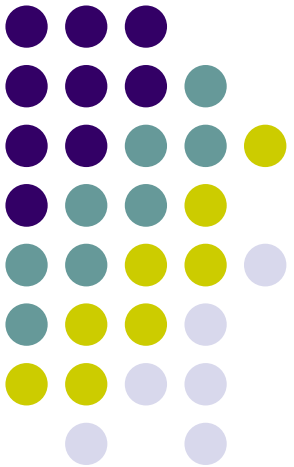
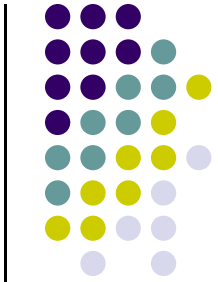
Computer Graphics
CS 543 – Lecture 1 (Part 3)

Prof Emmanuel Agu

Computer Science Dept.

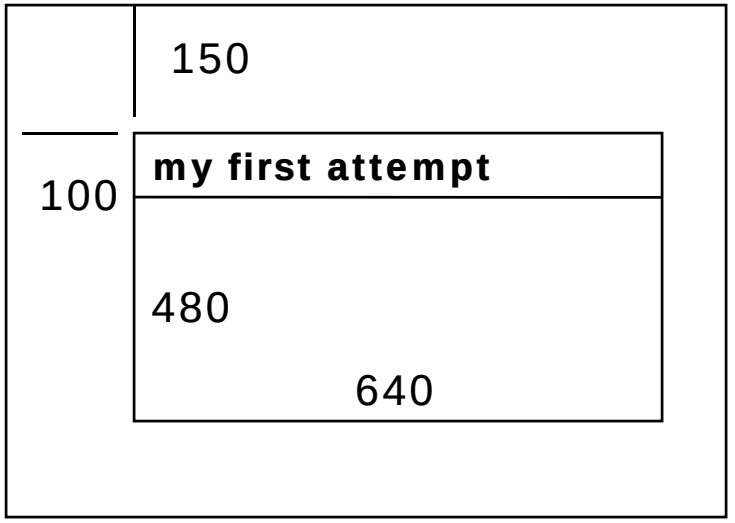
Worcester Polytechnic Institute (WPI)

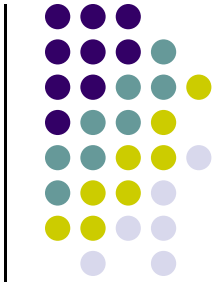




Recall: OpenGL Skeleton

```
void main(int argc, char** argv){
    // First initialize toolkit, set display mode and create window
    glutInit(&argc, argv);   // initialize toolkit
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(640, 480);
    glutInitWindowPosition(100, 150);
    glutCreateWindow("my first attempt");
    glewInit( );

    // … now register callback functions
    glutDisplayFunc(myDisplay);
    glutReshapeFunc(myReshape);
    glutMouseFunc(myMouse);
    glutKeyboardFunc(myKeyboard);

    myInit( );
    glutMainLoop( );
}
```





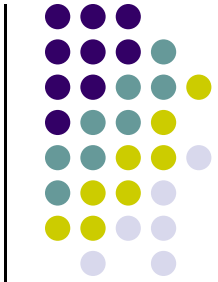
Old Way: OpenGL Drawing

- Drawing done in ***display*** function
- ***Display*** function called once when program starts
- Recall: First register callback in main( ) function

```
glutDisplayFunc( myDisplay );
```

- Then, implement **`myDisplay`** function

```
void myDisplay( void )
{
    // put drawing commands here

}
```



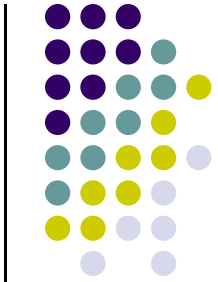
Old way: Drawing Primitives

- Draw points, lines, polylines, polygons
- Primitives specified using glBegin, glEnd format:

```
glBegin(primType)

    // define your primitives here

glEnd( )
```

- primType: **GL_POINTS, GL_LINES**, **GL_POLYGON….**

Old way: Drawing Example



- Example: draw three dots. How?
  - Specify vertices
  - Vertices connected in manner determined by **`primType`**
- Immediate mode
  - Generate points, render them (points not stored)

