Computer Graphics -Basics of OpenGL

Junjie Cao @ DLUT

Spring 2016

http://jjcao.github.io/ComputerGraphics/

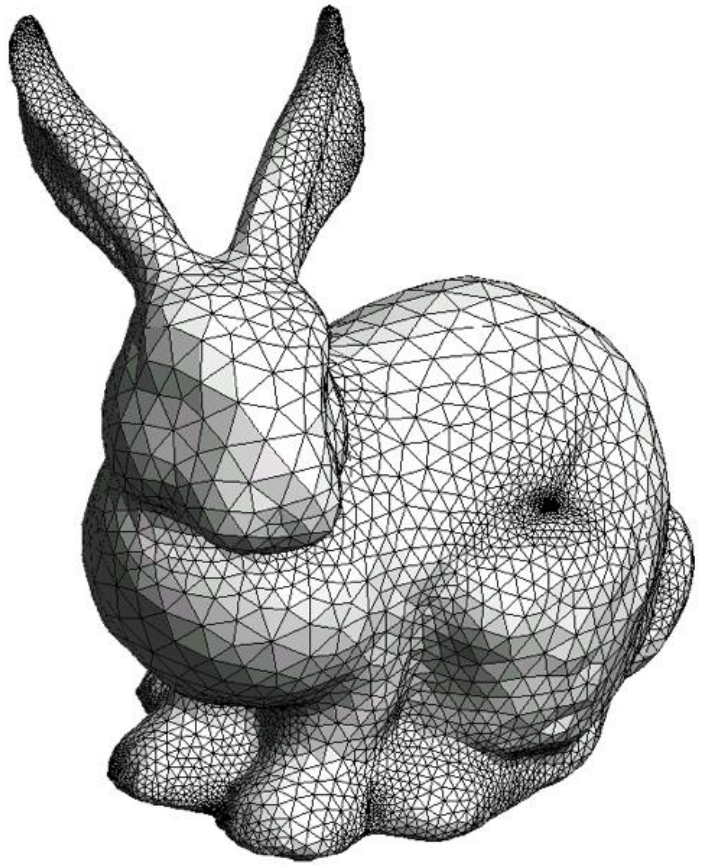
Primitives

- Specified via vertices

- General scheme

```
glBegin(type);
    glVertex3f(x1,y1,z1);
    …
    glVertex3f(xN,yN,zN);
glEnd();
```

- **type** determines interpretation of vertices
- Can use glVertex2f(x,y) in 2D



OpenGL Geometric Drawing Primitives

- OpenGL geometric primitives can create a set of points, a line, or a polygon from vertices

- OpenGL support **ten** types of primitives

- A drawing primitive must start with
  glBegin(Type);
- And finish with
  glEnd();

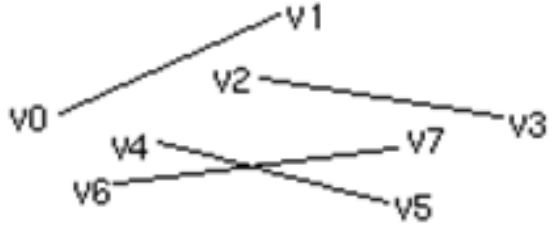
- Calls to other functions are allowed betwen glBegin(Type) and glEnd()

- Between them the primitive
- Type=GL_POLYGON

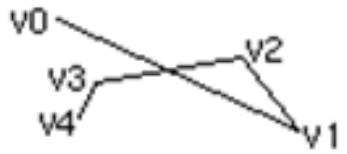
```
glBegin(GL_POLYGON);
    glVertex2f(-0.5, -0.5);
    glVertex2f(-0.5, 0.5);
    glVertex2f( 0.5, 0.5);
    glVertex2f( 0.5, -0.5);
glEnd();
```

OpenGL Geometric Drawing Primitives (cont)

