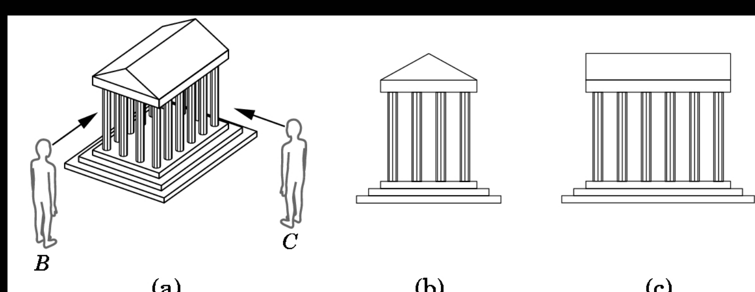
Graphics Hardware and OpenGL

Ubi Soft, Prince of Persia: The Sands of Time

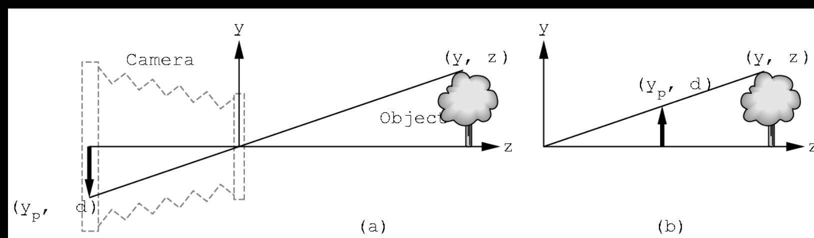
What does graphics hardware have to do fast?

Camera Views

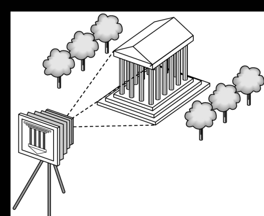
Different views of an object in the world

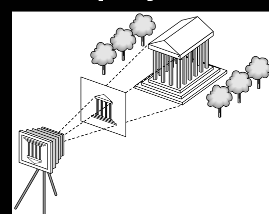


Camera Views

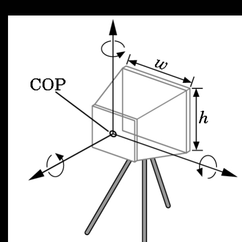


Lines from each point on the image are drawn through
the center of the camera lens (the ***center of projection*** (COP)).





Camera Views



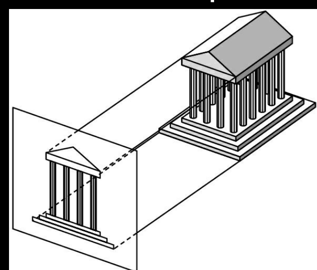
Many camera parameters…

For a physical camera:

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

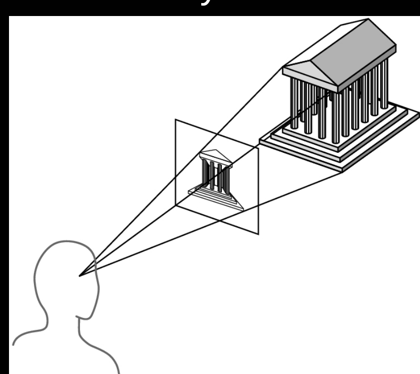
Camera Projections

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



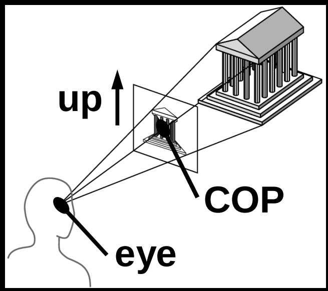
Camera Projections

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



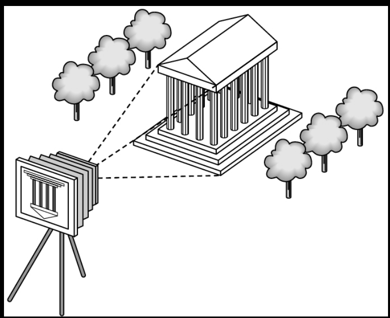
Camera Transformations

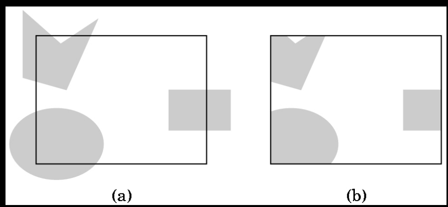
- Camera positioning just results in more transformations on the objects:
  - –Transformations that position the object relative to the camera



