CIS367 - Computer Graphics

Image Models

Erik Fredericks - frederer@gvsu.edu

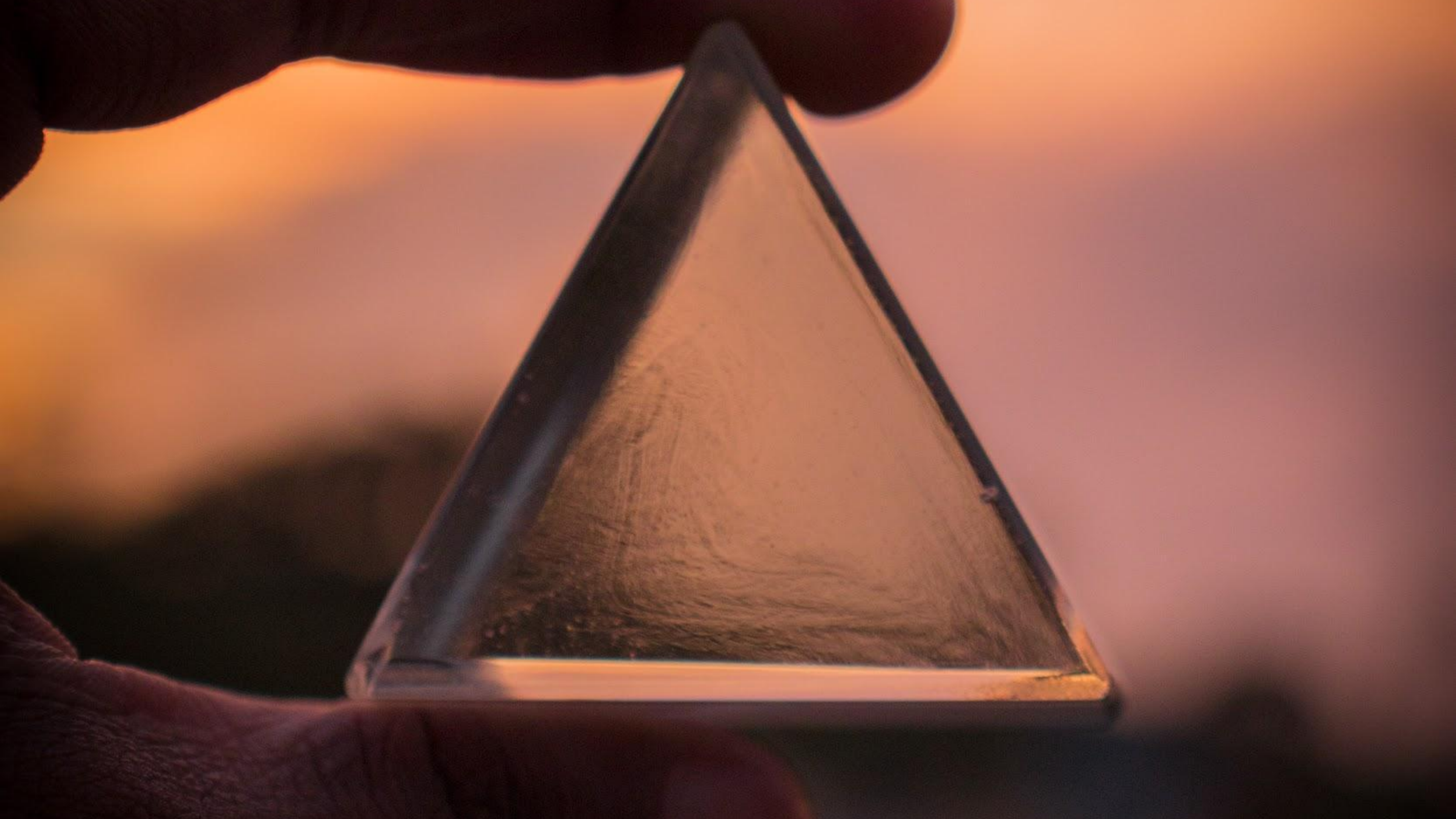
Overview

Pipelines, processing, triangles

OR

Models and architectures

... and triangle



First, the how

How do we make graphics?

