15-462 Computer Graphics I

Lecture 2

Basic Graphics Programming

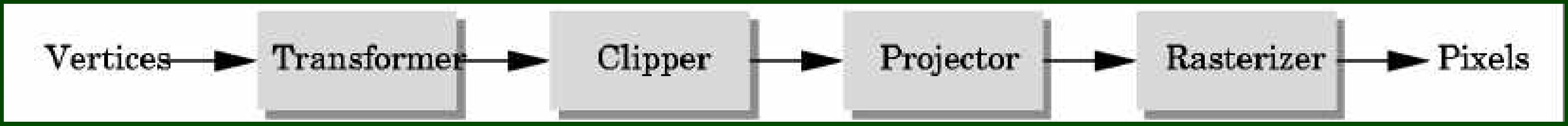
Graphics Pipeline
OpenGL API
Primitives: Lines, Polygons
Attributes: Color
Example

[Angel Ch. 2]

January 16, 2003
Frank Pfenning
Carnegie Mellon University

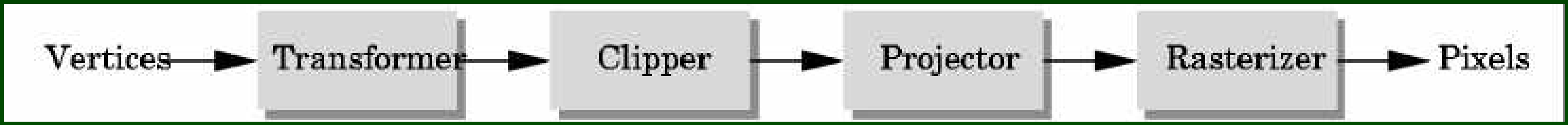
http://www.cs.cmu.edu/~fp/courses/graphics/

A Graphics Pipeline



- Pipelines and parallelism
- Latency vs throughput
- Efficiently implementable in hardware
- Not so efficiently implementable in software

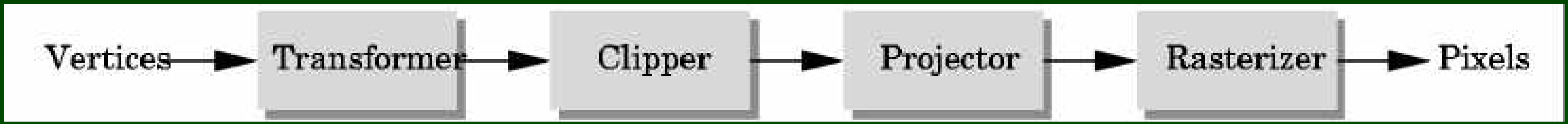
Programming a Pipeline



Vertices → Transformer → Clipper → Projector → Rasterizer → Pixels

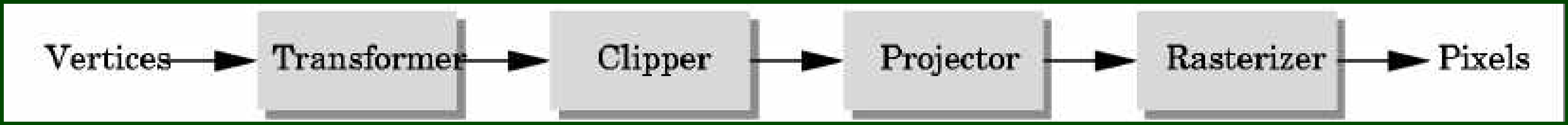
- Specify the operation of each box
- Replace or accumulate
- State and lack of modularity
- Immediate mode graphics
  - On-line (OpenGL)
- Modeling-rendering pipeline
  - Off-line (Pixar's Renderman)

Vertices



- Vertices in world coordinates
- void glVertex3f(GLfloat x, GLfloat y, GLfloat z)
  - Vertex (x, y, z) sent down the pipeline
  - Function call returns
- Use GL*type* for portability and consistency
- glVertex{234}{sfid}[v](*TYPE coords*)

Transformer

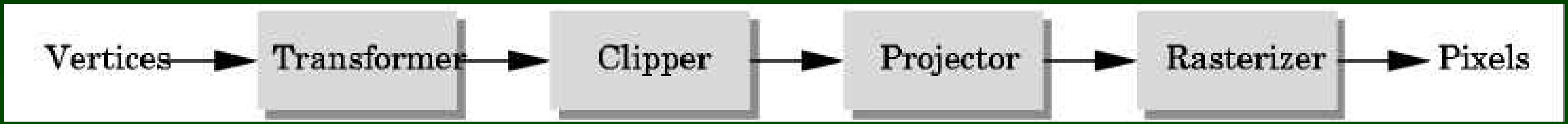


- Transformer in world coordinates
- Must be set before object is drawn!

```
glRotatef(45.0, 0.0, 0.0, -1.0);
glVertex2f(1.0, 0.0);
```

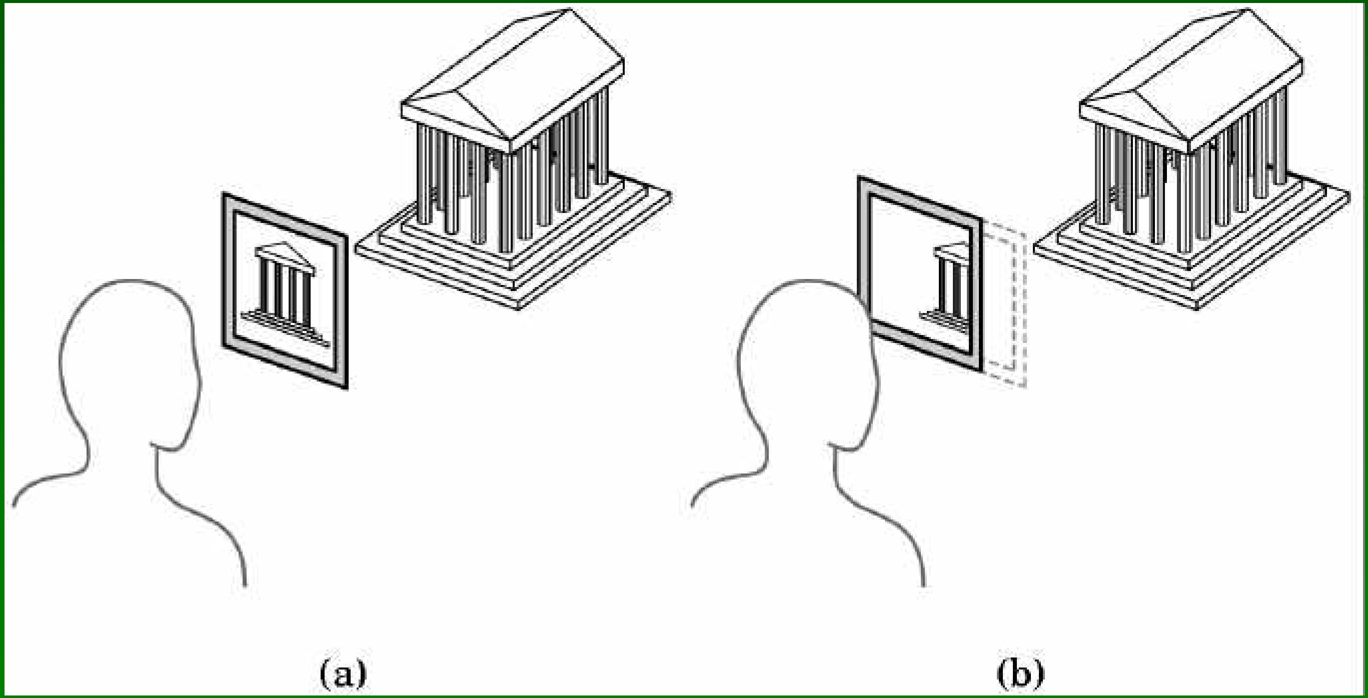
- Complex [Angel Ch. 4]

