OpenGL® Lectures OpenGL Introduction

By
Tom Duff
Pixar Animation Studios
Emeryville, California
and
George Ledin Jr
Sonoma State University
Rohnert Park, California

What is OpenGL®

- OpenGL (Open Graphics Library) was developed originally by SGI (Silicon Graphics Incorporated)
- Utilities in OpenGL library can be called from C, C++
- Bindings available also for other programming languages such as Java, Tcl, Python, Ada, and Fortran
- Built on graphics hardware and works fast
- Portable to most other systems and able to use other architectures' graphics capabilities

What does OpenGL do and not do

- Generates objects made of points, lines, and polygons
- Controls over lighting, object surface properties, transparency, anti-aliasing and texture mapping
- Window management is not supported in the basic version of OpenGL
- However there are additional libraries built on top of OpengGL to do what OpenGL does not support

OpenGL Family

- GL
  - The basic Graphics Library. Simple essential commands, e.g. glEnable();
  - To use it, you must #include <GL/gl.h>

- GLU
  - Graphics Library Utilities.
  - More complex commands, e.g. drawing a cylinder, routines for setting up viewing and projection matrices, polygonal tesselation and surface rendering.
  - To use it, you must #include <GL/glu.h>

- GLUT
  - Graphics Library Utilities Toolkit.
  - More sophisticated windowing features, e.g., sphere
  - To use it, you must #include <GL/glut.h>

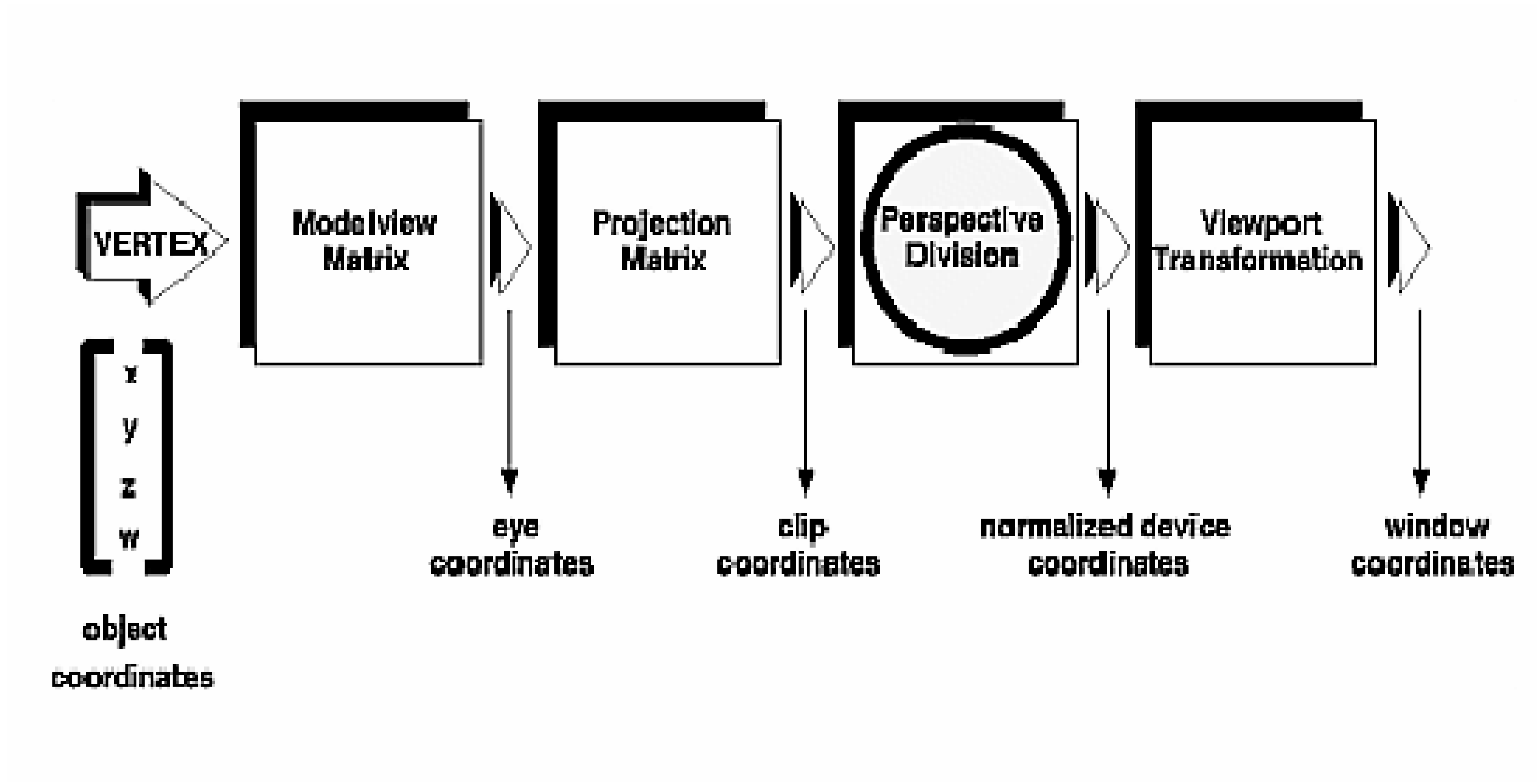
- GLX
  - Graphics Library for X-windows
  - Commands for drawing GL shapes in X

