Interactive Computer Graphics
CS 432

Prof. David E. Breen

Department of Computer Science

Objectives

- Introduction to Interactive Computer Graphics
  - Software
  - Hardware
  - Applications
- Top-down approach
- Shader-Based OpenGL compatible with
  - OpenGL 3.1 (and later)
  - Open GL ES 2.0
  - webGL

Credits

- Course structure based on Ed Angel and Dave Shreiner, *Interactive Computer Graphics, A Top-down Approach with OpenGL (Sixth Edition)*, Addison-Wesley, 2012
- Slides based on lectures for CS/EECE 412 Computer Graphics at the University of New Mexico by Prof. Edward Angel

Prerequisites

- Good programming skills in C++
- Linux-based software development
- Basic Data Structures
  - Linked lists
  - Arrays
- Geometry
- Linear Algebra
  - Vectors & matrices

Requirements

- Weekly Programming Projects
- Optional Term Project
  - Defined by each student
- Grad Students Only
  - Summarize 2 research papers
- Go to class web site

Resources

- Can run OpenGL on any system
  - Windows: check graphics card properties for level of OpenGL supported
  - Linux
  - Mac: need extensions for 3.1 equivalence
- Get GLUT from web if needed
  - Provided on Macs
  - freeglut available on web
- Get GLEW from web

References

- www.opengl.org
  - Standards documents
  - Sample code
- The OpenGL Programmer's Guide (the Redbook) 8th Edition
  - The definitive reference
  - OpenGL 4.1
- OpenGL Shading Language, 3rd Edition
- All Addison-Wesley Professional

Image Formation

Objectives

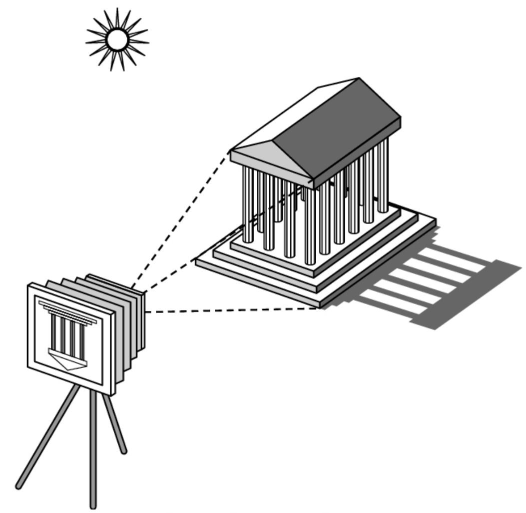
- Fundamental imaging notions
- Physical basis for image formation
    - Light
    - Color
    - Perception
- Synthetic camera model
- Other models

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

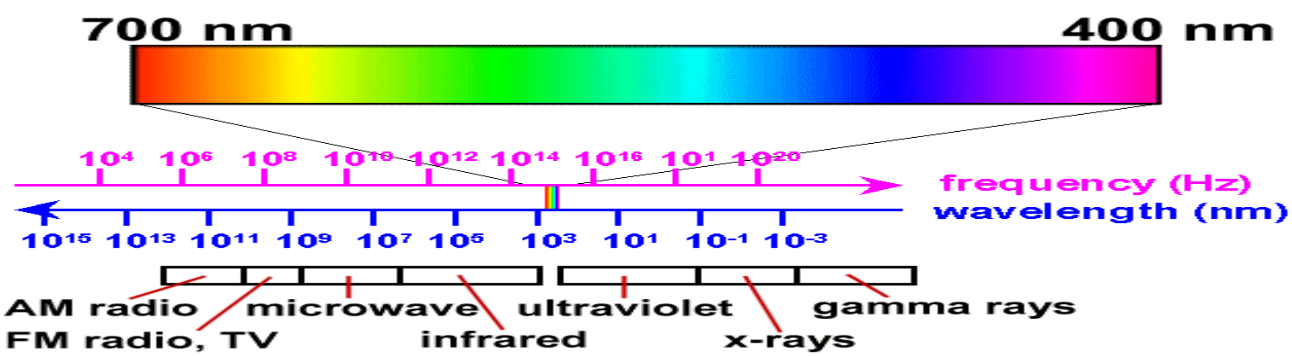
- Objects
- Viewer
- Light source(s)



