Introduction to Rendering

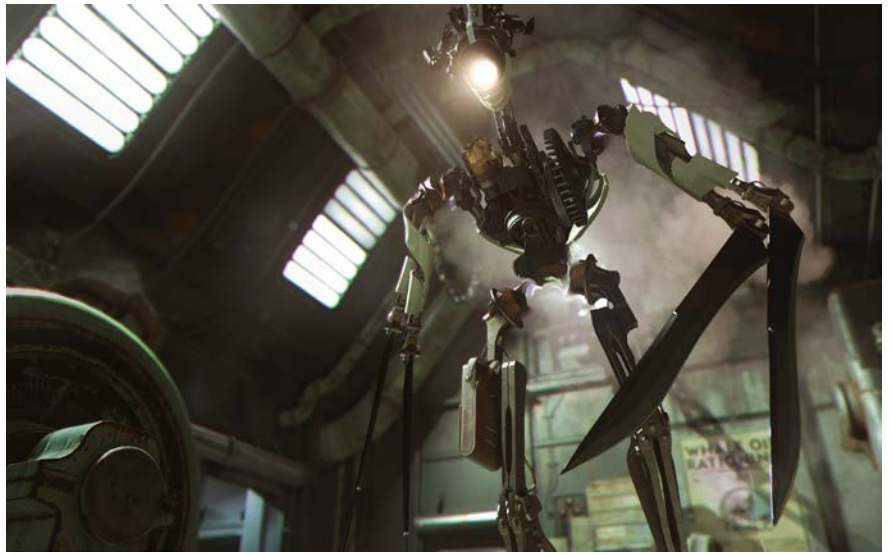
CS 418: Interactive Computer Graphics

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Eric Shaffer

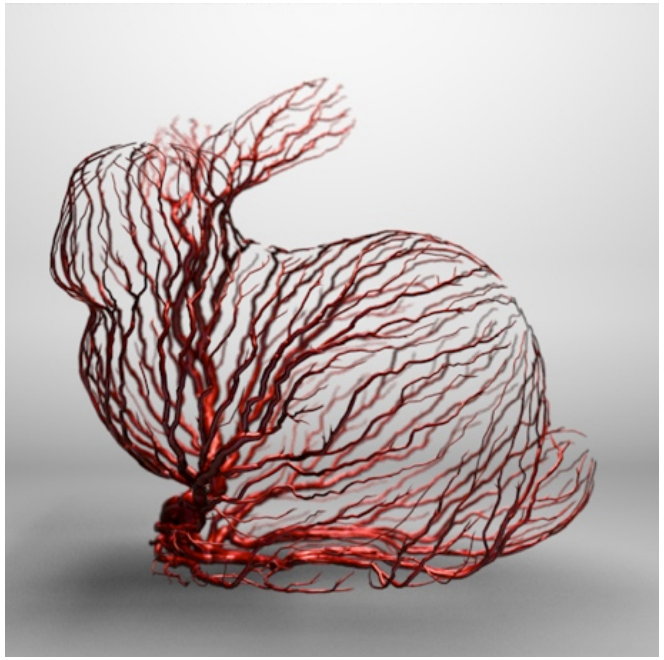
Computer Graphics is Used By…

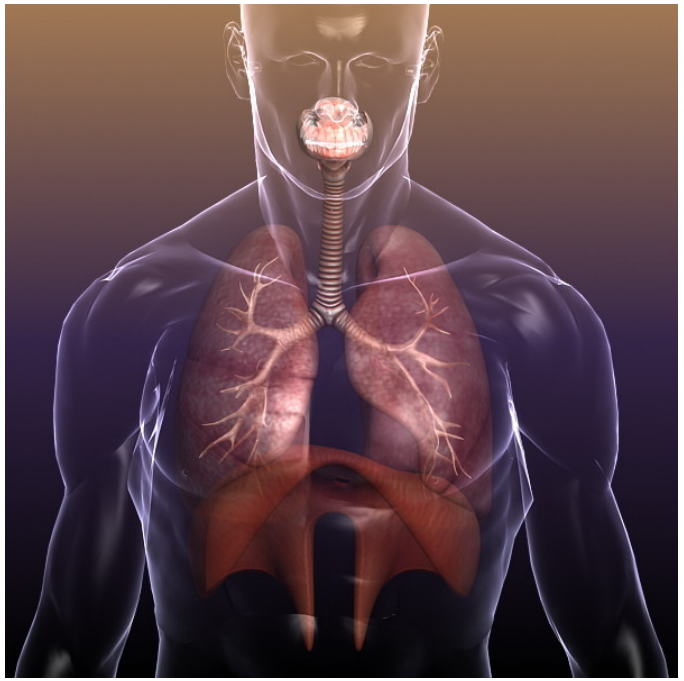
- Video Game Industry
  - Revenue of $100B globally in 2017



Computer Graphics is Used By…

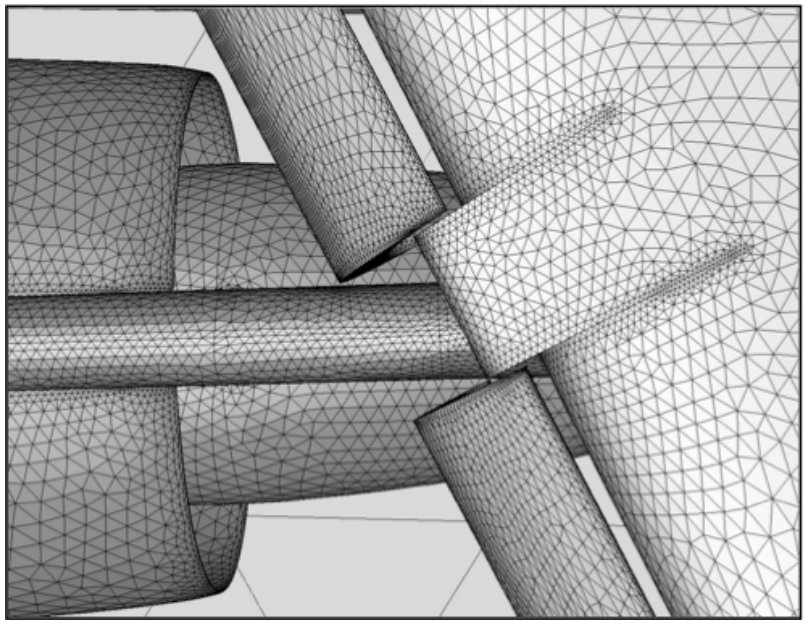
- Medical Imaging and Scientific Visualization
  - Imaging one of the biggest advances in medicine
  - Sci Vis allows people to see previously hidden phenomena

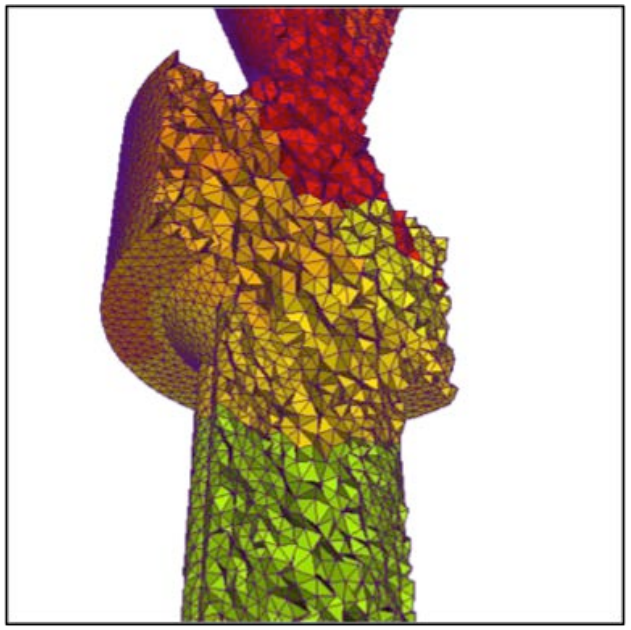




Computer Graphics is Used By…

- Computer Aided Design
  - Engineering, Architecture, the Maker movement





Computer Graphics is Used By…

- Movie Industry
  - Production computer graphics…non-interactive (CS 419)