- If you use OpenGL in Windows, you should #include <windows.h>

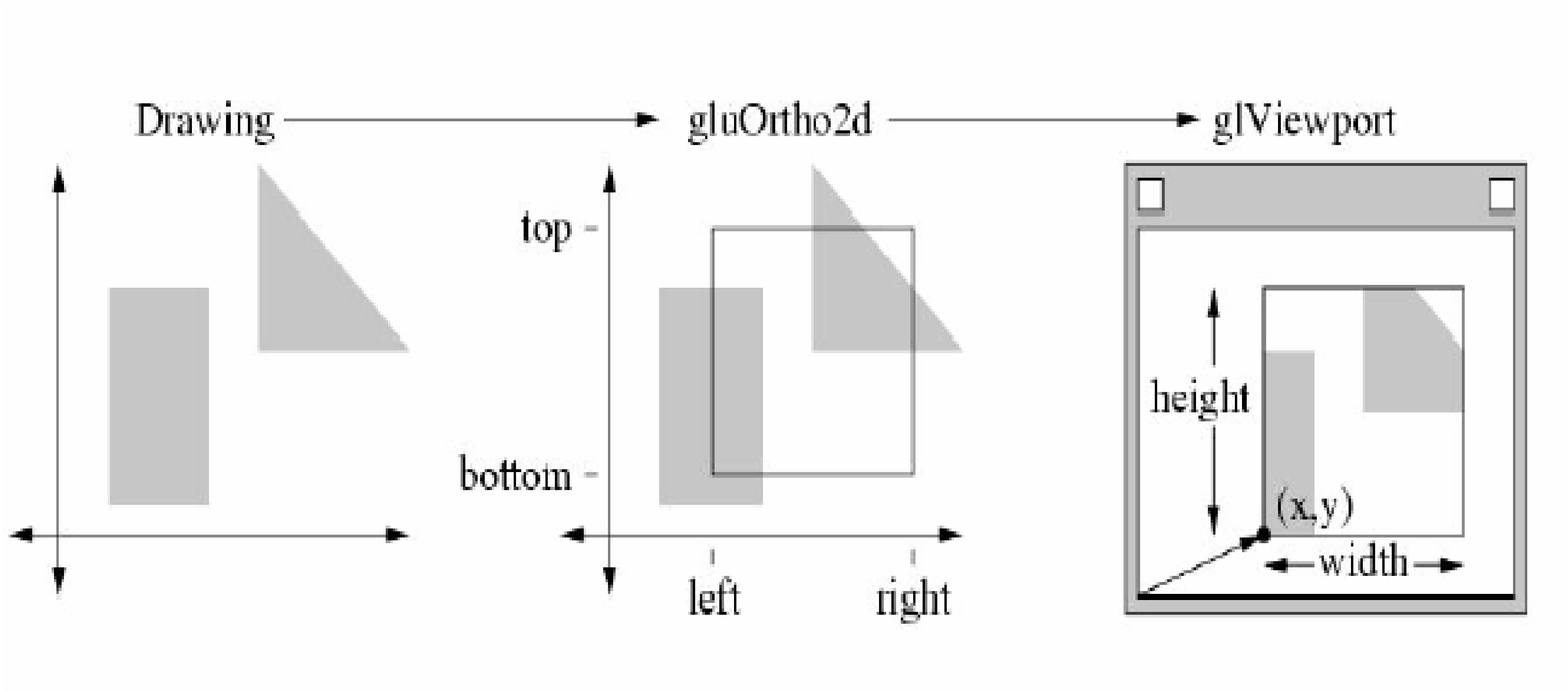
How to use OpenGL

- The functions in gl library have names that begin with gl.
  - eg: glVertex3f() (drawing a vertex)
- The functions in glu library have names that begin with glu.
  - eg: gluOrtho2D() (setting up orthogonal view)
- The functions in glut library have names that begin with glut.
  - eg: glutReshapeFunc() (invoked when user changes the window size)

OpenGL Graphics Pipeline



Projection and Viewport



Orthogonal Projection

- Orthographic projection is used for 2D drawing

  Perspective projection is often used for 3D drawing

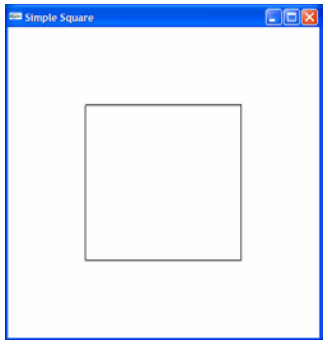
- 2D Viewing: Orthographic View
  - gluOrtho2D(left, right, bottom, top)
    - Specifies the coordinates of 2D region to be projected into the viewport.

Viewport

- Viewport
  - The sub-window into which the current graphics are being drawn.
- glViewport(x, y, width, height)
  - OpenGL assumes that you are mapping your graphics to the entire screen window by default.
  - (x, y) is the lower-left corner of the viewport.

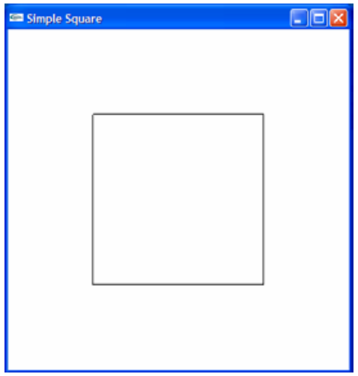
How to draw a simple square in OpenGL

```
/*  SimpleSquare.c
 *  This program draws a white square outline on a black background.
 */
#include <windows.h>
#include <GL/glut.h>         /* glut.h includes gl.h and glu.h*/

void display(void)
{
/*  set the clear value for the buffer to white for white background */
     glColor3f(0.0,0.0,0.0);
/* Clear color buffer with the current clear value.
   If not, you will see background behind your screen window */
     glClear(GL_COLOR_BUFFER_BIT);

/* draw white polygon (rectangle) outline centered at (0,0) */
     glPolygonMode(GL_FRONT, GL_LINE);

/* draw unit square polygon centered at (0,0) */
     glBegin(GL_POLYGON);

/* set the points counter clockwise for the front of the polygon
       If not, you will not see the outline, you will see solid color instead */

      glVertex2f (-0.5, -0.5);      /* left bottom corner */
      glVertex2f (0.5, -0.5);       /* right bottom corner */
      glVertex2f (0.5, 0.5);        /* right top corner */
      glVertex2f (-0.5, 0.5);       /* left top corner */

     glEnd();

/* force execution of GL commands */
     glFlush();
}
```



How to draw a simple square in OpenGL, Continued

```
void main(int argc, char** argv)
{
// glutInit glutInit will initialize the GLUT library and
// negotiate a session with the windows system. Also, it
// extracts any command line options understood by the GLUT library.
     glutInit(&argc,argv);

/* Initialize mode and open a window in upper left corner of your screen
     */

// Specify the window size 400 * 400 pixels
     glutInitWindowSize(400,400);
     glutCreateWindow("Simple Square");

/* Set background color as white by clearing color buffer with white
     color */
     glClearColor (1.0, 1.0, 1.0, 0.0);
     glutDisplayFunc(display);

/* glutMainLoop enters the GLUT event processing loop.
   This routine should be called at most once in a GLUT program.
   Once called, this routine will never return.
   It will call as necessary any callbacks that have been registered. */
     glutMainLoop();
}
```

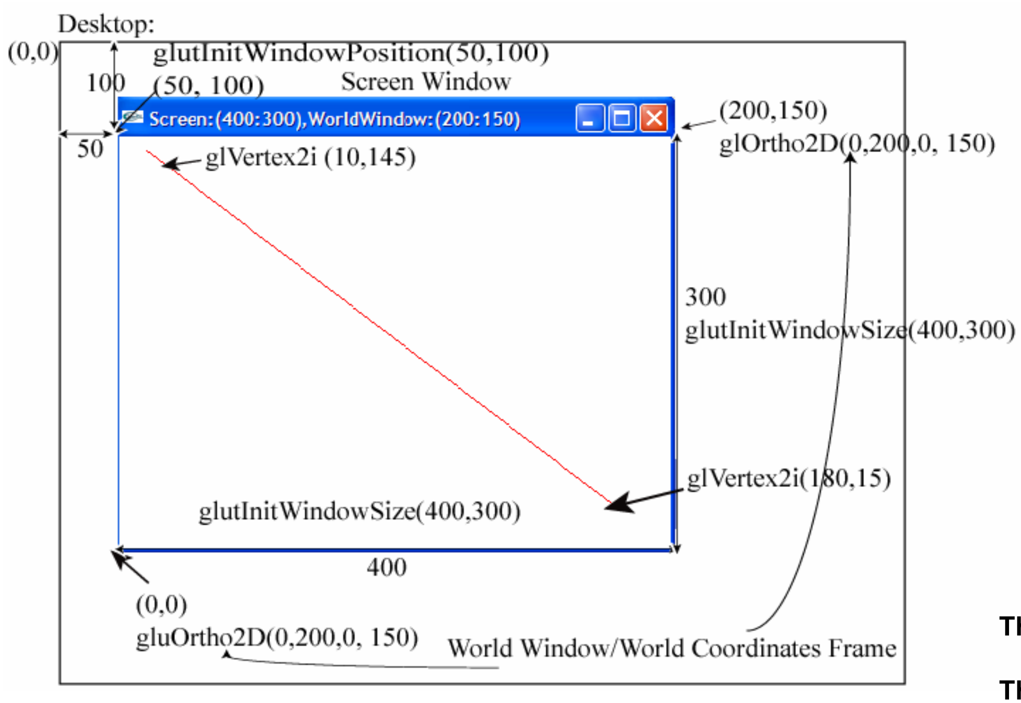


World Coordinates → World Window → Screen Window

- World Coordinates
  - The real world coordinates.
- World Window/World Coordinate Frame
  - The window/frame which models the world coordinates
- Screen Window
  - The screen which displays objects mapped from the World Coordinates Frame