We'll need an API

We're programmers after all

Application program interface (API) to specify:

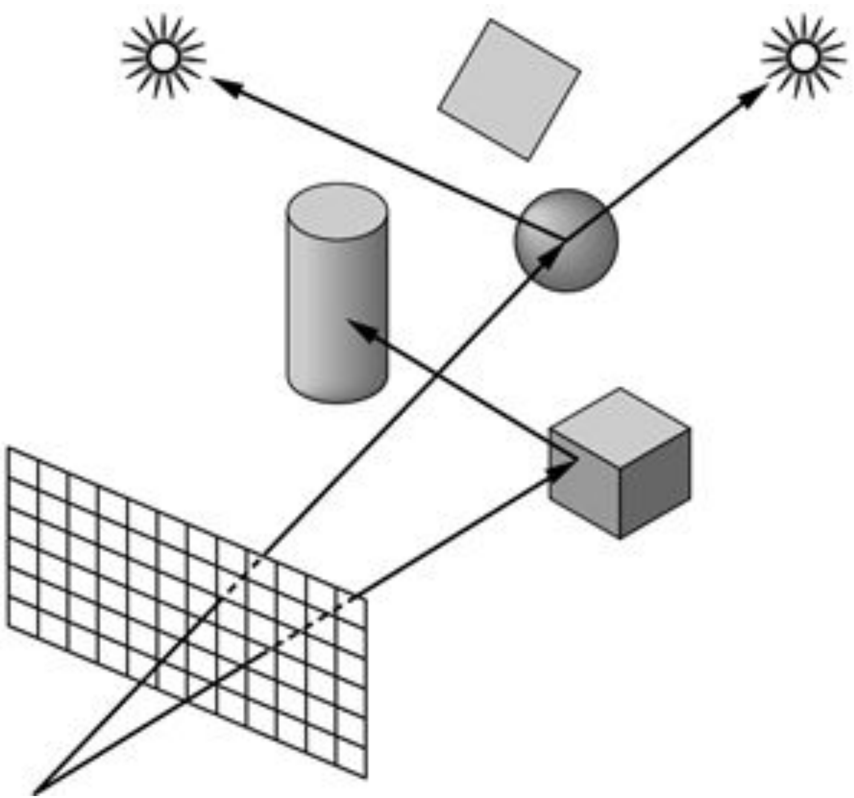
- Objects
- Lighting
- Camera(s)
- Materials
- etc.

What do we consider?

For physical approaches (e.g., ray tracing)...

- ALL rays of light
  - Path, velocity, etc.
  - Reflections
  - Translucency

- Database of all objects at all times
  - Slow!



Time to learn about the PIPELINE

In ➔ [something] ➔ [something else] ➔ … ➔ out

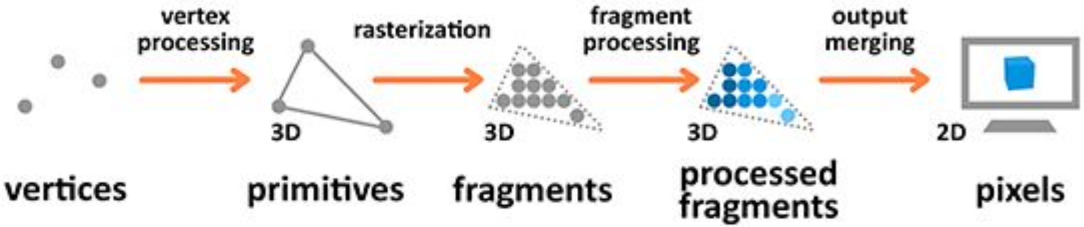
We will process objects one at a time

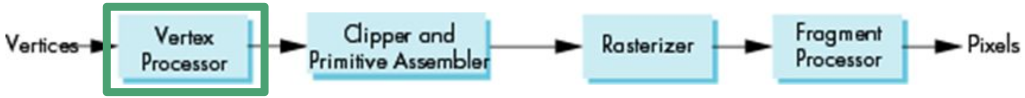
- Parallelizing helps
  - GPUs are … great with this approach

Each object is a graphical primitive

- Possibly millions of vertices **each**


Vertices
Vertex Processor
Clipper and Primitive Assembler
Rasterizer
Fragment Processor
Pixels


vertex processing
rasterization
fragment processing
output merging
3D
3D
3D
2D
vertices
primitives
fragments
processed fragments
pixels



