Introduction to Computer Graphics

Sorin Babii
sorin.babii@cs.upt.ro

Contact Information

Lecture & Lab Notes:

- https://cv.upt.ro

Sorin Babii:

- Office: ASPC, P9
- email: sorin.babii@cs.upt.ro

References

- Interactive computer graphics: a top-down approach with WebGL, Edward Angel, Dave Shreiner, Pearson, 2015
- 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.opengl.org
- get.webgl.org
- www.kronos.org/webgl
- www.chromeexperiments.com/webgl
- learningwebgl.com

Objectives

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

Pre-requisites

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

Week 1

- Introduction
- Detailed Outline and Examples
- Example Code in JS
- What is Computer Graphics?
- Image Formation

Examples

http://www.cs.upt.ro/~sorin/webgl/

Course: Part 1

- Week 1
- Introduction
    - What is Computer Graphics?
    - Applications Areas
    - History
    - Image formation
    - Basic Architecture
- Docs: Angel, ch. 1

Course: Part 2

- Weeks 2-4
- Basic WebGL Graphics
    - Architecture
    - JavaScript
    - Web execution
    - Simple programs in 2D and 3D
    - Basic shaders and GLSL
- Docs: Angel: ch. 2

Course: Part 3

- Week 5
- Interaction
  - -Client-Server Model
  - -Event-driven programs
  - -Event Listeners
  - -Menus, Buttons, Sliders
  - -Position input
- Docs: Angel: ch. 3

Course: Part 4

- Weeks 6-9
- 3D Graphics
  - -Geometry
  - - Transformations
  - -Homogeneous Coordinates
  - - Viewing
  - -Lighting and Shading
- Docs: Angel: ch. 4…6

Course: Part 5

- Weeks 10-12
- Discrete Methods
    - Buffers
    - Pixel Maps
    - Texture Mapping
    - Compositing and Transparency
    - Off-Screen Rendering
- Docs: Angel: ch. 7

Course: Part 6

- Weeks 13-14
- Hierarchy and Procedural Methods
  - Tree Structured Models
    - Traversal Methods
    - Scene Graphs
    - Particle Systems
    - Agent Based Models
- Docs: Angel, Ch. 9-10

Course: Part 7

- Advanced Rendering
- Docs: Angel, Ch. 12
- Week 15

Example: Video

- 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

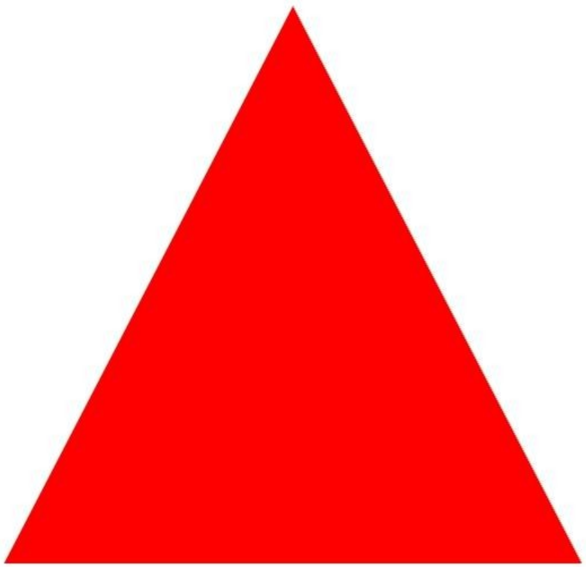
- JavaScript

  contains the graphics rules

WebGL Code

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

Example:
http://www.cs.upt.ro/~sorin/webgl/triangle.html



Example: Code (HTML)

```
<!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>
```

Example: 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>
```

Example: JS File - triangle.js

```
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 )
];
```

Example: 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 );
```

Example: JS File (cont)

```
// Associate our 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 );
}
```

JavaScript

- 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 ???

- 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 ???

