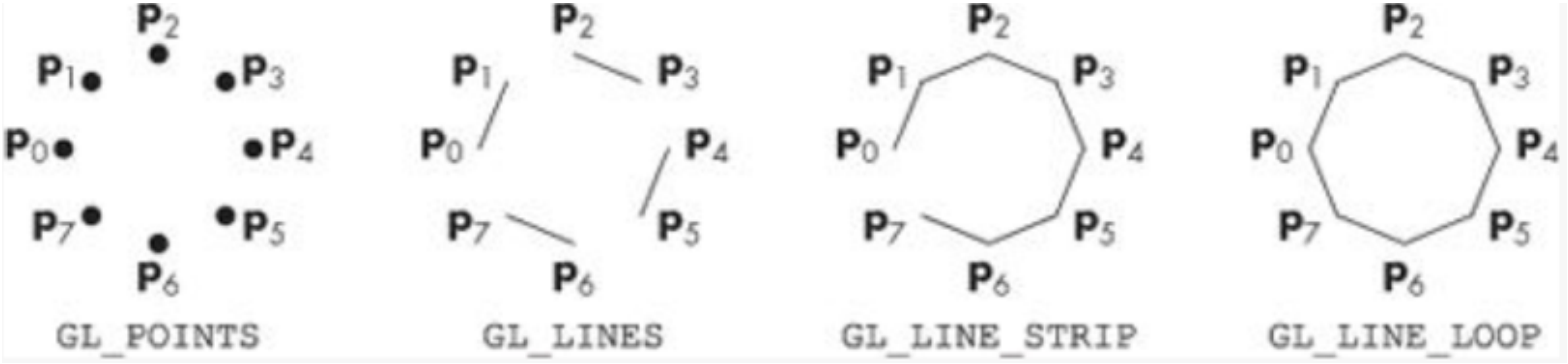
Two classes of primitives



Angel and Shreiner

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

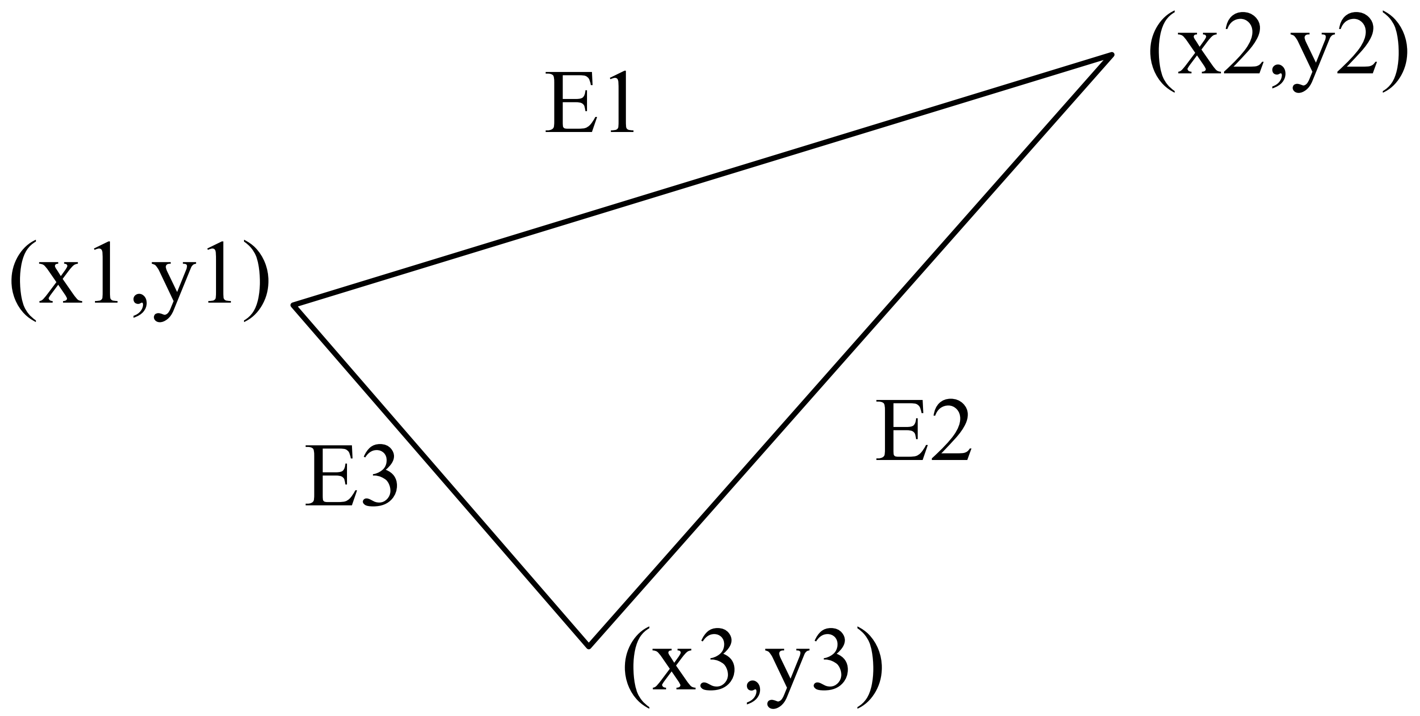
Point and line segment types



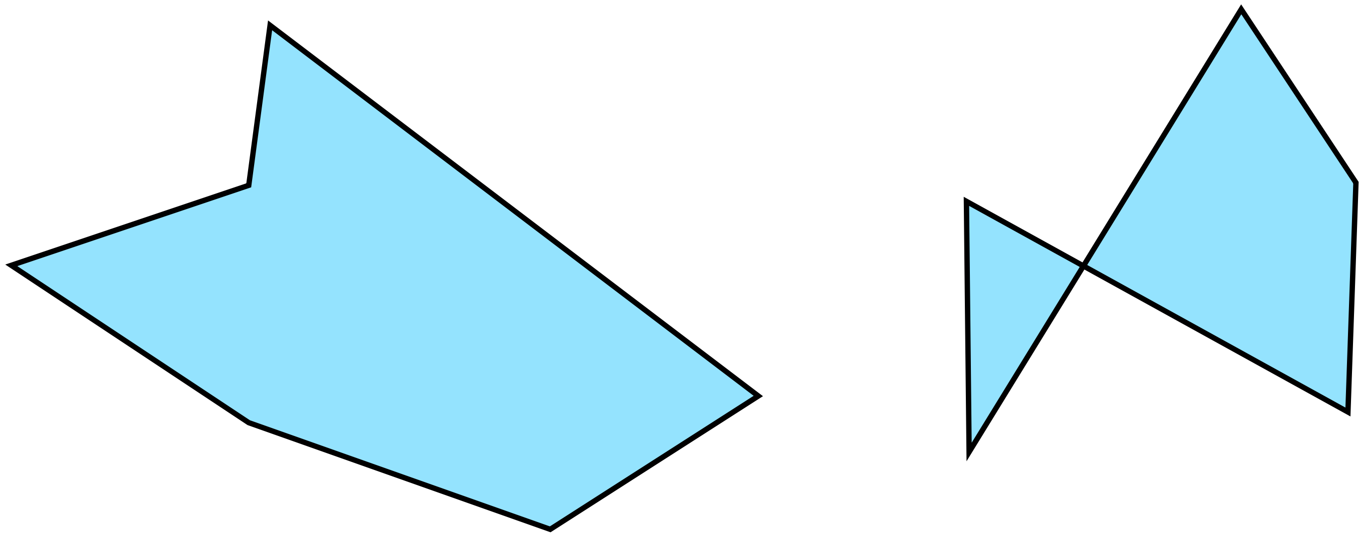
Angel and Shreiner

Polygons

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