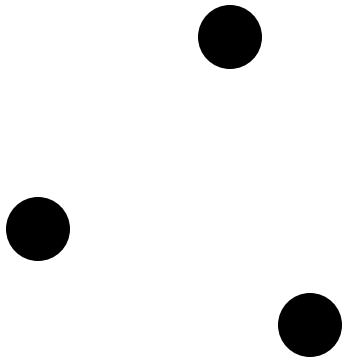
```
void myDisplay( void )
{
   ....

   glBegin(GL_POINTS)
        glVertex2i(100,50);
        glVertex2i(100,130);
        glVertex2i(150, 130);
       glFlush( );
   glEnd( )
```

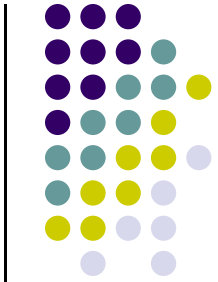
primType

Forces drawing to complete

x

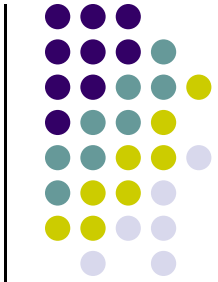
y





Immediate Mode Graphics

- Geometry specified by vertices
  - Locations in space(2 or 3 dimensional)
  - Points, lines, circles, polygons, curves, surfaces
- Immediate mode
  - Each time a vertex is specified in application, its location is sent to the GPU
  - Old style uses **glVertex**
  - Creates bottleneck between CPU and GPU
  - Removed from OpenGL 3.1

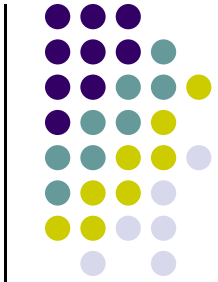


Better Way of Drawing: Retained Mode Graphics

1. Generate points
2. Store vertices into an array
3. Draw points from array using **`glDrawArray`**
4. First declare types for points and vectors

- Useful to declare types *point3* for <x,y> locations*, vec3* for <x,y,z> vector coordinates with their constructors
- put declarations in *header file vec.h*

```
#include “vec.h”

Vec3 vector1;
```

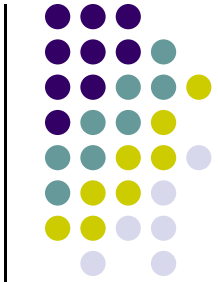


New Way of Drawing

- Generate points & store vertices into an array

```
point3 points[3] = { point2(100,50),
                     point2(100,130),
                     point2(150, 130); }
```

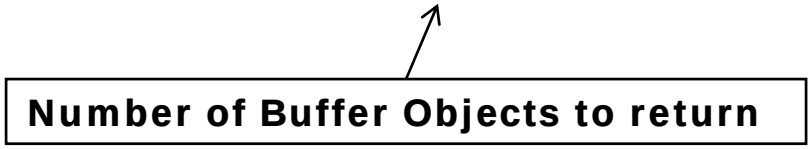
- Draw points from array using **`glDrawArray`**



Move points GPU memory

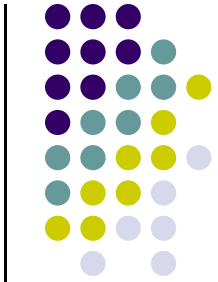
- Rendering from GPU memory significantly faster. Move data there
- Fast GPU memory for data called ***Buffer Objects***
- Three steps:
  1. Create VBO and give it name (unique ID number)

```
GLuint buffer;
glGenBuffers(1, &buffer); // create one buffer object
```



  2. Make created VBO currently active one

```
glBindBuffer(GL_ARRAY_BUFFER, buffer); //data is array
```



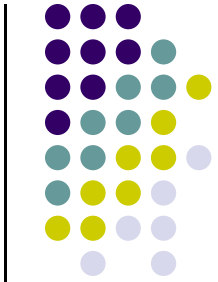
Move points GPU memory

3. Move **`points`** generated earlier to VBO

```
glBufferData(GL_ARRAY_BUFFER, buffer, sizeof(points),
points, GL_STATIC_DRAW ); //data is array
```

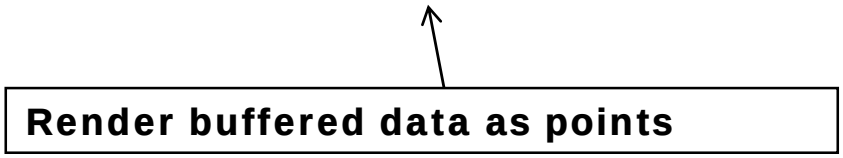
Data to be transferred to GPU memory (generated earlier)