Understanding Screen Window and World Window (Orthogonal Projection) - Part 1

Question: How to draw a line on the Screen Window using World Coordinates?



Desktop:
(0,0)
glutInitWindowPosition(50,100)
100
(50, 100)
Screen Window
Screen:(400:300),WorldWindow:(200:150)
50
(200,150)
glOrtho2D(0,200,0, 150)
glVertex2i (10,145)
300
glutInitWindowSize(400,300)
glVertex2i(180,15)
glutInitWindowSize(400,300)
400
(0,0)
gluOrtho2D(0,200,0, 150)
World Window/World Coordinates Frame

The aspect ratio of Screen Window:
400:300 = 4:3
The aspect ratio of World Window:
200:150 = 4:3

Answer:
The following steps are needed:

(1) Set up World Window Coordinates. In this case the lower left corner is (0,0), and the upper right corner is (200, 150) .

(2) If you want your object to display proportionally on the Screen, the aspect ratio of the Screen window needs to match the aspect ratio of the World window.

Understanding Screen Window and World Window (Orthogonal Projection) - Part 2

Question: What will happen if you resize the window?

Window1:

Aspect ratio of Screen Window: 400:300=4:3

Aspect ratio of World Window (orthogonal projection): 200:150=4:3

Window2:

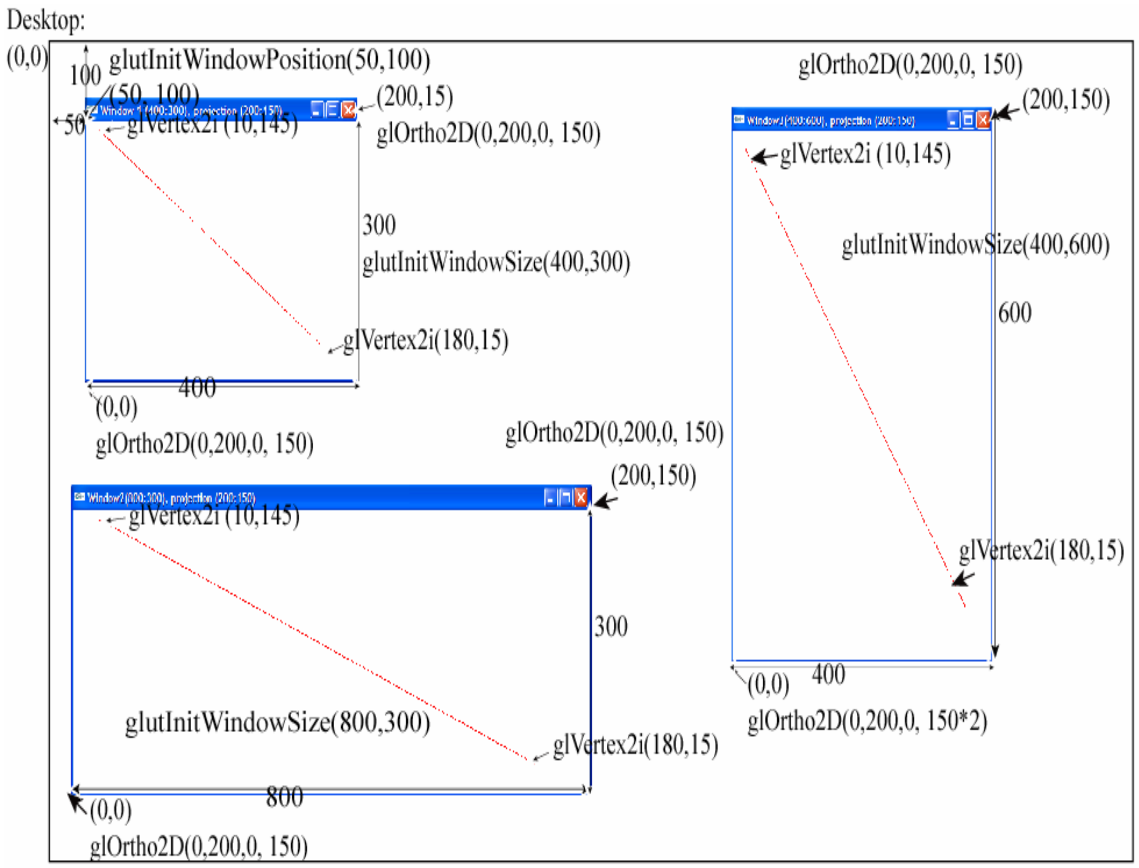
Aspect ratio of Screen Window: 800:300=8:3

Aspect ratio of World Window (orthogonal projection): 200:150=4:3

Window3:

Aspect ratio of Screen Window: 400:600=2:3

Aspect ratio of World Window (orthogonal projection): 200:150=4:3



Answer:

- **If you don't adjust the aspect ratio of the World Window coordinates, after you resize the Screen Window, the objects in the Screen Window will appear distorted and their original proportions will change.**

Understanding Screen Window and World Window (Orthogonal Projection) - Part 3

Question: How to keep the proportions of the object when the Screen Window is resized?

Window1:

Aspect ratio of Screen Window: 400:300=4:3

Aspect ratio of World Window (orthogonal projection): 200:150=4:3

Window2:

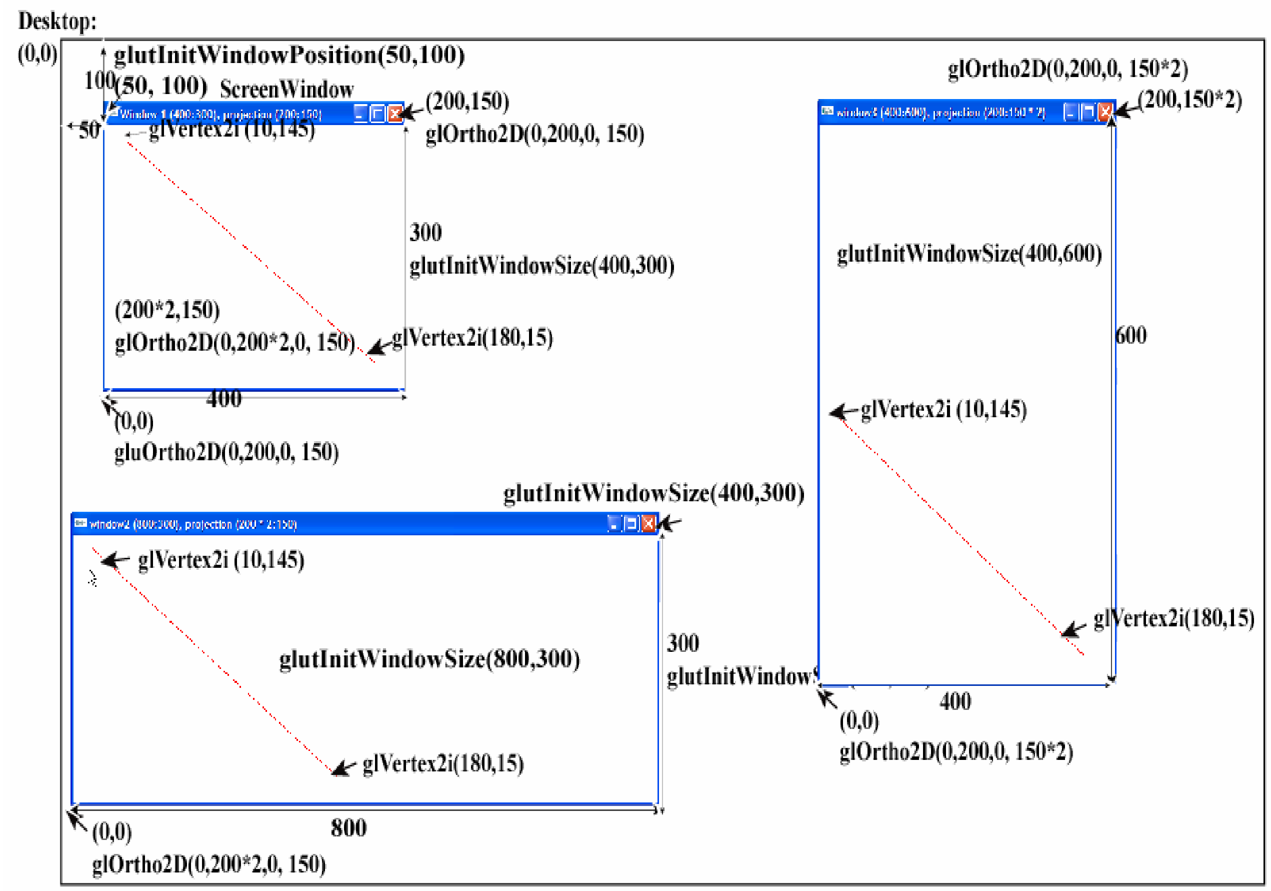
Aspect ratio of Screen Window: 800:300=8:3

Aspect ratio of World Window (orthogonal projection): 200*2:150=8:3

Window3:

Aspect ratio of Screen Window: 400:600=2:3

Aspect ratio of World Window (orthogonal projection): 200:150*2=2:3


Desktop:
(0,0)
glutInitWindowPosition(50,100)
100
(50, 100) ScreenWindow
50
Window 1 (400:300), projection (200:150)
glVertex2i (10,145)
(200,150)
glOrtho2D(0,200,0, 150)
300
glutInitWindowSize(400,300)
(200*2,150)
glOrtho2D(0,200*2,0, 150)
glVertex2i(180,15)
400
(0,0)
gluOrtho2D(0,200,0, 150)
glOrtho2D(0,200,0, 150*2)
(200,150*2)
window3 (400:600), projection (200:150*2)
glutInitWindowSize(400,600)
600
glVertex2i (10,145)
glVertex2i(180,15)
400
(0,0)
glOrtho2D(0,200,0, 150*2)
glutInitWindowSize(400,300)
window2 (800:300), projection (200*2:150)
glVertex2i (10,145)
glutInitWindowSize(800,300)
300
glutInitWindow
glVertex2i(180,15)
(0,0)
800
glOrtho2D(0,200*2,0, 150)

Answer:
If you would like to keep the distort the original proportion of your objects, you need to adjust your World Window coordinate's aspect ratio to be the same as the aspect ratio of the Screen Window.

Understanding Screen Window and World Window (Orthogonal Projection) - Part 4

Question: How to keep the object in the center of the Screen Window as it resizes?

Note:
The aspect ratio of Screen Window and the World Window is the same, therefore, the line proportion is not distorted.

Window1:

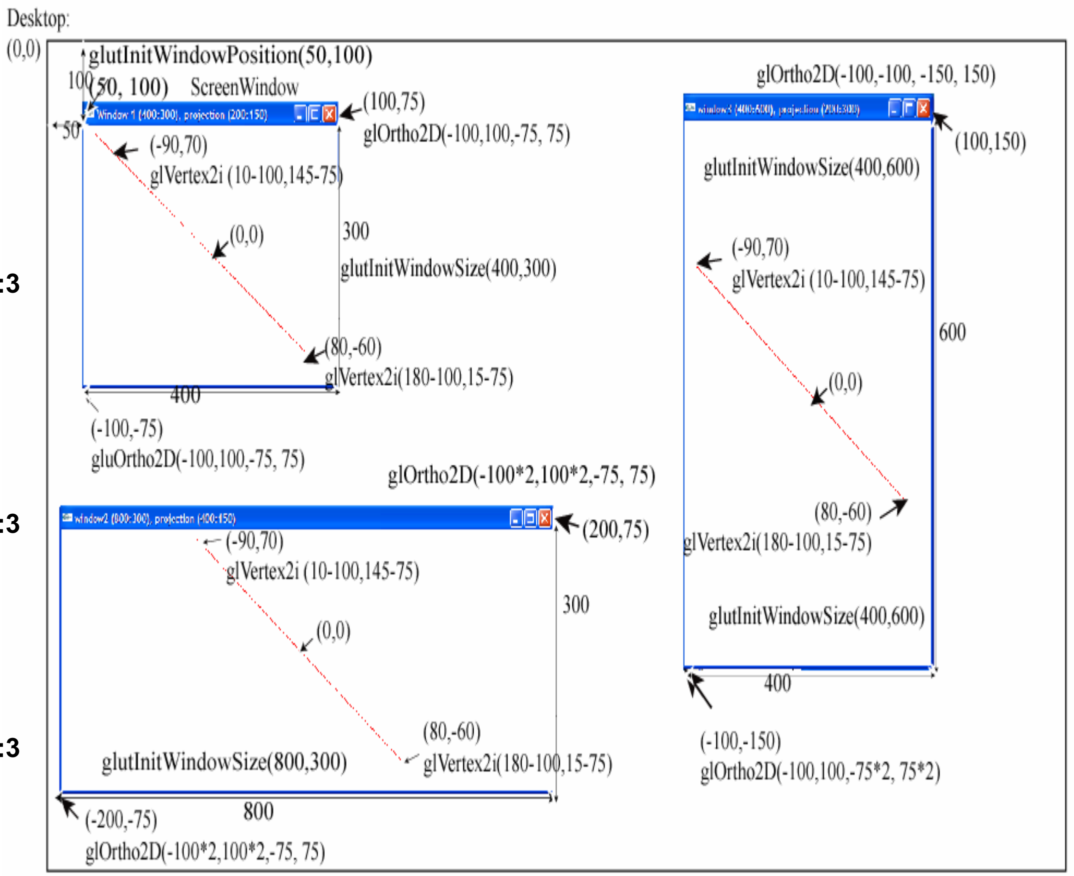
Aspect ratio of Screen Window: 400:300=4:3
Aspect ratio of World Window (orthogonal projection): (100-(-100)):(75-(-75))=4:3