Clipping

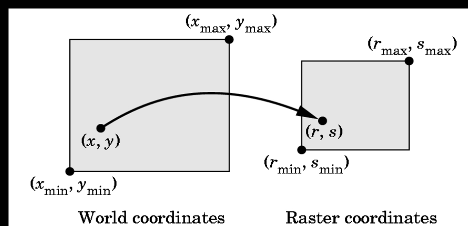
Not everything is visible on the screen

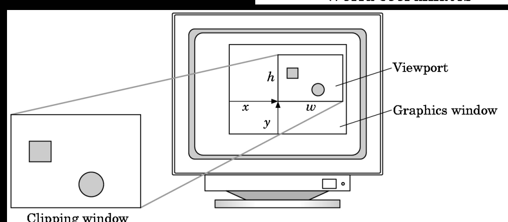


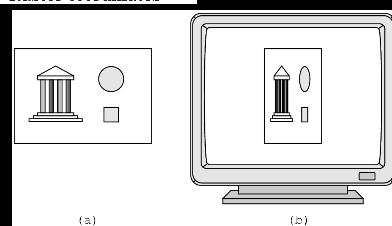


Rasterizer

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

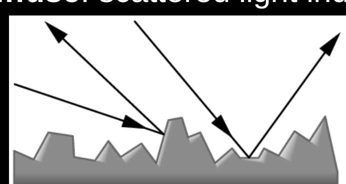


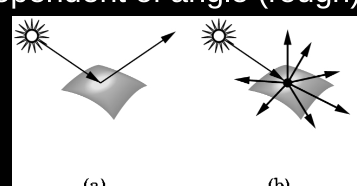




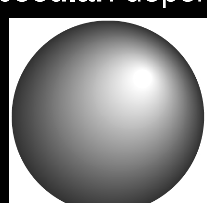
Shading: Material Properties

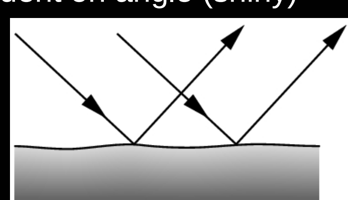
- **Ambient**: same at every point on the surface
- **Diffuse**: scattered light independent of angle (rough)





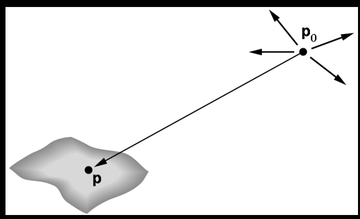
- **Specular**: dependent on angle (shiny)





Light Sources

- **Point light sources** are common:



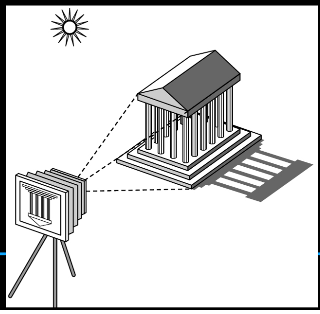
Special Tricks

- Gouraud Shading:

  Compute an appropriate color for each vertex, then smooth-shade between the different vertex colors.

- Shadows on ground plane:

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



So...

How does the graphics hardware make these operations fast?

OpenGL

- C programming language
- OpenGL libraries
  - for defining a 3D scene
  - convert scene description to pixels
  - use state variables (current color, current transform...)
  - platform independent
- GLUT libraries
  - handle windows, menus, keyboard input

OpenGL – “Hello World” example

```
int main (int argc, char *argv[]) {

   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_RGB);
   glutInitWindowSize(640, 480);
   glutCreateWindow(“Hello World”);

   glutDisplayFunc(display);

   glutMainLoop( );
   return(0);
}
```

OpenGL – “Hello World” example

```
void display( ) {
   glOrtho(-1, 1, -1, 1, -1, 1);

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

   glColor3f(1, 0, 0);
   glBegin(GL_TRIANGLES);
   glVertex2f(-0.5, -0.5);
   glVertex2f( 0.5, -0.5);
   glVertex2f( 0  ,  0.5);
   glEnd( );
   glFlush( );
}
```

