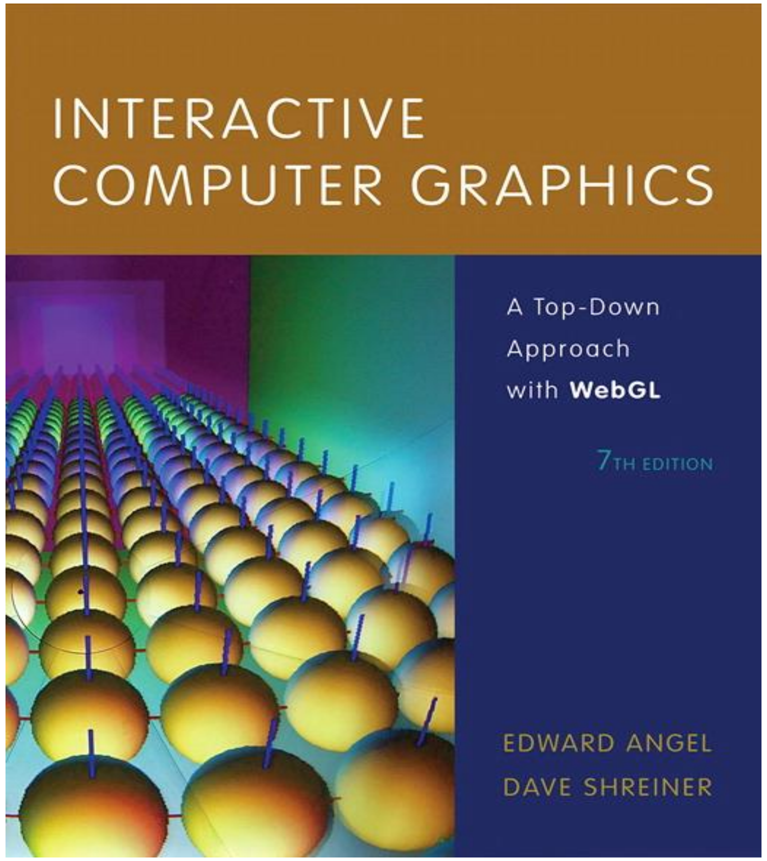
Introduction to Computer Graphics with WebGL

Week1

Instructor: Hooman Salamat

Textbook

- Interactive Computer Graphics by Edward Angel and Dave Shreiner


INTERACTIVE COMPUTER GRAPHICS

A Top-Down Approach with **WebGL**

7TH EDITION

EDWARD ANGEL

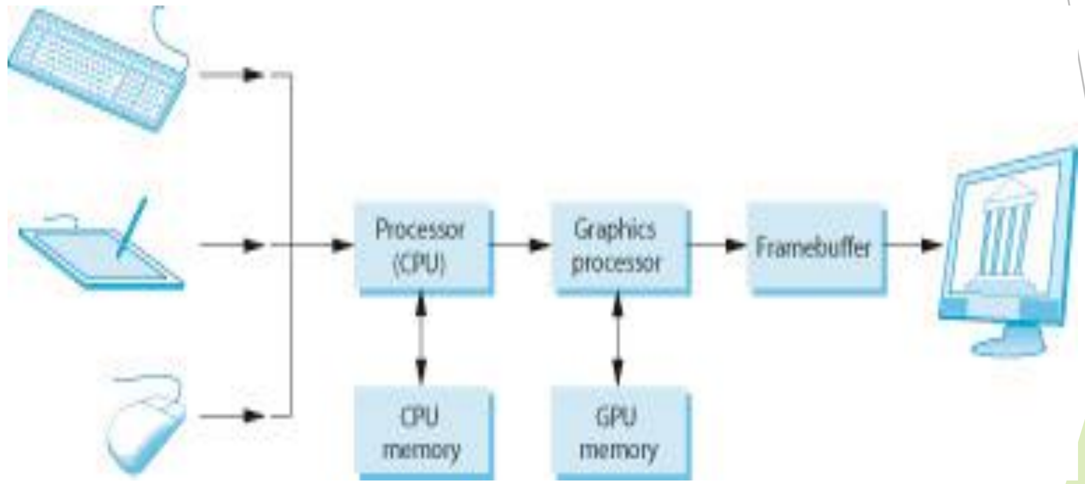
DAVE SHREINER

Syllabus

- Week 1: Introduction and Overview
- Week 2: Introduction to WebGL
- Week 3: GLSL and Shaders
- Week 4: Input and Interaction
- Week 5: Geometry and Transformations
- Week 6: Modeling and Viewing
- Week 7: Projection Matrices and Shadows
- Week 8: Lighting and Shading
- Week 9: Buffers and Texture Mapping
- Week 10: Discrete Techniques
- Week 11: Off-Screen Rendering.
- Week 12: Hierarchy
- Week 13: Implementation
- Week 14: Curves and Surfaces
- Week 15: Global Rendering

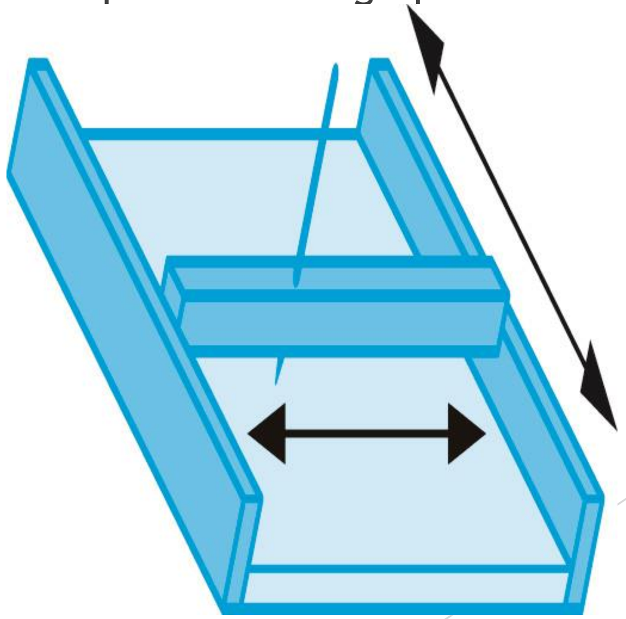
A Graphics System

- The image we see on the output device is an array (**the raster**) of picture elements, or pixels produced by graphics system
- All modern graphics systems are raster based
- Every pixel corresponds to a location in the image
- Collectively, the pixels are stored in a part of memory called the **framebuffer**.
- frame buffer depth or precision
    - 1-bit-deep => Two colors
    - 8-bit deep => 256 colors
    - Full color => 24 bit or more