Window2:

Aspect ratio of Screen Window: 800:300=8:3
Aspect ratio of World Window (orthogonal projection): (200-(-200)):(75-(-75))=8:3

Window 3:

Aspect ratio of Screen Window: 400:600=2:3
Aspect ratio of World Window (orthogonal projection): (100-(-100)):(150-(-150))=2:3

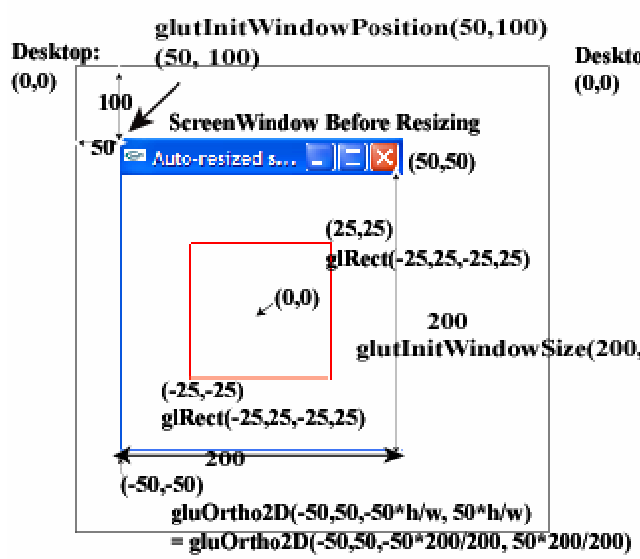

Desktop:
(0,0)
glutInitWindowPosition(50,100)
100
(50, 100)
ScreenWindow
50
(100,75)
glOrtho2D(-100,100,-75, 75)
(-90,70)
glVertex2i (10-100,145-75)
(0,0)
300
glutInitWindowSize(400,300)
(80,-60)
glVertex2i(180-100,15-75)
400
(-100,-75)
gluOrtho2D(-100,100,-75, 75)
glOrtho2D(-100,-100, -150, 150)
(100,150)
glutInitWindowSize(400,600)
(-90,70)
glVertex2i (10-100,145-75)
600
(0,0)
(80,-60)
glVertex2i(180-100,15-75)
glutInitWindowSize(400,600)
400
(-100,-150)
glOrtho2D(-100,100,-75*2, 75*2)
glOrtho2D(-100*2,100*2,-75, 75)
(200,75)
(-90,70)
glVertex2i (10-100,145-75)
300
(0,0)
(80,-60)
glutInitWindowSize(800,300)
glVertex2i(180-100,15-75)
(-200,-75)
800
glOrtho2D(-100*2,100*2,-75, 75)

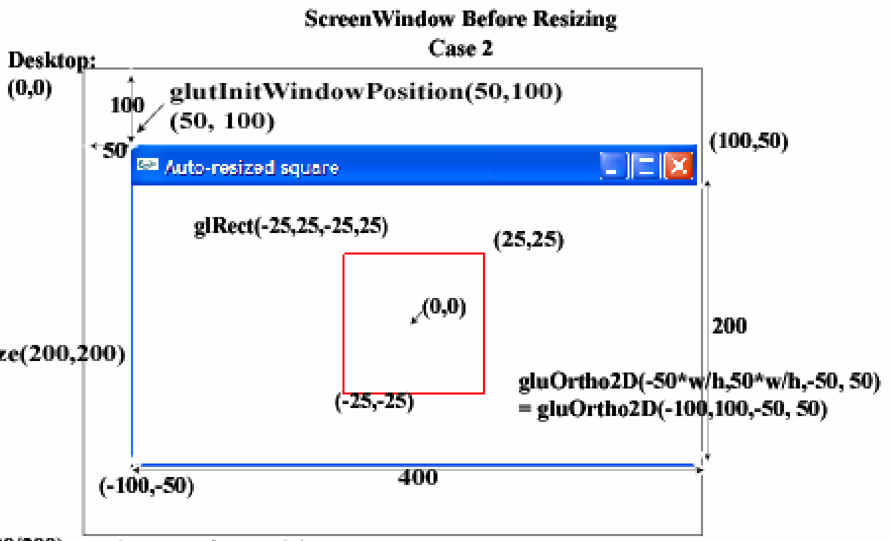
Answer:

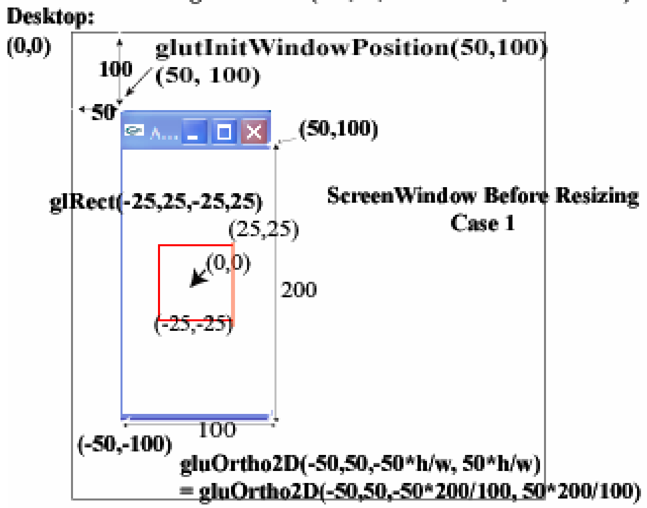
*** If you want your original object to stay in the center of the Screen Window, when you set up your World Window coordinates, make the center to be (0,0) (instead of the lower left corner which is the default 0,0)).**

Understanding Screen Window and World Window (Orthogonal Projection) - Part 5

Question: How to keep the object the same shape automatically after resizing the Screen Window?


