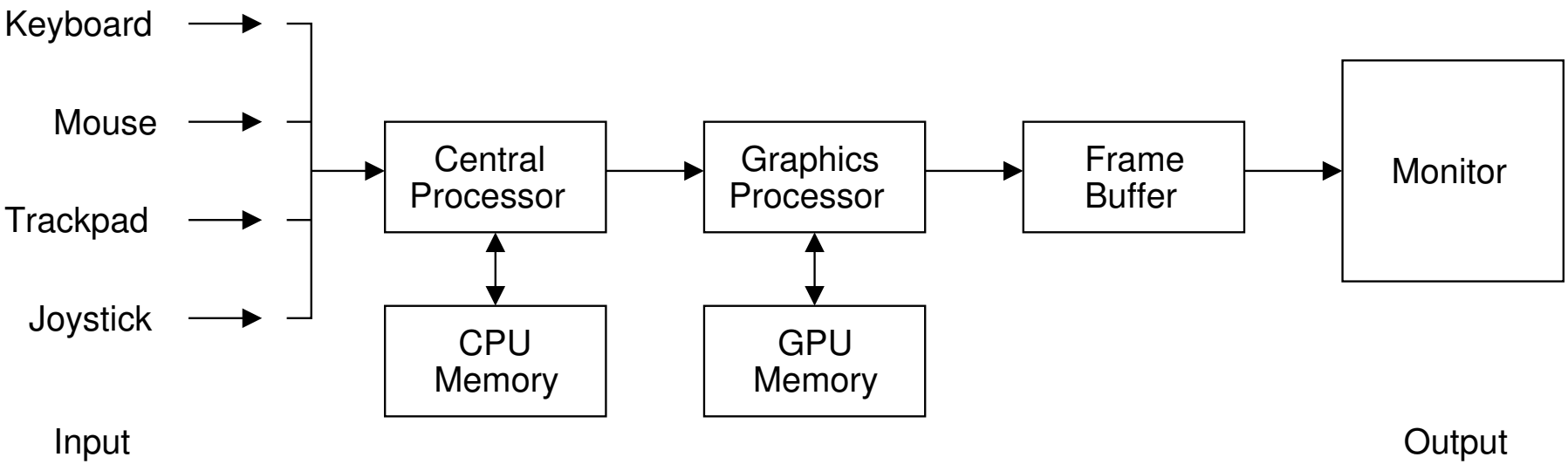
Introduction to Computer Graphics

Thumrongsak Kosiyatrakul
tkosiyat@cs.pitt.edu

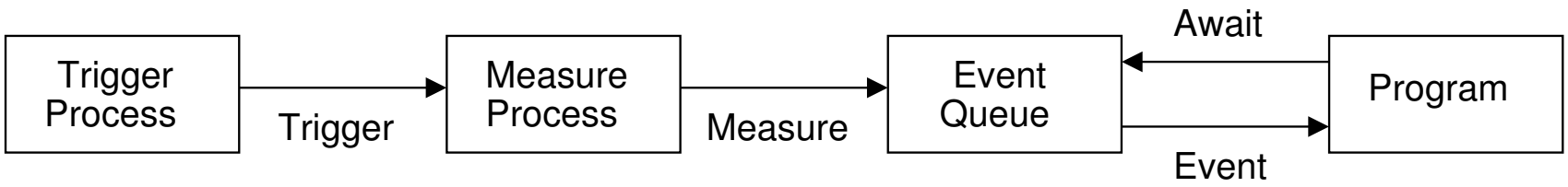
Application of Computer Graphics

- Display of Information
- Design
- Simulation and Animation
- User Interfaces

A Graphics System



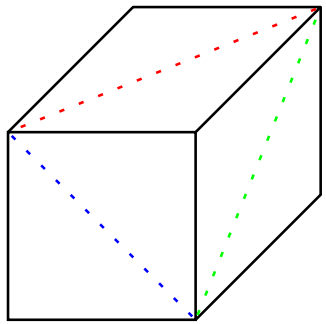
Event Mode



- When an event occurs:
  - Mouse click, drag, button (up or down)
  - Keyboard
  - Timeout (idle)
- A user-defined function in the program will be called

Objects

- An object can be constructed using a number of geometric primitives:
  - points,
  - lines,
  - polygons.
- A simple cube:



  - There are 6 faces
  - Each face is defined by two triangles (polygons)
- A complex object can be constructed using a number of simple objects.

