Graphics Pipeline
Rendering approaches

1. object-oriented
   foreach object ...

2. image-oriented
   foreach pixel ...

vertices → 3D rendering pipeline → image
Z-buffer Rendering

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

GRAPHICS PIPELINE

3D Polygons  GRAPHICS PIPELINE  Image Pixels
Pipelining operations

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

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

Angel and Shreiner
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)
Other polygons

triangulation
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
Then organized into primitives that are clipped and culled...
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
Fragments are shaded to compute a color at each pixel
Pixel operations

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

Vertices

Primitives

Fragments

Fragments (shaded)

Pixels
## Graphics pipeline

<table>
<thead>
<tr>
<th>Vertices</th>
<th>Vertex Generation</th>
<th>Primitive Generation</th>
<th>Fragment Generation</th>
<th>Pixel Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vertices</td>
<td>Vertex stream</td>
<td>Primitive stream</td>
<td>Fragment stream</td>
<td>Pixels</td>
</tr>
<tr>
<td>Primitives</td>
<td>Vertex stream</td>
<td>Primitive stream</td>
<td>Fragment stream</td>
<td>Pixels</td>
</tr>
<tr>
<td>Fragments</td>
<td>Primitive stream</td>
<td>Primitive stream</td>
<td>Fragment stream</td>
<td>Pixels</td>
</tr>
<tr>
<td>Pixels</td>
<td>Fragment stream</td>
<td>Fragment stream</td>
<td>Pixel Operations</td>
<td>Output image (pixels)</td>
</tr>
</tbody>
</table>

### Memory Buffers
- Vertex Data Buffers
- Textures
- Output image (pixels)