OpenGL and Graphics Hardware

Overview of OpenGL
Pipeline Architecture
Alternatives

Administration

- Future classes in Wean 5409
- Web page off my home page:
  `http://www.cs.cmu.edu/~djames/15-462/Fall03`
- Assignment #1 out next week (Tuesday)

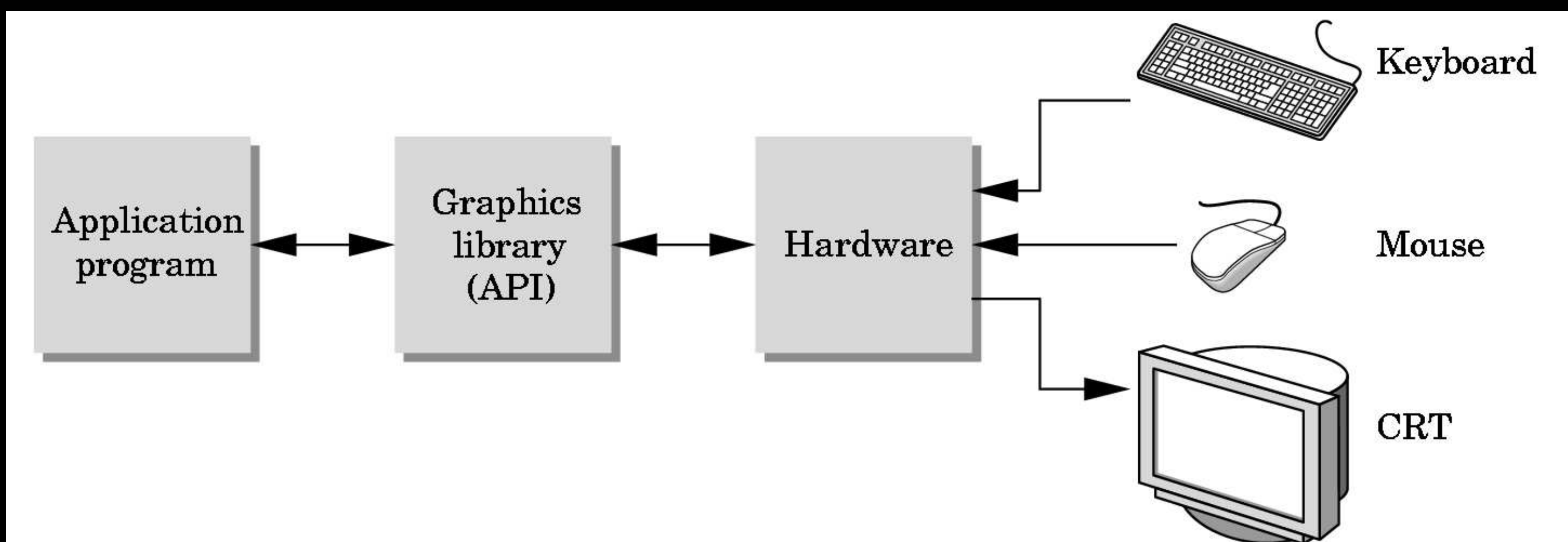
Question

- How many of you have programmed in OpenGL?

- How extensively?

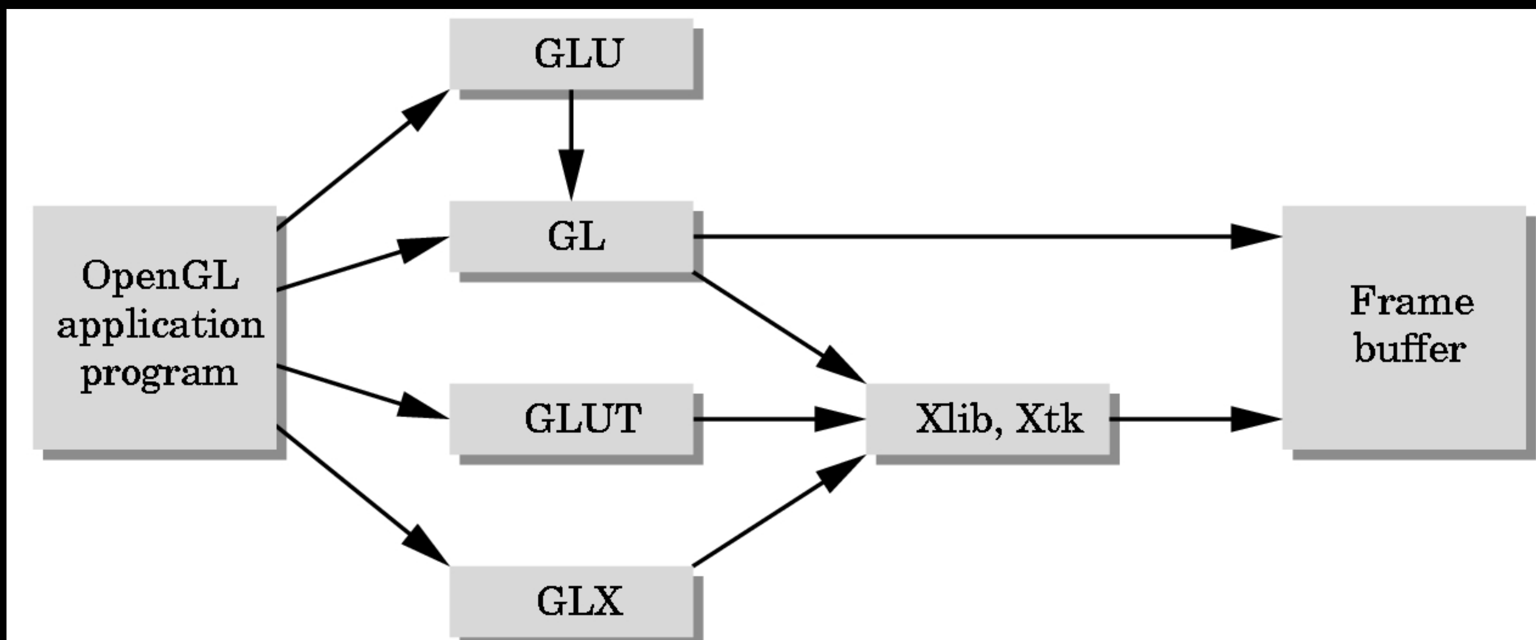
What is OpenGL?

- A low-level graphics API for 2D and 3D interactive graphics. OS independent.
- Descendent of GL (from SGI)
- Implementations: For the Linux PCs we have Mesa, a freeware implementation.



What OpenGL isn't:

- A windowing program or input driver, since those couldn't be OS independent.



GL: core graphics capability
GLU: utilities on top of GL
GLUT: input and windowing functions

How does it work?

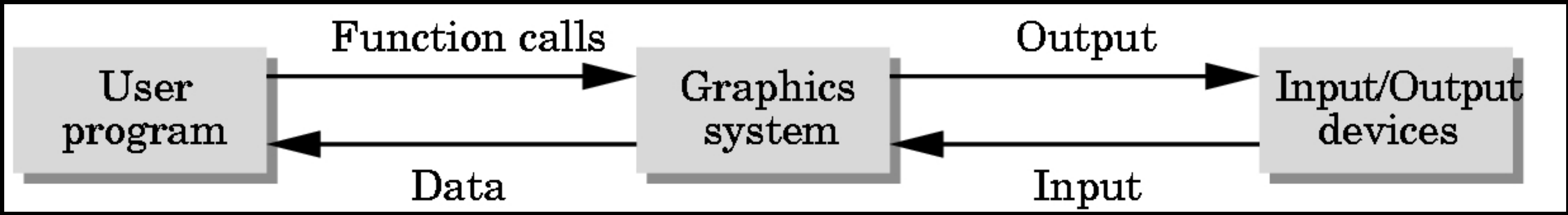
- **From the programmer's point of view:**

  Specify geometric objects

  Describe object properties

  Define how they should be viewed

  Move camera or objects around for animation



How does it work?

