

1. Graphics Systems and Models

What is Computer Graphics?

Computer Graphics

- Computer graphics deals with all aspects of creating images with a computer
 - Hardware
 - Software
 - Applications

Example

- Where did this image come from?

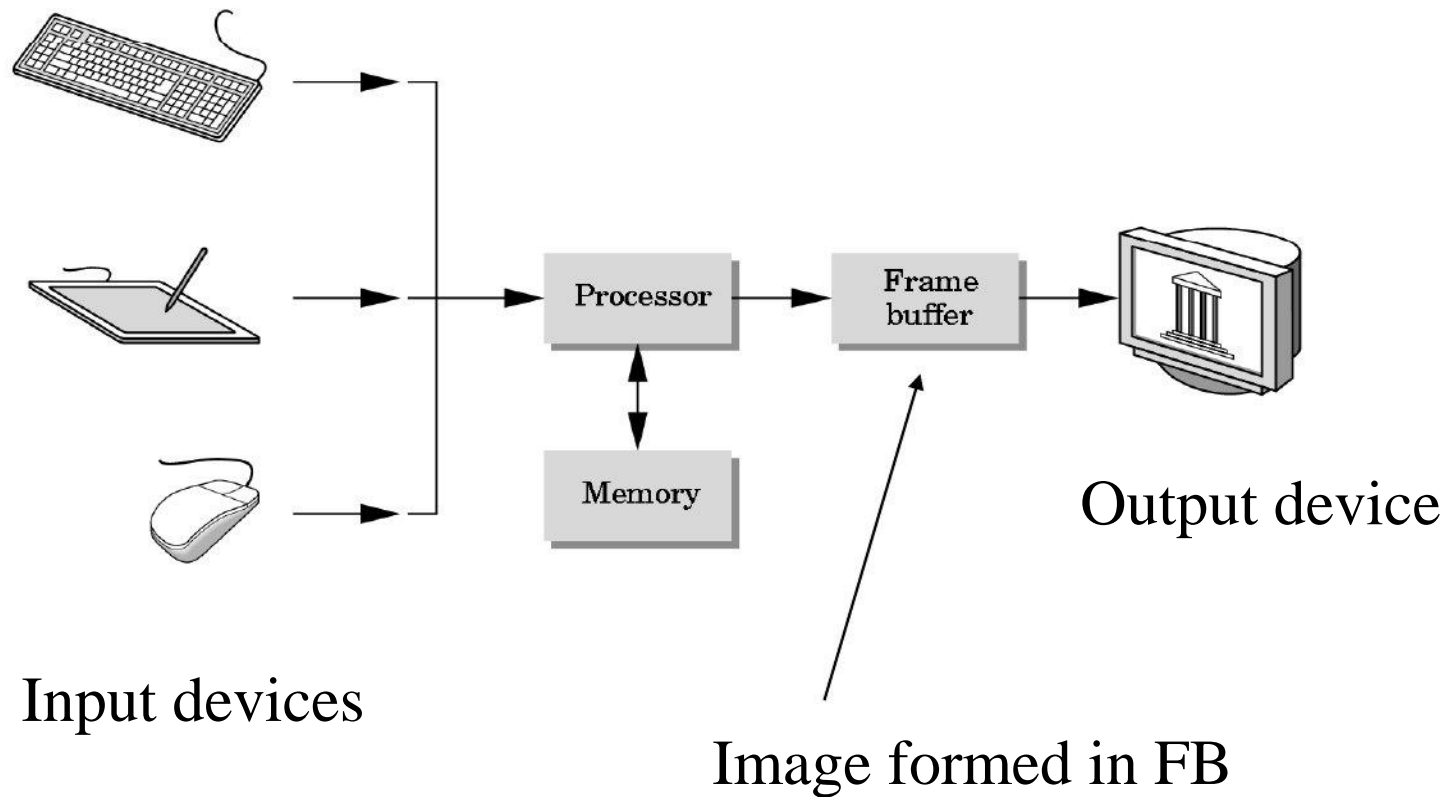


- What hardware/software did we need to produce it?

Preliminary Answer

- Application: The object is an artist's rendition of the sun for an animation to be shown in a domed environment (planetarium)
- Software: **Maya** for modeling and rendering (Maya is built on top of OpenGL)
- Hardware: PC with graphics card for modeling and rendering

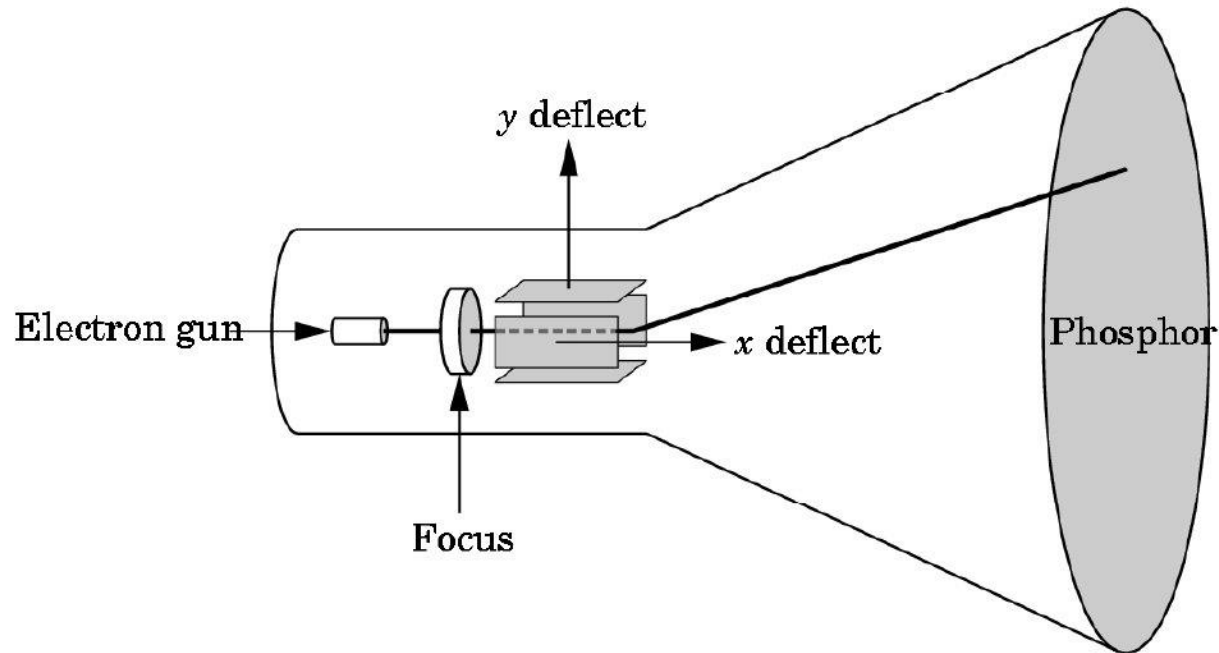
Basic Graphics System



Computer Graphics: 1950-1960

- Computer graphics goes back to the earliest days of computing
 - Pen plotters
 - Printing devices for vector graphics
 - Simple displays using A/D converters to go from computer to calligraphic CRT
- Cost of refresh for CRT too high
 - Computers slow, expensive, unreliable

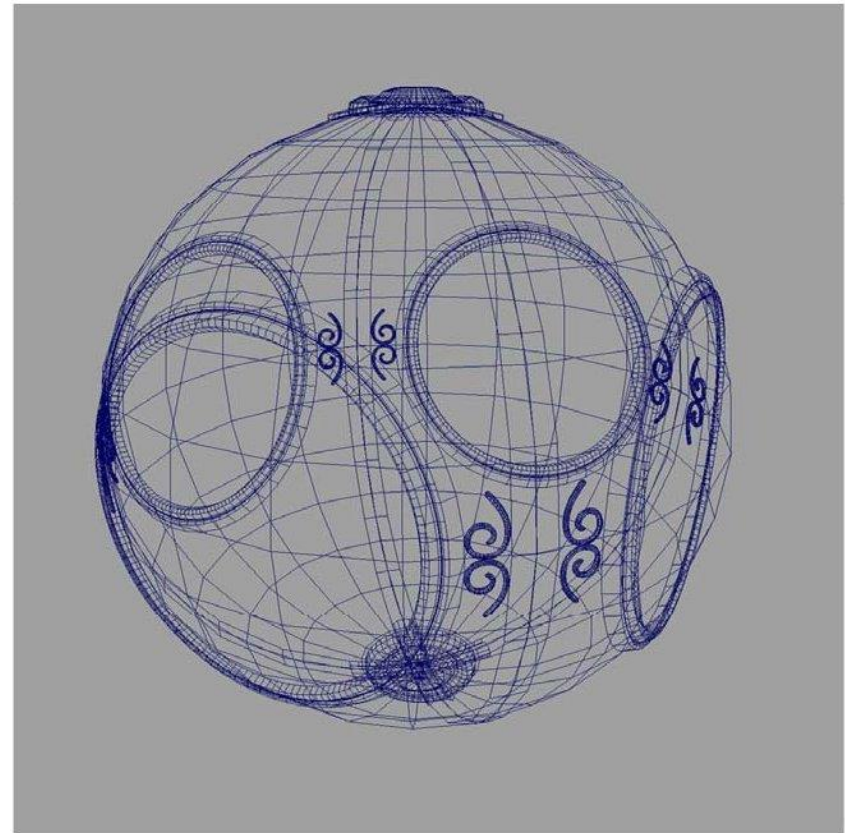
CRT



Can be used either as a line-drawing device (calligraphic) or to display contents offframe buffer (raster mode)

Computer Graphics: 1960-1970

- **Wireframe** graphics
 - Draw only lines
- Sketchpad
- Display Processors
- Storage Tube



wireframe representation
of sun object

Ivan Sutherland (1963) - Sketchpad



First GUI

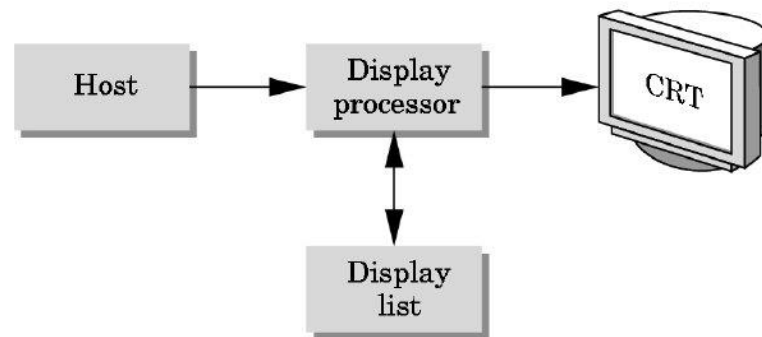
- pop-up menus
- constraint-based drawing
- hierarchical modeling

Sketchpad

- Ivan Sutherland's PhD thesis at MIT
 - Recognized the potential of man-machine interaction
 - Loop
 - Display something
 - User moves light pen
 - Computer generates new display
 - Sutherland also created many of the now common algorithms for computer graphics

Display Processor

- Rather than have the host computer try to refresh display use a special purpose computer called a display processor (DPU)



- Graphics stored in **display list** (display file) on display processor
- Host compiles display list and sends to DPU

