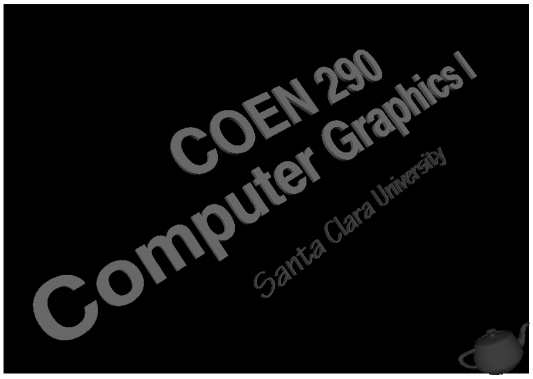
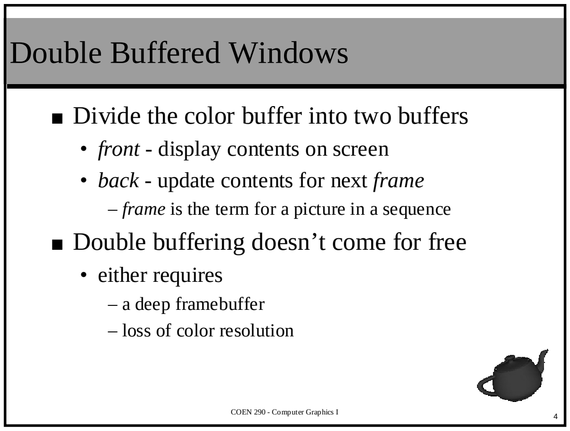


Evening’s Goals

- Discuss animation
- Introduce hidden surface removal (HSR) methods
- Analyze user interaction
  - discuss what GLUT provides

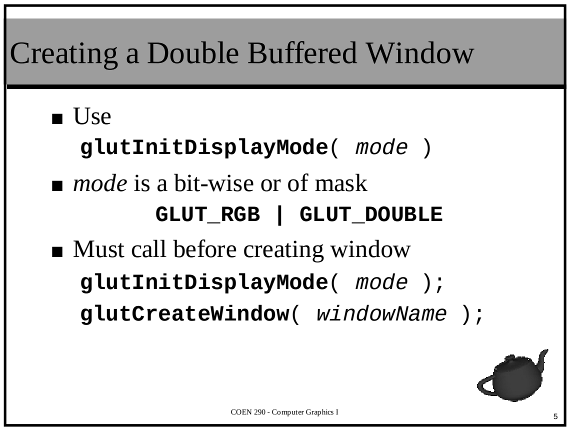
Animation

- Interactive graphics needs things to move
- Very simple
  - *double buffered* window
  - *motion variable* to keep track of movement
  - way to swap buffers



Double Buffered Windows

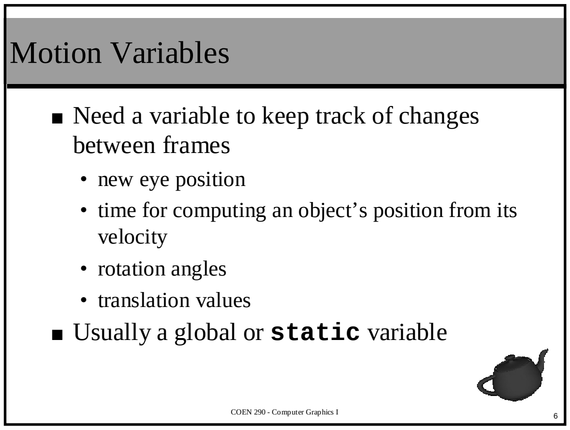
- Divide the color buffer into two buffers
  - *front* - display contents on screen
  - *back* - update contents for next *frame*
    - *frame* is the term for a picture in a sequence
- Double buffering doesn’t come for free
  - either requires
    - a deep framebuffer
    - loss of color resolution



Creating a Double Buffered Window

- Use
  glutInitDisplayMode(*mode*)
- *mode* is a bit-wise or of mask
  GLUT_RGB | GLUT_DOUBLE
- Must call before creating window

```
glutInitDisplayMode( mode );
glutCreateWindow( windowName );
```



Motion Variables

- Need a variable to keep track of changes between frames
  - new eye position
  - time for computing an object’s position from its velocity
  - rotation angles
  - translation values
- Usually a global or **`static`** variable

One Last Thing

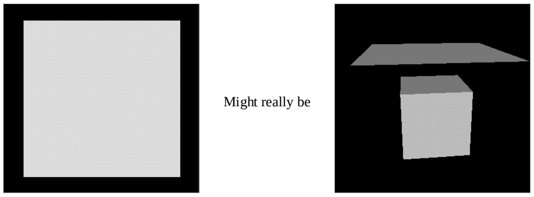
- Must swap *front* and *back* buffers

```
glutSwapBuffers();
```

- Last call in your rendering routine

The Problem ...