Desktop:
(0,0)
glutInitWindowPosition(50,100)
(50, 100)
100
50
ScreenWindow Before Resizing
Auto-resized s...
(50,50)
(25,25)
glRect(-25,25,-25,25)
(0,0)
200
glutInitWindowSize(200,200)
(-25,-25)
glRect(-25,25,-25,25)
200
(-50,-50)
gluOrtho2D(-50,50,-50*h/w, 50*h/w)
= gluOrtho2D(-50,50,-50*200/200, 50*200/200)


ScreenWindow Before Resizing
Case 2
Desktop:
(0,0)
100
glutInitWindowPosition(50,100)
(50, 100)
50
Auto-resized square
(100,50)
glRect(-25,25,-25,25)
(25,25)
(0,0)
200
gluOrtho2D(-50*w/h,50*w/h,-50, 50)
= gluOrtho2D(-100,100,-50, 50)
(-25,-25)
(-100,-50)
400


Desktop:
(0,0)
glutInitWindowPosition(50,100)
100
(50, 100)
50
A...
(50,100)
glRect(-25,25,-25,25)
ScreenWindow Before Resizing
Case 1
(25,25)
(0,0)
200
(-25,-25)
100
(-50,-100)
gluOrtho2D(-50,50,-50*h/w, 50*h/w)
= gluOrtho2D(-50,50,-50*200/100, 50*200/100)

Window before resizing:

Aspect ratio of Screen Window: 100:200=1:1

Aspect ratio of World Window: (50-(-50)):(100-(-100))=1:2

Window after resizing (Case 1):

Aspect ratio of Screen Window: 100:200=1:1

Aspect ratio of World Window: (50-(-50)):(100-(-100))=1:2

Window after resizing (Case 2):

Aspect ratio of Screen Window: 400:200=2:1

Aspect ratio of World Window: (100-(-100)):(50-(-50))=2:1

Answer: (h is height and w is width)

* If the resized window h >= w, multiply the World Window height by a factor of h/w. (See Case 1)
* If the resized window w > h, multiply the World Window height by a factor of w/h. (See Case 2)
* Keep the viewport the same aspect ratio as that of the World Window. In this case, the viewport is set to Screen Window size which also matches the World Window aspect ratio.

Understanding Viewport
(polygon_viewport.c)

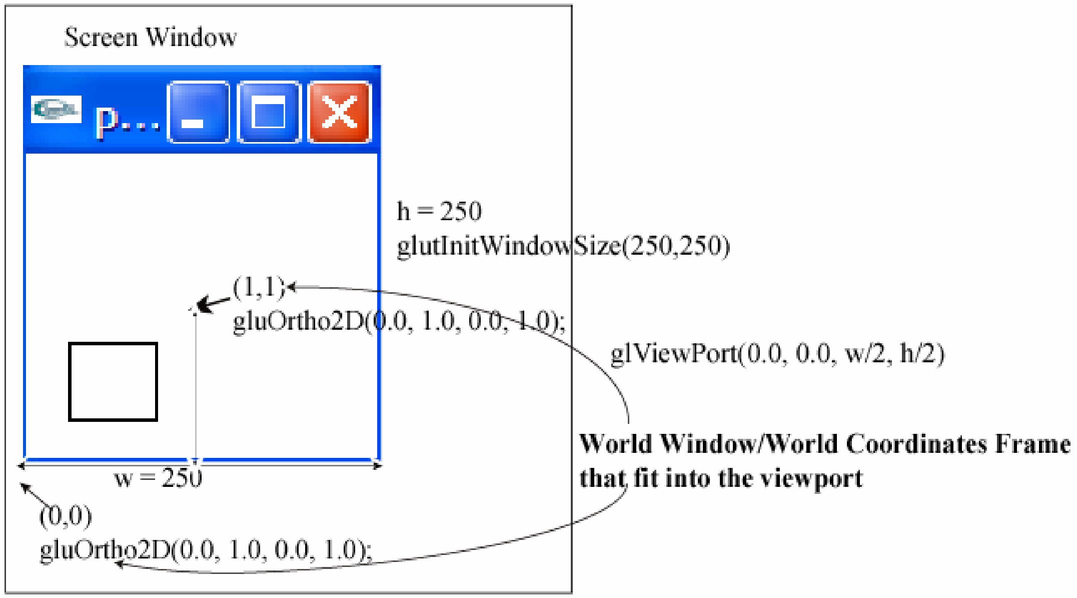
Question: What is a viewport?

Viewport C Function:
Void glViewport(Glint x, GLint y, GLint width, GLint height);

The viewport is the area of the window that OpenGL commands can draw into.

```
void display(void )
{
glClear  (GL_COLOR_BUFFER_BIT); // clear all pixels
   glPolygonMode(GL_FRONT   , GL_LINE);
/* draw white polygon (rectangle) with corners at
  (0.25, 0.25, 0.0) and (0.75, 0.75, 0.0)  */
   glColor3f (0.0, 0.0, 0.0);
   glBegin(GL_POLYGON );
      glVertex3f (0.25, 0.25, 0.0);
      glVertex3f (0.75, 0.25, 0.0);
      glVertex3f (0.75, 0.75, 0.0);
      glVertex3f (0.25, 0.75, 0.0);
   glEnd ();

/*  start execution of OpenGL functions  */
   glFlush  ();
}
```

Desktop:


Screen Window
p...
h = 250
glutInitWindowSize(250,250)
(1,1)
gluOrtho2D(0.0, 1.0, 0.0, 1.0);
glViewPort(0.0, 0.0, w/2, h/2)
World Window/World Coordinates Frame that fit into the viewport
w = 250
(0,0)
gluOrtho2D(0.0, 1.0, 0.0, 1.0);

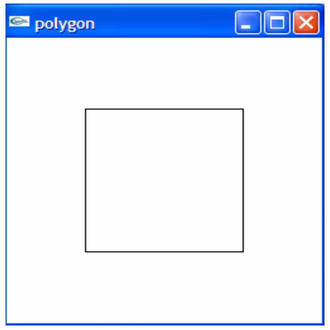
The aspect ratio of Screen Window: 250:250 = 1:1
The aspect ratio of World Window: 1:1= 1:1

Answer:
*** The viewport is the area of the window that the OpenGL commands can draw into.**

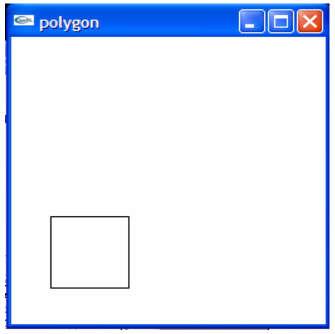
Example of glViewport

- Assume the output for the following viewport is (A).
  - w = width of the screen
  - h = height of the screen
  - **glViewport(0, 0, w, h);**
- Then the output for the following viewport is (B).
  - **glViewport(0, 0, w/2, h/2);**

(A)



(B)

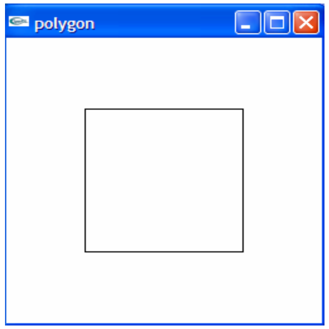


Viewport Code Segment A

- **Polygon_viewport.c**

```
void reshape (int w, int h)
{
  glClearColor (1.0, 1.0, 1.0, 0.0);  // set white
background
/* initialize viewport  */
   // The lower corner of the viewport is (0,0).
   // The height is h and width is w.
   glViewport(0, 0, w, h);

   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   glOrtho2D(0.0, 1.0, 0.0, 1.0);
   glMatrixMode(GL_MODELVIEW);
}

void display(void)
{    …
/* draw white polygon (rectangle) with corners at
  (0.25, 0.25, 0.0) and (0.75, 0.75, 0.0)  */
   glColor3f (0.0, 0.0, 0.0);
   glBegin(GL_POLYGON);
      glVertex3f (0.25, 0.25, 0.0);
      glVertex3f (0.75, 0.25, 0.0);
      glVertex3f (0.75, 0.75, 0.0);
      glVertex3f (0.25, 0.75, 0.0);
   glEnd();
…
}
```