Although…Production CG is Changing…

GDC 2017 NEWS STAR WARS

Star Wars: Rogue One's best character was rendered in real time, a cinema first

A breakthrough moment for film, according to Epic

by *Julia Alexander* | *Mar 1, 2017, 1:16pm EST*

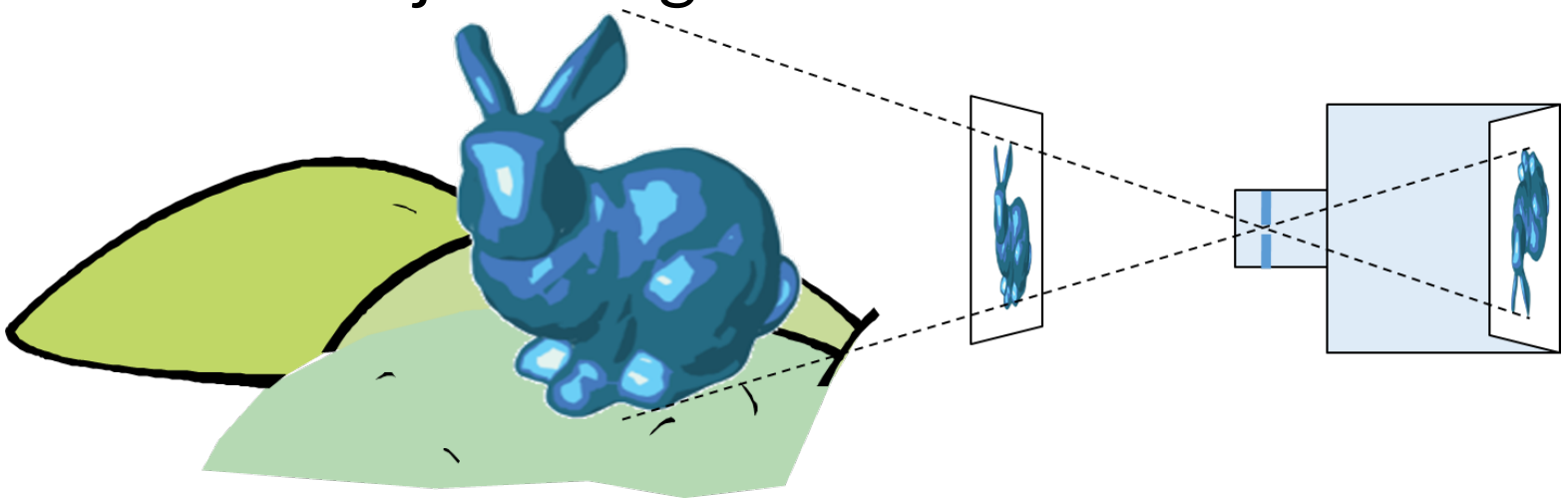
TWEET SHARE PIN

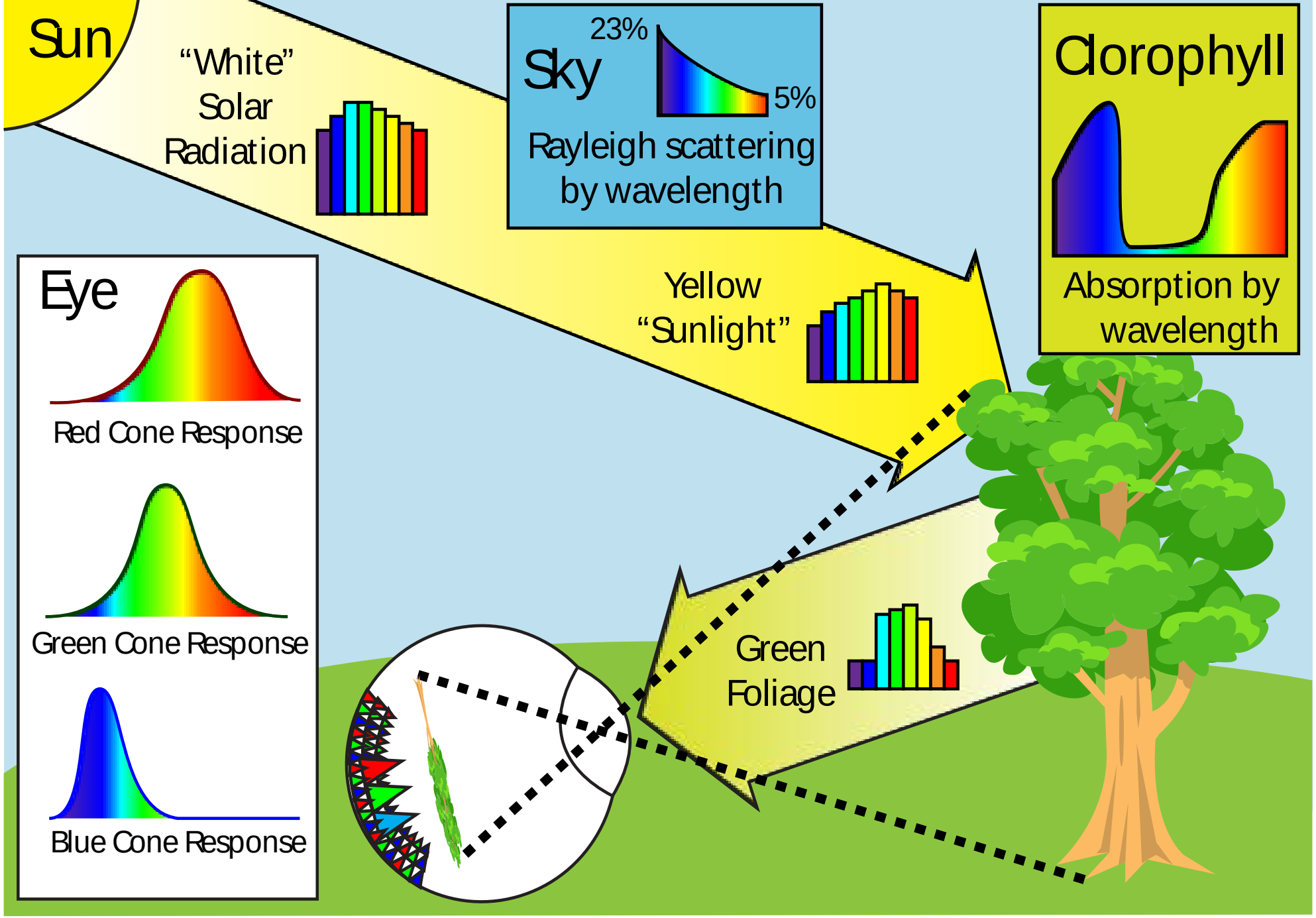
Why would this be useful when making a movie?

Why is the speedup in producing the scene greater than just the one time speedup of real-time over production?

3D Graphics: Image Formation

- Goal in CG (usually)is to generate a 2D image of a 3D scene…
  - The input data is a scene description
  - Output is an image
- One approach is to computationally mimic a camera or human eye
- In the scene…there are objects…lights…and a viewer





"Chlorophyll"

Light is EM radiation

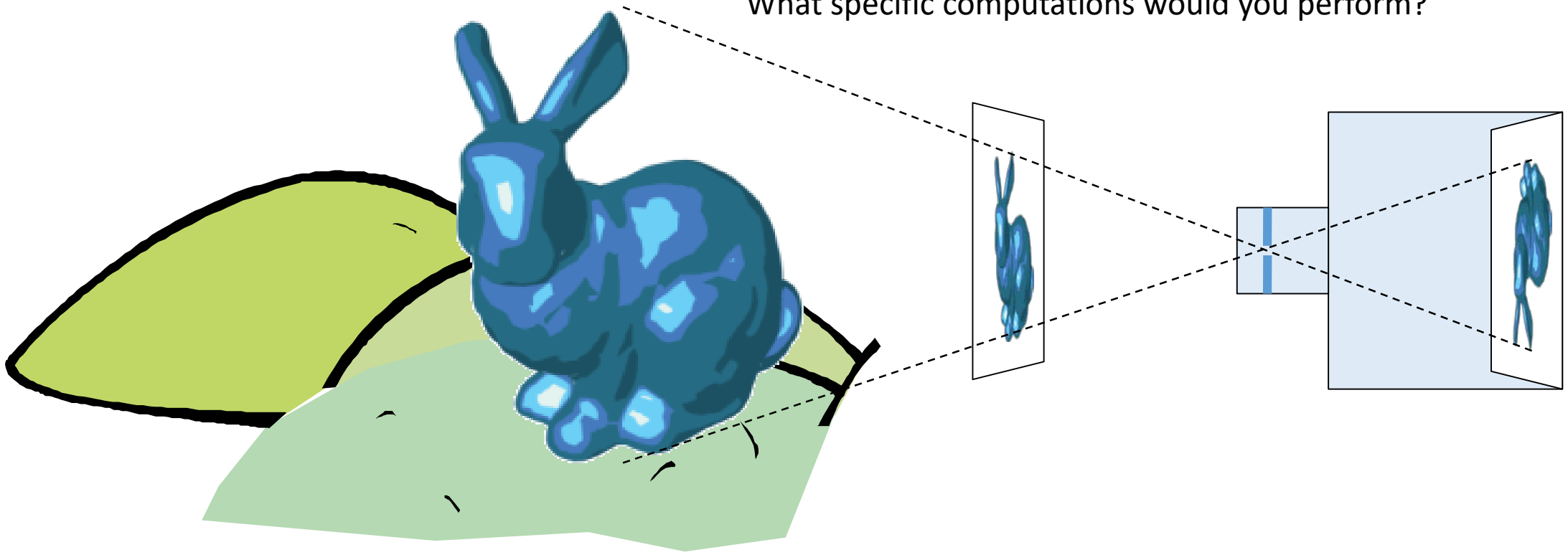
Usually multiple wavelengths mixed together in a power spectrum

The spectrum sensed by our eyes gets modified multiple times.

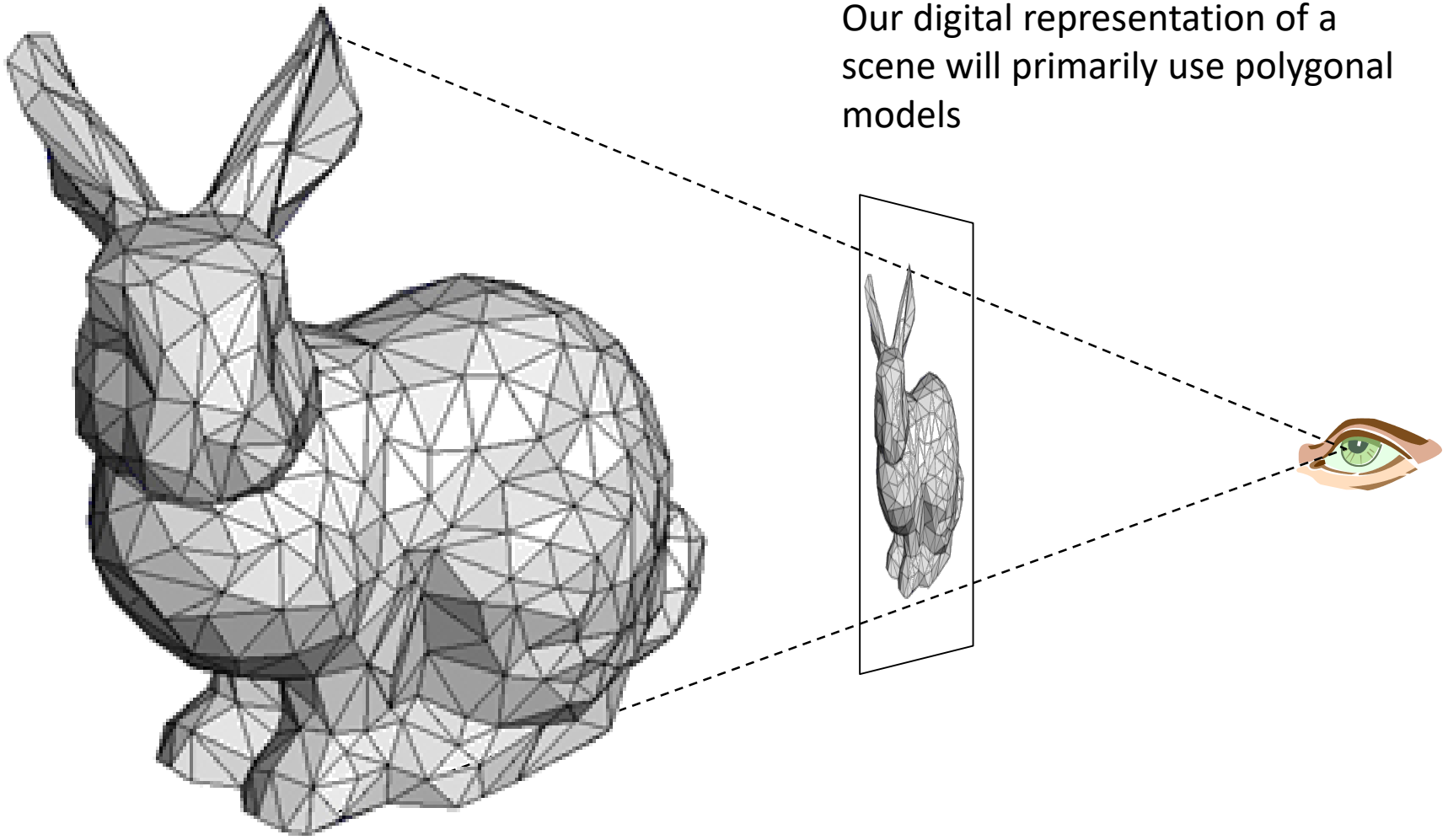
Human perception of color relies on 3 different cell types that sense different regions of the spectrum

Synthetic Camera Model

How can we computationally mimic a camera?
What specific data would you need?
What specific computations would you perform?