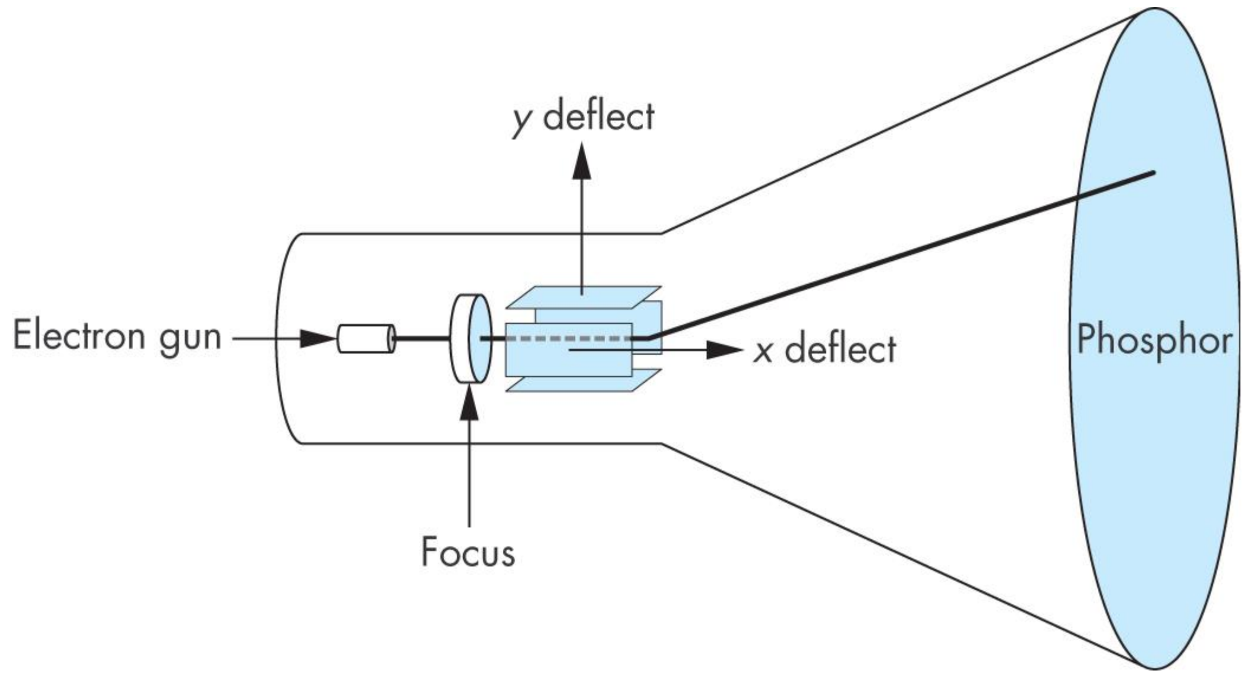
Computer Graphics: 1950-1960

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

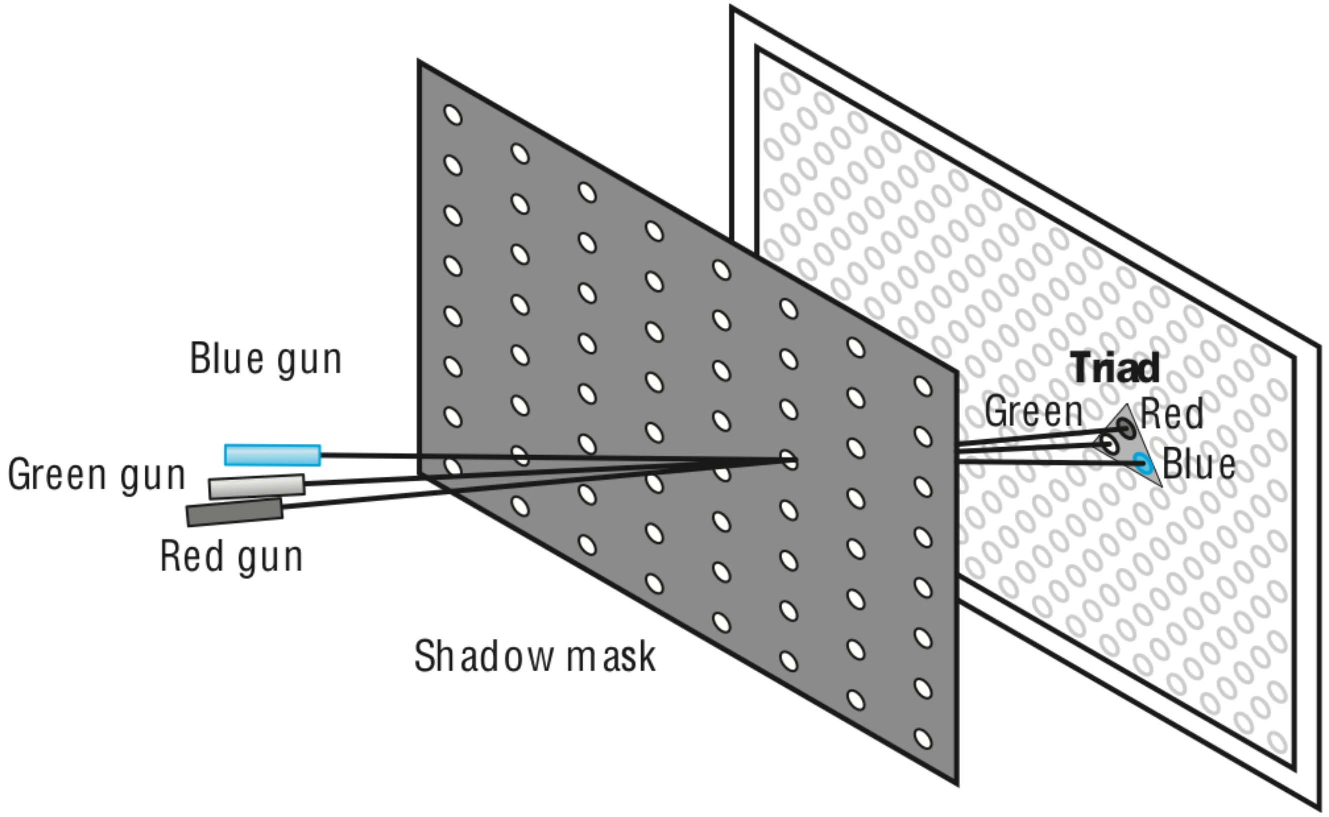


Cathode Ray Tube (CRT)

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

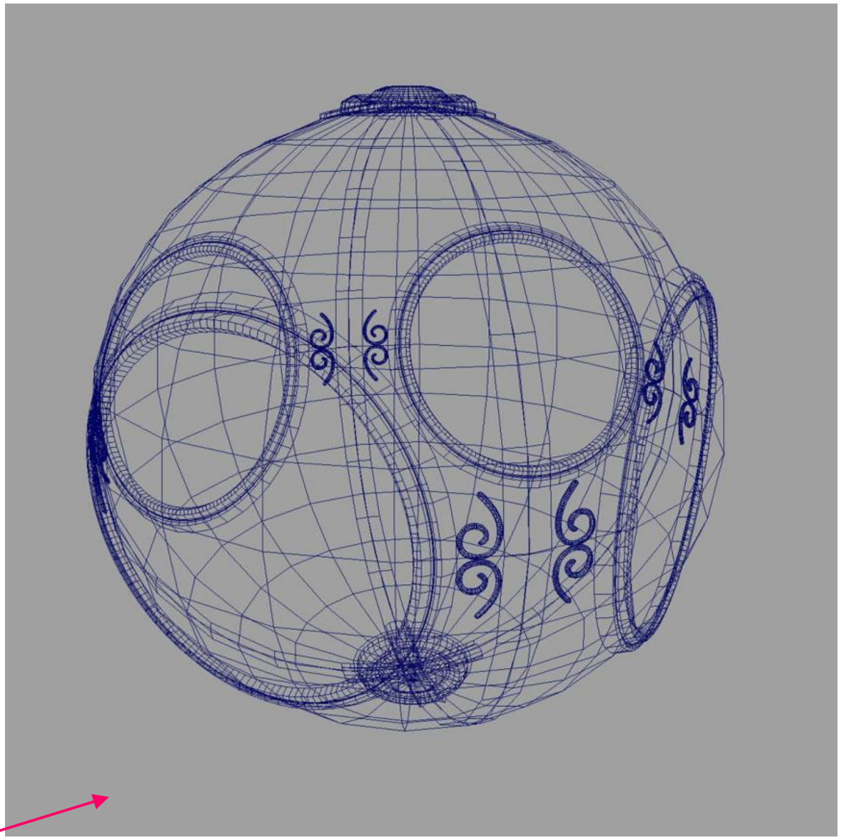


Shadow Mask CRT



Computer Graphics: 1960-1970

- *Wireframe* graphics
  - Draw only lines
- Sketchpad https://sketch.io/sketchpad
- Display Processors
- Storage tube



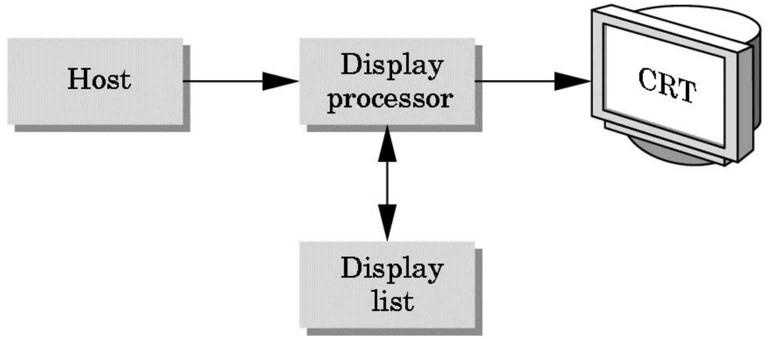
wireframe representation of sun object

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)



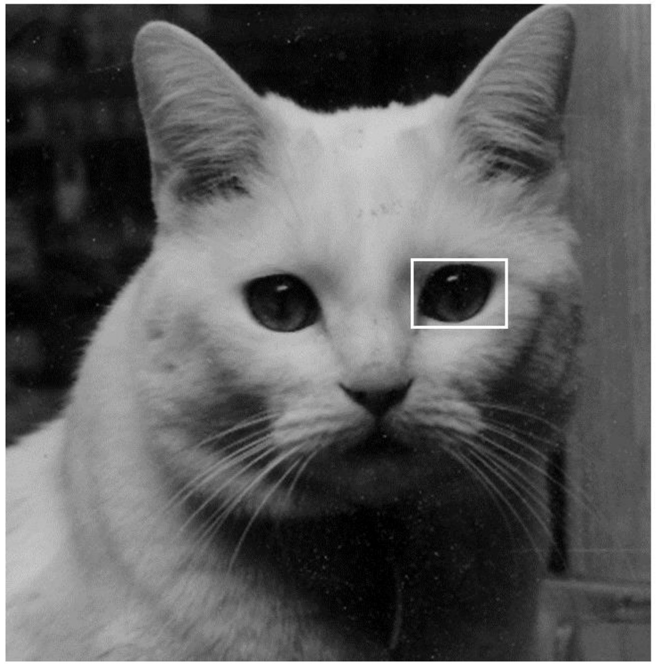
- Graphics stored in display list (display file) on display processor
- Host *compiles* display list and sends to DPU