- w = width of the screen
- h = height of the screen
- **glViewport(0, 0, w, h);**

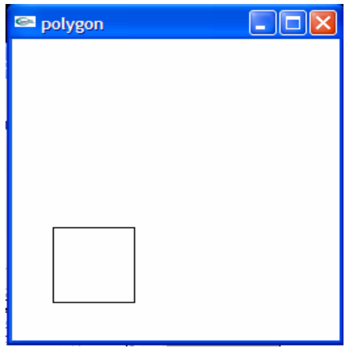


Viewport Code Segment B

- **Polygon_viewport.c**

```
void reshape (int w, int h)
{
  glClearColor (1.0, 1.0, 1.0, 0.0);  // set white
background
/* initialize viewport  */
 // The lower corner of the viewport is (0,0).
// The height is h/2 and width is w/2.
   glViewport(0, 0, w/2, h/2);

   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   glOrtho2D(0.0, 1.0, 0.0, 1.0);
   glMatrixMode(GL_MODELVIEW);
}

void display(void)
{    …
/* draw white polygon (rectangle) with corners at
  (0.25, 0.25, 0.0) and (0.75, 0.75, 0.0)  */
   glColor3f (0.0, 0.0, 0.0);
   glBegin(GL_POLYGON);
      glVertex3f (0.25, 0.25, 0.0);
      glVertex3f (0.75, 0.25, 0.0);
      glVertex3f (0.75, 0.75, 0.0);
      glVertex3f (0.25, 0.75, 0.0);
   glEnd();
…
}
```

- w = width of the screen
- h = height of the screen
- **glViewport(0, 0, w/2, h/2);**



Viewport Code Segment, Continued

- **Polygon_viewport.c**

```
void main(int argc, char** argv)
{
   …
/* A reshape callback is also triggered immediately before a window's
first display callback after a window. (In this case) The reshape
callback is triggered when a window is reshaped. */
   glutReshapeFunc(reshape);
   …
   glutMainLoop(); // process all events
}
```

Overview of Some Basic OpenGL Functions

- Basic data types
- Graphic primitives
- Glu Objects
- Glut Objects
- Color attributes

OpenGL data types

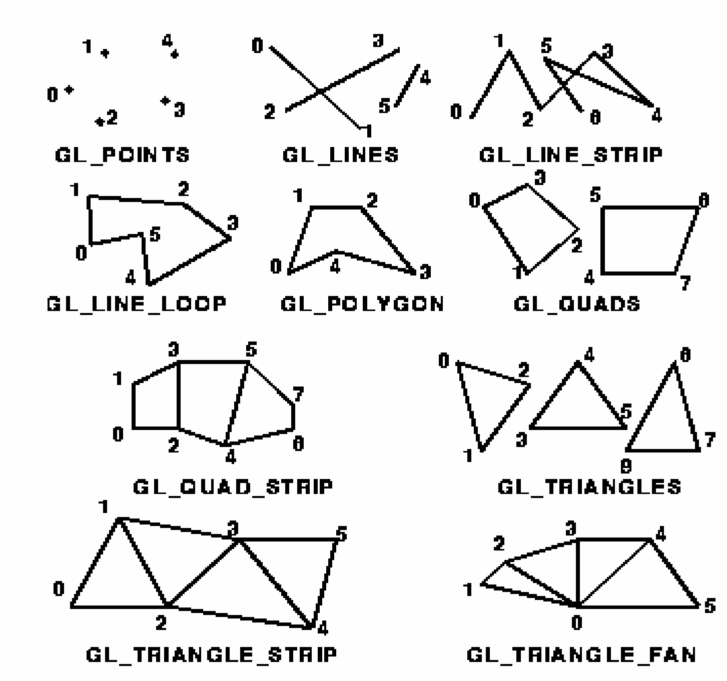
- F → GLfloat → #define GLfloat float
- I → Glint → #define GLint int
- D → GLdouble → #define GLdouble double
- S → GLshort → #define GLshort short
- B → GLbyte → #define GLbyte char

…

- (For more information see ‘gl.h’)

OpenGL Primitives

- In OpenGL, the programmer is provided the following primitives for use in constructing geometric objects.

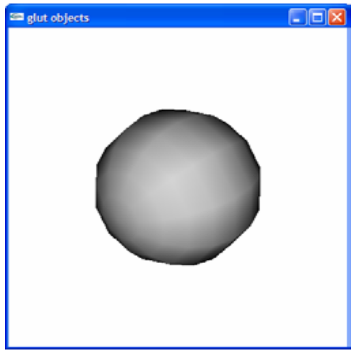


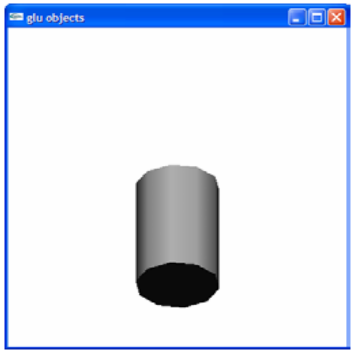
- Each geometric object is described by a set of vertices and the type of primitive to be drawn. Whether and how the vertices are connected is determined by the primitive type.

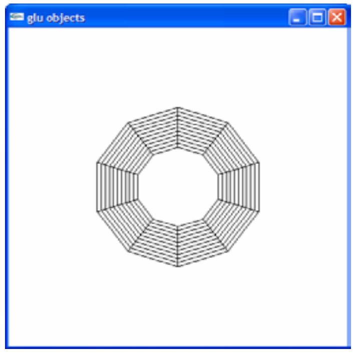
- We will look at these primitives in detail later.

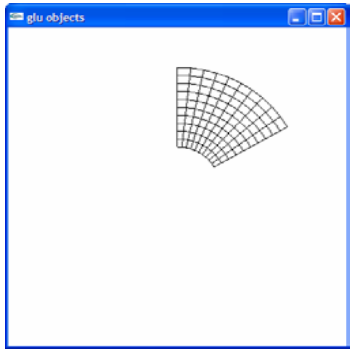
Glu Objects

- Glu has a few objects defined.
- These objects are sphere, cylinder, disk, partial disk.
- We will look at these objects in detail later.



