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

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

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.

Diagram:
- Vertices: (x1,y1), (x2,y2), (x3,y3)
- Edges: E1, E2, E3
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

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

- Color         `glClearColor(1.0, 1.0, 1.0, 1.0);`
- Point size    `glPointSize(2.0);`
- Line width    `glLineWidth(3.0);`
Coordinate systems and transformations
Viewing transformations

Figure 1. Block diagram of OpenGL.
Viewing transformations

- viewing coordinates are based on the position and orientation of a the virtual camera
- 2D projection of the scene
- normalized device coordinates
- finally we get device or screen coordinates
- modeling -> world -> viewing -> projection -> normalized -> device
Viewing transformations

Projection: map 3D scene to 2D image

- viewing coordinates are based on the position and orientation of a the virtual camera
- 2D projection of the scene
- normalized device coordinates
- finally we get device or screen coordinates
- modeling -> world -> viewing -> projection -> normalized -> device
Orthographic projection

- square or rectangular viewing volume
- anything outside volume is not drawn
- all objects of same dimension appear the same regardless of distance from camera
OpenGL Orthogonal Viewing

```python
glOrtho(xmin, xmax, ymin, ymax, near, far)
glOrtho(left, right, bottom, top, near, far)
```

Clipping volume for an orthographic projection

near and far measured from camera
Perspective projection

Orthographic, or parallel projection
- square or rectangular viewing volume
- anything outside volume is not drawn
- all objects of same dimension appear the same regardless of distance from camera
OpenGL Perspective Viewing

\[ \text{glFrustum}(x_{\text{min}}, x_{\text{max}}, y_{\text{min}}, y_{\text{max}}, \text{near}, \text{far}) \]

Clipping volume (frustrum) for a perspective projection
Using Field of View

With `glFrustum` it is often difficult to get the desired view. `gluPerpective(fovy, aspect, near, far)` often provides a better interface.

\[
\text{aspect} = \frac{w}{h}
\]

\[
\text{front plane}
\]
Viewport transformation

Angel and Shreiner
Viewport transformation

Viewport is the whole window

Viewport is the lower left corner
Viewing transformations

- **Camera transformation**: rigid body transformation that places the camera at the origin and in a convenient orientation
- **Projection transformation**: project to canonical view volume all coordinates end up between 0 and 1 or -1 and 1
- **Viewport or windowing transformation**: normalized coordinates to pixel coordinates
Scalars, points and vectors
Cartesian coordinates

OpenGL Super Bible, 5th Ed.

two-dimensional  three-dimensional
Points

- A point is a location in space

Reference: Angel, Chapter 3
Vectors

- A vector is a directed line segment
Vectors

- A vector is a directed line segment
Vectors

- Vectors have length and direction

\[ l = |V| = \sqrt{x^2 + y^2 + z^2} \]
Vectors

- Vectors have length and direction

$$\hat{V} = \frac{V}{|V|}$$

$$|\hat{V}| = 1$$
Vectors

- These vectors are all the same
Vectors

- Vector scaling

\[ V \rightarrow 2V \]
Vectors

- Vector addition \( Z = V + W \)
  
  “head-to-tail rule“
<whiteboard>