Geometric Primitives

- A geometric primitive can be defined by a series of vertices
  - A point (1 vertice)
  - A line (2 vertices)
  - A triangle (3 vertices)
- The location of a vertice should be independent of
  - image transformation (transform later)
  - viewer (again handle this later)
- A series of vertices of an object can be generated
  - manually,
  - by a series of formulas (equations), or
  - both.
- **Make it simple**

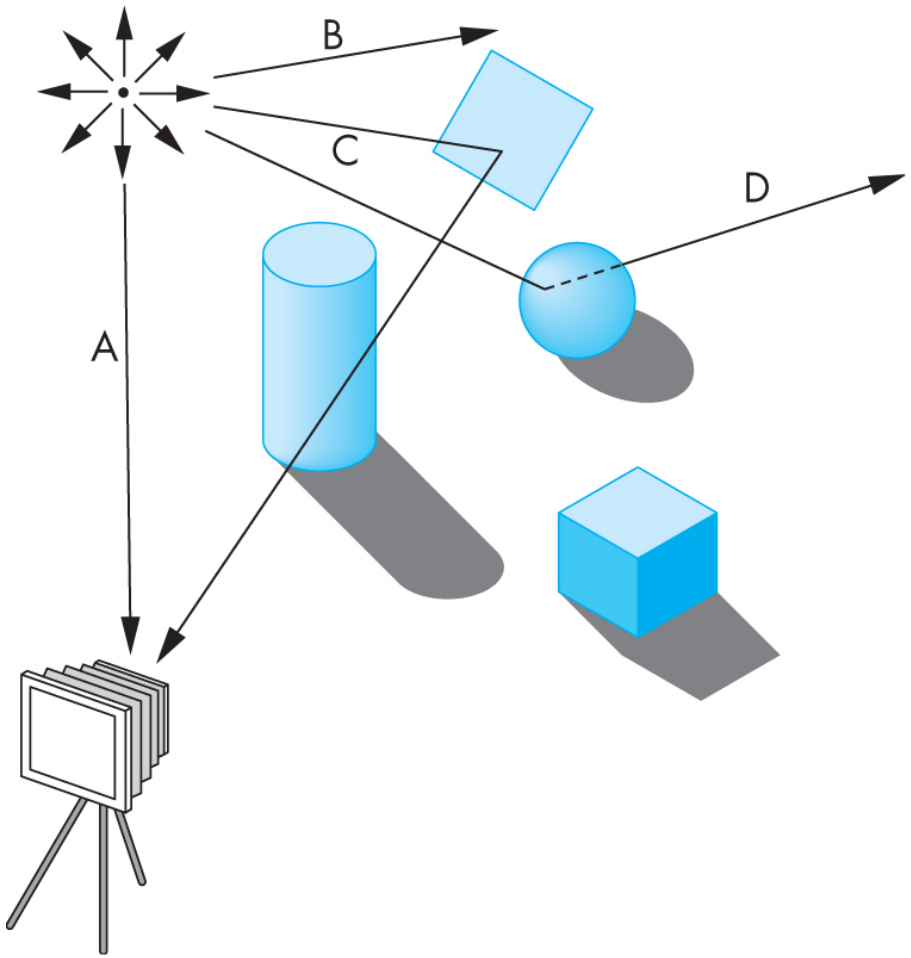
Viewers

- A viewers forms an image
  - human, camera, printer, etc.
- Given a set of objects in a world, an image of those objects depends on various factors:
  - Location of the viewer,
  - Direction of the viewer,
  - Lens (naked eyes vs telephoto vs wide angle)

Light and Images

- Without lighting effect, an image of an object will be flatten
  - A sphere looks like a circle.
  - A corner of a wall looks like a flat wall.
- A light source consists of various properties such as colors, intensity of each color, type of light source, and location.
- Surfaces of an objects also has light-reflecting properties such as reflexivity of each color, distance from light source, and direction of reflection.
- These factors effect how an image of an object is formed.