- **`GL_STATIC_DRAW:`** buffer object data will be specified once by application and used many times to draw
- **`GL_DYNAMIC_DRAW:`** buffer object data will be specified repeatedly and used many times to draw



Draw points

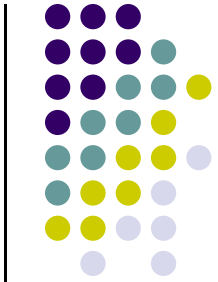
```
glDrawArrays(GL_POINTS, 0, N);
```



- Display function using **`glDrawArrays:`**

```
void mydisplay(void){
      glClear(GL_COLOR_BUFFER_BIT); // clear screen
      glDrawArrays(GL_POINTS, 0, N);
      glFlush( );        // force rendering to show
}
```

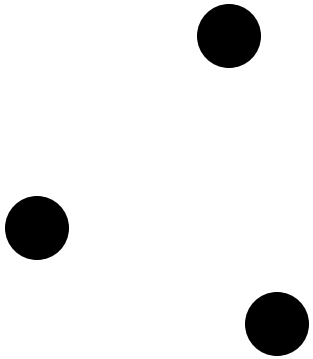
- Other possible arguments to **`glDrawArrays`** instead of **`GL_POINTS?`**



`glDrawArrays()` Parameters

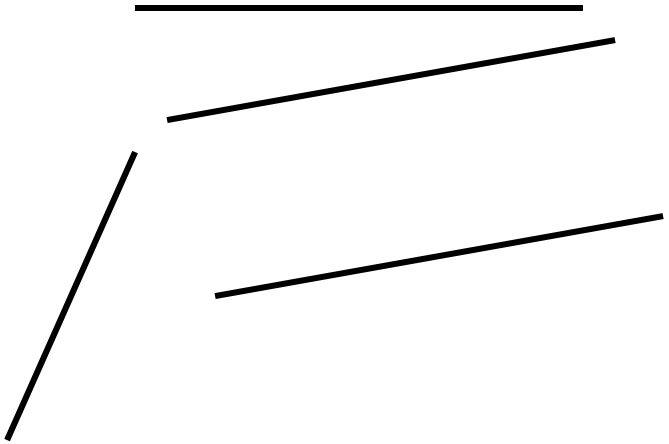
`glDrawArrays`(GL_POINTS, ….)

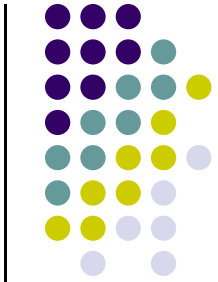
– draws dots



`glDrawArrays`((GL_LINES, …)

– Connect vertex pairs to draw lines

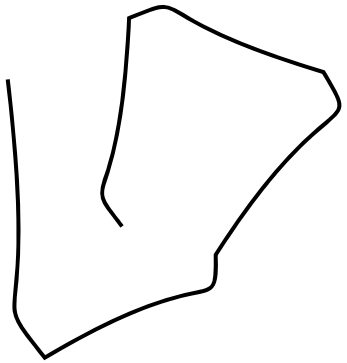




`glDrawArrays()` Parameters

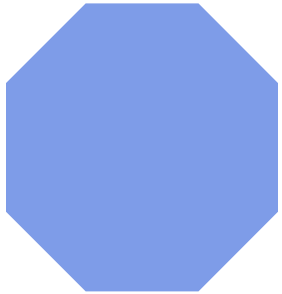
`glDrawArrays`(GL_LINE_STRIP,..)

– polylines



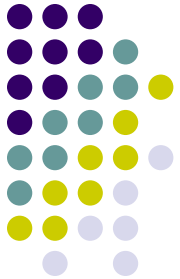
`glDrawArrays`(GL_POLYGON,..)