Direct View Storage Tube

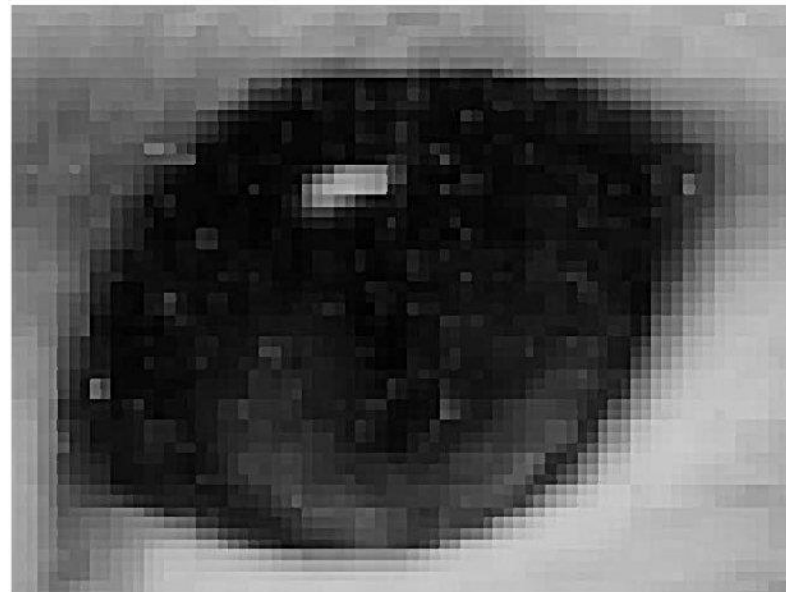
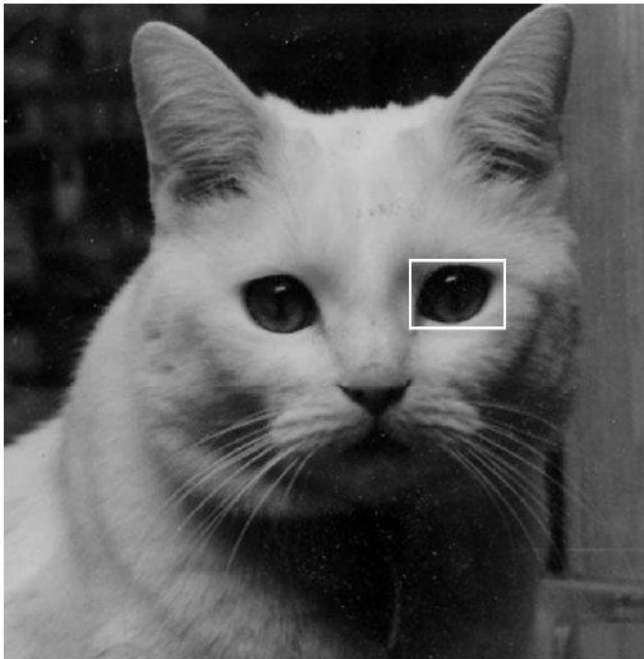
- Created by Tektronix
 - Did not require constant refresh
 - Standard interface to computers
 - Allowed for standard software
 - Plot3D in Fortran
 - Relatively inexpensive
 - Opened door to use of computer graphics for CAD community

Computer Graphics: 1970-1980

- Raster Graphics
- Beginning of graphics standards
 - International Federation of Information Processing Societies (IFIPS)
 - Graphical Kernel System (GKS): European effort
 - Becomes ISO 2D standard
 - Core: North American effort
 - 3D but fails to become ISO standard
- Workstations and PCs

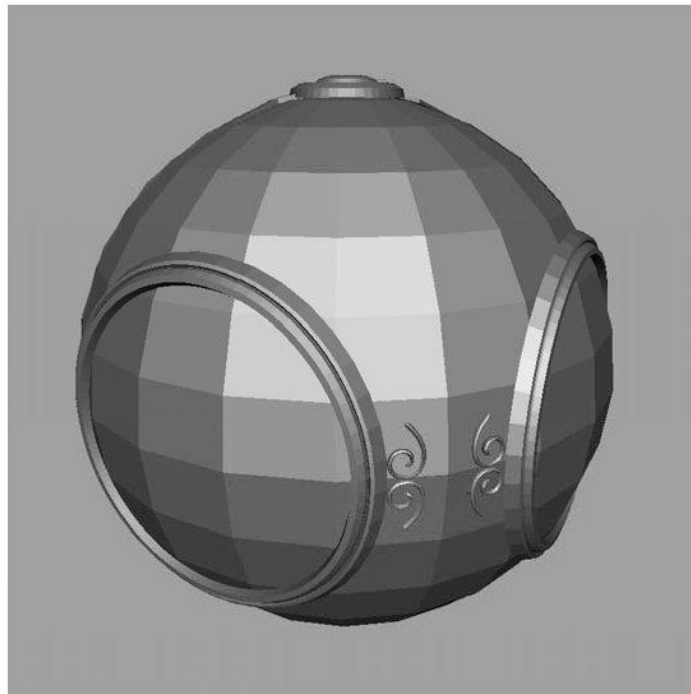
Raster Graphics

- Image produced as an array (the raster) of picture elements (pixels) in the frame buffer



Raster Graphics

- Allows us to go from lines and wire frame images to filled polygons

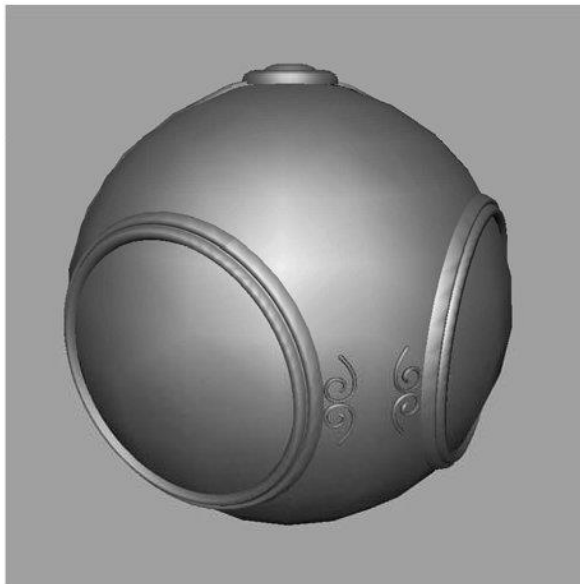


PCs and Workstations

- Although we no longer make the distinction between workstations and PCs, historically they evolved from different roots
 - Early workstations characterized by
 - Networked connection: client-server model
 - High-level of interactivity
 - Early PCs included frame buffer as part of user memory
 - Easy to change contents and create images

Computer Graphics: 1980-1990

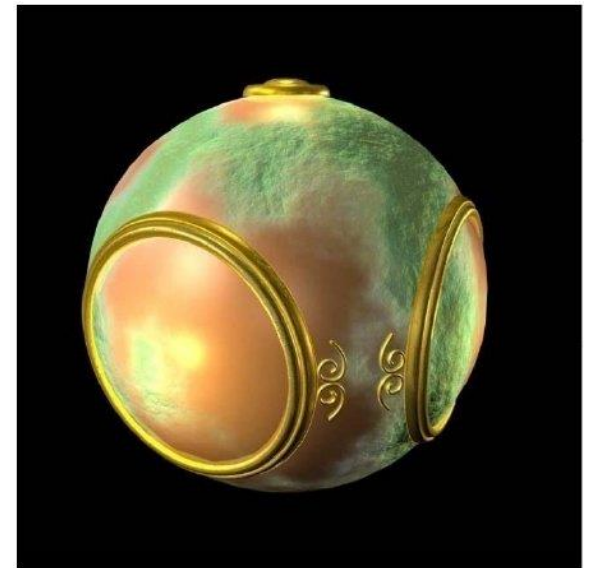
Realism comes to computer graphics



smooth shading



environment



bump mapping

Computer Graphics: 1980-1990

- Special purpose hardware
 - **Silicon Graphics** geometry engine
 - VLSI implementation of graphics pipeline
- Industry-based standards
 - **PHIGS** (Programmer's Hierarchical Interactive Graphics System)
 - **RenderMan** by Pixar
- Networked graphics: X Window System
- Human-Computer Interface (HCI)

Computer Graphics: 1990-2000

- OpenGL API
- Completely computer-generated feature-length movies are successful (Toy Story)
- New hardware capabilities
 - Texture mapping
 - Blending
 - Accumulation, stencil buffers

Computer Graphics: 2000-

- Photorealism
- Graphics cards for PCs dominate market
 - Nvidia, ATI
- Game boxes and game players determine direction of market
- Computer graphics routine in movie industry: Maya, Lightwave
- Programmable pipelines

Image Formation

Objectives

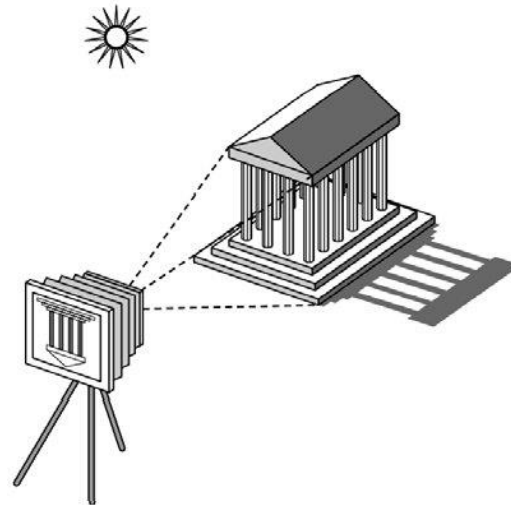
- Fundamental imaging notions
- Physical basis for image formation
 - Light
 - Color
 - Perception
- Synthetic camera model

Image Formation

- In computer graphics, we form images which are generally two dimensional using a process analogous to how images are formed by physical imaging systems
 - Cameras
 - Microscopes
 - Telescopes
 - Human visual system

Elements of Image Formation

- Objects
- Viewer
- Light source(s)



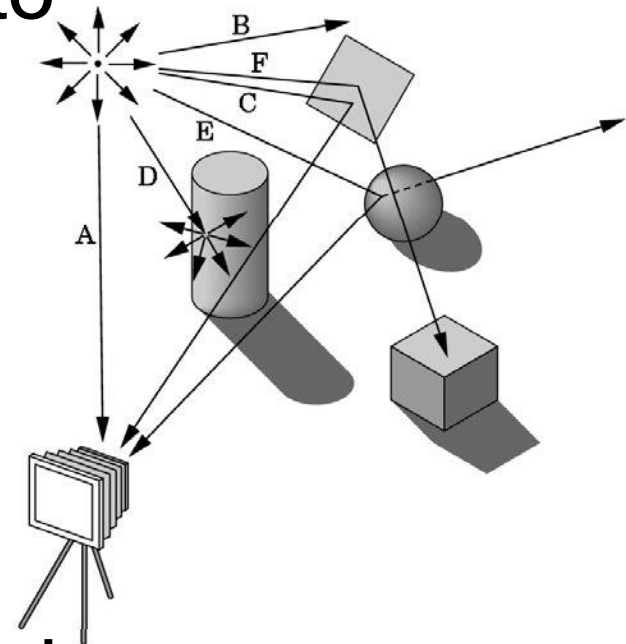
- Attributes that govern how **light interacts with the materials** in the scene
- Note the independence of the objects, the viewer, and the light source(s)

Light

- Light is the part of the electromagnetic spectrum that causes a reaction in our visual systems
- Generally these are wavelengths in the range of about 350-750 nm (nanometers)
- Long wavelengths appear as reds and short wavelengths as blues

Ray Tracing and Geometric Optics

One way to form an image is to follow rays of light from a point source finding which rays enter the lens of the camera. However, each ray of light may have multiple interactions with objects before being absorbed or going to infinity.