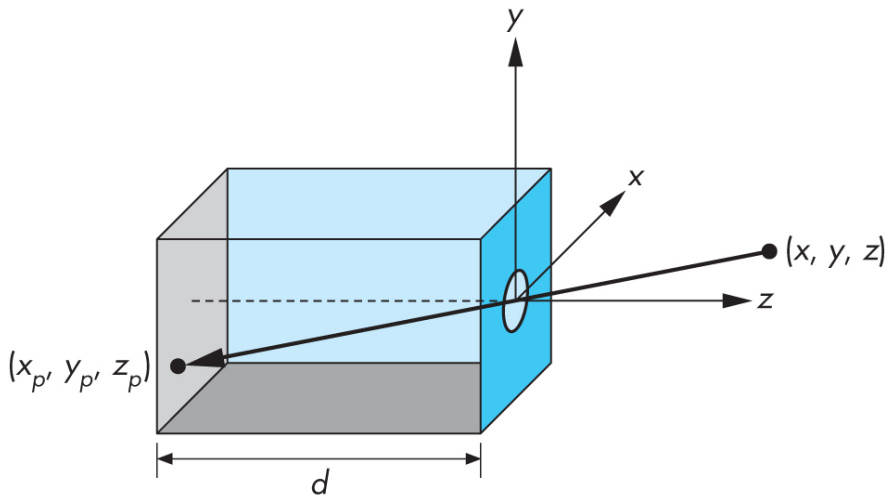
Imaging Models



- A ray starts from a light source and travels to infinity
- Image of objects is formed based on rays that enter the camera

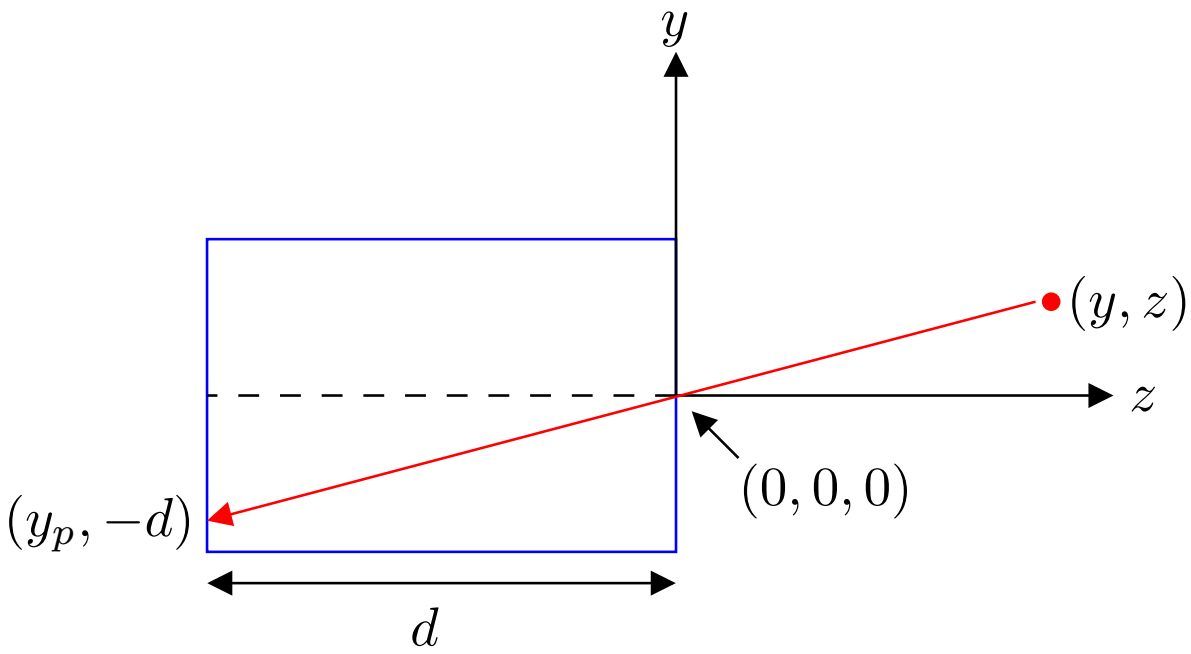
Imaging Systems

- Image formations depend on imaging systems as well, type of cameras, human eye, etc
- We can model how a point appear on the back of a camera mathematically
- Pinhole camera



- A point (x, y, z) appears on the film at point (x_p, y_p, z_p)