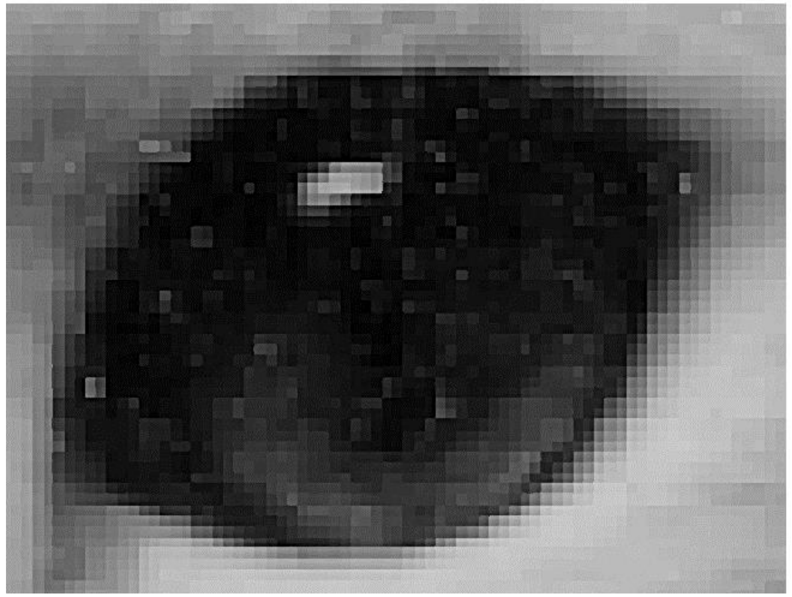
Computer Graphics: 1970-1980

- 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

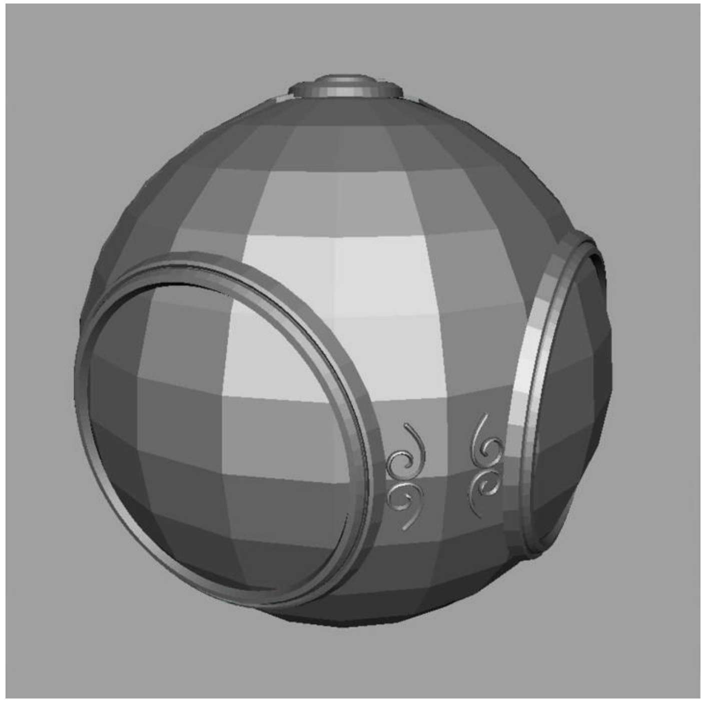
- Image produced as an array (the *raster*) of picture elements (*pixels*) in the *frame buffer*





Raster Graphics

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

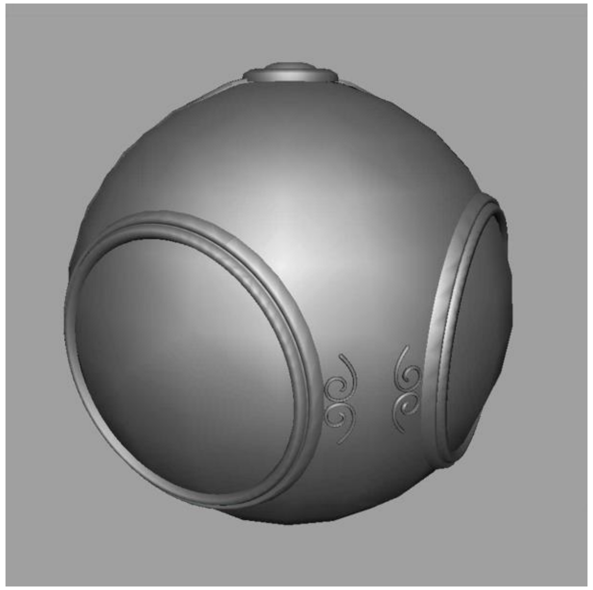


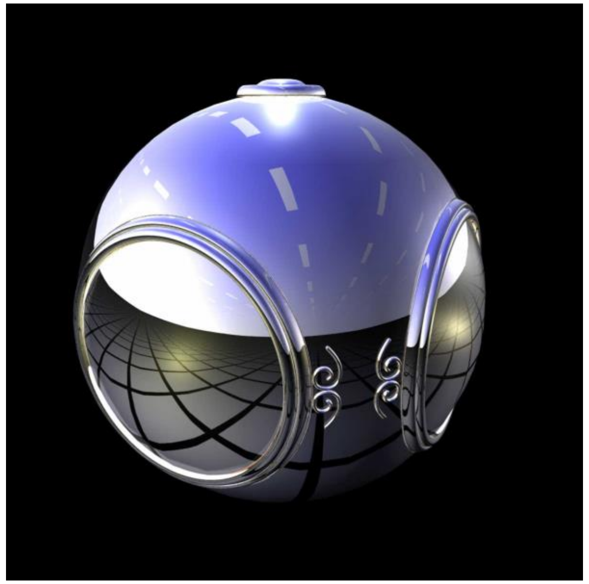
PCs and Workstations

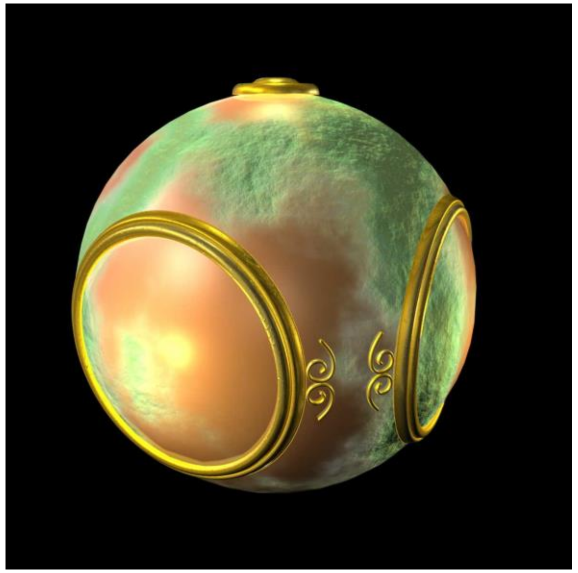
- Although we no longer make the distinction between workstations and PCs, historically they evolved from different roots
  - Early workstations characterized by
    - Networked connection: client-server model
    - High-level of interactivity
  - Early PCs included frame buffer as part of user memory
    - 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 https://renderman.pixar.com/view/renderman
- Networked graphics: X Window System
- Human-Computer Interface (HCI)