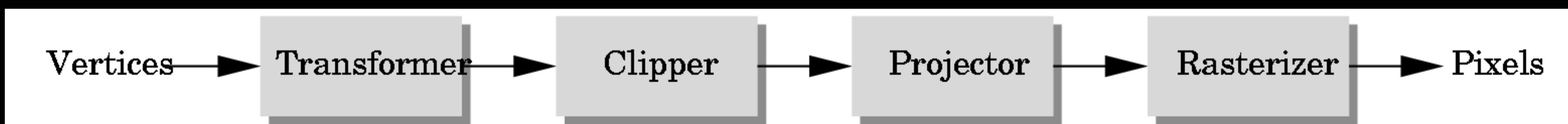
State machine with input and output:

- State variables: color, current viewing position, line width, material properties, ...
- These variables (the state) then apply to every subsequent drawing command
- Input is description of geometric object
- Output is pixels sent to the display

How does it work?

From the implementor's perspective:

OpenGL pipeline



Primitives + material properties	**Rotate Translate Scale**	**Is it visible?**	**3D to 2D**	**Convert to pixels**	**Display**

Let's walk through the pipeline…

Primitives: drawing a polygon

- Put GL into draw-polygon state

```
glBegin(GL_POLYGON);
```

- Send it the points making up the polygon

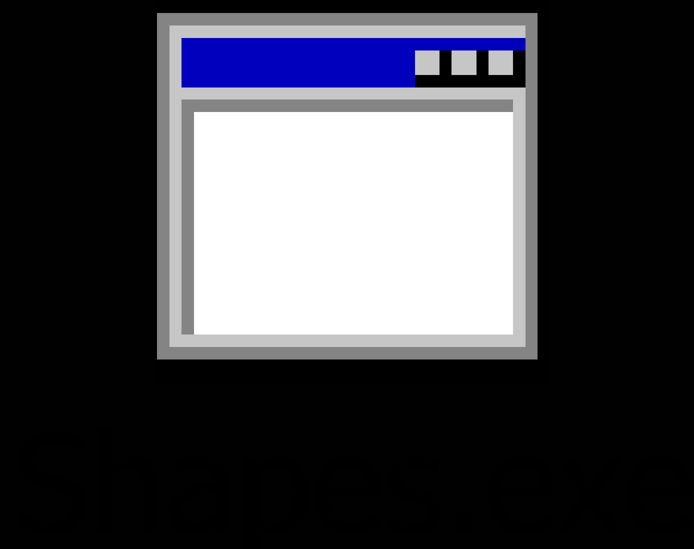
```
glVertex2f(x0, y0);
glVertex2f(x1, y1);
glVertex2f(x2, y2) ...
```

- Tell it we’re finished

```
glEnd();
```

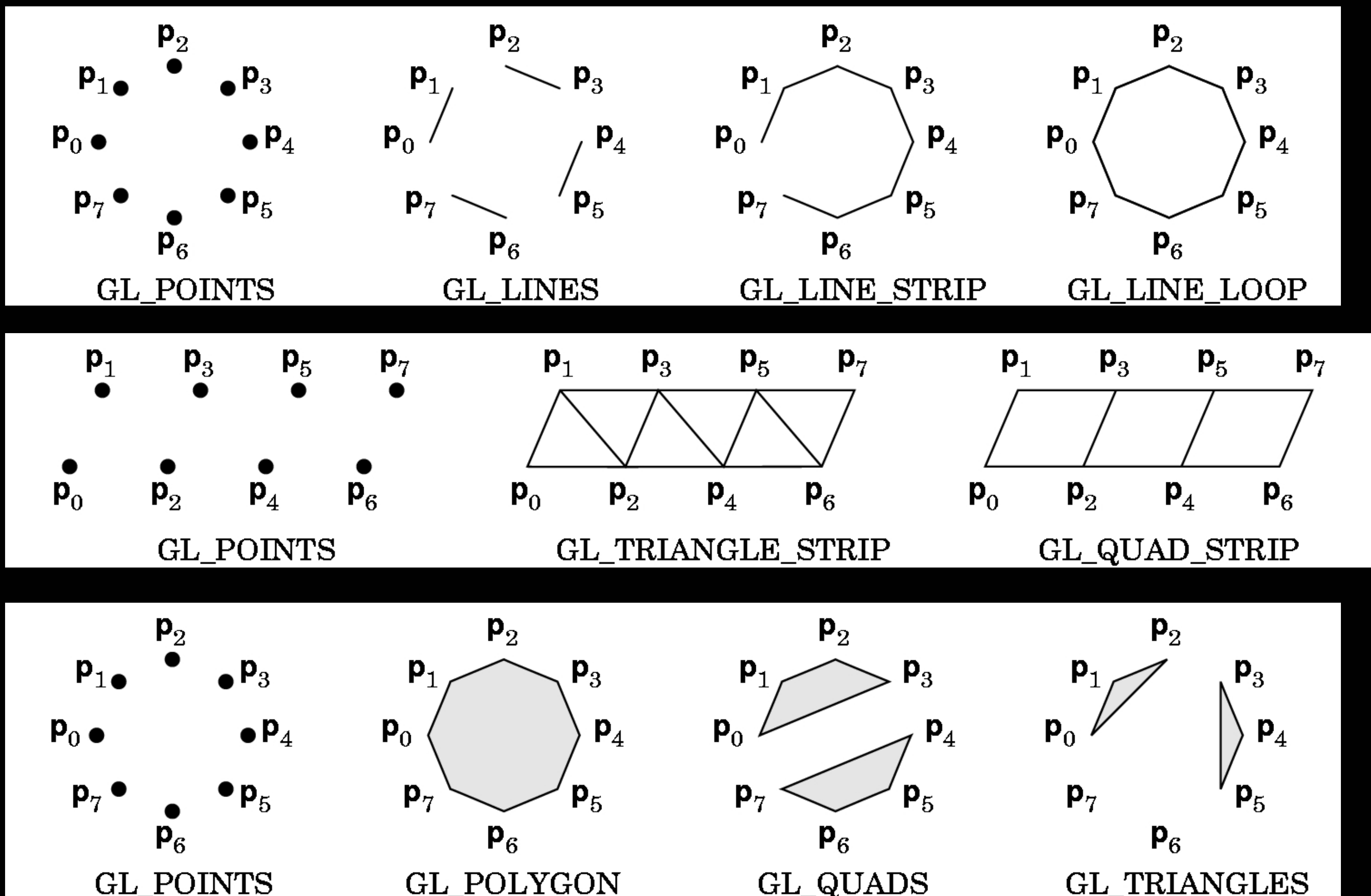
Build models in appropriate units (microns, meters, etc.). Transform to screen coordinates (pixels) later.

Specifying Primitives



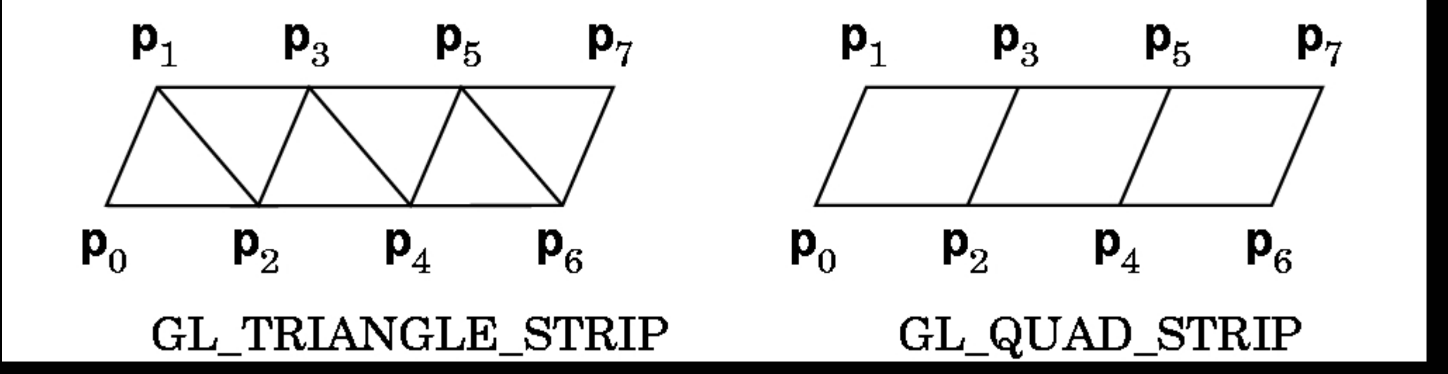
Code for all of today’s examples available from http://www.xmission.com/~nate/tutors.html

Primitives: points, lines, polygons



Primitives: points, lines, polygons

- Why triangles, quads, and strips?