OpenGL – “Hello World” example

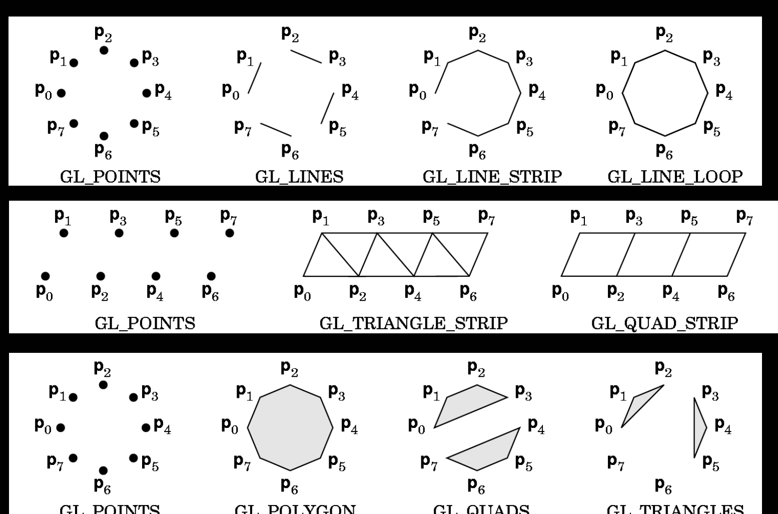
- You also need headers:

```
#include <stdlib.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glut.h>
```

- ..and a Makefile that links in the proper libraries

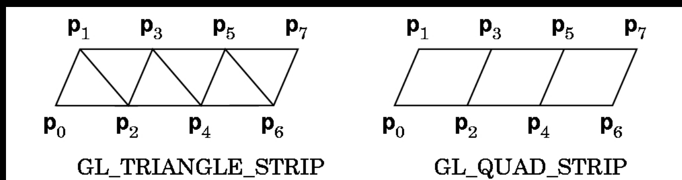
  See the starter code!

OpenGL functionality --- Primitives



Primitives

- Why triangles, quads, and strips?



Specifying Primitives

shapes.exe

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

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!

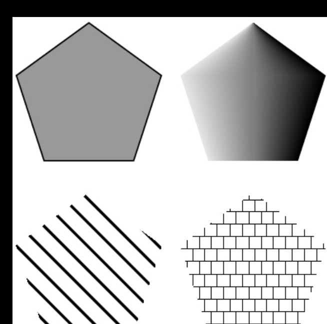
  Red, green & blue color model.

  Components are from 0-1.

Primitives: Material Properties

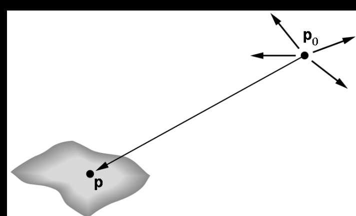
Many other material properties available:

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



Light Sources

lightposition.exe



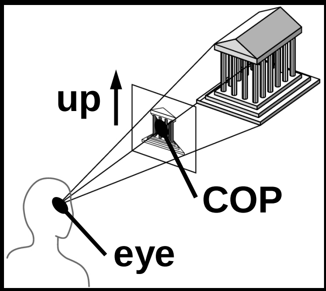
Transforms

transformation.exe

Camera Transformations

- Alternative to `glOrtho`

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



Graphics Hardware

What alternatives are there to the
triangles-through-the-pipeline approach?

Pixel Planes and Pixel Flow (UNC)

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

Pixel Planes:

programmable processor per pixel

fast rasterization of single triangle

“hey pixels, figure out if you are in this triangle”

what happens when triangles get very small?

Pixel Planes and Pixel Flow (UNC)

Pixel-Flow:

processors each take a subset of the geometry and render a full-size image

images are then combined using depth information

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

- Geometry compression (far beyond triangle strips)
- Progressive transmission (fill in detail)
- 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, …

Admin

- Assignment goes out Tuesday
- You should have Wean Hall 5336 access

- My office hours are Tuesday 1-2pm
  (or send email to set up an appointment)

- The OpenGL book (linked off the web page) is quite good --- make use of it as a resource!