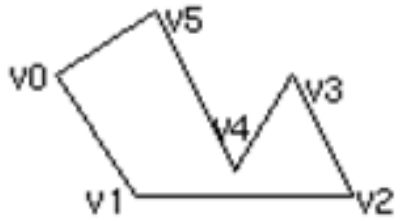
GL_POINTS



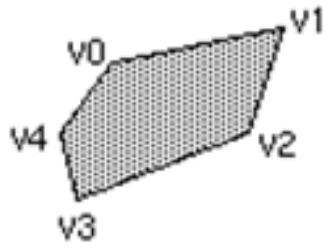
GL_LINES



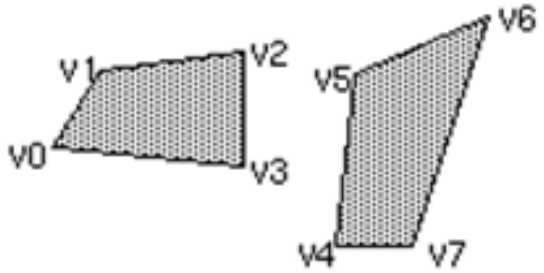
GL_LINE_STRIP



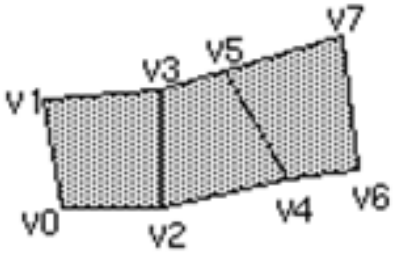
GL_LINE_LOOP



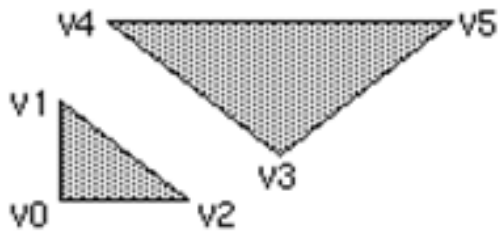
GL_POLYGON



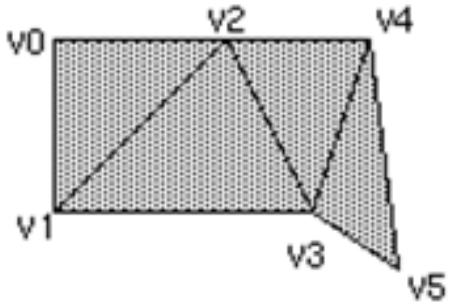
GL_QUADS



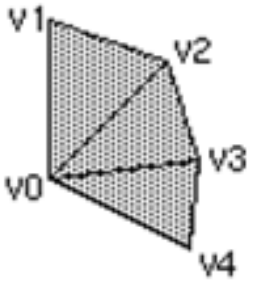
GL_QUAD_STRIP



GL_TRIANGLES



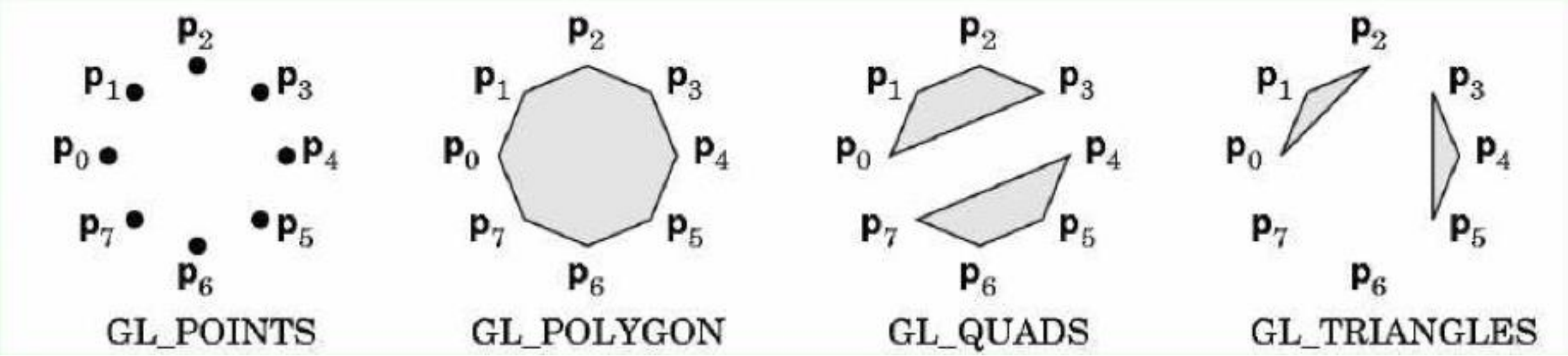
GL_TRIANGLE_STRIP



GL_TRIANGLE_FAN

Polygons

- **Polygons enclose an area**

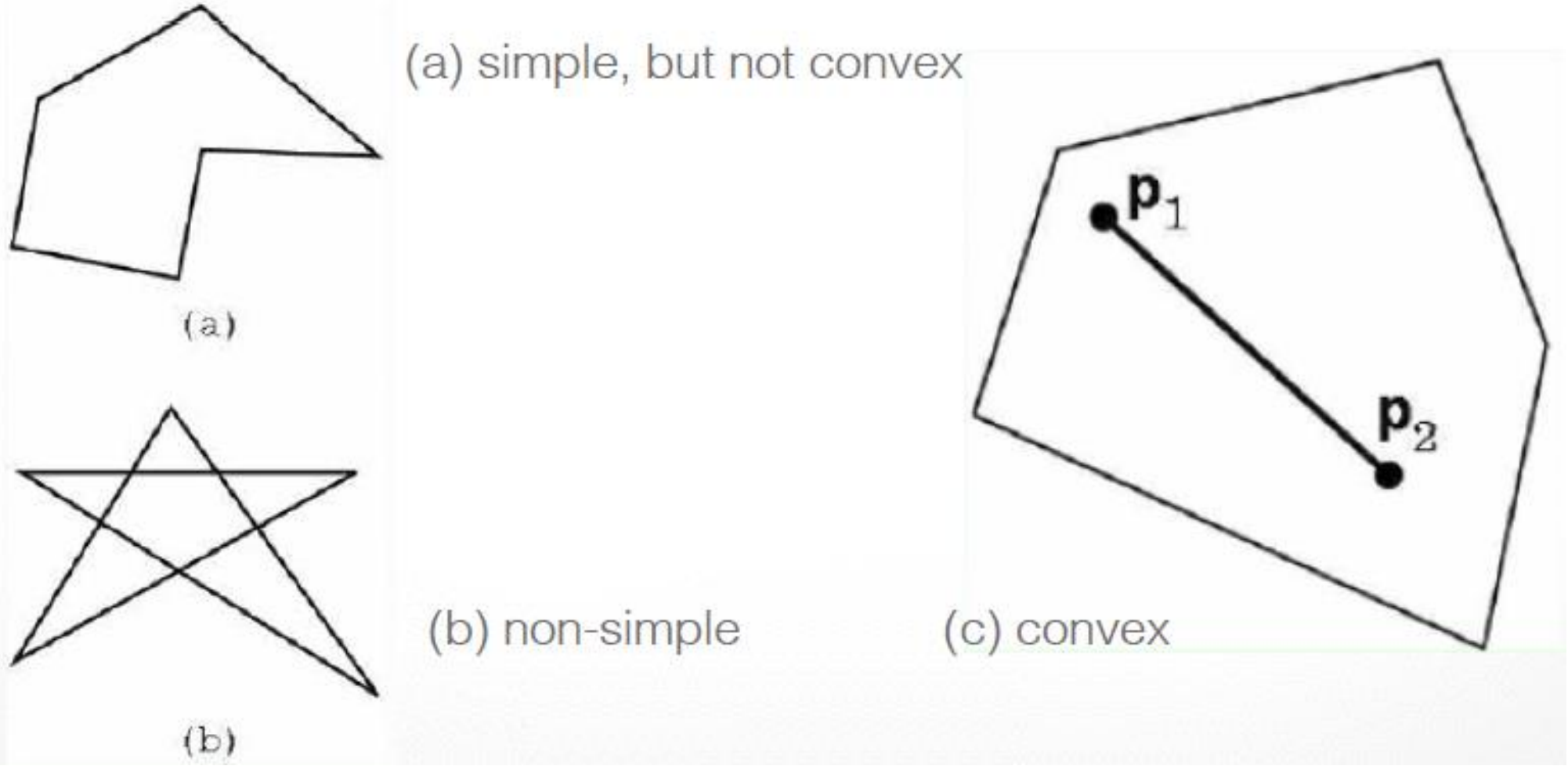


- Rendering of area (fill) depends on attributes

- **All vertices must be in one plane in 3D**

Polygons Restrictions

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



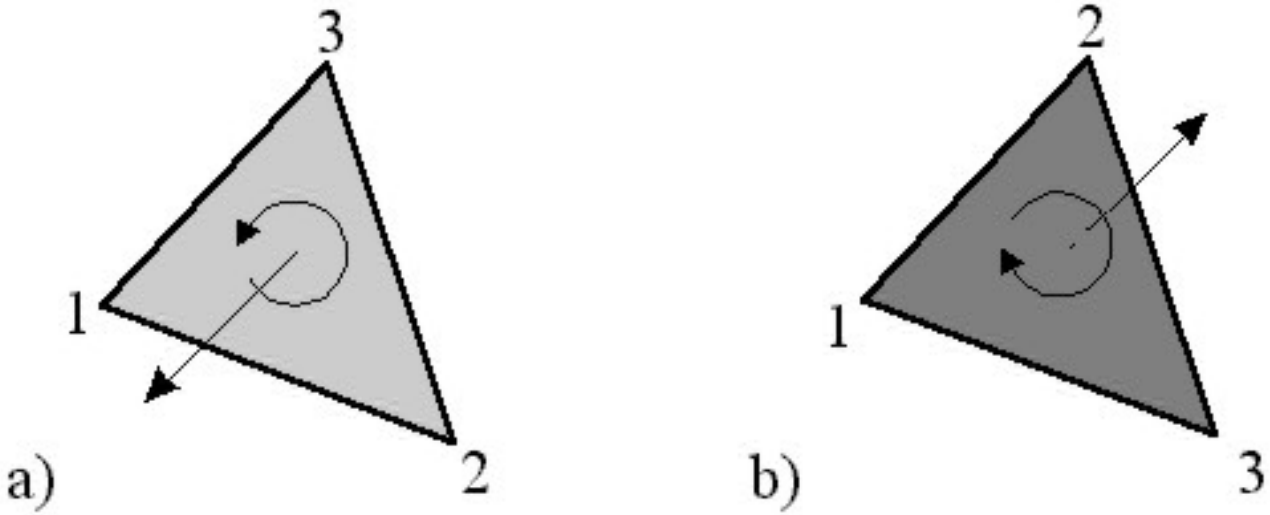
Why **Polygons Restrictions?**

- 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 (**tesselation**)
- **Triangles are most efficient**