- Hardware may be more efficient for triangles
- Strips require processing less data
  – fewer glVertex calls

Primitives: Material Properties

- **`glColor3f(r,g,b);`**

  All subsequent primitives will be this color.

  Colors are not attached to objects.

  `glColor3f(r,g,b)` *changes the system state.*

  Everyone who learns GL gets bitten by this!

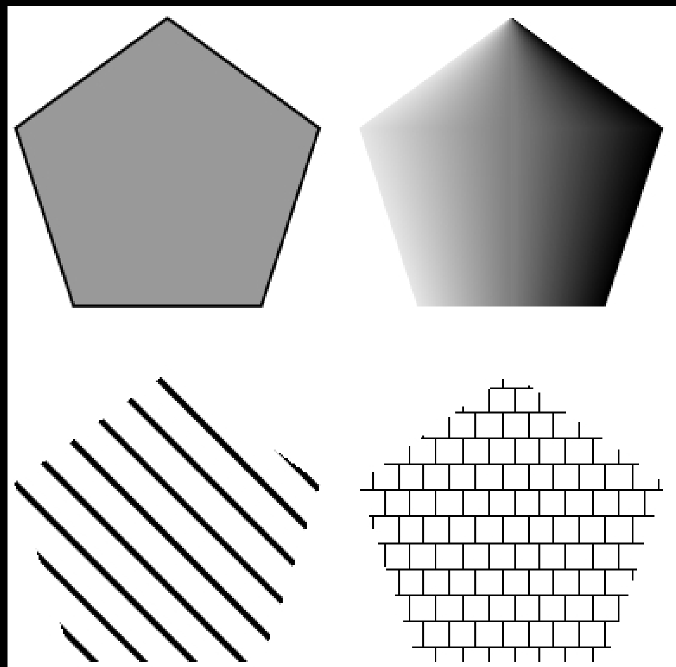
  Red, green & blue color model.

  Components are from 0-1.

Primitives: Material Properties

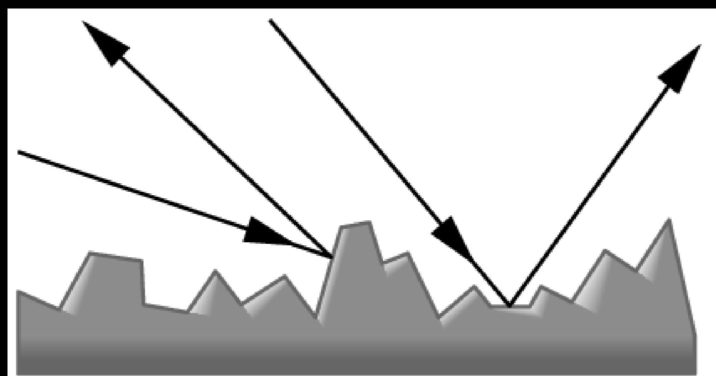
Many other material properties available:

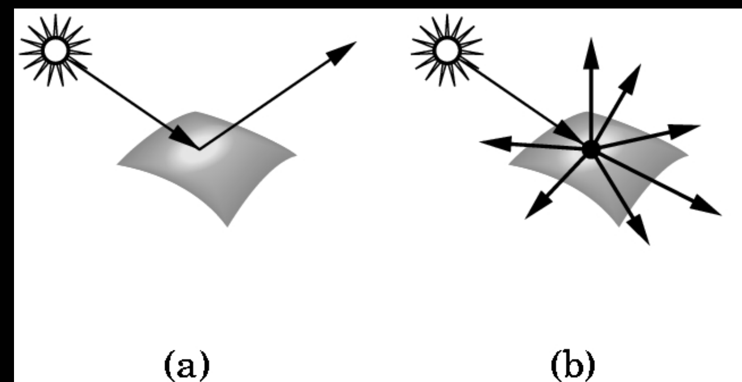
```
glEnable(GL_POLYGON_STIPPLE);
glPolygonStipple(MASK); /* 32x32 pattern of bits */
…
glDisable (GL_POLYGON_STIPPLE);
```



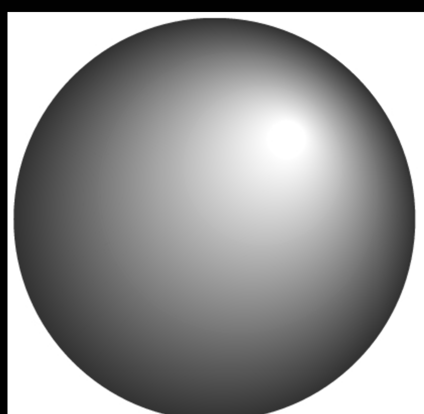
Primitives: Material Properties

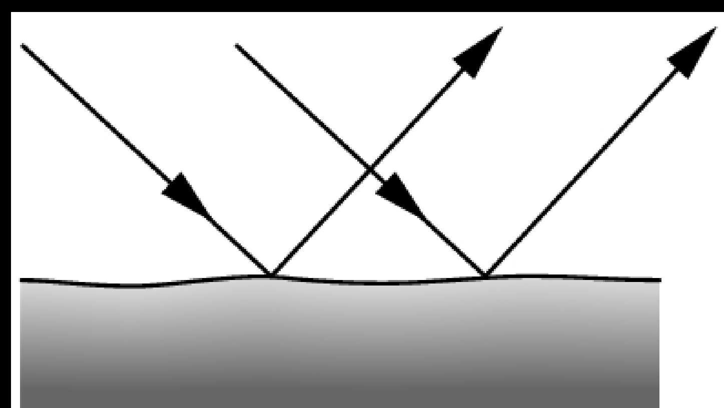
- **Ambient**: same at every point on the surface
- **Diffuse**: scattered light independent of angle (rough)





- **Specular**: dependent on angle (shiny)