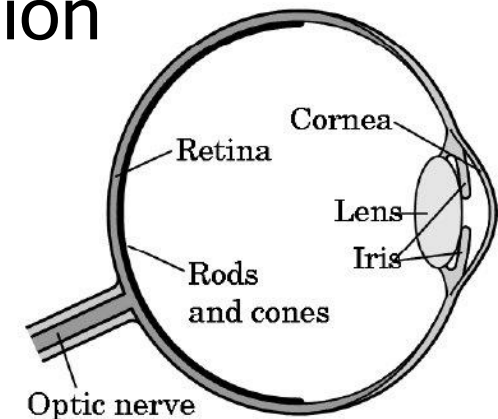


Luminance and Color Images

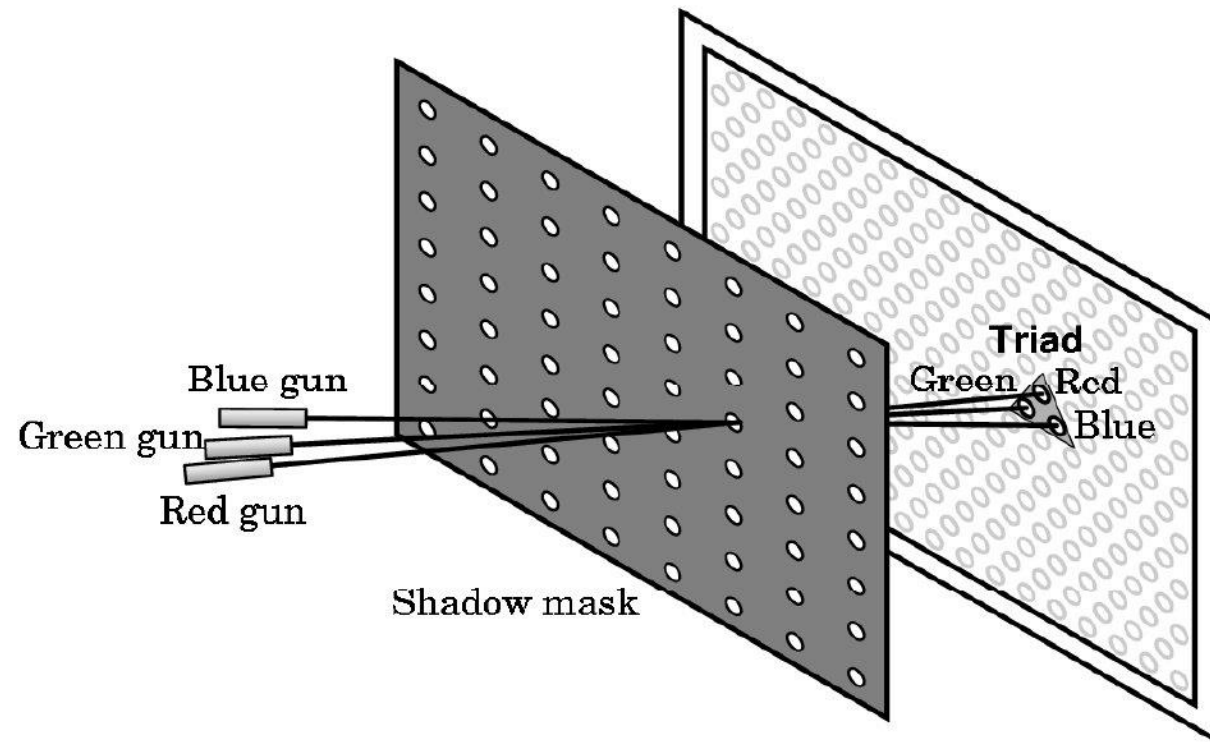
- Luminance Image
 - Monochromatic
 - Values are gray levels
 - Analogous to working with black and white
- Color Image
 - Has perceptual attributes of hue, saturation, and lightness
 - Do we have to match every frequency in visible spectrum? No!

Three-Color Theory

- Human visual system has two types of sensors
 - Rods: monochromatic, night vision
 - Cones
 - Color sensitive
 - Three types of cones
 - Only three values (the tristimulus values) are sent to the brain
- Need only match these three values
 - Need only three primary colors



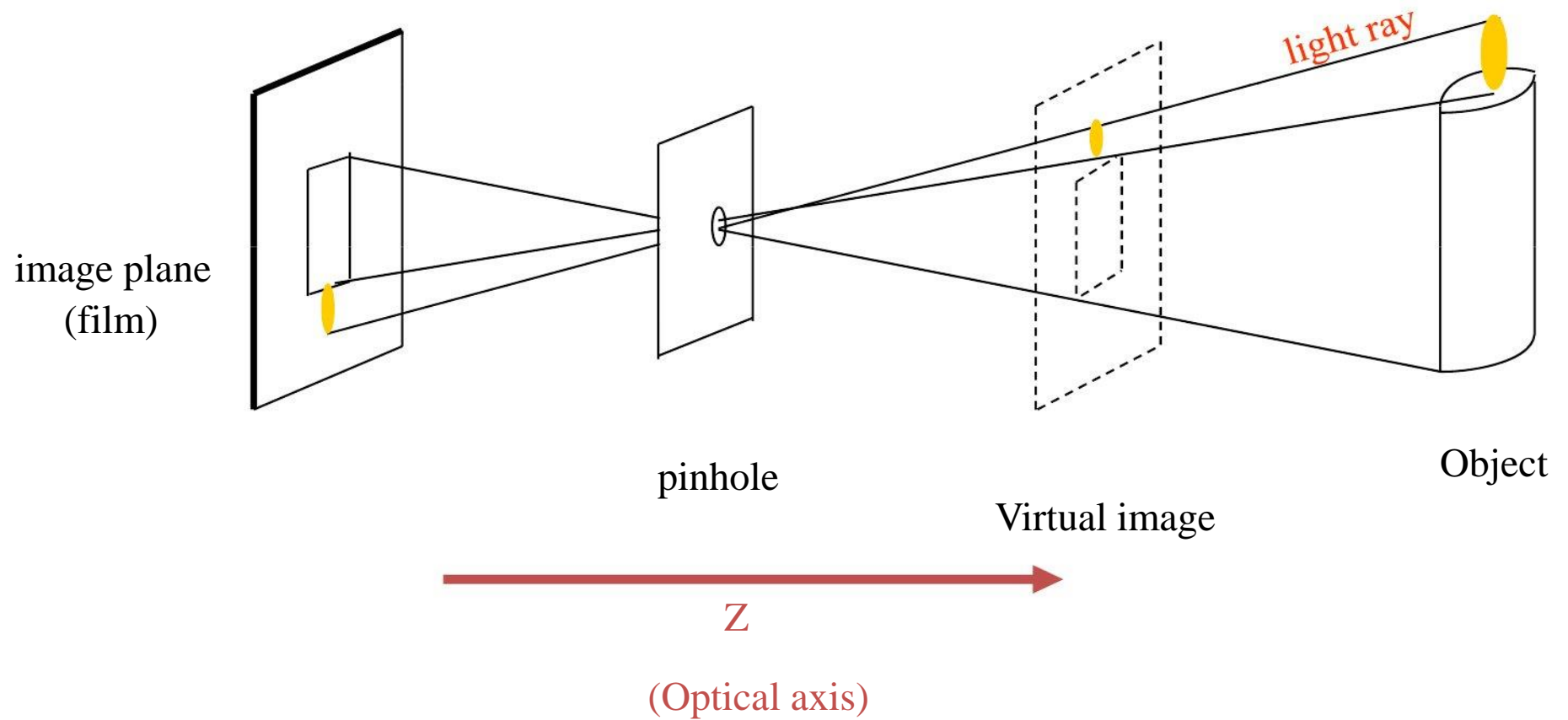
Shadow Mask CRT



Additive and Subtractive Color

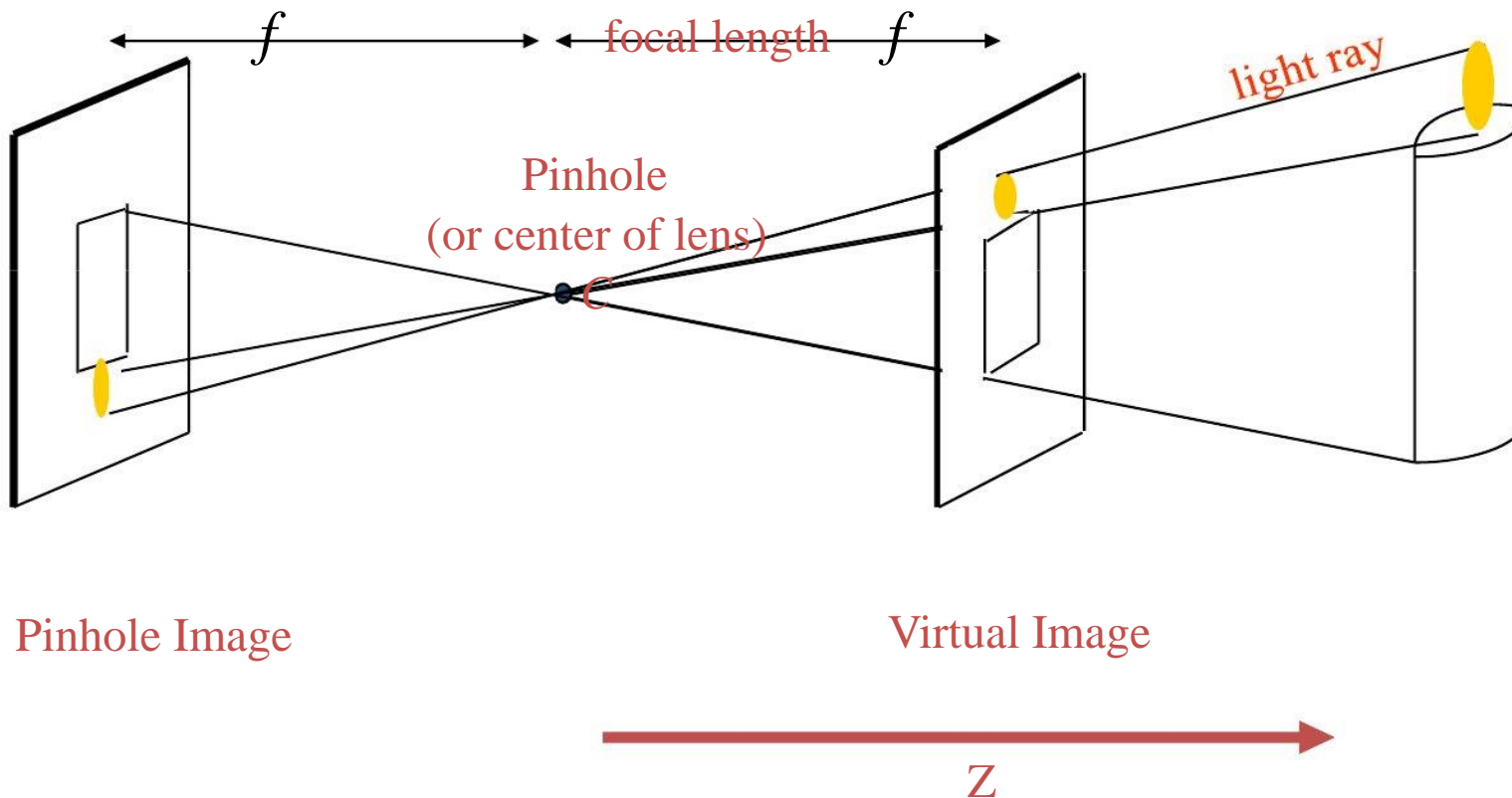
- **Additive color**
 - Form a color by adding amounts of three primaries
 - CRTs, projection systems, positive film
 - Primaries are **Red (R)**, **Green (G)**, **Blue (B)**
- **Subtractive color**
 - Form a color by filtering white light with cyan (C), Magenta (M), and Yellow (Y) filters
 - Light-material interactions
 - Printing
 - Negative film

Image Formation - Pinhole Camera

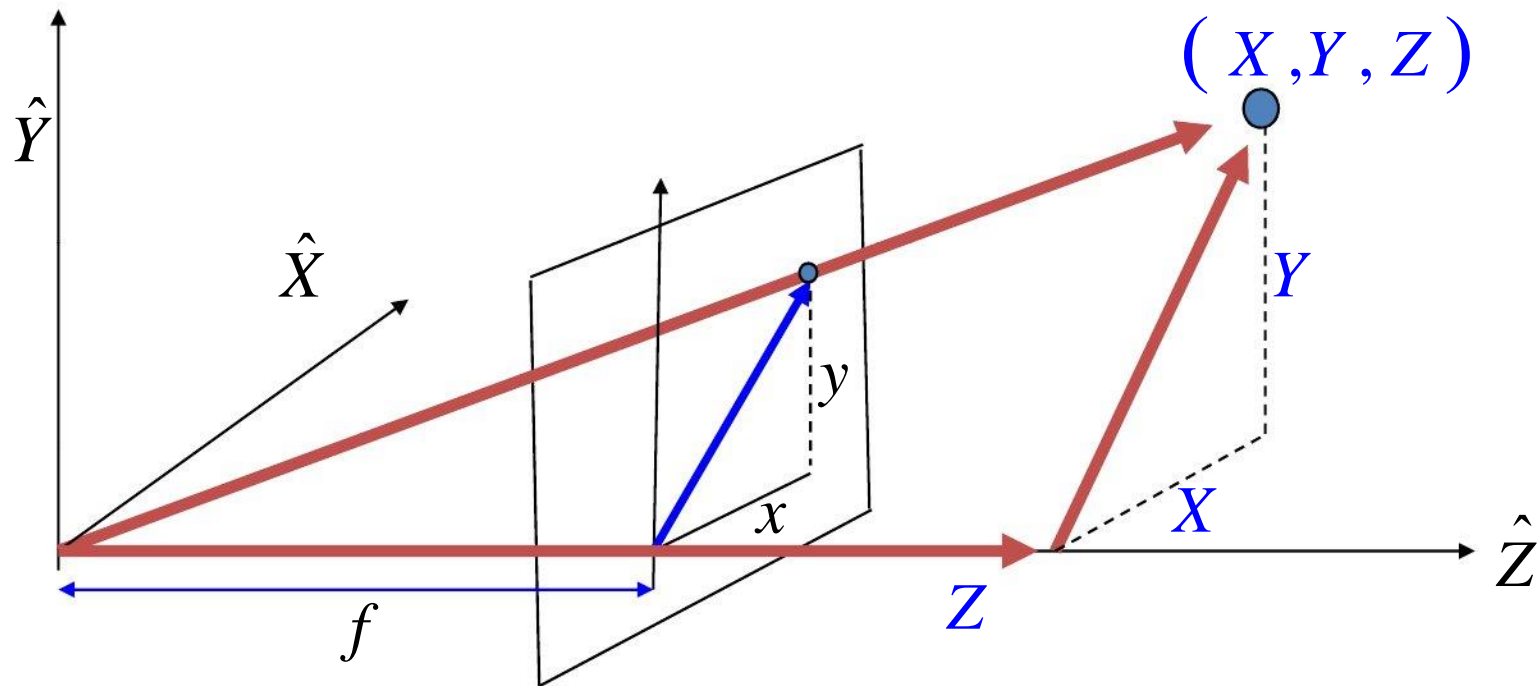


Perspective Projection

- Pinhole camera → Virtual image



Projection Equation



Similar triangles:

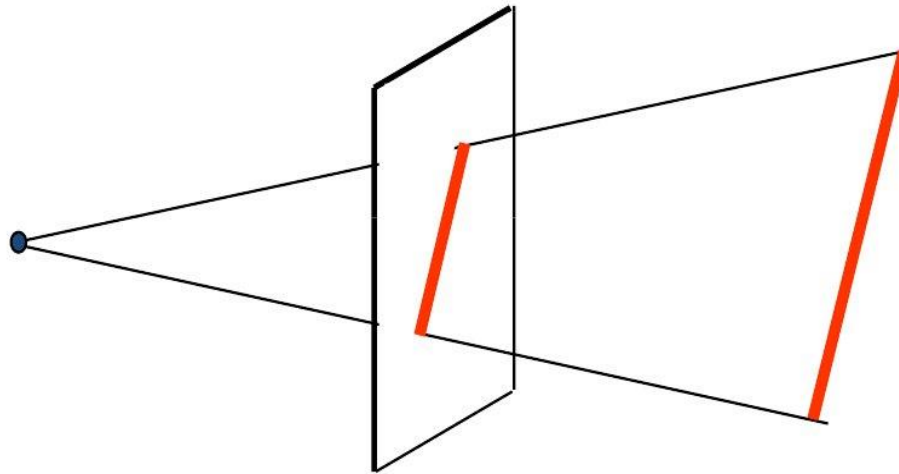
$$\frac{x}{X} = \frac{y}{Y} = \frac{f}{Z}$$



$$(x, y) = \frac{f}{Z}(X, Y)$$

Perspective Projection: Properties

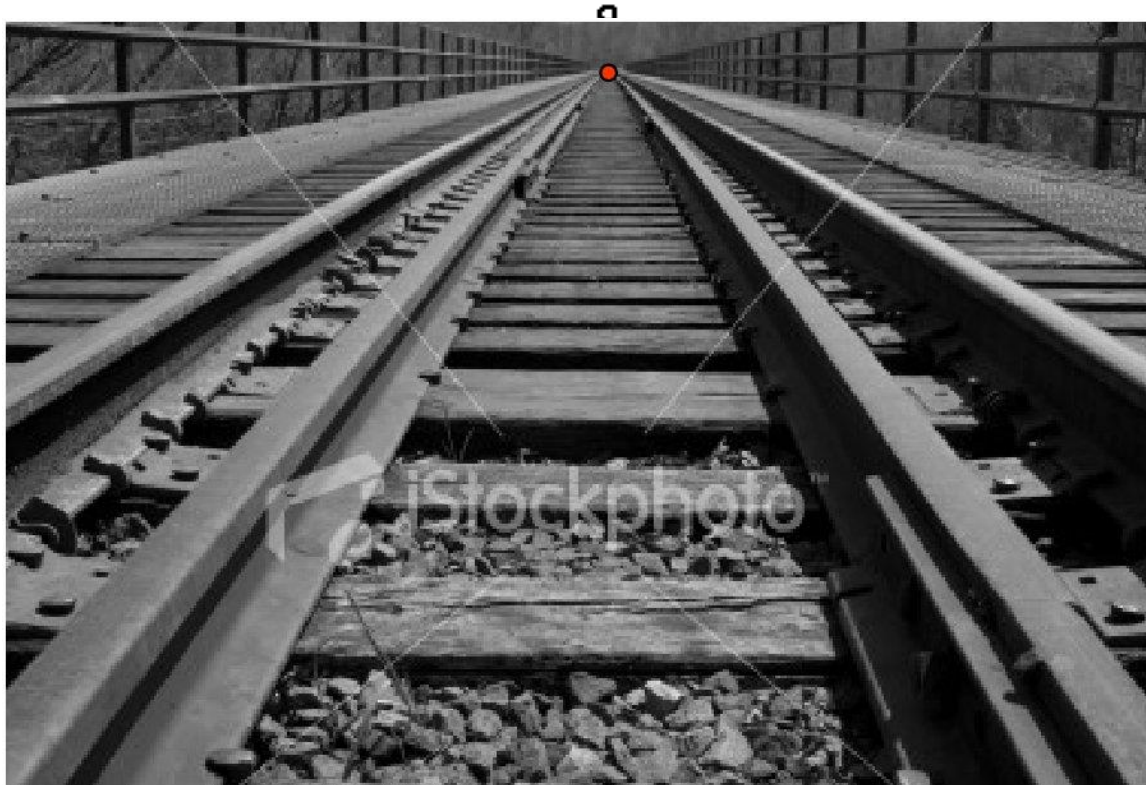
- 3D points \rightarrow image points
- 3D straight lines \rightarrow image straight lines



- 3D Polygons \rightarrow image polygons

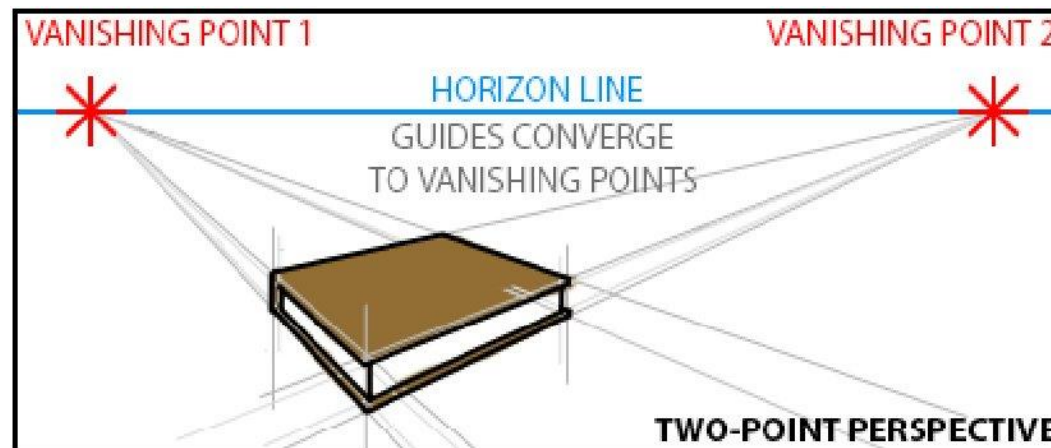
Properties: Vanishing Points

- Image of an infinitely distant 3D point



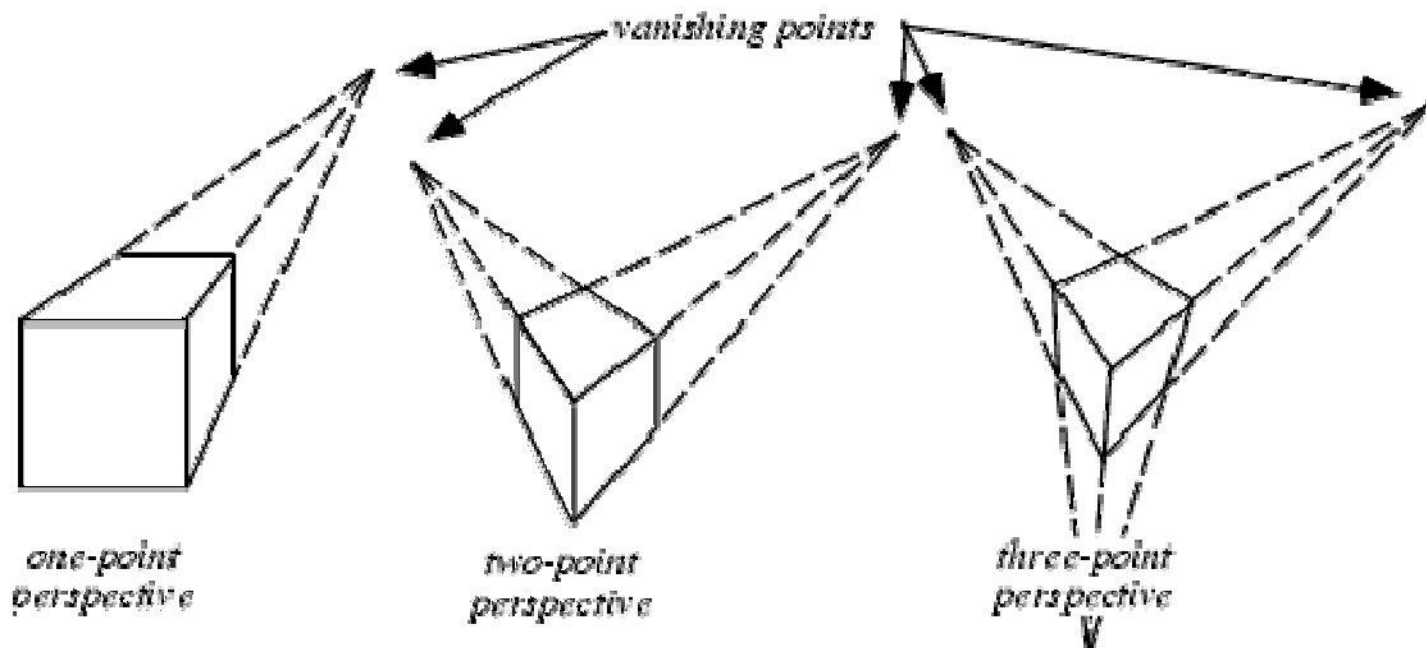
Vanishing Point + Horizon

- Vanishing point
 - Vanishing ray parallel to World Line
→ gives World Line's direction