Vertex processing

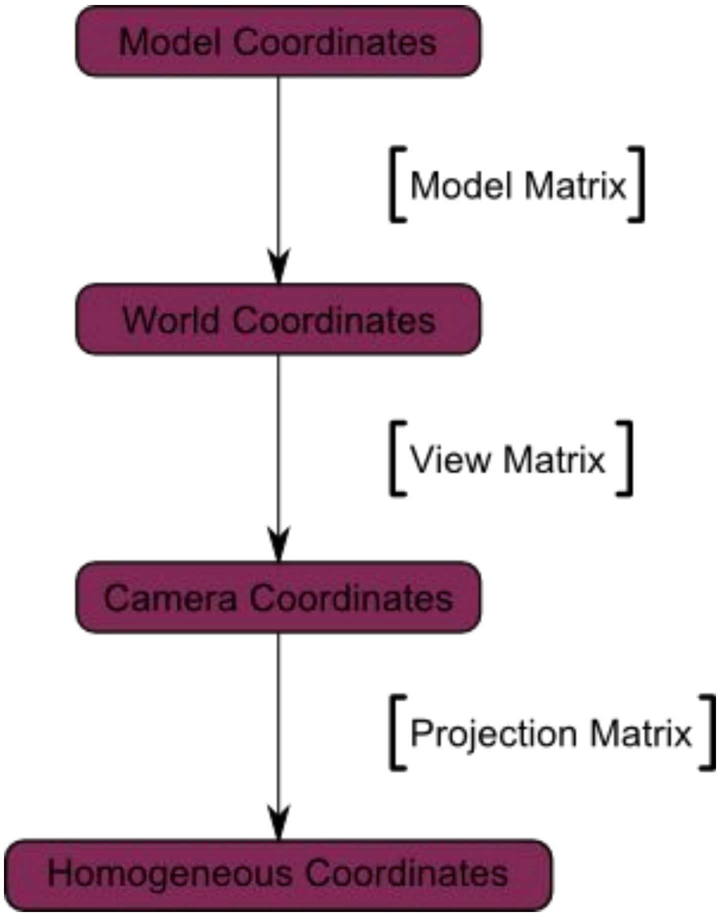
First block of the pipeline

- Coordinate transformations
- Each vertex processed **independently**
- May compute color / change vertex attributes

What takes up the majority of processing?

- Matrix math!
- Matrices are combined via **concatenation (multiplication)**
  - Changes in coordinate systems

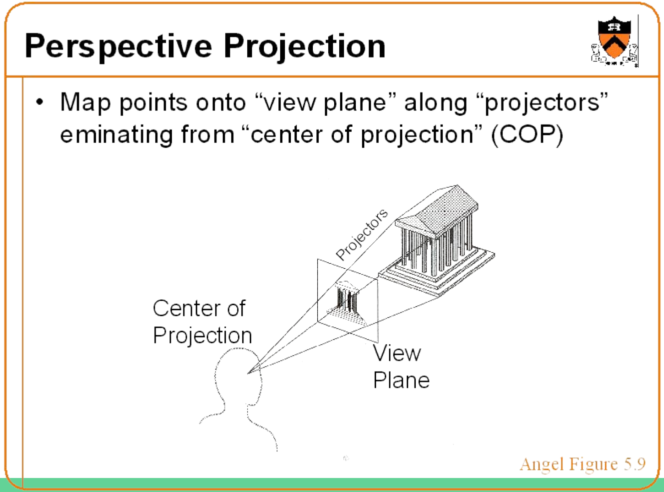
Projection transformation to 3D (plus others)

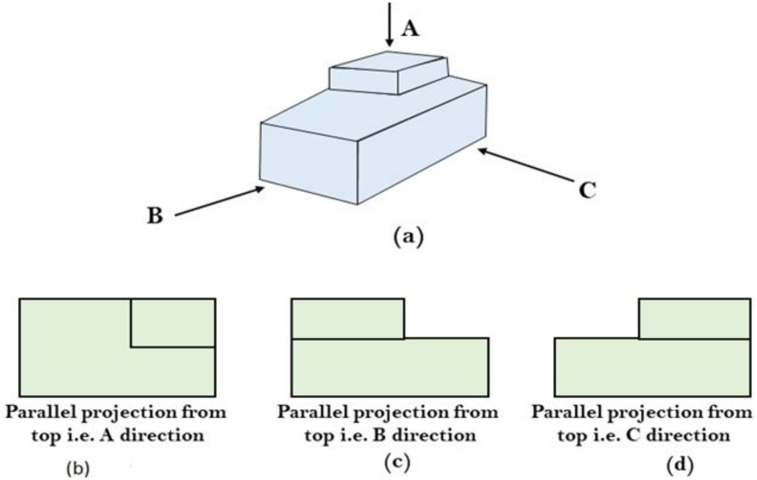

Model Coordinates
Model Matrix
World Coordinates
View Matrix
Camera Coordinates
Projection Matrix
Homogeneous Coordinates