- 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

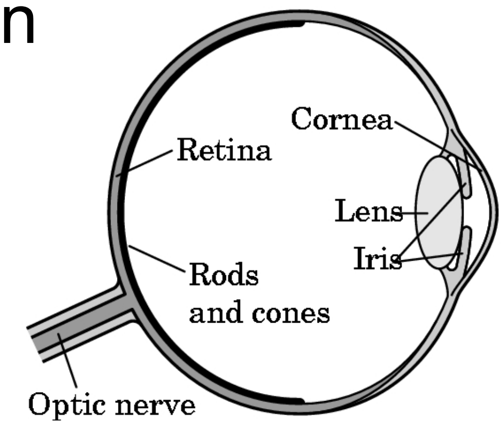


Luminance and Color Images

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

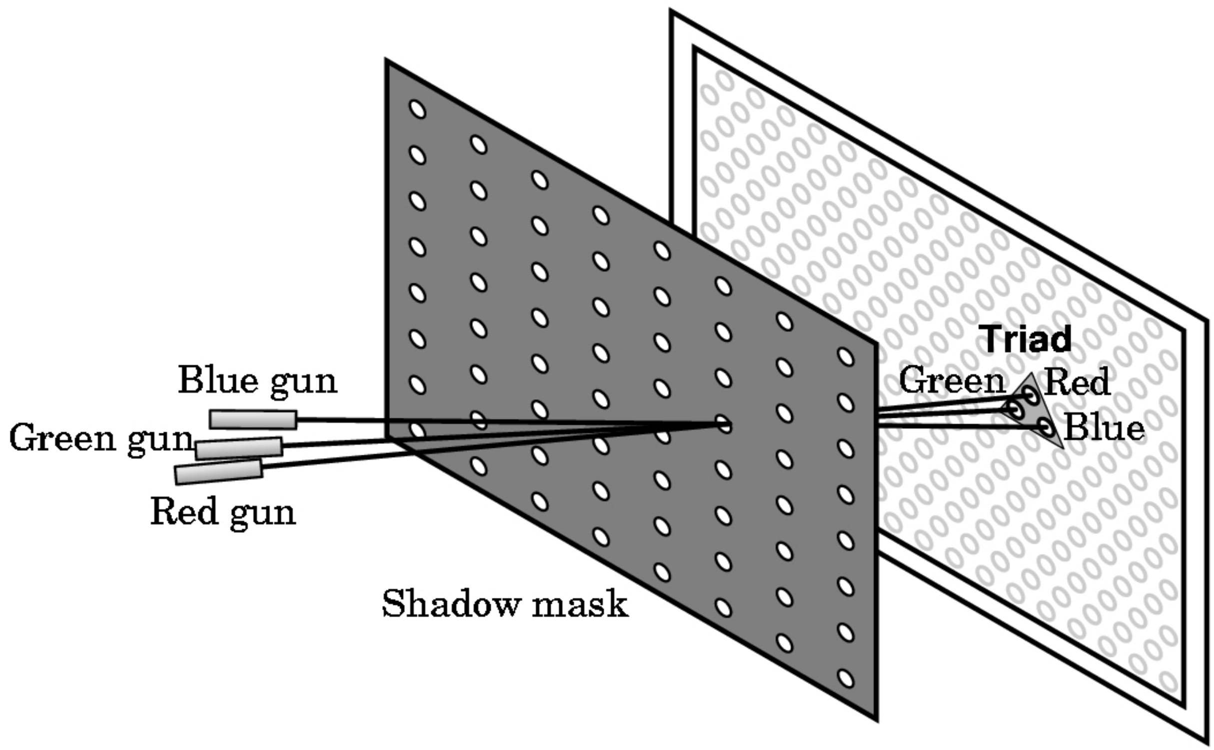
Three-Color Theory

- 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



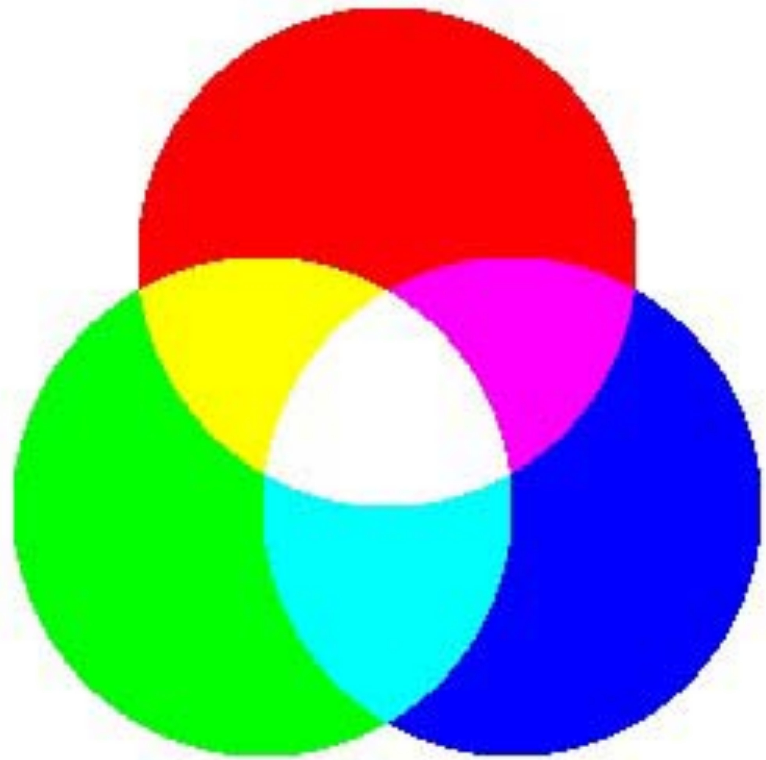
