# 1. Graphics Systems and Models

# Overview

- These lectures are for a senior/graduate elective for computer science and engineering majors(and others with good programming skills)
- The course is based on a modern approach using programmable shaders in the new textbook: Ed Angel and Dave Shreiner, Interactive Computer Graphics, A Top-down Approach with WebGL(Seventh Edition), Addison-Wesley
- These lectures cover Chapters 1-7 in detail and survey Chapters 8-12

# Outline

- Introduction
- Example Code in JS
- What is Computer Graphics?
- Image Formation
- Models and Architectures

- Reading: Chapter 1
- Exercises: Run some examples on your browser

# Introduction

# Introduction

- Course Overview
- Required Background
- References

# Objectives

- Broad introduction to Computer Graphics
  - Software
  - Hardware
  - Applications
- Top-down approach
- Shader-Based WebGL
  - Integrates with HTML5
  - Code runs in latest browsers

# Prerequisites

- Good programming skills in C (or C++)
- Basic Data Structures
  - Linked lists
  - Arrays
- Geometry
- Simple Linear Algebra

# Why is this course different?

- Shader-based
  - Most computer graphics use OpenGL but still use fixed-function pipeline
  - does not require shaders
  - Does not make use of the full capabilities of the graphics processing unit (GPU)
- Web
  - With HTML5, WebGL runs in the latest browsers
  - makes use of local hardware
  - no system dependencies

# References

- Interactive Computer Graphics (7th Edition)
- The OpenGL Programmer's Guide (the Redbook) 8th Edition
- OpenGL ES 2.0 Programming Guide
- WebGL Programming Guide
- WebGL Beginner's Guide
- WebGL: Up and Running
- JavaScript: The Definitive Guide

# Web Resources

- www.cs.unm.edu/~angel/
- www.cs.unm.edu/~angel/WebGL/7E
- www.opengl.org
- get.webgl.org
- www.kronos.org/webgl
- www.chromeexperiments.com/webgl
- learningwebgl.com

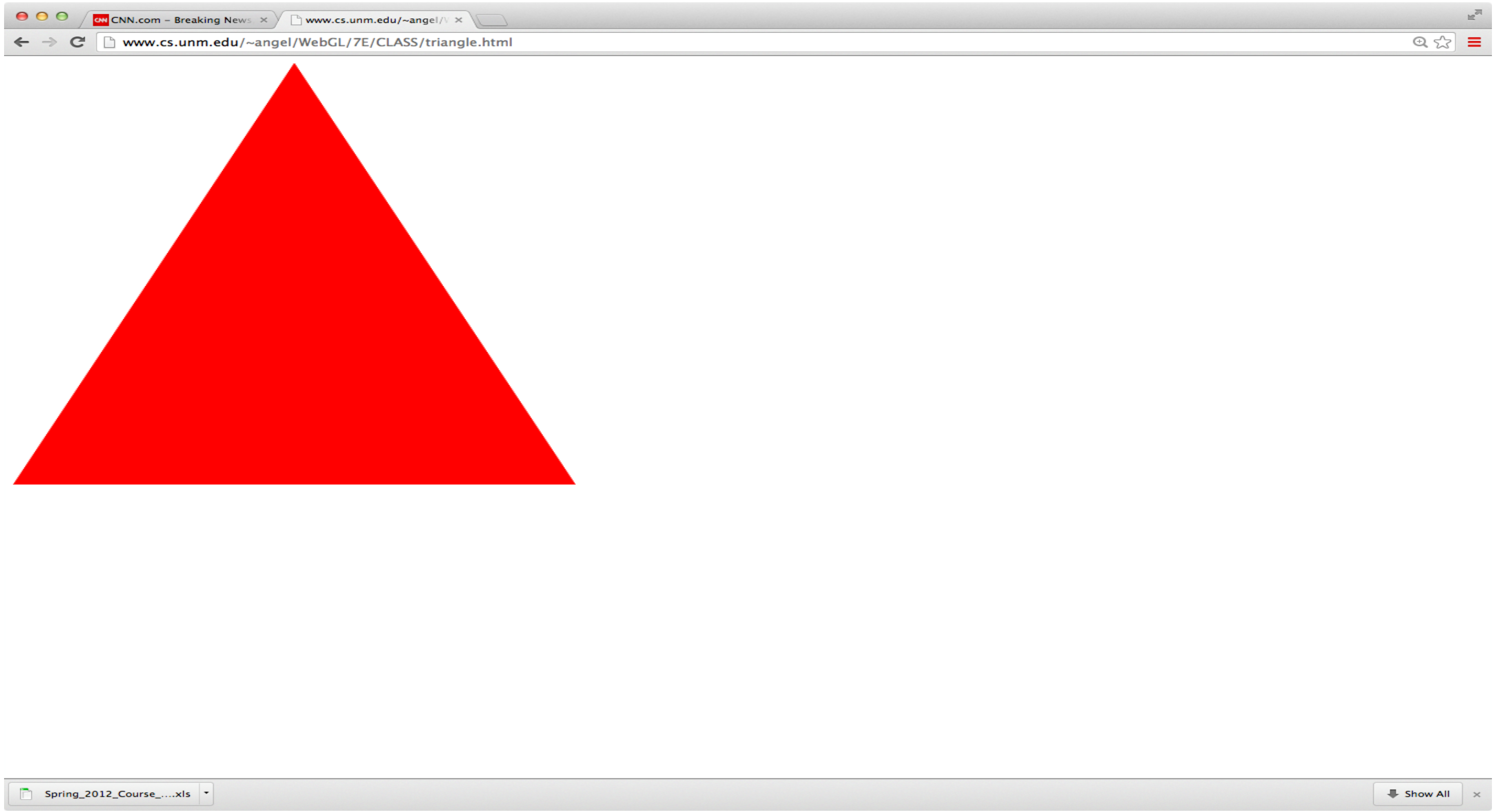
# Example Code in JS

- Example: Draw a triangle
  - Each application consists of (at least) two files
  - HTML file and a JavaScript file
- HTML
  - describes page
  - includes utilities
  - includes shaders
- JavaScript
  - contains the graphics

# Coding in WebGL

- Can run WebGL on any recent browser
  - Chrome
  - Firefox
  - Safari
  - IE
- Code written in JavaScript
- JS runs within browser
  - Use local resources

# Example: triangle.html



# Example Code

```
<!DOCTYPE html>
<html>
<head>
<script id="vertex-shader" type="x-shader/x-vertex">
attribute vec4 vPosition;
void main(){
  gl_Position = vPosition;
}
</script>
<script id="fragment-shader" type="x-shader/x-fragment">
precision mediump float;
void main(){
    gl_FragColor = vec4( 1.0, 0.0, 0.0, 1.0 );
}
</script>
```

# HTML File (cont)

```
<script type="text/javascript" src="../Common/webgl-utils.js"></script>
<script type="text/javascript" src="../Common/initShaders.js"></script>
<script type="text/javascript" src="../Common/MV.js"></script>
<script type="text/javascript" src="triangle.js"></script>
</head>
<body>
<canvas id="gl-canvas" width="512" height="512">
Oops ... your browser doesn't support the HTML5 canvas element
</canvas>
</body>
</html>
```

# JS File

```
var gl;
var points;

window.onload = function init(){
    var canvas = document.getElementById( "gl-canvas" );
     gl = WebGLUtils.setupWebGL( canvas );
     if ( !gl ) { alert( "WebGL isn't available" );  }

// Three Vertices

var vertices = [
        vec2( -1, -1 ),
        vec2(  0,  1 ),
        vec2(  1, -1 )
];
```

# JS File (cont)

```
//  Configure WebGL
//
    gl.viewport( 0, 0, canvas.width, canvas.height );
    gl.clearColor( 1.0, 1.0, 1.0, 1.0 );

//  Load shaders and initialize attribute buffers

    var program = initShaders( gl, "vertex-shader", "fragment-shader" );
    gl.useProgram( program );

// Load the data into the GPU

    var bufferId = gl.createBuffer();
    gl.bindBuffer( gl.ARRAY_BUFFER, bufferId );
    gl.bufferData( gl.ARRAY_BUFFER, flatten(vertices), gl.STATIC_DRAW );
```

# JS File (cont)

```
// Associate out shader variables with our data buffer

    var vPosition = gl.getAttribLocation( program, "vPosition" );
    gl.vertexAttribPointer( vPosition, 2, gl.FLOAT, false, 0, 0 );
    gl.enableVertexAttribArray( vPosition );
    render();
};

function render() {
   gl.clear( gl.COLOR_BUFFER_BIT );
   gl.drawArrays( gl.TRIANGLES, 0, 3 );
}
```

# Exercise

- Run triangle.html from the class website
- Load the triangle.html and triangle.js to your computer and run them from there
- Edit the two files to change the color and display more than one triangle

# JavaScript Notes

- JavaScript (JS) is the language of the Web
  - All browsers will execute JS code
  - JavaScript is an interpreted object-oriented language
- References
  - Flanagan, JavaScript: The Definitive Guide, O'Reilly
  - Crockford, JavaScript, The Good Parts, O'Reilly
  - Many Web tutorials

# JS Notes

- Is JS slow?
  - JS engines in browsers are getting much faster
  - Not a key issues for graphics since once we get the data to the GPU it doesn't matter how we got the data there
- JS is a (too) big language
  - We don't need to use it all
  - Choose parts we want to use
  - Don't try to make your code look like C or Java

# JS Notes

- Very few native types:
  - numbers
  - strings
  - booleans
- Only one numerical type: 32 bit float
  - var x = 1;
  - var x = 1.0; // same
  - potential issue in loops
  - two operators for equality == and ===
- Dynamic typing

# Scoping

- Different from other languages
- Function scope
- variables are *hoisted* within a function
    - can use a variable before it is declared
- Note functions are first class objects in JS

# JS Arrays

- JS arrays are objects
  - inherit methods
  - var a = [1, 2, 3];
    is not the same as in C++ or Java
  - a.length     // 3
  - a.push(4); // length now 4
  - a.pop();   // 4
  - avoids use of many loops and indexing
  - Problem for WebGL which expects C-style arrays

# Typed Arrays

JS has typed arrays that are like C arrays

```
var a = new Float32Array(3)
var b = new Uint8Array(3)
```

Generally, we prefer to work with standard JS arrays and convert to typed arrays only when we need to send data to the GPU with the flatten function in MV.js

# A Minimalist Approach

- We will use only core JS and HTML
  - no extras or variants
- No additional packages
  - CSS
  - JQuery
- Focus on graphics
  - examples may lack beauty
- You are welcome to use other variants as long as I can run them from your URL

# What is Computer Graphics?

# Computer Graphics

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

# Example

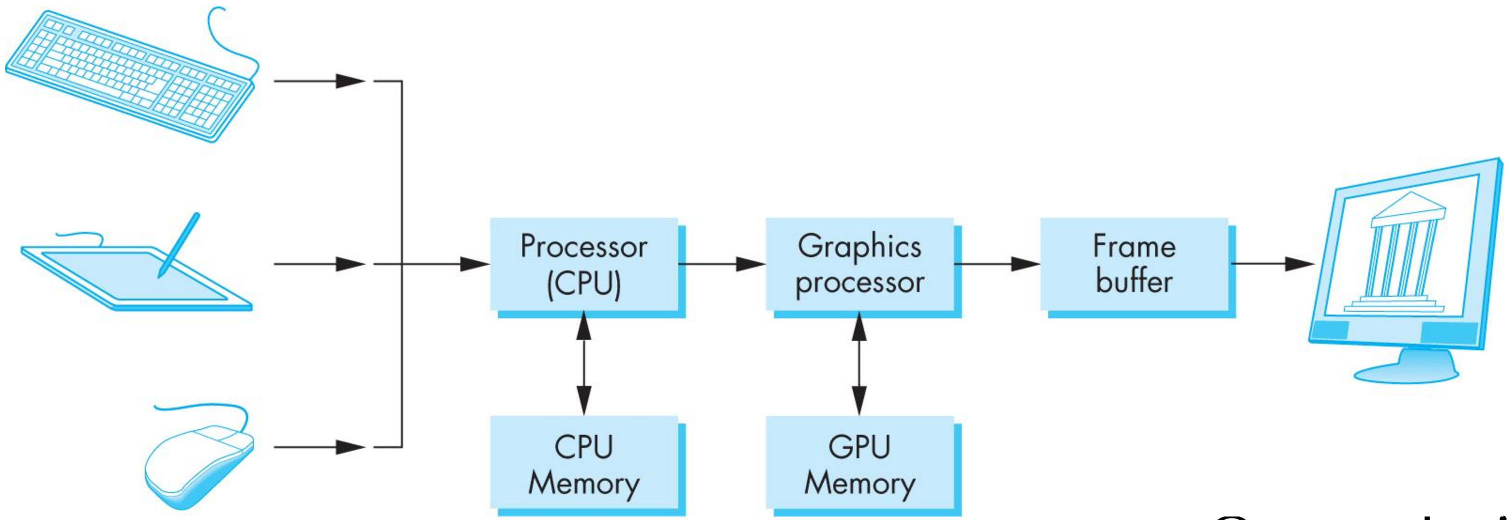
- Where did this image come from?