- 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

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
```

⇒ Problem for WebGL, which expects C-style arrays

JS: 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

Suggestions

- Use only core JS and HTML
  - no extras or variants
- No additional packages
  - CSS
  - JQuery
- Focus on graphics
  - well, may lack beauty
- Any variants are welcome, as long as I can run them from your URL

What is
Computer Graphics?

Computer Graphics

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

Example

- What's this image?

- What hardware/software was used?

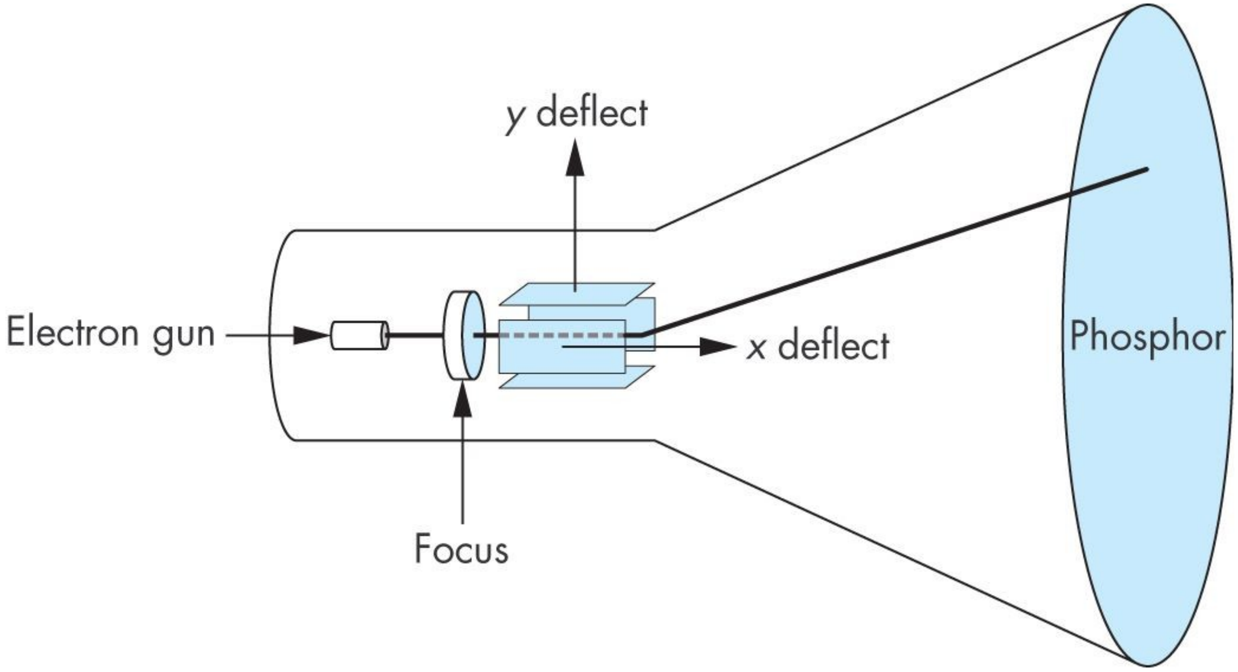
Answer

- **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

Computer Graphics: 1950-1960

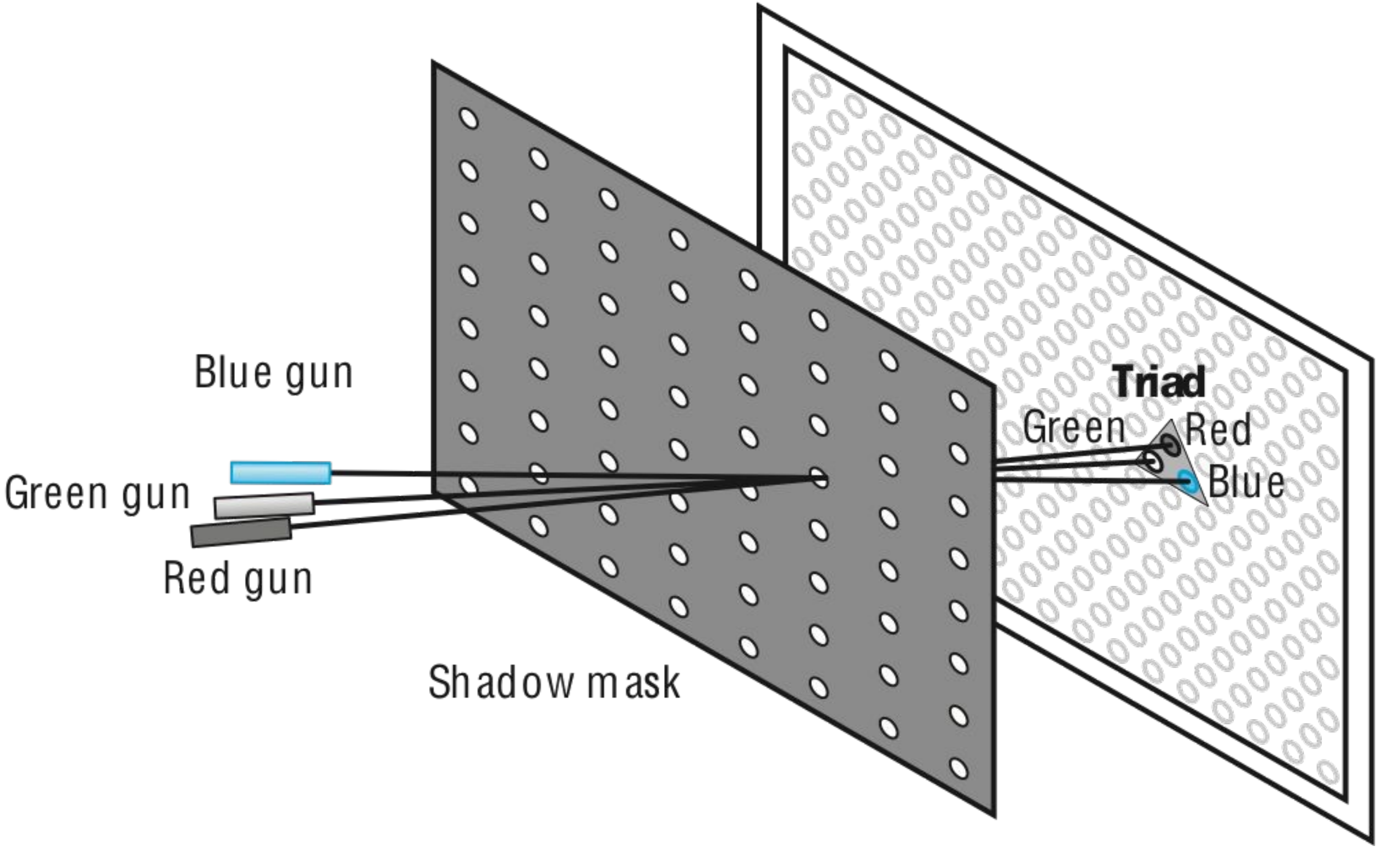
- earliest days of computing
  - Strip charts
  - Pen plotters
  - Simple displays using A/D converters to go from computer to vector displays
- Cost of refresh for CRT too high
  - Computers slow, expensive, unreliable

Cathode Ray Tube (CRT)



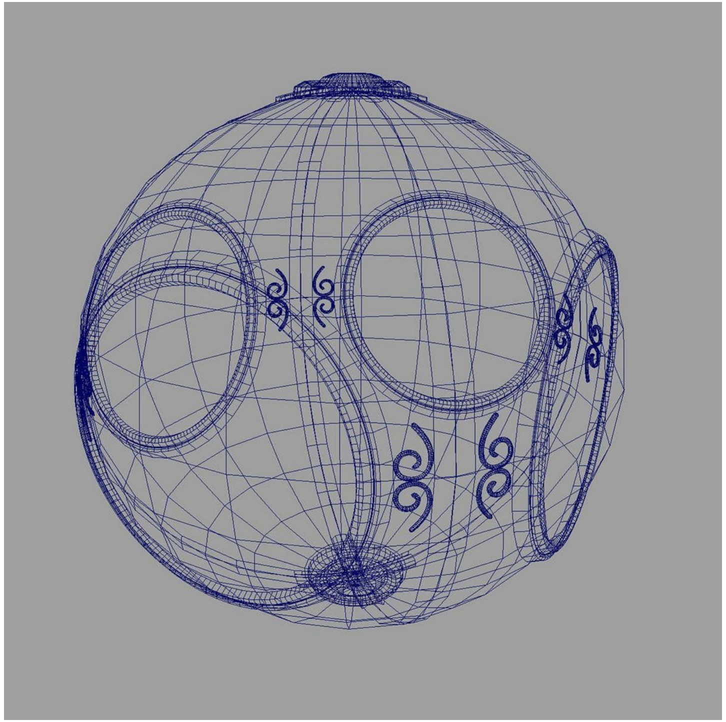
Can be used either as a line-drawing device (vector) or to display contents of frame buffer (raster mode)