- Things are not always what they seem



Hidden Surface Removal Algorithms

- Determine which primitives are “in front of” others
  - usually depends on the eye’s position
- Methods
  - painter’s algorithm
  - backface culling
  - depth buffering

Painter's Algorithm

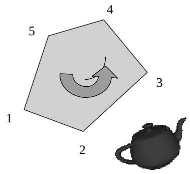
- Technique inspired by how painters paint
  - layer foreground objects over top of background objects
- How do we do this?
  - sort graphics primitives in eye space based on their distance from the eye
  - use z' after transforming into eye space

Painter's Algorithm (cont.)

- Very limited algorithm
  - no intersecting primitives
    - must tessellate all intersecting primitives
  - sorting is slow
  - must re-sort after eye moves
- Nothing in OpenGL to help

Backface Culling

- Eliminate polygons based on *vertex winding*
- If all polygons are ordered consistently, remove all ordered the same
  - by convention, choose *clockwise* ordered polygons to be *back facing*
- Facedness determined in window space by the polygon's area



Determining a Polygon’s Area

- In window coordinates, compute

$$area = \tfrac{1}{2}\sum_{i=0}^{n-1} x_i y_{i\oplus 1} - x_{i\oplus 1} y_i$$

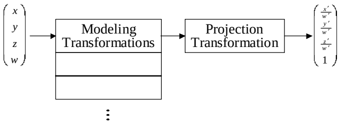
where $i \oplus 1 = (i+1) \bmod n$

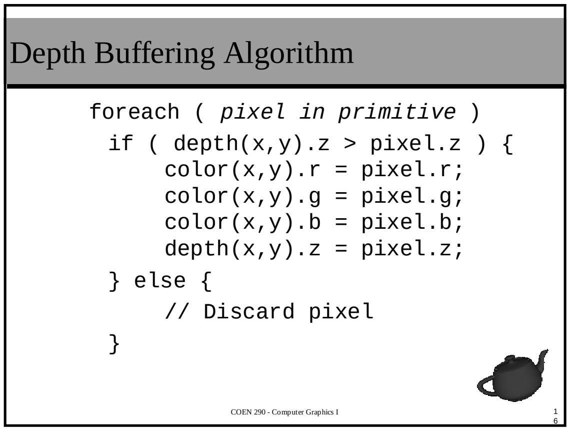
The Down Side …

- Backface culling is only of limited use
  - works best with *closed convex shapes*
    - – spheres, cylinders, cubes, etc.
- Really used for increasing rendering performance
  - reduces the number of pixels you need to fill for a frame

Depth Buffering

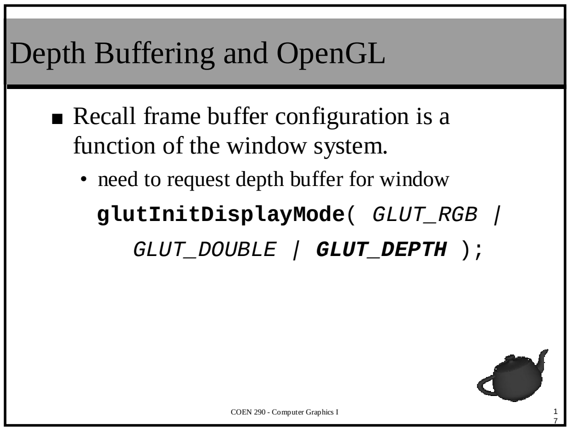
- Test every pixel to see whose in front
- Requires an additional buffer the size of the window to store *depth* values
  - *depth* is really distance from eye


$\begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}$
Modeling Transformations
⋮
Projection Transformation
$\begin{pmatrix} \frac{x'}{w'} \\ \frac{y'}{w'} \\ \frac{z'}{w'} \\ 1 \end{pmatrix}$



Depth Buffering Algorithm

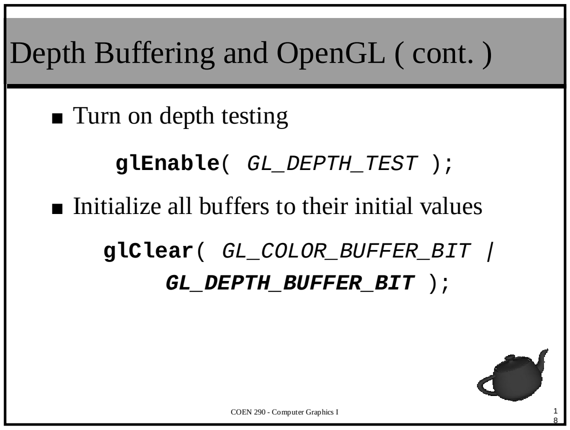
```
foreach ( pixel in primitive )
  if ( depth(x,y).z > pixel.z ) {
        color(x,y).r = pixel.r;
        color(x,y).g = pixel.g;
        color(x,y).b = pixel.b;
        depth(x,y).z = pixel.z;
  } else {
        // Discard pixel
  }
```



