CS 543 - Computer Graphics: OpenGL, Continued

by

Robert W. Lindeman

gogo@wpi.edu

(with help from Emmanuel Agu ;-)

Last time....

- OpenGL set up
- Basic structure
- OpenGL skeleton
- Callback functions, *etc*.

Last Time: OpenGL Skeleton

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

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

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

This Time: Do Some Real Stuff

- How to actually draw something
- OpenGL commands & data types
- Interaction with OpenGL program
- Interaction examples using keyboard and mouse
- Notes:
  - State-based system
  - Set the state(s)
  - Send vertices

Example: Rendering Callback

- Do all your drawing in the display function
  - Called initially & when picture changes (*e.g.*, resize)
- Recall, first register callback in main( ) function

```
glutDisplayFunc( myDisplay );
```

- Then, implement display function

```
void myDisplay( void )  {
  // put drawing stuff here
  ...
  glBegin( GL_LINES );
    glVertex3fv( v[0] );
    glVertex3fv( v[1] );
    ...
  glEnd( );
}
```

Basic Drawing Primitives

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

```
glBegin( primType );
   // define your primitives here
glEnd( );
```

- **primType**
  - One of: `GL_POINTS`, `GL_LINES`, `GL_POLYGON`, *etc.*

Basic Drawing Primitives: Example

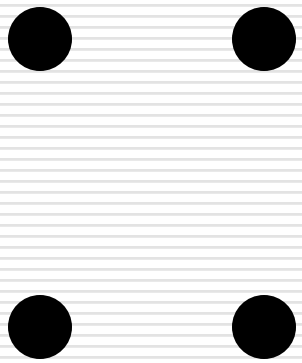
- Example: Draw four dots
- Basic idea
  - send sequence of vertices
  - Connect them in manner determined by **primType**

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

Parameters for **`glBegin( )`**

- Draw dots

```
glBegin( GL_POINTS );
```



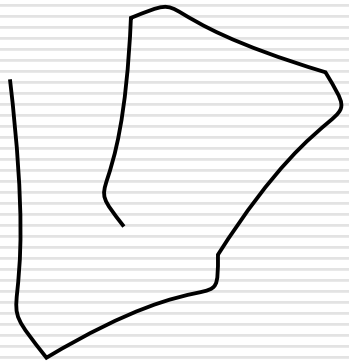
- Draw lines

```
glBegin( GL_LINES );
```

Parameters for `glBegin( )`

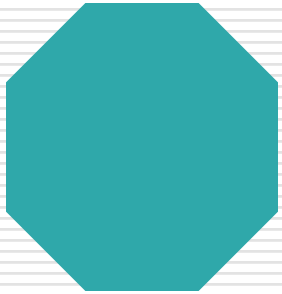
- Draw polylines

```
glBegin( GL_LINE_STRIP );
```