- Horizon: all vanishing points for World Lines in (or parallel to) plane

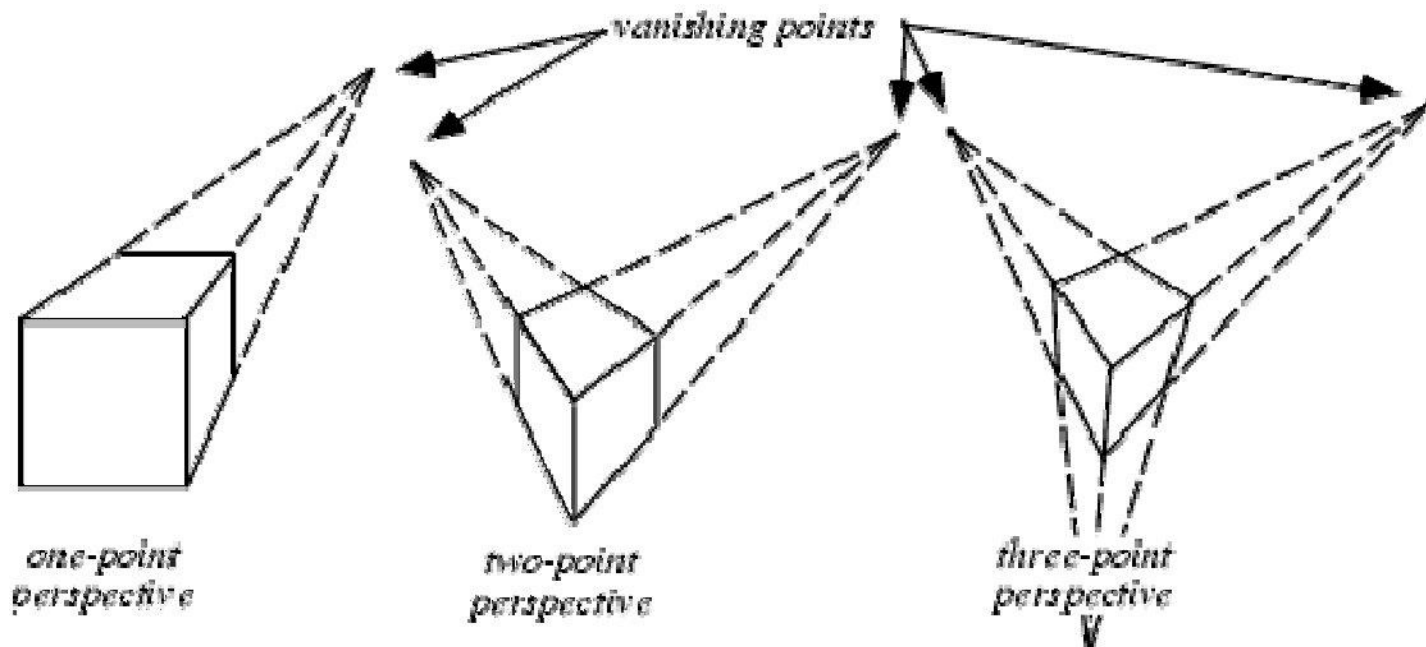
Properties: Vanishing Points



3D vertical and horizontal parallel
to image plane

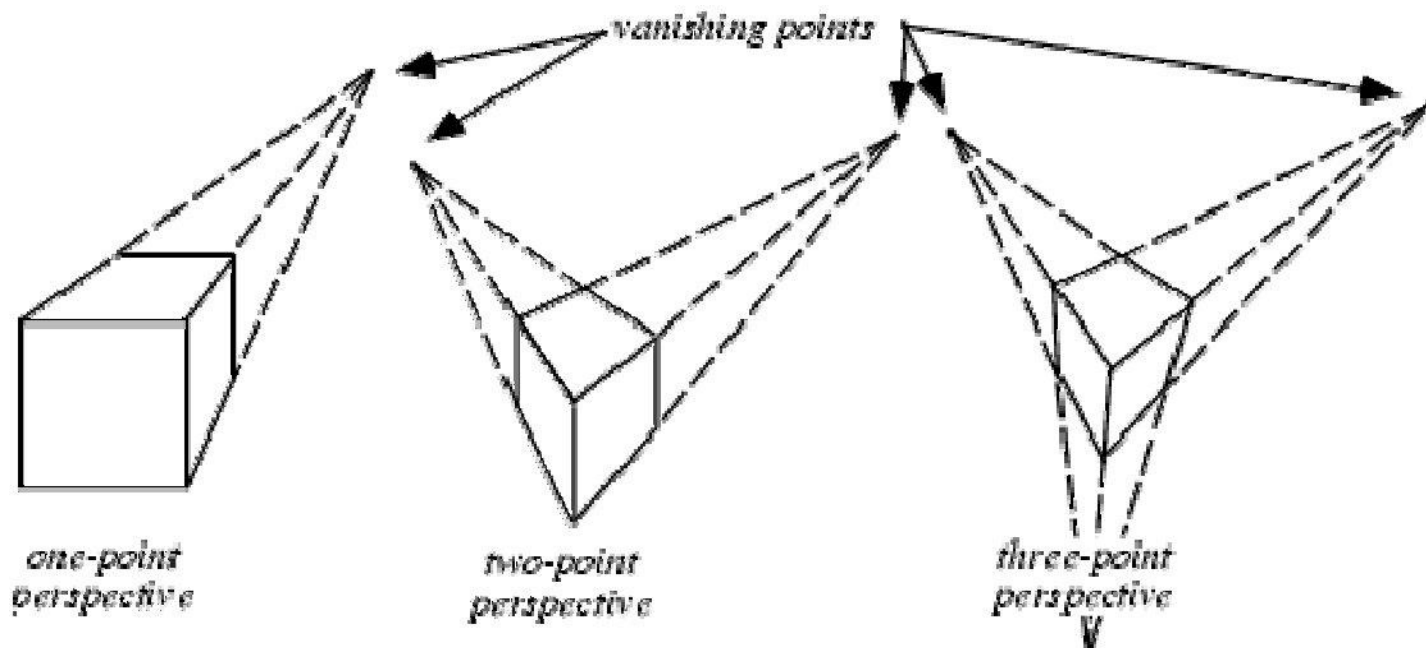
(Two vanishing points at “infinity”
in image plane)

Properties: Vanishing Points



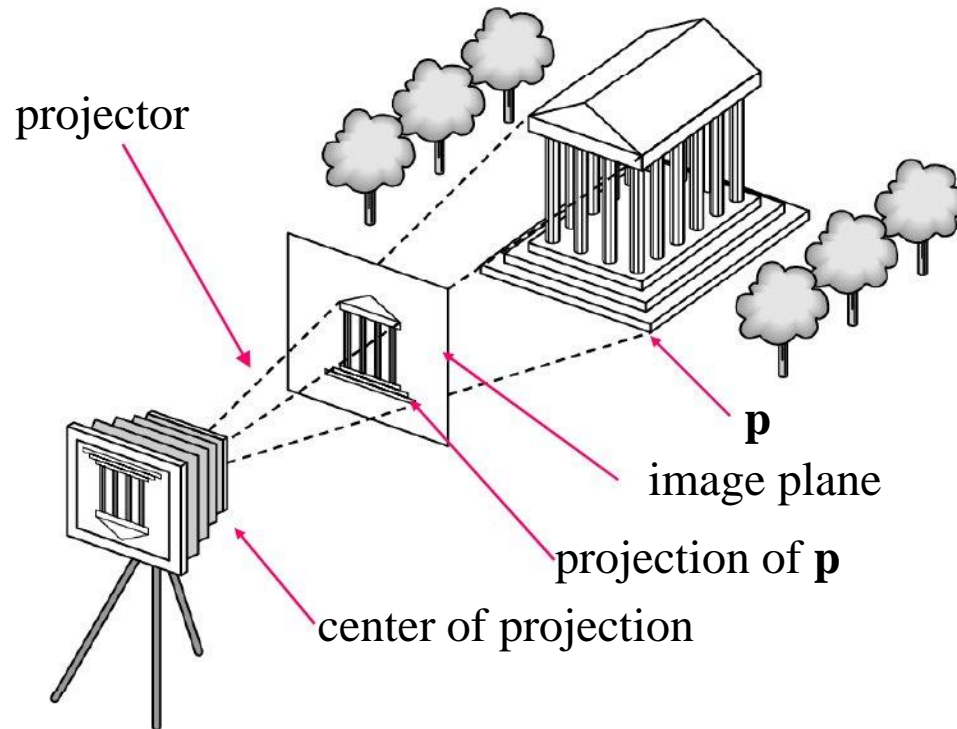
3D vertical parallel to image vertical
(One vanishing point at image “infinity”)

Properties: Vanishing Points



No scene lines parallel to image plane
(three finite vanishing points)

Synthetic Camera Model



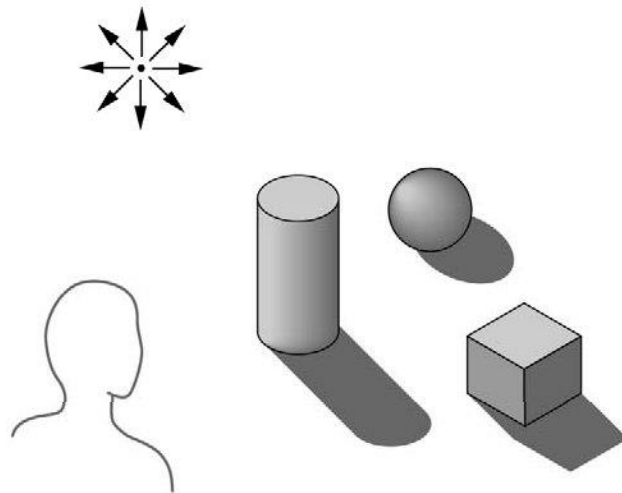
- Object and viewer specifications are independent
- All projectors are straight lines through center of projection
- Image has limited

Advantages

- Separation of objects, viewer, light sources
- **Two-dimensional graphics** is a special case of three-dimensional graphics
- Leads to **simple software API**
 - Specify objects, lights, camera, attributes
 - Let implementation determine image
- Leads to fast hardware implementation

Global vs Local Lighting

- Cannot compute color or shade of each object independently
 - Some objects are **blocked** from light
 - Light can **reflect from object to object**
 - Some objects might be **translucent**



Why not ray tracing?

- Ray tracing seems more physically based, so why don't we use it to design a graphics system?
- Possible and is actually simple for simple objects such as polygons and quadrics with simple point sources
- In principle, can produce **global lighting effects** such as shadows and multiple reflections but ray tracing is **slow** and **not well-suited for interactive applications**

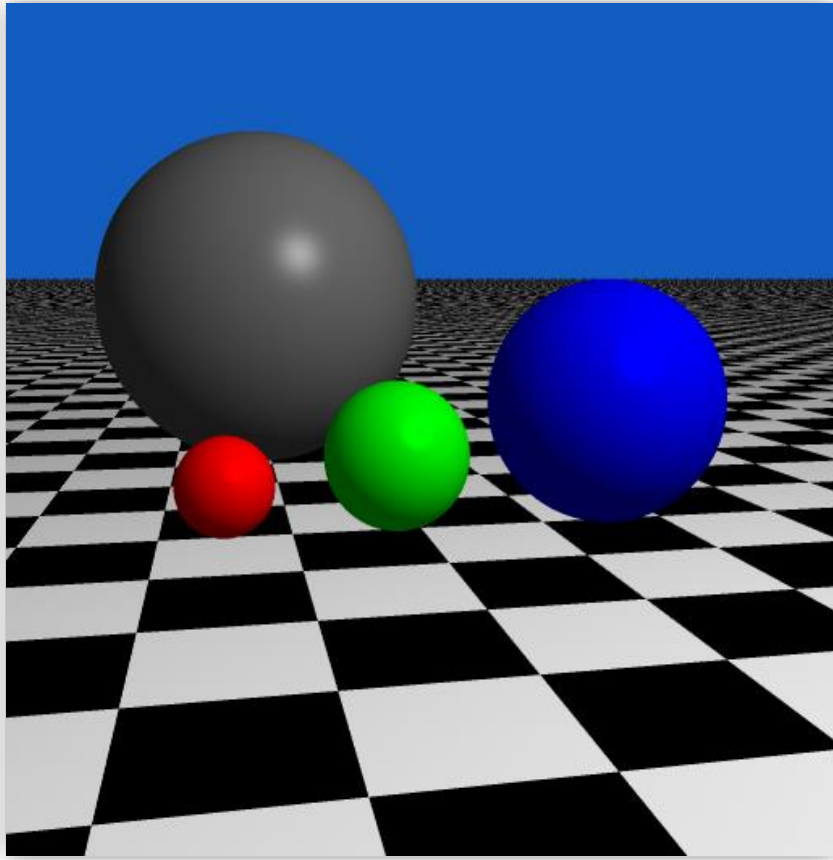
Ray Tracing

- Ray tracing is an image-precision algorithm: **Visibility determined on a per-pixel basis**
 - Trace one (or more) rays per pixel
 - Compute closest object (triangle, sphere, etc.) for each ray
- Produces realistic results
- Computationally expensive