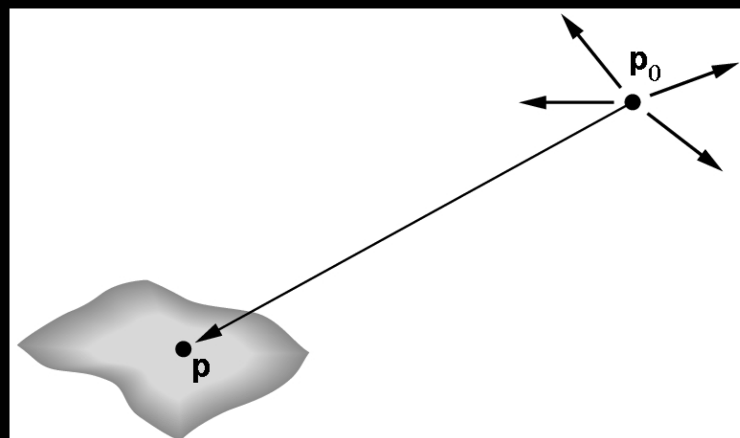




lightmaterial.exe

Light Sources

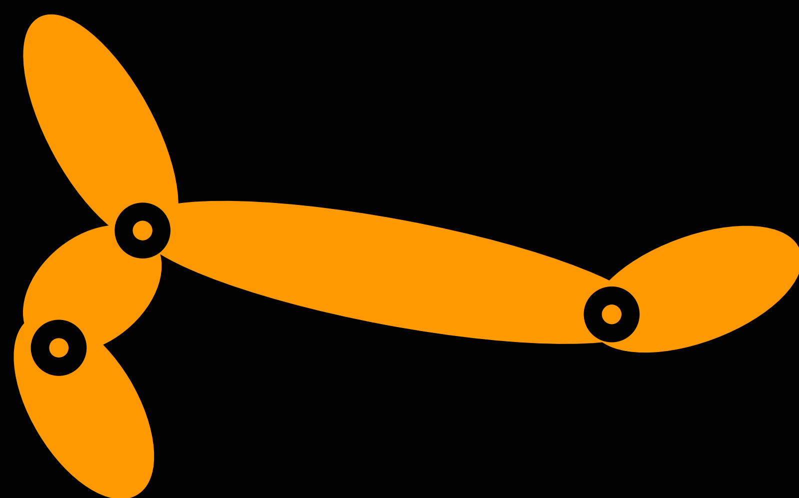
- **Point light sources** are common:



lightpositions.exe

Transforms

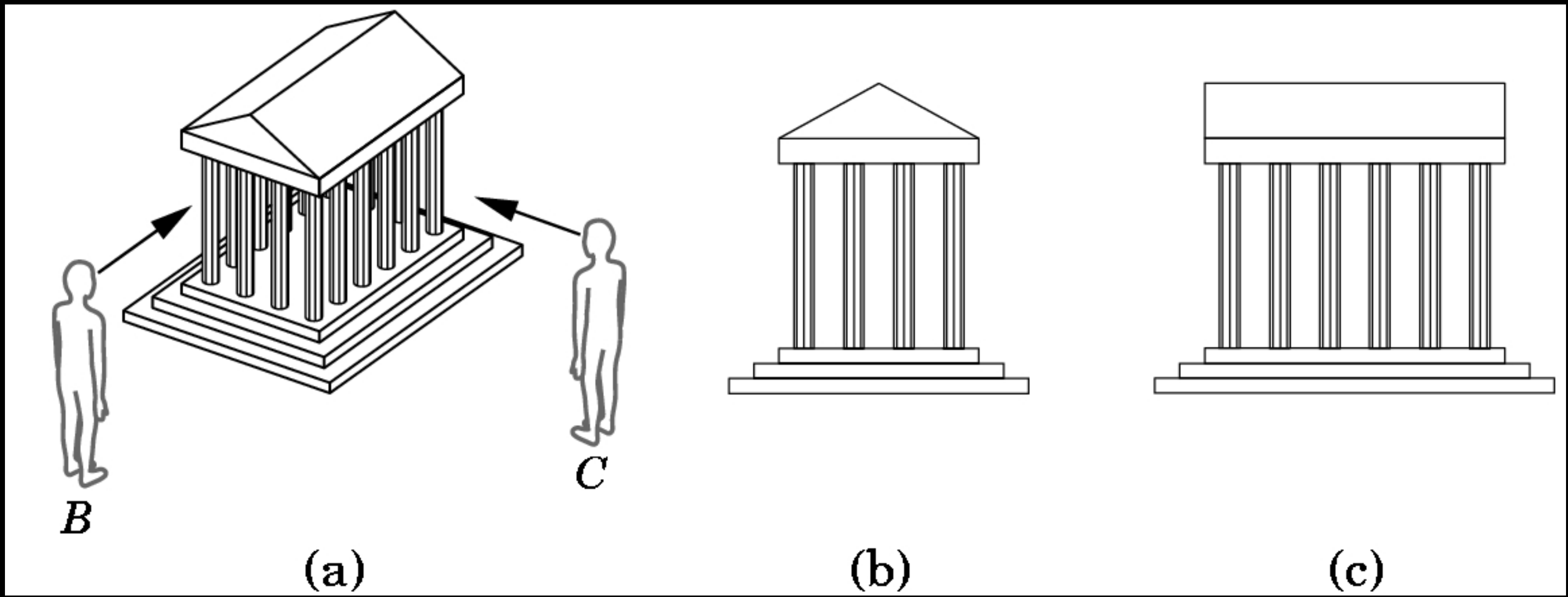
- Rotate
- Translate
- Scale
- glPushMatrix(); glPopMatrix();



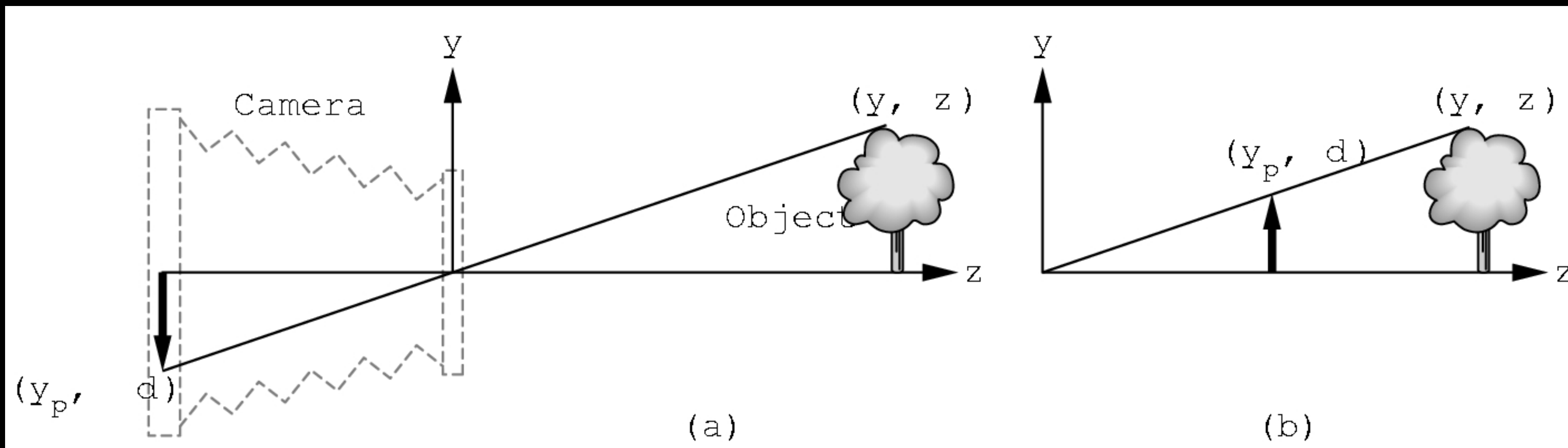
transformations.exe

Camera Views

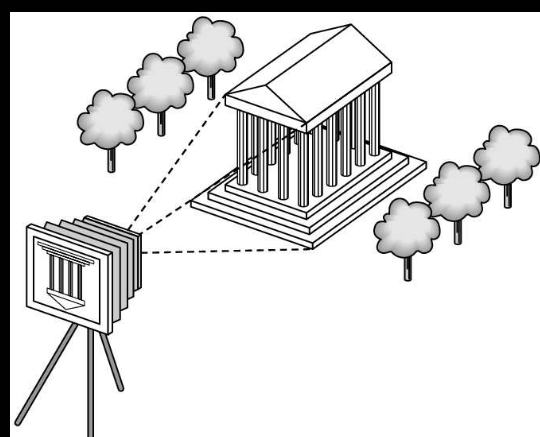
Different views of an object in the world