Projection

Combines 3D view with 3D objects to make 2D image

- Perspective projection
  - All projectors meet at center of projection
- Parallel projection
  - Projectors parallel: center of project is replaced by direction



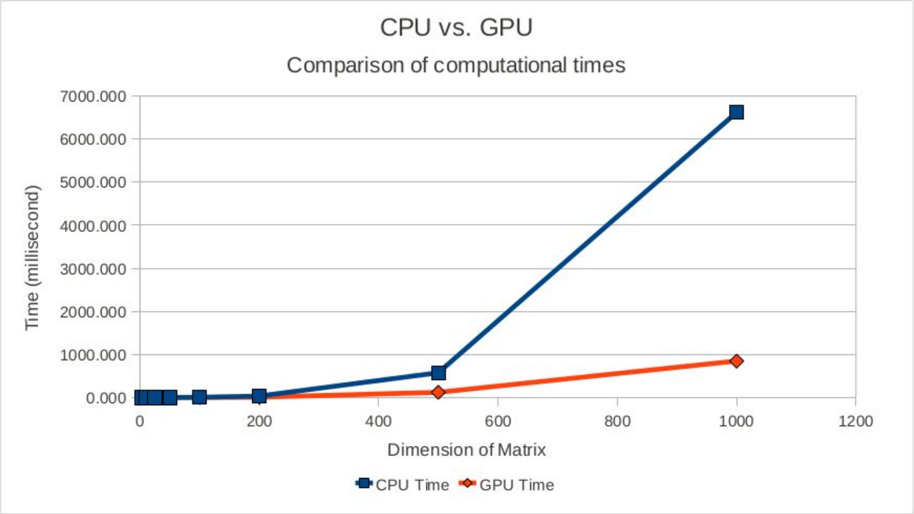


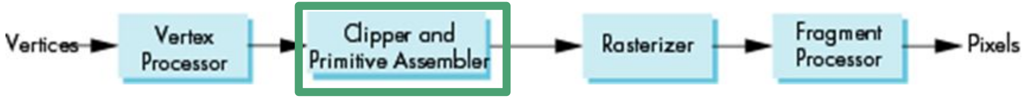
Vertex processing

What takes up the majority of processing?

Converting coordinate systems!

- Matrix transformations
- GPUs are **really** good at this
    - Like, really good