- What hardware/software did we use 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 but Maya is built on top of OpenGL
- **Hardware**: PC with graphics card for modeling and rendering

# Basic Graphics System




Output device

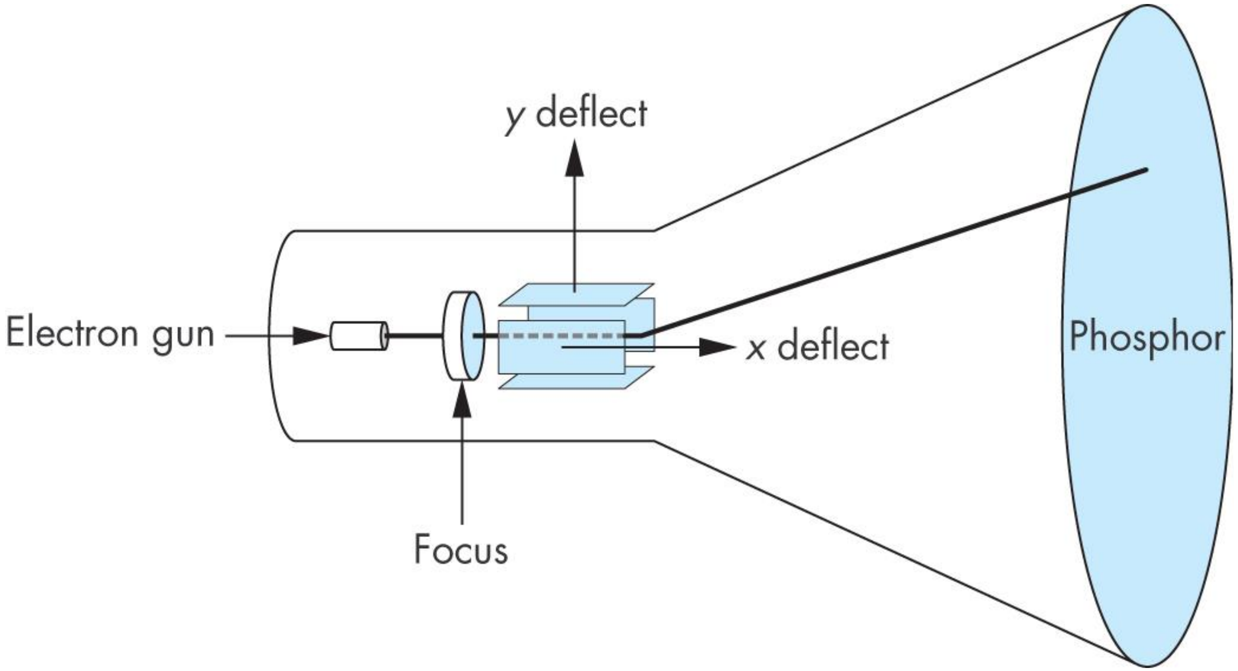
Input devices

Image formed in frame buffer

# Computer Graphics: 1950-1960

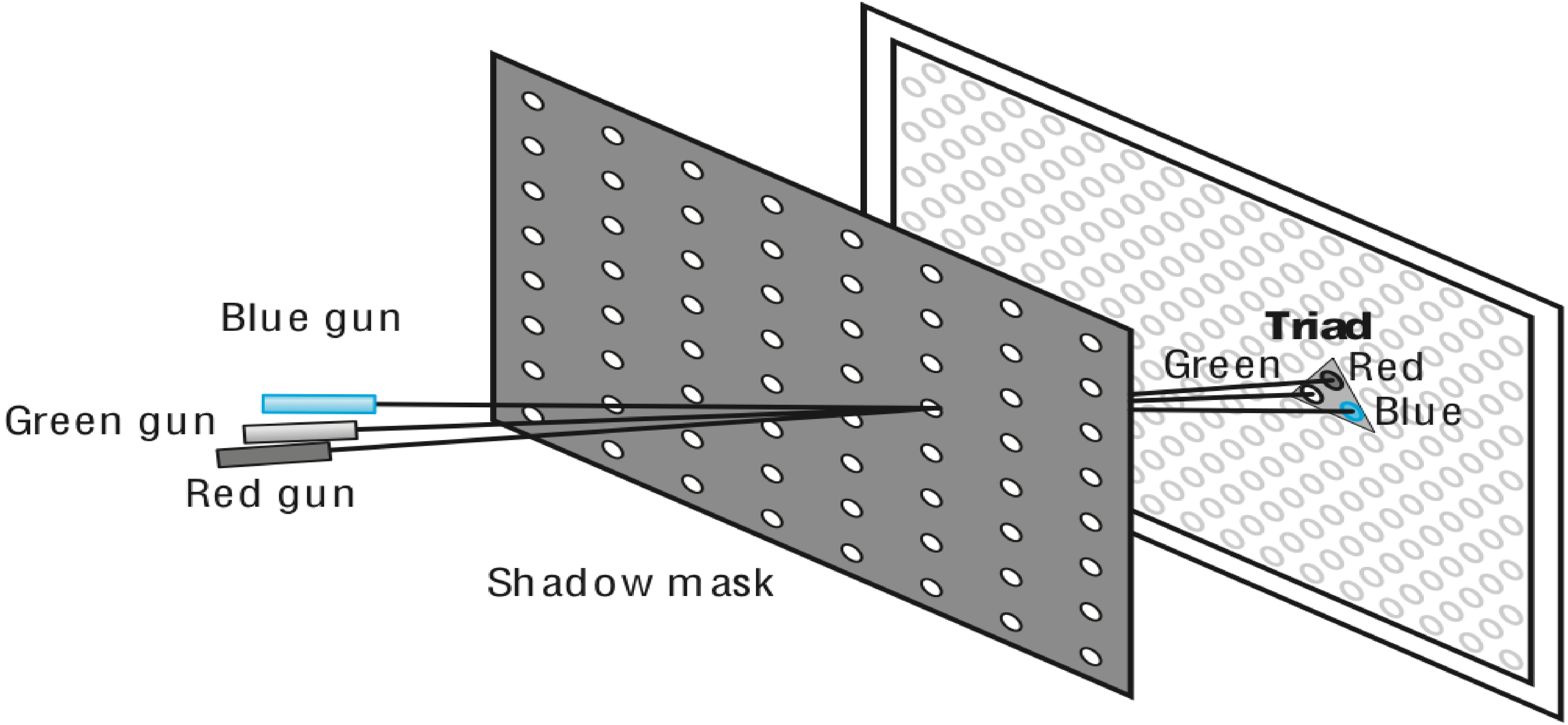
- Computer graphics goes back to the earliest days of computing
  - Strip charts
  - Pen plotters
  - Simple displays using A/D converters to go from computer to calligraphic CRT
- Cost of refresh for CRT too high
  - Computers slow, expensive, unreliable

# Cathode Ray Tube (CRT)




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

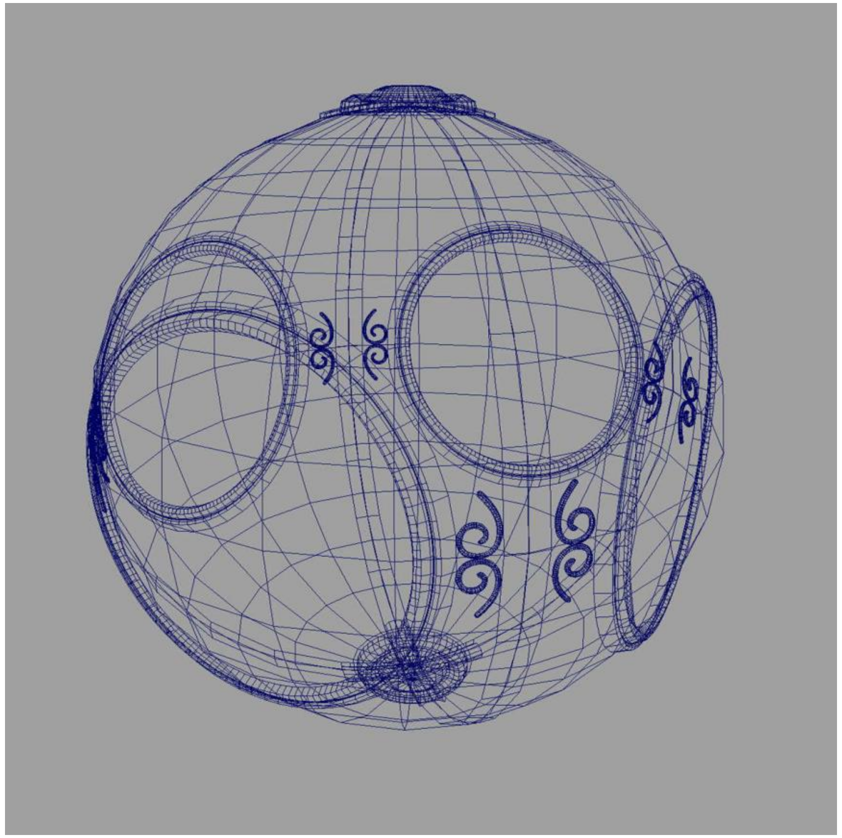
# Shadow Mask CRT



# Computer Graphics: 1960-1970

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

wireframe representation
of sun object

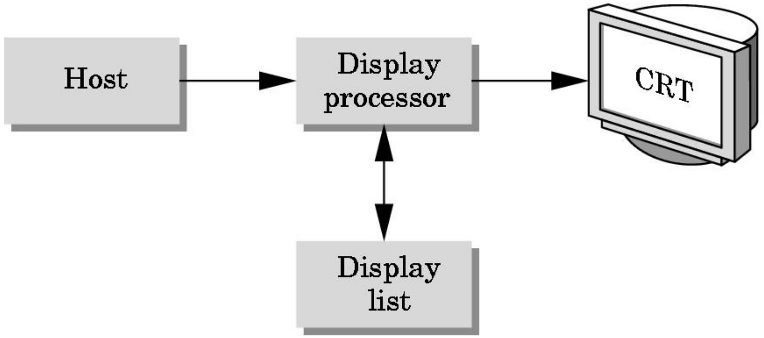


# Sketchpad

- 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)



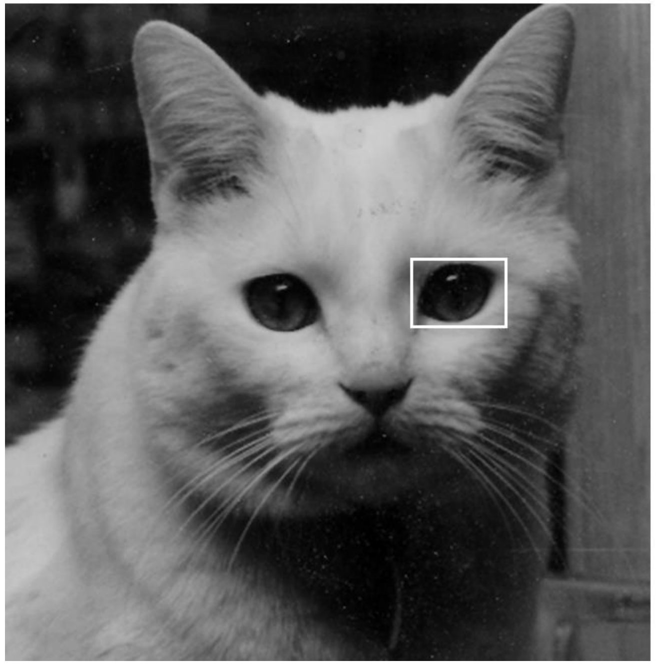

- Graphics stored in display list (display file) on display processor
- Host *compiles* display list and sends to DPU

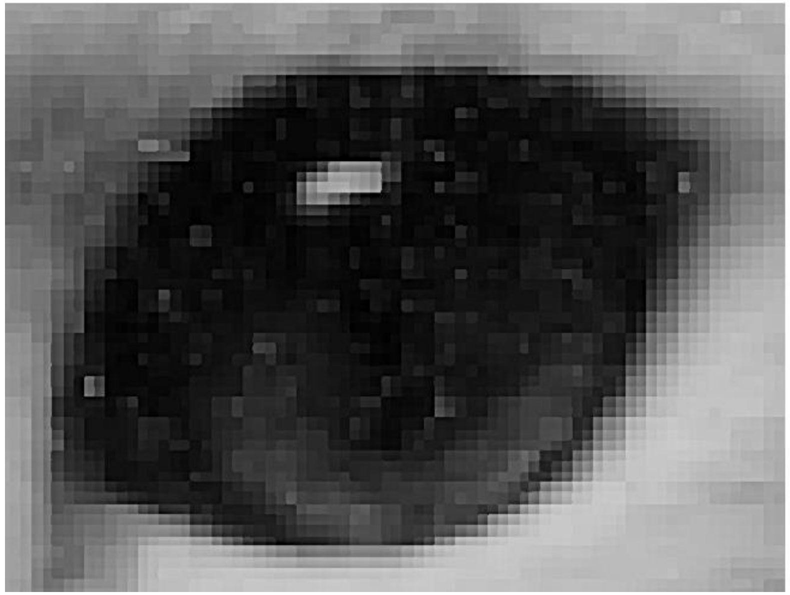
# Computer Graphics: 1970-1980

- Raster Graphics
- Beginning of graphics standards
  - IFIPS
    - GKS: European effort
      - Becomes ISO 2D standard
    - Core: North American effort
      - 3D but fails to become ISO standard
- Workstations and PCs