Pinhole Camera



- Note that ratio must be the same

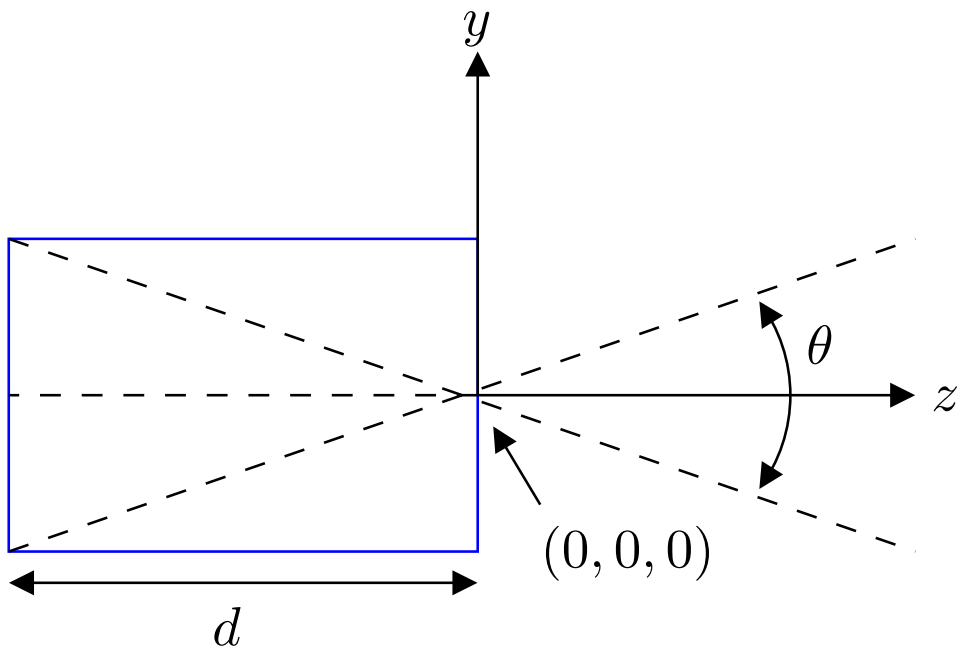
$$\frac{y_p}{d} = \frac{y}{z} \rightsquigarrow y_p = \frac{d \times y}{z} \rightsquigarrow y_p = \frac{y}{z/d}$$

- Similarly,

$$x_p = \frac{x}{z/d}$$

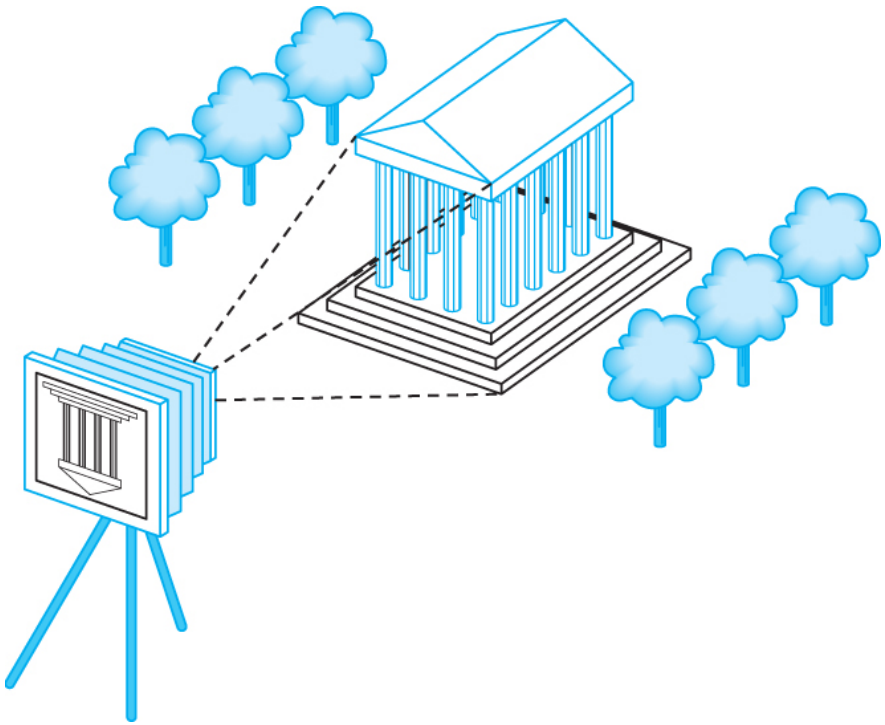
Pinhole Camera

- We can only see objects that are projected onto the back of the camera
- This is called **angle of view** (θ) as shown below



