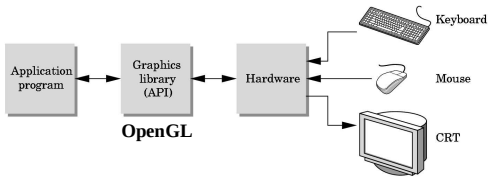
Graphics Pipeline

Graphics API and Graphics Pipeline
Efficient Rendering and Data transfer
Event Driven Programming

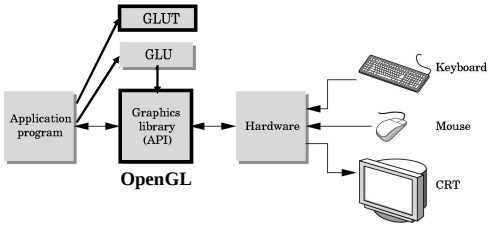
What is graphics API ?

- A low-level interface to graphics hardware
- OpenGL
  - About 120 commands to specify 2D and 3D graphics
  - OS independent



What it isn't:

A windowing program or input driver because



GLUT: window management, keyboard, mouse, menue
GLU: higher level library, complex objects

How many of you have programmed in OpenGL?

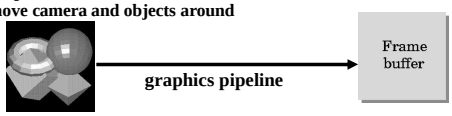
How extensively?

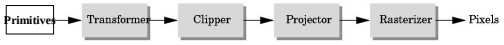
How does it work?

From the implementor's perspective:

geometric objects
properties: color…
move camera and objects around

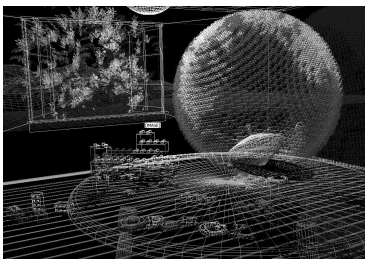
pixels



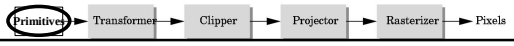


Primitives + material properties | **Rotate Translate Scale** | **Is it visible?** | **3D to 2D** | **Convert to pixels**

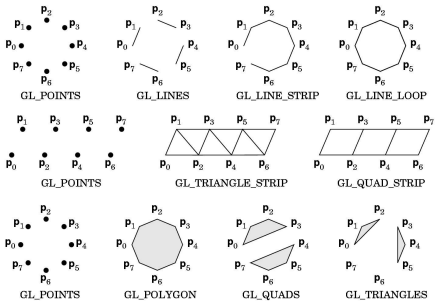
Primitives: drawing a polygon



Build models in appropriate units (microns, meters, etc.).
From simple shapes: triangles, polygons,…



Primitives: drawing a polygon



Primitives → Transformer → Clipper → Projector → Rasterizer → Pixels

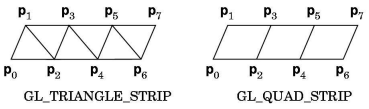
Primitives: drawing a polygon

- **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();

Primitives → Transformer → Clipper → Projector → Rasterizer → Pixels

Triangle Strips

Minimize number of vertices to be processed



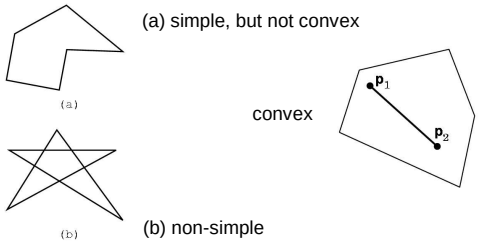
TR1 = p0, p1, p2
TR2 = p1, p2, p3

Strip = p0, p1, p2, p3, p4,…

9

Polygon Restrictions

- **OpenGL Polygons must be simple**
- **OpenGL Polygons must be convex**



10

Primitives: Material Properties

color, transparency, reflection properties, shading properties

RGB (Red, Green, Blue)

red	**1**	**0**	**0**
black	**0**	**0**	**0**
white	**1**	**1**	**1**
magenta	**1**	**0**	**1**

Primitives → Transformer → Clipper → Projector → Rasterizer → Pixels

Material Properties: Color

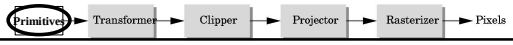
- **glColor3f (r, g, b);**
  Red, green & blue color model
  Components are 0-1