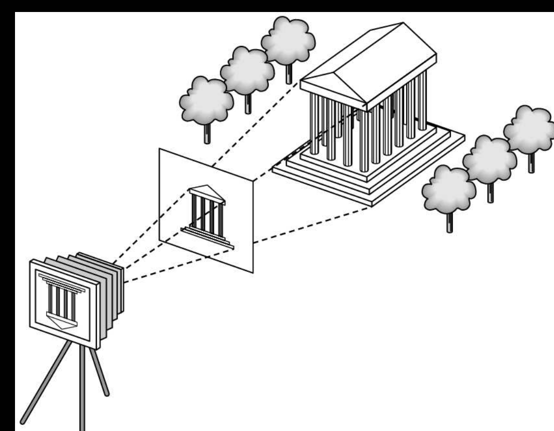


Camera Views

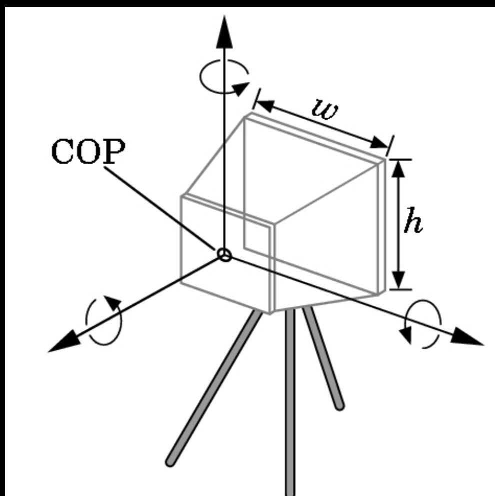


Lines from each point on the image are drawn through the center of the camera lens (the ***center of projection*** (COP)).





Camera Views



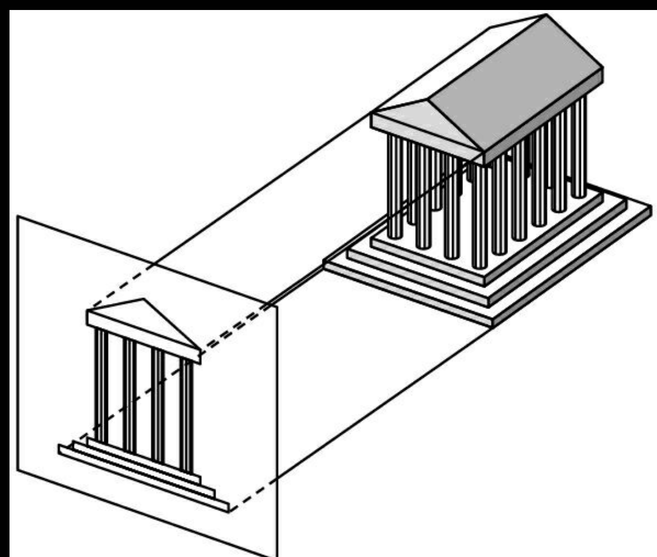
Many camera parameters…

For a physical camera:

- position (3)
- orientation (3)
- lens (field of view)

Camera Projections

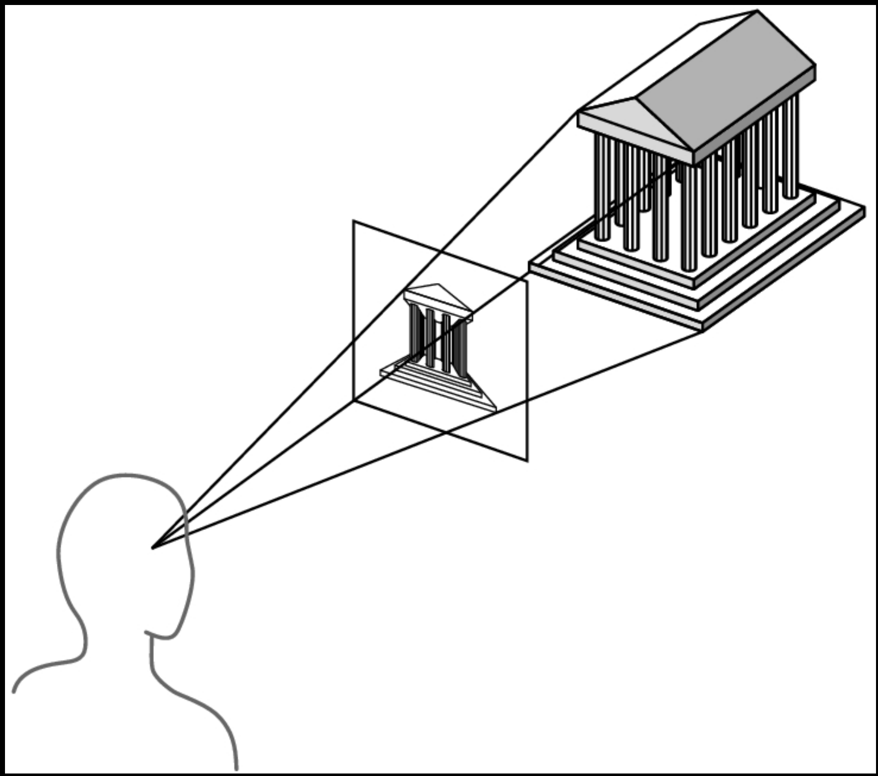
- **Orthographic projection**
  - long telephoto lens.
- Flat but preserving distances and shapes. All the projectors are now parallel.



- **glOrtho (left, right, bottom, top, near, far);**

Camera Projections

- **Perspective projection**
- Example: pin hole camera
- Objects farther away are smaller in size

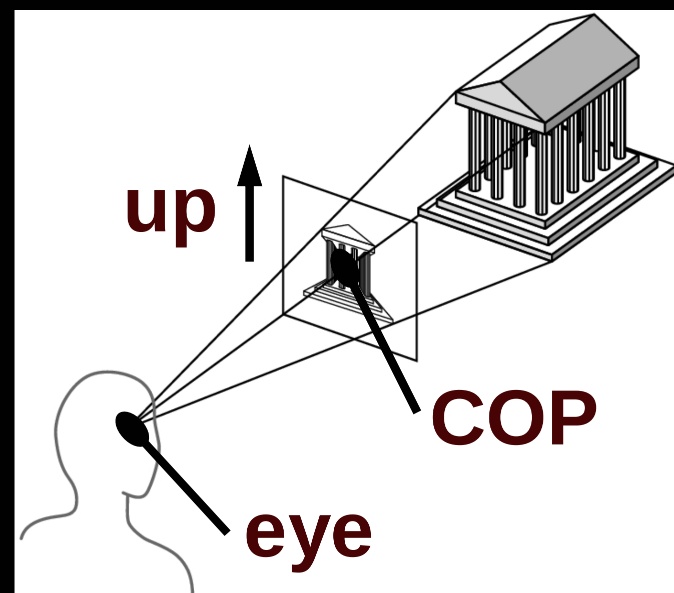


Camera Transformations

- Camera positioning just results in more transformations on the objects:
  - Transformations that position the object relative to the camera

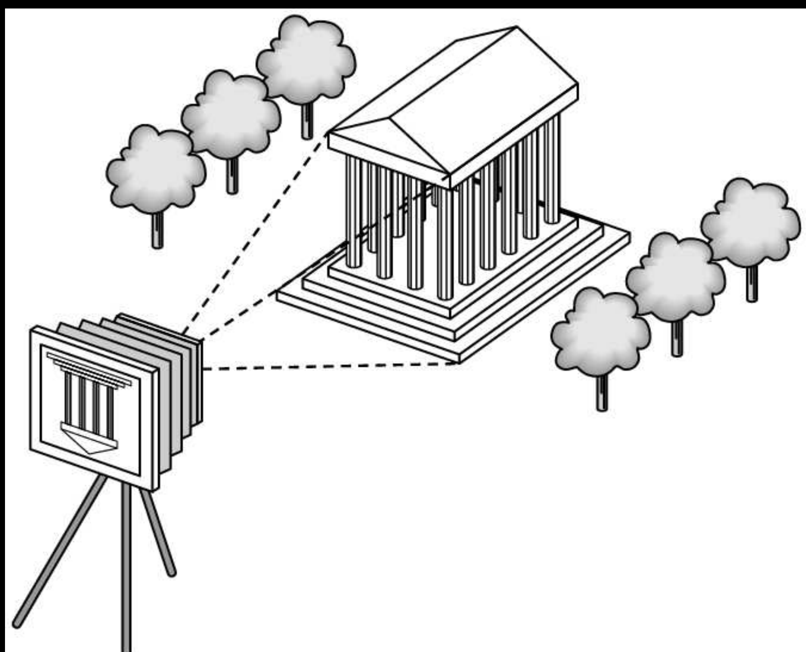
- Example:

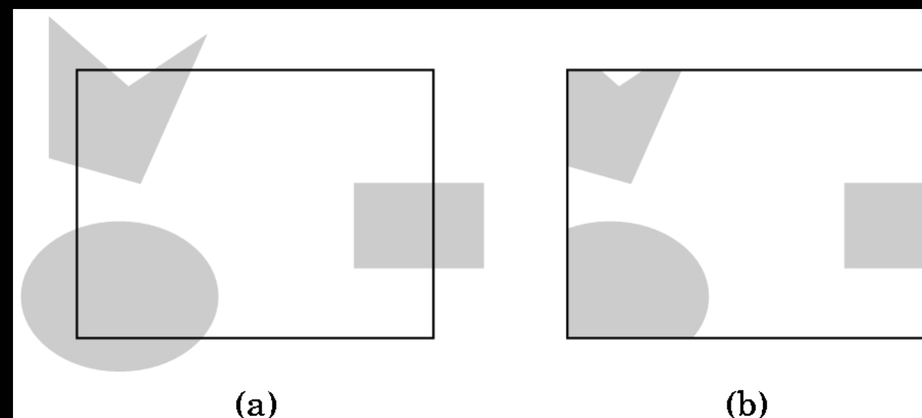
```
void gluLookAt
(eyex,eyey,eyez,
 centerx,centery,centerz,
 upx,upy,upz )
```



Clipping

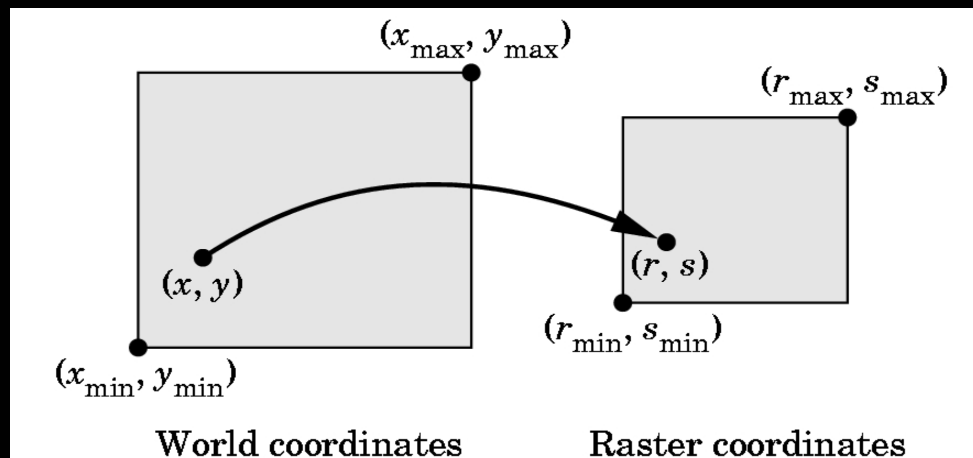
Not everything is visible on the screen

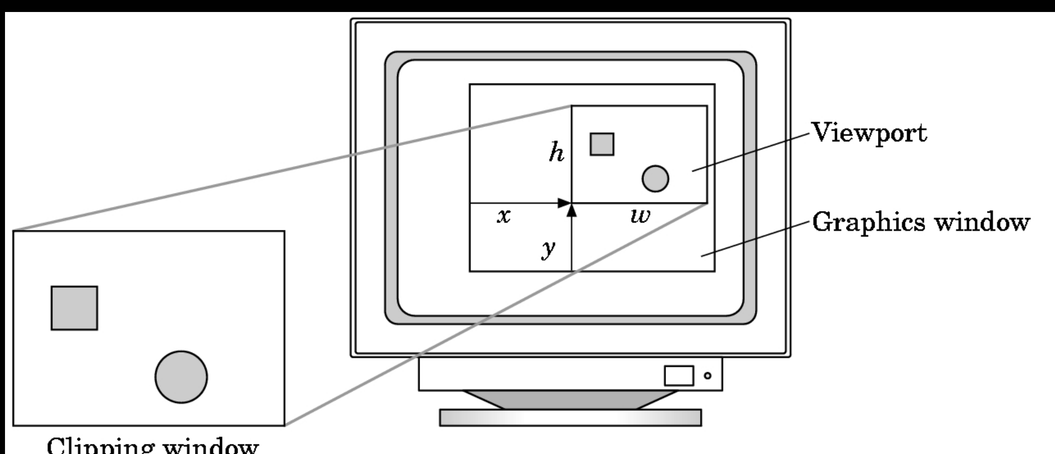


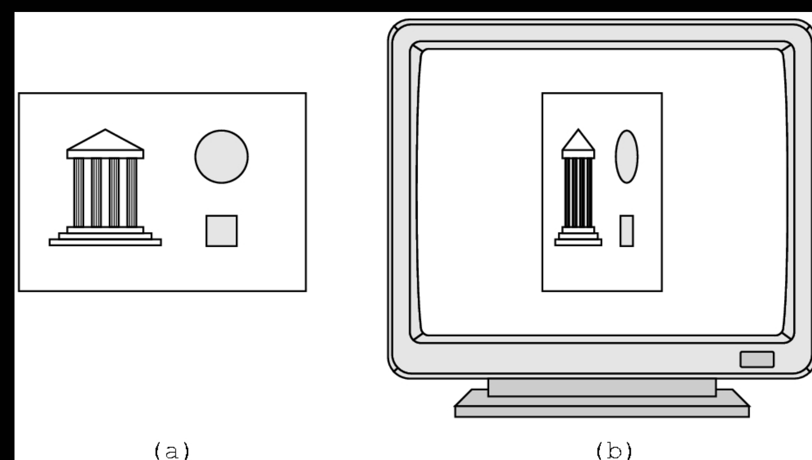


Rasterizer

- Transforms pixel values in world coordinates to pixel values in screen coordinates







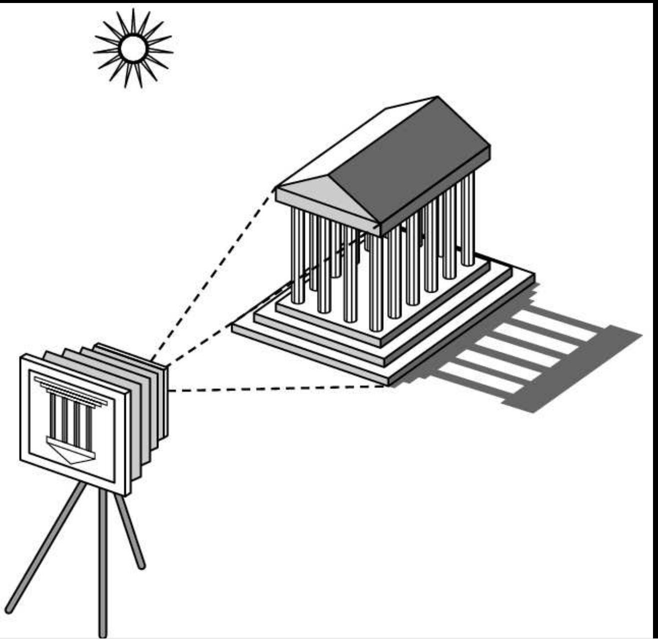
Special Tricks

- Gouraud Shading:

  Change the color between setting each vertex, and GL will smooth-shade between the different vertex colors.