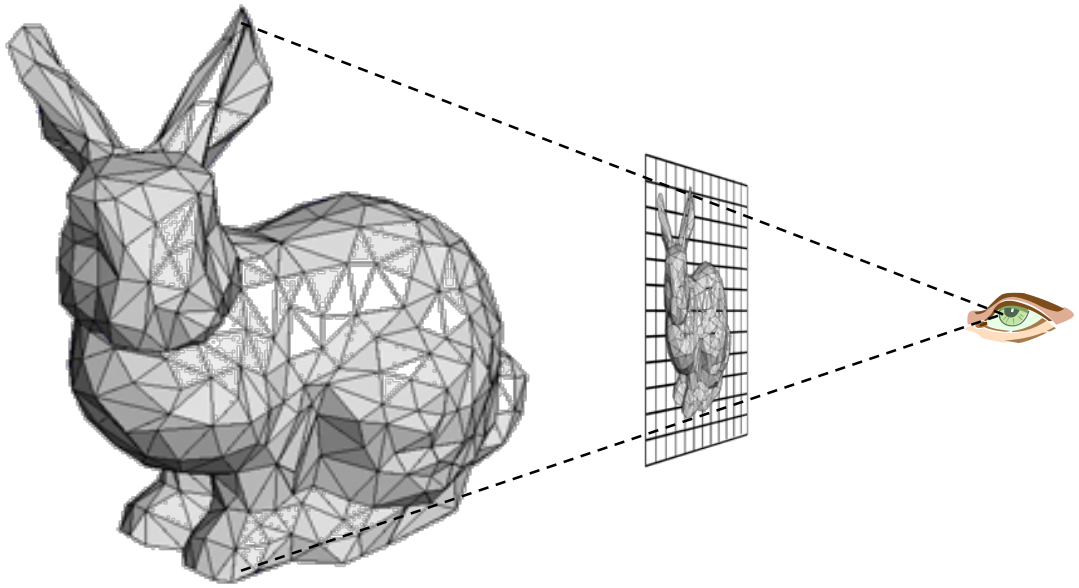


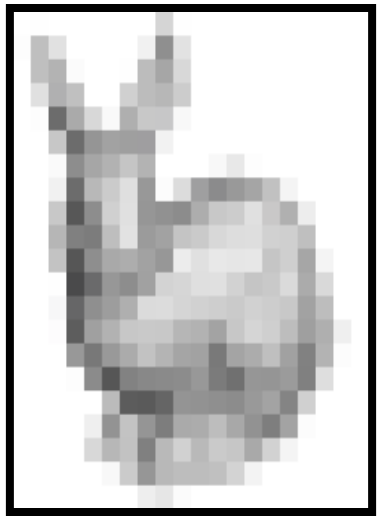
Polygonal Models

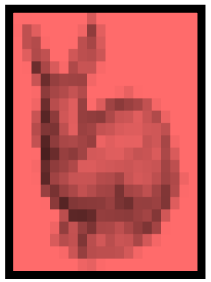


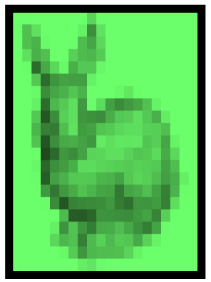
Our digital representation of a scene will primarily use polygonal models

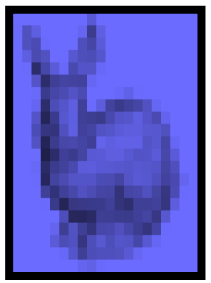
Pixel Discretization











By Gilles Tran - http://www.oyonale.com/modeles.php?lan

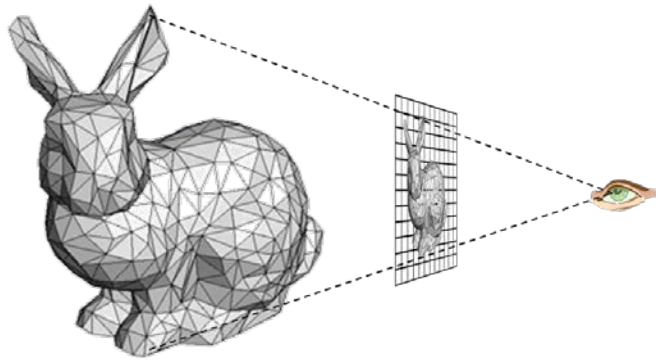
- **Rendering** or **image synthesis** is the automatic process of generating a photorealistic or non-photorealistic image from a 2D or 3D mode

|

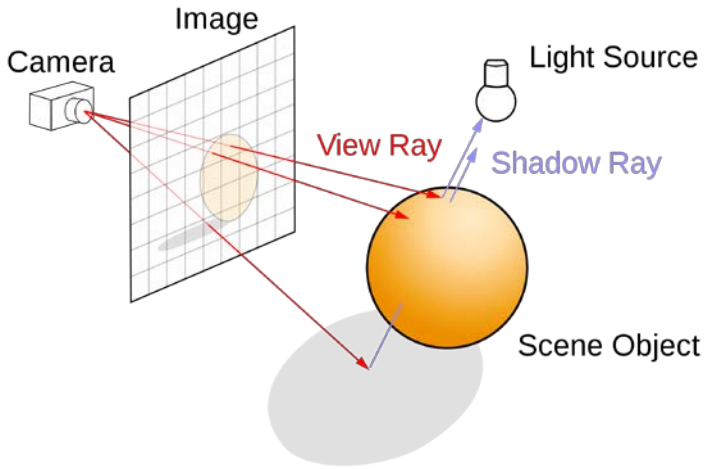
- Rendering methods generally use one of two approaches
  - Rasterization (focus of CS 418)
  - Ray Tracing (focus of CS 419)
  - Though, sometimes you can use both….
  - …and the are other methods like radiosity

Rasterization versus Ray Tracing

- To oversimplify….
- In rasterization, geometric primitives are projected onto an image plane and the rasterizer figures out which pixels get filled.



- In ray-tracing, we model the physical transport of light by shooting a sampling ray though each pixel in an image plane and seeing what the ray hits in the scene



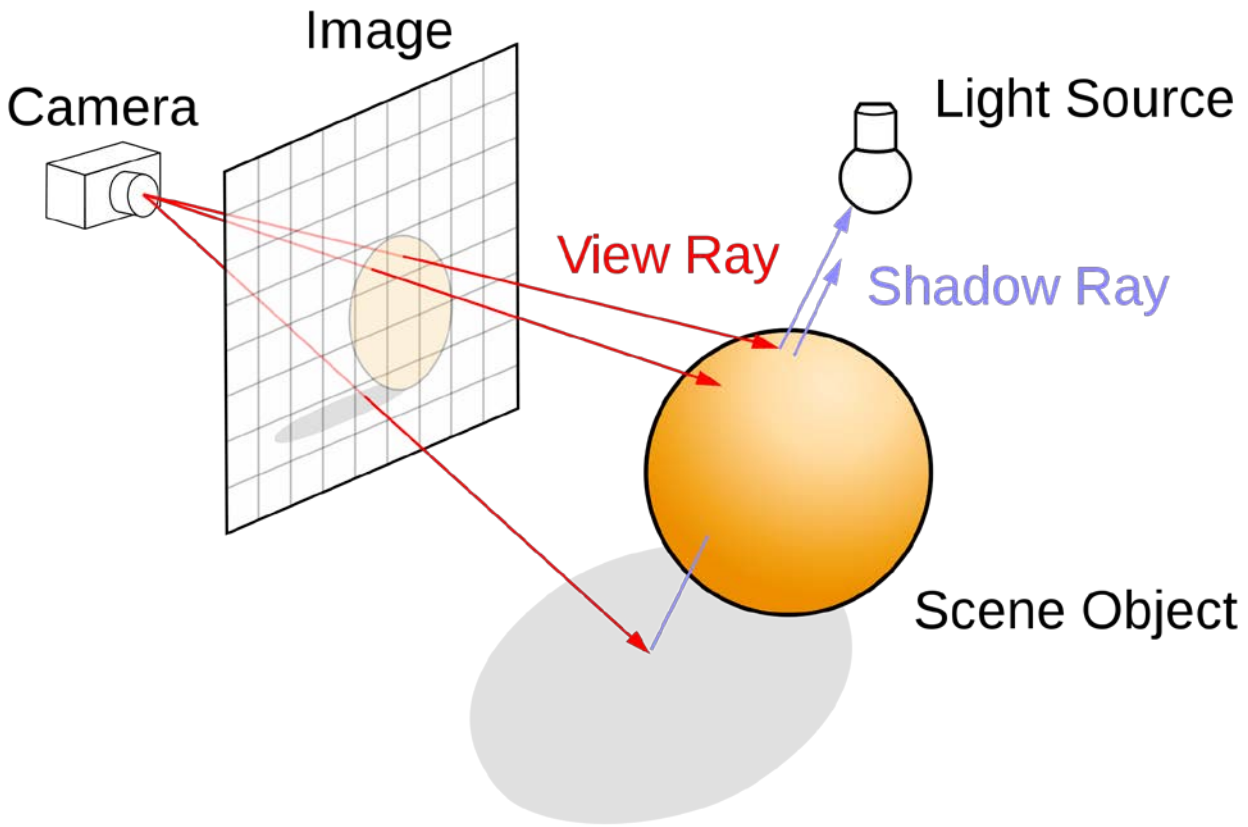
Ray Tracing

Follow ray of light….

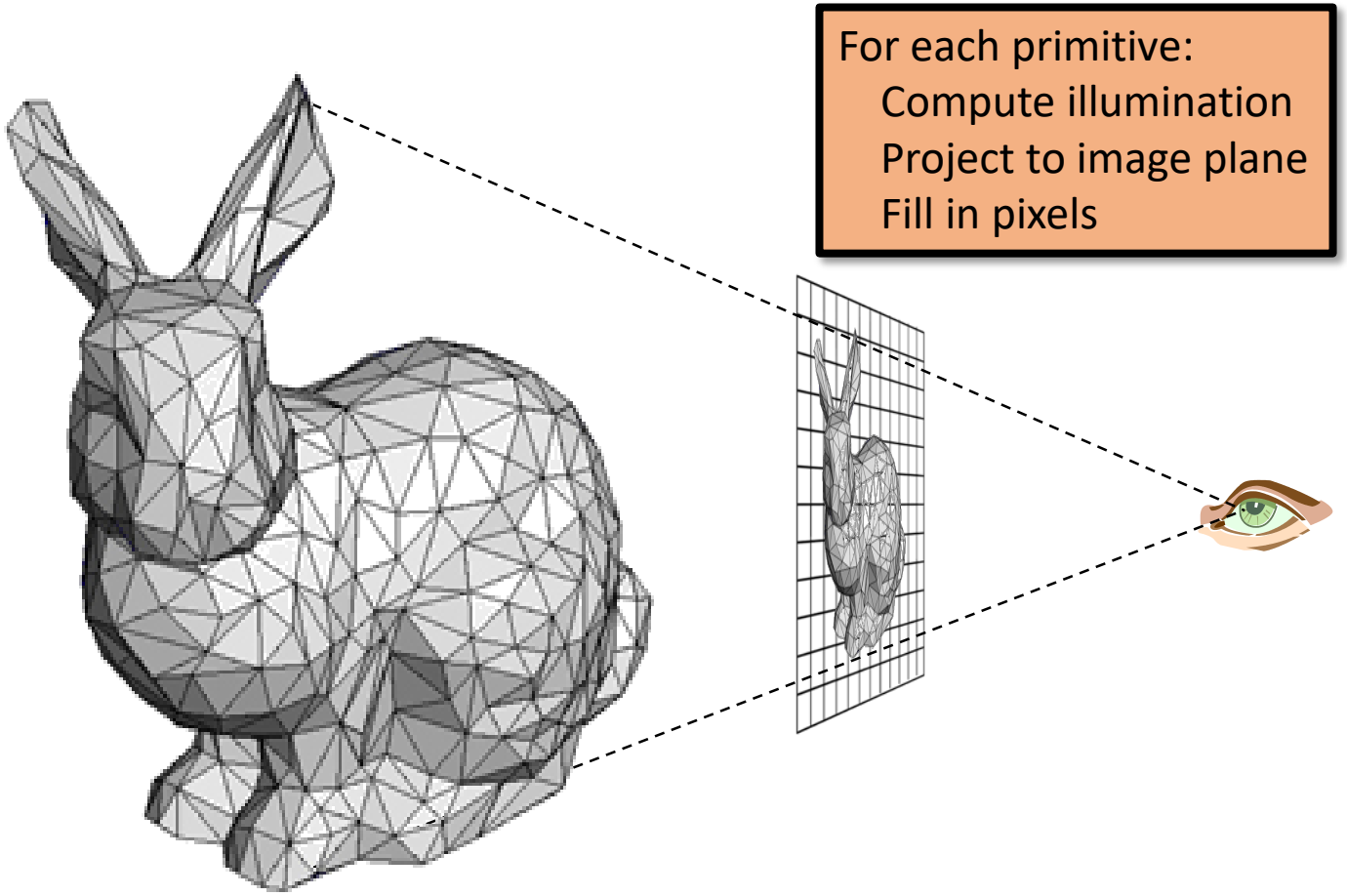
Can trace from an eyepoint through a pixel

See what object the ray hits…

How would you check to see if the object is lit?