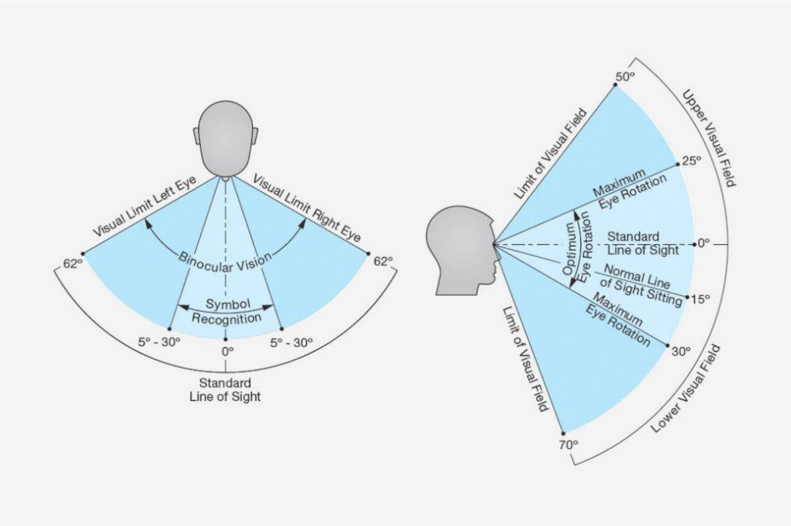


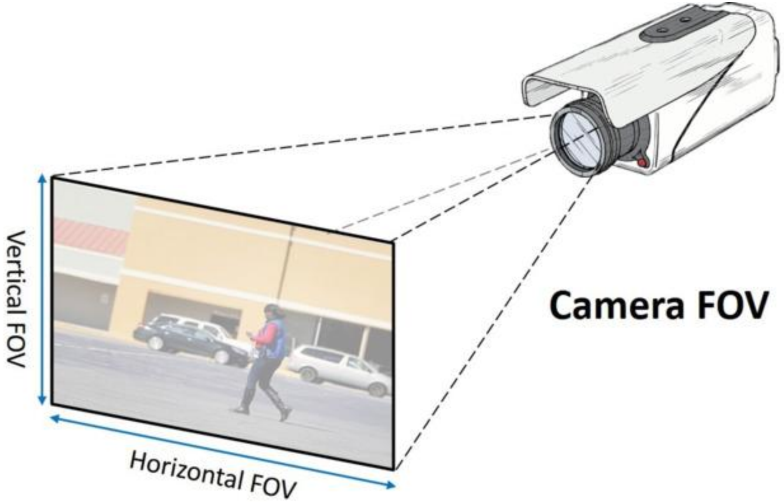


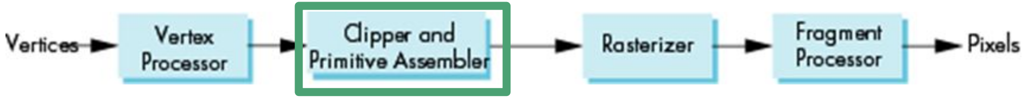
Clipping/Primitive Assembly

Clipping

- Can't see everything at once (so only draw what you can see!)
  - Helps to save processor power