– convex filled polygon



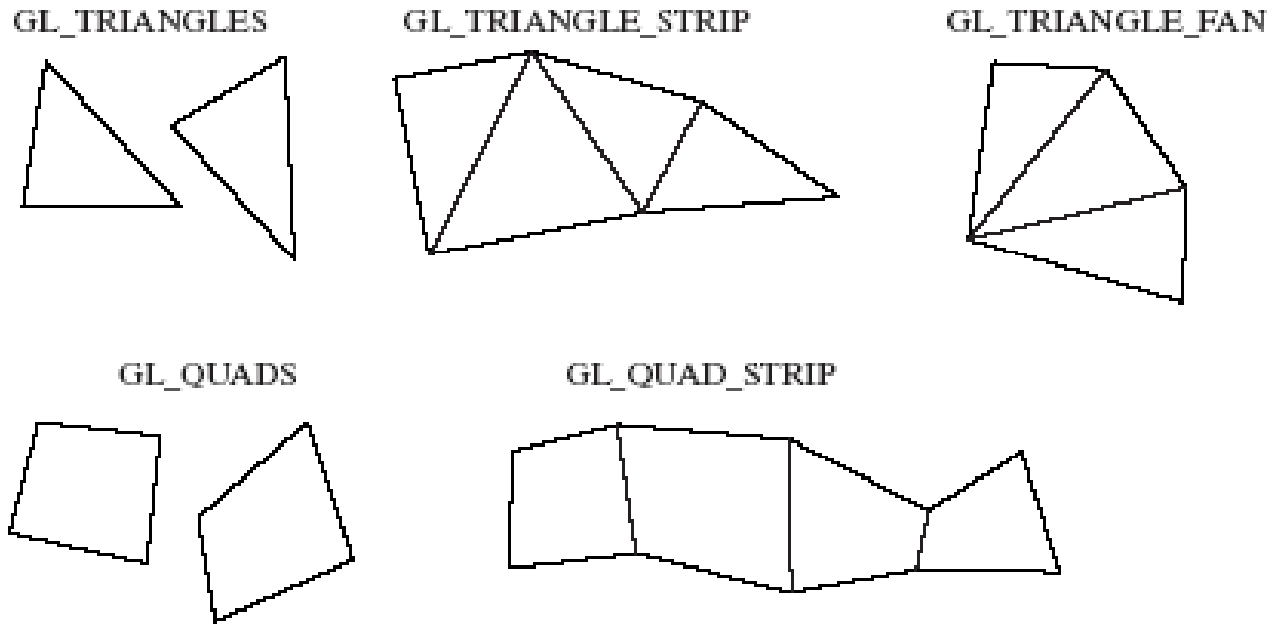
`glDrawArrays`(GL_LINE_LOOP)

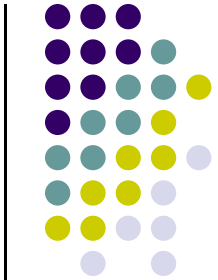
– Close loop of polylines
(Like GL_LINE_STRIP but closed)



`glDrawArrays()` Parameters

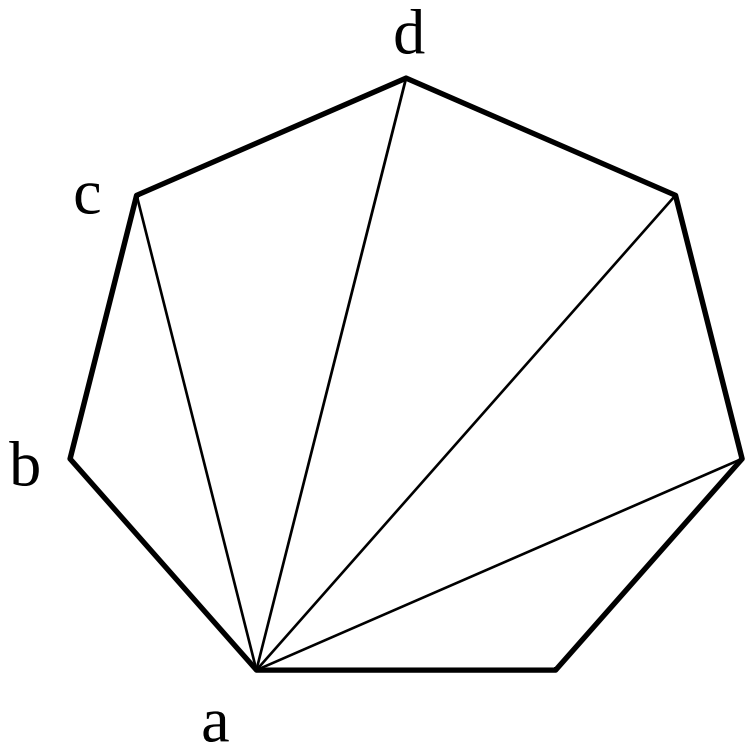
- Triangles: Connect 3 vertices
  - GL_TRIANGLES, GL_TRIANGLE_STRIP, GL_TRIANGLE_FAN
- Quad: Connect 4 vertices
  - GL_QUADS, GL_QUAD_STRIP