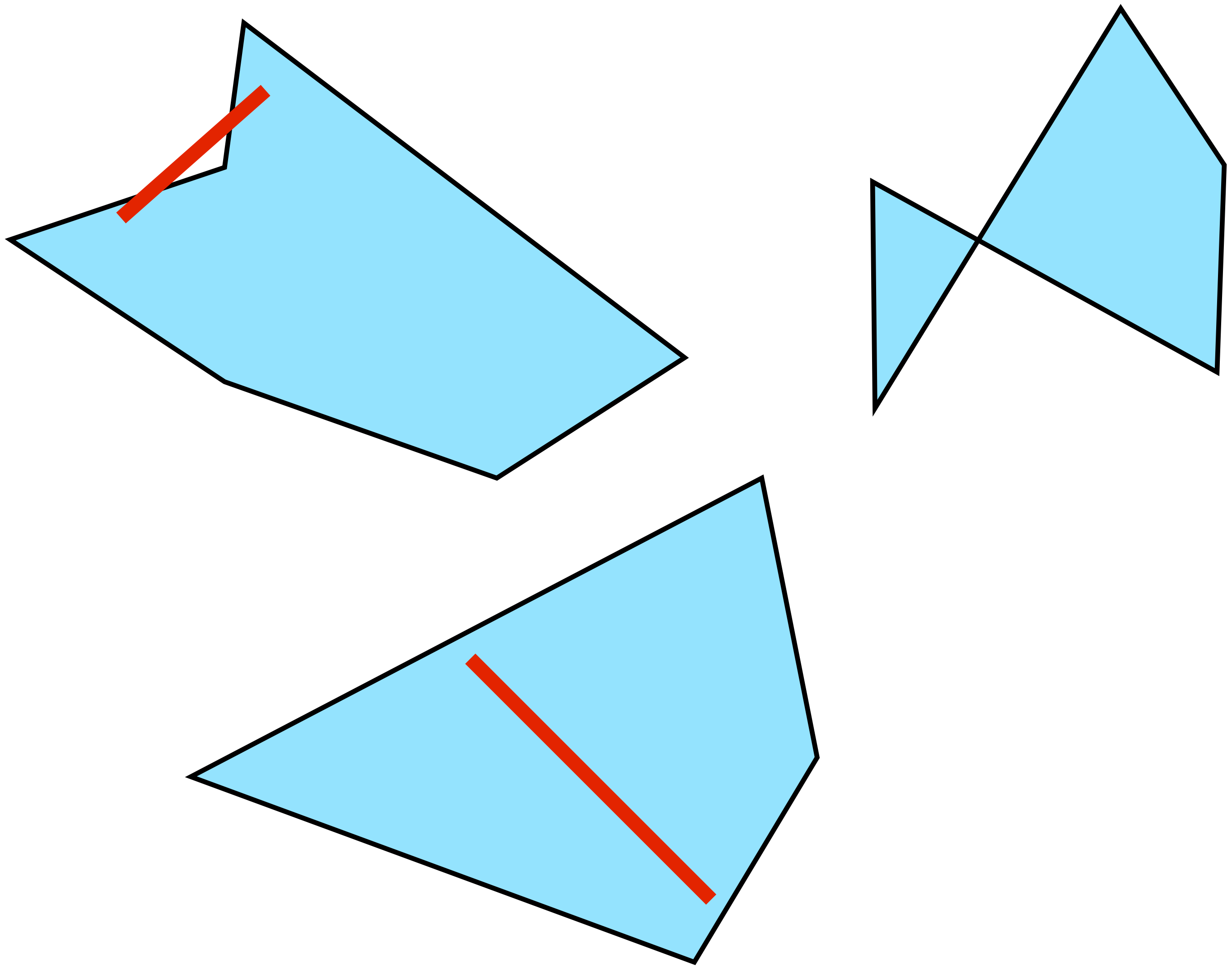


Valid polygons



- Simple
- Convex
- Flat

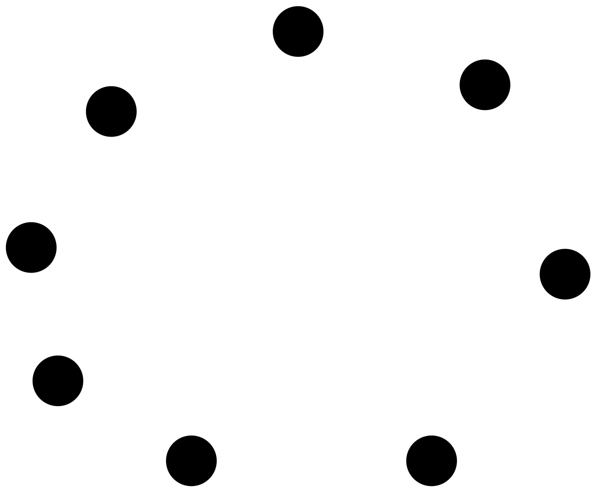
Valid polygons



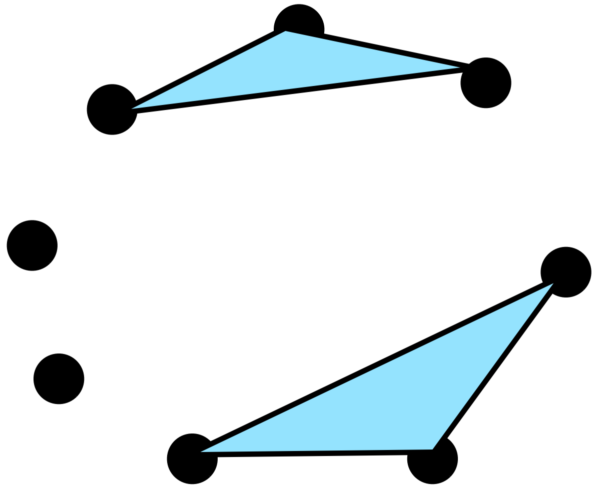
- Simple
- Convex
- Flat

OpenGL polygons

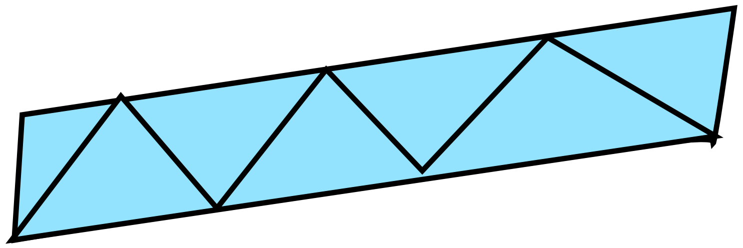
- Only triangles are supported (in latest versions)