- Need only match these three values
  - Need only three *primary* colors

Shadow Mask CRT



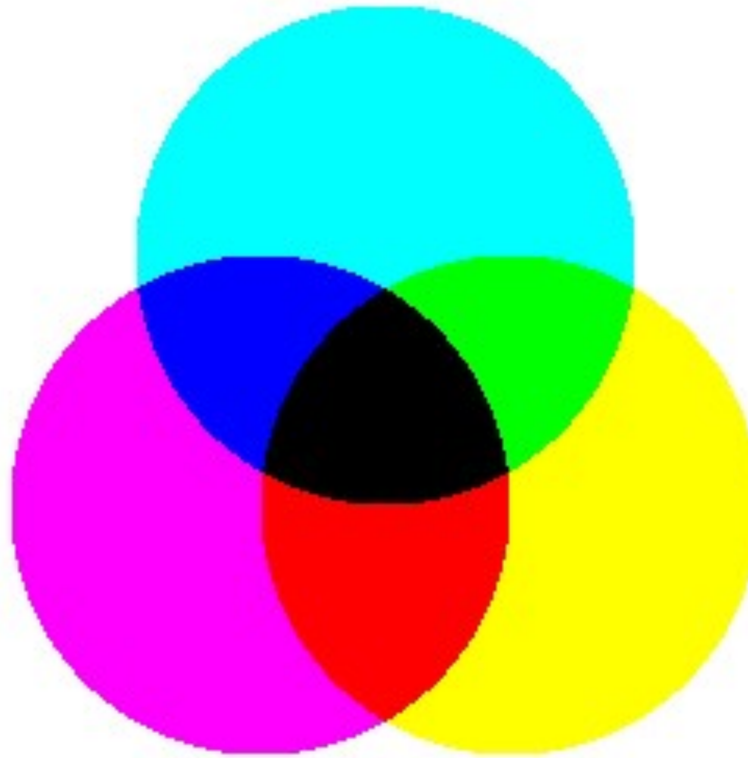
Additive and Subtractive Color

- Additive color
  - Form a color by adding amounts of three primaries
    - CRTs, projection systems, positive film
  - Primaries are Red (R), Green (G), Blue (B)



