

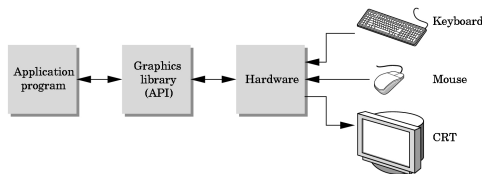
# Open GL and Graphics Hardware

Review of OpenGL  
 Machine Architecture  
 Alternatives

How many of you have programmed in OpenGL?  
 How extensively?

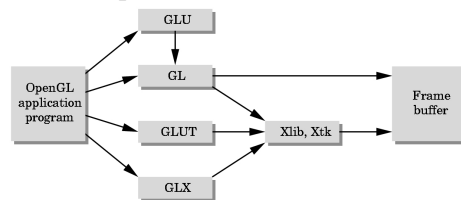
## What is OpenGL?

- 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 it isn't:

A windowing program or input driver because those couldn't be OS independent.

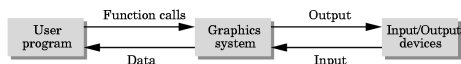


**GL:** core graphics capability  
**GLU:** utilities on top of GL  
**GLUT:** input and windowing functions

## 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



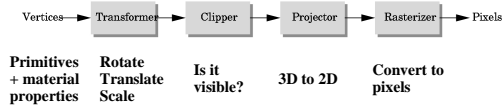
## How does it work?

### 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



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.

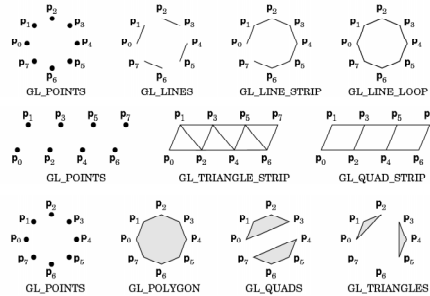
## Specifying Primitives



Shapes.exe

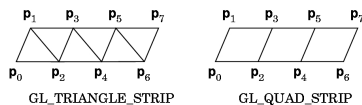
Code for all of today's examples available from  
<http://www.xmission.com/~nate/tutors.html>

## 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 vertices)

## Primitives: Material Properties

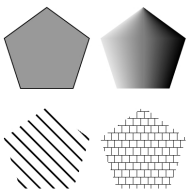
- `glColor3f(r, g, b);`  
All subsequent primitives will be this color—  
colors are not attached to objects but this call  
changes the state of the system
- Everyone who learns gl gets bitten by this!

Red, green & blue color model  
Components are 0-1

### Primitives: Material Properties

Many other material properties available:

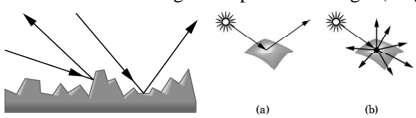
```
glEnable(GL_POLYGON_STIPPLE);
glPolygonStipple(MASK); /* 32x32 pattern of bits */
...
glDisable(GL_POLYGON_STIPPLE);
```



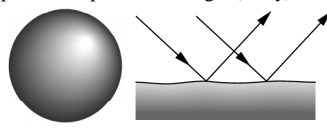
Computer Graphics 15-462 (Fall 2002) 13

### Primitives: Material Properties

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




Specular: dependent on angle (shiny)



Computer Graphics 15-462 (Fall 2002) 14

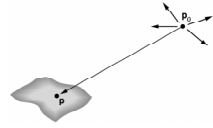
### Primitives: Material Properties



Computer Graphics 15-462 (Fall 2002) 15

### Light Sources

Most often point light sources

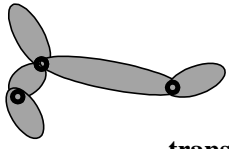


**lightpositions.exe**

Computer Graphics 15-462 (Fall 2002) 16

### Transforms

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

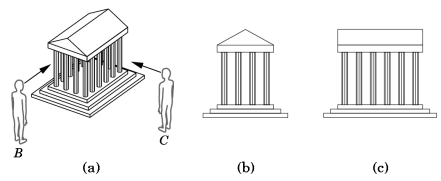


**transformations.exe**

Computer Graphics 15-462 (Fall 2002) 17

### Position it relative to the camera

Different views of the objects in the world



Computer Graphics 15-462 (Fall 2002) 18

### Position it relative to the camera

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

Computer Graphics 15-462 (Fall 2002) 19

### Position it relative to the camera

Many camera parameters...  
For a physical camera:  
position (3)  
orientation (3)  
lens (field of view)

Orthographic projection: long telephoto lens.  
Flat but preserving distances and shapes. All the projectors are now parallel.  
glOrtho (left, right, bottom, top, near, far);

Computer Graphics 15-462 (Fall 2002) 20

### Position it relative to the camera

Perspective projection

Computer Graphics 15-462 (Fall 2002) 21

### Camera Transformations

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

Computer Graphics 15-462 (Fall 2002) 22

### Clipping

Not everything should be visible on the screen

Computer Graphics 15-462 (Fall 2002) 23

### Rasterizer

Go from pixel value in world coordinates to pixel value in screen coordinates

Computer Graphics 15-462 (Fall 2002) 24

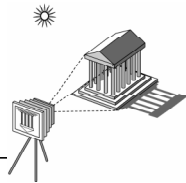
## Special Tricks

- **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 shadow map



## Special Tricks

- **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**

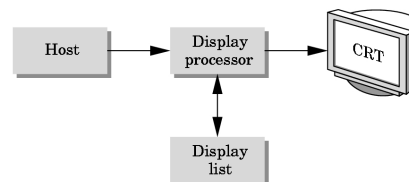
We will give you example code for each assignment. Modifying existing code is much easier than writing "hello world" (unfortunately)

- **Documentation:**

Book

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

## 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

---

Computer Graphics 15-462 (Fall 2002) 31

### 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

---

Computer Graphics 15-462 (Fall 2002) 32

### 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

---

Computer Graphics 15-462 (Fall 2002) 33

### 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

---

Computer Graphics 15-462 (Fall 2002) 34

### Implementing Algorithms in Hardware

Some work well, others are harder

- **Z-buffer**  
computations are bounded, predictable

---

Computer Graphics 15-462 (Fall 2002) 35

### 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

---

Computer Graphics 15-462 (Fall 2002) 36

## Current chip design may not be the long term answer

- **Observation: # triangles == # of pixels**
- **Could focus on interactivity**  
Latency becomes a problem
- **Could focus on animation**  
Avoid repeating computations  
Image-based rendering?

Computer Graphics 15-462 (Fall 2002)

37

## 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.

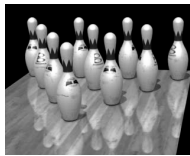
Computer Graphics 15-462 (Fall 2002)

38

## 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**



Computer Graphics 15-462 (Fall 2002)

39

## 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

Computer Graphics 15-462 (Fall 2002)

40

## 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?

Computer Graphics 15-462 (Fall 2002)

41