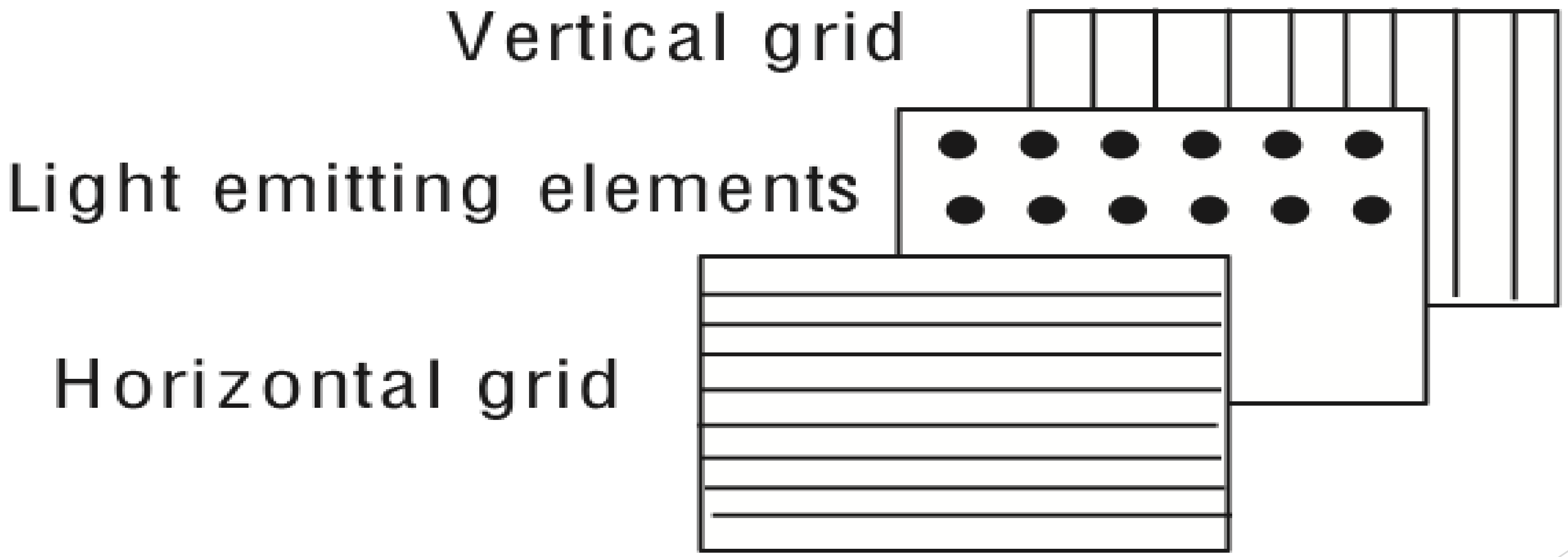
Computer Graphics: 1990-2000

- 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-

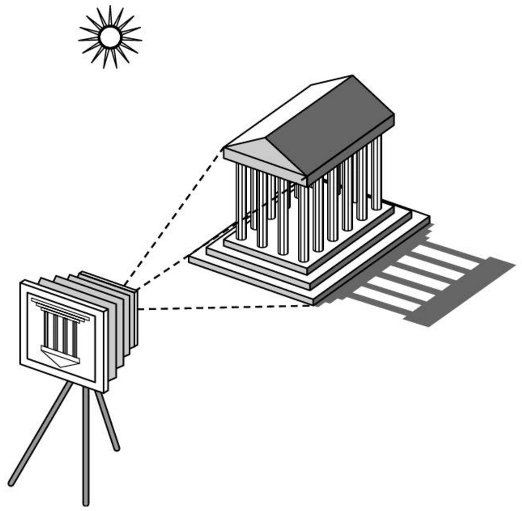
- Graphics is now ubiquitous
  - Cell phones
  - Embedded
- OpenGL ES and WebGL
- Alternate and Enhanced Reality
- 3D Movies and TV

Image Formation

- 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

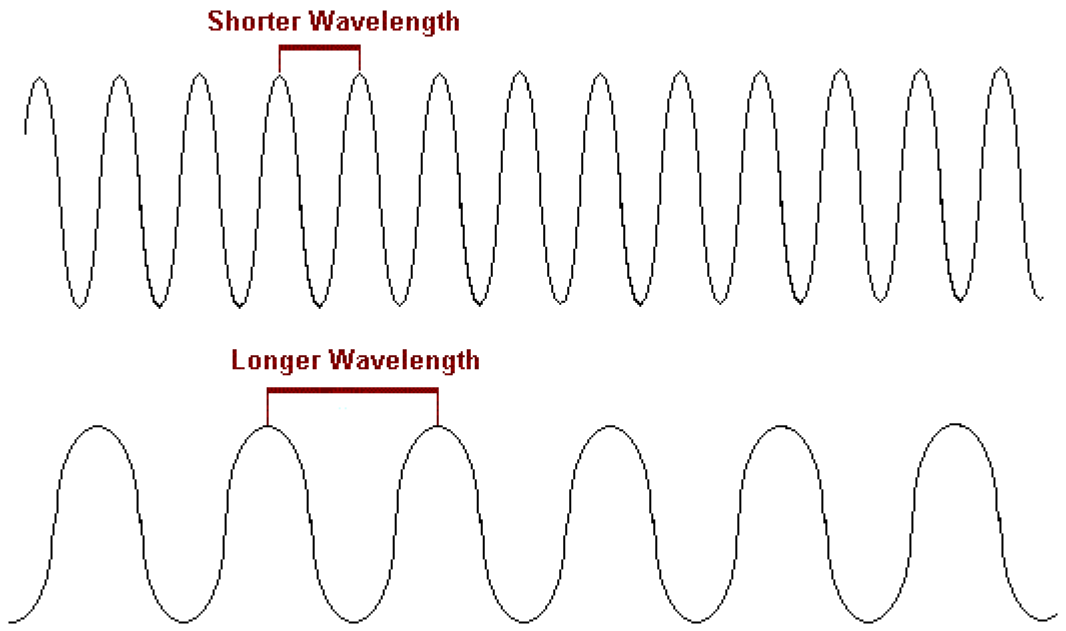
Elements of Image Formation

- 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)



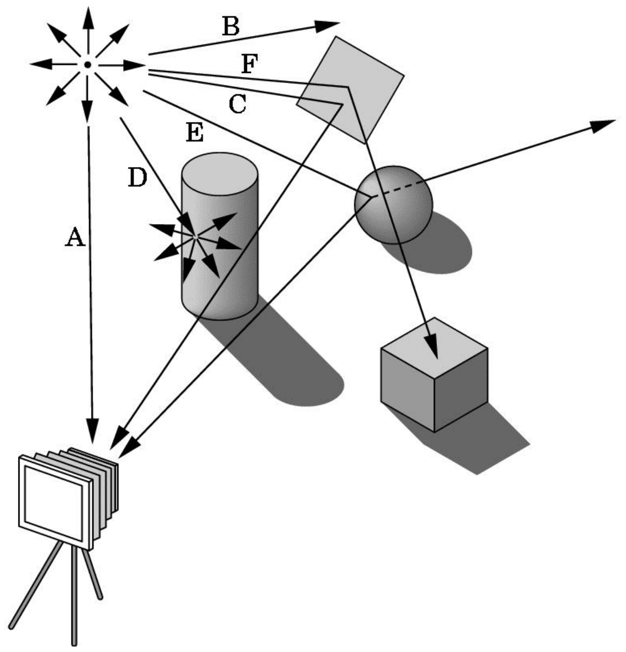
Light

- *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.

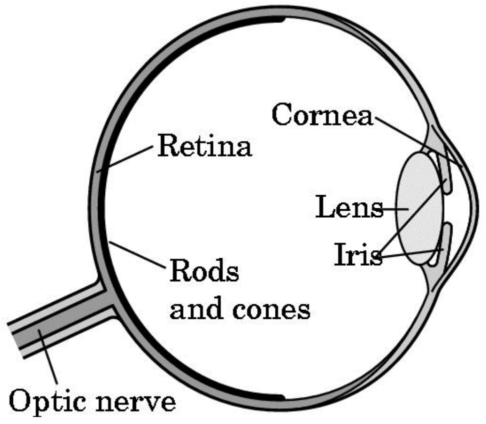


Luminance and Color Images

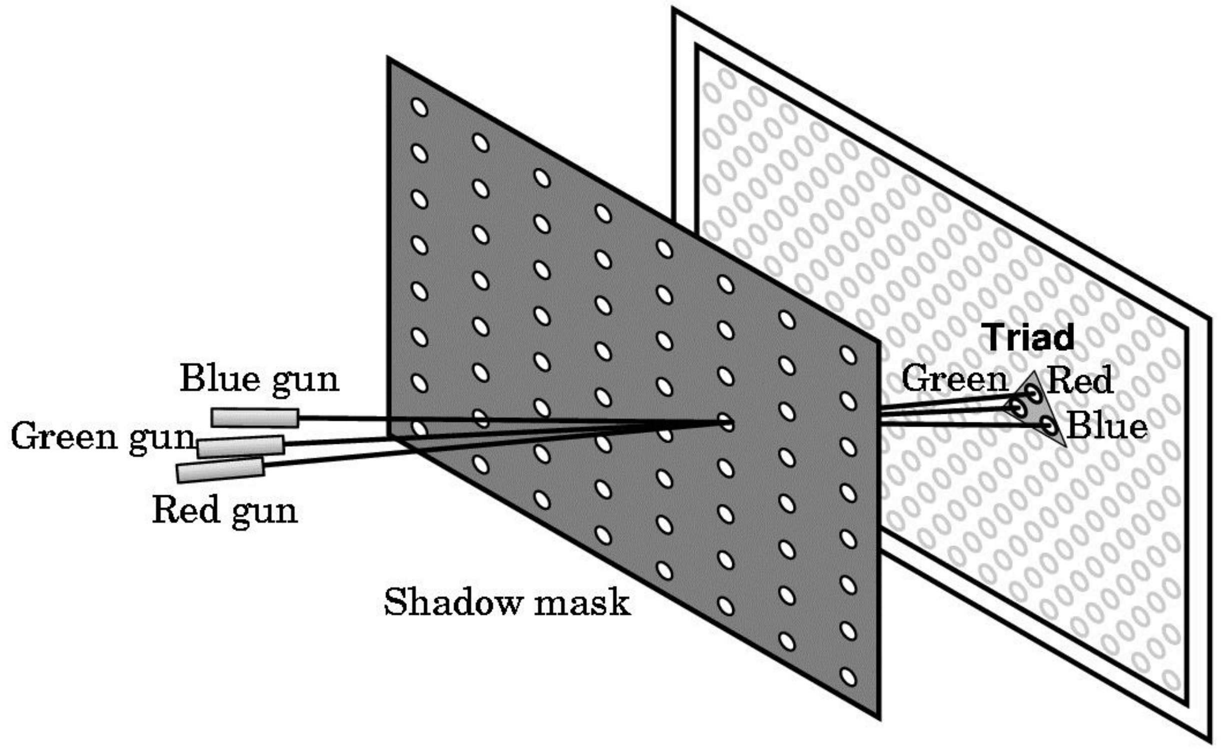
- Luminance Image
  - Monochromatic
  - Values are gray levels
  - Analogous to working with black and white film or television
- Color Image
  - Has perceptional attributes of hue, saturation, and lightness
  - Do we have to match every frequency in visible spectrum? No!