Clipper

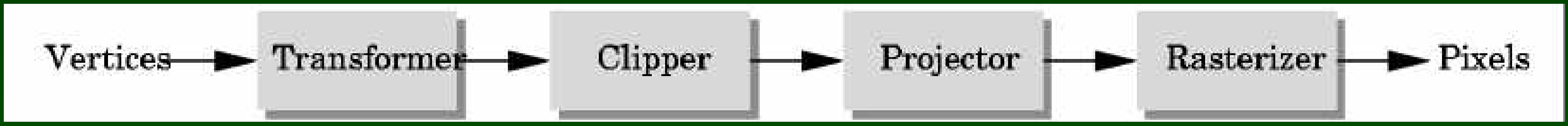


Vertices → Transformer → Clipper → Projector → Rasterizer → Pixels

- Mostly automatic from viewport



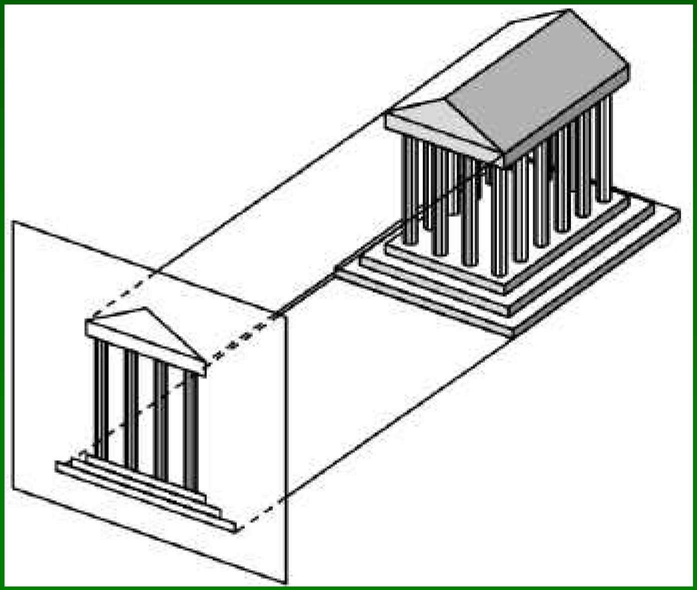
Projector



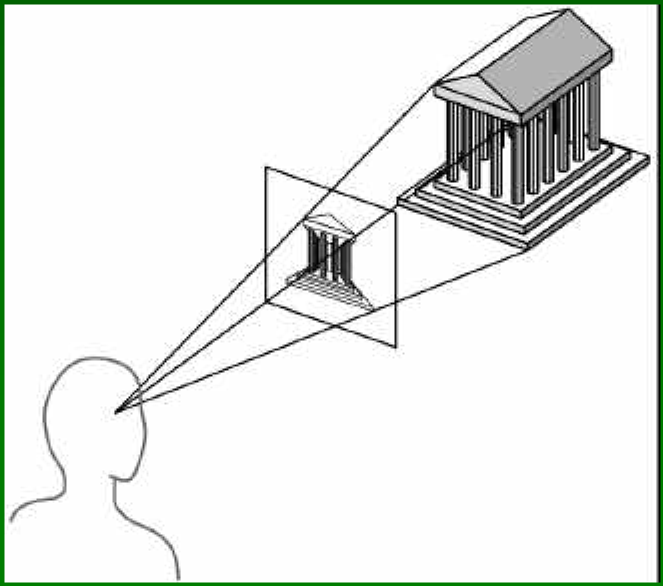
Vertices → Transformer → Clipper → Projector → Rasterizer → Pixels

- Complex transformation [Angel Ch. 5]