Front/Back Rendering

- Polygons have a **front and a back**, possibly with **different attributes**!
- The **ordering of vertices** in the list determines which is the front side:

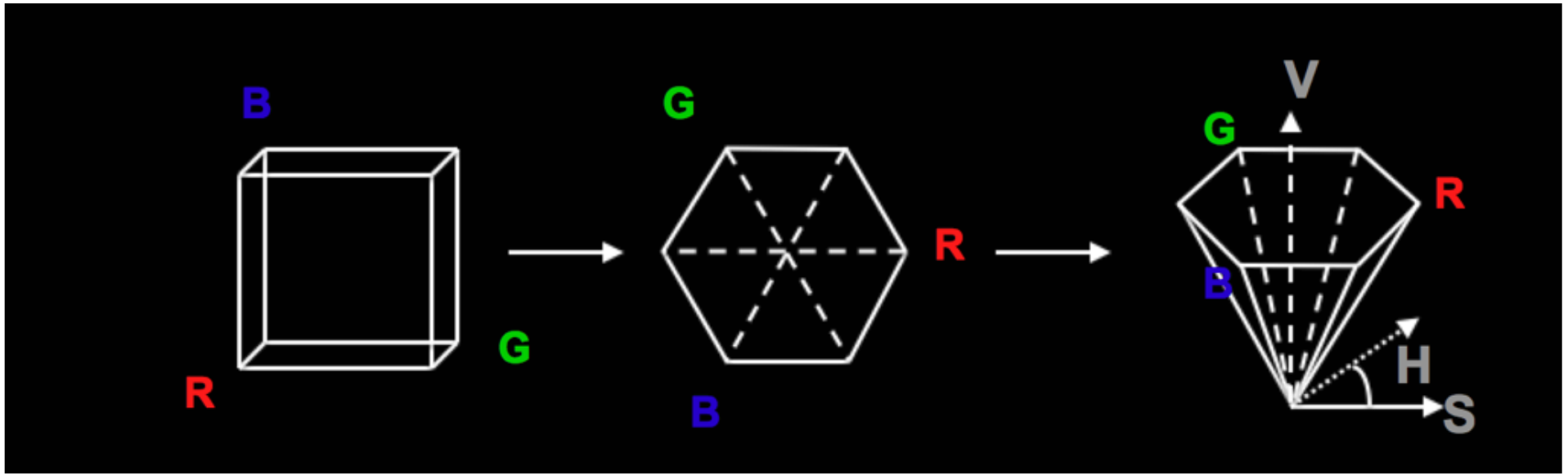
```
glBegin(GL_POLYGON);
    glVertex2f(-0.5, -0.5);
    glVertex2f(-0.5, 0.5);
    glVertex2f( 0.5, 0.5);
    glVertex2f( 0.5, -0.5);
glEnd();
```



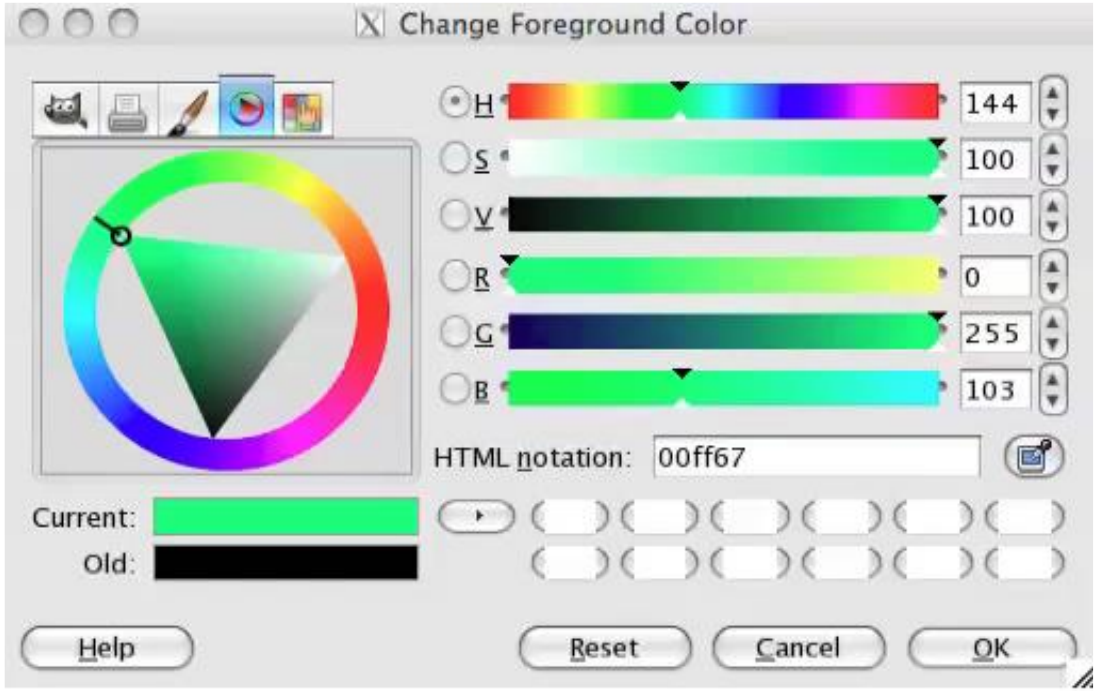
Attributes: Color, Shading, Reflections

- Part of the OpenGL **state**
- Set **before** primitives are drawn
- **Remain in effect until changed!**