Primitives → Transformer → Clipper → Projector → Rasterizer → Pixels

Specifying Primitives

run shapes.exe

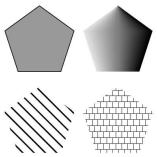
Code for all of today's examples available from http://www.xmission.com/~nate/tutors.html

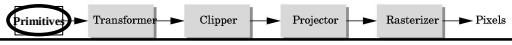


Primitives: Material Properties

Many other material properties available:

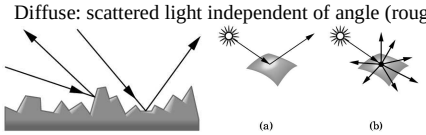
```
glEnable(GL_POLYGON_STIPPLE);
glPolygonStipple(MASK); /* 32x32 pattern of bits */
…
glDisable (GL_POLYGON_STIPPLE);
```



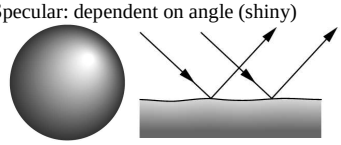


Primitives: Material Properties

Diffuse: scattered light independent of angle (rough)

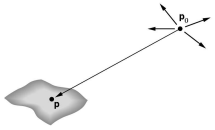


Specular: dependent on angle (shiny)

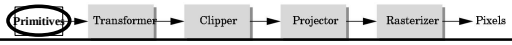


Light Sources

Most often point light sources



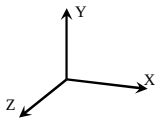
run lightposition.exe



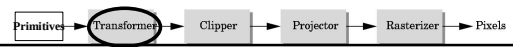
Transforms

- Rotate
- Translate
- Scale

- glRotate(x,y,z);
- glTranslate(x,y,z);
- draw geometry

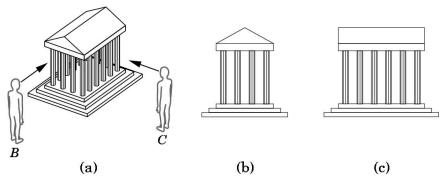


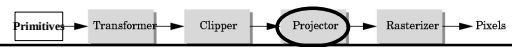
run transformation.exe



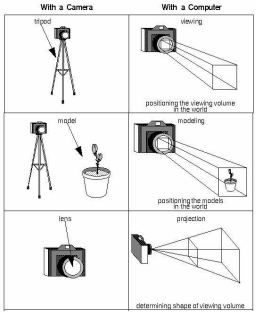
Position it relative to the camera

Different views of the objects in the world





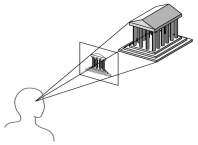
Position it relative to the camera

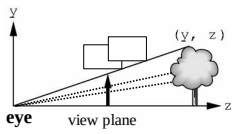


Primitives → Transformer → Clipper → **Projector** → Rasterizer → Pixels

Position it relative to the camera

Perspective projection



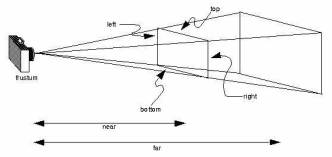


Primitives → Transformer → Clipper → **Projector** → Rasterizer → Pixels

Position it relative to the camera

Perspective projection

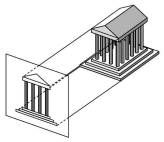
glFrustum (left, right, bottom, top, near, far);



Primitives → Transformer → Clipper → **Projector** → Rasterizer → Pixels

Position it relative to the camera

Orthographic projection
Flat but preserving distances and shapes.
All the projectors are now parallel.



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

Primitives → Transformer → Clipper → **Projector** → Rasterizer → Pixels

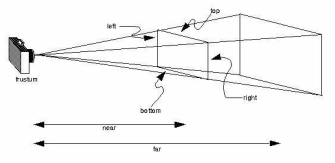
Camera Transformations

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

Primitives → Transformer → Clipper → **Projector** → Rasterizer → Pixels

Clipping

Not everything should be visible on the screen

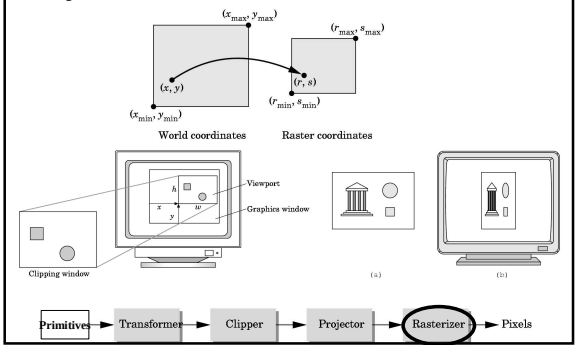


any vertices that lie outside of the viewing volume are clipped

Primitives → Transformer → **Clipper** → Projector → Rasterizer → Pixels

Rasterizer

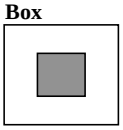
Go from pixel value in world coordinates to pixel value in screen coordinates