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



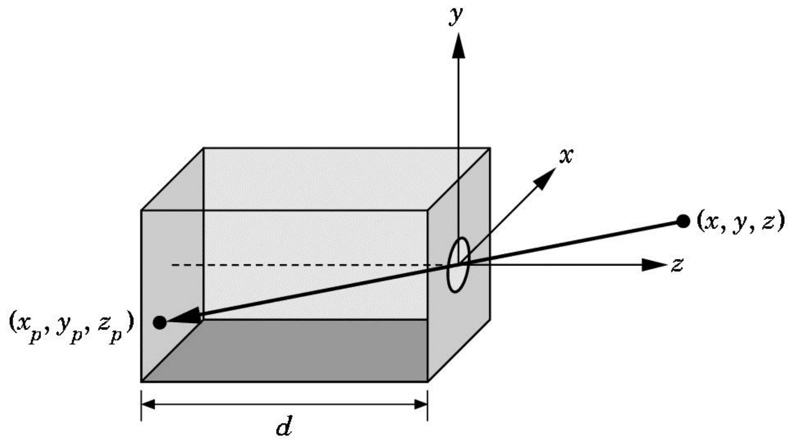
Shadow Mask CRT



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

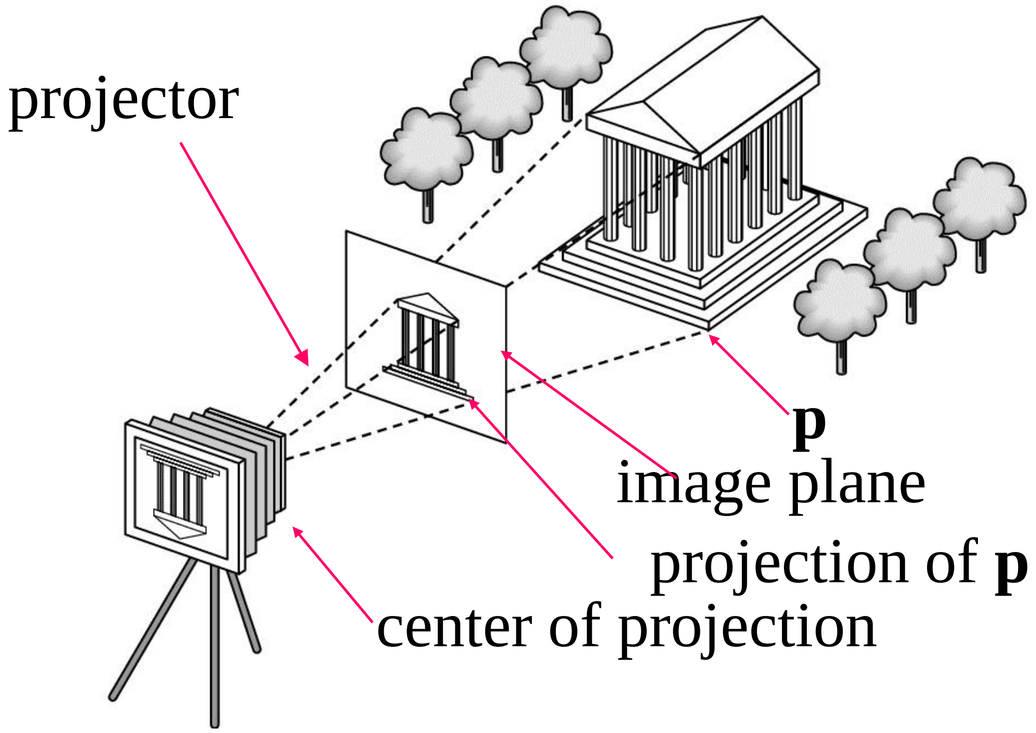


Use trigonometry to find projection of point at (x,y,z)

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

These are equations of simple perspective

Synthetic Camera Model

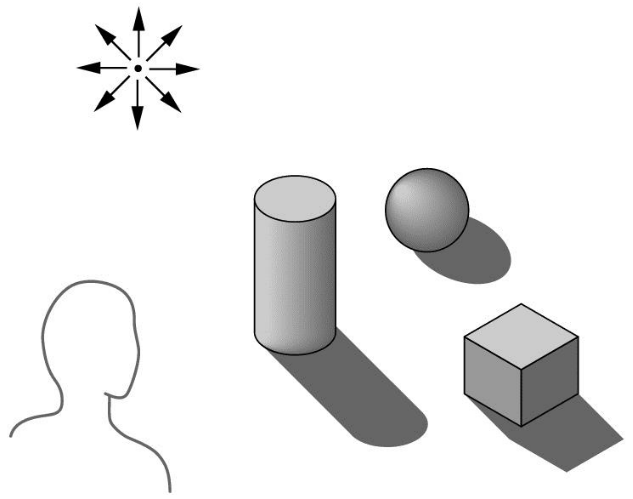


Advantages

- 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

OpenGL

- Every computer has special graphics hardware that controls what you see on the screen.
- OpenGL tells this hardware what to do.
- The Open Graphics Library is one of the oldest, most popular graphics libraries game creators have.
- It was developed in 1992 by Silicon Graphics Inc. (SGI) and used for GLQuake in 1997.
- The GameCube,Wii, PlayStation, and the iPhone all use OpenGL.
- Cross-platform standard: OpenGL today is supported on all platforms

OpenGL

- OpenGL is a C-style graphics library with no classes or objects.
- OpenGL is basically a large collection of functions.
- Internally, OpenGL is a state machine.
- Function calls alter the internal state of OpenGL, which then affects how it behaves and how it renders objects to the screen.
- Because it is a state machine, it is very important to carefully note which states are being changed.

Microsoft DirectX

- The alternative to OpenGL is Microsoft’s DirectX.
- DirectX holds a larger number of libraries, including sound and input.
- OpenGL is almost like Direct3D library in DirectX.
- The latest version of DirectX is DirectX 12.
- DirectX 12 was shipped with Windows 10.
- The Xbox one uses a version of DirectX 12.0.

OpenGL ES

- OpenGL ES is a modern version of OpenGL for embedded systems such smart phones, tablets, and video game consoles.
- It is quite similar to recent versions of OpenGL but with a more restricted set of features. It is used on high-end mobile phones such as the Android, BlackBerry, and iPhone.
- It is also used in military hardware for heads-up displays on things like warplanes.
- OpenGL ES supports the programmable pipeline and has support for shaders.
- WebGL 2.0 is a JS implementation of ES 3.0

WebGL

- WebGL is an application programming interface (API) for advanced graphics on the web.
- WebGL 2 is based on OpenGL ES 3.0 and provides an API for 3D graphics.
- The rendering surface that is used for WebGL is the HTML 5 canvas element, which was originally introduced by Apple in the WebKit open-source browser.
- Apple was using HTML canvas to render 2D graphics such as Dashboard widgets and Safari browser.
- Vladimir Vukicevic at Mozzila started experimenting with 3D graphics for the canvas element. The initial prototype was called Canvas 3D.
- In 2009, Khronos group (founded in Jan. 2000) started WebGL working group.
- Code written in JavaScript. Automatic memory management is provided as part it.
- Early applications of WebGL include Zygote Body.
- In November 2012 Autodesk announced that they ported most of their applications to the cloud running on local WebGL clients. These applications included Fusion 360 and AutoCAD 360.

Why is WebGL important?