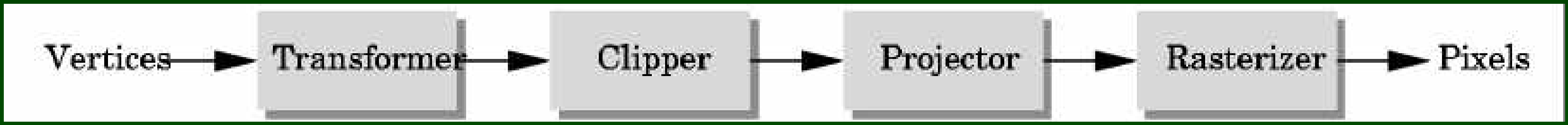
Orthographic



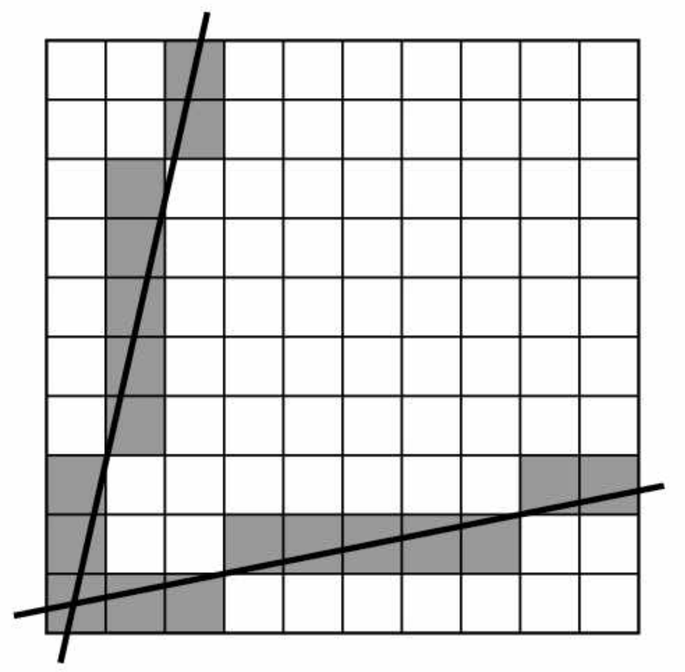
Perspective



Rasterizer



- Interesting algorithms [Angel Ch. 7]
- To window coordinates

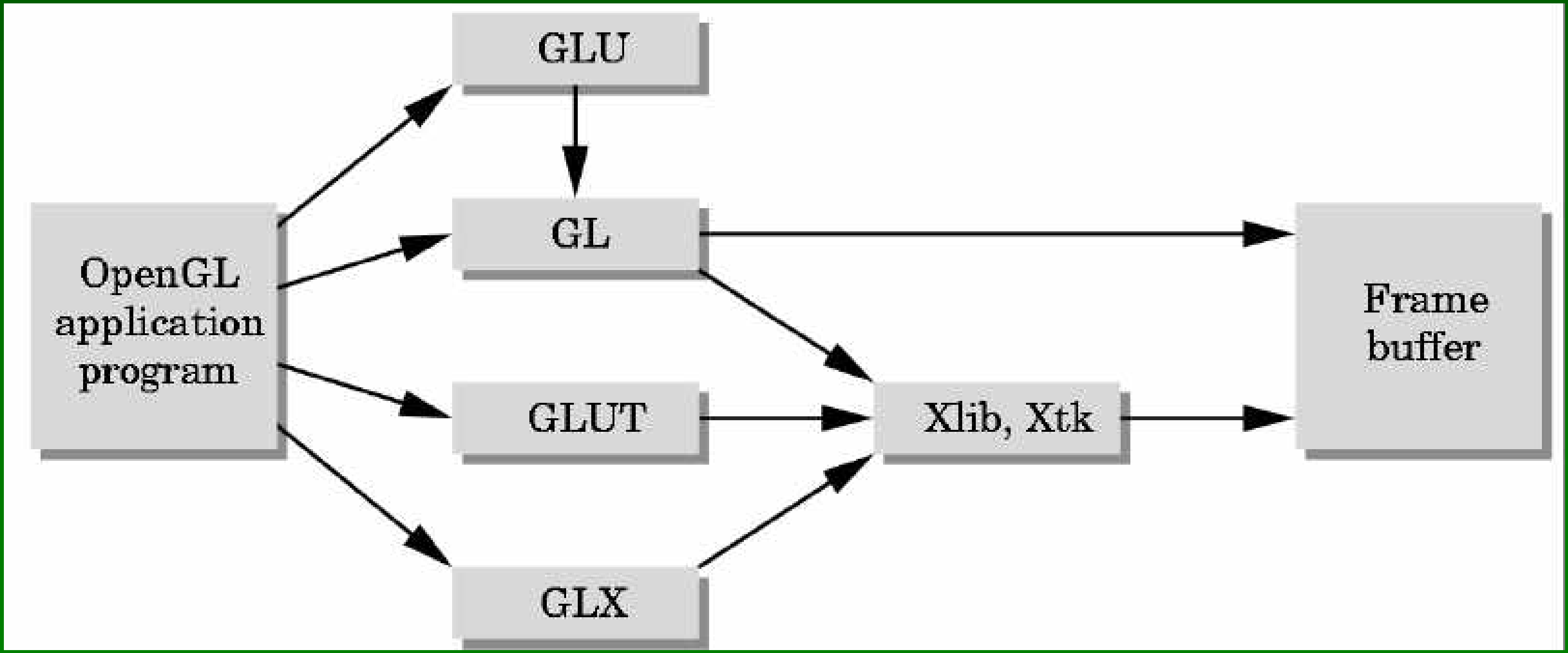


Outline

1. A Graphics Pipeline
2. The OpenGL API
3. Primitives: vertices, lines, polygons
4. Attributes: color
5. Example: drawing a shaded triangle

OpenGL Library Organization

- GLU (OpenGL Utility Library), modeling
- GLUT (GL Utility Toolkit), window system interface



Graphics Functions

- Primitive functions
- Attribute functions
- Transformation functions
- Viewing functions
- Input functions
- Control functions

Outline

1. A Graphics Pipeline
2. The OpenGL API
3. Primitives: vertices, lines, polygons
4. Attributes: color
5. Example: drawing a shaded triangle

Primitives

- Specified via vertices
- General schema

```
glBegin(type);
    glVertex*(...);
    ...
    glVertex*(...);
glEnd();
```

- *type* determines interpretation of vertices

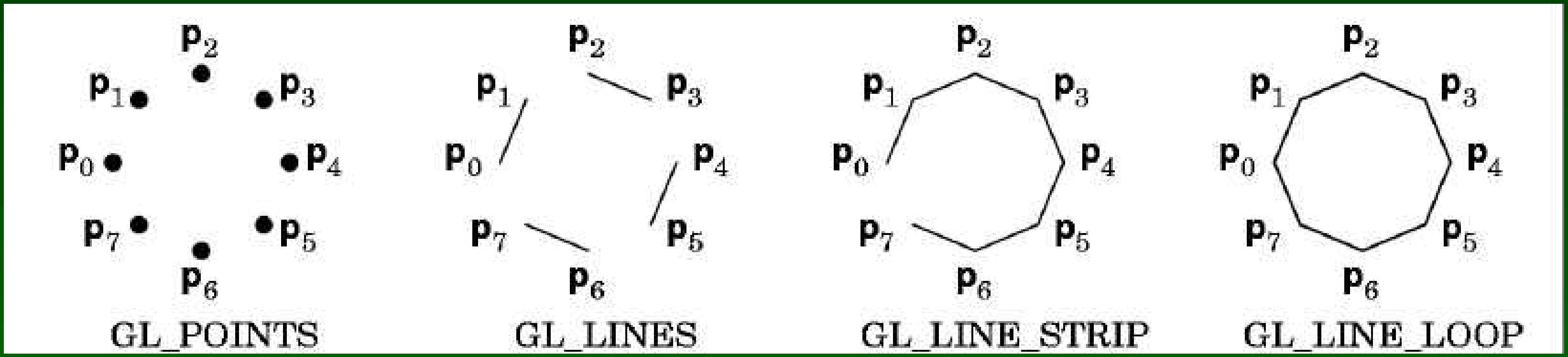
Example: Square Outline

- *Type* GL_LINE_LOOP

```
glBegin(GL_LINE_LOOP);
   glVertex2f(0.0, 0.0);
   glVertex2f(1.0, 0.0);
   glVertex2f(1.0, 1.0);
   glVertex2f(0.0, 1.0);
glEnd();
```

- z coordinate defaults to 0
- Calls to other functions are allowed between glBegin(*type*) and glEnd();