Color CRT



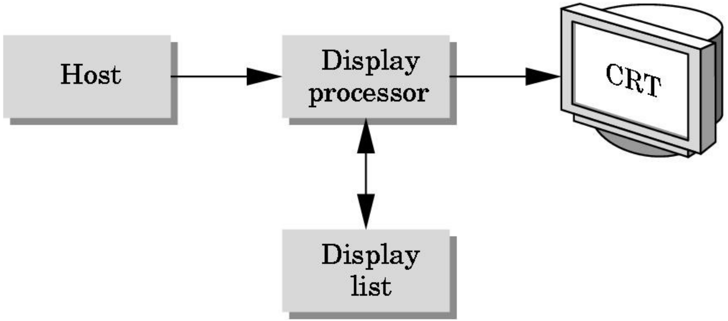
Computer Graphics: 1960-1970

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



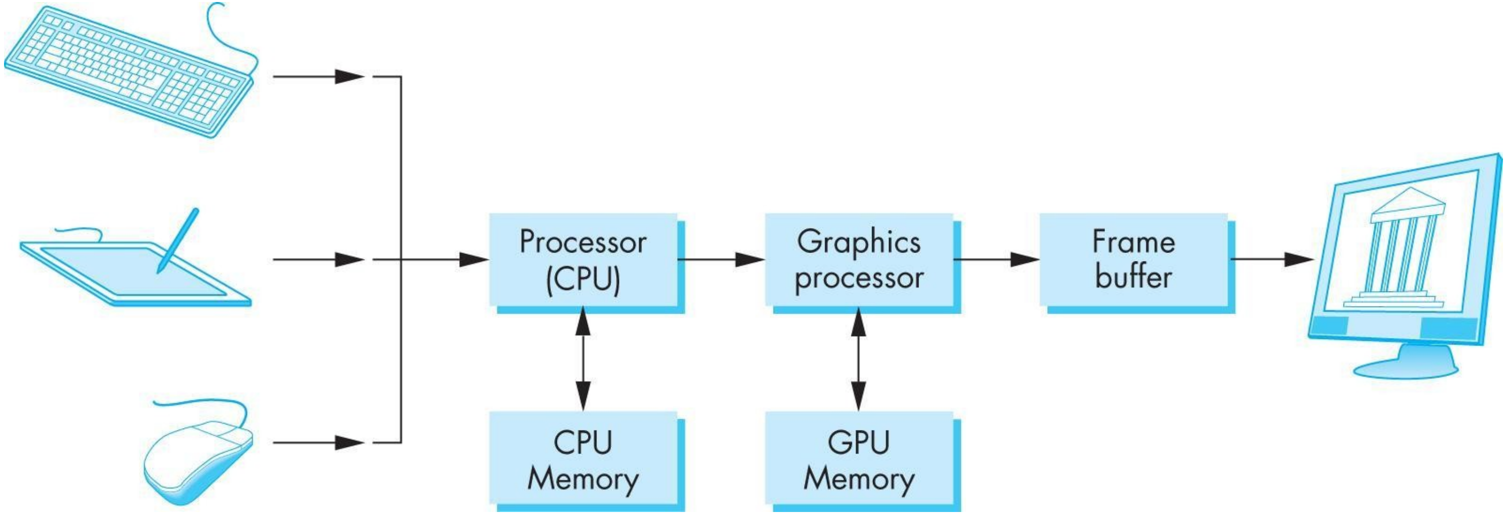
Display Processor

- Use a special purpose computer called a *display processor* (DPU) instead of host computer for refreshing display



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

Basic Graphics System



Computer Graphics: 1970-1980

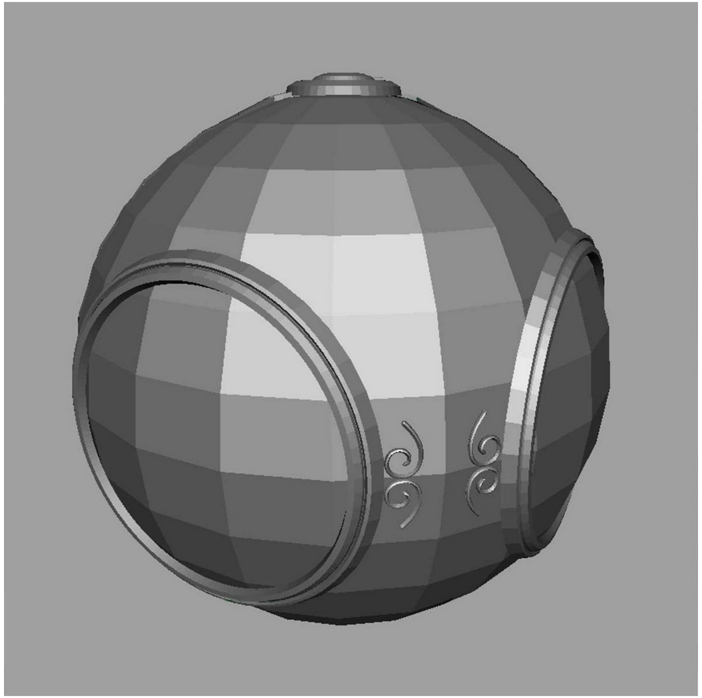
- Raster Graphics
- Beginning of graphics standards
- Workstations and PCs

Raster Graphics

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