Rasterization

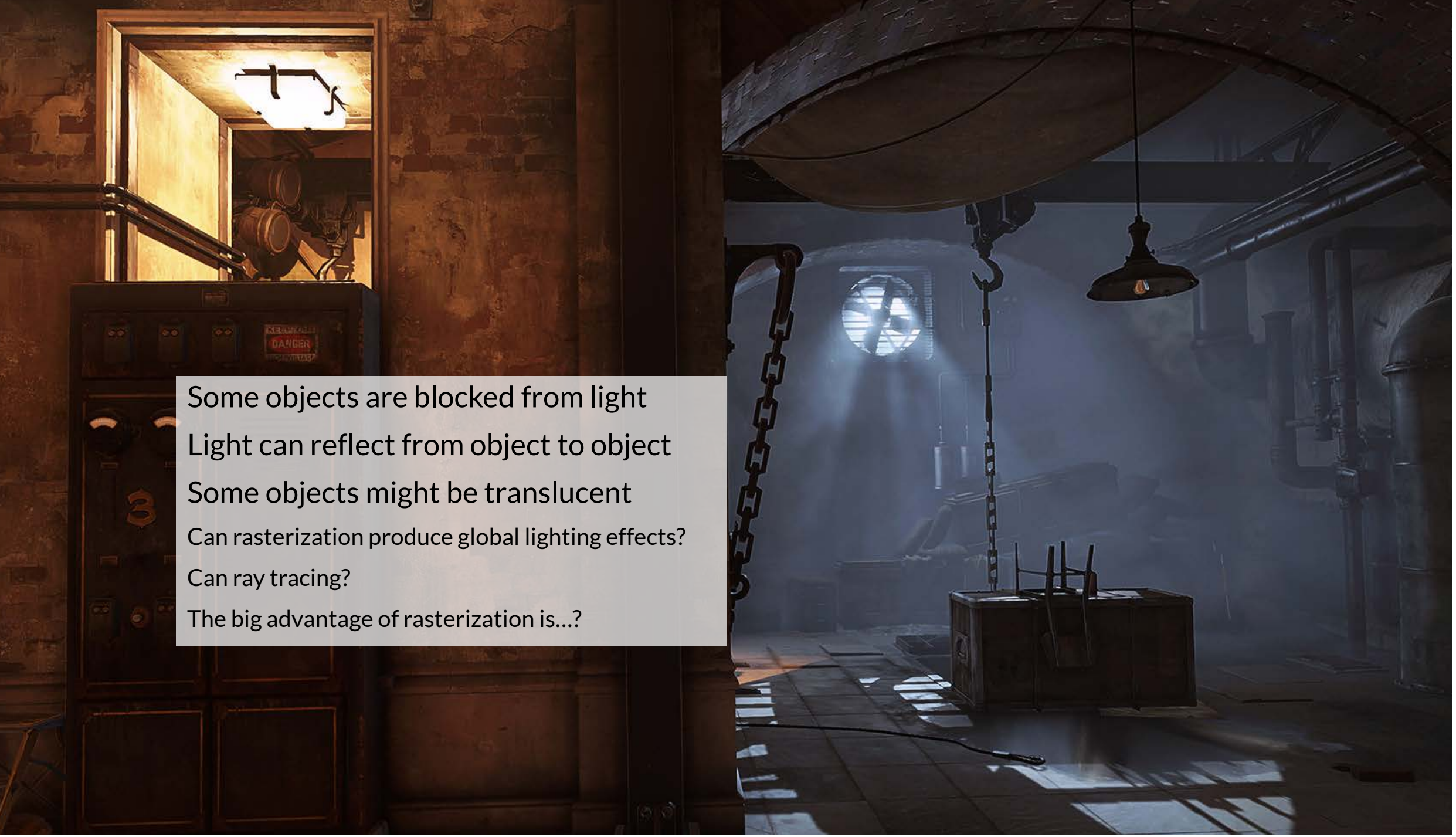


Global versus Local Illumination

For true photo-realism:

We cannot compute color or shade of each object independently

Why?



Some objects are blocked from light

Light can reflect from object to object

Some objects might be translucent

Can rasterization produce global lighting effects?

Can ray tracing?

The big advantage of rasterization is…?

Rasterization Engines

- Most low-level graphics libraries use a camera model
- API typically requires you to specify
  - Objects in the scene
  - Materials the objects are made of
  - Viewer (position, view direction, field of view,…)
  - Lights - what parameters do you think typically are used?

- The engine (i.e. the library) will use pipeline-style processing
  - The input geometry flows through several processing stages

API = Application Programming Interface

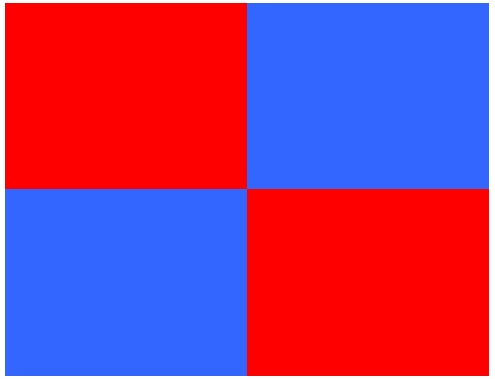
Definitions: Pixel and Raster

A *pixel* is the smallest controllable picture element in an image

A *raster* is a grid of pixel values

Typically rectangular grid of color values

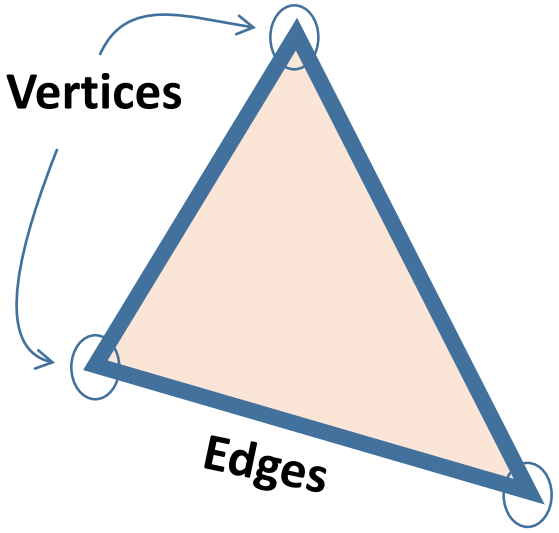
(1.0, 0.0, 0.0), (0.0, 0.0, 1.0)
(0.0, 0.0, 1.0), (1.0, 0.0, 0.0)



RGB Color Representation
A color is a triple (R,G,B) representing a mix of red, green, and blue light.
Each color channel has a value in [0, 1] indicating how much light is emitted.

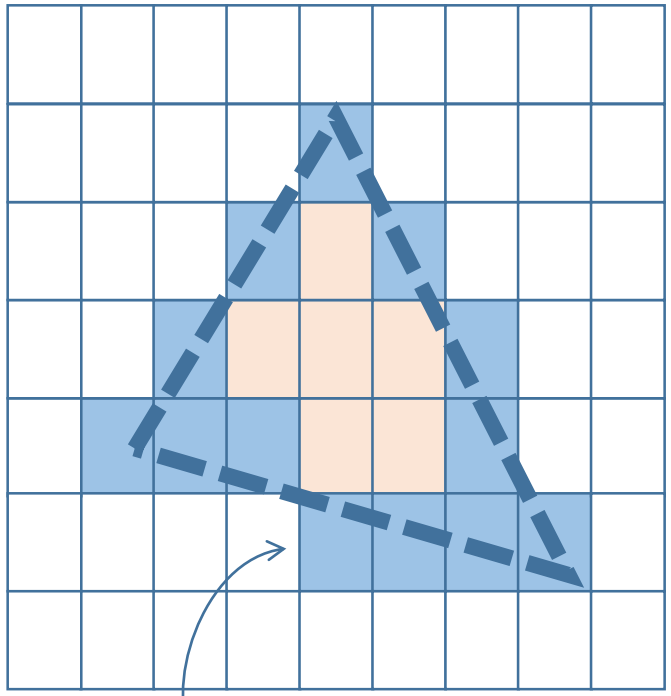
Rasterization

Primitives



Generate a raster image
from a vector description

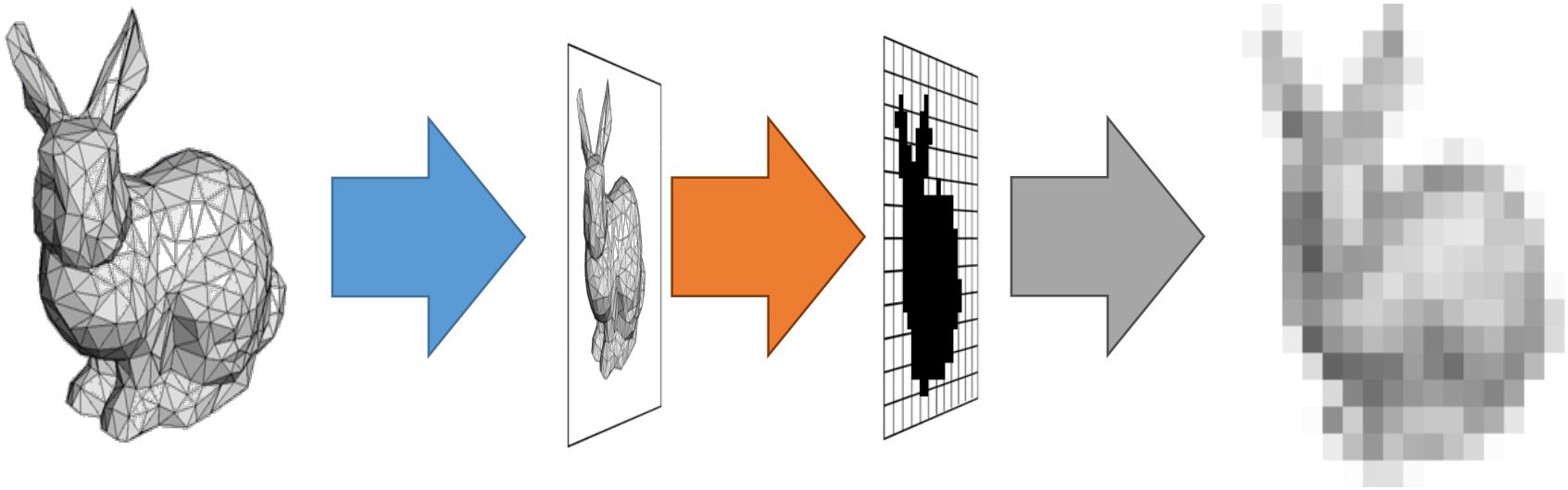
Pixels



Vector Graphics Representation
Is a purely mathematical representation of shape. For example, a line is y=mx+b.
Typically, ***vector graphics*** refers to 2D shapes, but the idea applies to 3D as well.