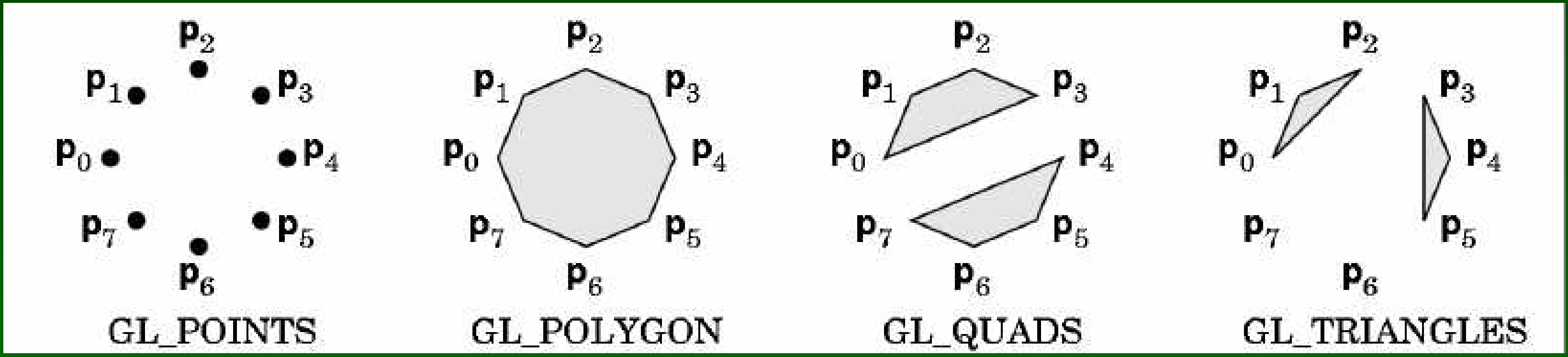
Points and Line Segments



- Make sense in three dimensions

Polygons

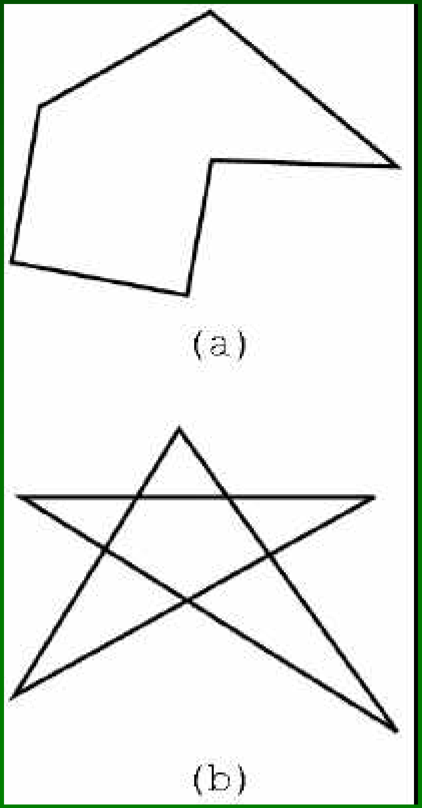
- Polygons enclose an area



- Rendering of area (fill) depends on attributes
- All vertices must be in one plane

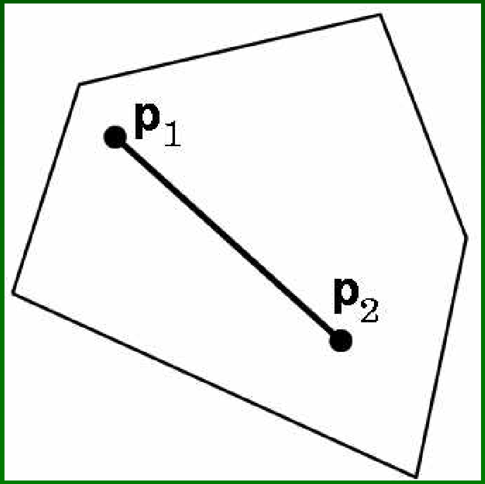
Polygon Restrictions

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



(a) simple, but not convex

convex



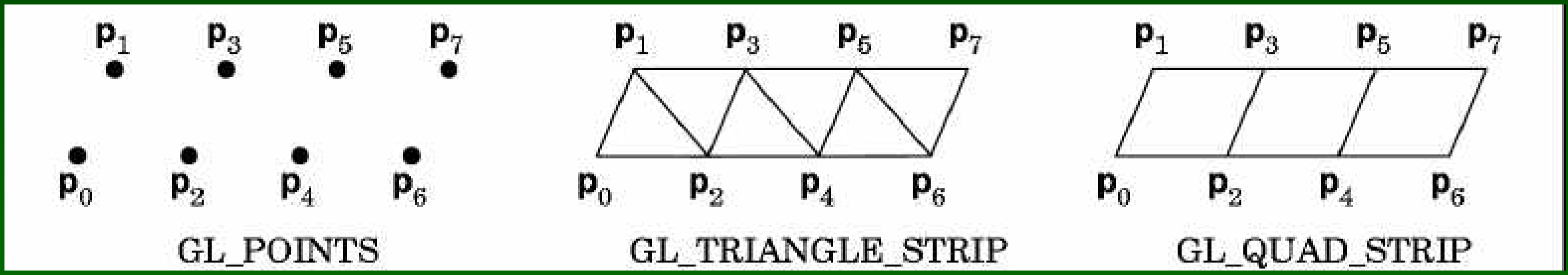
(b) non-simple

Why Polygon Restrictions?

- Non-convex and non-simple polygons are expensive to process and render
- Convexity and simplicity is expensive to test
- Behavior of OpenGL implementation on disallowed polygons is “undefined”
- Some tools in GLU for decomposing complex polygons (tessellation)
- Triangles are most efficient

Polygon Strips

- Efficiency in space and time
- Reduces visual artefacts



- Polygons have a front and a back, possibly with different attributes!