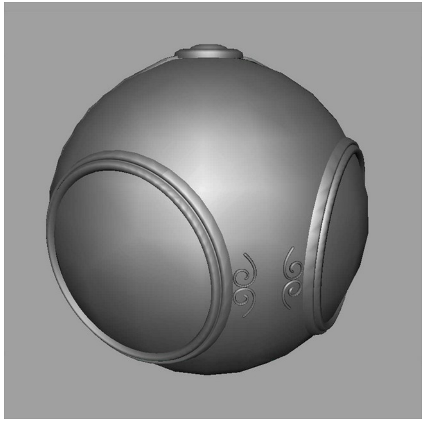
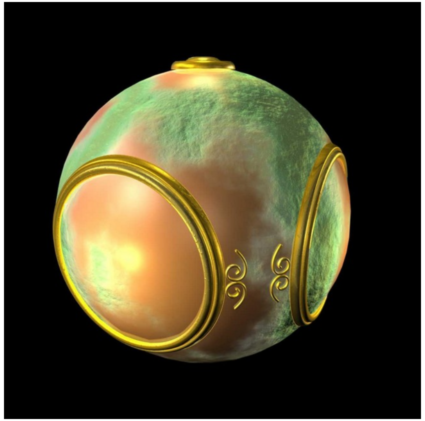
Raster Graphics

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



Computer Graphics: 1980-1990

Realism

		
smooth shading	environment mapping	bump mapping

Computer Graphics: 1980-1990

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

Computer Graphics: 1990-2000

- OpenGL API
- Computer-generated movies (Toy Story)
- New hardware capabilities
  - Texture mapping
  - Blending
  - Accumulation, stencil buffers

Computer Graphics: 2000-2010

- Photorealism
- Market dominated by graphics cards for PCs