- Web
  - With HTML5, WebGL runs in the latest browsers
    - Chrome
    - Firefox
    - Safari
    - WebGL is partially supported in IE 11.
- make use of the full capabilities of the graphics processing unit (GPU)
- no platform dependencies (runs on browsers)
- With WebGL, you get hardware-accelerated 3D graphics inside the browser.

Designing a Graphics API

- Using an immediate-mode API. The whole scene needs to be redrawn on every frame, regardless of whether it has changed. Graphics Library doesn’t save any internal model of the scene that should be drawn. Application keeps track of the model of the scene and doing initialization.
- WebGL is an immediate-mode API.
- Using a retained-mode API. Graphics Library contains an internal model or scene graph with all the objects that should be rendered. Application doesn’t issue drawing commands to draw the complete scene on every frame.
- SVG (Scalable Vector Graphics) is a retained-mode API.

Three.js

- JavaScript 3D engine
- Object oriented scene graph
  - Scene = camera + objects + lights + materials
- Renders with WebGL
- Makes use of GPU but hides details of rendering and modeling.
- Cube is one of the geometries built into three.js.
- three.js includes libraries with interactive controls.
- The standard lighting models are included with three.js.
- With WebGL we can build our lighting models in a varietyof ways including within the shaders.
- Both three.js and WebGL support texture mapping using either images available in standard formats (gif,jpeg) or images defined in the code.

Vertex

- The basic unit in OpenGL is the vertex. A vertex is a point in space.
- Extra information can be attached to these points—how it maps to a texture, if it has a certain weight or color—but the most important piece of information is its position.

- Games spend a lot of their time sending OpenGL vertices or telling OpenGL to move vertices in certain ways.
- The game may first tell OpenGL that all the vertices it’s going to send are to be made into triangles. In this case, for every three vertices OpenGL receives, it will attach them together with lines to create a polygon, and it may then fill in the surface with a texture or color.

Pipeline

- Modern graphics hardware is very good at processing vast sums of vertices, making polygons from them and rendering them to the screen.
- This process of going from vertex to screen is called the pipeline.
- The pipeline is responsible for positioning and lighting the vertices, as well as the projection transformation.
- This takes the 3D data and transforms it to 2D data so that it can be displayed on your screen.

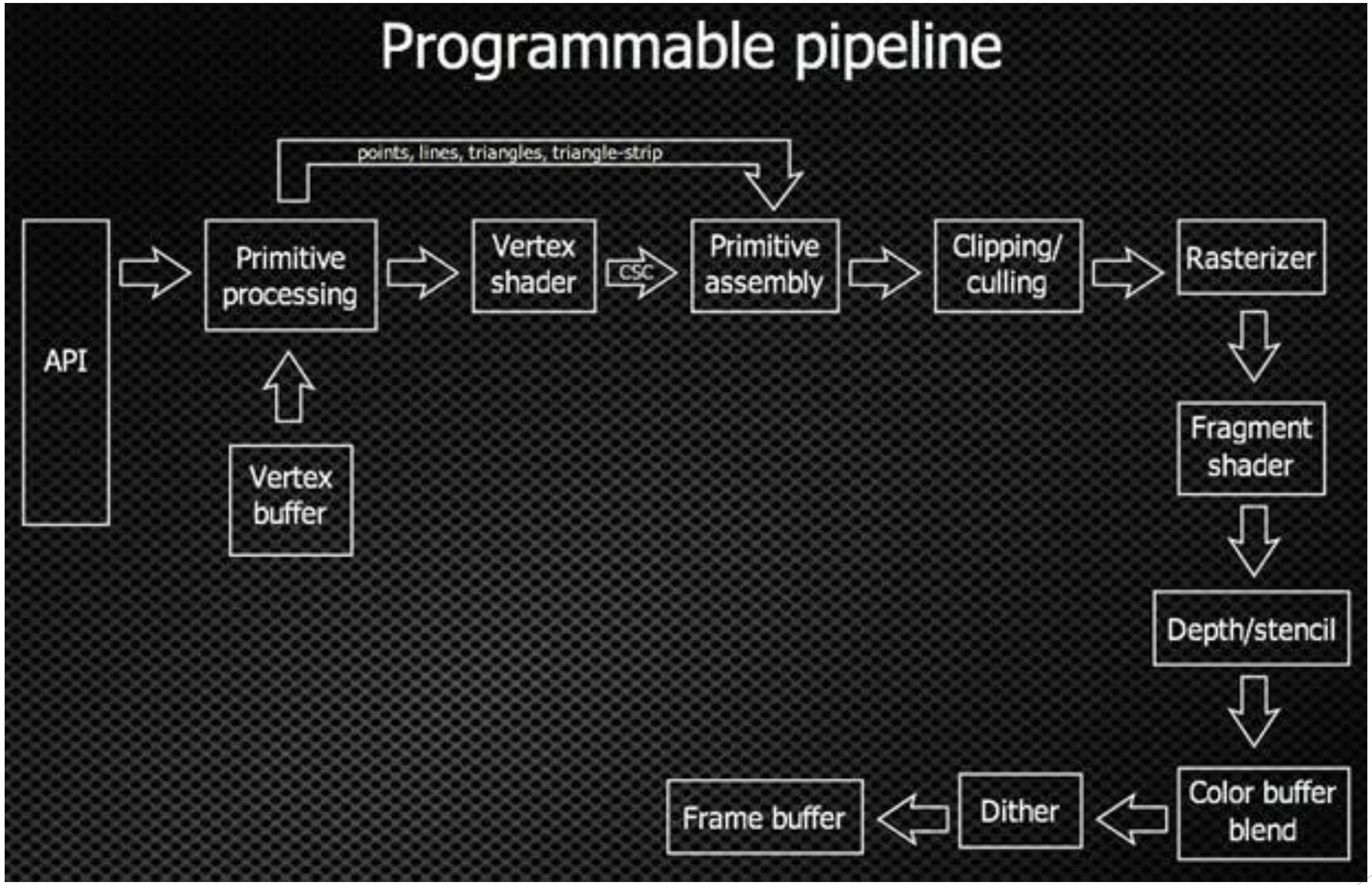
Sprite

- Even modern 2D games are made using vertices.
- The 2D sprites are made up of two triangles to form a square.
- This is often referred to as a quad.
- The quad is given a texture and it becomes a sprite.
- Two-dimensional games use a special projection transformation that ignores all the 3D data in the vertices, as it’s not required for a 2D game.
- Sprites are 2D bitmaps that are drawn directly to a render target without using the pipeline for transformations, lighting or effects. Sprites are commonly used to display information such as health bars, number of lives, or text such as scores. Some games, especially older games, are composed entirely of sprites.

Pipeline

- The pipeline has become programmable.
- Programs can be uploaded to the graphics card.
- There are few steps in the pipeline.
- All the steps could be applied in parallel.
- Each vertex can pass through a particular stage at the same time, provided the hardware supports it.
- This is what makes graphics cards so much faster than CPU processing.

Pipeline


Programmable pipeline
points, lines, triangles, triangle-strip
API
Primitive processing
Vertex shader
CSC
Primitive assembly
Clipping/ culling
Rasterizer
Vertex buffer
Fragment shader
Depth/stencil
Color buffer blend
Dither
Frame buffer

Pipeline Steps

- 1.Input stage: The CPU sends instructions (compiled shading language programs) and geometry data (all the vertices' properties including position, color, and texture data) to the graphics processing unit, located on the graphics card.
- 2.Vertex shading: Shaders are simple programs that describe the traits of either a vertex or a pixel. Vertex shaders describe the traits (position, texture coordinates, colors, etc.) of a vertex, while pixel shaders describe the traits (color, z-depth and alpha value) of a pixel. A vertex shader is called for each vertex in a primitive (possibly after tessellation); thus one vertex in, one (updated) vertex out. Each vertex is then rendered as a series of pixels onto a surface (block of memory) that will eventually be sent to the screen.
- 3.Geometry shading: Geometry shaders were added a little more recently than pixel and vertex shaders. Geometry shaders take a whole primitive (such as a strip of lines, points, or triangles) as input and the vertex shader is run on every primitive. Geometry shaders can create new vertex information, points, lines, and even primitives. Geometry shaders are used to create point sprites and dynamic tessellation, among other effects. Point sprites are a way of quickly rendering a lot of sprites; it's a technique that is often used for particle systems to create effects like fire and smoke. Dynamic tessellation is a way to add more polygons to a piece of geometry. This can be used to increase the smoothness of low polygon game models or add more detail as the camera zooms in on the model.
- 4.Tessellation shading: If a tessellation shader is in the graphic processing unit and active, the geometries in the scene can be subdivided.