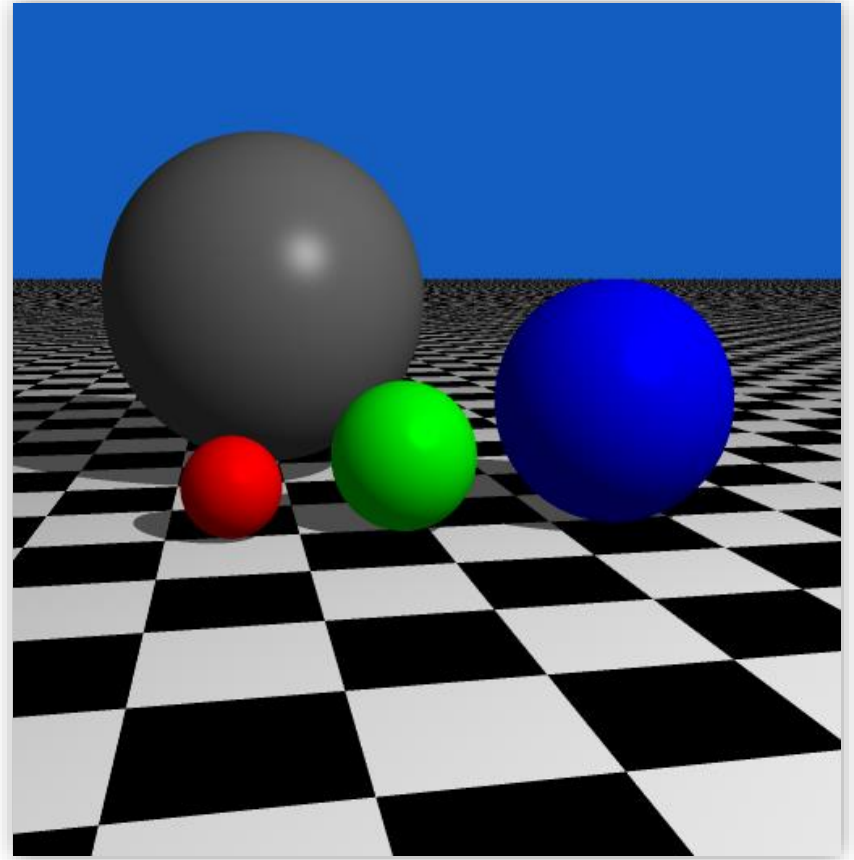


1024×1024, 16 rays/pixel
~ 10 hours on a 99 MHz HP workstation

Shadows

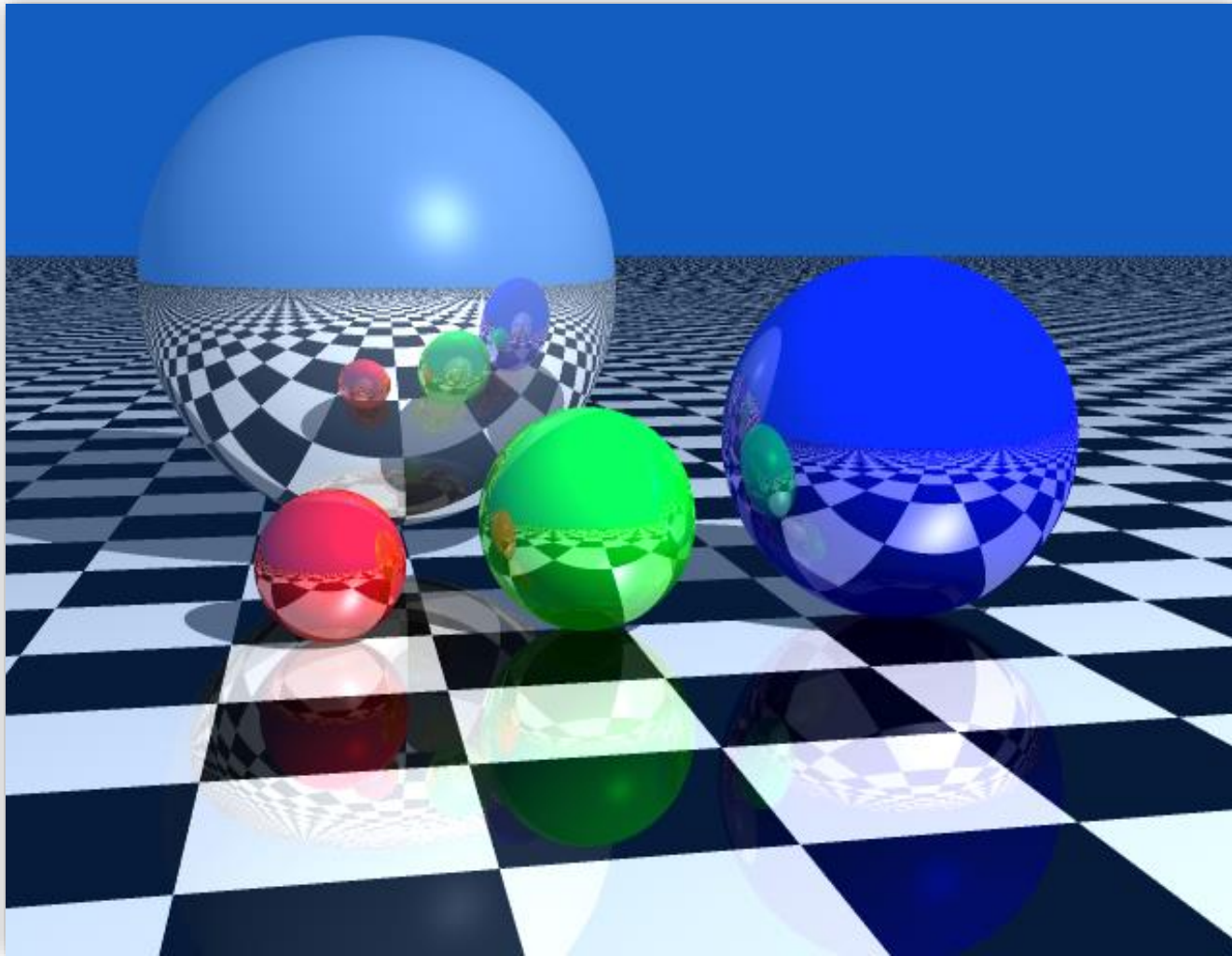


Standard Scene

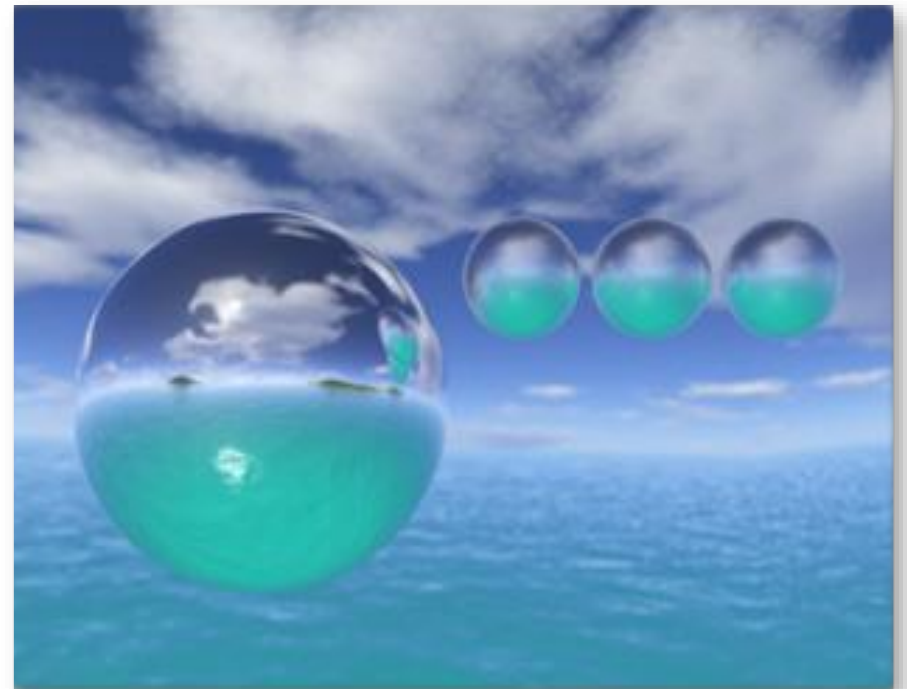
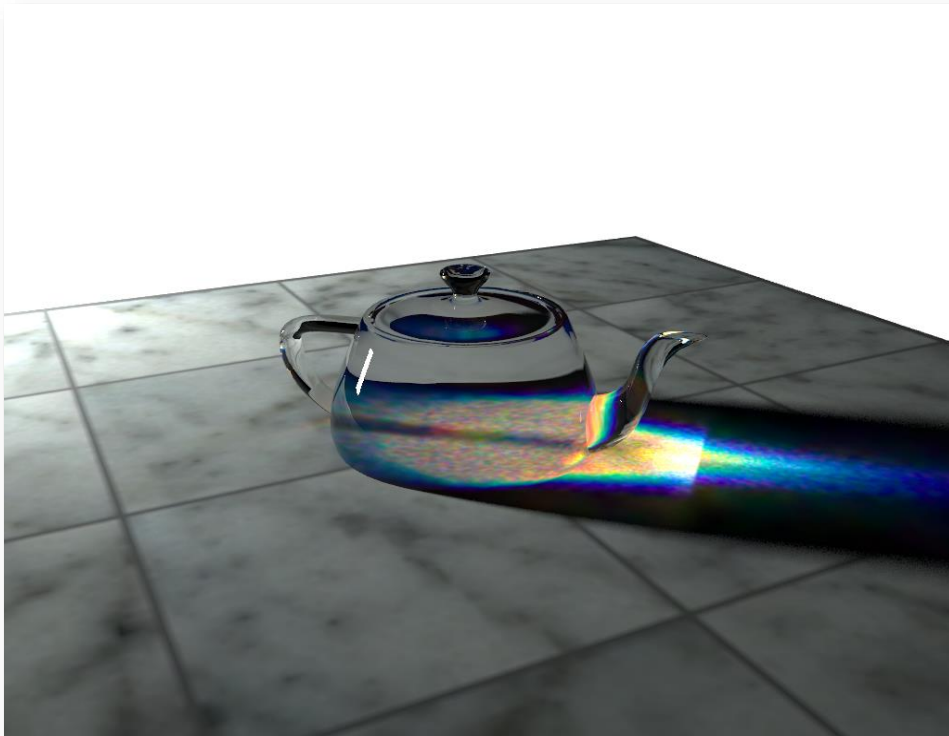