    - Nvidia, ATI
- Game boxes
- Movie industry: Maya, Lightwave
- Programmable pipelines
- New display technologies

Computer Graphics 2011-

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

Image Formation

Objectives

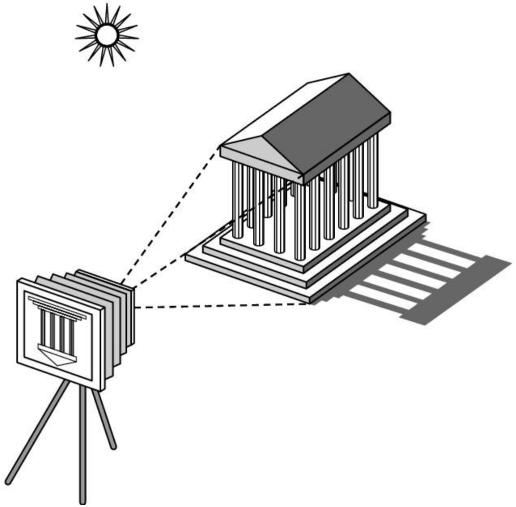
- Fundamental imaging notions
- Physical basis for image formation
  - Light
  - Color
  - Perception
- Synthetic camera model
- Other models

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

- Objects
- Viewer
- Light source(s)



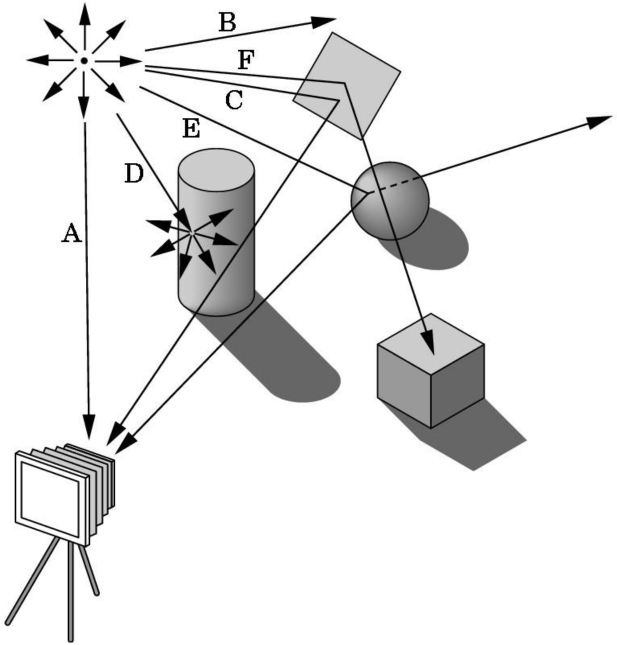