Outline

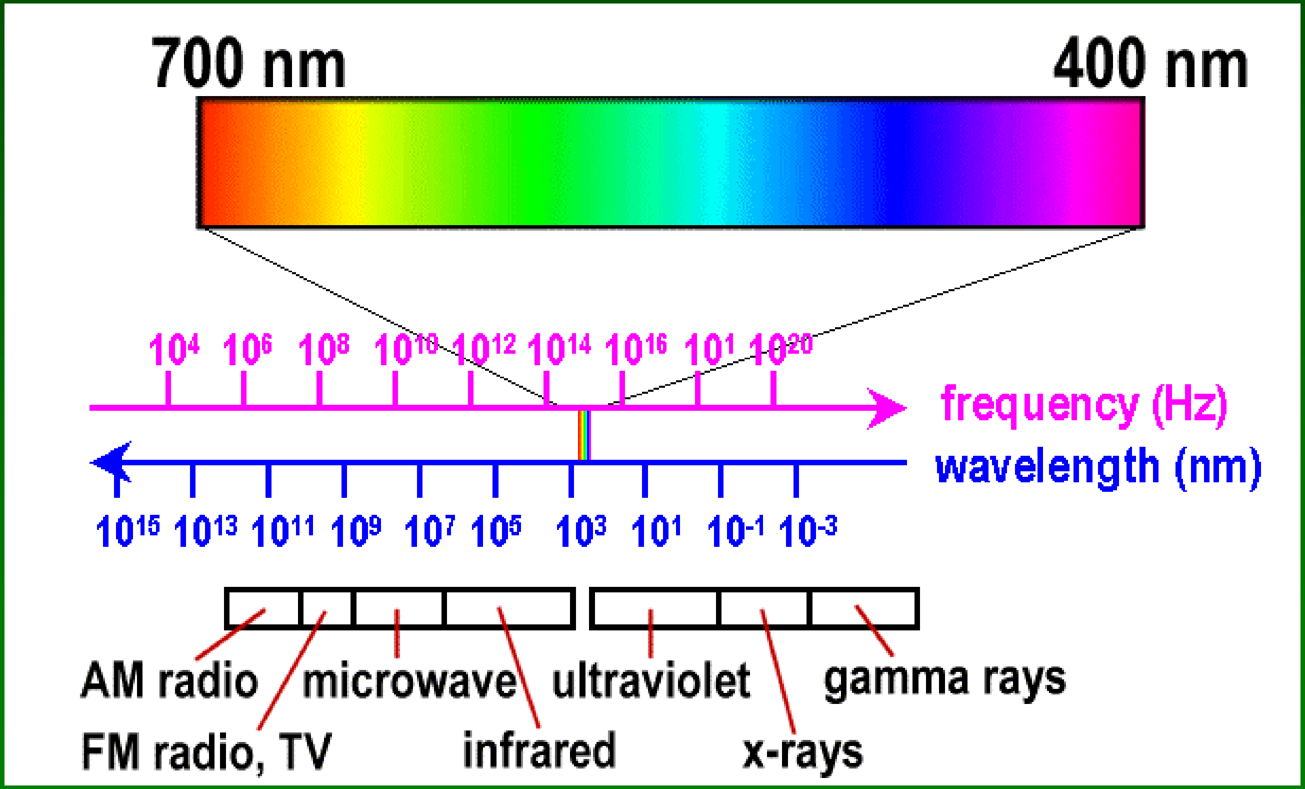
1. A Graphics Pipeline
2. The OpenGL API
3. Primitives: vertices, lines, polygons
4. Attributes: color
5. Example: drawing a shaded triangle

Attributes

- Part of the state of the graphics pipeline
- Set before primitives are drawn
- Remain in effect!
- Examples:
  - Color, including transparency
  - Reflection properties
  - Shading properties

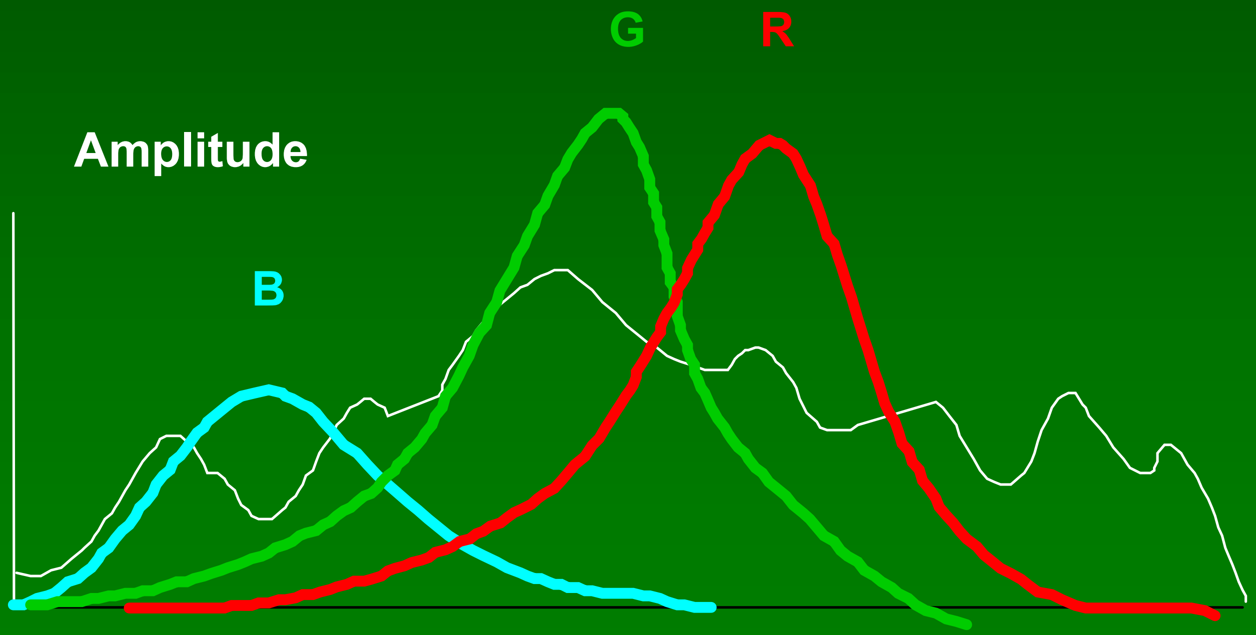
Physics of Color

- Electromagnetic radiation
- Can see only tiny piece of the spectrum



Color Filters

- Eye can perceive only 3 basic colors
- Computer screens designed accordingly

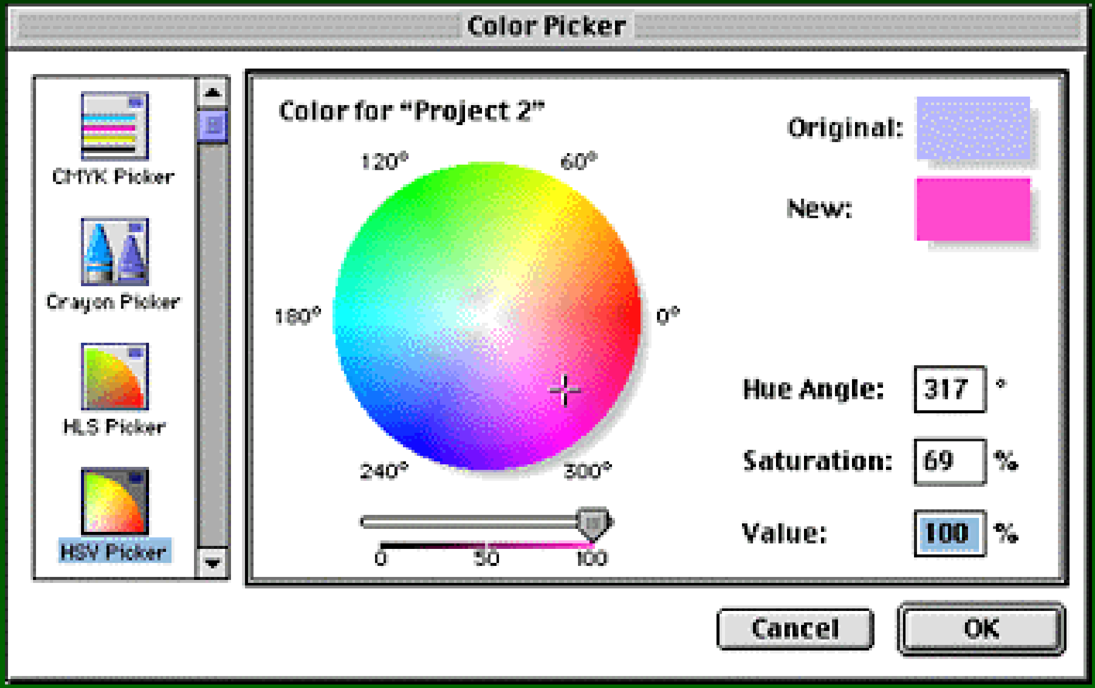


Color Spaces

- RGB (Red, Green, Blue)
  - Convenient for display
  - Can be unintuitive (3 floats in OpenGL)
- HSV (Hue, Saturation, Value)
  - Hue: what color
  - Saturation: how far away from gray
  - Value: how bright
- Others for movies and printing