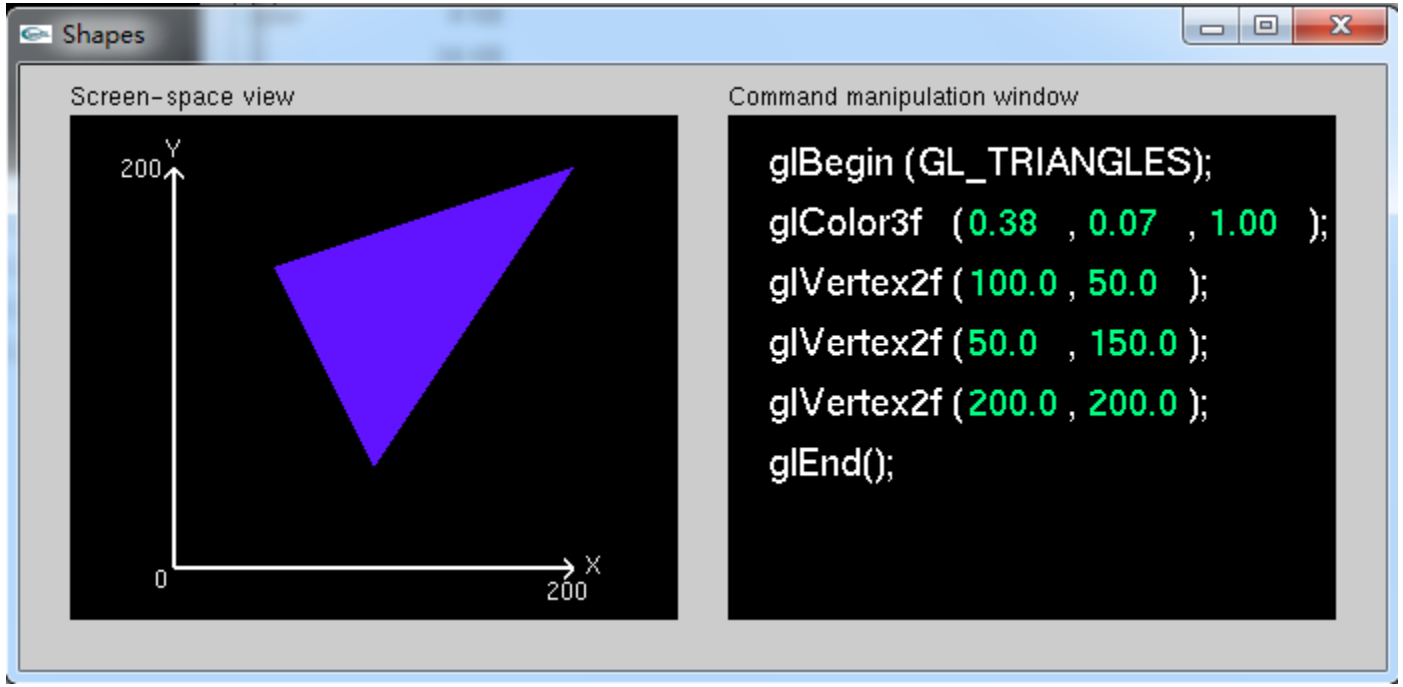
RGB vs HSV



- **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?
- Other formats for **movies and printing**



OpenGL 1.x: Geometric Drawing Primitives (cont)



Nate_Robins_tutorials: Shapes

Draw a complicated 3D Object in OpenGL 1.x

```
void DrawMeshWire( CMesh *m )
  glBegin( GL_TRIANGLES );
      for ( int i = 0; i < m->numFaces * 3; i+=3 )  {
          glVertex3f( m->vertex[m->faces[i]*3], m->vertex[m->faces[i]*3+1],
                                          m->vertex[m->faces[i]*3+2] );
          glVertex3f( m->vertex[m->faces[i+1]*3], m->vertex[m->faces[i+1]*3+1],
                                          m->vertex[m->faces[i+1]*3+2] );
          glVertex3f( m->vertex[m->faces[i+1]*3], m->vertex[m->faces[i+1]*3+1],
                                          m->vertex[m->faces[i+1]*3+2] );
      }
  glEnd();
```

Use GLUT (OpenGL Utility Toolkit)

- For fast prototyping, you can use GLUT to interface with different window systems
- GLUT is a window independent API – programs written using OpenGL an d GLUT can be ported to X windows, MS windows, and Macintosh with no effort
- GLUT does not contain all the bells and whistles though (no sliders, no dialog boxes, no menu bar, etc)

Example: Drawing a shaded polygon

- Initialization: the “main” function

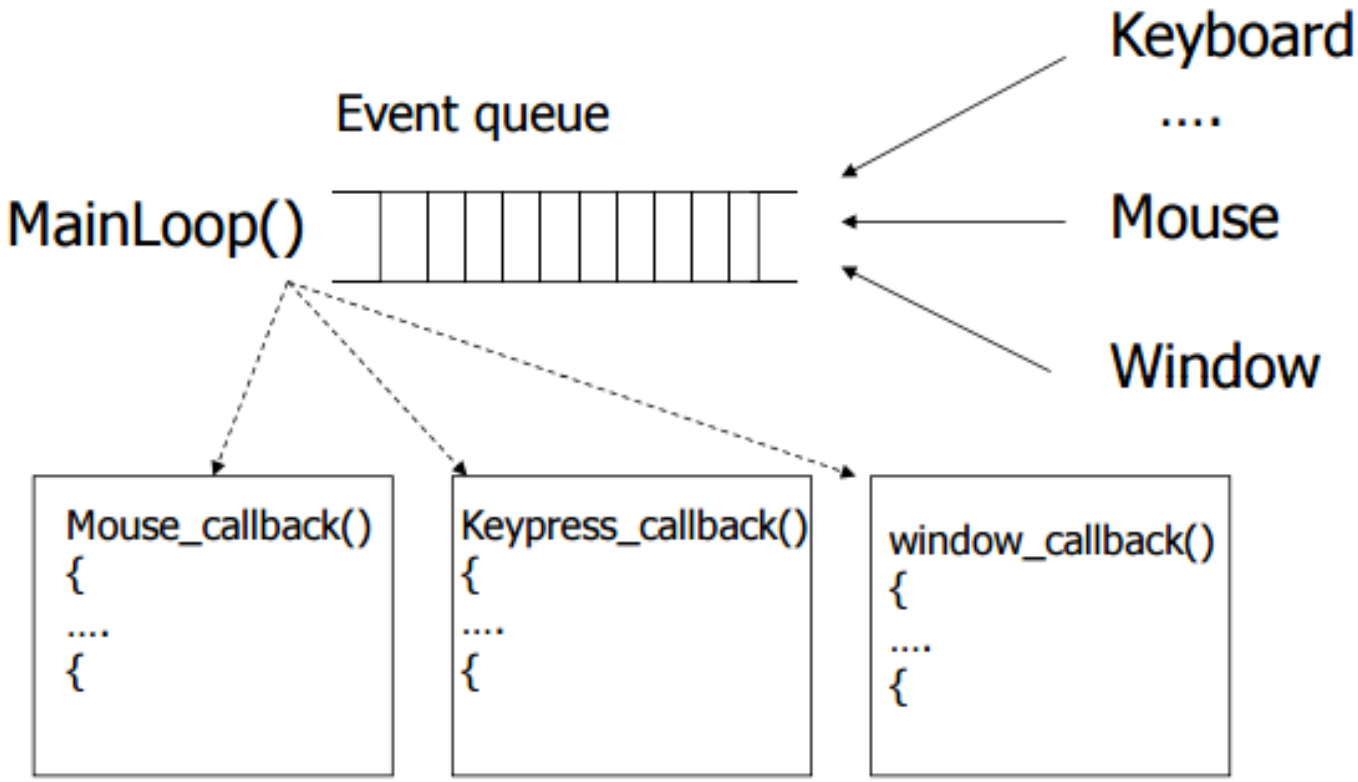
```
int main(int argc, char ** argv)
{
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB);
    glutInitWindowSize(500,500);
    glutInitWindowPosition(100,100);
    glutCreateWindow(argv[0]);
    init();
    …
```