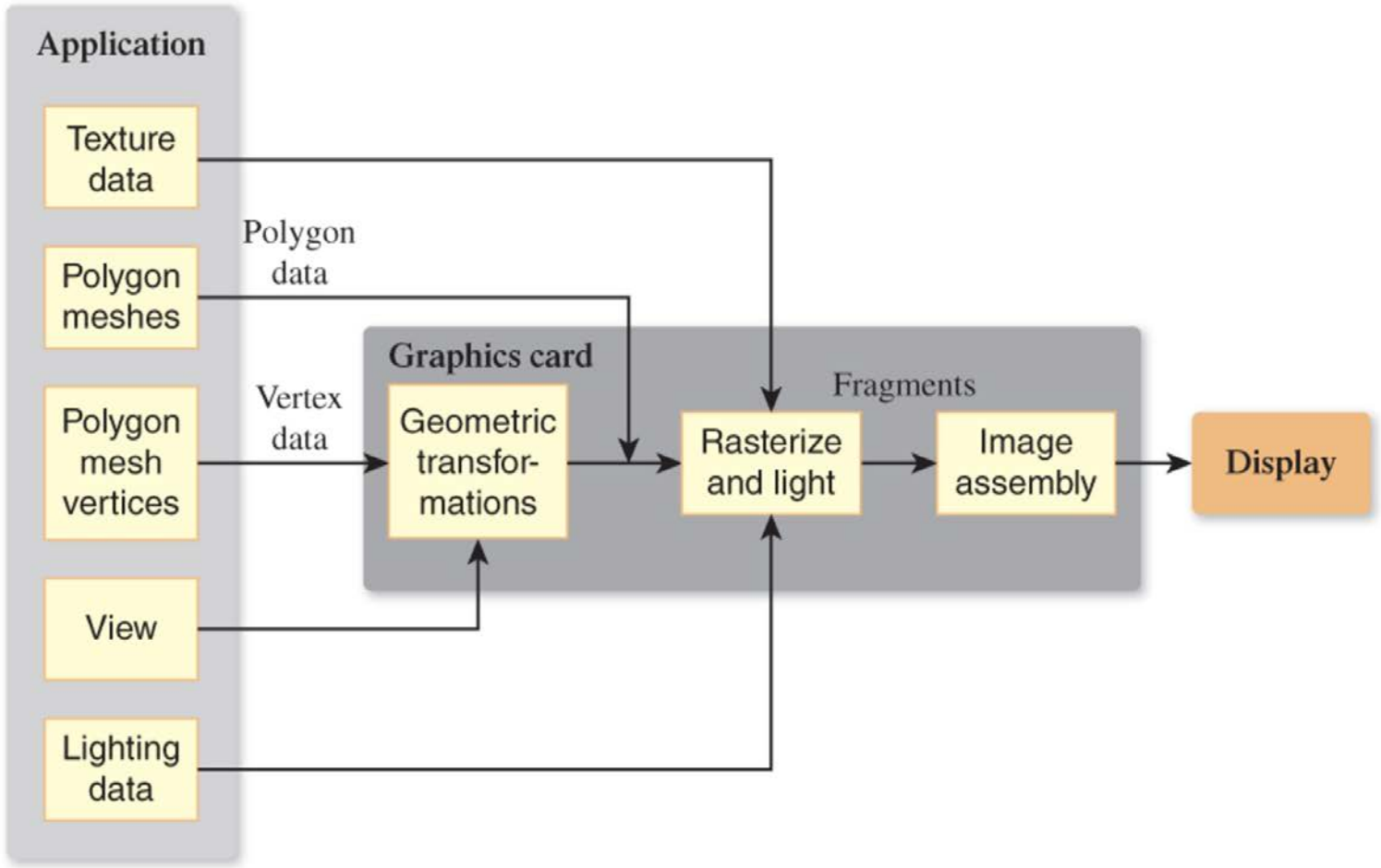
3D Graphics Pipeline

Vertex Processing | Rasterization | Fragment Processing



Fragments
Are like pixels…but they aren't necessarily the finalized pixels you see in an image.
Each fragment has a 2D location in a raster and a color.
Final pixel value is typically found by applying *hidden surface removal* and possibly *compositing* to a set of fragments.

Rasterization is a Pipeline

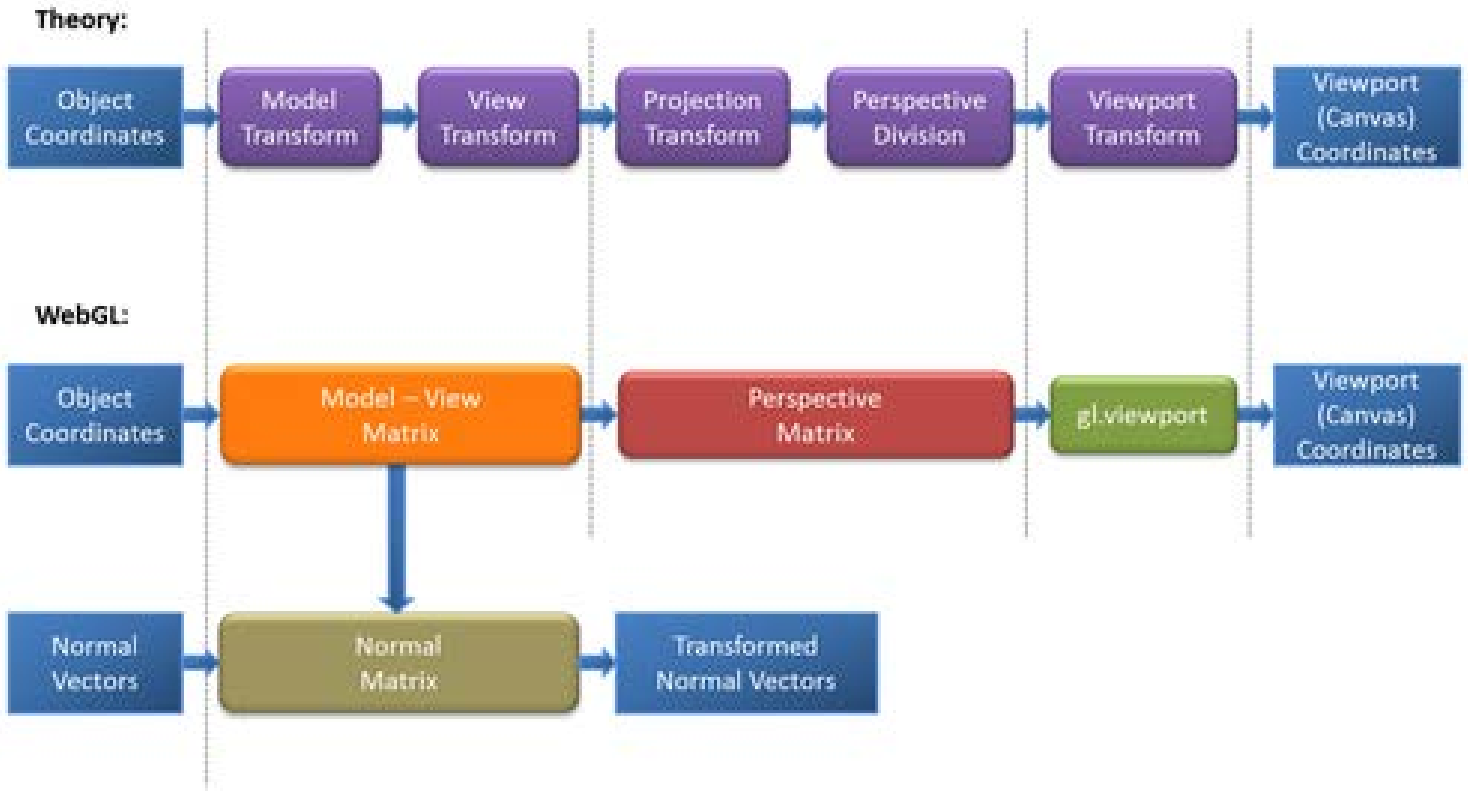


- Data for objects in the scene usually in the form of polygonal meshes
- Most of the work to render an image is done on the Graphics Processing Unit (GPU)
- GPU code will have at least two parts
  - **Vertex Shader**
  - **Fragment Shader**

Vertex Shader

- Program that runs on the GPU
- Typically transforms vertex locations from one coordinate system to another
  - Transformations can be useful for placing objects in your scene
  - Also, some operations on the geometry are easier when done in specific coordinate system
- Change of coordinates usually equivalent to a matrix transformation
- Vertex processor can also computes vertex colors

Changing Coordinate Systems



Model Transformation:
Move a model from a local coordinate system to a position in the "world"

View Transformation:
Keeping camera fixed, move all the objects in the world so that they are seen as if from a specific viewpoint

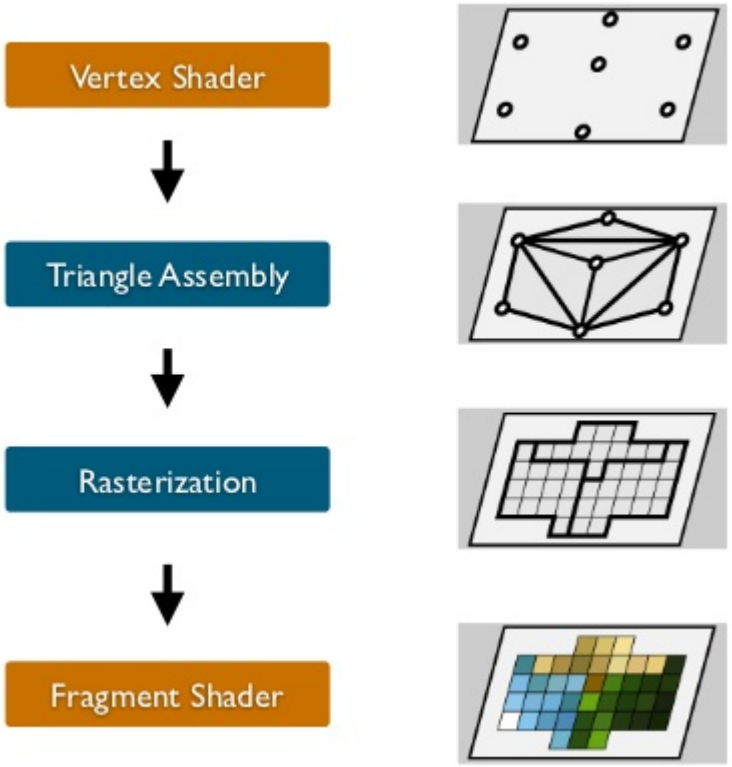
Projection Transformation:
Change coordinates so that a 3D to 2D projection of the geometry is done correctly

Viewport Transformation:
Change from 2D coordinates in [-1,1] to pixel coordinates

Pipeline Step: Primitive Assembly

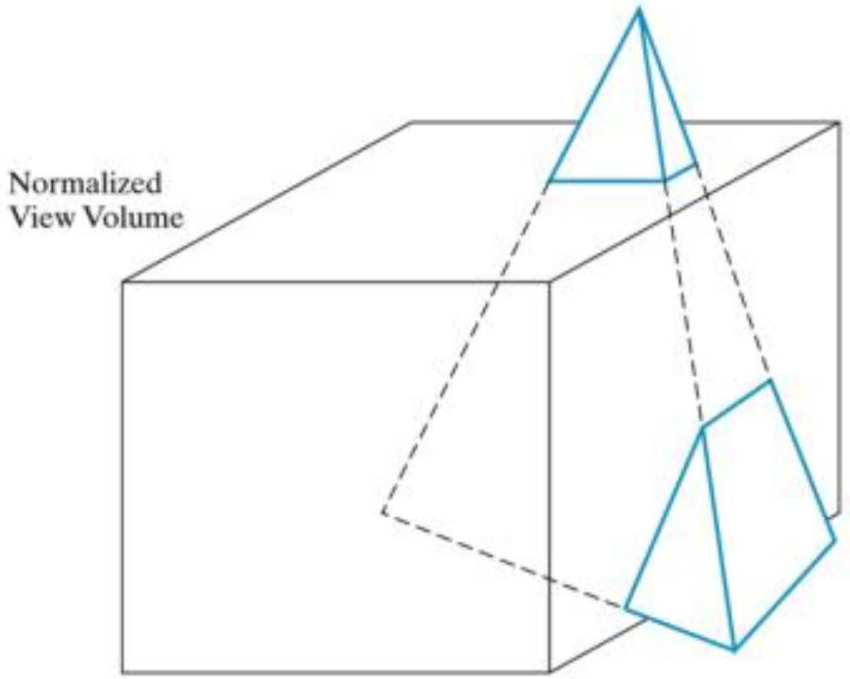
Vertices must be collected into geometric objects before ***clipping*** and ***rasterization*** can take place