RGB vs HSV

Apple Color Picker



Color for "Project 2"

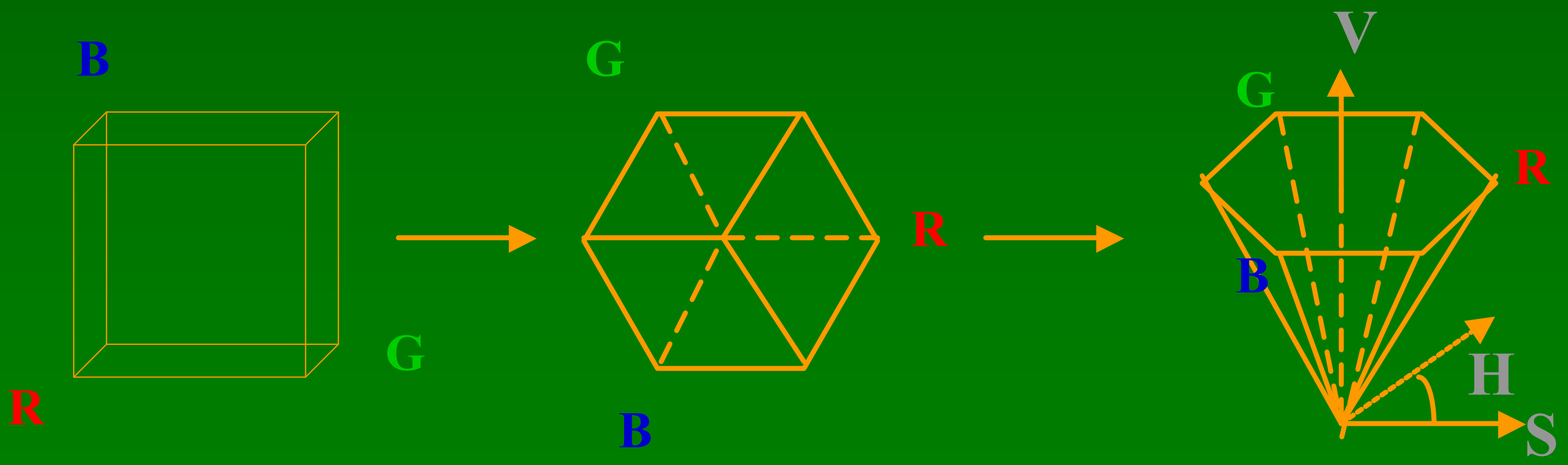
Original:

New:

Hue Angle: 317 °

Saturation: 69 %

Value: 100 %



Outline

1. A Graphics Pipeline
2. The OpenGL API
3. Primitives: vertices, lines, polygons
4. Attributes: color
5. Example: drawing a shaded triangle

Example: Drawing a shaded polygon

- Initialization: the “main” function

```
int main(int argc, char** argv)
{
  glutInit(&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowSize (500, 500);
  glutInitWindowPosition (100, 100);
  glutCreateWindow (argv[0]);
  init ();
...
```

GLUT Callbacks

- Window system independent interaction
- glutMainLoop processes events

```
...
  glutDisplayFunc(display);
  glutReshapeFunc(reshape);
  glutKeyboardFunc (keyboard);
  glutMainLoop();
  return 0;
}
```

Initializing Attributes

- Separate in “init” function

```
void init(void)
{
  glClearColor (0.0, 0.0, 0.0, 0.0);

  /* glShadeModel (GL_FLAT); */
  glShadeModel (GL_SMOOTH);
}
```

The Display Callback

- Handles exposure events
- Install with glutDisplayFunc(display)

```
void display(void)
{
  glClear (GL_COLOR_BUFFER_BIT); /* clear buffer */
  triangle ();                   /* draw triangle */
  glFlush ();                    /* force display */
}
```

Drawing

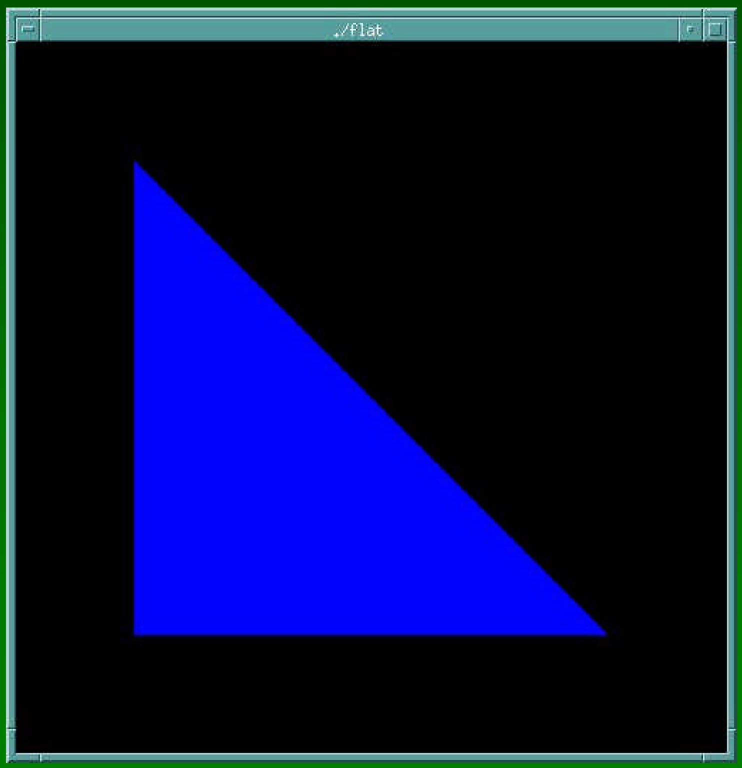
- In world coordinates; remember state!

```
void triangle(void)
{
  glBegin (GL_TRIANGLES);
    glColor3f (1.0, 0.0, 0.0); /* red */
    glVertex2f (5.0, 5.0);
    glColor3f (0.0, 1.0, 0.0); /* green */
    glVertex2f (25.0, 5.0);
    glColor3f (0.0, 0.0, 1.0); /* blue */
    glVertex2f (5.0, 25.0);
  glEnd();
}
```