- Attributes that govern how light interacts with the materials in the scene
- Note the independence of the objects, the viewer, and the light source(s)

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

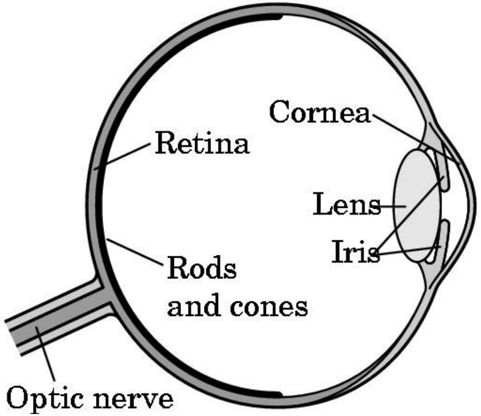
Ray Tracing; Geometric Options

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.



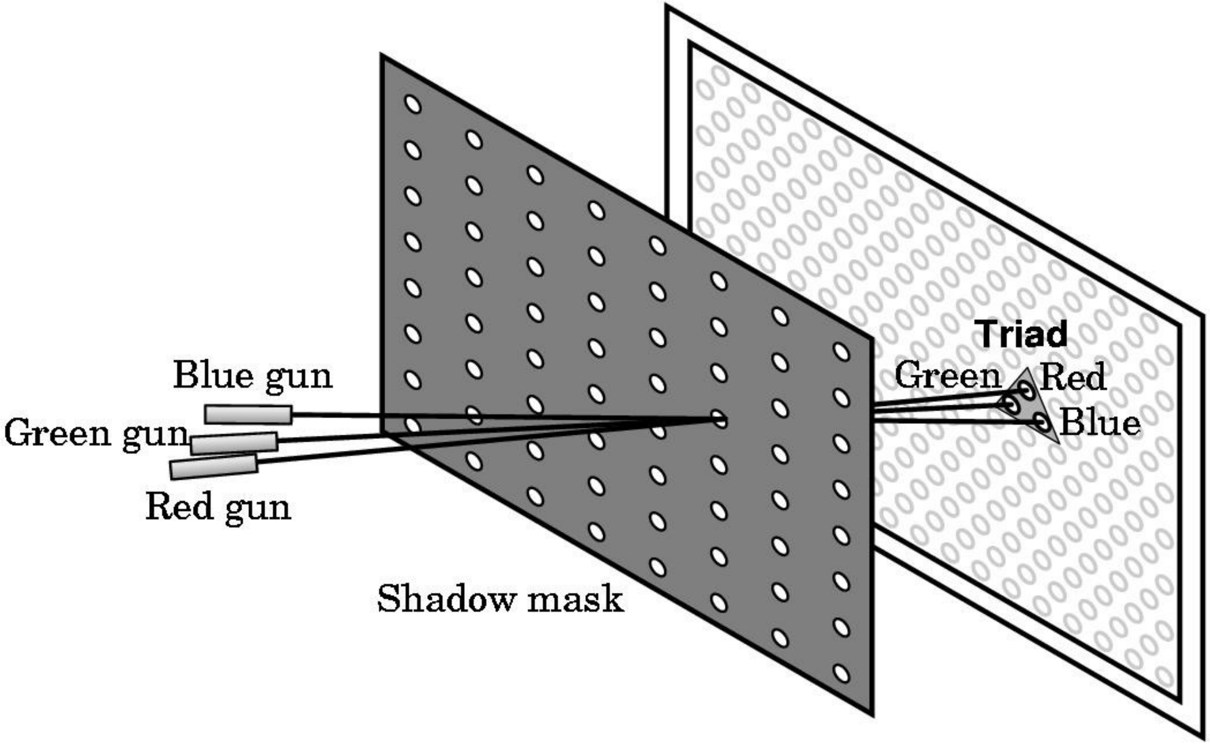
Three-Color Theory

- 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

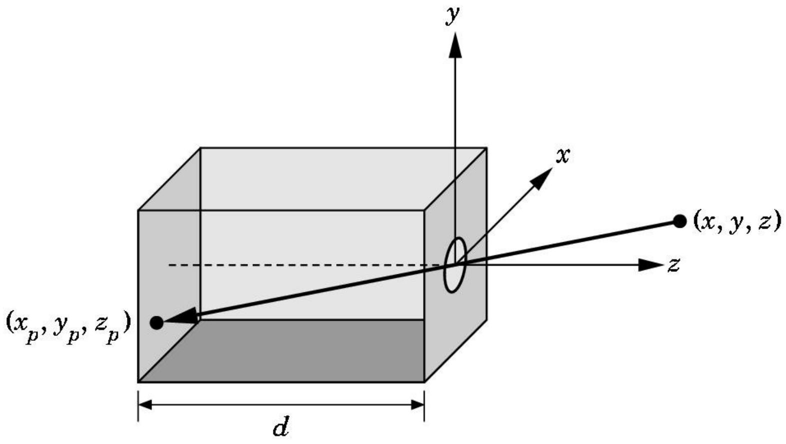


- Need only match these three values
  - -Need only three *primary* colors

Shadow Mask CRT

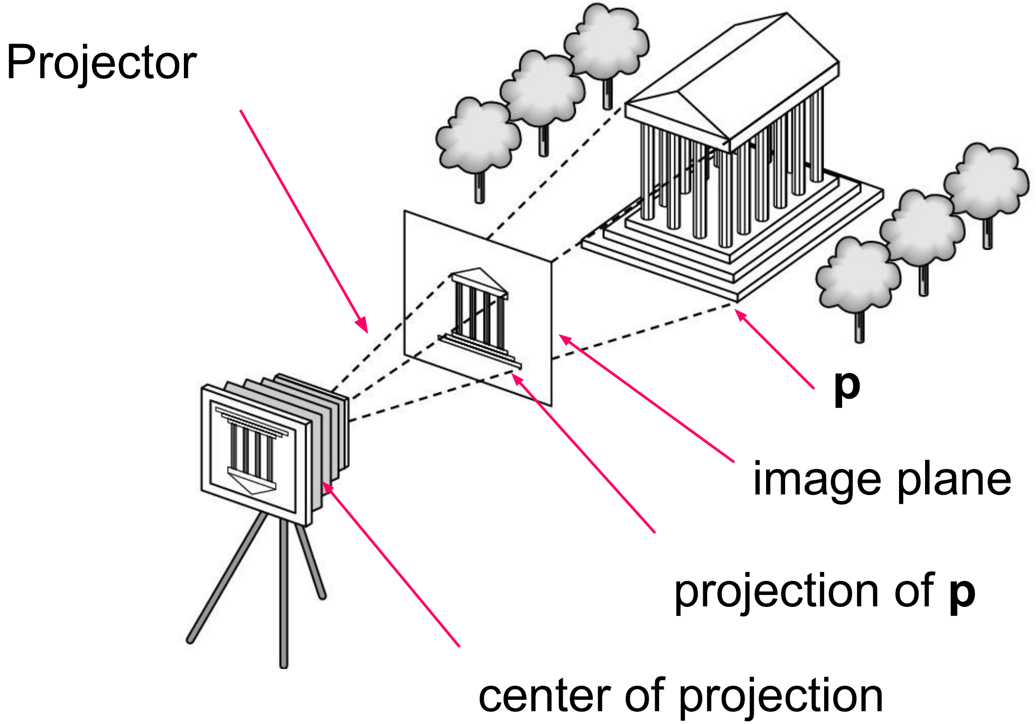


