Graphics Pipeline
Rendering approaches

**image-oriented**
foreach pixel ...

**object-oriented**
foreach object ...

![Diagram of 3D rendering pipeline]

- Camera
- Light Source
- View Ray
- Shadow Ray
- Scene Object
- Image
- 3D rendering pipeline

vertices → 3D rendering pipeline → image
Pipelining operations

An arithmetic pipeline that computes $c + (a \times b)$
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
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
Point and line segment types

GL_POINTS

GL_LINES

GL_LINE_STRIP

GL_LINE_LOOP

[Angel and Shreiner]
OpenGL polygons

- Only triangles are supported (in latest versions)

GL_POINTS  GL_TRIANGLES  GL_TRIANGLE_STRIP  GL_TRIANGLE_FAN
Graphics Pipeline
(slides courtesy K. Fatahalian)
Vertex processing

Vertices are transformed into “screen space”
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”

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

Fragments are shaded to compute a color at each pixel

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)
Modern OpenGL/Vulkan pipeline

University of California Riverside
1992: Initially fixed functionality pipeline
2004: Added programmable shaders
2008: Fixed pipeline deprecated
2009: Fixed paths removed
   - Still available for compatibility
   - Fixed pipe emulated with shaders
Pipeline

- **Input**: geometry
- **Output**: image (on screen)
- **Programmable stages**

Diagram:
- vertex input
- vertex shader
- tessellation
- geometry shader
- post vertex processing
- primitive assembly
- rasterization
- fragment shader
- per-sample operations
Vertex input

- Supply input data to pipeline
- Stream of vertices
- Indices (for meshes)
Vertex shader

- Programmable (user-defined)
- For per-vertex operations
- Used to transform vertices
- Can do other things here
  - Eg, per-vertex lighting
  - Define colors at vertices
  - Interpolate within triangles
Tessellation

- Programmable (user-defined)
- Optional stage
- For subdividing primitives

vertex input

vertex shader

tessellation

geometry shader

post vertex processing

primitive assembly

rasterization

fragment shader

per-sample operations

Image source: [?]
Geometry shader

- Programmable (user-defined)
- Optional stage
- Input: one primitive (at a time)
- Output: (many) primitives
- Possible uses:
  - Instancing
  - Turn lines into curves
  - Draw points as squares, diamonds, or stars (plots!)
  - Bad use: tessellation
Clipping
- removes (part of) primitive
- if outside image
- if too close/far

Perspective divide
- \((x, y, z, w) \rightarrow (\frac{x}{w}, \frac{y}{w}, \frac{z}{w})\)
- We will see this later
Primitive assembly

- Turn primitives into \textit{base} primitives
  - Triangle strip to triangles
  - Line loop to segments
- Back-face culling
  - do not render the backside
  - cannot see it anyway
Rasterization

- **Input:** primitive (e.g., triangles)
- **Output:** fragments
Fragment shader

- Programmable (user-defined)
- Input: fragment data
  - interpolated vertex data
- Output: depth, color
- Compute color of pixel
  - Phong shading
  - texture mapping
  - bump mapping
Per-sample operations

- Z-buffering (occlusion)
  - Discard hidden pixels
  - Optimization: before fragment shader if possible
- Masking, blending
- Storing results