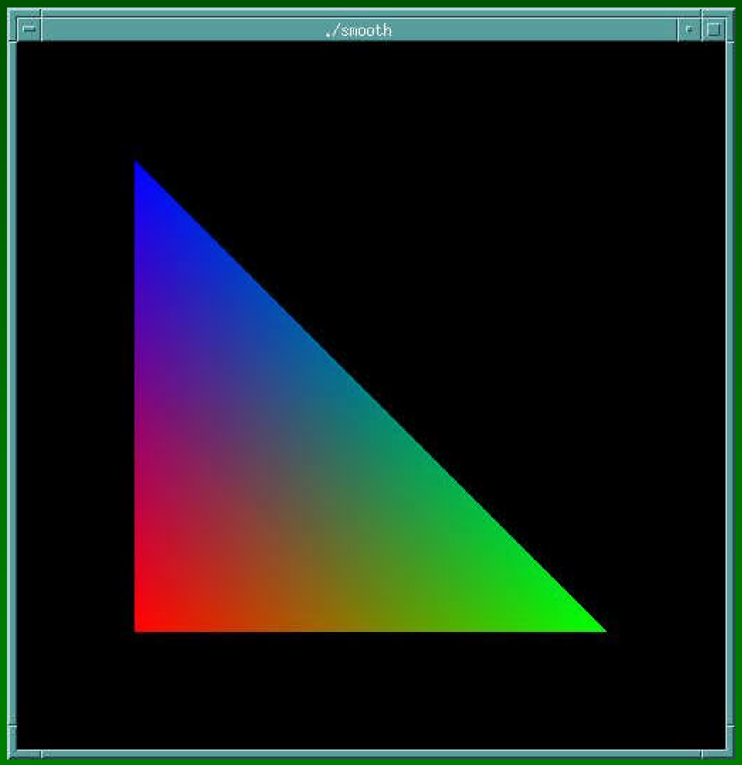
The Image

- Color of last vertex with flat shading

glShadeModel(GL_FLAT)



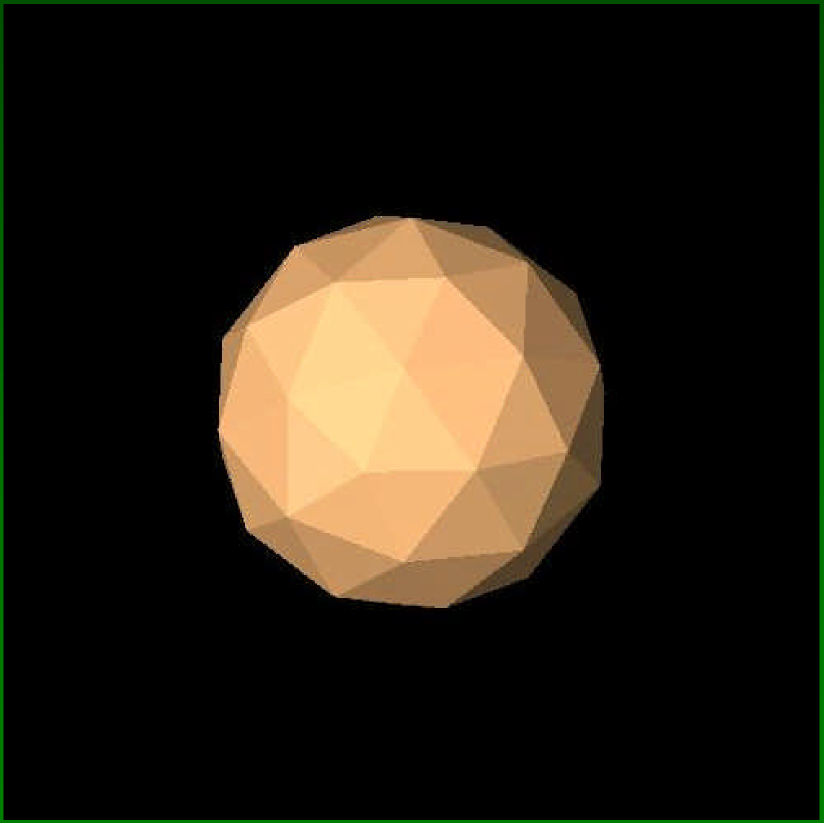
glShadeModel(GL_SMOOTH)