# Raster Graphics

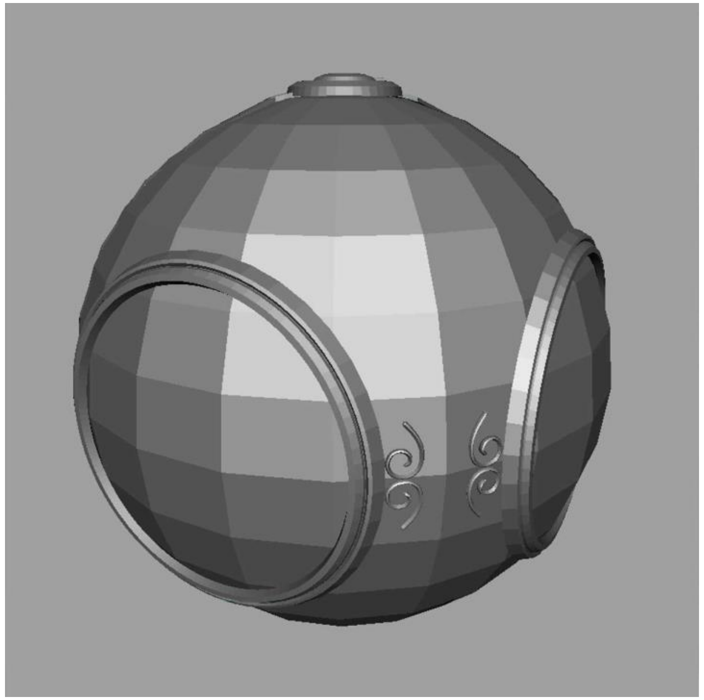
- Image produced as an array (the *raster*) of picture elements (*pixels*) in the *frame buffer*





# Raster Graphics

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

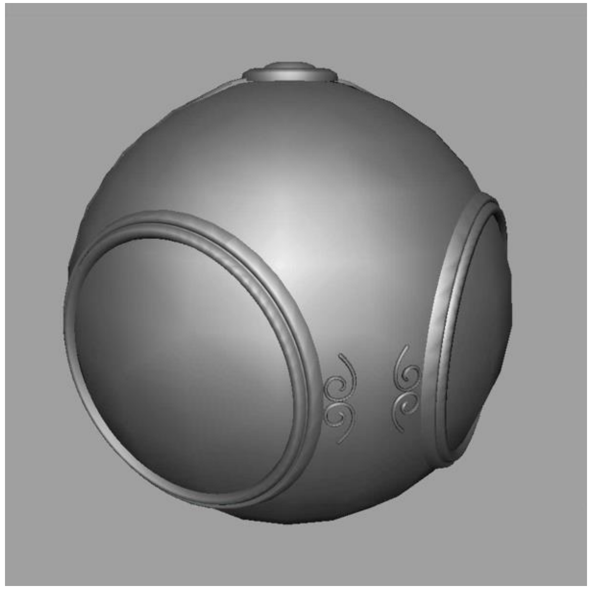


# PCs and Workstations

- 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

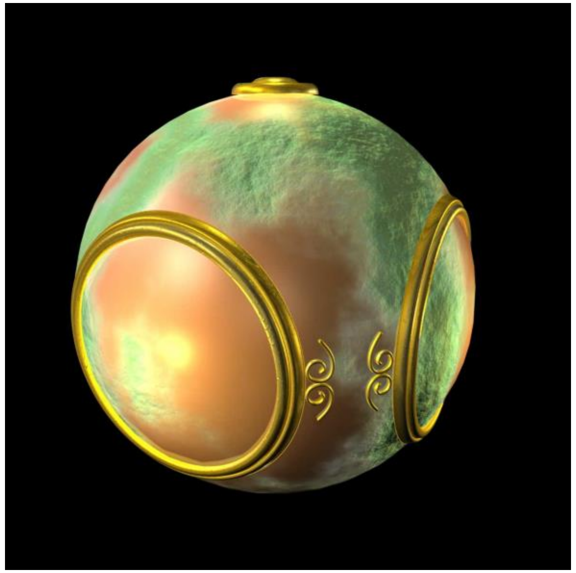
# Computer Graphics: 1980-1990

Realism comes to computer graphics



smooth shading

environment mapping



bump mapping

# Computer Graphics: 1980-1990

- Special purpose hardware
  - Silicon Graphics geometry engine
    - VLSI implementation of graphics pipeline
- Industry-based standards
  - PHIGS
  - RenderMan
- Networked graphics: X Window System
- Human-Computer Interface (HCI)

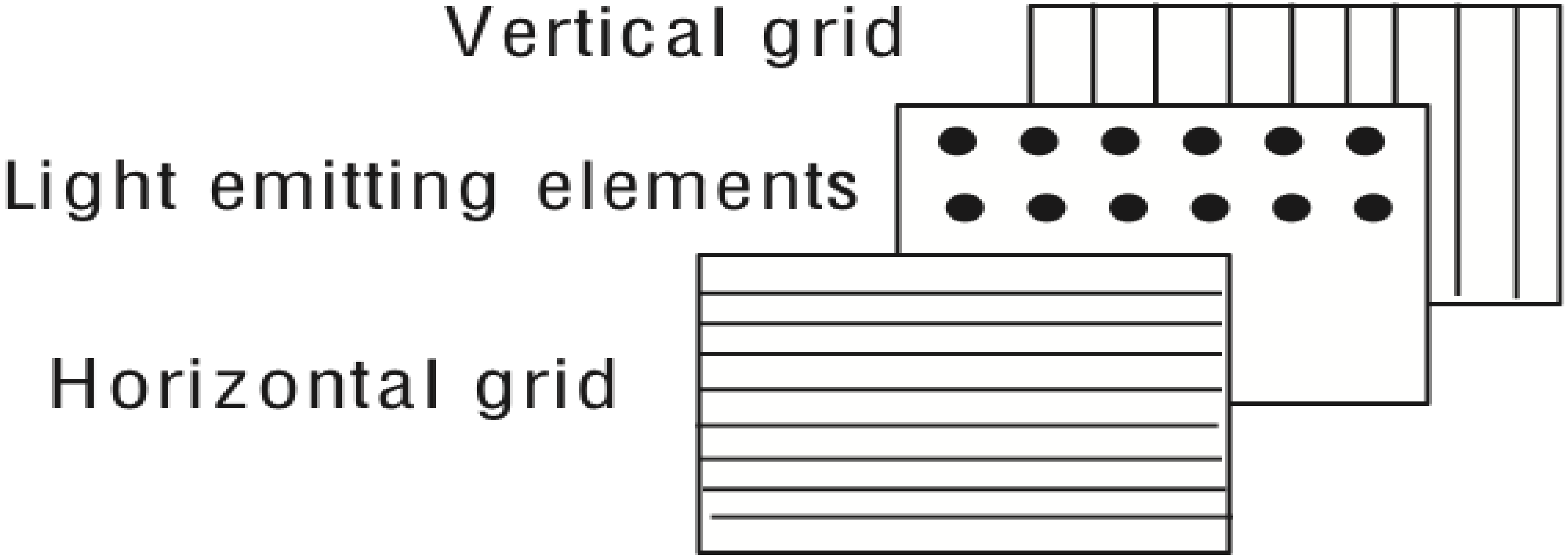
# Computer Graphics: 1990-2000

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

# Computer Graphics: 2000-2010

- 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
- New display technologies

# Generic Flat Panel Display



# Computer Graphics 2011-

- Graphics is now ubiquitous
  - Cell phones
  - Embedded
- OpenGL ES and WebGL
- Alternate and Enhanced Reality
- 3D Movies and TV

# Image Formation

# Objectives

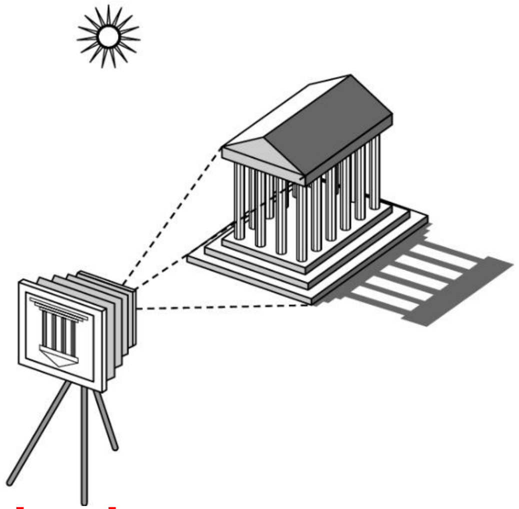
- Fundamental imaging notions
- Physical basis for image formation
  - Light
  - Color
  - Perception
- Synthetic camera model
- Other models

# 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)



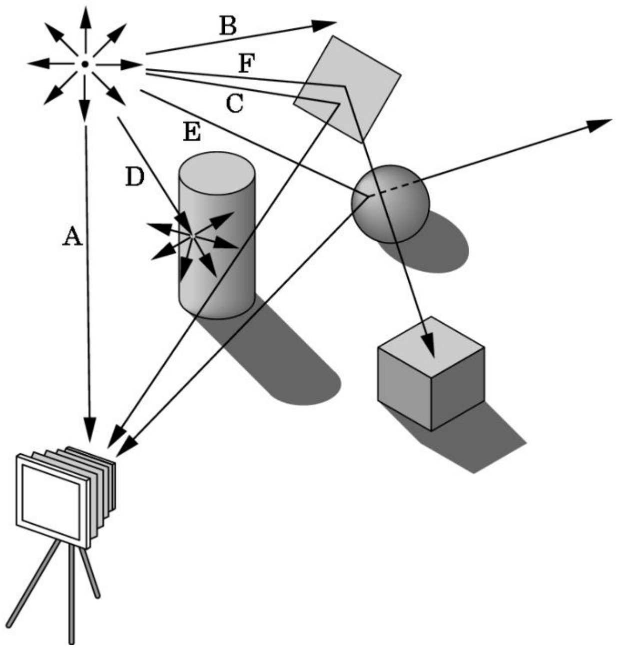
- Attributes that govern how light interacts with the materials in the scene
- Note the independence of the objects, the viewer, and the light source(s)