Scene With Shadows

Reflection

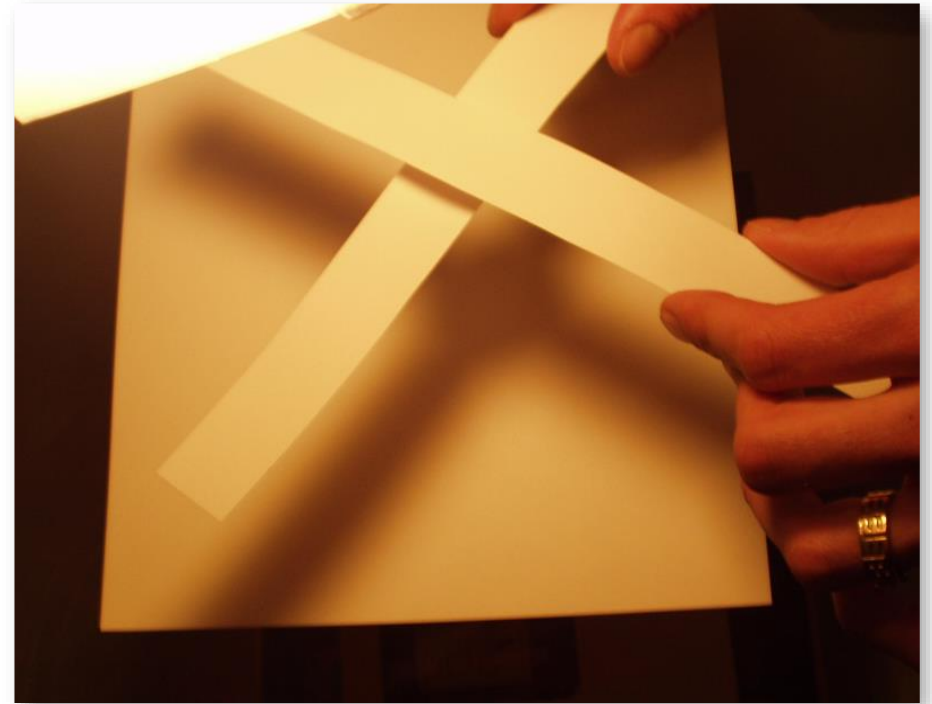


Translucency Examples

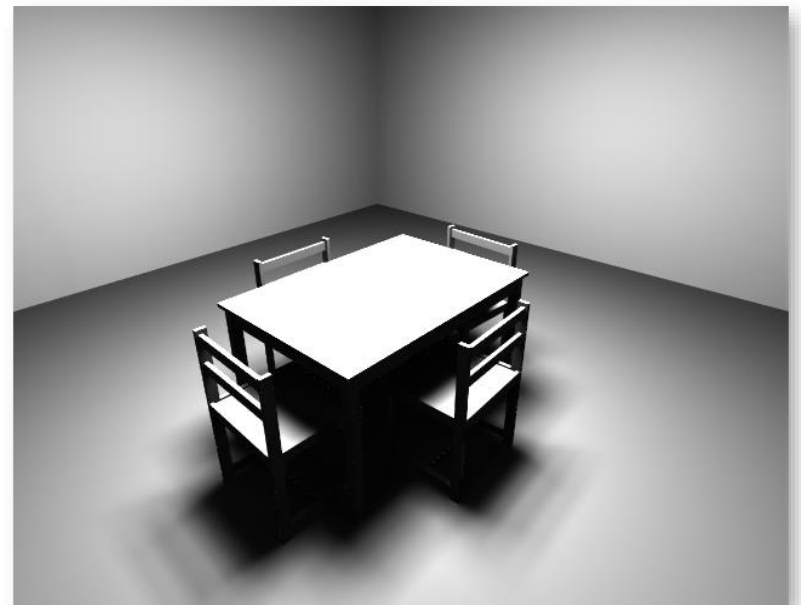
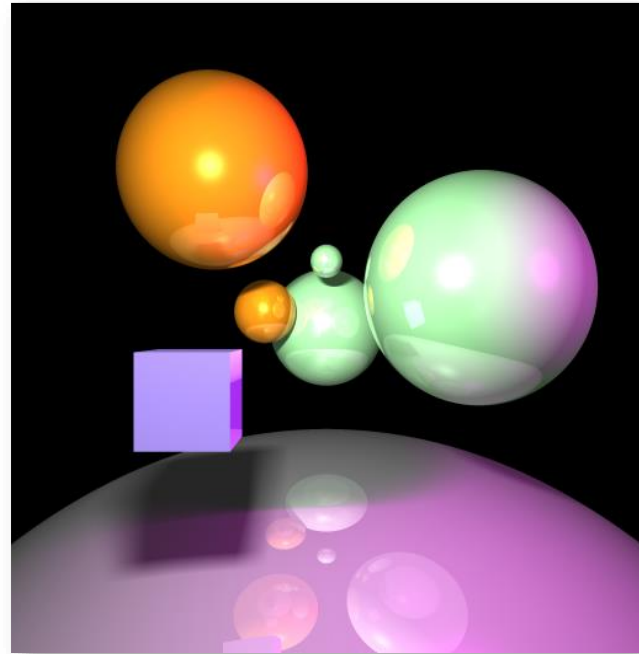
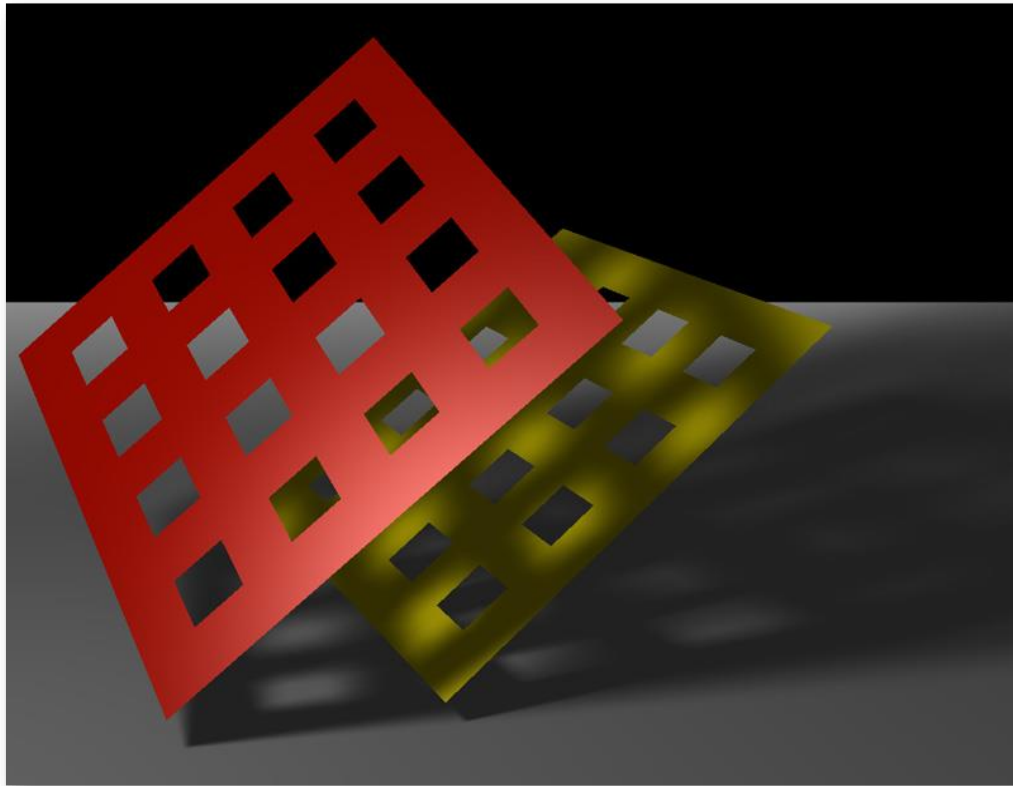


Soft Shadows

- In most graphics applications (and in our ray tracer so far), we've assumed point light sources
 - In the real world, lights have area
 - This leads to soft shadows in the real world, which we can't yet simulate in our ray tracer



Soft Shadow Examples

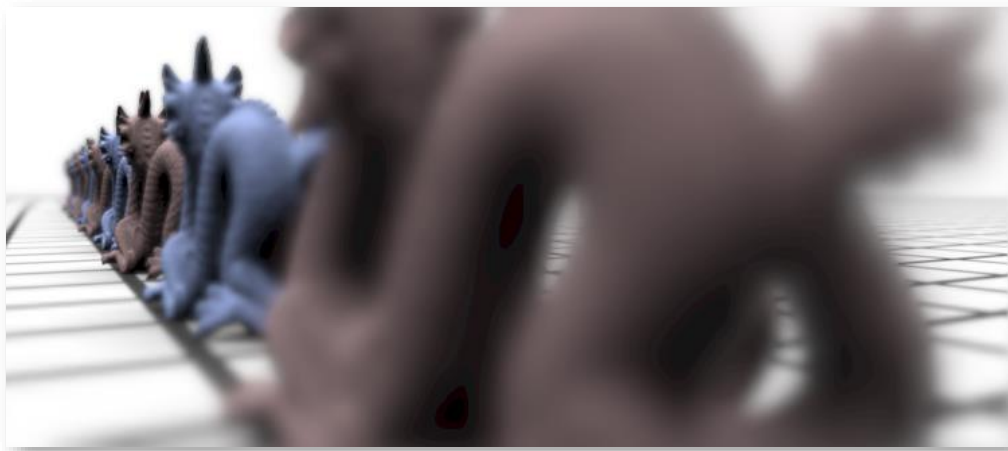
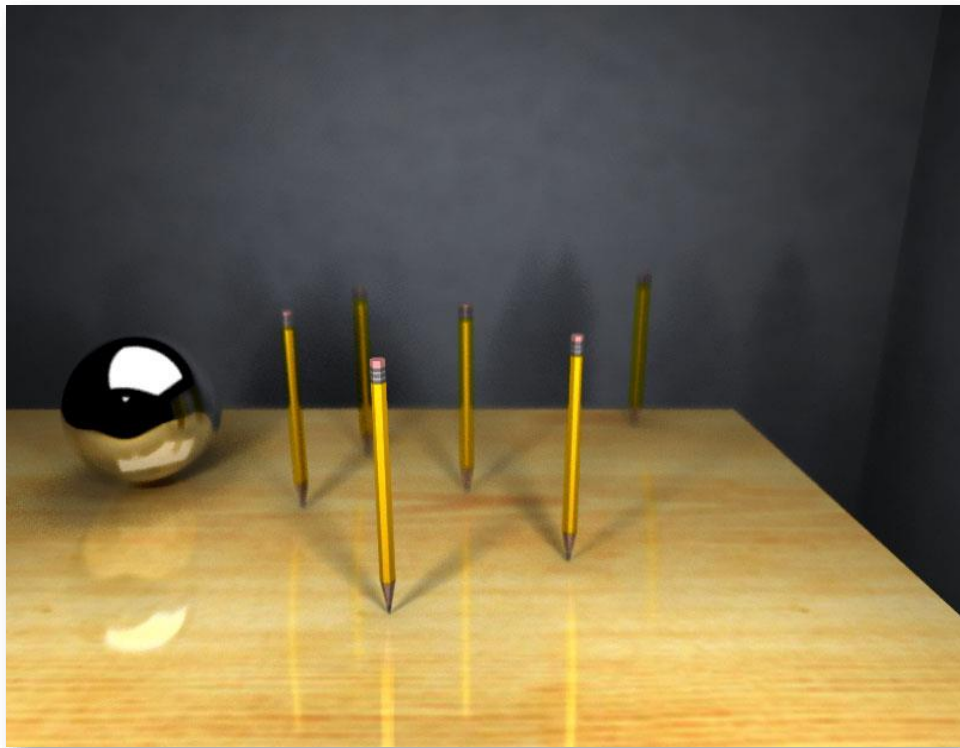


Depth of Field

- Our ray tracer up to this point simulates a pinhole camera
 - Real world cameras have lenses, differing aperture sizes, differing exposure times, etc.
 - We're going to focus (no pun intended) on depth of field



Depth of Field Examples



Models and Architectures

Objectives

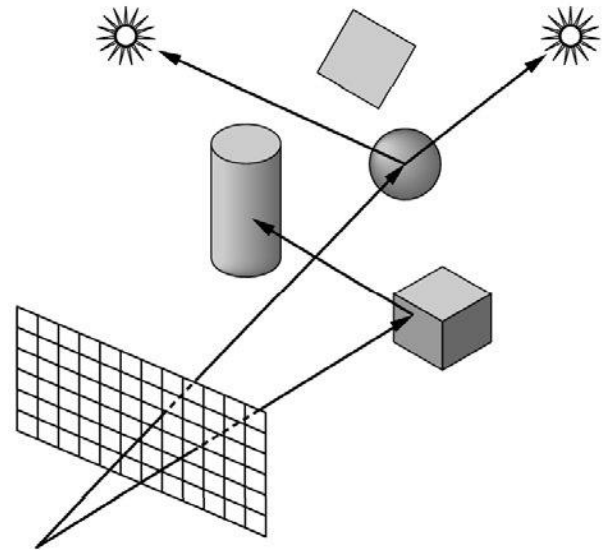
- Learn the basic design of a graphics system
- Introduce **graphics or rendering pipeline** architecture
- Examine software components for an interactive graphics system

Image Formation Revisited

- Can we mimic the synthetic camera model to design graphics hardware software?
- **A**pplication **P**rogrammer **I**nterface (**API**)
 - Need only specify
 - Objects
 - Materials
 - Viewer
 - Lights
- But how is the API implemented?

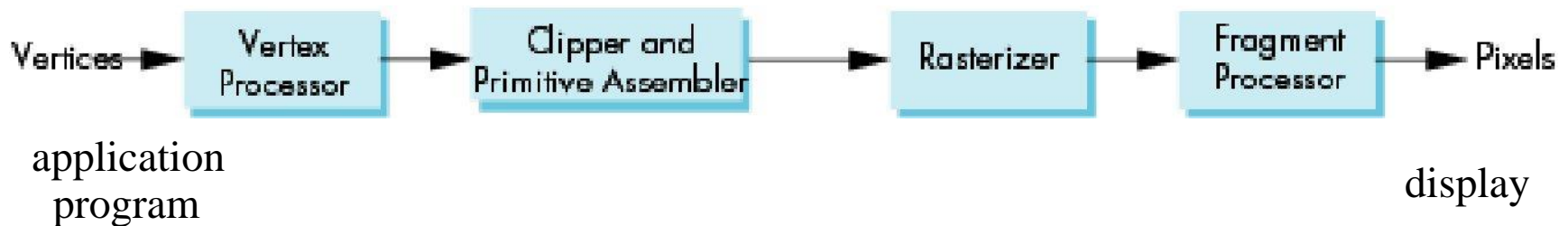
Physical Approaches

- **Ray tracing**: follow rays of light from center of projection until they either are absorbed by objects or go off to infinity
 - Can handle global effects
 - Multiple reflections
 - Translucent objects
 - **Slow**
 - Must have whole data base available at all times
- **Radiosity**: **Energy based** approach
 - **Very slow**



Practical Approach

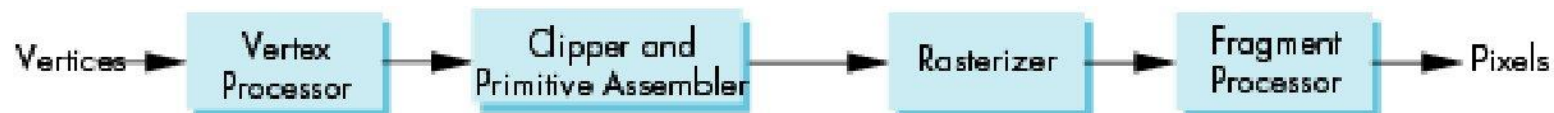
- Process objects one at a time in the order they are generated by the application
 - Can consider only **local lighting**
- Pipeline architecture