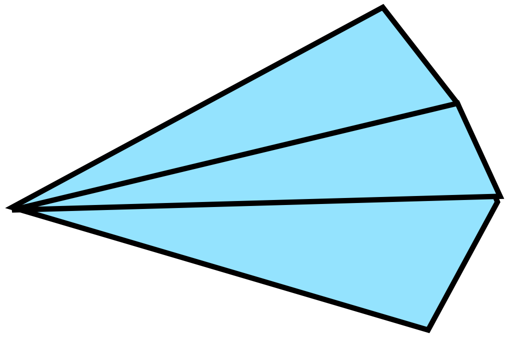
GL_POINTS



GL_TRIANGLES

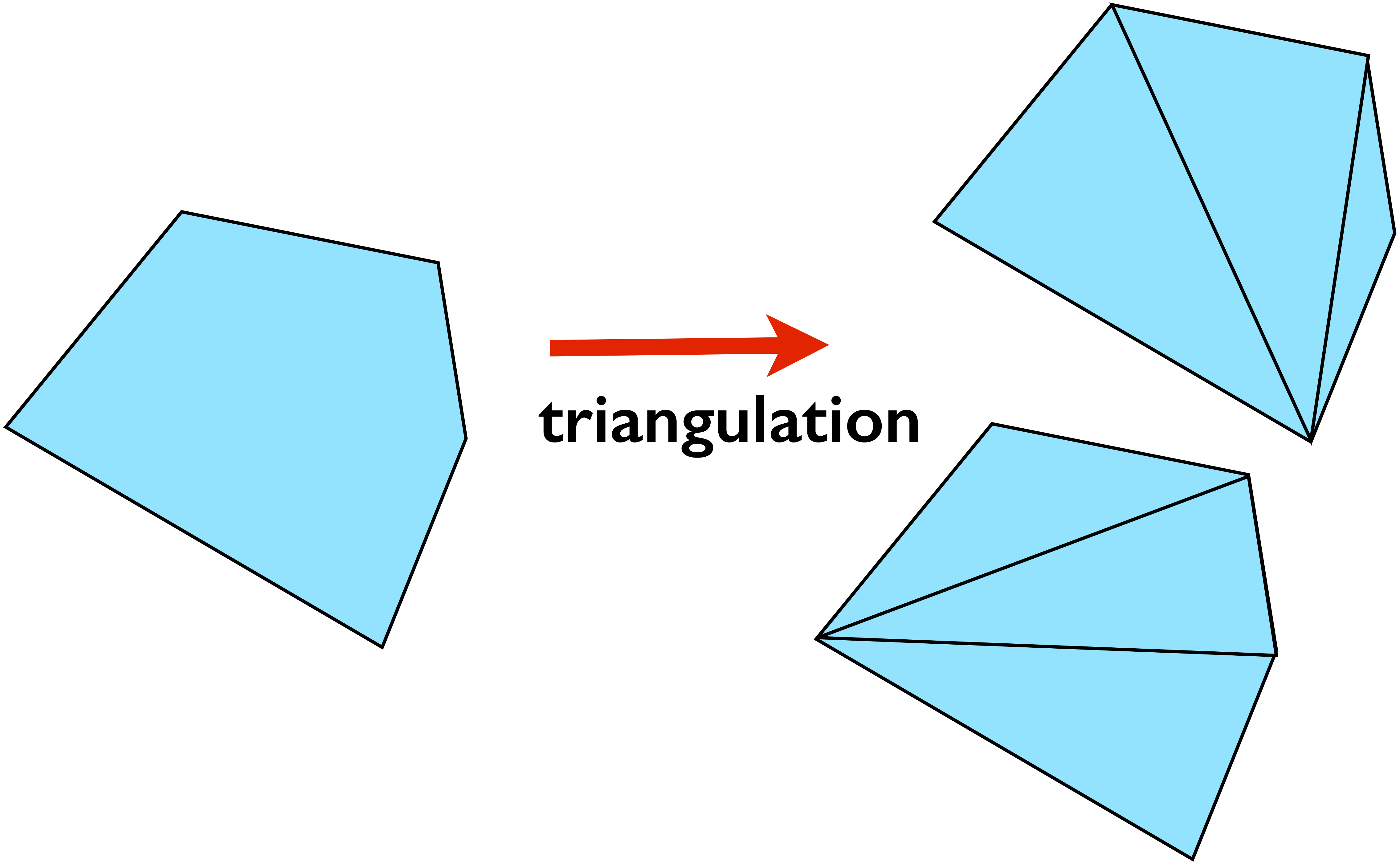


GL_TRIANGLE_STRIP



GL_TRIANGLE_FAN

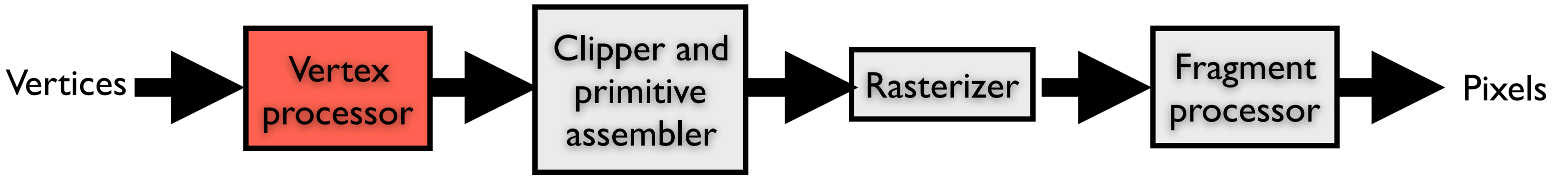
Other polygons



triangulation

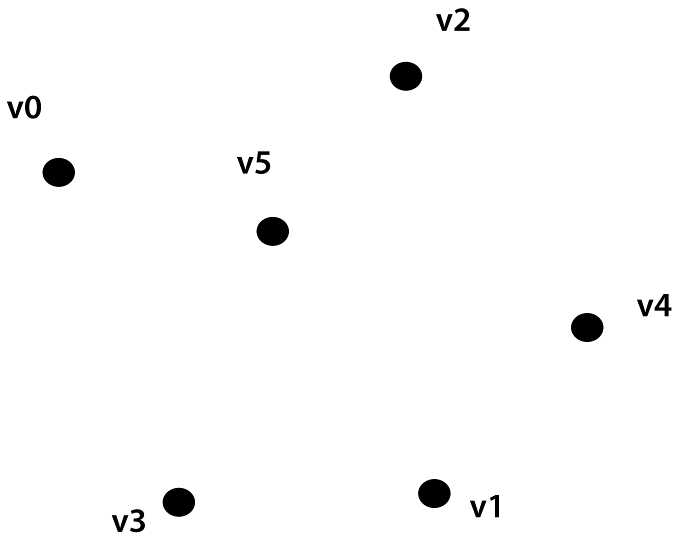
Graphics Pipeline

(slides courtesy K. Fatahalian)

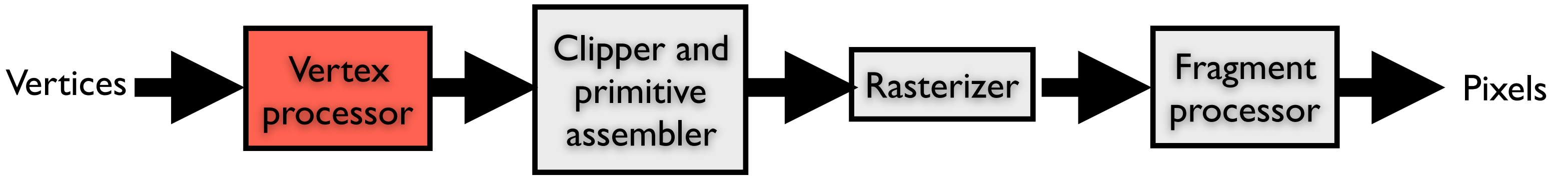


Vertex processing

Vertices are transformed into “screen space”