Pipeline Steps

- 5.Primitive setup: This is the process of creating the polygons from the vertex information. This involves connecting the vertices together according to the OpenGL states. Games most commonly use triangles or triangle strips as their primitives.
- 6.Rasterization: The conversion of geometric entities to pixel colors and locations in the framebuffer is know as rasterization or scan coversion.
- 7.Pixel shading: Pixel shaders are applied to each pixel that is sent to the frame buffer. Pixel shaders are used to create bump mapping effects, specular highlights, and per-pixel lighting. Bump map effects is a method to give a surface some extra height information. Bump maps usually consist of an image where each pixel represents a normal vector that describes how the surface is to be perturbed. The pixel shader can use a bump map to give a model a more interesting texture that appears to have some depth. Per-pixel lighting is a replacement for OpenGL's default lighting equations that work out the light for each vertex and then give that vertex an appropriate color. Per-pixel uses a more accurate lighting model in which each pixel is lit independently. Specular highlights are areas on the model that are very shiny and reflect a lot of light.
- 8.Frame buffer: The frame buffer is a piece of memory that represents what will be displayed to the screen for this particular frame. Blend settings decide how the pixels will be blended with what has already been drawn. The framebuffer is on the same circuit board as GPU and accessed thru GPU. The GPU can be located on the motherboard of the system or on a graphics card. Recent GPU contain over 100 processing units, each of which is user programmable.

HTML5 Canvas

- HTML5 canvas is a rectangular area of your web page where you can draw graphics by using JavaScript.
- WebGL is designed as a rendering context for the HTML5 canvas element.
- The original 2D rendering context (CanvasRenderingContext2D interface)
  - var context2D = canvas.GetContext("2D");
- A WebGL rendering context (WebGLRenderingContext interface) can be retrieved from the canvas element in the same way
  - var contextWebGL = canvas.getContext("webgl");

An HTML5 Canvas Code Snippet

```
<!DOCTYPE HTML>
<html lang="en">
<head>
    <meta charset="utf-8">
    <script type="text/javascript">
        function draw() {
            var canvas = document.getElementById("canvas");
            if (canvas.getContext) {
                var context2D = canvas.getContext("2d");
                // Draw a red rectangle
                context2D.fillStyle = "rgb(255,0,0)";
                context2D.fillRect(20, 20, 80, 80); }   }
    </script>
</head>
<body onload="draw();">
    <canvas id="canvas" width="300" height="300">  Your browser does not support the HTML5 canvas element.
    </canvas>
</body>
</html>
```

Scalable Vector Graphics

- Scalable Vector Graphics (SVG) is language used to describe 2D graphics with XML. It’s based on vector graphics, which means it uses geometrical primitives such as point, lines and curves that are stored as mathematical expressions.
- In 1998, Microsoft, Macromedia, and some other companies submitted Vector Markup Language (VML) as a proposed standard to W3C.
- Adobe and Sun created another proposal called Precision Graphics Markup Language (PGML)
- W3C took a little bit of VML and a little bit of PGML and created SVG in 2001.
- SVG Mobile Recommendation included two simplified profiles for mobile phones (SVG Tiny for simple mobile devise, SVG Basic targets higher level mobile devices).

An SVG Code Snippet

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="100%" height="100%" version="1.1“ xmlns="http://www.w3.org/2000/svg">
<rect x="50" y="30" width="300" height="100" fill="red" stroke-width="2“ stroke="black"/>
<circle cx="100" cy="200" r="40" stroke="black“ stroke-width="2" fill="blue"/>
<polygon points="200,200  300,200  250,100" fill="green" stroke-width="2" stroke="black"  />
</svg>
```

VRML and X3D

- Virtual Reality Markup Language (VRML) and its successor X3D are both XML based technologies to describe 3D graphics.

```
<x3d width='500px' height='400px'>
     <scene>
       <shape>
         <appearance>
           <material diffuseColor='1 0 0'></material>
         </appearance>
         <box></box>
       </shape>
       <transform translation='-3 0 0'>
         <shape>
           <appearance>
             <material diffuseColor='0 1 0'></material>
           </appearance>
           <cone></cone>
         </shape>
       </transform>
     </scene>
   </x3d>
```

A very simple WebGL output

```
<!DOCTYPE html>
<html>
    <canvas id='c'></canvas>
    <script>
        var c = document.getElementById('c');
        var gl = c.getContext('experimental-webgl');
        gl.clearColor(0,0,0.8,1);
        gl.clear(gl.COLOR_BUFFER_BIT);
    </script>
</html>
```

Draw a Triangle using WebGL

- 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

Setting up a basic WebGL

1. Write some basic HTML code that includes a <canvas> tag. The <canvas> provides a drawing area for WebGL. Then you need to write some JavaScript code to create a reference to your canvas so you can create a WebGLRenderingContext.
2. Writing the source code for your vertex shader and your fragment shader.
3. Write source code that uses WebGL API to create a shader object for both the vertex shader and fragment shader. You need to load the source code into the shader objects and compile the shader objects.
4. Create a program object and attach the compiled shader objects to this program object. After this you can link the program object and then tell WebGL that you want to use this program object for rendering.
5. Set up the WebGL bugger objects and load the vertex data for your geometry (in this case, the triangle) into the buffer.
6. Tell WebGL which buffer you want to connect to which attribute in the shader, and then, finally, draw your geometry.

triangle.html File

```
<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>
```

GLSL

- WebGL requires every application to provide two shaders written in OpenGL Shading Language (GLSL). GLSL is similar to C but add vectors and matrices as basic types.

```
<!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>
```

triangle.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 )
];
//  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 );
// 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 );
}
```

Coordinate System

- A Coordinate System is sometimes called a space.
- WebGL uses a three dimensional, orthonormal, right-handed coordinate system.
  - Use your right hand and assign x-, y-, and z-axes to your thumb, index finger, and middle finger.
  - Axes are orthogonal (perpendicular) to each other.

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

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
- To handle binary data, the typed arrays specification has a concept of a buffer and one or several views of the buffer.
- The buffer is a fixed-length binary data buffer that is represented by the type ArrayBuffer..
- var buffer = new ArrayBuffer(8) creates an 8 byte buffer.
- There are several different views that you can create of an ArrayBuffer.

  var a = new Float32Array(buffer) ➔ two 4 bytes

  var b = new Uint16Array(buffer) ➔ 4 2 bytes

  var c = new Uint8Array(buffer) ➔ 8 1-byte

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

Array Buffer

```
 var bufferId = gl.createBuffer();
 gl.bindBuffer( gl.ARRAY_BUFFER, bufferId );
var vertices = [
         0.0, 0.5,  0.0,
        -0.5, -0.5, 0.0,
         0.5, -0.5, 0.0
];
 gl.bufferData( gl.ARRAY_BUFFER, new Float32Array(vertices),gl.STATIC_DRAW );
```

- The constructor for Float32Array creates a new ArrayBuffer
- The corresponding array buffer is 9x4=36 bytes

Using Chrome Developer Tool

- At the top-right corner of your browser window, you can select wrench menu and then select Tools → Developer Tools.
- On Windows and Linux, you can use the handy short cut Ctrl+Shift+I. The corresponding shortcut on a Mac is Command+Option+I.
- The Chrome Developer Tools consists of: Elements, Resources, Network, Scripts, Timeline, Profiles, Audits, Console.
- The Elements panel gives you an overview of the DOM tree.
- The Sources panel let's you look at all the resources that a web application consists of. http://webglsamples.org/dynamic-cubemap/dynamic-cubemap.html (developed by Gregg Tavares at Google)
- The Network panel let's you investigate which resources are downloaded over the network.
- The Sources panel provides a powerful debugger for your JavaScript.
- The Timeline panel gives you an overview of different events that happen while you load your web application.
- The performance panel helps you profile CPU and memory usage to find which functions take up the most time.
- The Audits panel gives you tips about what you can improve, such as you combine your JavaScript files or improve caching.
- The Console panel is special in that it's available as a separate panel, but you can also bring it up in any of other panels by pressing the Escape key. Here you will find logs that are printed with console.log() in the JavaScript code. You will also be notified about syntax and runtime errors in the JavaScript code.

Web Tracing Framework

- https://google.github.io/tracing-framework
- The tracing framework is a suite of tools and libraries, and there are many different ways to obtain each. The easiest way to get started is to install the Chrome Extension (on Mac/Linux) and see what it records on your page.
- chrome://flags/
    - Check out what experiments have been enabled on your chrome browser

WebGL Error Handling

- When WebGL detects an error, it generates an error code that is recorded.
- When an error code is recorded, no other errors are recorded until the WebGL calls the method: gl.getError()
- This method is used to query for recorded error. It returns the current error code and resets the current error to gl.NO_ERROR
- To be sure exactly which calls generates an error, you need to call gl.getError() after each WebGL call.