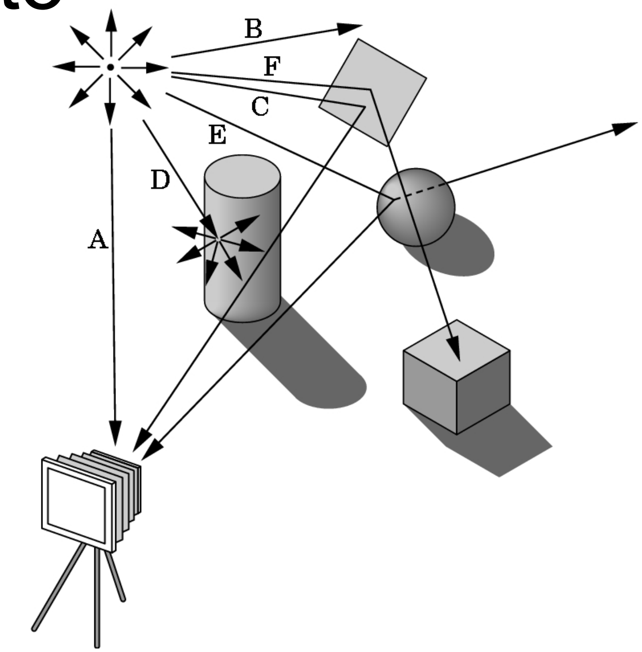
Additive and Subtractive Color

- Subtractive color
  - Form a color by filtering white light with cyan (C), Magenta (M), and Yellow (Y) filters
    - Light-material interactions
    - Printing
    - Negative film



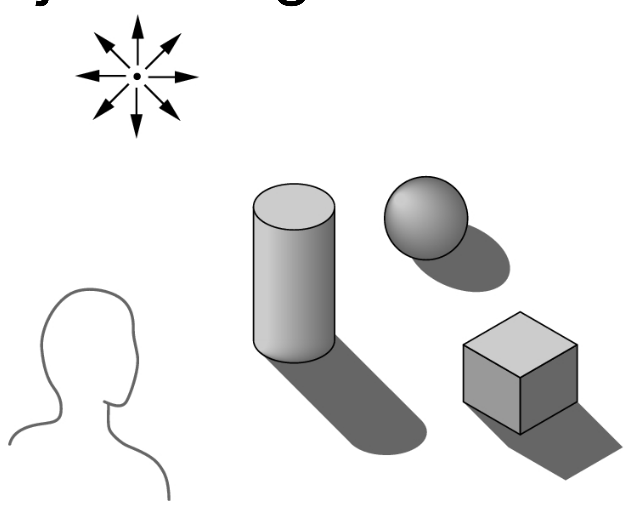
Ray Tracing and Geometric Optics

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.