# 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

# 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.

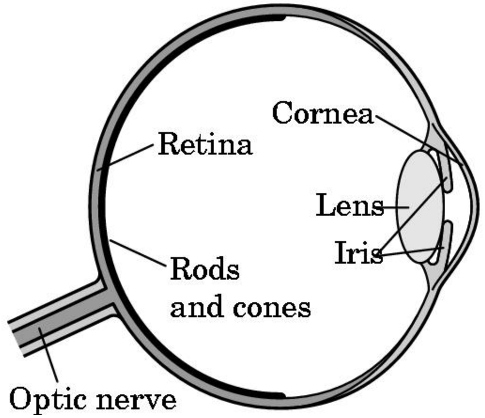


# Luminance and Color Images

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

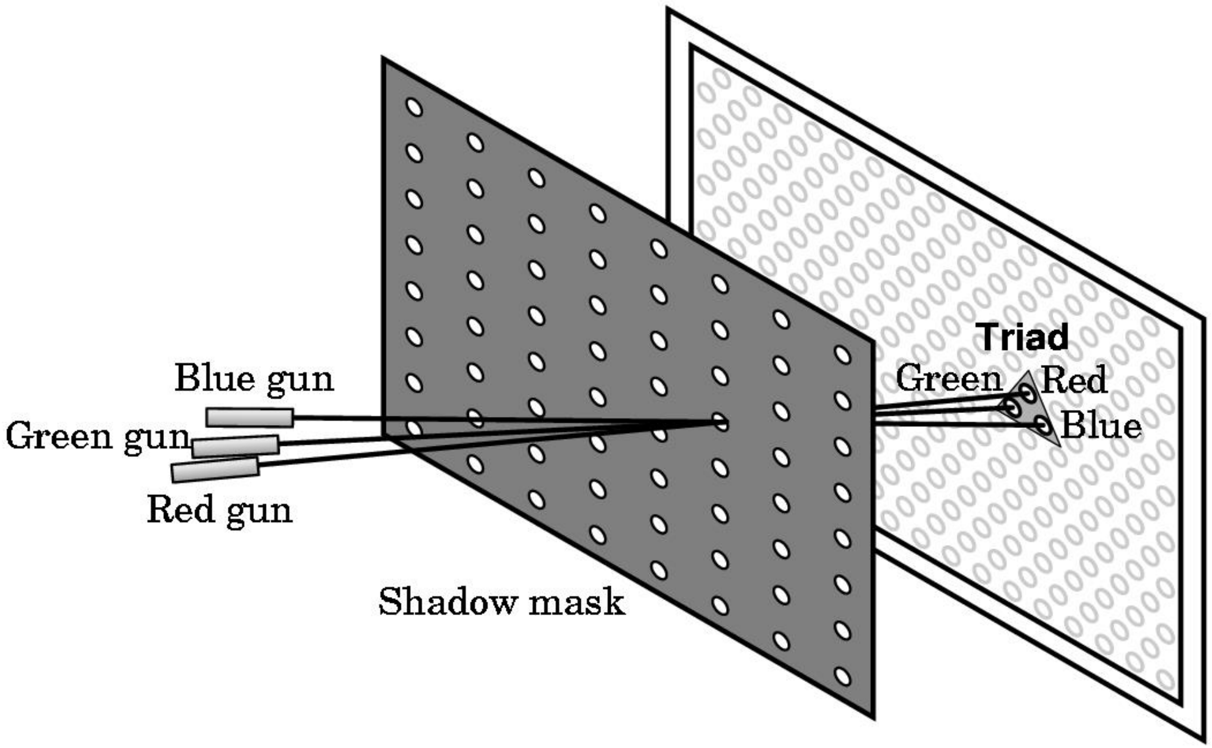
# Three-Color Theory

- 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

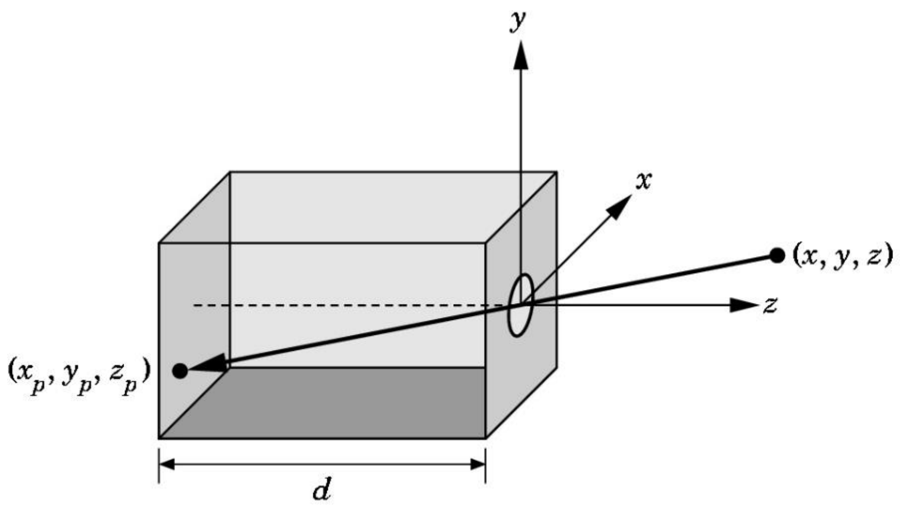
# Shadow Mask CRT



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

# Pinhole Camera

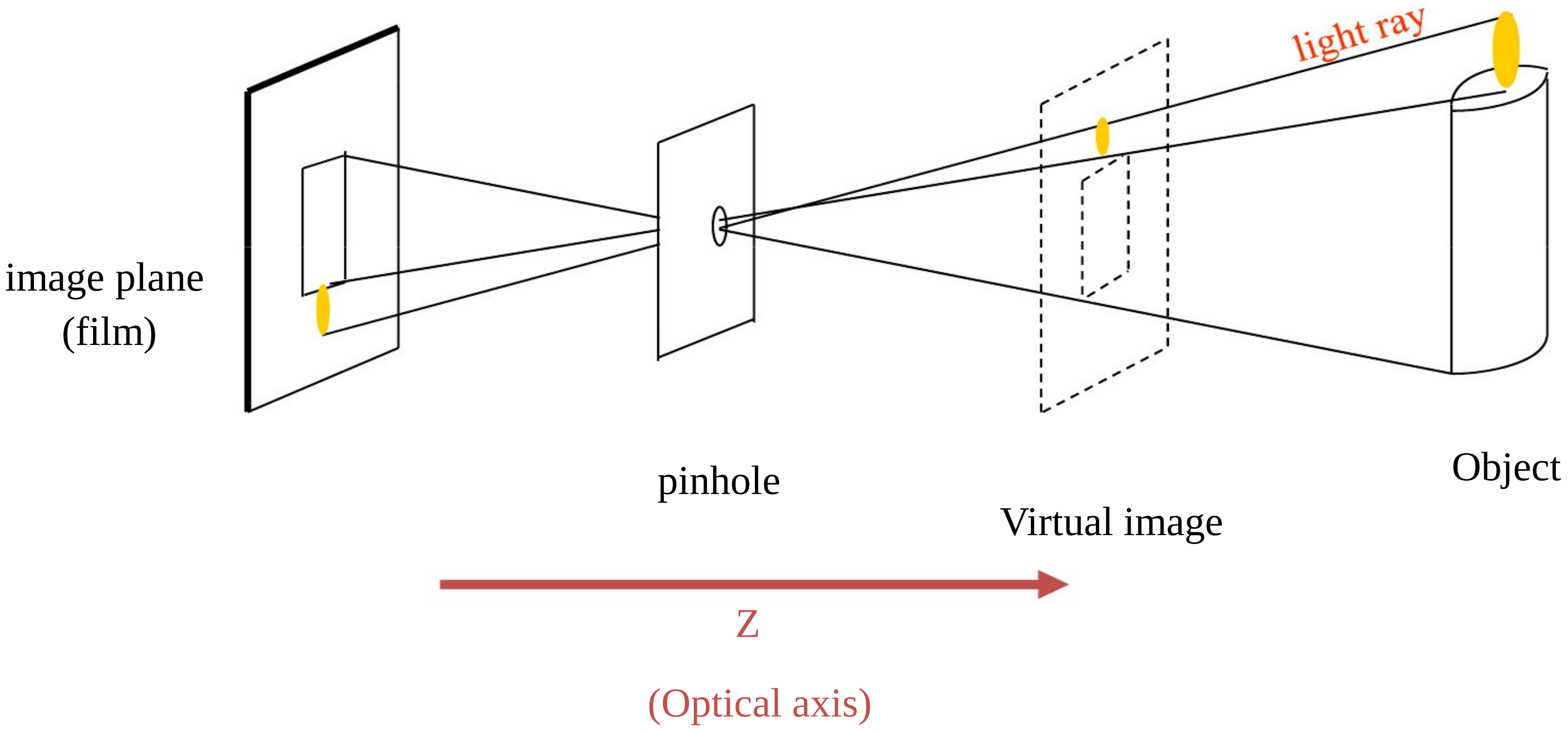



Use trigonometry to find projection of point at (x,y,z)

$x_p = -x/z/d \qquad y_p = -y/z/d \qquad z_p = d$

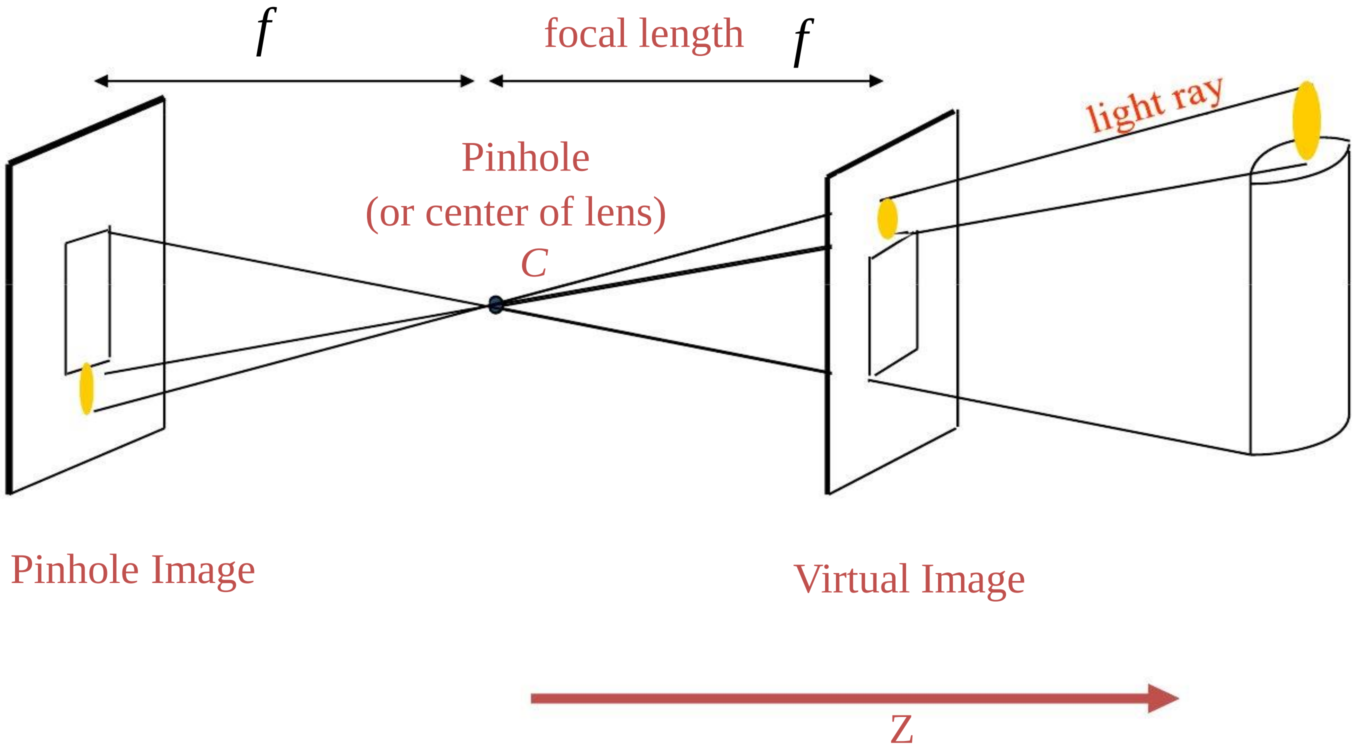
These are equations of simple perspective

# Image Formation – Pinhole Camera

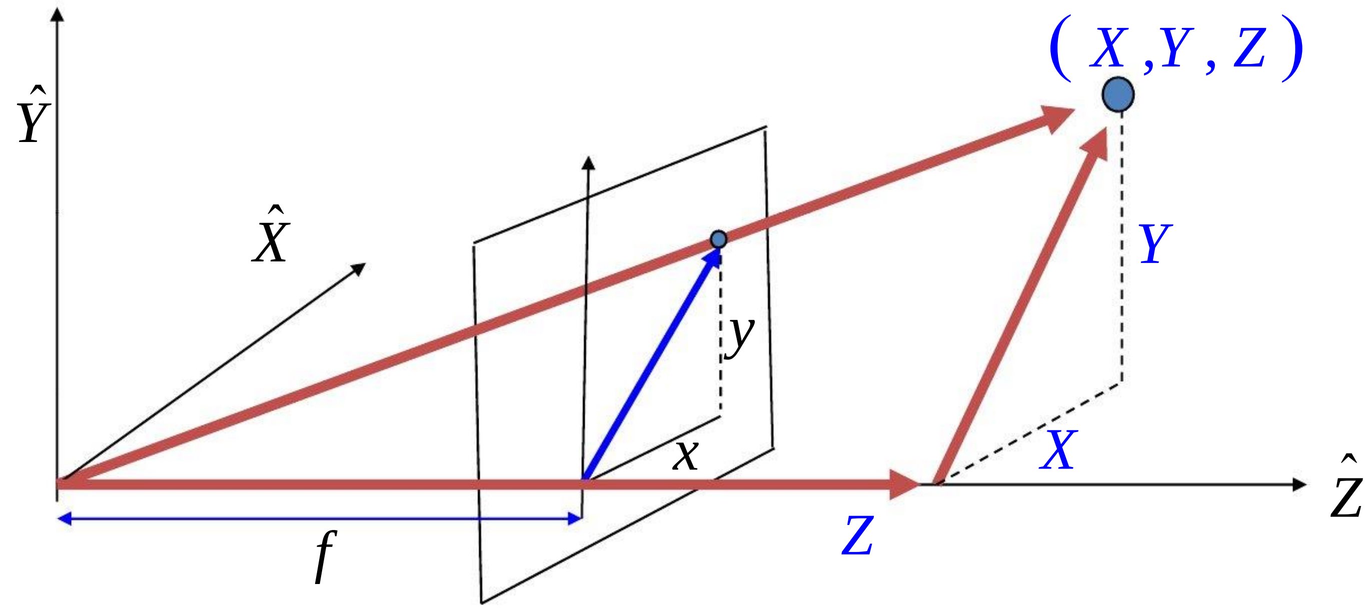


# Perspective Projection

- Pinhole camera ➔ Virtual image



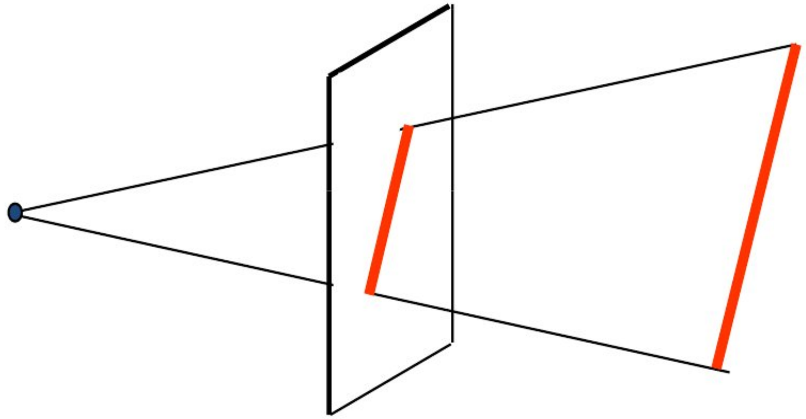
# Projection Equation




Similar triangles: $\frac{x}{X} = \frac{y}{Y} = \frac{f}{Z}$ $\longrightarrow$ $\boxed{(x, y) = \frac{f}{Z}(X, Y)}$