Depth Buffering and OpenGL

- Recall frame buffer configuration is a function of the window system.
  - need to request depth buffer for window

```
glutInitDisplayMode( GLUT_RGB |
    GLUT_DOUBLE | GLUT_DEPTH );
```



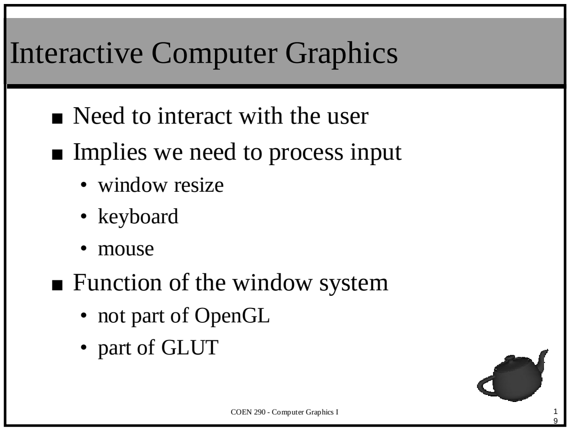
Depth Buffering and OpenGL (cont.)

- Turn on depth testing

```
glEnable( GL_DEPTH_TEST );
```

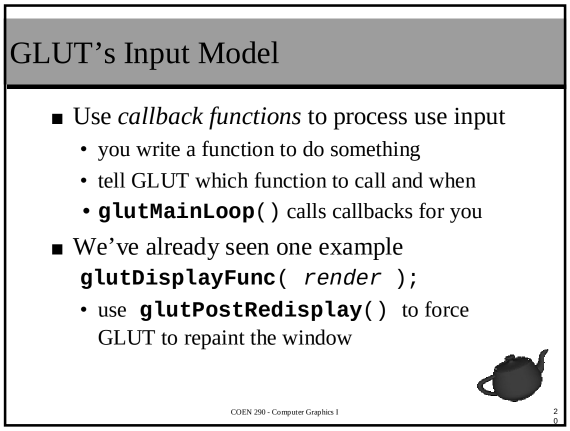
- Initialize all buffers to their initial values

```
glClear( GL_COLOR_BUFFER_BIT |
      GL_DEPTH_BUFFER_BIT );
```



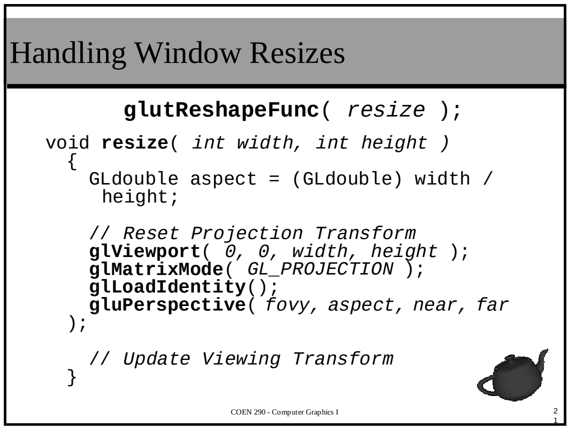
Interactive Computer Graphics

- Need to interact with the user
- Implies we need to process input
  - window resize
  - keyboard
  - mouse
- Function of the window system
  - not part of OpenGL
  - part of GLUT



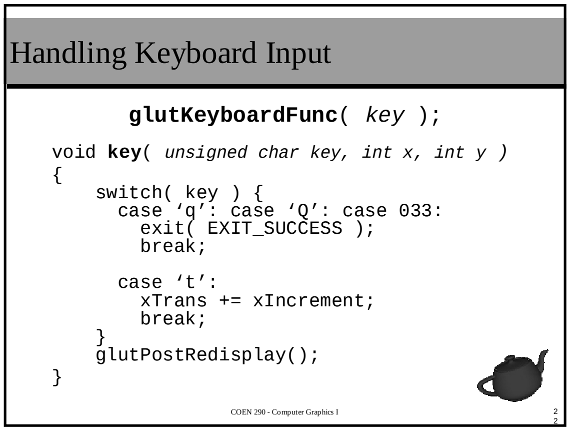
GLUT's Input Model

- Use *callback functions* to process use input
  - you write a function to do something
  - tell GLUT which function to call and when
  - **glutMainLoop**() calls callbacks for you
- We've already seen one example
  glutDisplayFunc(*render*);
  - use **glutPostRedisplay**() to force GLUT to repaint the window