- Shadows on ground plane:

  Render from the position of the light source and create a **shadow map**



- Fog (fog.exe)

Drawing A Box

```
void DrawBox()
{
    MakeWindow("Box", 400, 400);

    glOrtho(-1, 1, -1, 1, -1, 1);

    glClearColor(0.5, 0.5, 0.5, 1);
    glClear(GL_COLOR_BUFFER_BIT);

    glColor3f(1.0, 0.0, 0.0);

    glBegin(GL_POLYGON);
    /* or GL_LINES or GL_POINTS... */

    glVertex2f(-0.5, -0.5);
    glVertex2f( 0.5, -0.5);
    glVertex2f( 0.5,  0.5);
    glVertex2f(-0.5,  0.5);

    glEnd();
}
```

Setting up the window

- The coordinate system

  glOrtho(left, right, bottom, top, near, far);

  - e.g., glOrtho(0, 100, 0, 100, -1, 1);

  For now, near & far should always be -1 & 1

- Clearing the screen

  glClearColor(r, g, b, a);

  - a is the alpha channel; set this to 0.

  glClear(GL_COLOR_BUFFER_BIT);

  - glClear can clear other buffers as well, but we're only using the color buffer...

Getting Started

- Example Code

  We will give you example code for each assignment.

  Modifying existing code is much easier than writing “hello world” (unfortunately)

- Documentation:

  Book

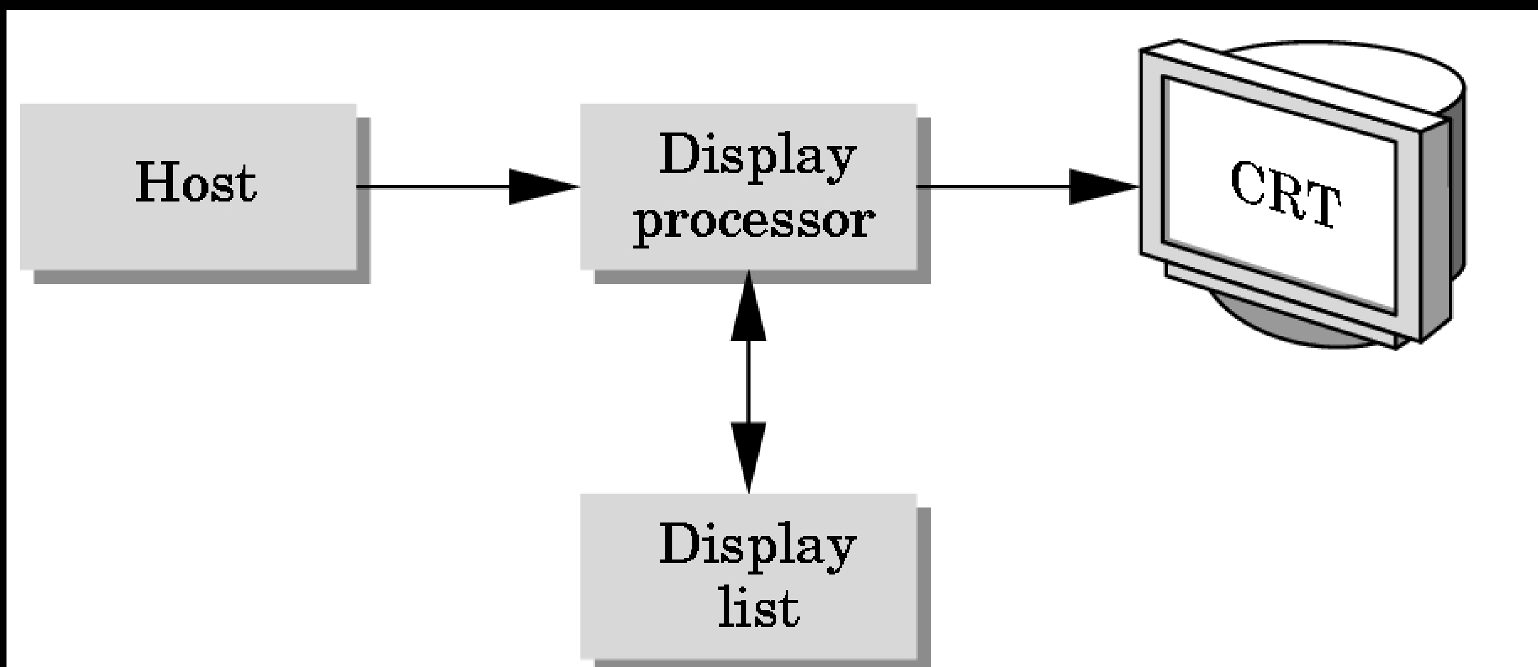
  Html-ified OpenGL man pages are on the course software page.

Future classes in Wean 5409

Graphics Hardware

Graphics Hardware

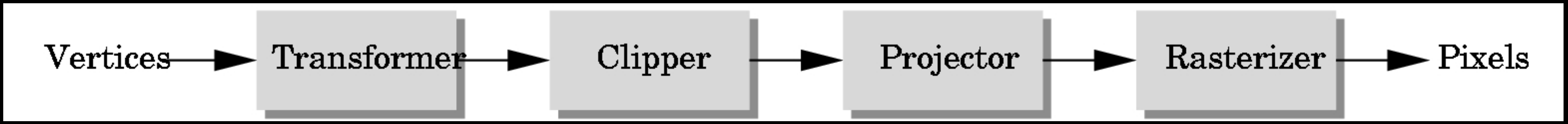
- First “graphics” processors just did display management, not rendering per se.
- bitblit for block transfer of bits



Goal

- Very fast frame rate on scenes with lots of interesting visual complexity
- Complexity from polygon count and/or texture mapping

Pipeline Architecture



- Pioneered by Silicon Graphics, picked up by graphics chips companies,

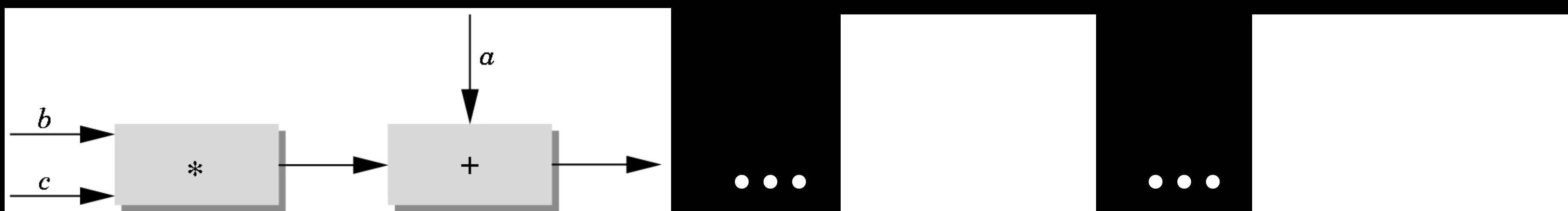
  Nvidia, 3dfx, S3, ATI,...

- OpenGL library was designed for this architecture (and vice versa)
- Good for opaque, textured polygons and lines

Why a Pipeline Architecture?