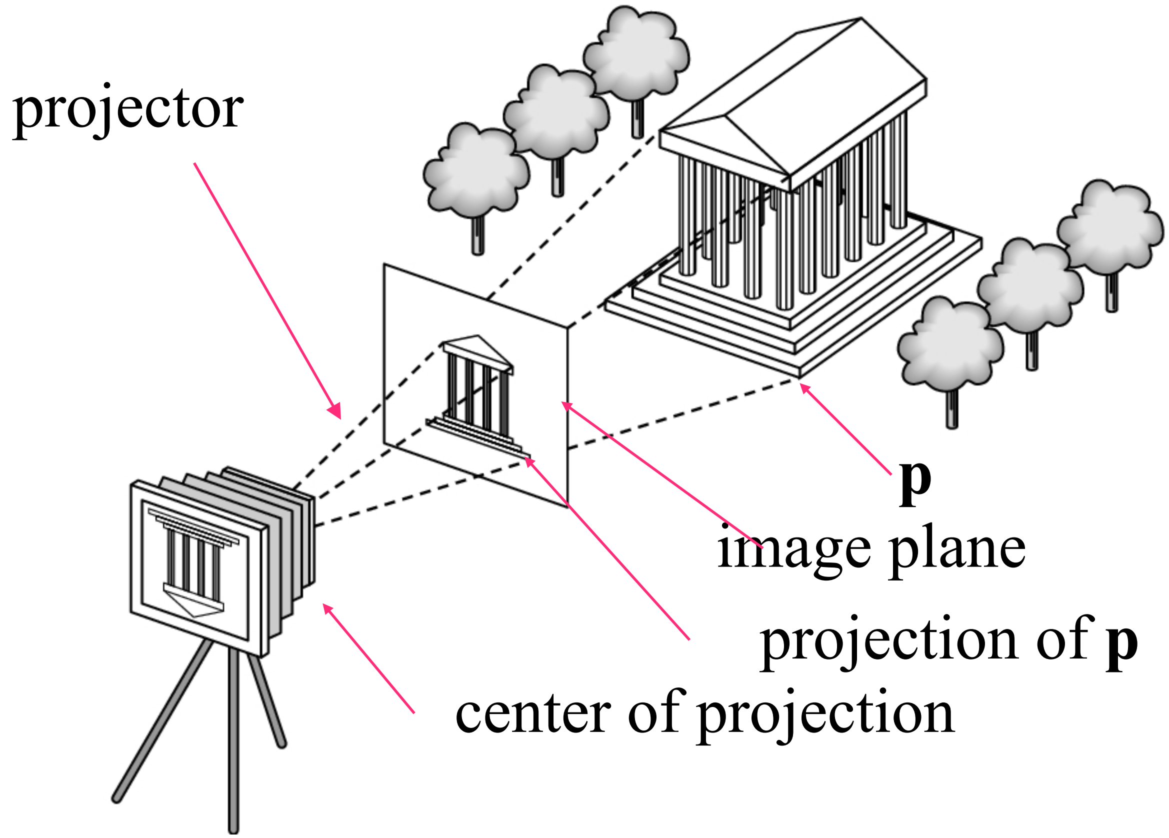
Global vs Local Lighting

- 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

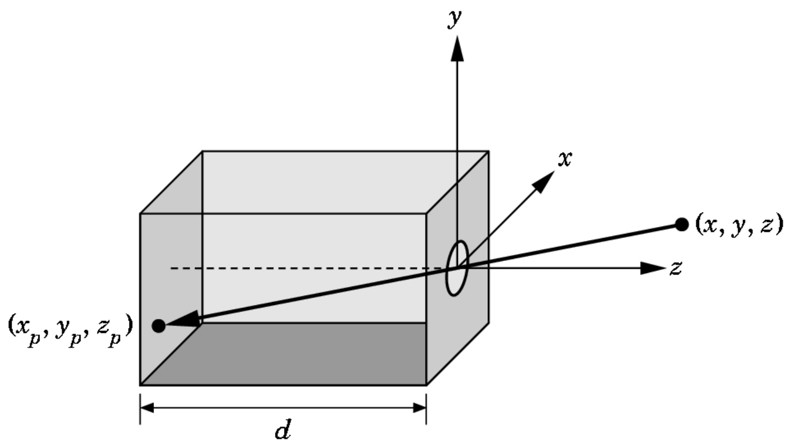


Synthetic Camera Model

- Local lighting
- Projects geometry onto image plane.
- Use local info to shade point



Pinhole Camera



Use trigonometry to find projection of point at (x,y,z)