- For WebGL:
  Points, Line Segments, Polygons
- Other APIs sometimes support more complex geometry (e.g. curves)



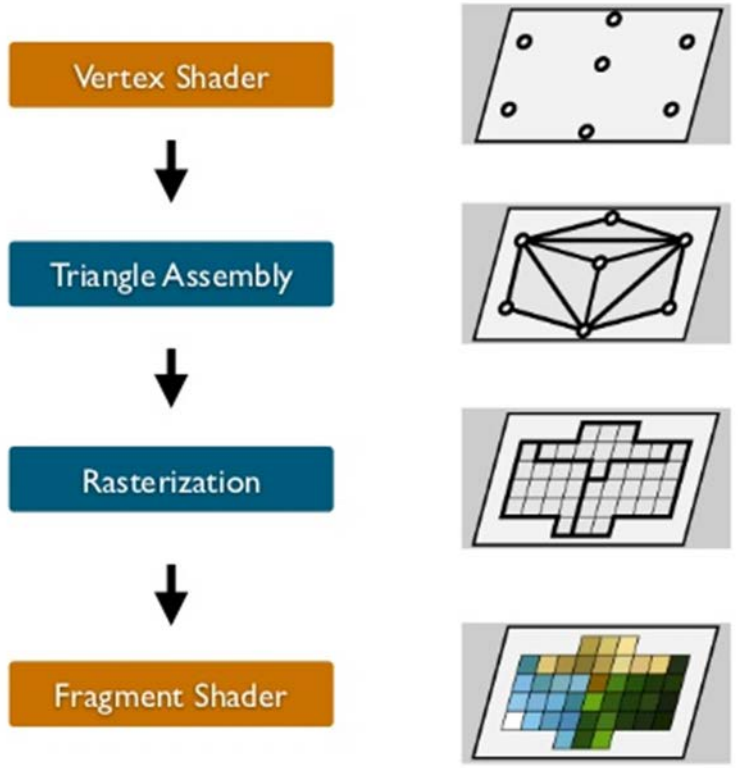
Pipeline Step: Clipping

- Our virtual camera can only see part of the world
- Objects not within this volume are said to be *clipped* out of the scene
- Why would we do this?
  Why not just render everything and keep only pixels that fall within the viewing window?



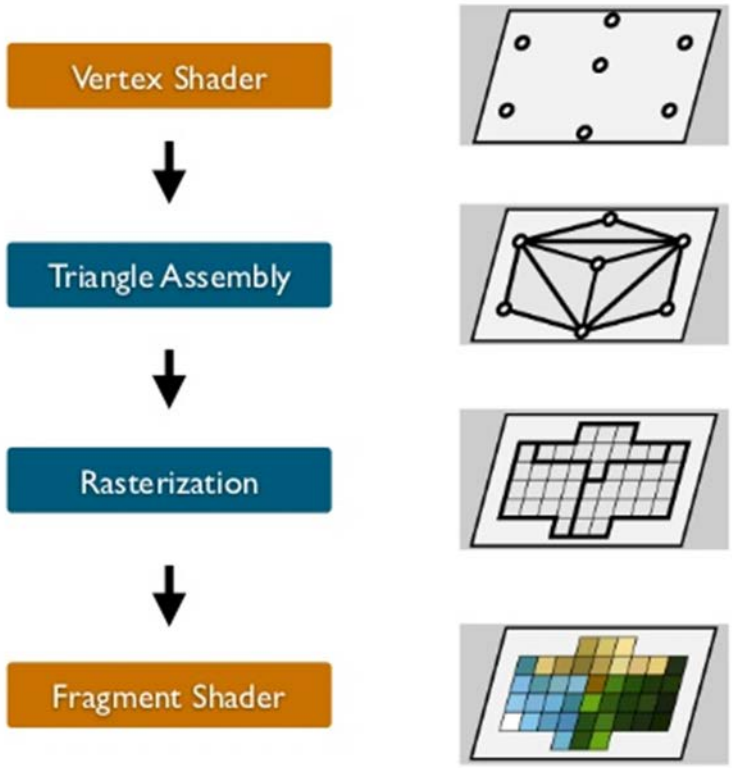
Rasterization

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



Pipeline Step: Fragment Processing

- Fragment shader computes color of the fragment
- Fragments are processed to determine the color of final pixel
  - Fragments at same location may need to be composited
- Is a fragment blocked by other fragments closer to the camera?
  - Hidden-surface removal



The WebGL Rasterization Engine

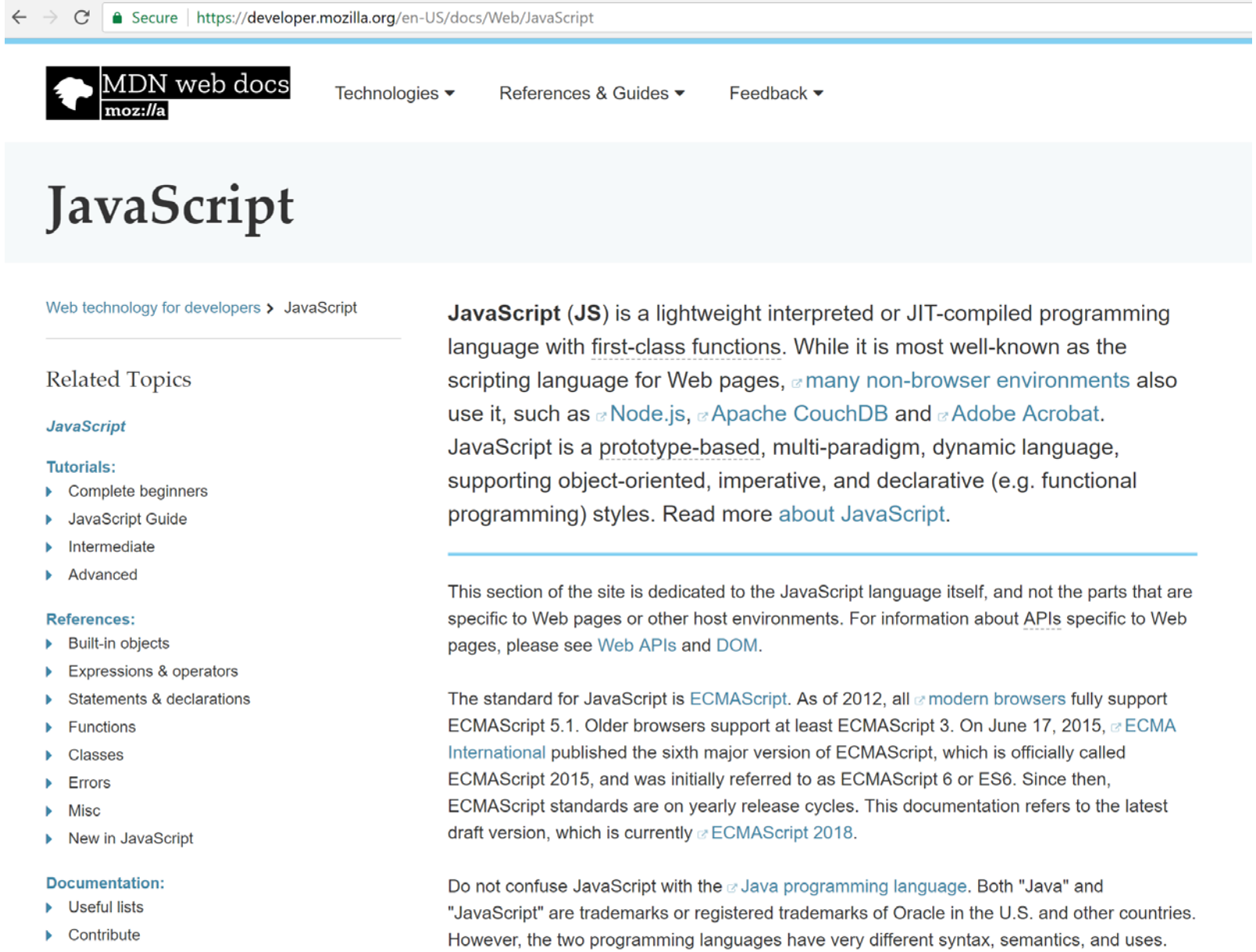
- WebGL relatively new (2011) 3D graphics support for web
- WebGL advantages
  - runs in browser
  - naturally cross-platform
  - don’t need to obtain/build other libraries
  - gives you “windowing” for free
  - easy to publish/share your stuff
- Disadvantages
  - Depends on how you feel about JavaScript
  - Performance can be tricky

Programming Language for CS 418

- HTML
- JavaScript
- WebGL
- WebGL version of the GLSL shading language (**runs on GPU**)
- Chrome as default browser
- Chrome DevTools to debug code
- **If you have a laptop, bring it to recitation section**
- Some WebGL examples: https://www.chromeexperiments.com/webgl

JavaScript

- We will provide example code
- You are responsible for learning what you need to complete the assignments
- Mozilla reference/tutorials are quite good



https://developer.mozilla.org/en-US/docs/Web/JavaScript

What is WebGL?
Let's start with a word about OpenGL

- Open standard for 3D graphics programming
  - Developed by Silicon Graphics in 1992
  - Available on most platforms…
  - Bindings available for lots of languages…
  - It's low level
- "Windowing" typically requires another library
  - e.g. GLUT
- Version 3.0 (2008) introduced programmable shaders
  - Deprecated fixed-function pipeline
- **Vulkan** API is the successor technology (still pretty new…2016ish)

WebGL is not exactly OpenGL

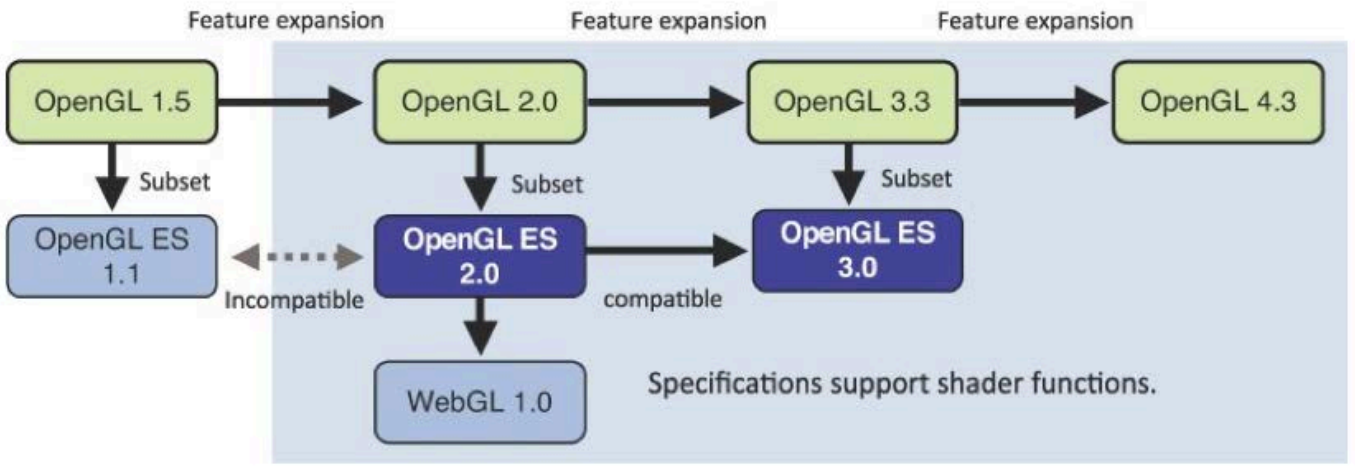


Figure from *WebGL Programming Guide: Interactive 3D Graphics Programming with WebGL* by Matsuda and Lea