Higher throughput

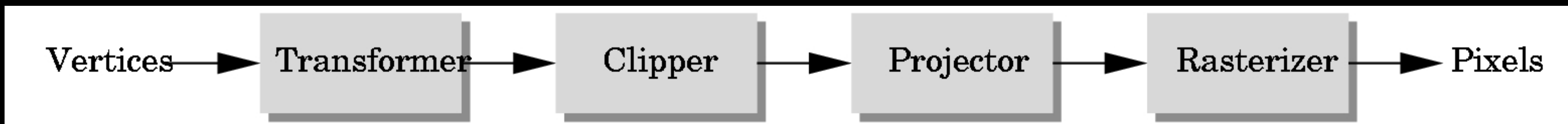
But potentially long latency



Parallel pipeline architecture

- each stage can employ multiple specialized processors, working in parallel, busses between stages
- #processors per stage, bus bandwidths carefully tuned for typical graphics use

Pipeline Stages



Immediate mode rendering

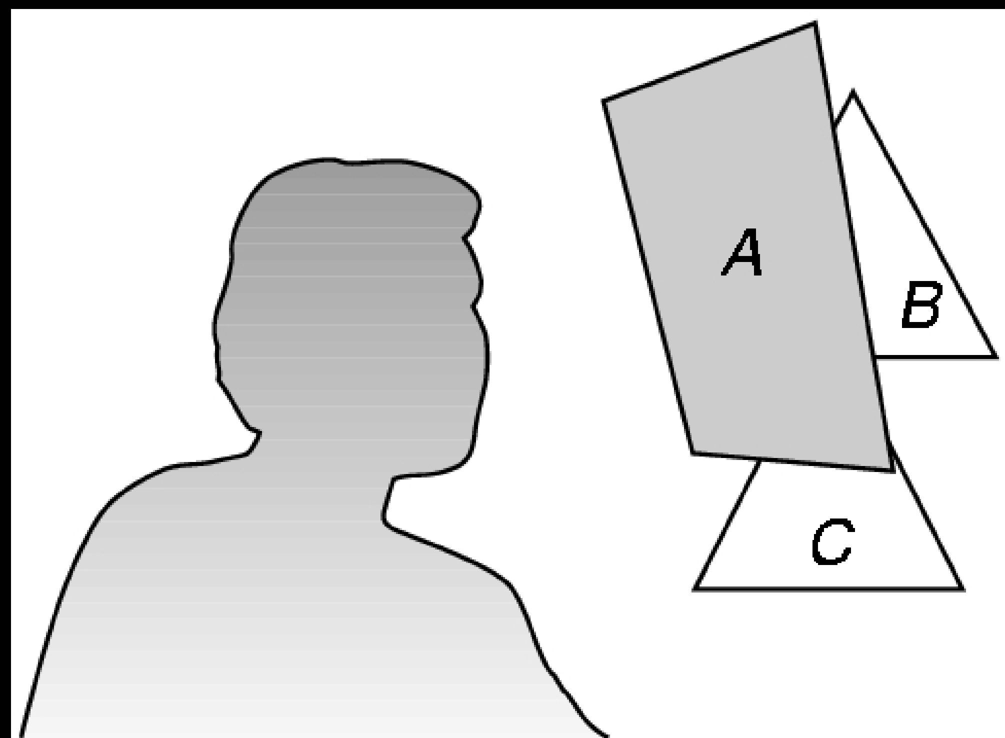
- application generates stream of geometric primitives (polygons, lines)
- system draws each one into buffer
- entire scene redrawn anew every frame

- **transform**
- **light**
- **clip**
- **perspective divide**
- **rasterize (scan convert)**
- **texture & fog**
- **z-buffer test**
- **alpha blend, dither**

Implementing Algorithms in Hardware

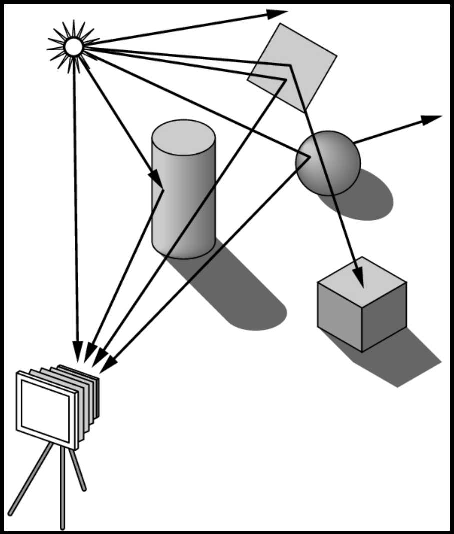
- Some work well, but others are harder
- Z-buffer

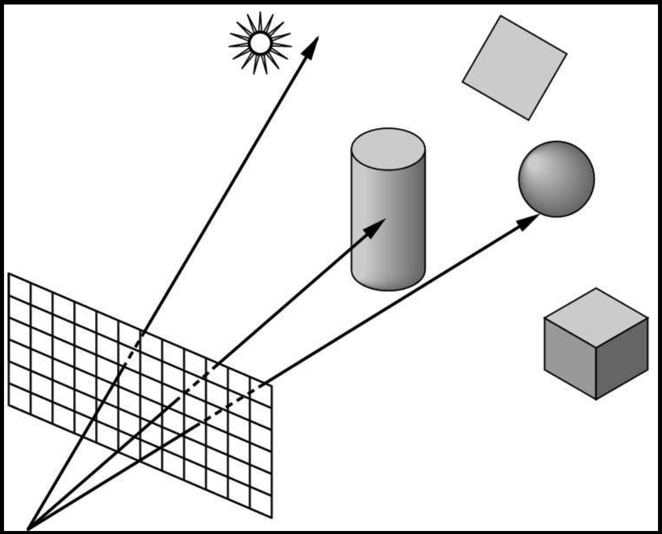
  computations are bounded, predictable



Implementing Algorithms in Hardware

- Ray tracing
  - Poor memory locality
  - Computational cost difficult to predict (esp. if adaptive)
  - SIMD (single instruction, multiple data) parallel approach
  - Keep copy of entire scene on each processor





- Recent graphics hardware approaches (Purcell et al., 2002)

Pixel Planes and Pixel Flow (UNC)

http://www.cs.unc.edu/~pxfl/

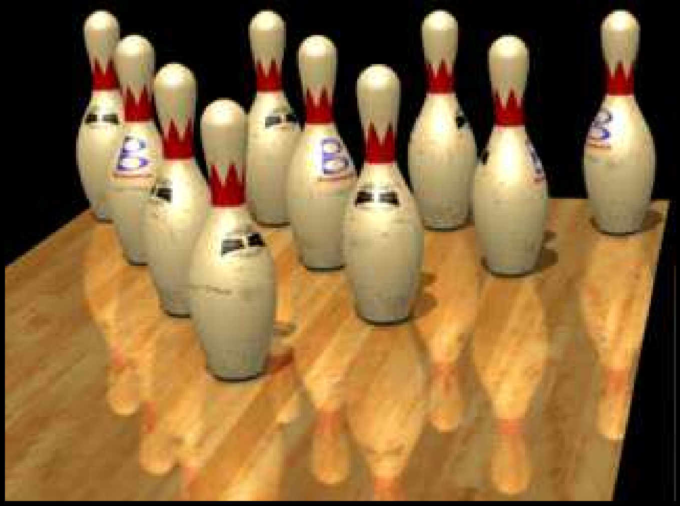
programmable processor per pixel

- good for programmable shading, image processing
- can be used for rasterization
- Pixel-Planes 4: 512x512 processors with 72bits of memory
- But most processors idle for most triangles
- Pixel-Planes 5: divide screen into ~20 tiles each with a bank of processors. Network is limit. 2Million tri/sec.

Pixel Planes and Pixel Flow (UNC)

Pixel-Flow: Image composition. Subdivide geometry to processors and recombine by depth using special hardware

Rendered on simulator and predicted to run in real time on physical hardware



Talisman (Microsoft)

http://research.microsoft.com/MSRSIGGRAPH/96/Talisman/

Observation: an image is usually much like the one that preceded it in an animation.

Goal: a $200-300 board

image-based rendering

- cache images of rendered geometry
- re-use with affine image warping (sophisticated sprites)
- re-render only when necessary to reduce bandwidth and computational cost

Current & Future Issues

- Interaction
- Geometry compression
- Progressive transmission
- Alternative modeling schemes (not polygon soup)
  - Parametric surfaces, implicit surfaces, subdivision surfaces
  - Generalized texture mapping: displacement mapping, light mapping
  - Programmable shaders
- Beyond just geometry:
  - dynamics, collision detection, AI, …

Future classes in Wean 5409