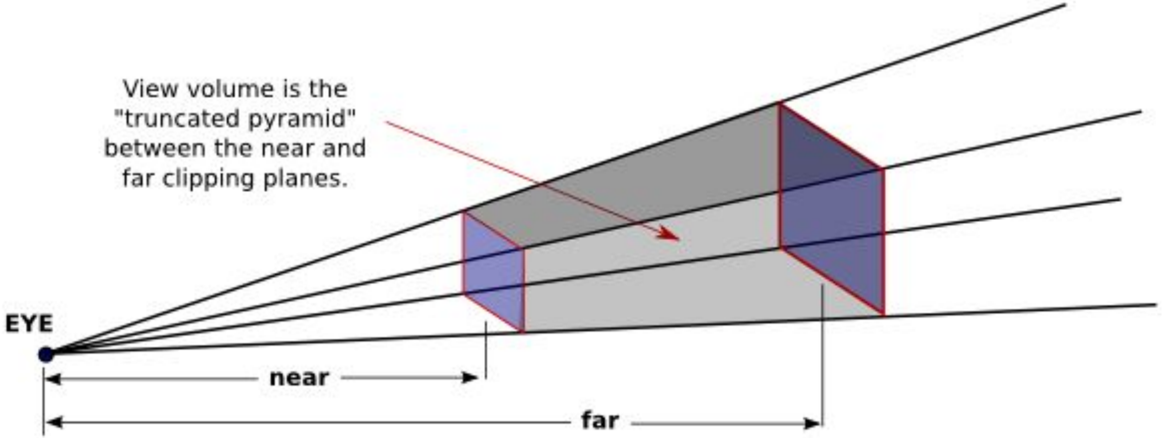


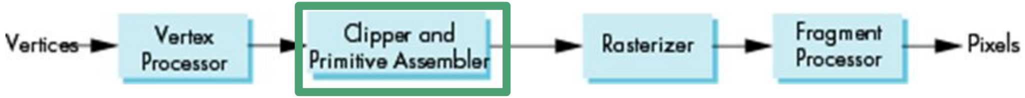


Clipping/Primitive Assembly

Find **clipping volume** of our camera

- Only images within this space are drawn
  - Including those partially in space

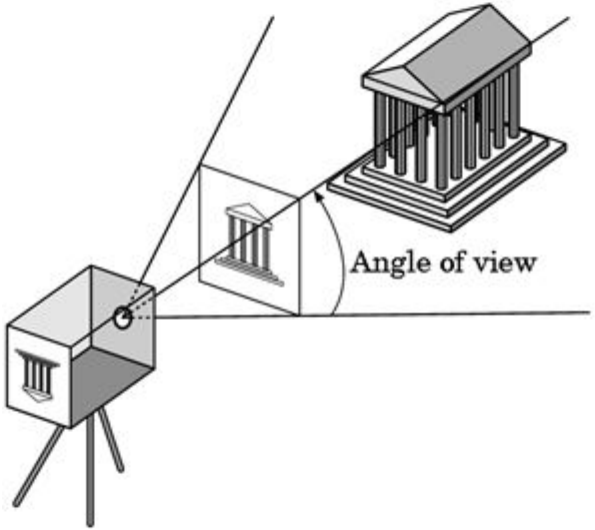


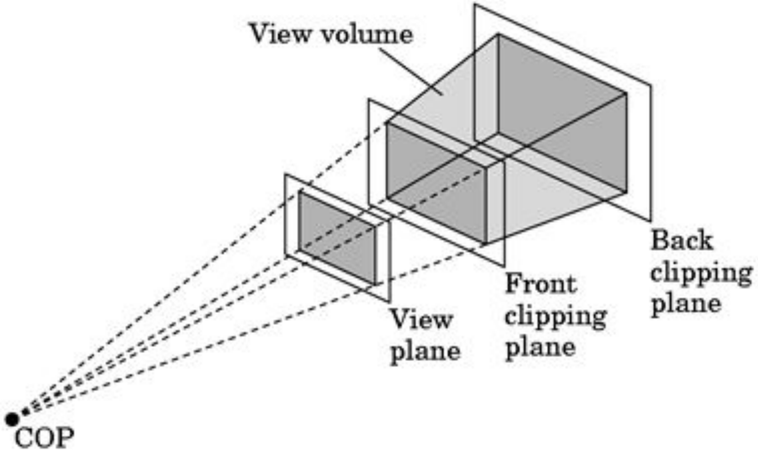


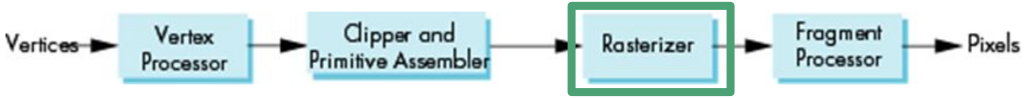
Clipping/Primitive Assembly

Vertices collected into geometric objects (prior to clipping)

- Line segments
- Polygons
- Curves / surfaces







Rasterization

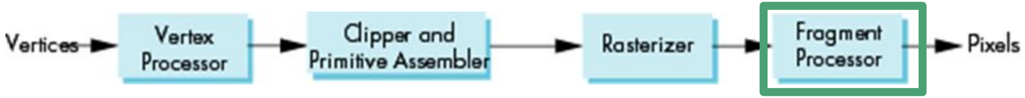
Clipping ➔ vertices

Rasterization converts to pixels for framebuffer

- Output is set of **fragments** for each primitive in clipping plane
- Fragment = potential pixel + information (e.g., color, location, depth)

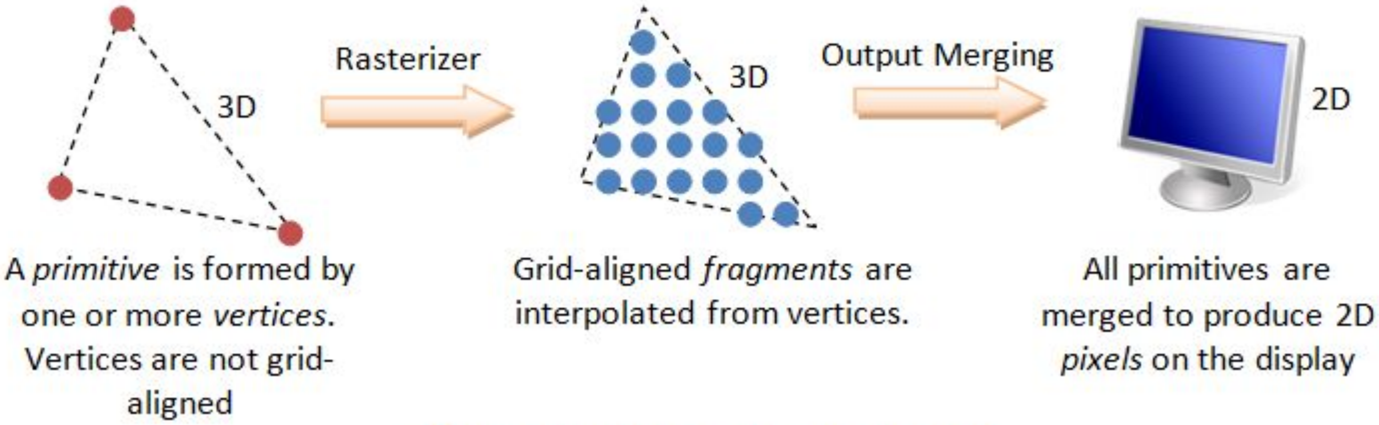
Example:

- Assembler specifies vertices of some object (triangle, square, etc.)
- Rasterizer determines which **pixels** comprise object



Fragment Processor

Fragments from rasterizer are assembled
Deals with visibility issues in 3D space
Performs blending with other colors / alpha transparency