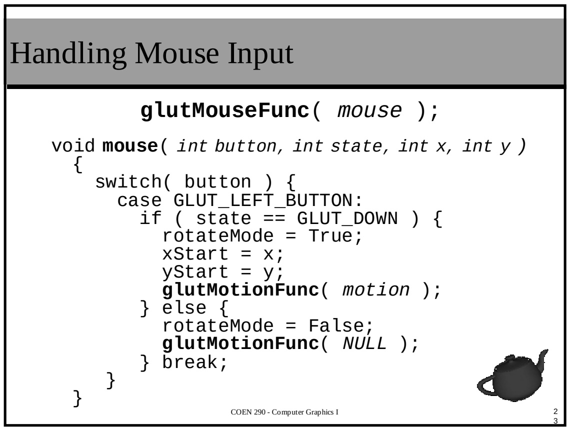
Handling Window Resizes

```
       glutReshapeFunc( resize );
void resize( int width, int height )
  {
    GLdouble aspect = (GLdouble) width /
     height;

    // Reset Projection Transform
    glViewport( 0, 0, width, height );
    glMatrixMode( GL_PROJECTION );
    glLoadIdentity();
    gluPerspective( fovy, aspect, near, far
  );

    // Update Viewing Transform
  }
```



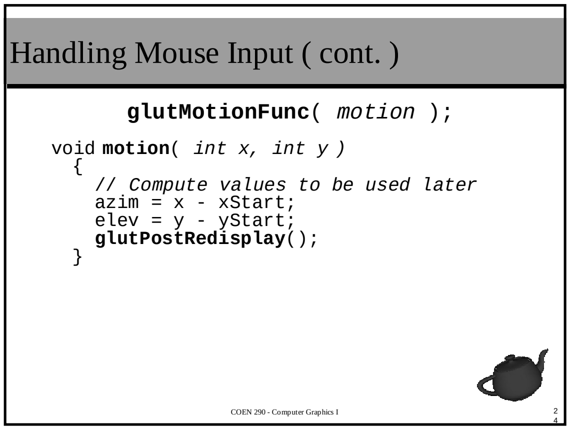
Handling Keyboard Input

```
        glutKeyboardFunc( key );

void key( unsigned char key, int x, int y )
{
    switch( key ) {
      case ‘q’: case ‘Q’: case 033:
        exit( EXIT_SUCCESS );
        break;

      case ‘t’:
        xTrans += xIncrement;
        break;
    }
    glutPostRedisplay();
}
```



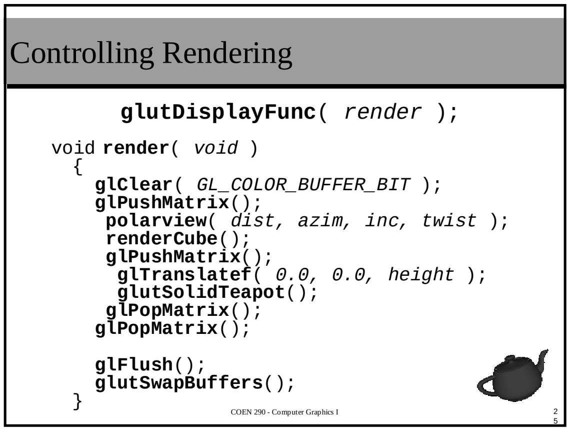
Handling Mouse Input

```
        glutMouseFunc( mouse );

void mouse( int button, int state, int x, int y )
  {
    switch( button ) {
      case GLUT_LEFT_BUTTON:
        if ( state == GLUT_DOWN ) {
          rotateMode = True;
          xStart = x;
          yStart = y;
          glutMotionFunc( motion );
        } else {
          rotateMode = False;
          glutMotionFunc( NULL );
        } break;
     }
  }
```



Handling Mouse Input (cont.)

```
        glutMotionFunc( motion );

void motion( int x, int y )
  {
    // Compute values to be used later
    azim = x - xStart;
    elev = y - yStart;
    glutPostRedisplay();
  }
```



Controlling Rendering

```
      glutDisplayFunc( render );

void render( void )
  {
    glClear( GL_COLOR_BUFFER_BIT );
    glPushMatrix();
     polarview( dist, azim, inc, twist );
     renderCube();
     glPushMatrix();
      glTranslatef( 0.0, 0.0, height );
      glutSolidTeapot();
     glPopMatrix();
    glPopMatrix();

    glFlush();
    glutSwapBuffers();
  }
```