- All steps can be implemented in hardware on the graphics card

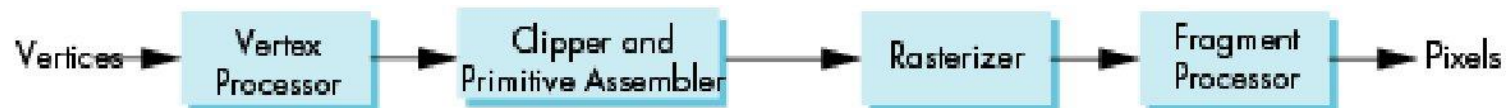
Vertex Processing

- Much of the work in the pipeline is in converting object representations from one coordinate system to another
 - Object coordinates
 - Camera (eye) coordinates
 - Screen coordinates
- Every change of coordinates is equivalent to a **matrix transformation**
- Vertex processor also computes **vertex colors**



Projection

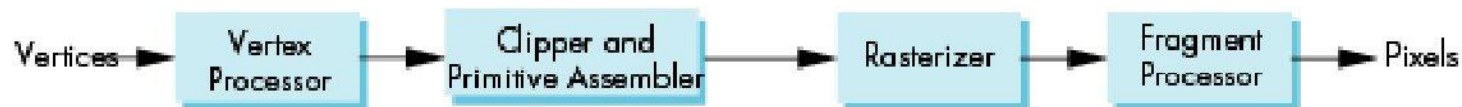
- Projection is the process that combines the 3D viewer with the 3D objects to produce the 2D image
 - **Perspective projections**: all projectors meet at the center of projection
 - **Parallel projection**: projectors are parallel, center of projection is replaced by a direction of projection



Primitive Assembly

Vertices must be collected into geometric objects **before clipping** and **rasterization** can take place

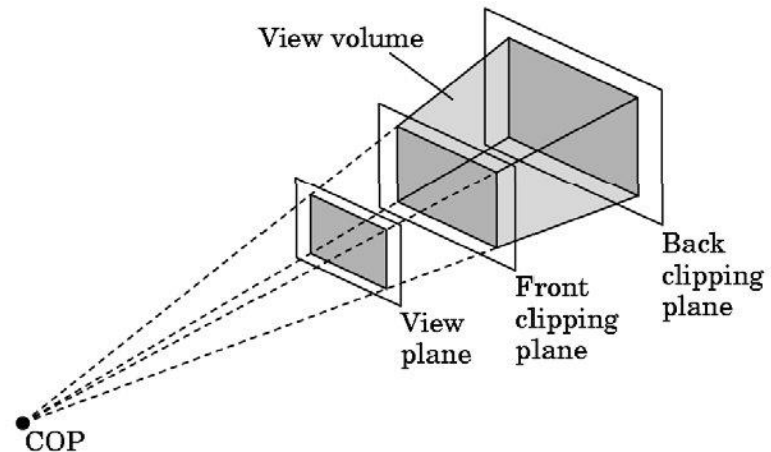
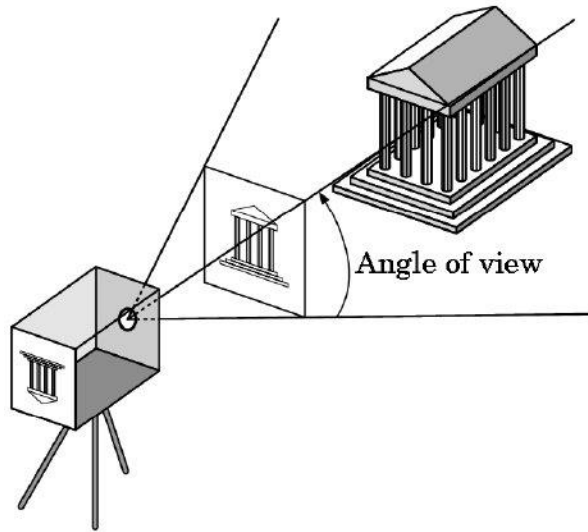
- **Line segments**
- **Polygons**
- **Curves and surfaces**



Clipping

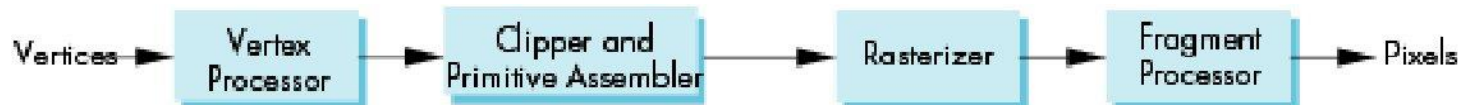
Just as a real camera cannot “see” the whole world, the virtual camera can only see part of the world or object space

- Objects that are not within this volume are said to be clipped out of the scene



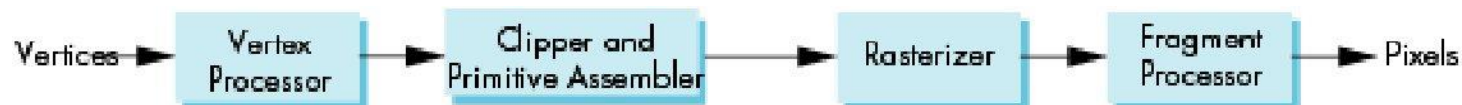
Rasterization

- If an object is not clipped out, the appropriate pixels in the frame buffer must be assigned colors
- Rasterizer produces a set of fragments for each object
- Fragments are “potential pixels”
 - Have a location in frame buffer
 - Color and depth attributes
- Vertex attributes are interpolated over objects by the rasterizer



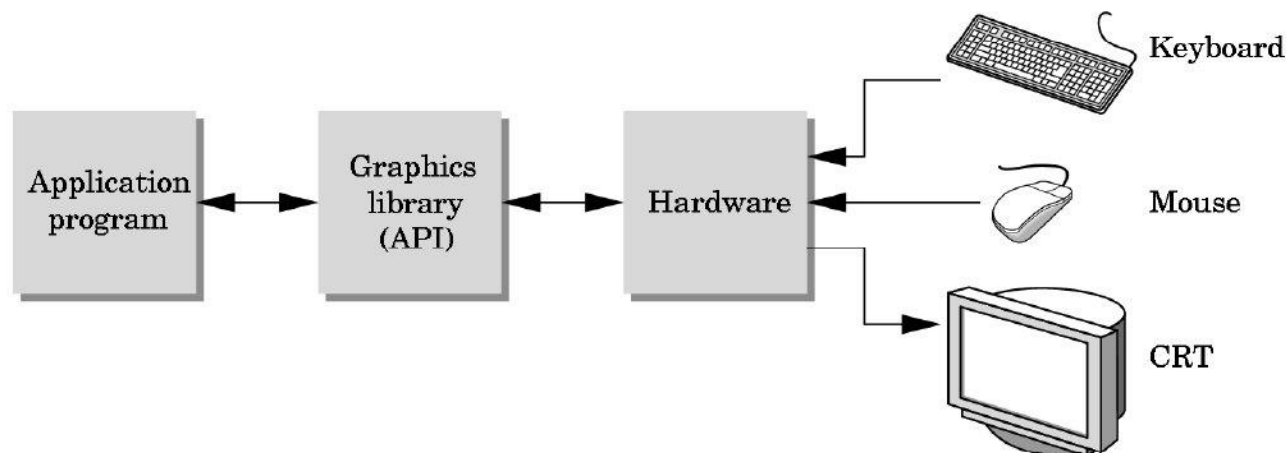
Fragment Processing

- Fragments are processed to determine the **color of the corresponding pixel** in the frame buffer
- **Colors** can be determined by **texture mapping** or **interpolation of vertex colors**
- Fragments may be **blocked** by other fragments closer to the camera
 - **Hidden-surface removal**



The Programmer's Interface

- Programmer sees the graphics system through a software interface: the **A**pplication **P**rogrammer **I**nterface (**API**)



API Contents

- Functions that specify what we need to form an image
 - Objects
 - Viewer
 - Light Source(s)
 - Materials
- Other information
 - Input from devices such as mouse and keyboard
 - Capabilities of system

Object Specification

- Most APIs support **a limited set of primitives** including
 - Points (0D object)
 - Line segments (1D objects)
 - Polygons (2D objects)
 - Some curves and surfaces
 - Quadrics
 - Parametric polynomials
- All are defined through **locations** in space or **vertices**

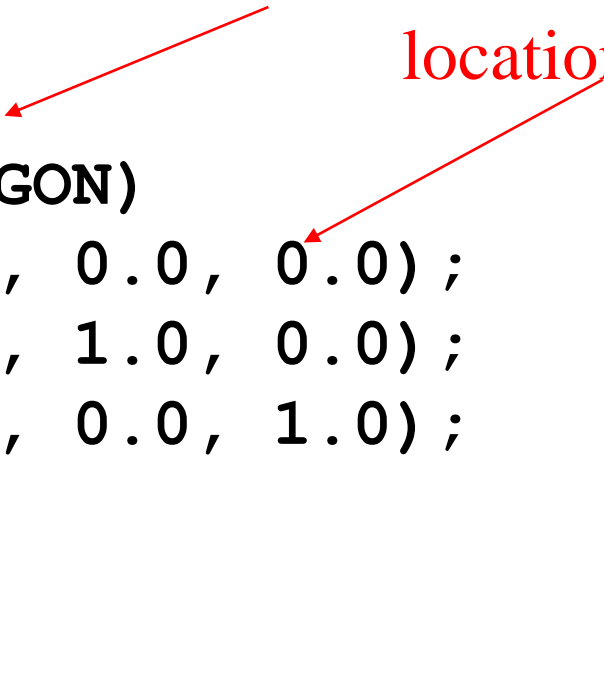
Example (old style)

type of object

location of vertex

```
glBegin(GL_POLYGON)
  glVertex3f(0.0, 0.0, 0.0);
  glVertex3f(0.0, 1.0, 0.0);
  glVertex3f(0.0, 0.0, 1.0);
glEnd( );
```

end of object definition



Example (GPU based)

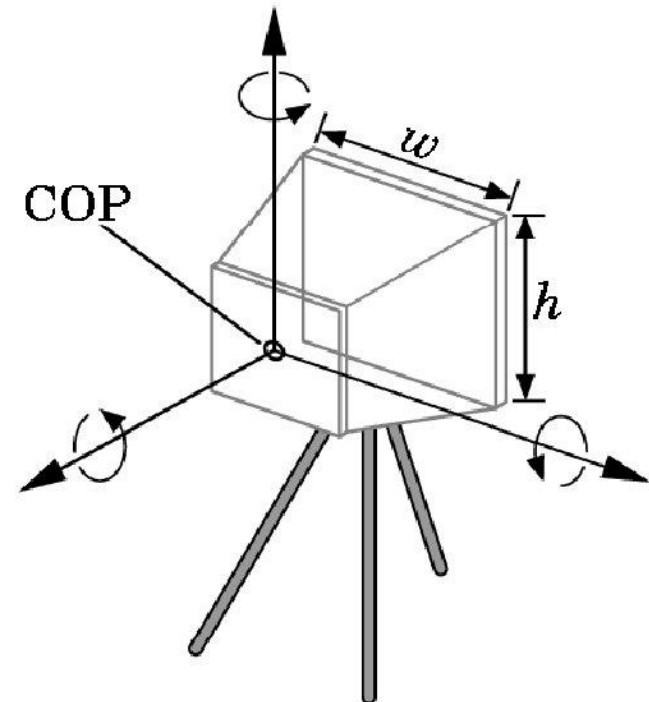
- Put geometric data in an array

```
vec3 points[3];  
points[0] = vec3(0.0, 0.0, 0.0);  
points[1] = vec3(0.0, 1.0, 0.0);  
points[2] = vec3(0.0, 0.0, 1.0);
```

- Send array to GPU
- Tell GPU to render as triangle

Camera Specification

- **Six** degrees of freedom
 - Position of center of lens
 - Orientation
- Lens
- Film size
- Orientation of film plane



Lights and Materials

- **Types of lights**
 - Point sources vs distributed sources
 - Spot lights
 - Near and far sources
 - Color properties
- **Material properties**
 - Absorption: color properties
 - Scattering
 - Diffuse
 - Specular