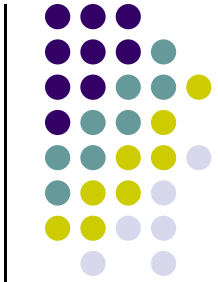


Triangulation

- Generally OpenGL breaks polygons down into triangles which are then rendered

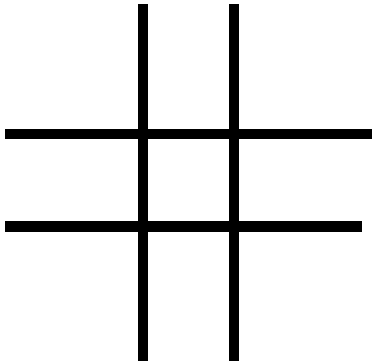


Line Attributes

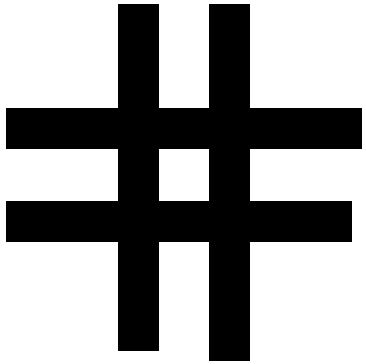


- Color, thickness, stippling.
- **glColor3f()** sets color.
- **glLineWidth(4.0)** sets thickness. Default thickness is 1.0.