# Perspective Projection: Properties

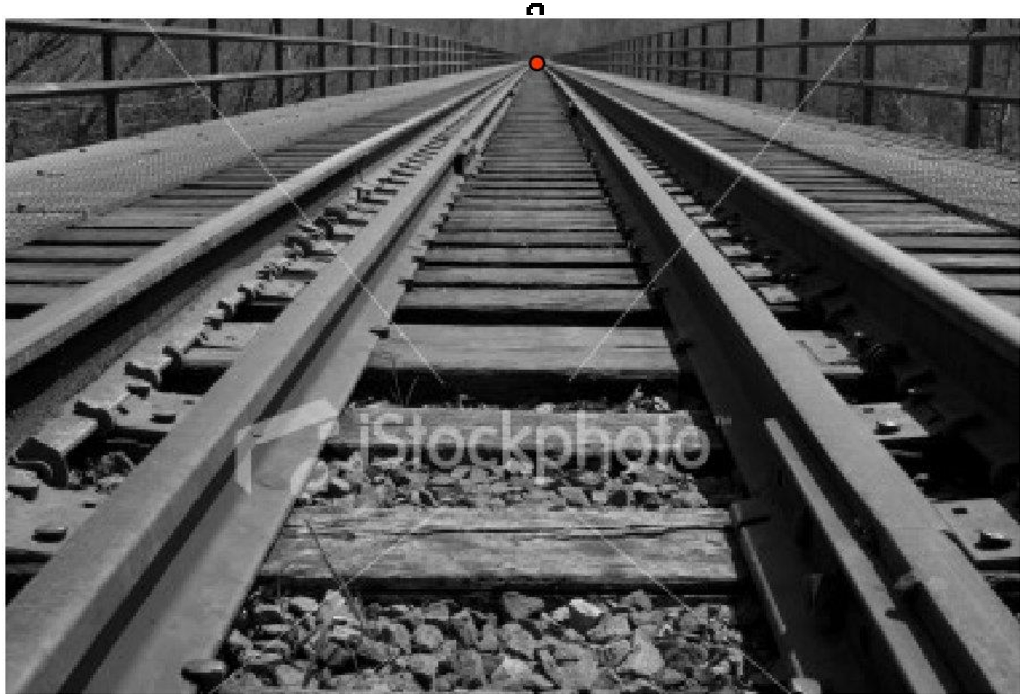
- 3D points ➔ image points
- 3D straight lines ➔ image straight lines



- 3D Polygons ➔ image polygons

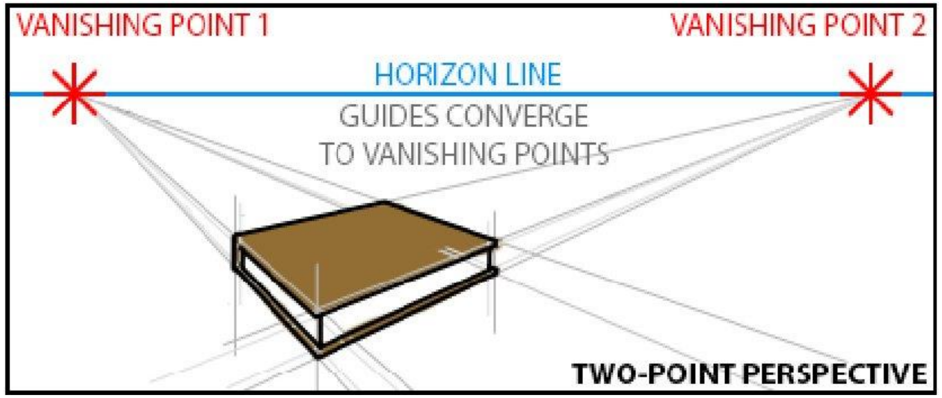
# Properties: Vanishing Points

- Image of an infinitely distant 3D point



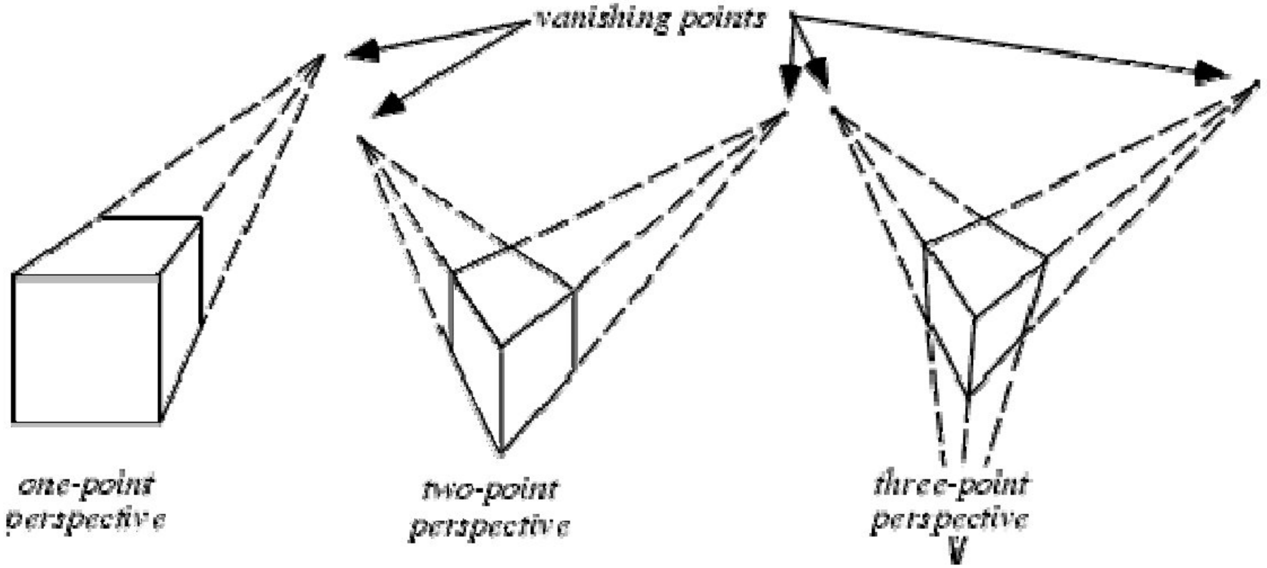
# Vanishing Point + Horizon

- 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

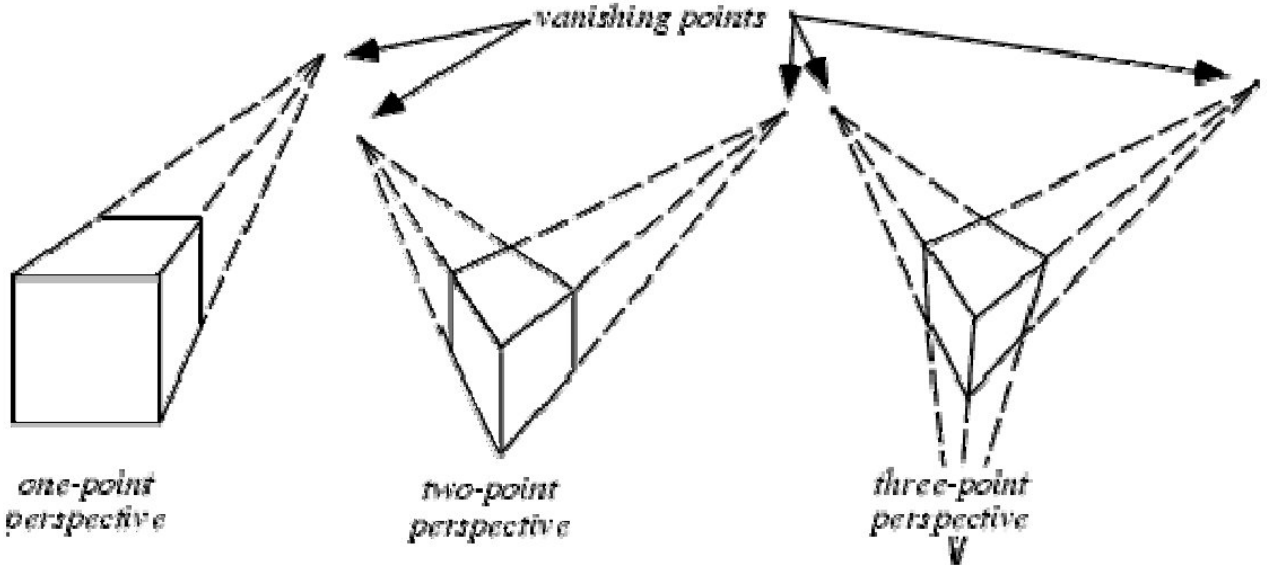
# Properties: Vanishing Points




3D vertical and horizontal parallel
to image plane

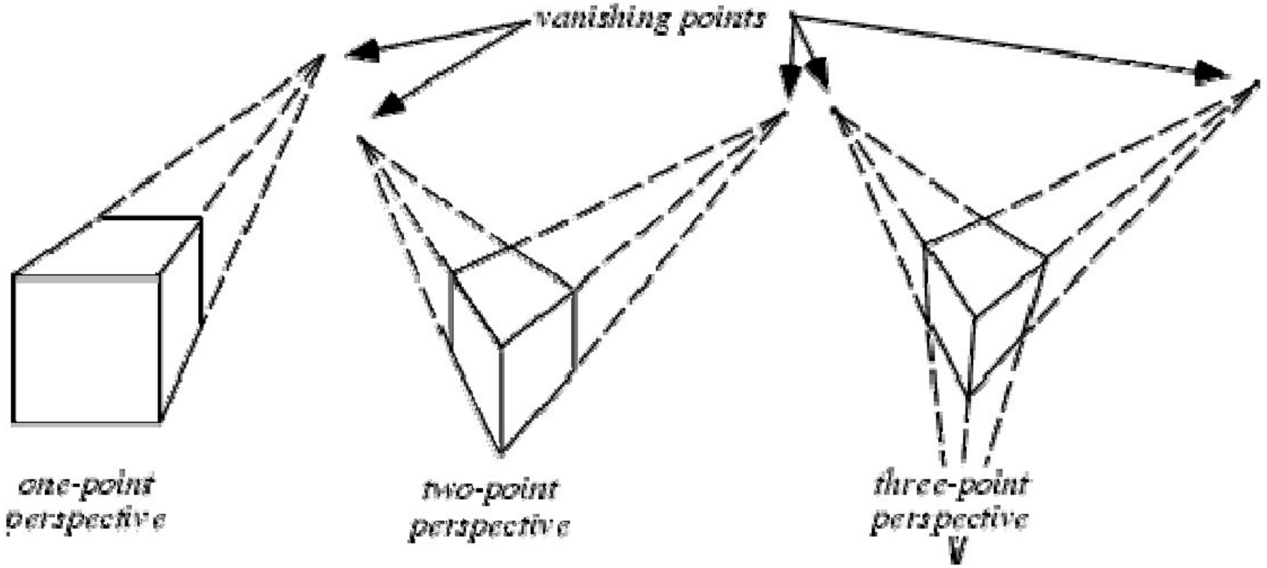
(Two vanishing points at "infinity" in image plane)