$x_p = -x/(z/d)$ $y_p = -y/(z/d)$ $z_p = d$

These are equations of simple perspective

Advantages – Local Lighting

- 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

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 with GPUs is close to real time

Models and Architectures

Objectives

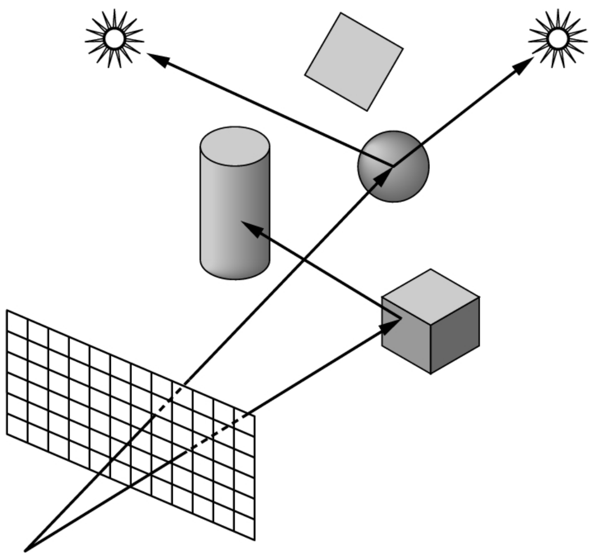
- Learn the basic design of a graphics system
- Introduce pipeline architecture
- Examine software components for an interactive graphics system

Image Formation Revisited

- Can we mimic the synthetic camera model to design graphics hardware & software?
- Application Programmer Interface (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

application program

display

- 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 view 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

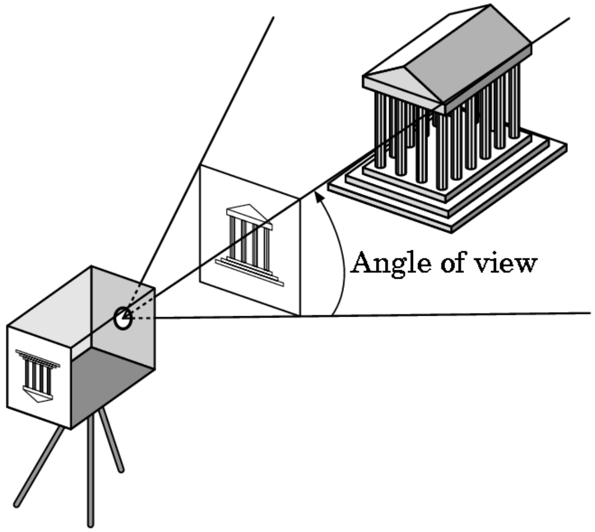
Vertices must be collected into geometric objects before clipping and rasterization can take place

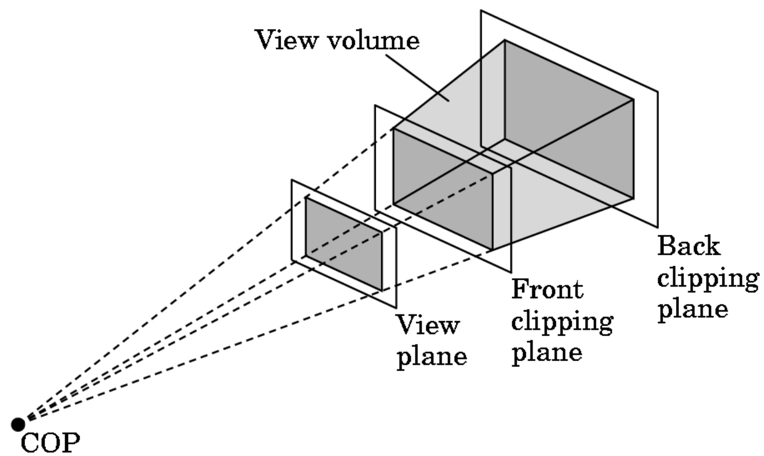
- Line segments
- Polygons
- Curves and surfaces