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()
{
    glClearColor (0.0,0.0,0.0,0.0);
    // glShadeModel (GL_FLAT);
    glShadeModel (GL_SMOOTH);
}
```

The Display Callback

- The routine where you render the object

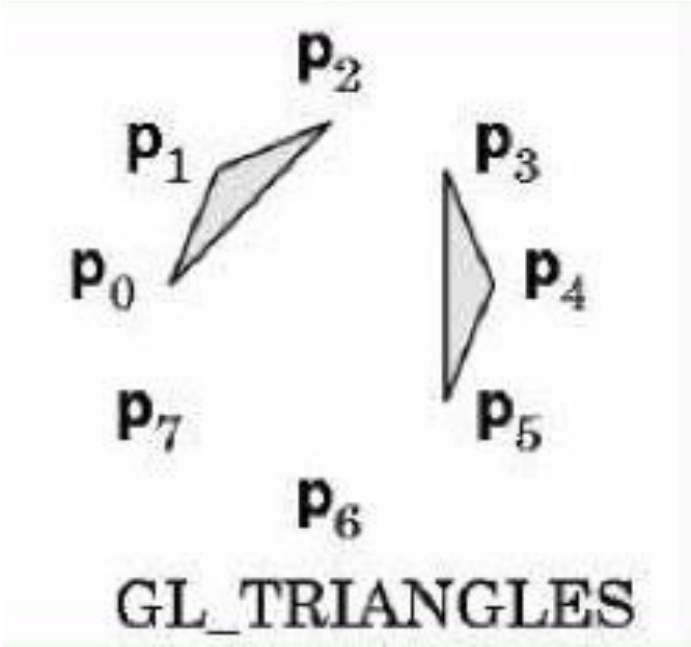
- Install with glutDisplayFunc(display)

```
void display()
{
    glClear(GL_COLOR_BUFFER_BIT); // clear buffer
    setupCamera(); // set up camera
    triangle(); // draw triangle
    glutSwapBuffers(); // force display
}
```

Drawing in OpenGL 1.x

- In world coordinates; remember state!

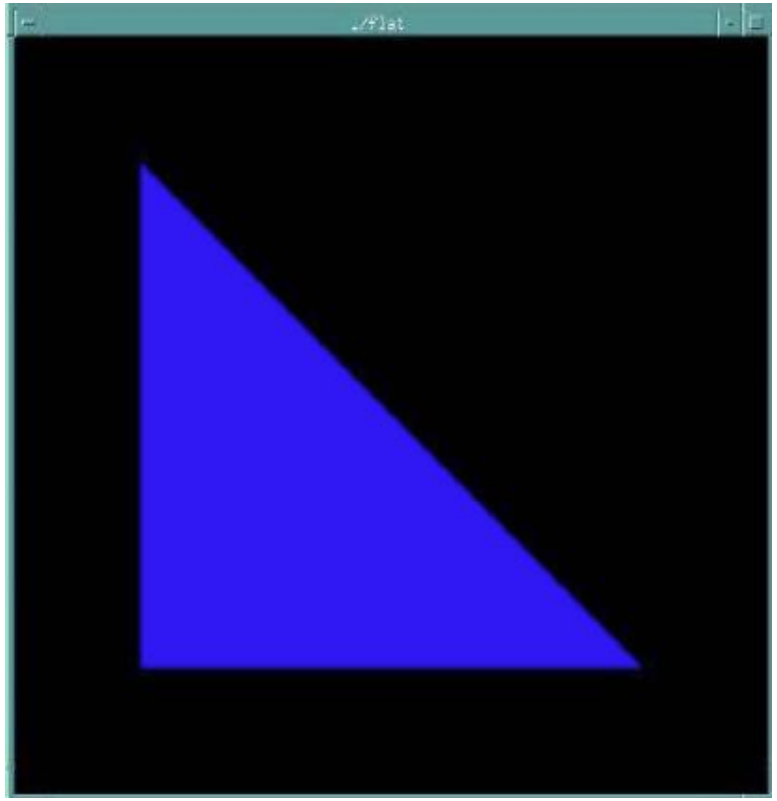
```
void triangle()
{
    glBegin(GL_TRIANGLES);
        glColor3f(1.0,0.0,0.0); // red
        glVertex2f(5.0,5.0);
        glColor3f(0.0,1.0,0.0); // green
        glVertex2f(25.0,5.0);
        glColor3f(0.0,0.0,1.0); // blue
        glVertex2f(5.0,25.0);
    glEnd();
}
```



The Image

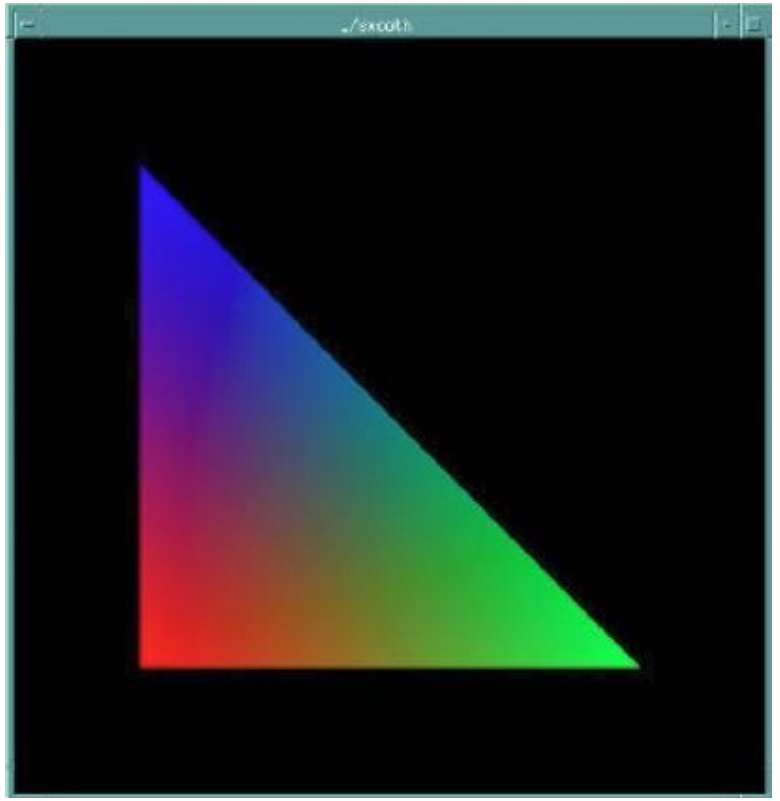
glShadeModel(GL_FLAT)

color of last vertex



glShadeModel(GL_SMOOTH)

each vertex separate color smoothly interpolated



Flat vs Smooth Shading

glShadeModel(GL_FLAT)

glShadeModel(GL_SMOOTH)

Projection

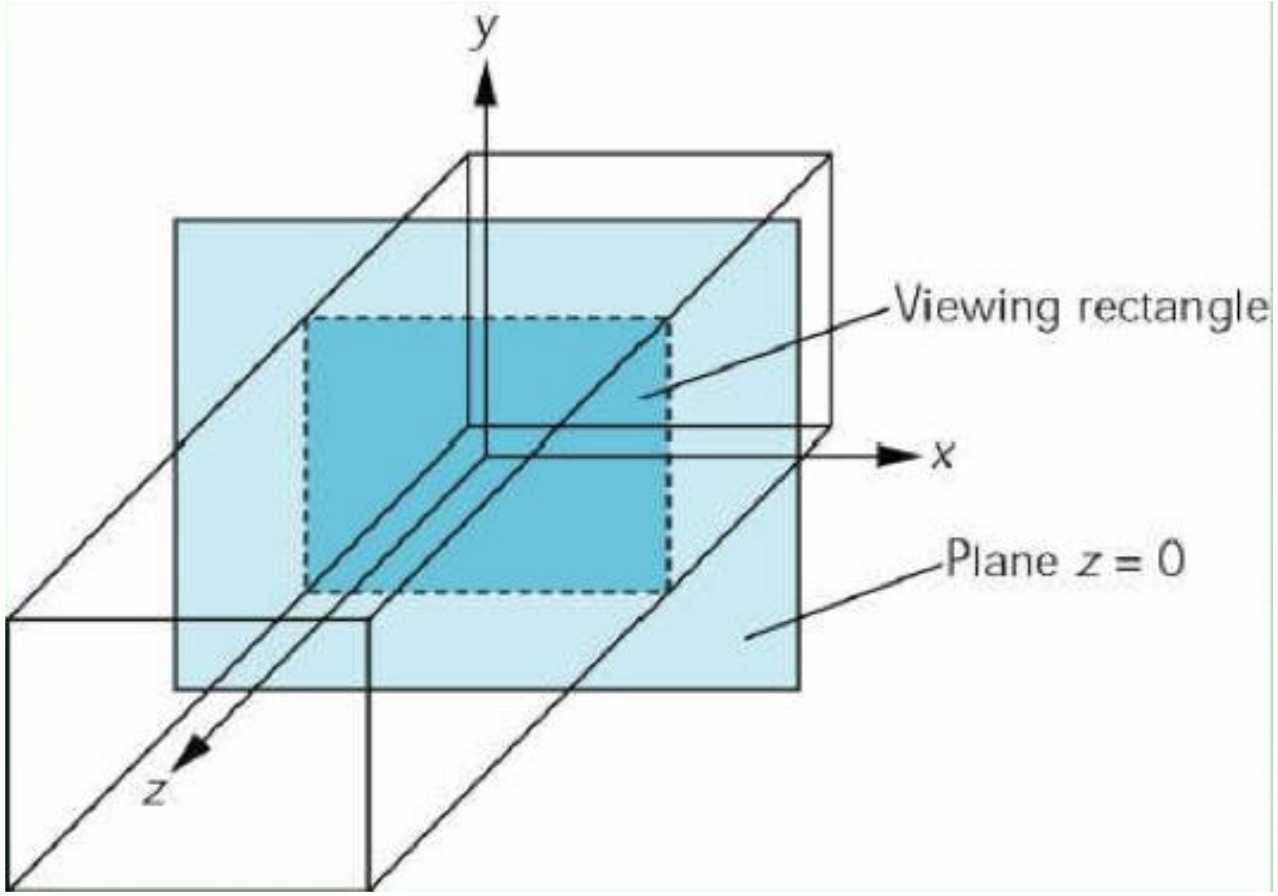
- Mapping world to screen coordinates