# Properties: Vanishing Points



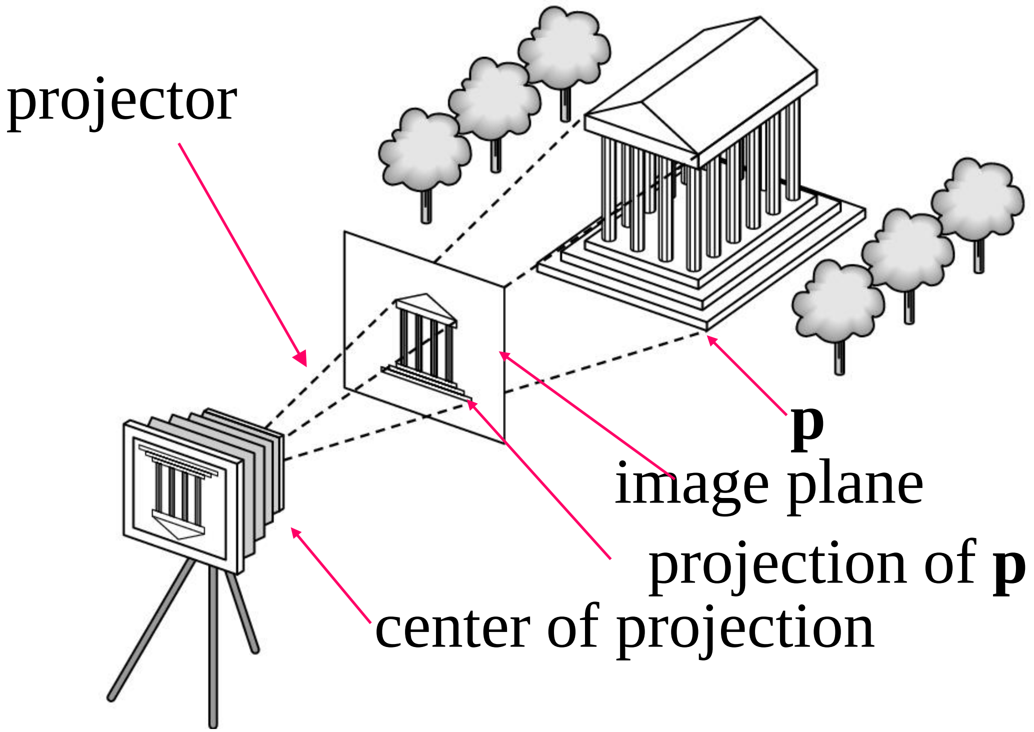

3D vertical parallel to image vertical
(One vanishing point at image "infinity")

# Properties: Vanishing Points




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

# Synthetic Camera Model



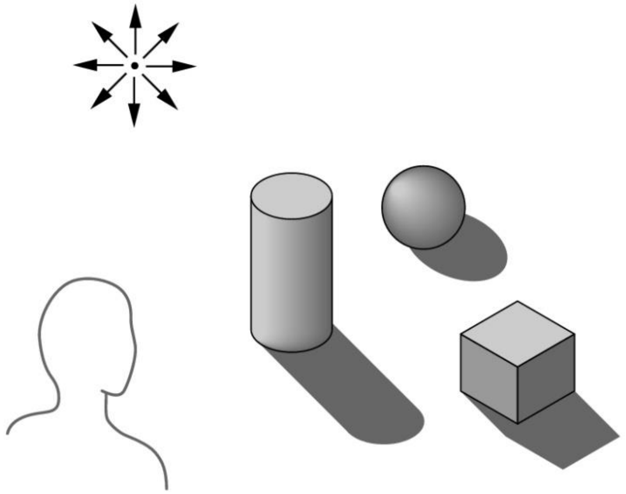

- Object and viewer specifications are independent
- All projectors are straight lines through center of projection
- Image has limited

# 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?

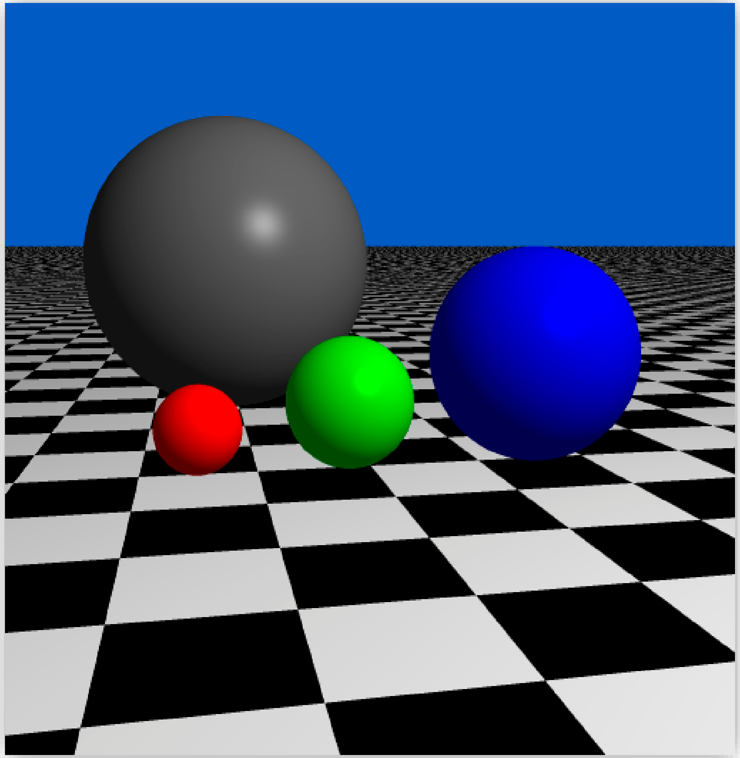
- Ray tracing seems more physically based so why don't we use it to design a graphics system?
- Possible and is actually simple for simple objects such as polygons and quadrics with simple point sources
- In principle, can produce global lighting effects such as shadows and multiple reflections but ray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

# Ray Tracing

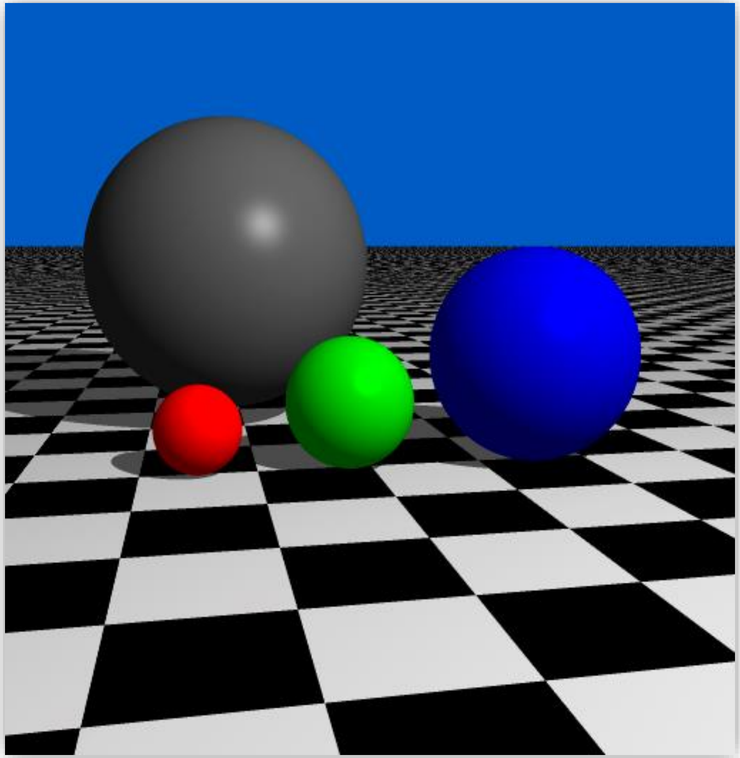
- Ray tracing is a image-precision algorithm: Visibility determined on a per-pixel basis
  - Trace one (or more) rays per pixel
  - Compute closest object (triangle, sphere, etc.) for each ray
- Produces realistic results
- Computationally expensive