- Draw convex polygons

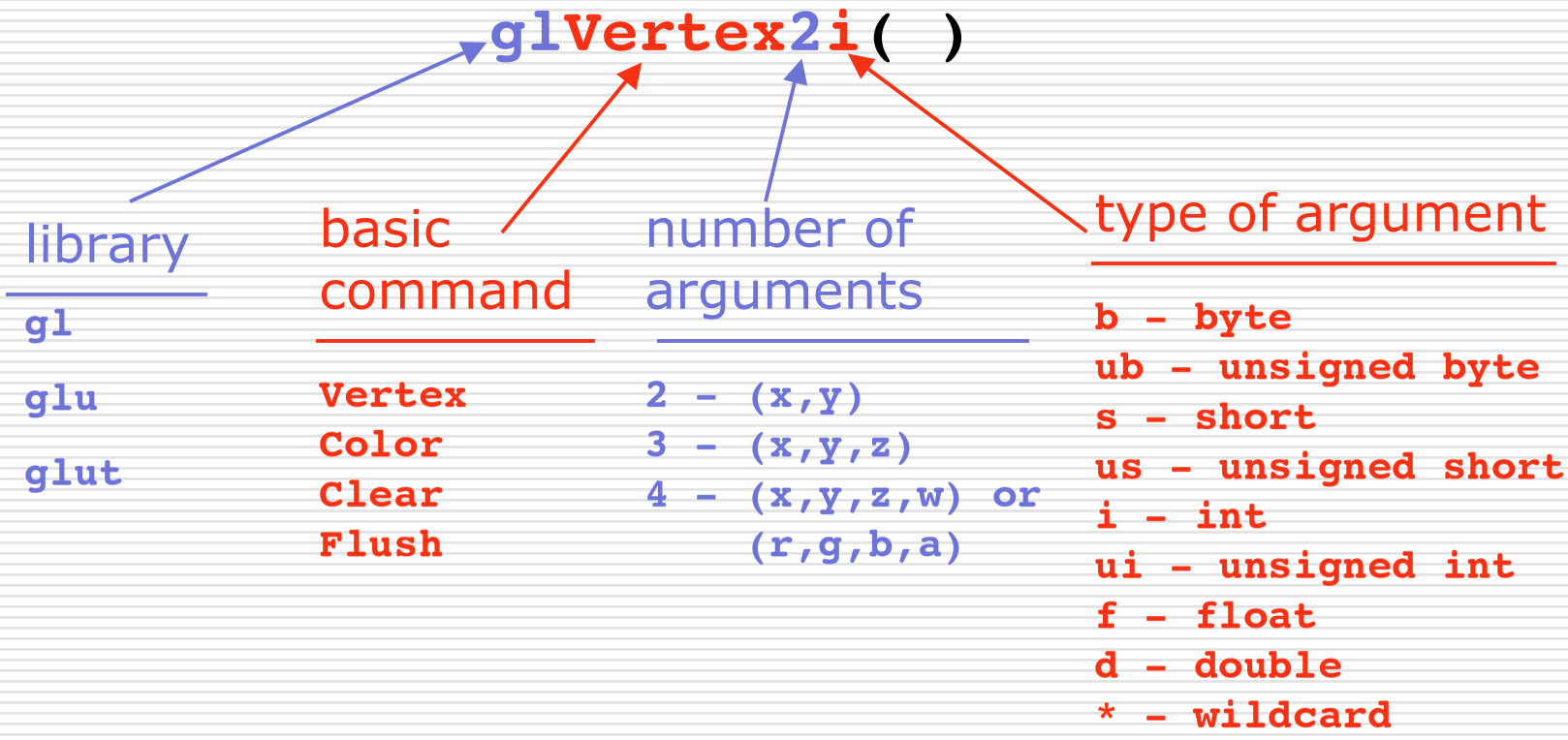
```
glBegin( GL_POLYGON );
```



Parameters for **glBegin( )**

```
GL_POINTS // dots
GL_LINES // lines, in pairs
GL_LINE_STRIP // polylines
GL_LINE_LOOP // closed loop
GL_TRIANGLES // triangles, in threes
GL_QUADS // quad, in fours vertices
GL_POLYGON // convex filled polygon
```

OpenGL Command Format



Some OpenGL Commands

```
glVertex2i( ); // x,y vertex position
glColor3f( );  // RGB color
glRecti( );    // aligned rectangled
glClearColor( ); // clear color in RGB
glClear( );    // clears screen
glFlush( );    // forces image drawing
```

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`

Keyboard Interaction

- Declare prototype

```
myKeyboard( unsigned int key, int x, int y );
```

- Register callback

```
glutKeyboardFunc( myKeyboard );
```

- Key values
  - ASCII value of key pressed
- X, Y values
  - Coordinates of mouse location
- The function **`myKeyboard( )`** contains a large switch statement to act depending on which key was pressed

Example: Keyboard Callback

- How to use keyboard to control program

  1. Register callback in main( ) function

  `glutKeyboardFunc( myKeyboard );`

  2. Implement keyboard function

```
void myKeyboard( unsigned int key, int x, int y )  {
  // put keyboard stuff here
  ...
  switch( key )  {   // check which key
    case 'f':
      // do stuff
      break;
  }
  ...
}
```

Mouse Interaction

- Declare prototype

```
myMouse( int button, int state, int x, int y )
myMovedMouse( ... );
```

- Register callbacks

```
glutMouseFunc( myMouse ) // when mouse button pressed
glutMotionFunc( myMovedMouse ) // when mouse moves
```

- **button** parameter value will be one of

```
GLUT_LEFT_BUTTON, GLUT_MIDDLE_BUTTON,
  GLUT_RIGHT_BUTTON
```

- **state** parameter value will be one of

```
GLUT_UP, GLUT_DOWN
```

- **x, y** parameters will have the mouse coordinates

Mouse Interaction Example

- Each mouse click generates separate events
- So, need to do your own checking, *etc*. in your mouse function
- Example: draw (or select) rectangle on screen

Mouse Interaction Example (cont.)


```
void myMouse( int button, int state, int x, int y )  {
  static GLintPoint corner[2];
  static int numCorners = 0;  // initial value is 0
  if( button == GLUT_LEFT_BUTTON && state == GLUT_DOWN ) {
    corner[numCorners].x = x;
    corner[numCorners].y = screenHeight – y; //flip y coord
    numCorners++;
    if( numCorners == 2 )  {
      // draw rectangle or do whatever you planned to do
      glRecti( corner[0].x, corner[0].y, corner[1].x, corner[1].y );
      numCorners = 0;
    }
  }
  else  {
    if( button == GLUT_RIGHT_BUTTON && state == GLUT_DOWN )  {
      glClear( GL_COLOR_BUFFER_BIT ); // clear the window
    }
  }
  glFlush( );
}
```

OpenGL State

- OpenGL tracks states
  - Drawing color
  - Point size
- Rendered object's appearance based on current state
- State variable remains active until changed

References

- Hill, chapter 2