```
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);
}
```

GL_PROJECTION and GL_MODELVIEW will be discussed later

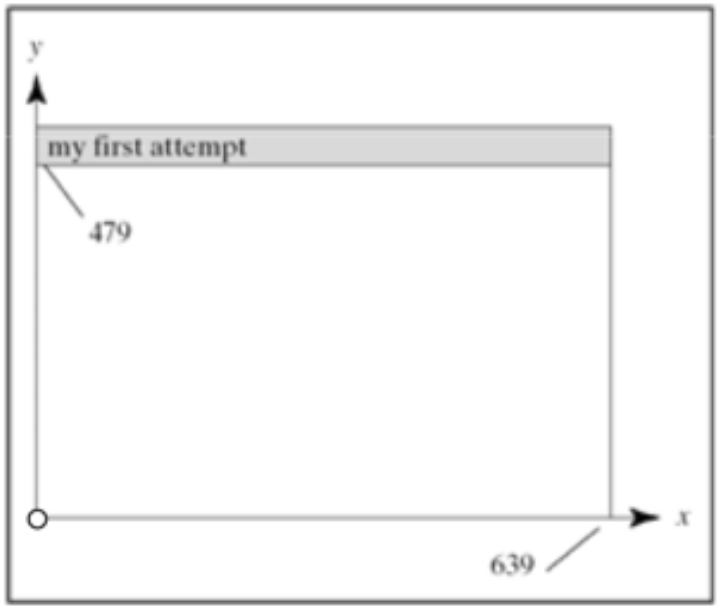
Orthographic Projection

- glOrtho2D(left, right, bottom, top)
- In world coordinates!



Screen coordinates

- Bottom left corner is origin

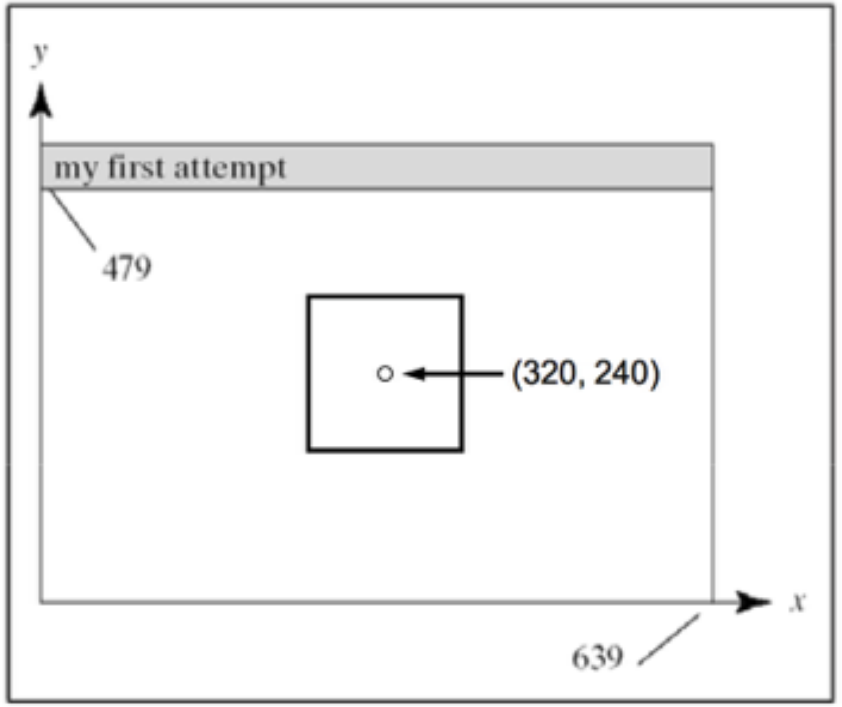


from Hill

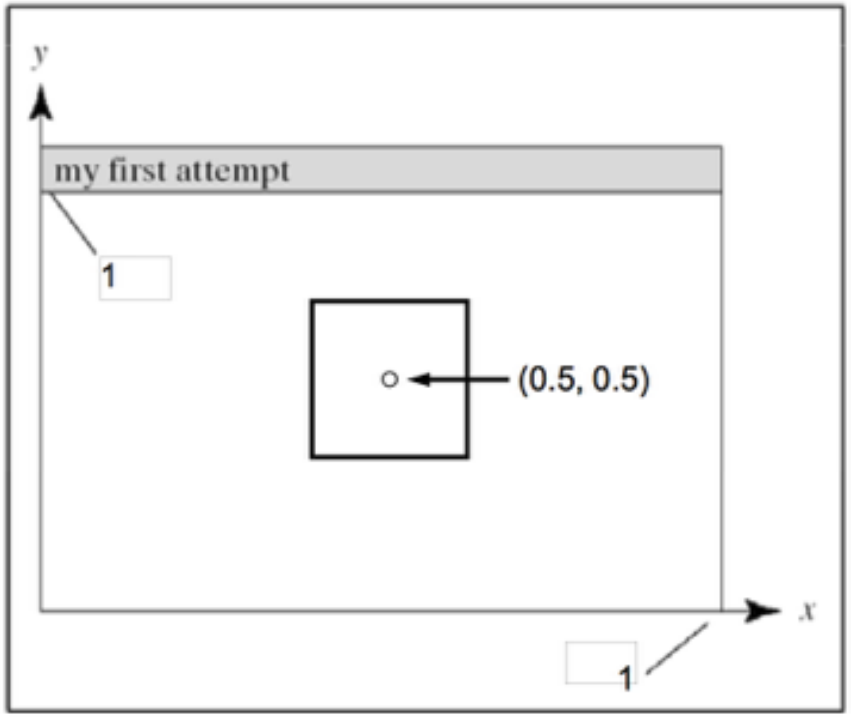
- `gluOrtho2D()` sets the units of the screen coordinate system
  - `gluOrtho2D(0, w, 0, h)` means the coordinates are in units of pixels
  - `gluOrtho2D(0, 1, 0, 1)` means the coordinates are in units of "fractions of window size" (regardless of actual window size)

Screen coordinates

`gluOrtho2D(0, 640, 0, 480)`

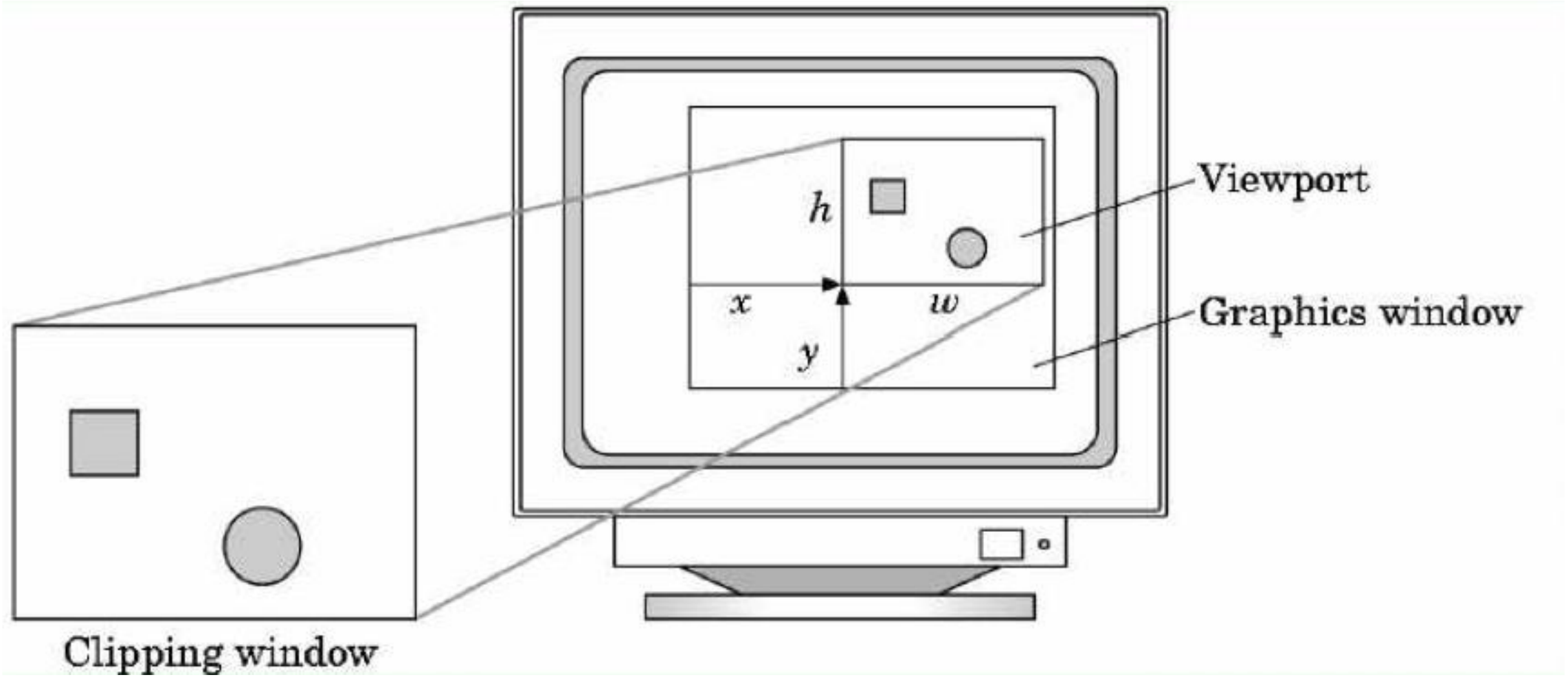


`gluOrtho2D(0, 1, 0, 1)`



Viewport

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



OpenGL Lighting

- Provides a limited variety of light sources
- We can have **point sources, spotlights and ambient** sources
  - Each source has separate diffuse, specular and ambient RGB parameters
- Materials are modeled in a complementary manner
  - For each surface separate **ambient, diffuse and specular** components must be used
- Lighting calculations must be enabled and each light source must be enabled individually
  - glEnable(GL_LIGHTING);
  - glEnable(LIGHT1);
- Enabling lighting makes OpenGL to do the shading calculations
- Once **lighting is enabled**, colours assigned by **glColor() are no longer valid**

Specifying a Light Source