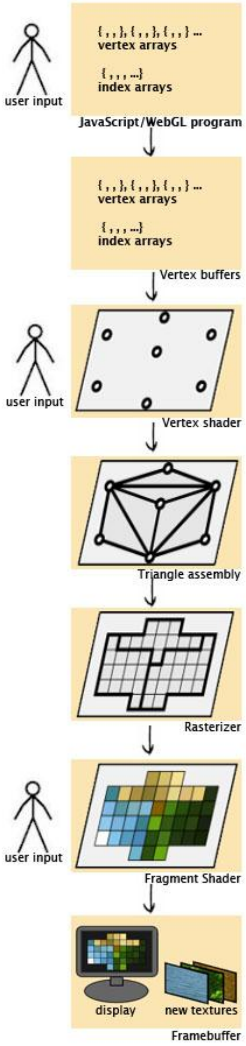


Vertex, Primitives, Fragment and Pixel

The Pipeline

More on the pipeline

https://dev.opera.com/articles/introduction-to-webgl-part-1/rendering-pipeline.jpg



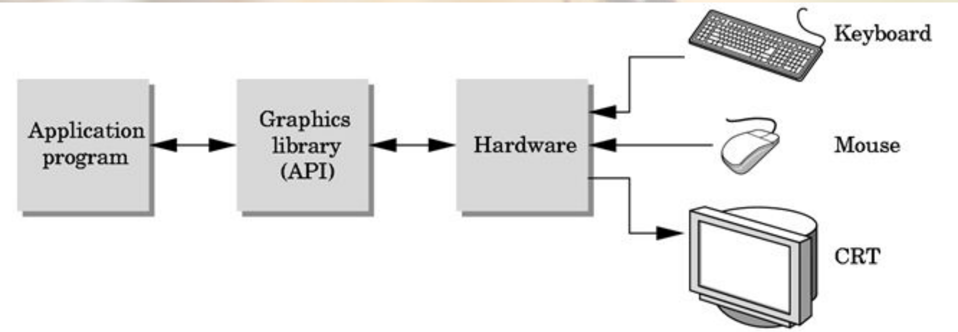
API

Application programmer interface

- How programmers interact with a graphics system (graphics library)

Common examples:

- SDL
- OpenGL / WebGL
- pyGame



API

Functions and libraries necessary to draw images to screen

- Objects
- Viewer(s)
- Light sources / cameras
- Materials
- Animation

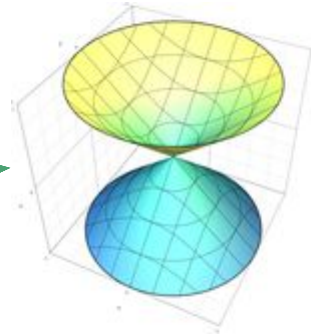
Other support

- Capture user input
- Audio
- Understand system capabilities

Object Specification

Most APIs support some manner of **primitive**:

- Points (0D object)
- Line segments (1D objects)
- Polygons (2D objects)
- Curves/surfaces
  - Quadrics
  - Parametric polynomials



Generally defined via **vertices** (location in space)

Polygon Example (old style -- pre-GPU)

Object type

Vertex location

```
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 (new style -- GPU enabled)

Geometric data placed in array

