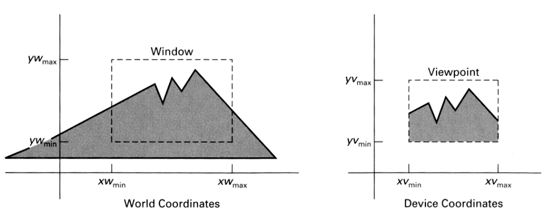
4. Graphics Programming

Optional reading

- Angel, sections 1.6-1.8, chapter 2
- Foley et al., section 5.4,
- Woo et al., chapters 1, 2, 8

Specifying a view in 2D

How do you specify a view of a 2D picture?



Most graphics systems let you specify:

- the part of a picture to display (the **window**)
- the place to display that picture on the screen (the **viewport**)

Specifying a view in 2D, cont.

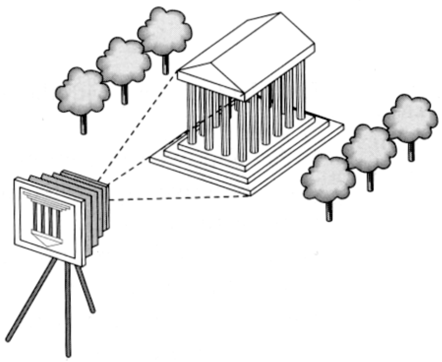
Typically, the picture is defined in any convenient coordinate system, called **world coordinates**.

The viewport is generally specified in coordinates in [0,1]x[0,1] - called **normalized device coordinates**.

Ultimately, these coordinates are mapped to integer pixel coordinates - also known as **device coordinates** or **screen coordinates**.

- glViewport(x, y, w, h);

The 3D synthetic camera model

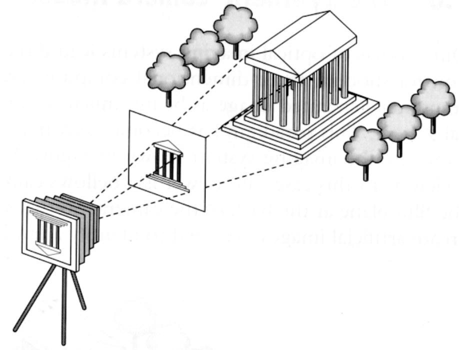


The **synthetic camera model** is a paradigm for creating images of 3D geometry.

It involves two components, specified *independently:*

- objects (a.k.a. **geometry**)
- viewer (a.k.a. **camera**)

Imaging with the synthetic camera

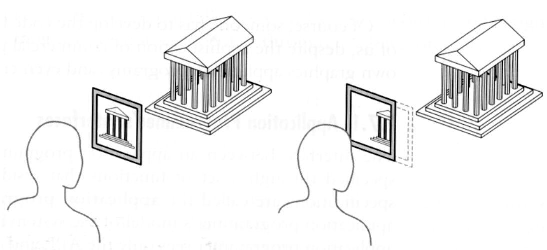


The image is rendered onto an **image plane** or **projection plane** (usually in front of the camera).

Projectors emanate from the **center of projection** (COP) at the center of the lens (or pinhole).

The image of an object point *P* is at the intersection of the projector through *P* and the image plane.

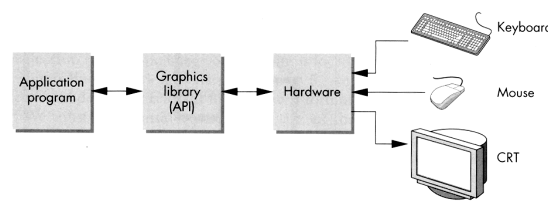
Clipping



We think of the image plane as having a finite (rectangular) extent.

Objects are **clipped** to a **clipping rectangle** or **clipping window**.

Graphics APIs



An application programmer's interface (API) provides an interface between the application code and the hardware.

Most popular graphics APIs (OpenGL, DirectX, PHIGS, GKS-3D) are based on the synthetic camera model.

Have functions to specify:

- objects
- viewer
- light sources
- material properties

OpenGL objects

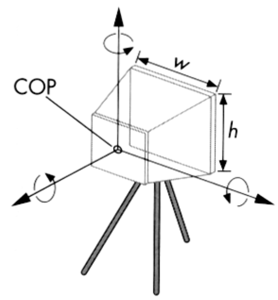
Most APIs support several different geometric **primitives**.

OpenGL provides:

- points `(GL_POINTS)`
- line segments `(GL_LINES)`
- polylines `(GL_LINE_STRIP)`
- unfilled polygons `(GL_LINE_LOOP)`
- filled polygons `(GL_POLYGON)`
- triangles `(GL_TRIANGLES)`
- quadrilaterals `(GL_QUADS)`
- strips `(GL_TRIANGLE_STRIP, GL_QUAD_STRIP)`
- fans `(GL_TRIANGLE_FAN)`

It also lets you read and write pixels in the framebuffer.

Specifying a viewer



Camera specification requires four kinds of parameters:

- *Position:* the COP.
- *Orientation:* rotations about axes with origin at the COP.
- *Focal length:* determines the size of the image on the film plane, or the **field of view**.
- *Film plane:* its width and height, and possibly orientation.

Specifying lights and materials

Light sources usually defined by:

- location
- strength
- color
- directionality

Materials usually defined by:

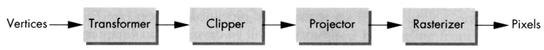
- various shading parameters
- texture maps

OpenGL rendering styles

OpenGL supports a variety of rendering styles:

- Wireframe
  - with depth-cueing
  - with antialiasing
- Visible polygons
  - with flat shading
  - with smooth (**Gouraud**) shading
  - with texture maps and shadows
  - with motion blur
  - with atmospheric effects

The geometric pipeline



Many commercial graphics workstations use a **pipeline** architecture, implemented in hardware, for processing geometry.

Works well because:

- Lots of data that is processed similarly
- Well-decomposed computation

Q: What's the downside of large pipelines?

The graphics pipeline

The pipeline metaphor can be extended to encompass just about everything we do in 3D graphics:

modeling
⇩
animation
⇩
scene "snapshot"
⇩
transformation
⇩
clipping
⇩
lighting and shading
⇩
hidden surface
⇩
projecting
⇩
rasterizing
⇩
compositing
⇩
post-processing

Summary

Here's what you should take home from this lecture:

- All the **boldfaced terms**.
- The basic idea of the synthetic camera model and how its basic components are specified.
- The basic concept of the geometry and graphics pipelines.