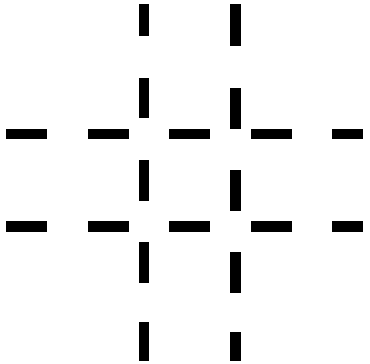
a). thin lines

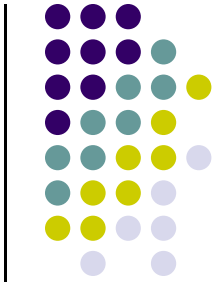


b). thick lines



c). stippled lines

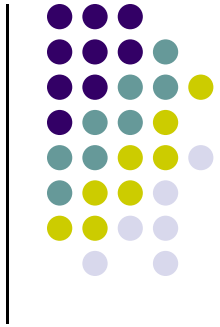


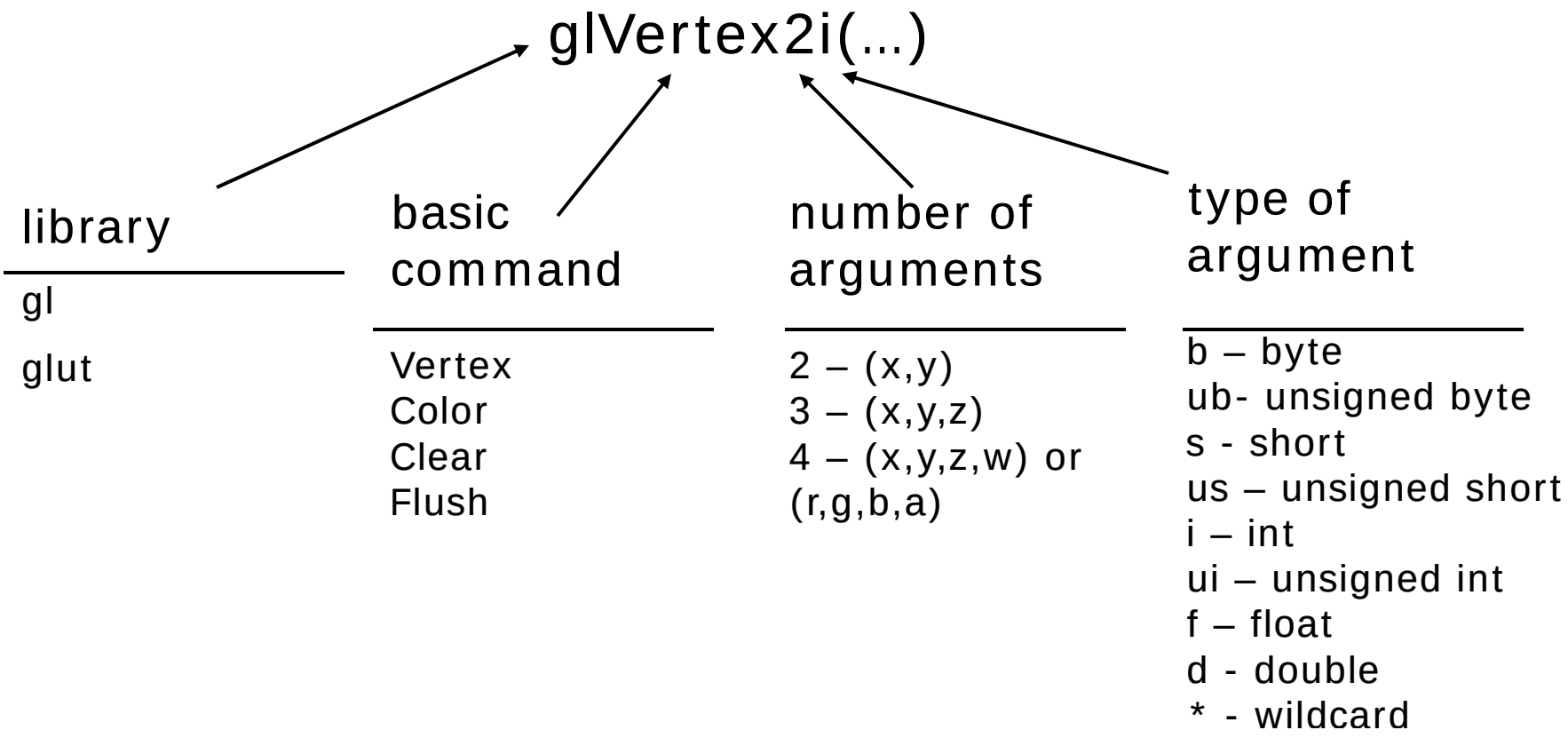


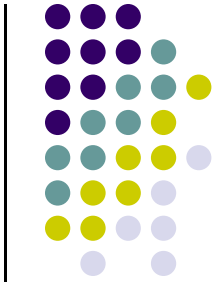
OpenGL State Variables

- OpenGL maintains **state variables** for **drawing attributes**
  - Current **drawing color**
  - Current **point size**
  - Current **line thickness**
  - Current **background color**
- All drawings use **current** value of state variables
- State variables retain and use old value until changed
- Example:
  - If you set drawing color to **blue**
  - All drawing is in **blue** till drawing color changed

OpenGL Command Format

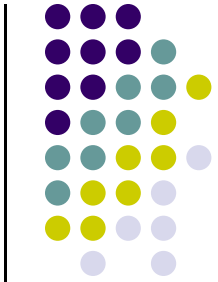






Some OpenGL Commands

- **`glPointSize( )`** – sets point size used in drawing
- **`glColor3f(R,G,B,alpha)`** – set RGB color
  - Sets RGB color and transparency (alpha).
  - RGB color in range 0 to 1.0
  - Alpha: 0.0 = fully opaque, 1.0 = fully transparent
- **`glClearColor(R,G,B,alpha)`**
- **`glClear(GL_COLOR_BUFFER_BIT )`**
  - Clears screen to background color
- **`glFlush( )`** – forces image drawing

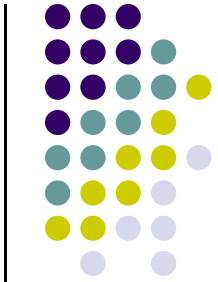


Double Buffering

- Set display mode to double buffers (front and back framebuffers)
  - **`glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);`**
    - Double buffering with RGB colors
- Front buffer visible on screen
- Drawing done in back buffers (not visible to user) until swapped in using **`glutSwapBuffers( )`**

```
void mydisplay(void){
      glClear(GL_COLOR_BUFFER_BIT); // clear screen
      glDrawArrays(GL_POINTS, 0, N);
      glutSwapBuffers( )
}
```

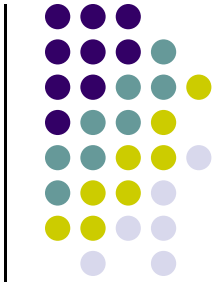
Back buffer drawing swapped in, becomes visible here



OpenGL Data Types

C++	OpenGL
Signed char	GLByte
Short	GLShort
Int	GLInt
Float	GLFloat
Double	GLDouble
Unsigned char	GLubyte
Unsigned short	GLushort
Unsigned int	GLuint

Example: Integer is 32-bits on 32-bit machine
but 64-bits on a 64-bit machine



References

- Angel and Shreiner, Chapter 2
- Hill, chapter 2