```
var points = [
  vec3(0.0, 0,0 0.0);
  vec3(0.0, 1.0, 0.0);
  vec3(0.0, 0.0, 1.0);
];
```

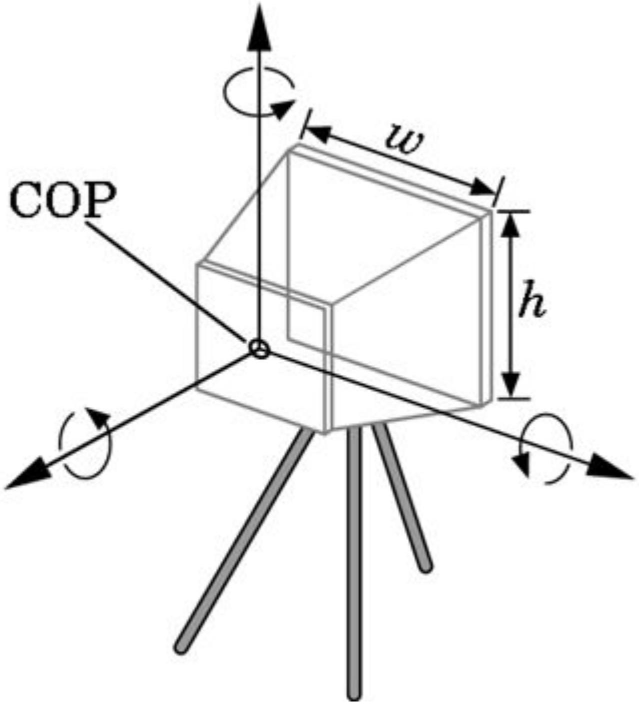
Then, send array to GPU (via function calls)
GPU renders triangle

Camera

6 degrees of freedom

- Center of lens position
- Orientation

Lens
Film size
Film plane orientation



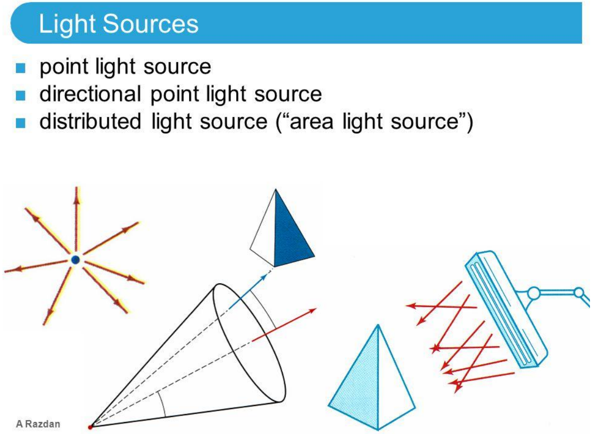
Lights and Materials

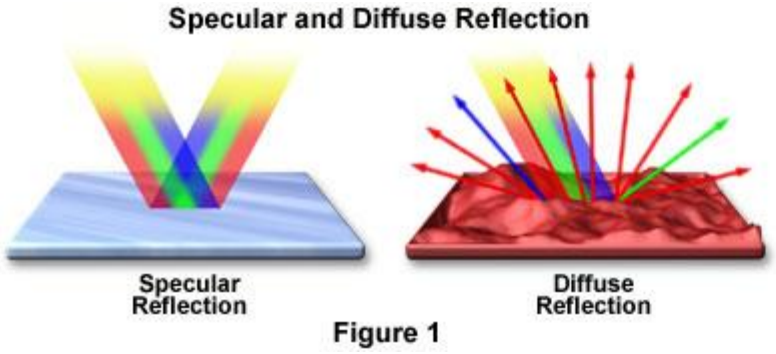
Lighting types

- Point vs. distributed
- Spotlights
- Near vs. far sources
- Color properties

Material properties

- Absorption: color properties
- Scattering
  - Diffuse
  - Specular





EOS demo

https://cislab.hpc.gvsu.edu/

https://www.cis.gvsu.edu/ciscomputinglabs/remote-connections/

https://hpcsupport.atlassian.net/servicedesk/customer/portal/3/topic/d44433c8-596b-404e-8954-bf66c810e72c/article/341213229

What I do:

ssh yourGVSUusername@ ~~arch01.cis.gvsu.edu~~

ssh yourGVSUusername@eos19.cis.gvsu.edu   >_<

Some Linux commands

mv - move file
cd - change directory
rm - delete file (THERE IS NO RECYCLE BIN FYI!)

—

To get the Common folder in your directory:

```
cd /WEB_STUDENT/<your username>
git clone https://github.com/esangel/WebGL.git
mv WebGL/Common .
```

Your site

Your **home/website directory** on your server is /WEB_STUDENT/<your-username>

- For instance, if you added a `test.html` page, you could see that at

  `https://student.computing.gvsu.edu/<your-username>`

Step 8 - Upload a file

On **your computer**, create a file called test.html:

```
<html>
<head>
  <title>My first site</title>
</head>
<body>
  <h1>This is my site.  Neat</h1>
</body>
</html>
```

Step 8 - Upload File / Move File

In your **SSH** windo ... ar icon and then cl ... e'

- Pick your ne ... file and i ... **r home directory**
  - So, it went to /home
- If you type ls, it will list ... directory
  - Or more specific:
- We need to ... directory!

```
sudo mv t ... ml /var/www/html/.
```

sudo not always necessary ➔ but our web directory is owned by the admin

Step 8 - Upload File

Either you could have created that file on EOS or you need to upload it to your directory.

Possibilities:

- Use SCP / sFTP
- Be on a school computer and copy it to that directory (/WEB_STUDENT/<your username>)

Step 9 - Check it out!

http://nehe.gamedev.net/

A ponderable:

Not worrying about color at the moment, how do you think we could make this with our current tools?