WebGL Error Codes

- gl.NO_ERROR: There has not been a new error recorded since the last call to gl.getError().
- gl.INVALID_ENUM: Example, you call gl.drawArrays and send in gl.TRIANGLE instead of gl.TRIANGLES
- gl.INVALID_VALUE: An argument of a numeric type is out of range. An example is if you call gl.clear() and send in 100.5 instead of gl.COLOR_BUFFER_BIT
- gl.INVALID_OPERATION: Calling a method that is not allowed in the current WebGL state. An example is if you call gl.bufferData() without first calling gl.bindbuffer().
- gl.OUT_OF_MEMORY: There is not enough memory to execute a specific method.

webgl-debug.js

- The authors of Chromium (which is the open-source web browser project from which Google Chrome draws its source) have developed a small JavaScript Library
- To use it in your WebGL application:
  - Download the JavaScript library from https://github.com/KhronosGroup/WebGLDeveloperTools/blob/master/src/debug/webgl-debug.js
  - Place it together with your WebGL source and include it with the following code: <script src="webgl-debug.js"></script>
  - When you create the WebGLRenderingContext, you wrap it with a call that is available in the JavaScript library:
    - gl=WebGLDebugUtils.makeDebugContext(canvas.getContext(“webgl”));
  - After you perform these steps, all generated WebGL errors that you would get by using gl.getError() are printed in the JavaScript console.

WebGL Inspector

- The WebGL Inspector was perhaps the first WebGL debugging tool. It hasn't been updated in a long time, but it is still useful.
- WebGL Inspector is open source and is available as a chrome extension originally written by Ben Vanik. http://benvanik.github.io/WebGL-Inspector/
- WebGL Inspector (Built into Safari) can capture a frame and step through it, building the scene one draw call at a time; view textures, buffers, state, and shaders; etc.
- Once you have installed WebGL Inspector and you browse to a webpage that contains WebGL content, a small red icon with the text "GL" is shown to the far right of the address bar (so called omnibox).
- If you are trying to debug a local file (file://). If so, navigate to chrome://settings/extensions, expand the WebGL Inspector item and check "Allow access to file URLs".

Google Web Tracing Framework

- The Google Web Tracing Framework (WTF) is a full tracing framework, including support for WebGL similar to WebGL Inspector and Canvas Inspector. It is under active development on github;

WebGL Inspector

- If you click on the GL icon, two buttons named capture and UI appear on the top right of the web page, just below the GL icon.
- If you click the UI button, the full UI of the WebGL inspector is shown at the bottom of Google Chrome.
- Initially the UI will not contain any recorded information.
- If you click UI again, the full UI disappears.
- If you click on Capture, one frame of your WebGL application is recorded.
- After a frame is recorded, you can use WebGL Inspector to look at a lot of interesting information.
- The WebGL Inspector has six toolbar items at the top of its window: Trace, Timeline, State, Textures, Buffers, Programs

WebGL Inspector

- The trace Panel shows all the WebGL calls that have been recorded during the captured frame. Highlight redundant calls in “yellow” are calls to the WebGL API that do not change any meaningful state of WebGL (Useful for optimization)
- The Timeline Panel shows you real-time statistics.
- The State Panel shows a snapshot of the state for your WebGL application.
- The Textures panel shows textures, images for textures, information about texture wrap mode, texture filtering, and from URLs the textures are loaded.
- The Buffer Panel displays information about the different buffers, their content, size and also where they are bound. The right of the Buffers panel displays the geometry to which the selected buffer corresponds.
- The Programs Panel shows all the shader programs that the application uses. You can see which uniforms and attributes the program uses. You can also see the source code for both vertex shader and program shader.

Chrome Experiment

- Chrome Experiments is a showcase of web experiments written by the creative coding community
- https://www.chromeexperiments.com/webgl
- https://cesiumjs.org/EarthKAMExplorer/
- *EarthKAM Explorer provides web-based 3D exploration of satellite images taken by middle school students through the ISS EarthKAM program. It is an accessible, compelling, and social experience. EarthKAM Explorer supports the Leap Motion controller for hand-gesture input, and is written in JavaScript using Cesium, an open-source WebGL virtual globe and map, so it runs in a browser without a plugin.*

WebGL reference card

- https://www.khronos.org/files/webgl/webgl-reference-card-1_0.pdf
- https://www.khronos.org/files/webgl20-reference-guide.pdf
- The WebGL Objects
  - **WebGLBuffer** OpenGL Buffer Object.
  - **WebGLProgram** OpenGL Program Object.
  - **WebGLRenderbuffer** OpenGL Renderbuffer Object.
  - **WebGLShader** OpenGL Shader Object.
  - **WebGLTexture** OpenGL Texture Object.
  - **WebGLUniformLocation** Location of a uniform variable in a shader program.
  - **WebGLActiveInfo** Information returned from calls to **getActiveAttrib** and **getActiveUniform**.Has the following read-only
    - properties: name, location, size, type.

Fractal sets

- https://en.wikipedia.org/wiki/Fractal
- A **fractal** is a natural phenomenon or a mathematical set that exhibits a repeating pattern that displays at every scale.

The Sierpinski triangle

- The Sierpinski triangle (also with the original orthography Sierpiński), also called the Sierpinski gasket or the Sierpinski Sieve, is a fractal and attractive fixed set with the overall shape of an equilateral triangle, subdivided recursively into smaller equilateral triangles. Originally constructed as a curve, this is one of the basic examples of self-similar sets, i.e., it is a mathematically generated pattern that can be reproducible at any magnification or reduction. It is named after the Polish mathematician Wacław Sierpiński but appeared as a decorative pattern many centuries prior to the work of Sierpiński.
- https://en.wikipedia.org/wiki/Sierpinski_triangle#/media/File:Sierpinski_triangle.svg

Mandelbrot set

- https://en.wikipedia.org/wiki/Mandelbrot_set
- The **Mandelbrot set** is the set of complex numbers c for which the sequence (c, $c^2 + c$, $(c^2+c)^2 + c$, $((c^2+c)^2+c)^2 + c$, $(((c^2+c)^2+c)^2+c)^2 + c$, …) does not approach infinity.
- Similar to Julia set https://en.wikipedia.org/wiki/Julia_set

```
For each pixel ($P_x$, $P_y$) on the screen, do:{
  x0 = scaled x coordinate of pixel (scaled to lie in the Mandelbrot X scale (-2.5, 1))
  y0 = scaled y coordinate of pixel (scaled to lie in the Mandelbrot Y scale (-1, 1))
  x = 0.0
  y = 0.0
  iteration = 0
  max_iteration = 1000
  while ( x*x + y*y < 2*2  AND  iteration < max_iteration )
  {  xtemp = x*x - y*y + x0
    y = 2*x*y + y0
    x = xtemp
    iteration = iteration + 1}
  color = palette[iteration]
  plot($P_x$, $P_y$, color) }
```

Exercise

- 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.
- Change the color of the triangle from white to red.
- Change the background from black to green.
- Make the triangle look smaller on the screen without change the vertex data by changing the viewport.
- Change the z-coordinates for the vertices of the triangle so the triangle disappears. When does this happen?

Graphics card

- The easiest way to find your graphics card is to run the DirectX Diagnostic Tool:
- Click **Start**.
- On the **Start** menu, click **Run**.
- In the **Open** box, type "dxdiag" (without the quotation marks), and then click **OK**.
- The DirectX Diagnostic Tool opens. Click the **Display** tab.
- On the **Display** tab, information about your graphics card is shown in the **Device**section. You can see the name of your card, as well as how much video memory it has.

Monitoring the GPU utilization

- 1. Process Explorer:
  Process Explorer inside the VM gives the most accurate measure of the GPU memory used by the applications. An aggregate of the memory used over the user applications can give a good estimate of the framebuffer sizing for each VM. download it here: https://technet.microsoft.com/en-us/sysinternals/processexplorer.aspx).

  a. Launch Process Explorer inside the VM. By default it does not show the GPU counters
  b. Goto View>select columns>Process GPU> select all relevant options
- c. Click ok. You should see new columns added detailing GPU committed and dedicated memory. Now, for each process for e.g. chrome.exe (Google chrome) you can see the exact amount of GPU memory being used. An aggregate of these counters for all the applications running
- d. Double click on the process to see detailed graphed usage
- e. While Process Explorer does not have logging options, It does graph the GPU usage in real time so we can save those screenshots to watch the usage over time.

Enable Chrome WEBGL on MAC

Step 1 would be to look the status of your Chrome GPU settings.
Paste this "`chrome://gpu/`" in Chrome web address field.
step 2:
Do following:
To force Chrome / Chromium to use hardware acceleration, open a new tab, type "`chrome://flags`" (without quotes), search for "Override software rendering list", enable it and restart Chrome / Chromium.

Web Resources

- www.opengl.org
- get.webgl.org
- www.kronos.org/webgl
- www.chromeexperiments.com/webgl
- learningwebgl.com
- https://www.khronos.org/files/webgl20-reference-guide.pdf
- https://webgl2fundamentals.org/
- Enable webgl2 in Chrome by going to chrome://flags/
- https://benvanik.github.io/WebGL-Inspector/
- https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API