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

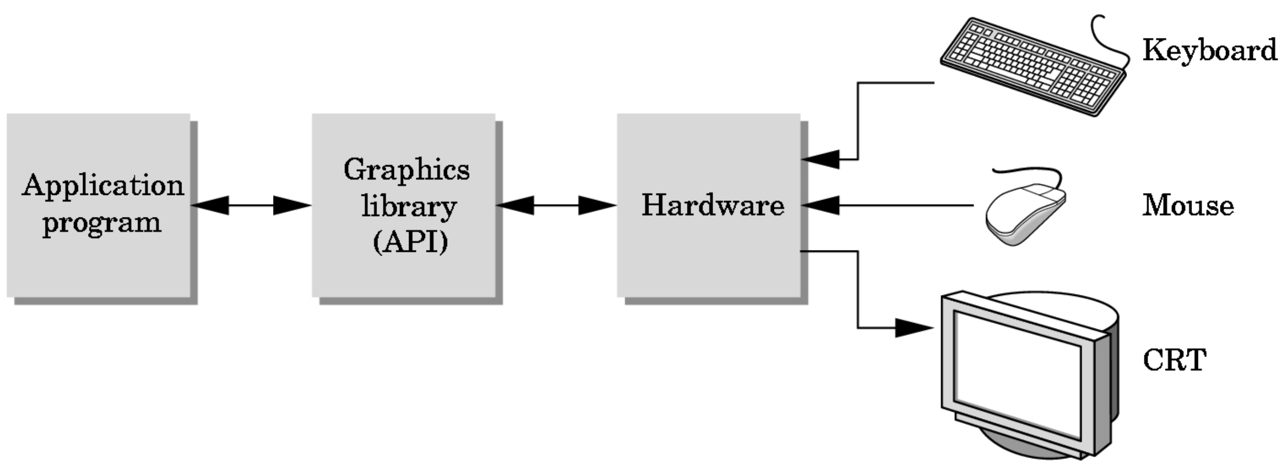
- If an object is not clipped out, the appropriate pixels in the frame buffer must be assigned colors
- Rasterizer produces a set of fragments for each object
- Fragments are “potential pixels”
  - Have a location in frame bufffer
  - Color and depth attributes
- Vertex attributes are interpolated over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

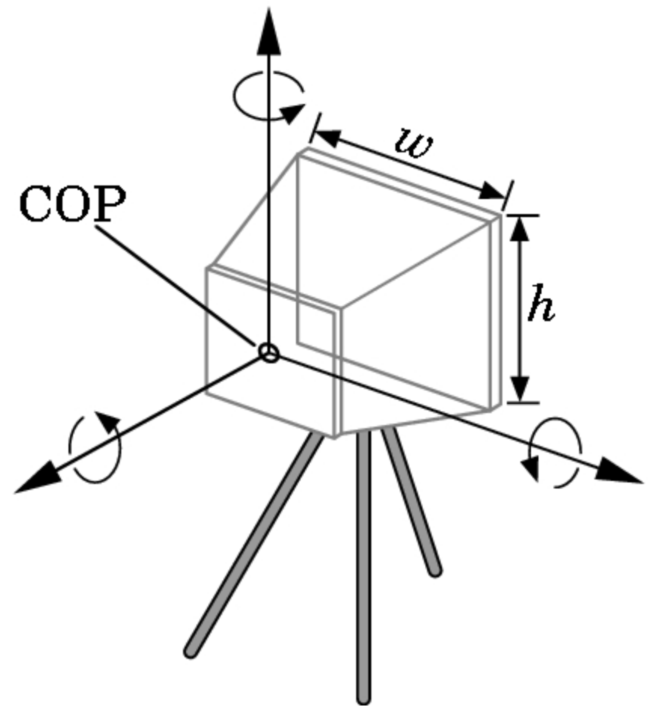
- Put geometric data in an array

```
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