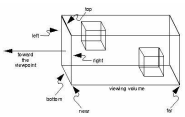
Drawing A Box

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

    glOrtho(-1.0,1.0,-1.0,1.0,-1.0,1.0);

    glClearColor(1.0, 0.0, 0.0, 1.0);
    glClear(GL_COLOR_BUFFER_BIT);

    glColor3f(0.0, 1.0, 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();
}
```

Box



(GLUT library)



Getting Started

- **Example Code**

  We will give you example code for each assignment.

- **Documentation:**

  Book

  OpenGL pages are on the web.

Graphics API and Graphics Pipeline
Efficient Rendering and Data transfer
Event Driven Programming

28

Graphics Hardware: Goal

Very fast frame rate on scenes with lots of interesting visual complexity

Graphics Hardware: Goal

- Pioneered by Silicon Graphics, picked up by graphics chips companies (Nvidia, 3dfx, S3, ATI,...).
- OpenGL library was designed for this architecture (and vice versa)
- Changed a lot over last years
- Programmable pixel and vertex shaders

Nvidia person in class
February 8

Efficient Rendering and Data transfer

Minimize number of OpenGL calls
Minimize number of Vertices
Minimize data transfer from CPU to GPU

Billions of vertices per second
Every six month the speed doubles

31

State Machine

- **Large set of state variables:**
  color
  current viewing position
  line width
  material properties...
- **These variables (the state) then apply to every subsequent drawing command**

32

State Machine

Minimize changes to the state and number of calls

```
glBegin(GL_POLYGON);
    glColor3f(0.0, 1.0, 0.0)
    glVertex2f(x0, y0);
    glColor3f(0.0, 1.0, 0.0)
    glVertex2f(x1, y1);
    glColor3f(0.0, 1.0, 0.0)
    glVertex2f(x2, y2) ...
glEnd();
```

33

State Machine

Minimize changes to the state and number of calls

```
glColor3f(0.0, 1.0, 0.0)
glBegin(GL_POLYGON);
        glVertex2f(x0, y0);
        glVertex2f(x1, y1);
        glVertex2f(x2, y2) ...
glEnd();
```

34

Transfer of data from CPU to GPU

Immediate transfer

```
glBegin(GL_TRIANGLES);
    glColor3f(0.0, 1.0, 0.0)
    glVertex2f(x0, y0);
    glColor3f(0.0, 1.0, 0.0)
    glVertex2f(x1, y1);
    glColor3f(0.0, 1.0, 0.0)
    glVertex2f(x2, y2)
glEnd();
```

Disadvantage:
static geometry – send the same data every frame
better to store geometry on graphics card memory

35

Display Lists

Store object in graphics card memory

- **Encapsulate a sequence of drawing commands**
- **Optimize and store**

```
GLuint listName = glGenLists(1);   /* new name */
glNewList (listName, GL_COMPILE); /* new list */
        define object (glColor, glVertex, …)
glEndList();
glCallList(listName);    /* draw one */
```

- **Vertex buffer objects**

36

Assignment 1

Height Fields

37

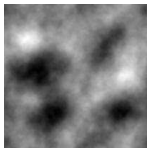
Height Fields

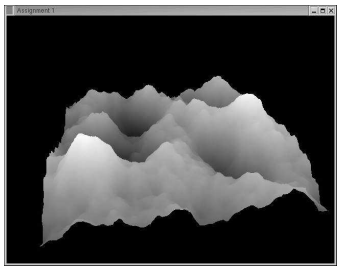
- **Why?**
  - Get started with OpenGL
  - Some room for creativity
- **Where?**
  - Wean 5336 or your machine at your risk!
- **How?**
  - Cross-realm authentication via andrew
  - Send problems to me or to the TA's (soon)
  - Make sure that you made directory with correct permissions—most common problem

38

Height Fields

- **What?**





- **When? -- Due midnight February 1st**

39

Next Class

Graphics API and Graphics Pipeline
Efficient Rendering and Data transfer
Event Driven Programming

40