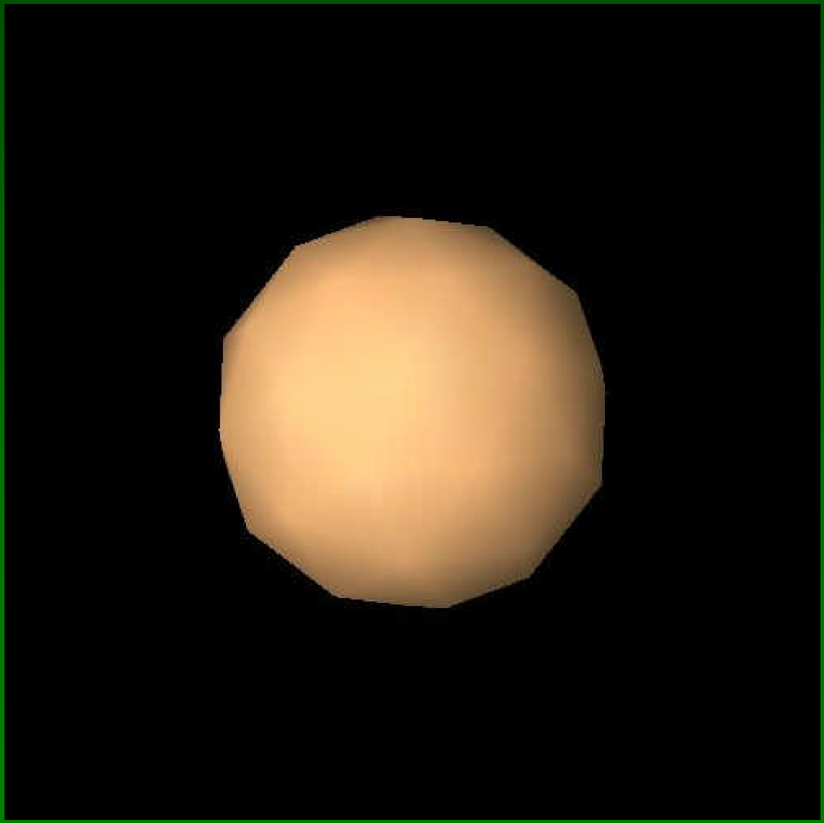
Preview: Smooth Shading

- Approximating a sphere

Flat Shading



Smooth Shading



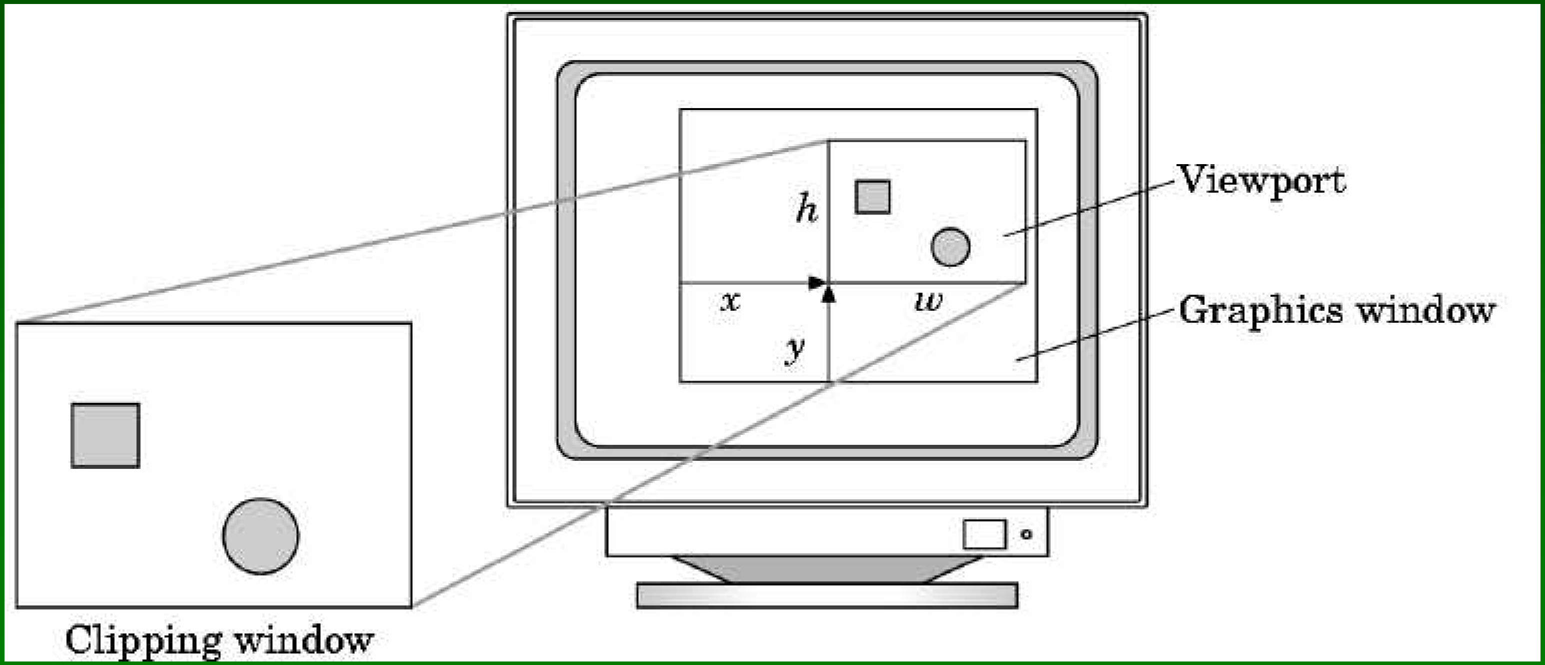
Projection

- Mapping world to screen coordinates

```
void reshape(int w, int h)
{
  glViewport (0, 0, (GLsizei) w, (GLsizei) h);
  glMatrixMode (GL_PROJECTION);
  glLoadIdentity ();
  if (w <= h)
    gluOrtho2D (0.0, 30.0, 0.0, 30.0 * (GLfloat) h/(GLfloat) w);
  else
    gluOrtho2D (0.0, 30.0 * (GLfloat) w/(GLfloat) h, 0.0, 30.0);
  glMatrixMode(GL_MODELVIEW);
}
```

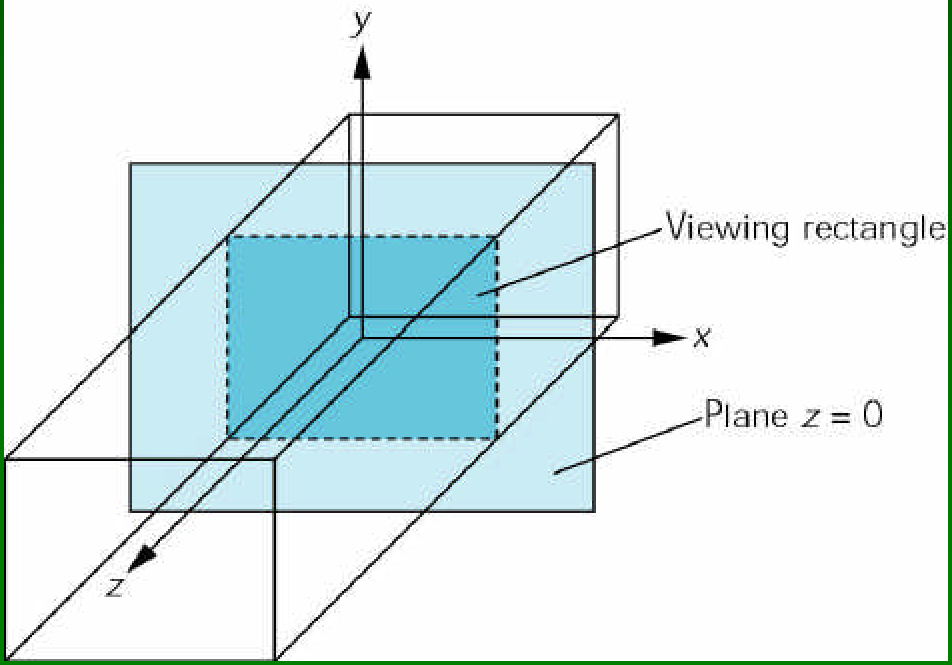
Viewport

- Determines clipping in window coordinates
- glViewPort(x, y, w, h)



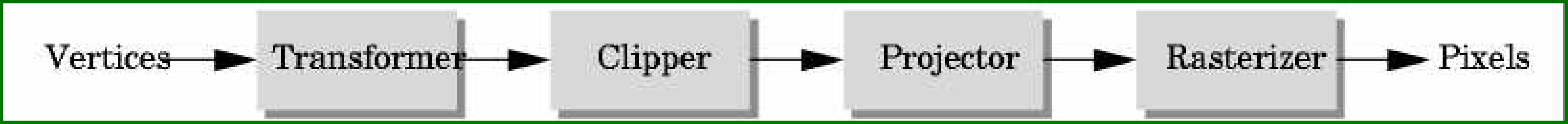
Orthographic Projection

- 2D and 3D versions
- glOrtho2D(left, right, bottom, top)
- In world coordinates!



Summary

1. A Graphics Pipeline
2. The OpenGL API
3. Primitives: vertices, lines, polygons
4. Attributes: color
5. Example: drawing a shaded triangle



Reminder

- Programming Assignment 1 out today
- Due in two weeks
- Compilation instructions on course page together with assignment
- Carefully follow account setup instructions for graphics lab (WeH 5336)