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

# Shadows

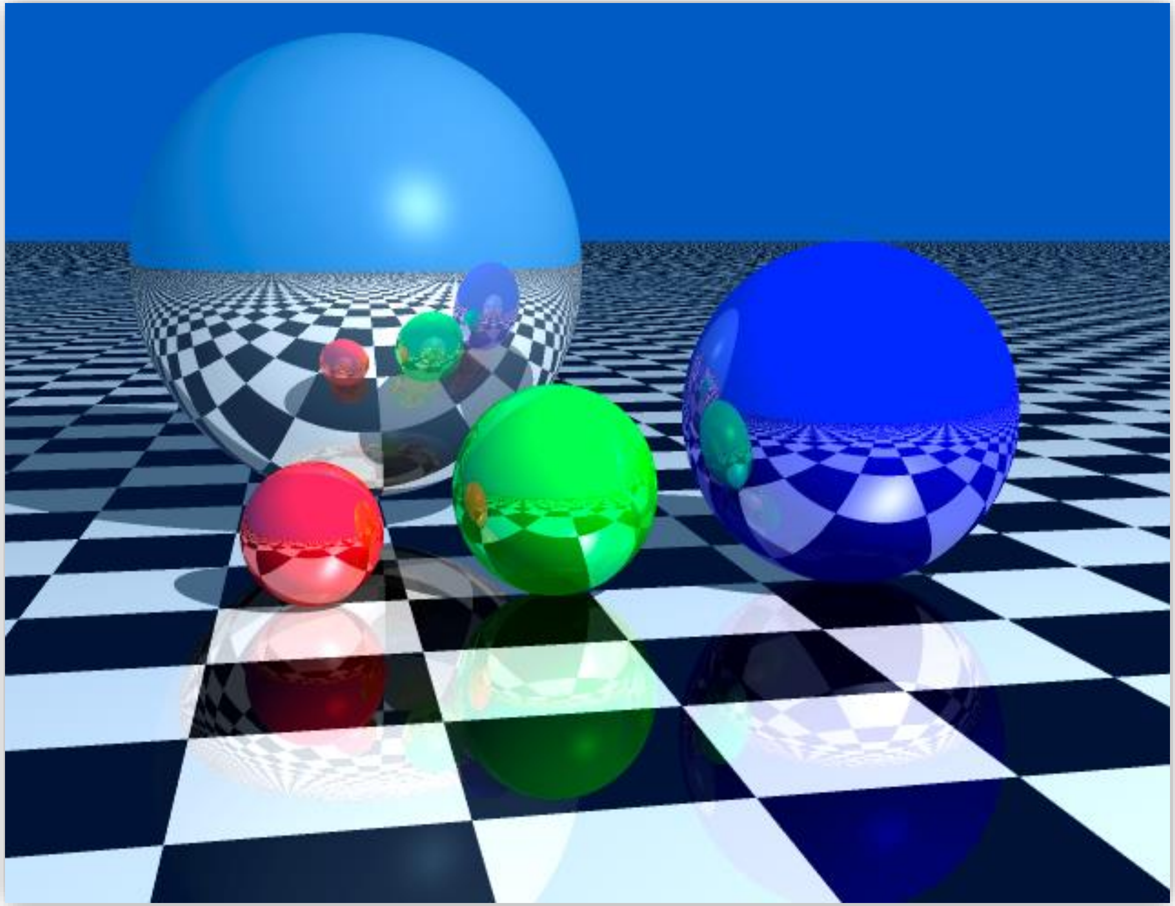


Standard Scene

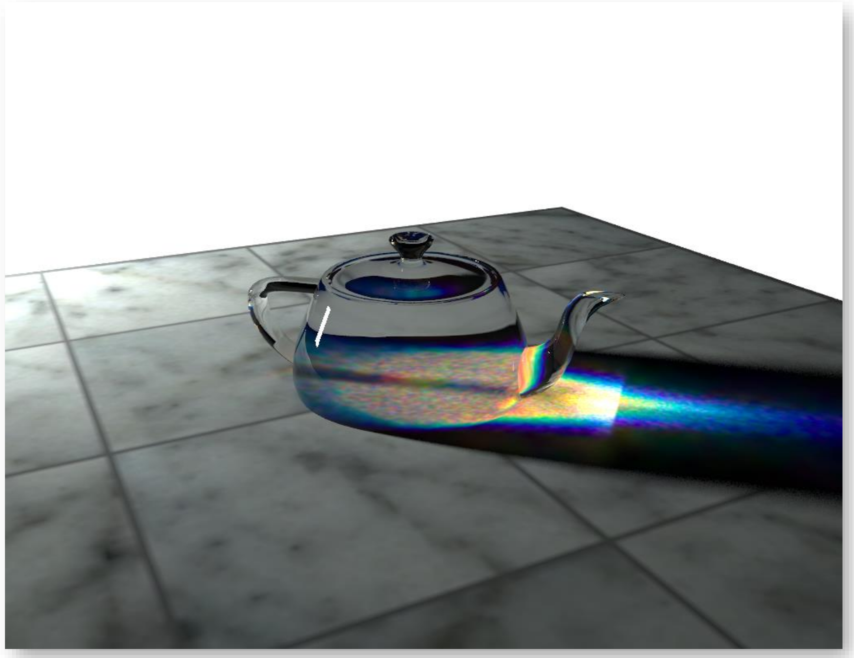


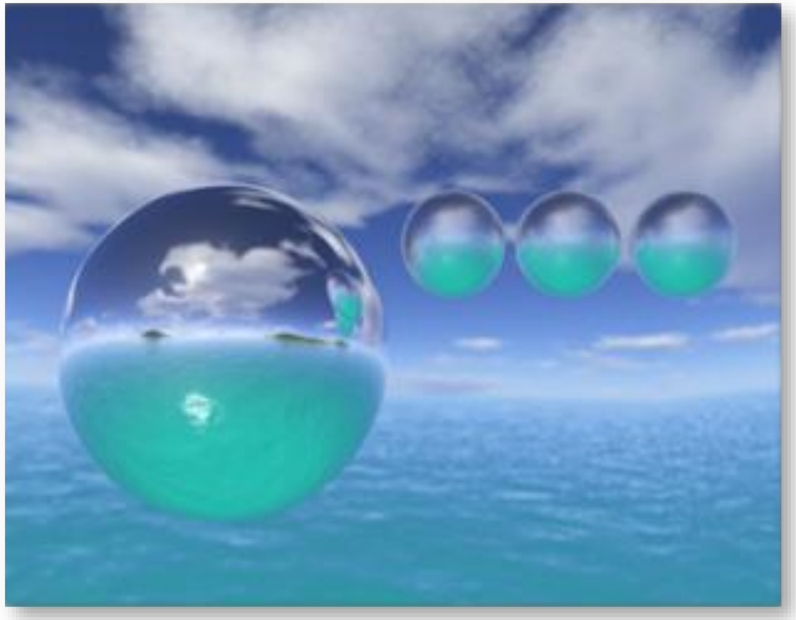
Scene With Shadows

# Reflection



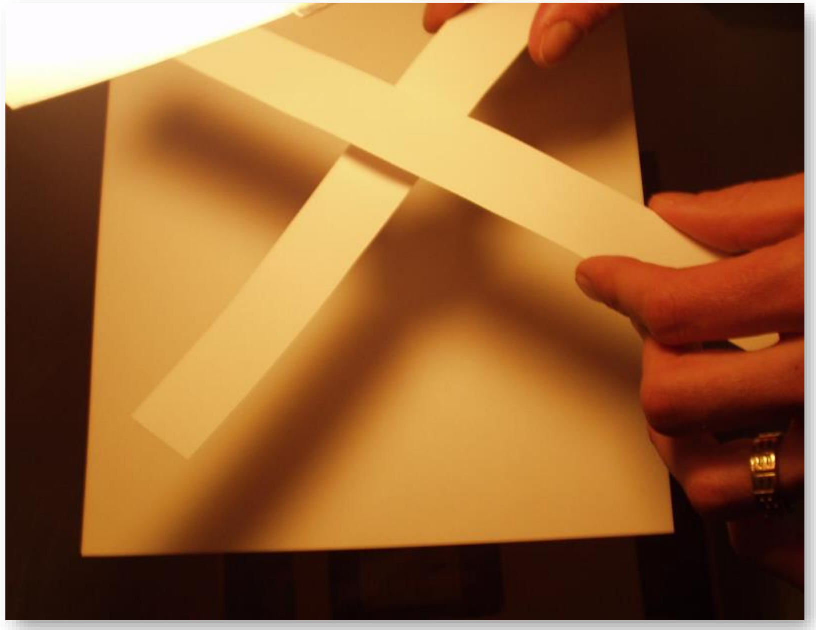
# Translucency Examples



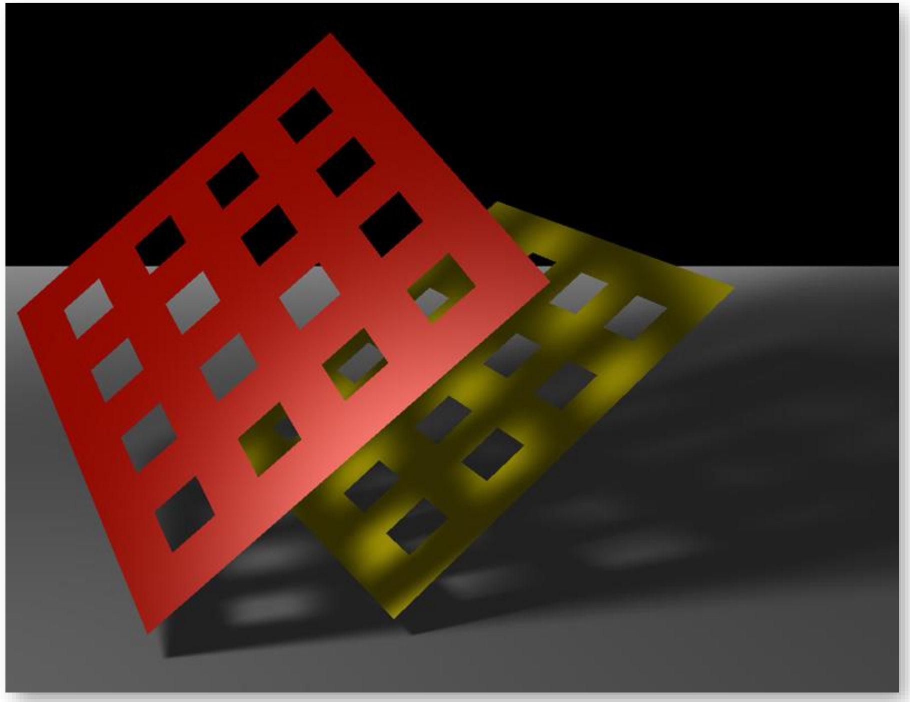


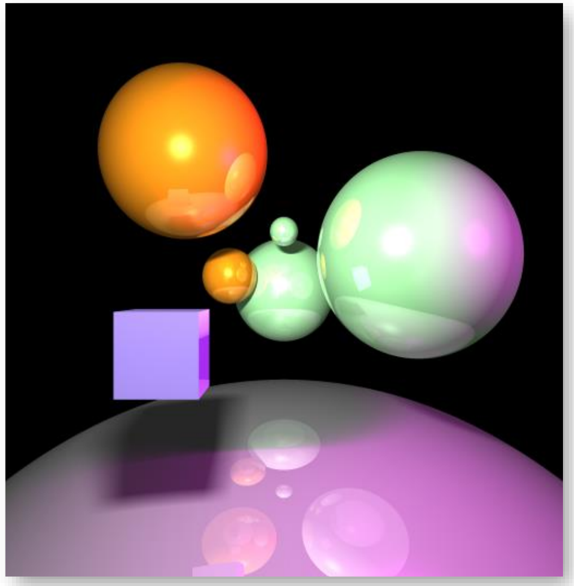
# Soft Shadows

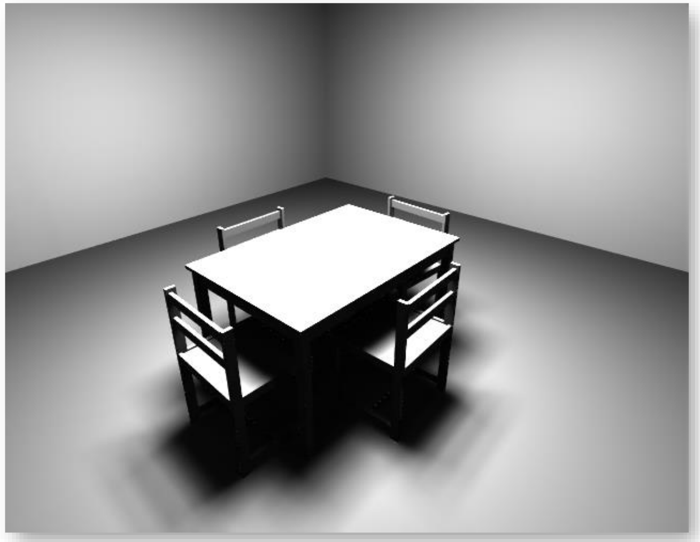
- In most graphics applications (and in our ray tracer so far), we've assumed point light sources
  - In the real world, lights have area
  - This leads to soft shadows in the real world, which we can't yet simulate in our ray tracer



# Soft Shadow Examples



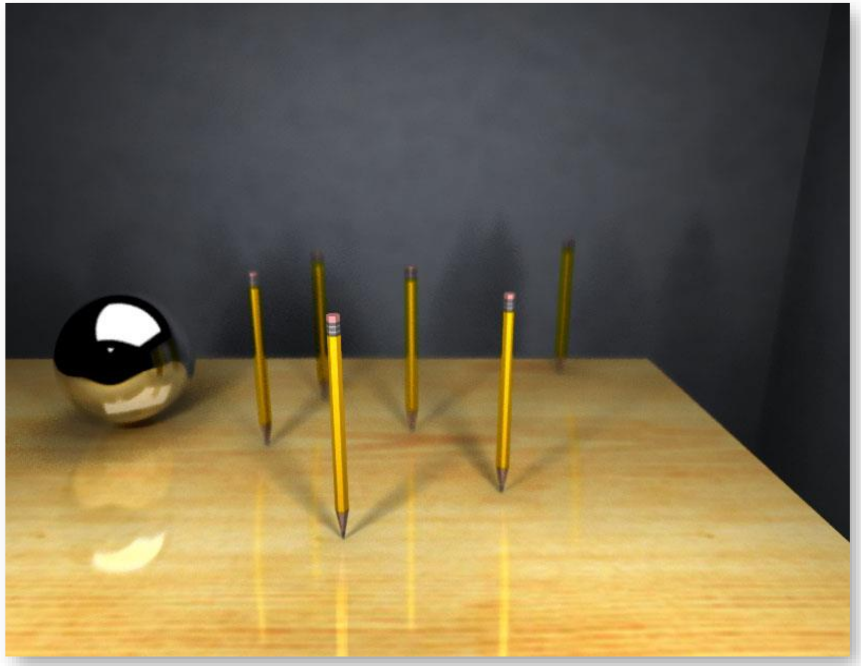


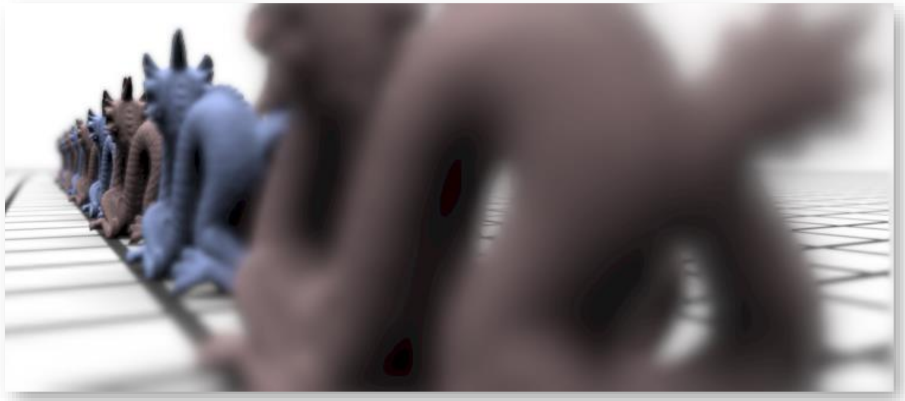


# Depth of Field

- 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

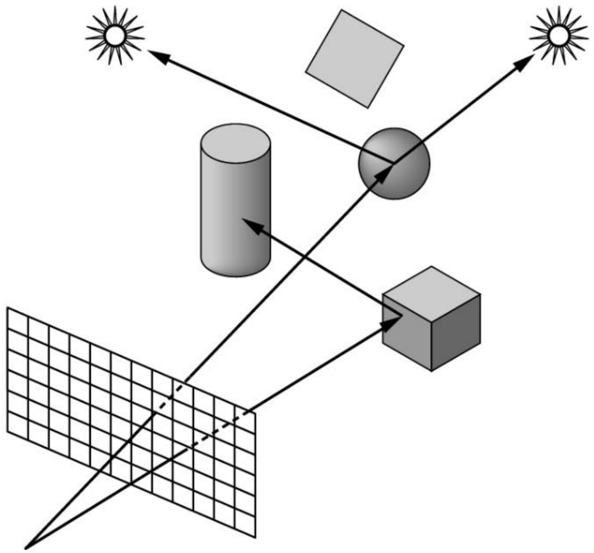
- Learn the basic design of a graphics system
- Introduce pipeline architecture
- Examine software components for an interactive graphics system

# 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

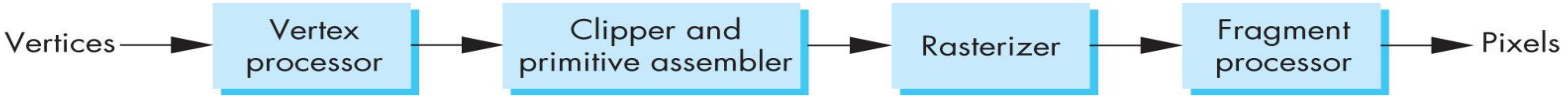
- **Ray tracing**: follow rays of light from center of projection until they either are absorbed by objects or go off to infinity
  - Can handle global effects
    - Multiple reflections
    - Translucent objects
  - Slow
  - Must have whole data base available at all times