- Light sources have a number of properties, such as colour, position, and direction

- The OpenGL function to create a light source is
  - void **glLight**(GLenum light, GLenum param, TYPE value);

- The directional light source allows to associate three different colour-related parameters with any particular light
  - **GL_AMBIENT, GL_DIFFUSE, and GL_SPECULAR**

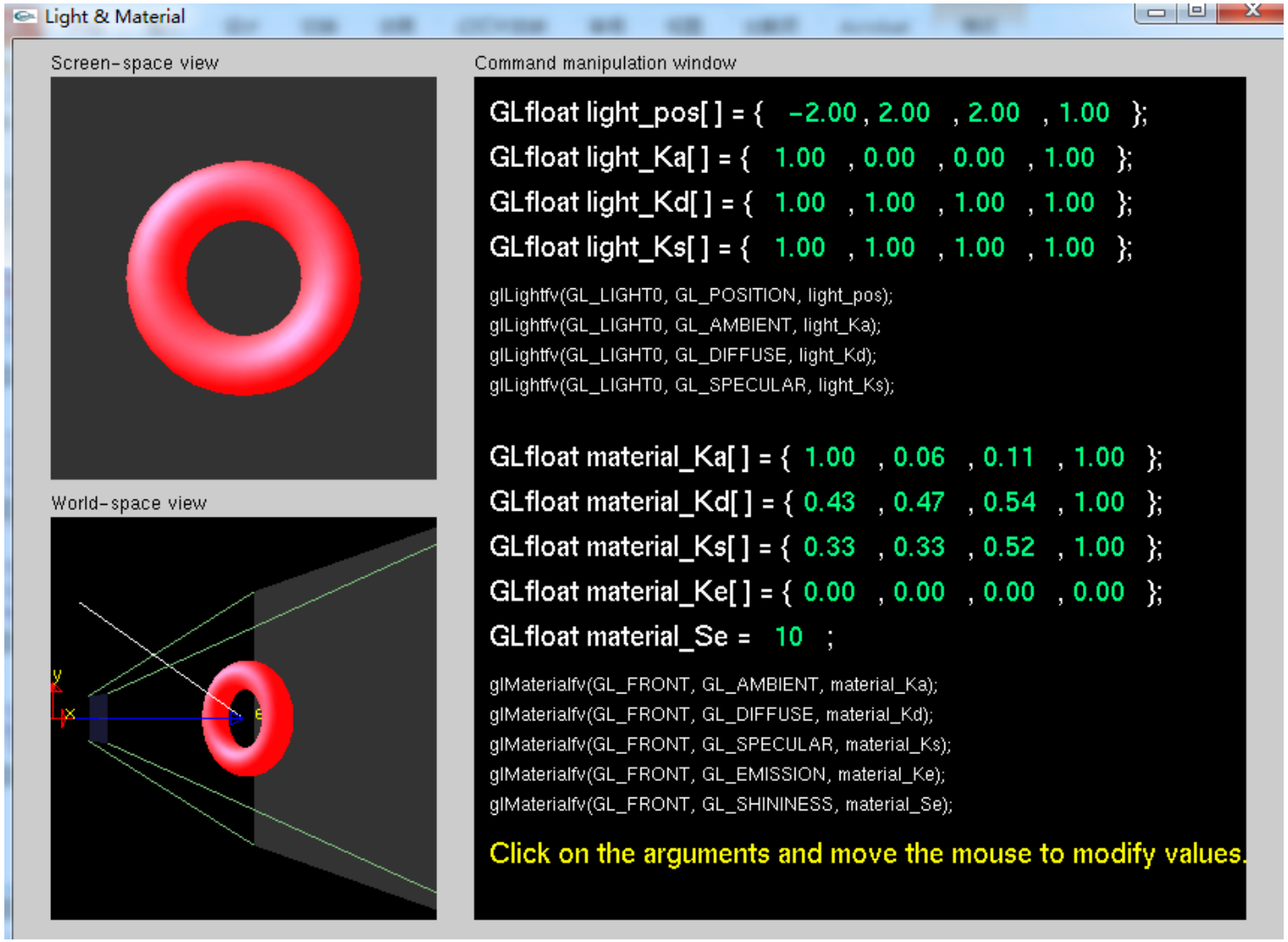
- The positional light source need to define
  - the **location** (GL_LOCATION), and the **colour** (ambient, diffuse and specular)

- Also can have a positional light source act as a spotlight

Specifying Material Properties

- Material properties match the lighting properties
  - A material has reflectivity properties for each type of light
- The basic function for setting material properties is:
  - void **glMaterial**(GLenum face, GLenum name, TYPE value);
- **Diffuse** and **Ambient** Reflection
  - The GL_DIFFUSE and GL_AMBIENT parameters set with glMaterial*() affect the colour of the diffuse and ambient light reflected by an object
- **Specular** Reflection
  - Specular reflection from an object produces highlights.
  - OpenGL allows you to set the effect that the material has on reflected light (with GL_SPECULAR) and control the size and brightness of the highlight (with GL_SHININESS
- **Emission**
  - By specifying an RGBA color for GL_EMISSION, you can make an object appear to be giving off light of that color

Light & Material



Nate_Robins_tutorials: lightmaterial

Specifying Geometric primitives

- OpenGL 1.x