WebGL Application Structure

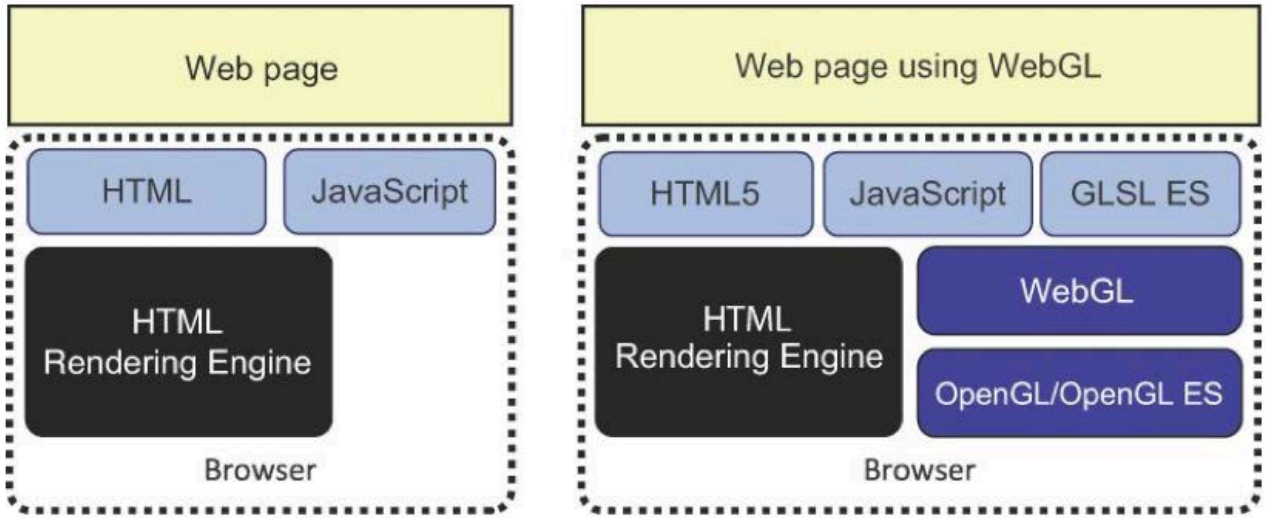


Figure from *WebGL Programming Guide: Interactive 3D Graphics Programming with WebGL* by Matsuda and Lea

Your application will generally just have HTML and JavaScript files

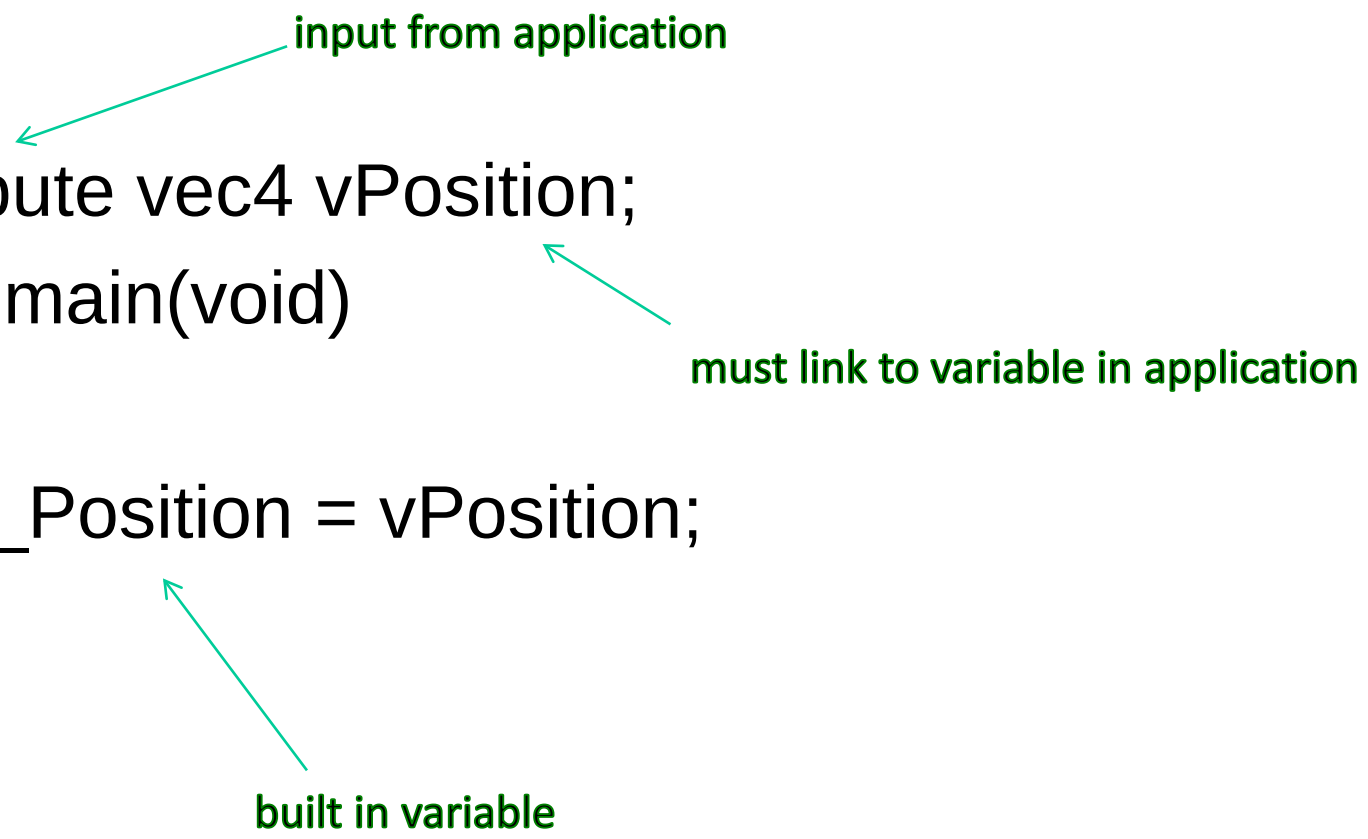
WebGL and GLSL

- WebGL requires you provide shader programs
- GLSL OpenGL Shading Language
- C-like with
  - Matrix and vector types (2, 3, 4 dimensional)
  - Overloaded operators
  - C++ like constructors
- Similar to NVIDIA's Cg and Microsoft HLSL
- Code sent to shaders as source code
- WebGL functions compile, link and get information to shaders

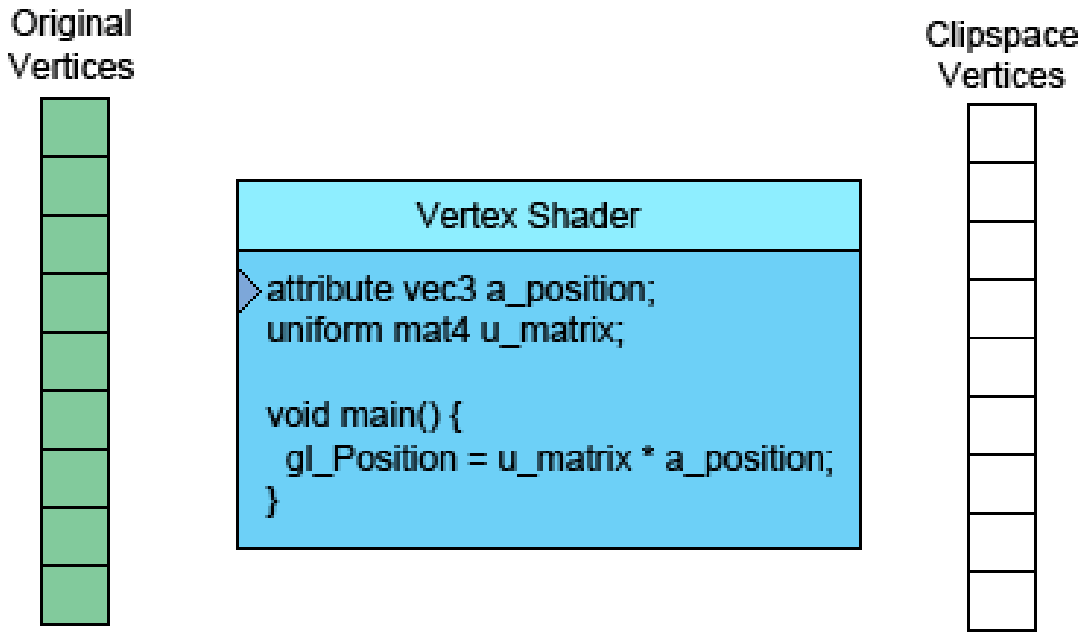
Shaders

- Shader source code will be in the HTML file or a JS file…usually
- Vertex Shaders generally move vertices around
  - Projection, animation, etc.
  - Assign a value to the built-in variable gl_Position
- Fragment Shaders generally determine a fragment color
  - Assign a value to the built-in variable gl_Position

Simple Vertex Shader



What a Vertex Shader Does…



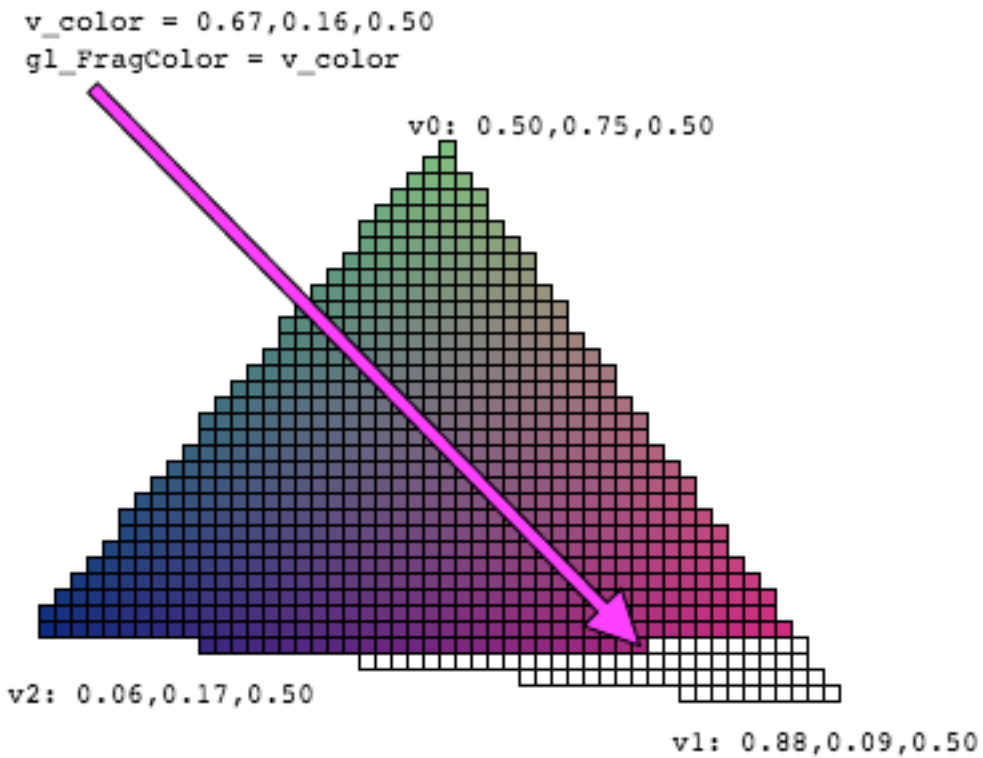
Taken from webglfundamentals.org

Can you guess what is slightly incorrect about this animation?