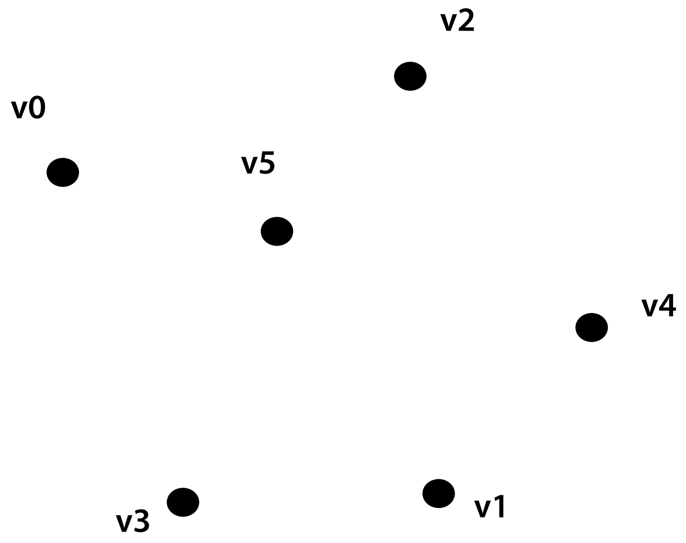
Vertices



Vertices → Vertex processor → Clipper and primitive assembler → Rasterizer → Fragment processor → Pixels

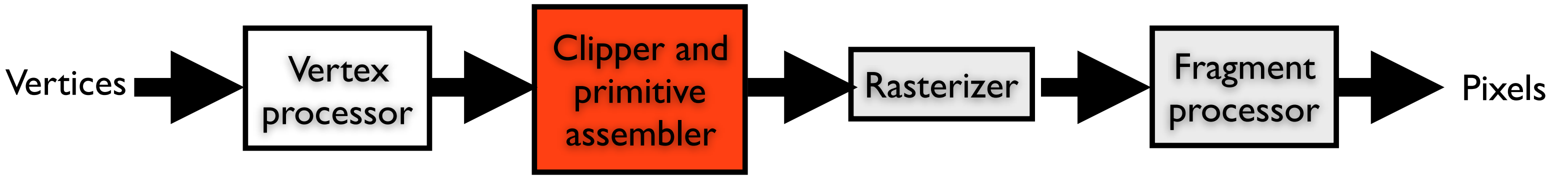
Vertex processing

Vertices are transformed into "screen space"



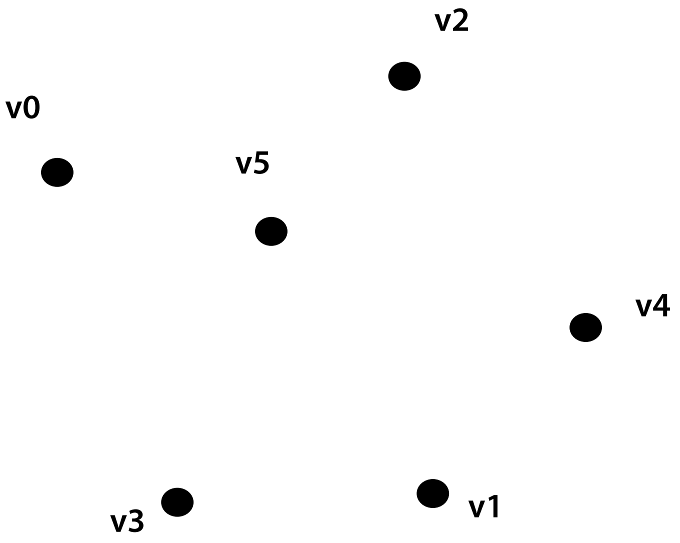
Vertices

EACH VERTEX IS TRANSFORMED INDEPENDENTLY

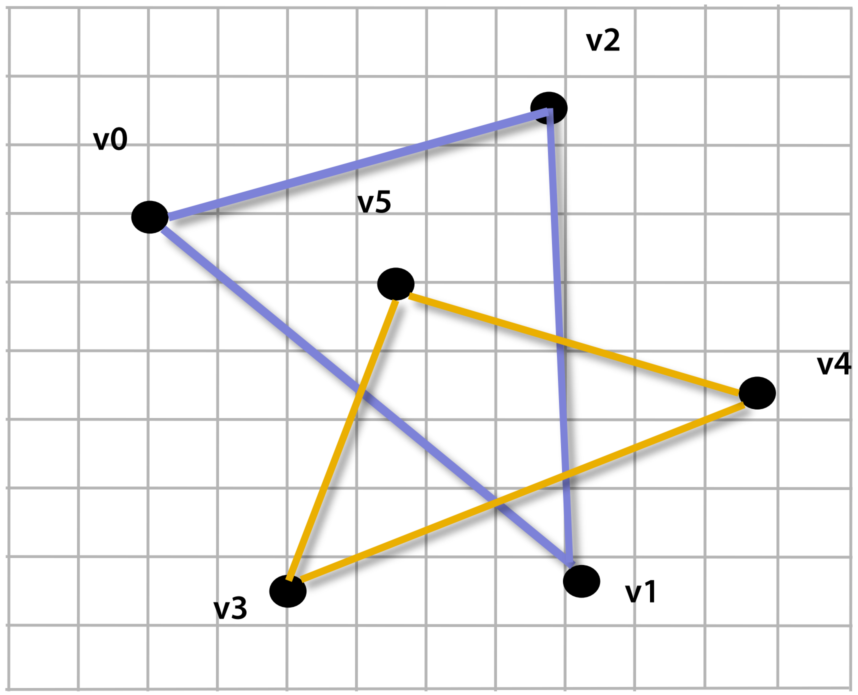

Vertices → Vertex processor → Clipper and primitive assembler → Rasterizer → Fragment processor → Pixels

Primitive processing

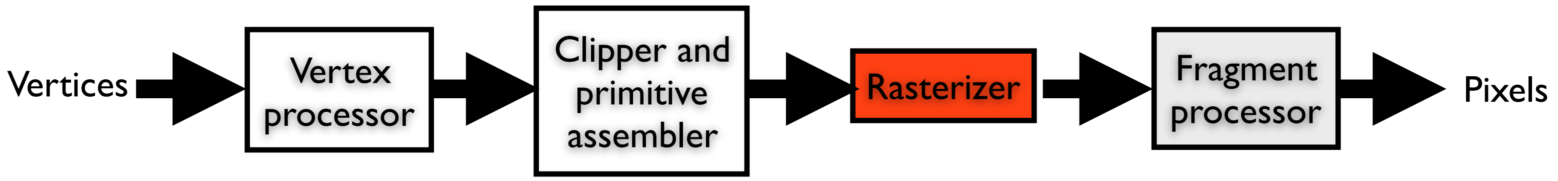
Then organized into primitives that are clipped and culled…



Vertices



Primitives (triangles)