```
glBegin(primType); //e.g. GL TRIANGLES
glVertex3f(x,y,z);
…
glEnd();
```

- OpenGL 3.x & up: use Vertex Buffer Objects (VBO)
  - Much more efficient since it allows the geometry to be stored in the graphics card and reduce the number of function calls

```
glGenBuffers(1, &vboHandle); // create a VBO handle
glBindBuffer(GL_ARRAY_BUFFER, vboHandle); // bind the handle to the current VBO
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices,
GL_STATIC_DRAW); // allocate space and copy the data over
…
```

GLEW: The OpenGL Extension Wrangler Library

- A cross-platform open-source C/C++ extension loading library loading library (windows, OS X, Linux, FreeBSD)
- Why GLEW is needed?
- It is not possible to link directly to functions that are provided in newer version of OpenGL. In windows, this means OpenGL 1.2 and up
- GLEW does the tedious work to helps you find the function pointers (addresses of the functions) for OpenGL extensions
- GLEW can also help you to check if a particular OpenGL extension is available on your machine

GLEW usage

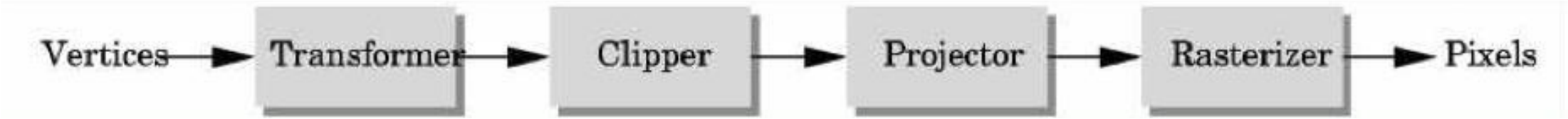
- Make sure you have GLEW on your machine
- Include the glew header file
  - #include <GL/glew.h>
- Initialize glew before calling any opengl functions
  - GLenum err = glewInit();
  - if (err != GLEW_OK) printf(" Error initializing GLEW! \n");
  - else printf(else printf("Initializing GLEW succeeded!\n);
- Check OpenGL features, for example, shaders
  - if (! GLEW_ARB_vertex_program)
    - printf(" ARB vertex program is not supported!!\n");
  - else printf(" ARB vertex program is supported!!\n");

“Hello world!” finished

More details will follow

Summary

- **A Graphics Pipeline**
- The OpenGL **API**
- **Primitives**: vertices, lines, polygons
- **Attributes**: color
- Example: drawing a **shaded triangle**



Suggestions

- Most people do old OGL because they found an out of date tutorial online.
- Modern OpenGL (Shaders & VBOs [Vertex Buffer Objects])