The Synthetic-Camera Model

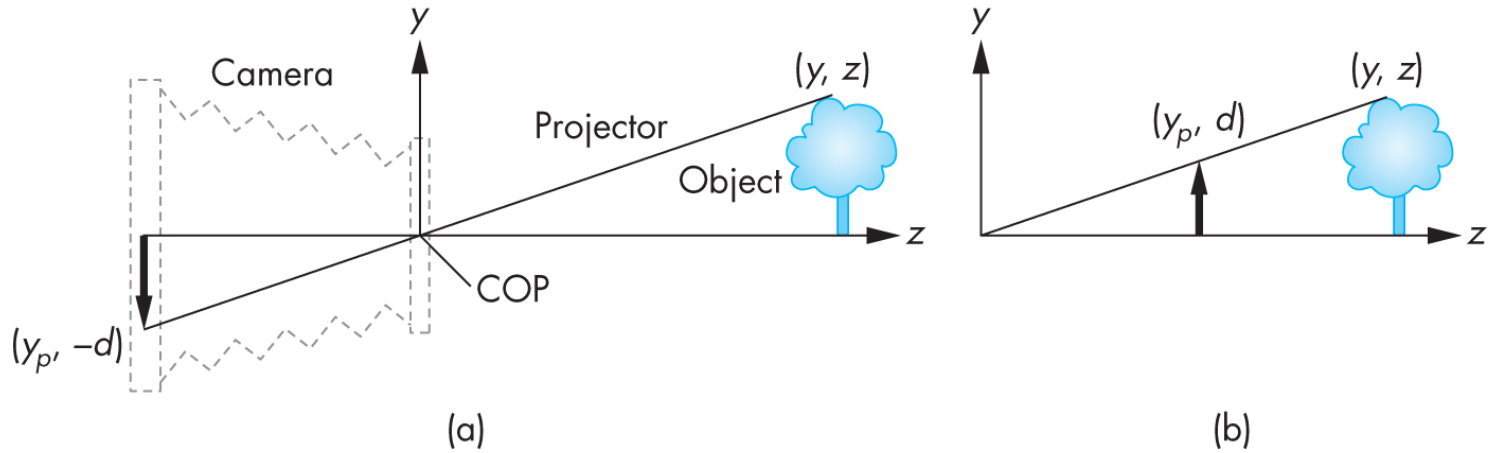
- We are trying to form an image the same way as in an optical system (e.g., camera)



- Specification of objects (size, orientation, etc) is independent of the specification of the camera (lens, size of film, etc)

The Synthetic-Camera Model

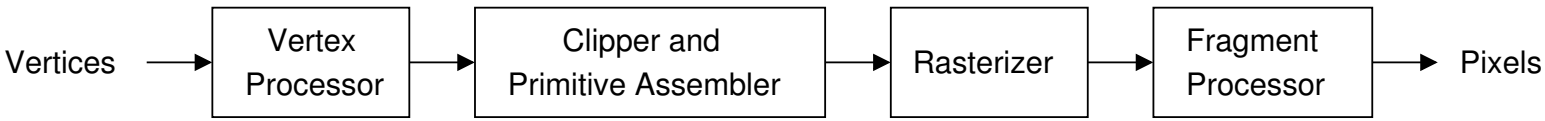
- Image can be formed the same way as in pinhole camera



- A line from a point of an object to the back of the camera is called a **projector**.
- All projectors must go pass through the center of the camera's lens called the **center of projection (COP)**
- We can simply flip the image by artificially move the back of the camera to the front (the **projection plane**).
- Objects or parts of objects that are not in the angle of view are considered **clipped**.

Graphics Pipeline

- A scene consists of multiple objects, each object comprises of a set of primitives, and each primitive comprises of a set of vertices.
- A scene may contains millions of vertices.
- Graphic hardware need to turn these vertices into pixels in frame buffer



Graphics Pipeline

- Vertex Processing processes each vertex independently
  - coordinate transformation
  - color
  - lighting effect
- Clipping and Primitive Assembly
  - clipping volume based on field of view
  - primitive by primitive basis
- Rasterization
  - converts primitives in terms of vertices into pixels for the frame buffer
  - results in a set of **fragments** for each primitive
- Fragment Processing
  - turns visible fragments into pixel and update the frame buffer
  - performs hidden surface removal
- In programmable pipeline, vertex and fragment processors can be programmed by an application