Vertices

Vertex processor

Clipper and primitive assembler

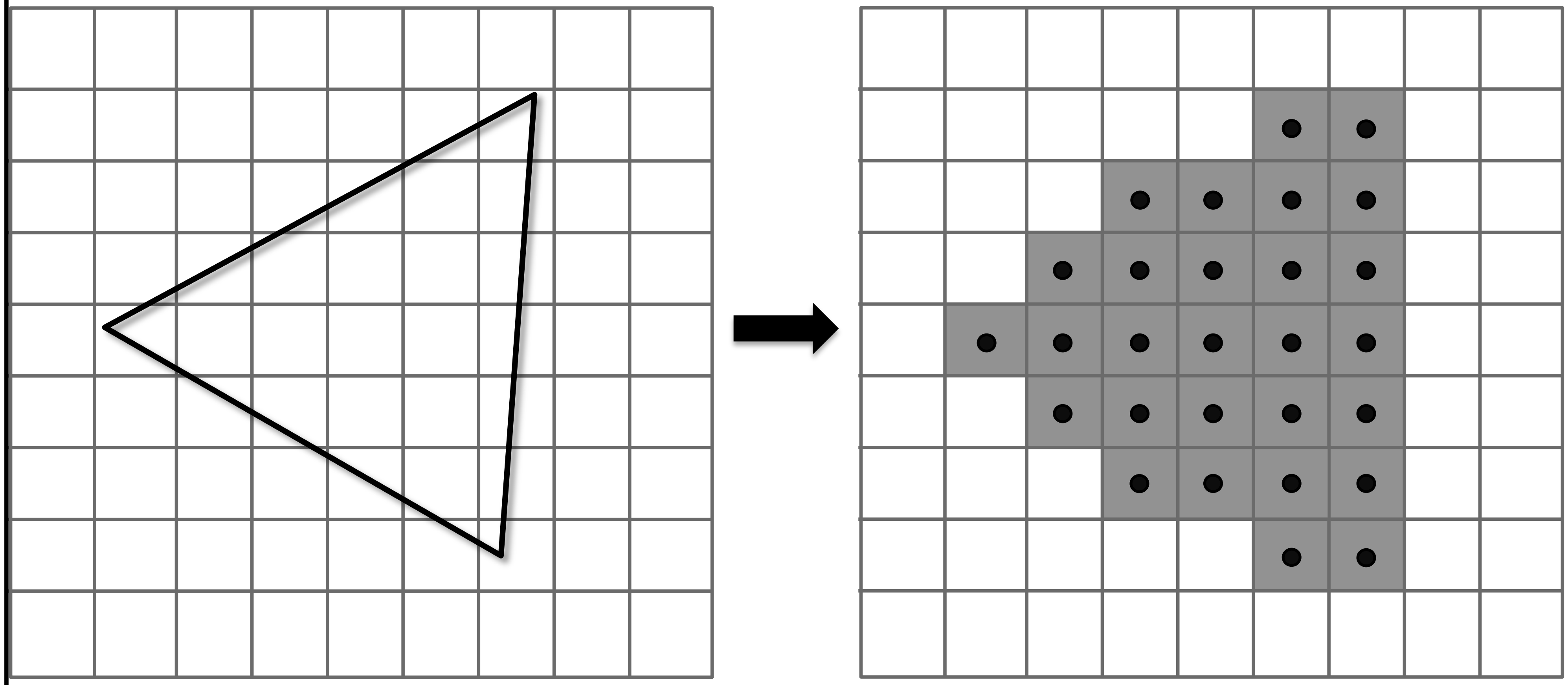
Rasterizer

Fragment processor

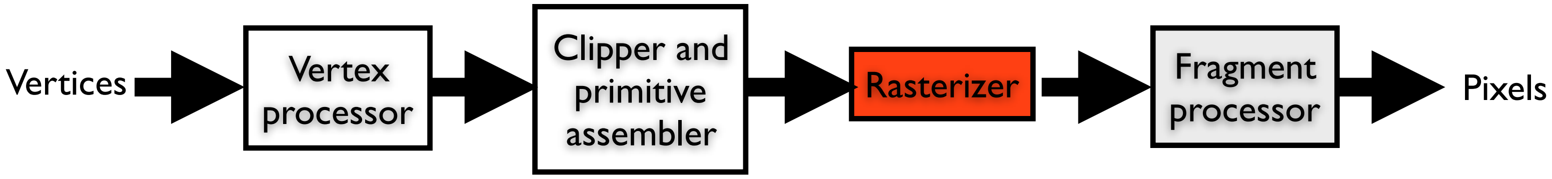
Pixels

Rasterization

Primitives are rasterized into “pixel fragments”