Pinhole Camera



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

Modeling a Synthetic Camera

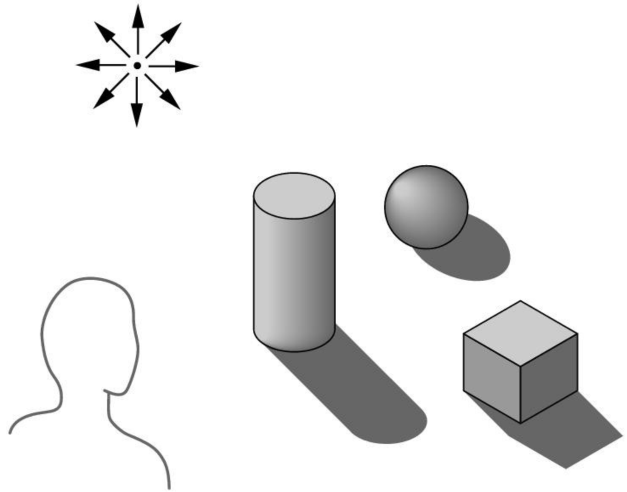


Advantages

- Separation of objects, viewer, light sources
- Two-dimensional graphics is a special case of three-dimensional graphics
- Simple software API
  - Specify objects, lights, camera, attributes
  - Let implementation determine image
- Fast hardware implementation

Global or 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



No 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 -- 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