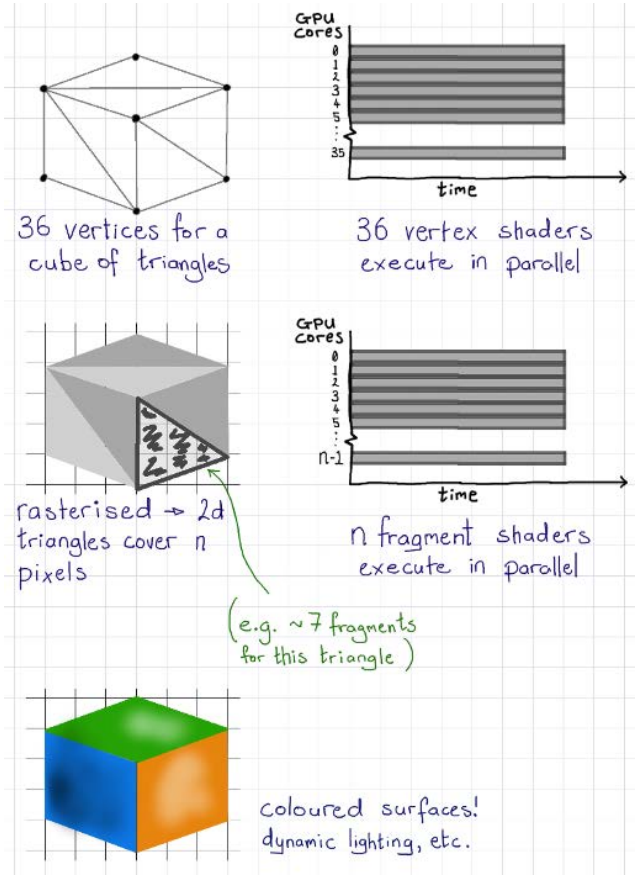
Simple Fragment Program

```
precision mediump float;
void main(void)
{
   gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);
}
```

What a Fragment Shader Does…



Processing on a GPU



The Graphics Processing Unit (GPU) will have a large number of cores.

This architecture supports a massively-threaded environment for processing vertices and fragments (think of fragments as pixels for now)

Image from http://antongerdelan.net/opengl/shaders.html

Shading

- ***Shading*** The process of generating a color using lighting and material information
- You can do this in either vertex shader or the fragment shader
- Why does per-fragment shading look more realistic?

per vertex shading

per fragment shading

What Should You Know?

- General principles of rasterization
- Pipeline model of a rasterization engine
- What a vertex shader does
- What a fragment shader does
- Difference between rasterization and ray-tracing

CS 418…About the Course

- **Interactive** Computer Graphics
- Focus on algorithms and techniques used in **rasterization**
  - Rasterization is fast enough for real-time complex 3D rendering
- The course will teach you how to use WebGL
  - Web-based rasterization engine
  - Similar features to many other technologies (e.g. OpenGL, Vulkan, D3D)
- We will also cover fundamental graphics algorithms
  - Things like line drawing that reside inside the WebGL library

Things you would not use WebGL for..

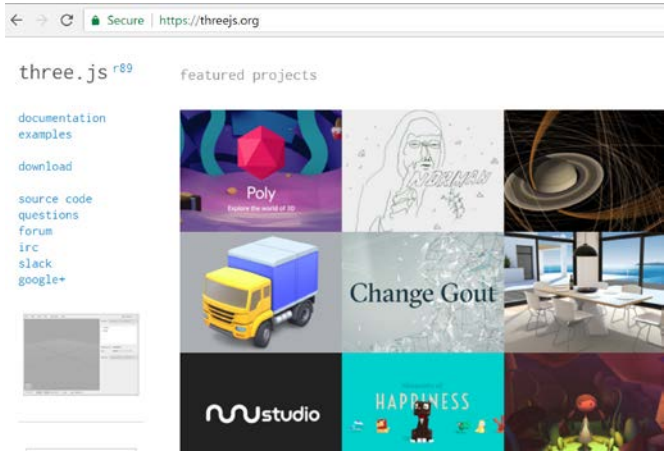
- **Making a Game**
  - Typically would use a game engine like Unity or Unreal

- **Making a Movie**
  - Renderman

- **3D Web App Development**
  - three.js which is built on WebGL

But to use three.js you need to understand WebGL

And, basic CG concepts need to be understood to use Unity or Renderman as well…



Class Mechanics

- Course Website: https://courses.engr.illinois.edu/cs418/index.html
  - Schedule, lecture materials, assignments
- Piazza: This term we will be using Piazza for class discussions https://piazza.com/illinois/spring2018/cs418/home.
- Grades available on Compass

Class Mechanics: Grades

- Machine Problem 1 15%
- Machine Problem 2 15%
- Machine Problem 3 15%
- Machine Problem 4 10%
- Exam 1 15%
- Exam 2 15%
- Exam 3 15%
- **No Final Exam**

Grading Scale

- Grades probably on usual scale:
  - 97 to 93: A
  - 93 to 90: A-
  - 90 to 87: B+
  - 87 to 83: B
  - 83 to 80: B-
  - …etc.
- I may adjust the intervals down…but not up
- **Extra Credit**: Show up for class.

  I'll take attendance in some manner 3 times randomly
  Each of those 3 classes is worth 0.5% of the total course grade

Course Policies

- MPs submitted after the due date lose 10% per day
- Discussing code is fine, copying code is not…
  If we discover plagiarized code, that code will receive a grade of 0
- **Do not use 3rd party code or copy and paste code you find randomly on the web**

  Just to be clear:

  Type the code yourself.
  If you are using existing code as reference, change it up when you type it (e.g. change the kind of loop used, what colors are used, etc.)
- In exceptional circumstances where extension may be reasonable (illness, family emergency etc.) arrangement must be made with the instructor
  e-mail: shaffer1@illinois.edu
- Exams are in the Computer-Based Testing Facility (CBTF)
- Post technical questions to Piazza

  Do not post your code visible to other students (**why?**)

  Do not expect us to debug your code (we will try to help….)

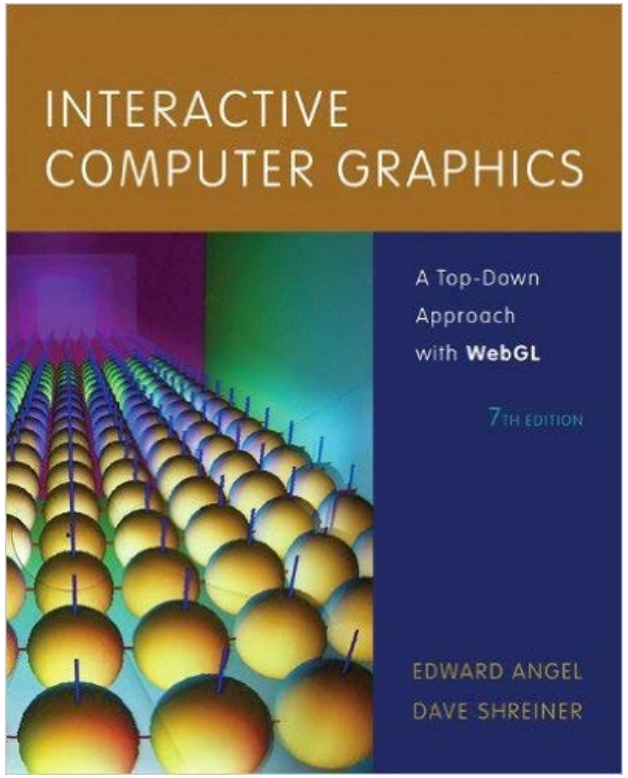
Class Mechanics: No Book

- We'll post notes online
  - It will save you $150

Language References and Resources

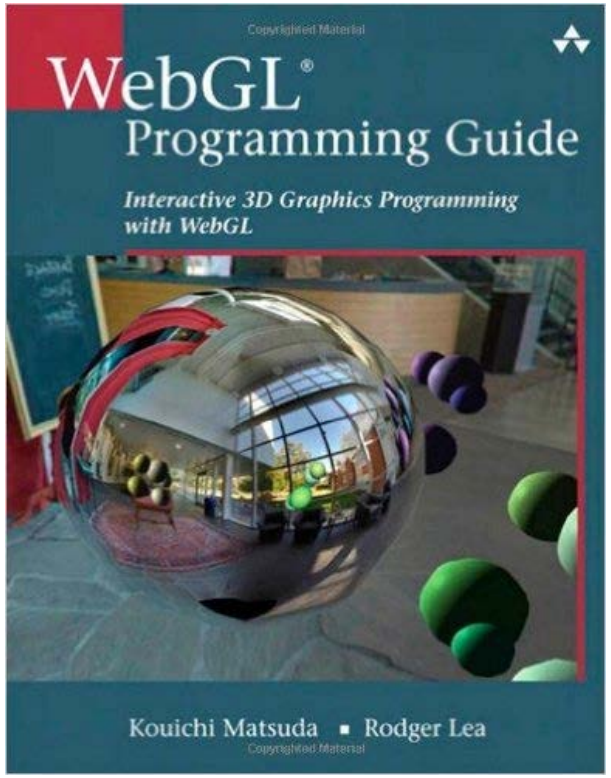
- JavaScript/HTML/CSS: https://developer.mozilla.org/en-US/docs/Web/JavaScript
- WebGL Specification: https://www.khronos.org/webgl/
- WebGL Tutorial: http://webglfundamentals.org/
- Suggested Editors: Brackets, LightTable
- Chrome DevTools Overview: https://developer.chrome.com/devtools

Suggested Books



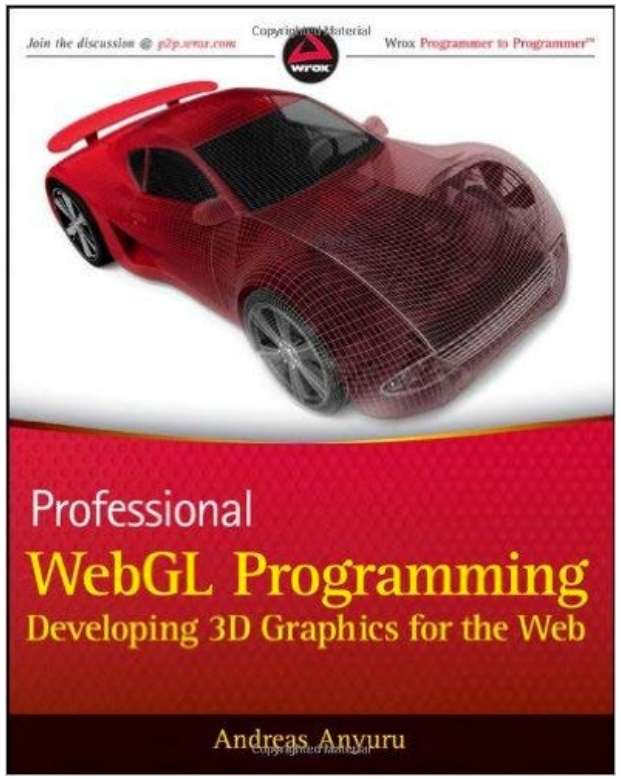
Interactive Computer Graphics: A Top-Down Approach with WebGL (7th Edition)
Mar 10, 2014
by Edward Angel and Dave Shreiner

Suggested Books



WebGL Programming Guide: Interactive 3D Graphics Programming with WebGL (OpenGL)Jul 19, 2013
by Kouichi Matsuda and Rodger Lea

Suggested Books



Professional WebGL Programming: Developing 3D Graphics for the Web
May 8, 2012
by Andreas Anyuru

Course Topics

- Real-time generation of 3D computer graphics through **rasterization**
- Low-level basic algorithms
  - Line-drawing
  - Hidden surface removal
  - Lighting and shading
  - Texturing
  - Scan conversion
- Using these capabilities in WebGL
- Modeling and viewing transformations
- Geometric modeling
- Animation

For Next Class

- If you have a laptop or your own PC
  - Install an editor (e.g. Brackets)
  - Install a browser supporting WebGL (e.g. Chrome)
  - Verify WebGL runs in that browser on your machine
  - https://courses.engr.illinois.edu/cs418/HelloColor.html
- If you don’t have your own computer, try an EWS lab