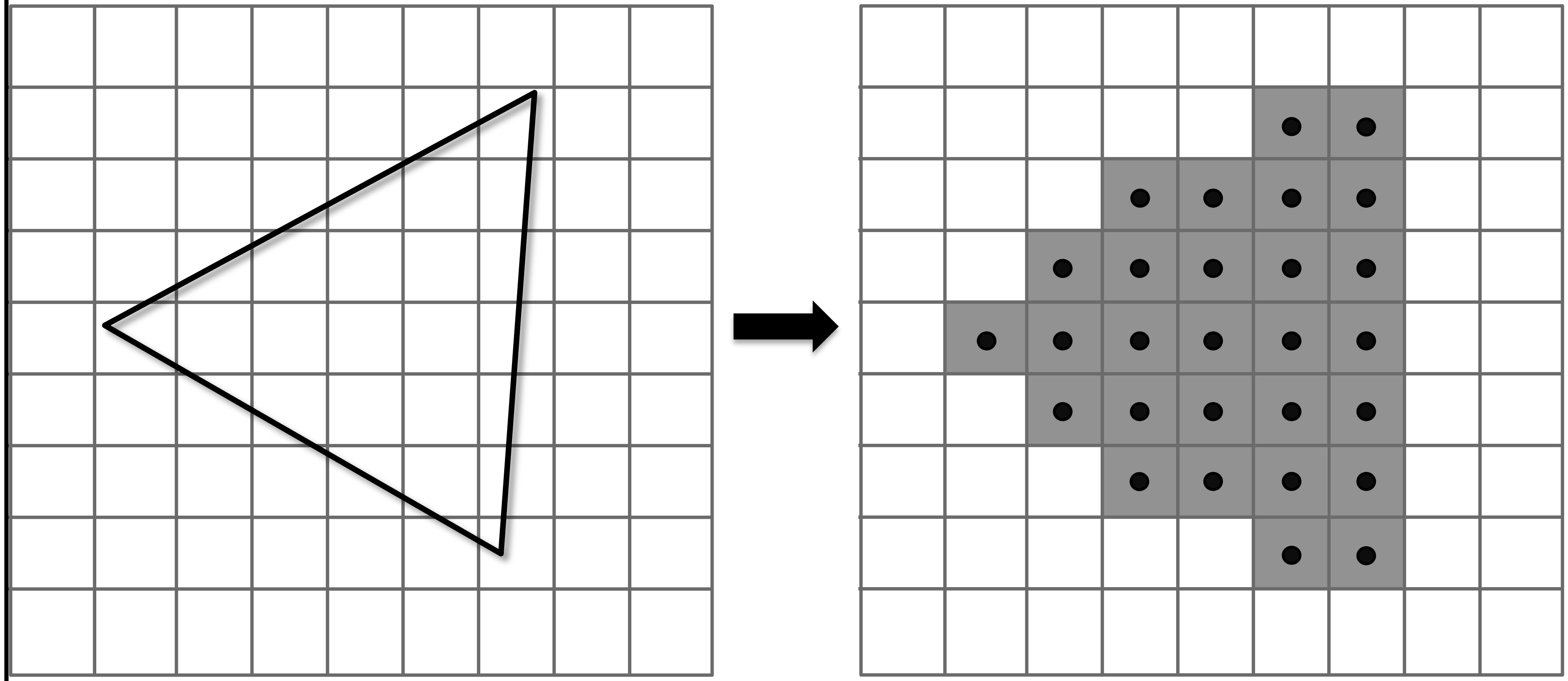


Fragments

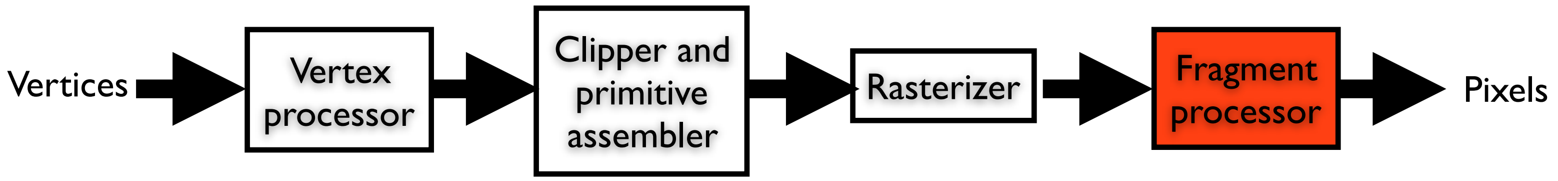


Rasterization

Primitives are rasterized into “pixel fragments”



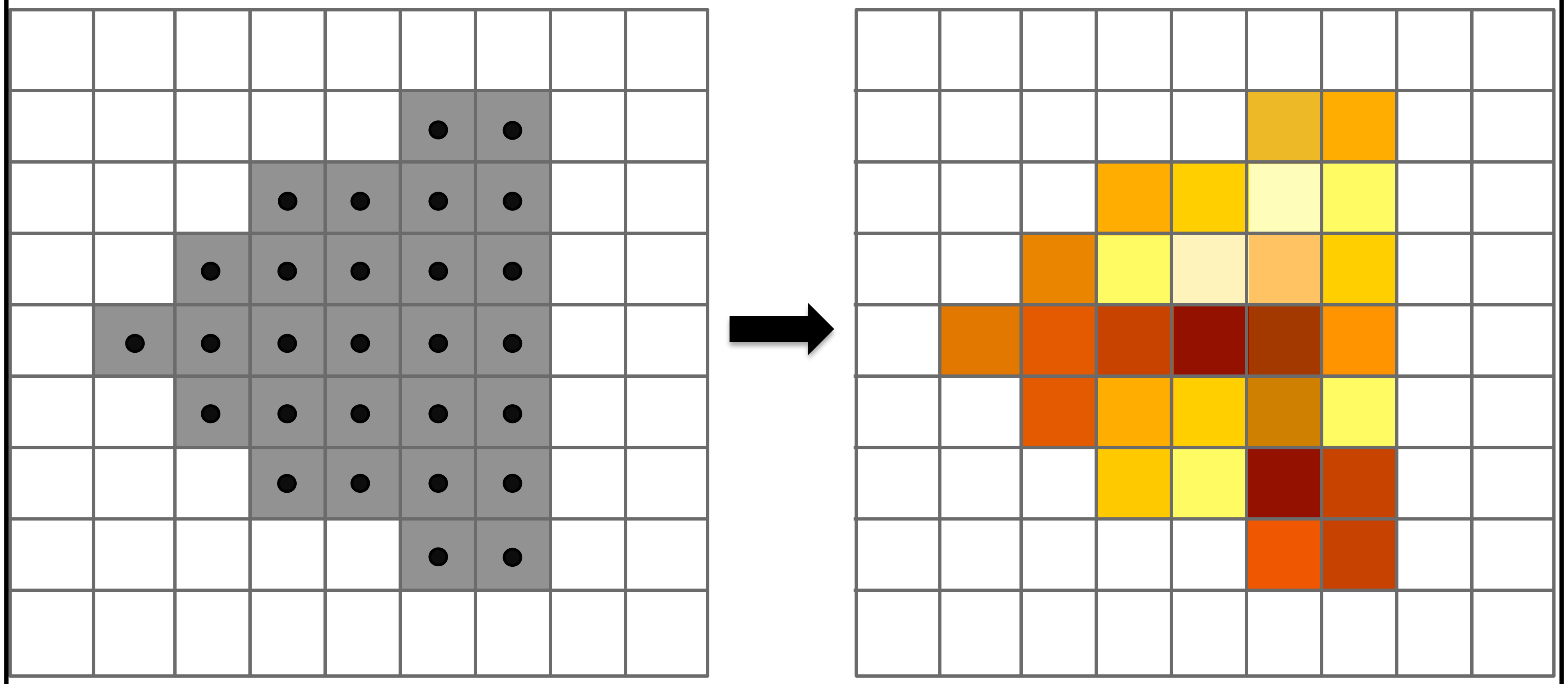
EACH PRIMITIVE IS RASTERIZED INDEPENDENTLY



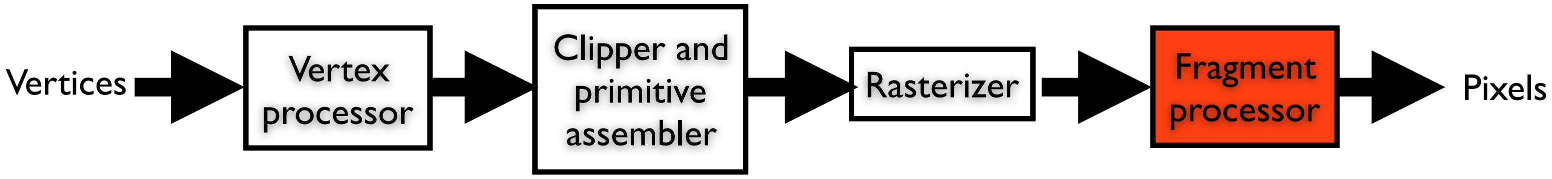
Vertices → Vertex processor → Clipper and primitive assembler → Rasterizer → Fragment processor → Pixels