- **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



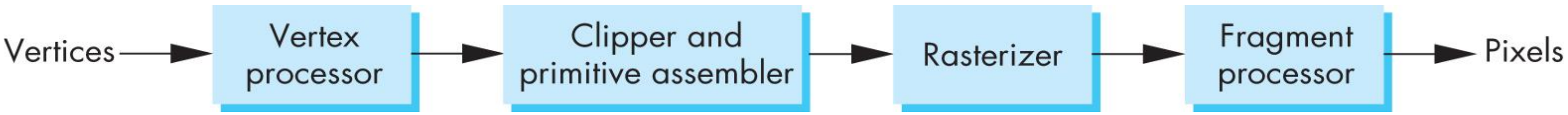

application program

display

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

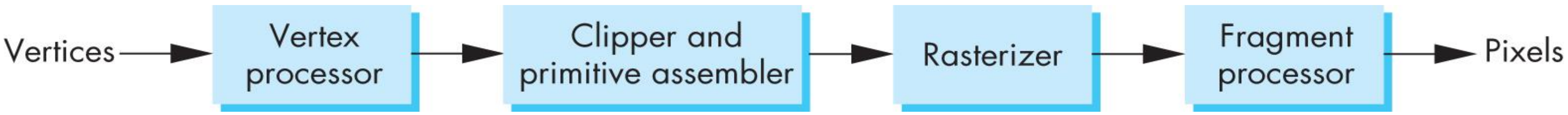
# Vertex Processing

- 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

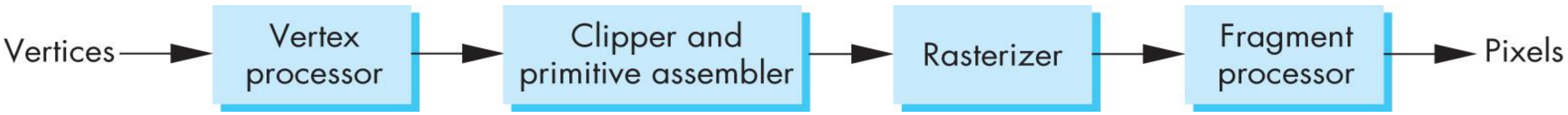
- *Projection* is the process that combines the 3D viewer with the 3D objects to produce the 2D image
  - Perspective projections: all projectors meet at the center of projection
  - Parallel projection: projectors are parallel, center of projection is replaced by a direction of projection



# Primitive Assembly

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

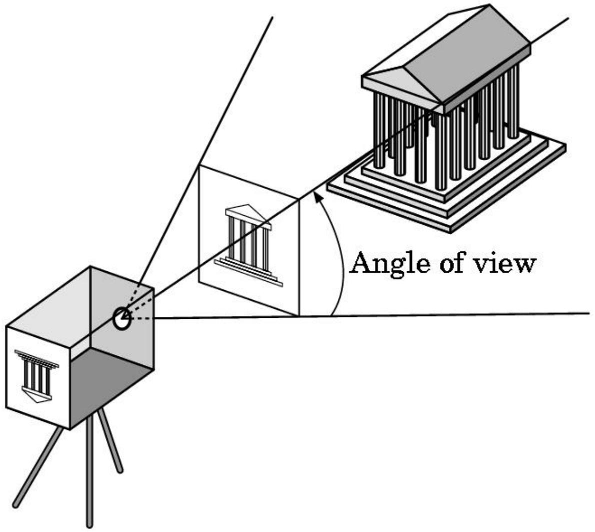
- Line segments
- Polygons
- Curves and surfaces

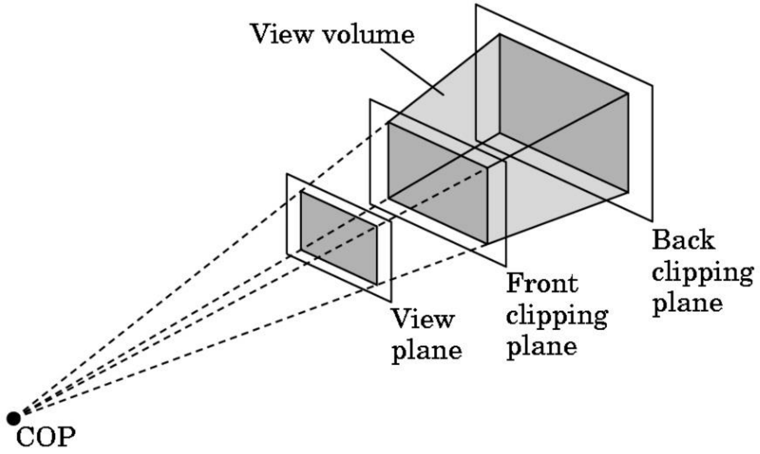


# Clipping

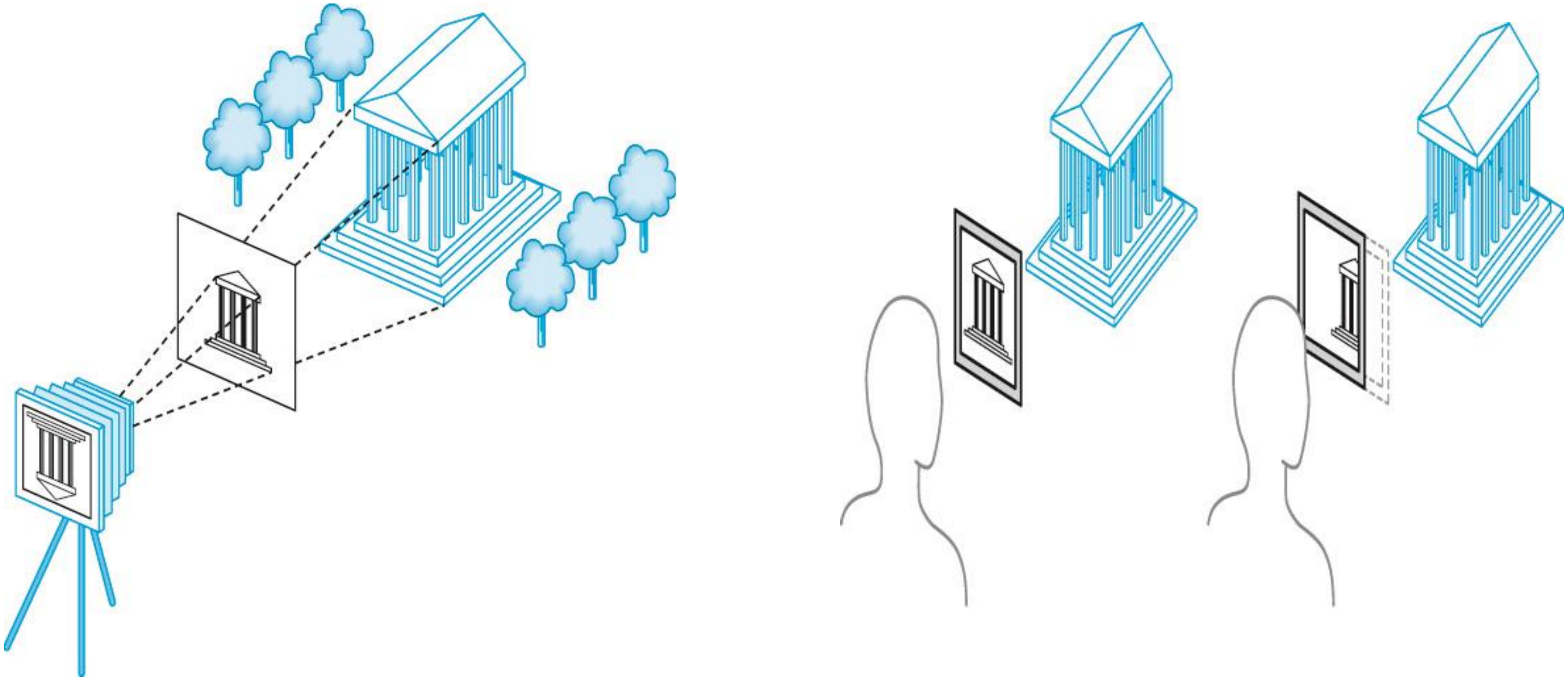
Just as a real camera cannot “see” the whole world, the virtual camera can only see part of the world or object space

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



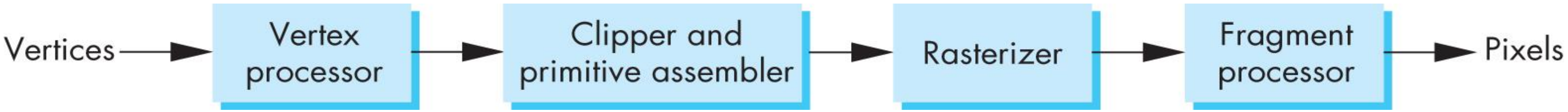



# Clipping



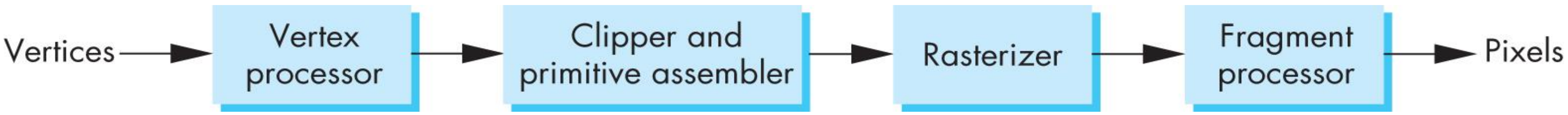
# Rasterization

- 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



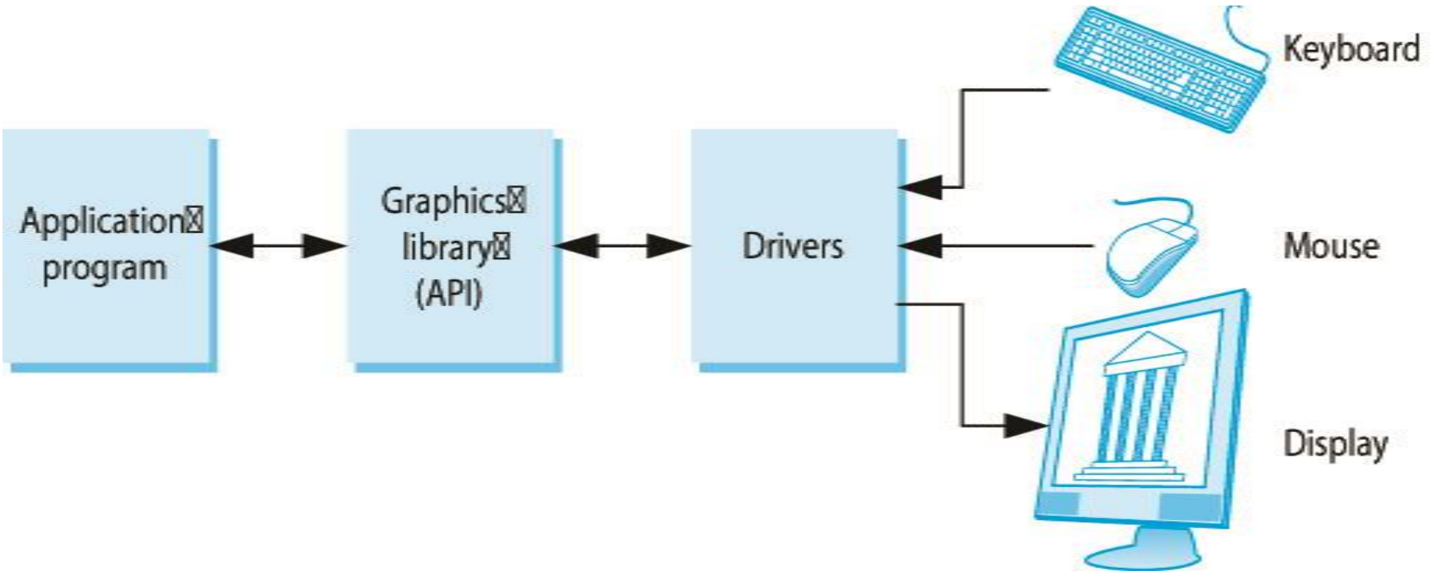
# Fragment Processing

- 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)

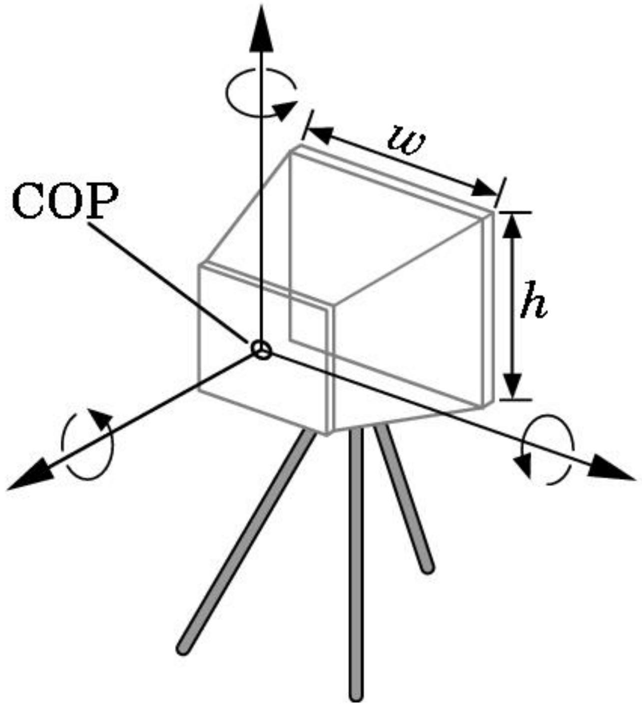
- Put geometric data in an array

```
var points = [
 vec3(0.0, 0.0, 0.0),
 vec3(0.0, 1.0, 0.0),
 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