Fragment processing

Fragments are shaded to compute a color at each pixel

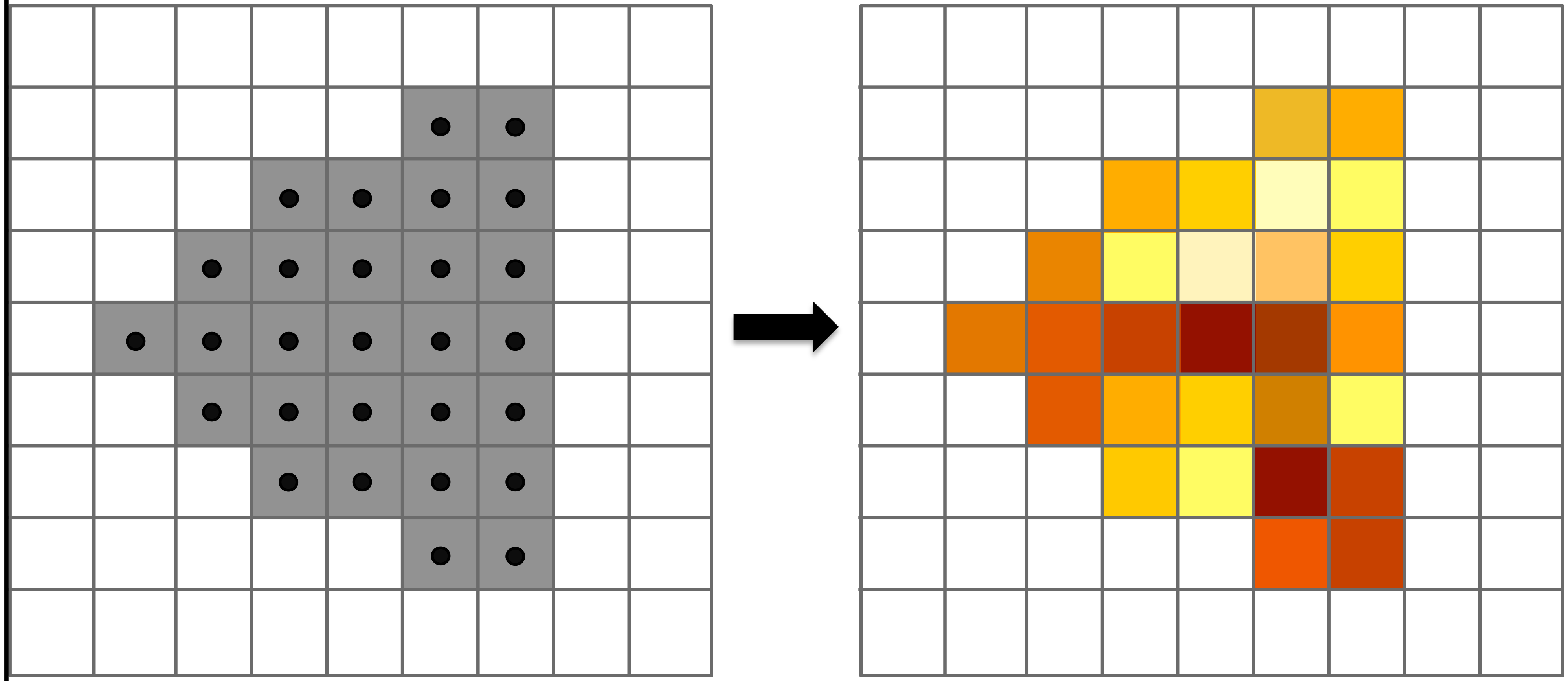


Shaded fragments

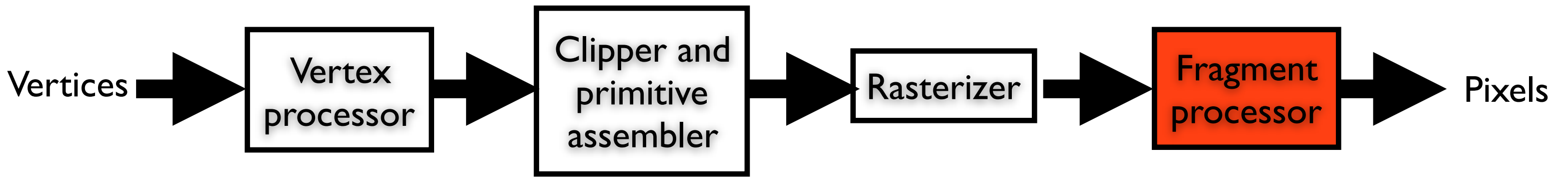


Fragment processing

Fragments are shaded to compute a color at each pixel

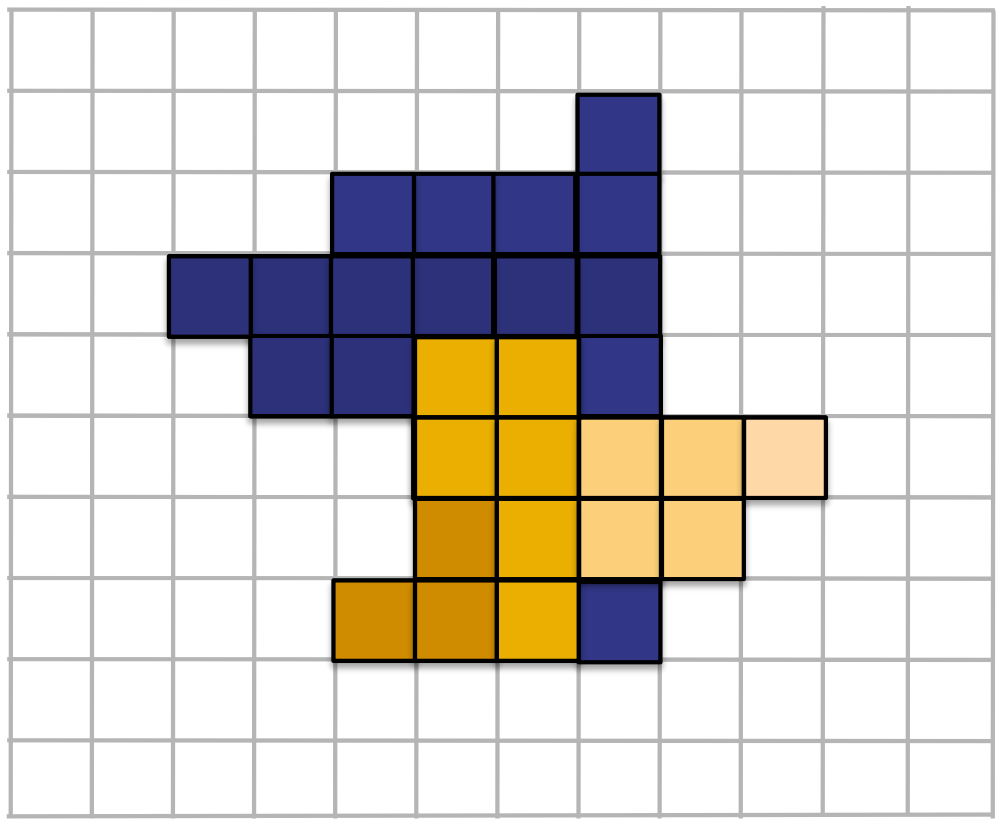


EACH FRAGMENT IS PROCESSED INDEPENDENTLY

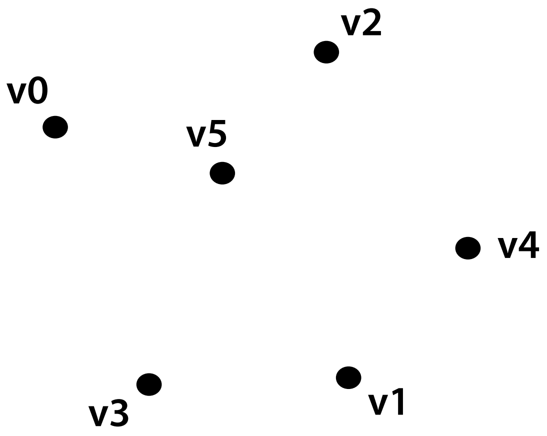

Vertices → Vertex processor → Clipper and primitive assembler → Rasterizer → Fragment processor → Pixels

Pixel operations

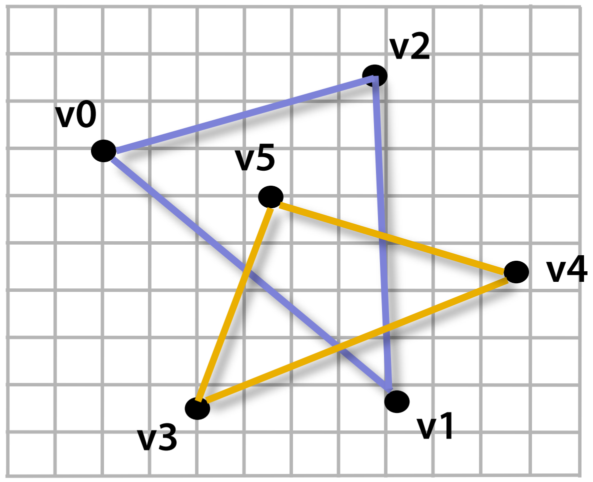
Fragments are blended into the frame buffer at their pixel locations (z-buffer determines visibility)


Pixels

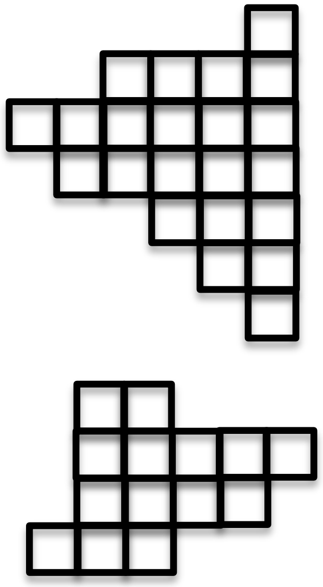
Pipeline entities



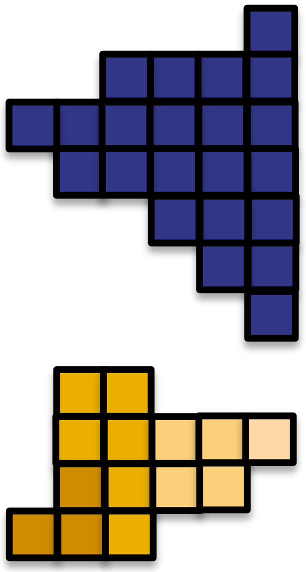
Vertices



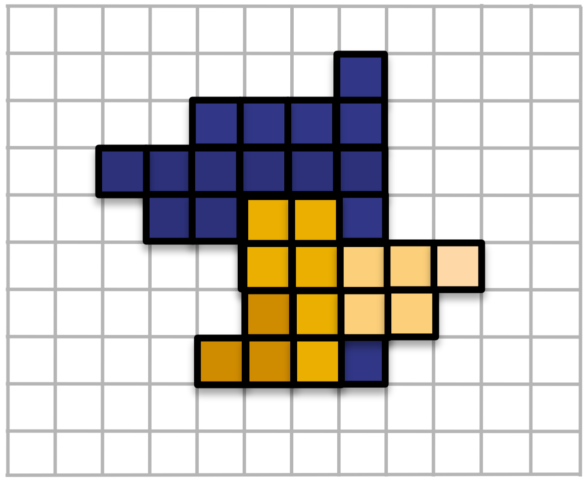
Primitives



Fragments

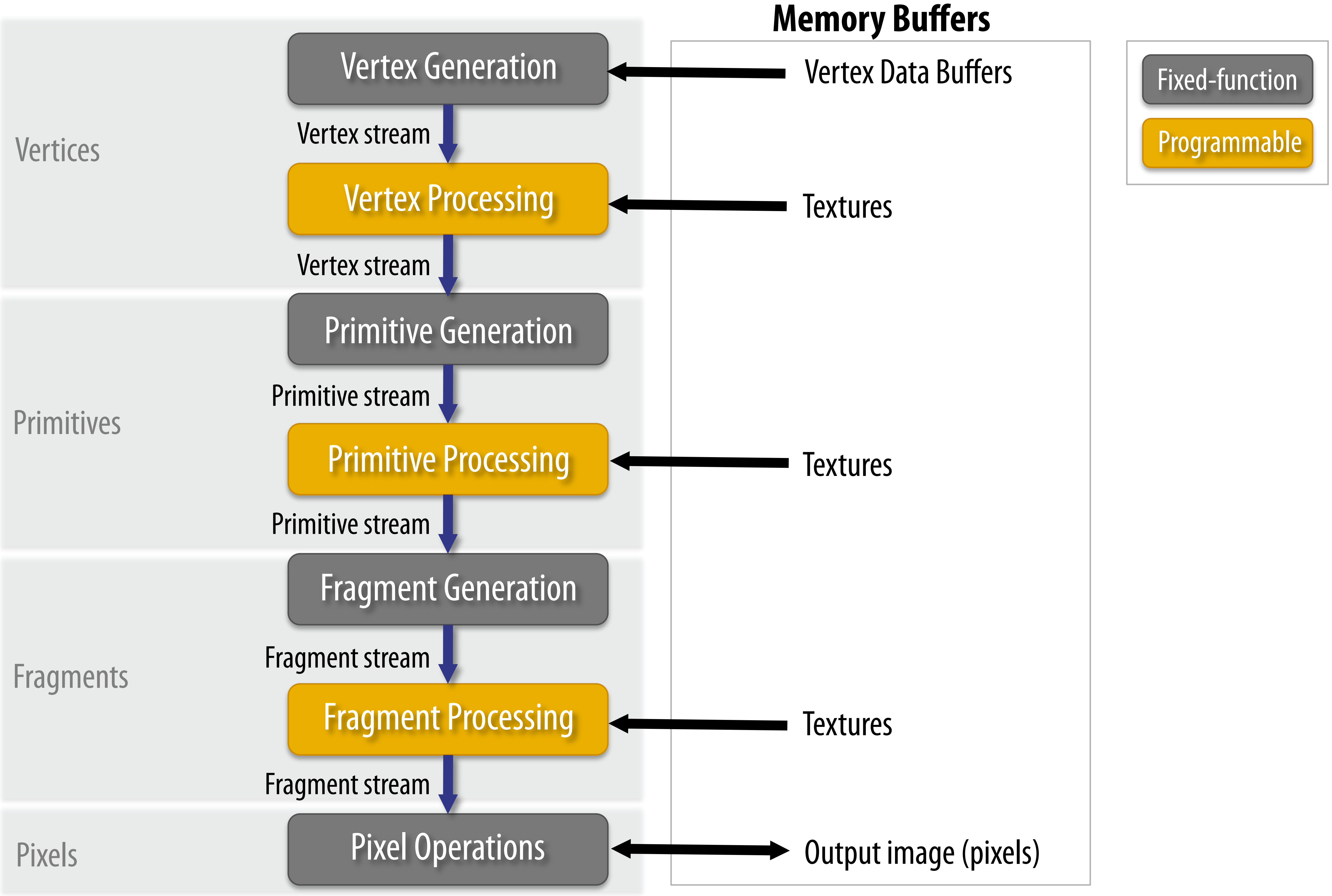


Fragments (shaded)



Pixels

Graphics pipeline


Memory Buffers
Vertices
Primitives
Fragments
Pixels
Vertex Generation
Vertex stream
Vertex Processing
Vertex stream
Primitive Generation
Primitive stream
Primitive Processing
Primitive stream
Fragment Generation
Fragment stream
Fragment Processing
Fragment stream
Pixel Operations
Vertex Data Buffers
Textures
Textures
Textures
Output image (pixels)
Fixed-function
Programmable