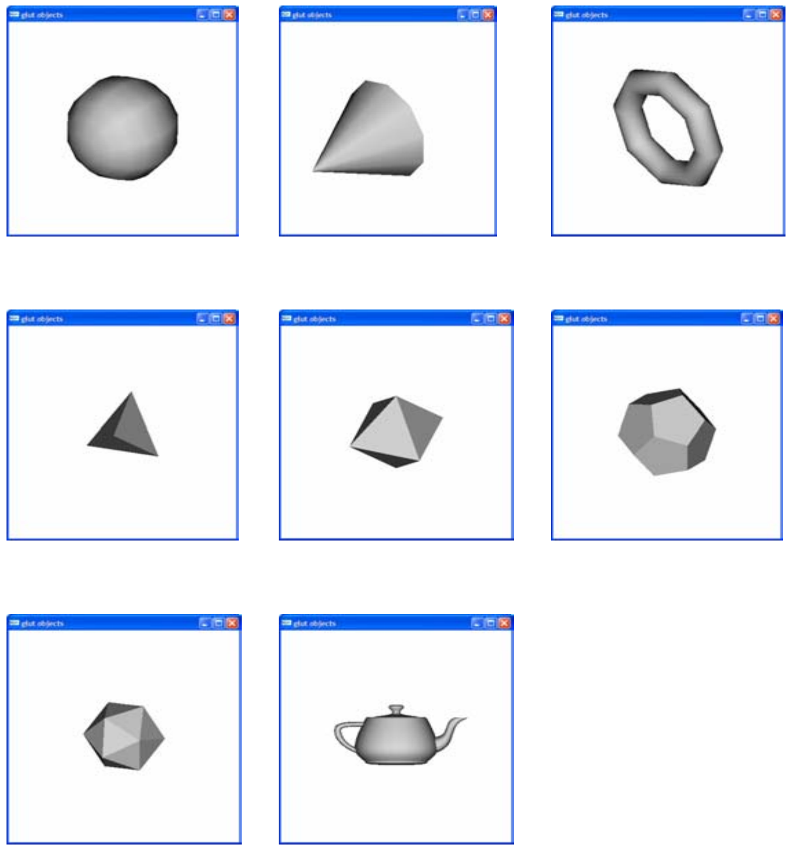






Glut Objects

- Glut has a few objects defined. You can use them with a single function call.
- These objects are sphere, cone, torus, tetrahedron, octahedron, docecahedron, isoahedron, and teapot.
- We will look at these objects in detail later.



OpenGL Color Attributes

- OpenGL supports two basic color modes
  - RGB (or RGBA)
    - Each color is a triplet of red, green, blue values
    - Our eyes add these primary colors to form the color that we see. The addictive mode is appropriate for monitors and projective systems.
    - In RGBA mode, the use fourth component is alpha (opacity).
    - E.g:
      - (1.0, 0.0, 0.0, 0.0) is bright red and transparent.
      - (1.0. 1.0, 1.0, 1.0) is white and opaque.
      - (0.0, 0.0, 0.0, 0.0) is black and transparent.
      - (0.0, 1.0, 0.0, 0.5) is green and translucent.

A note on color index mode

- Color index mode
  - Colors are specified as indices into a table of red, green, and blue values.
  - With inexpensive memory, this mode is not used often.
- We shall always use RGB/RGBA color.

Colors are part of OpenGL state

- In OpenGL, colors are part of the OpenGL state.
  - Colors are not attached to objects, but rather to the internal state of OpenGL.
  - The color used to render an object is the current color.

OpenGL Color Usages

(1)Setting Initial Display Mode for Colors

(2)Clear color buffer

(3)Set background color

(4)Set object color

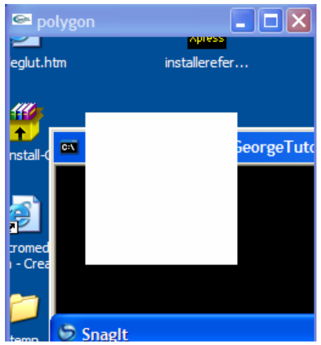
- OpenGL has many default settings, if you skip a step.

(1) Setting Initial Display Mode for Colors

- The initial display mode is used when creating windows.
- If you don’t set a specific mode, OpenGL uses its default
- void glutInitDisplayMode (unsigned int *mode*);
  - sets the initial display mode.
  - *mode* for color
    - GLUT_RGBA
      - Bit mask to select an RGBA mode window. This is the default if neither GLUT_RGBA nor GLUT_INDEX are specified.
    - GLUT_RGB
      - Does the same thing as RGBA, but it does not have the alpha (opacity) parameter.
    - GLUT_INDEX
- Glut Display mode, normally the bitwise OR-ing of GLUT display mode bit masks.
  - E.g. glutInitDisplayMode(GLUT_RGBA|GLUT_DEPTH)
    - Set the Display mode to RGBA and request a window with depth buffer.

(2) Clear Color Buffer

- We need to clear the color buffer before we draw any objects. If not, the background window will be messy, usually showing our desktop.



- To clear the color buffer, we use this function:
  - **glColor(GL_COLOR_BUFFER_BIT**)

(3) Set background color

- If you want to set the background color, you need to use glClearColor(…) and specify the color you want.

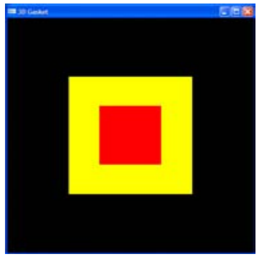
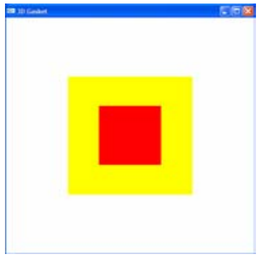
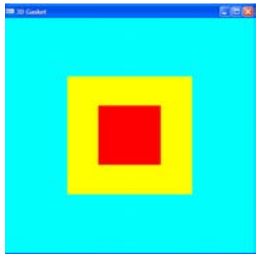
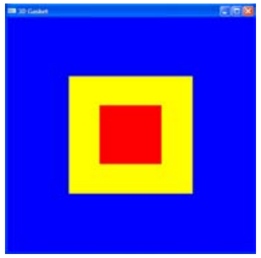
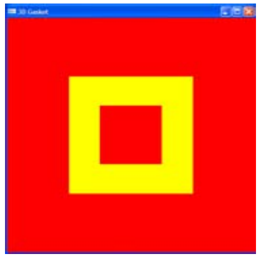
- glColor() grabs the last color defined by glClearColor to set the background color.

  - Therefore, you must use glClearColor() prior to using glColor().

  - If you don’t use the glClearColor to set the background, glClear() will set the background to its default value, which is black.

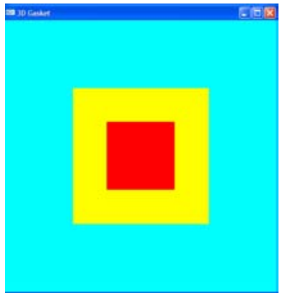
Guess what’s the background color of this display() function…

```
void display(void)
{
     glClearColor (1.0, 0.0, 0.0, 0.0); // Red
     glClearColor (0.0, 0.0, 1.0, 0.0); // Blue
     glClearColor (0.0, 1.0, 1.0, 0.0); // Cyan

/* clear frontground color buffer */
     glClear(GL_COLOR_BUFFER_BIT);

/* draw a bigger yellow polygon */
    glColor3f(1.0, 1.0, 0.0);
    //glClearColor (1.0, 1.0, 0.0, 0.0);
    glBegin(GL_POLYGON);
            glVertex2f (-0.5, -0.5);
            glVertex2f (0.5, -0.5);
            glVertex2f (0.5, 0.5);
            glVertex2f (-0.5, 0.5);
    glEnd();

/* draw a smaller red polygon */
    glColor3f(1.0, 0.0, 0.0);
    glBegin(GL_POLYGON);
            glVertex2f (-0.25, -0.25);
            glVertex2f (0.25, -0.25);
            glVertex2f (0.25, 0.25);
            glVertex2f (-0.25, 0.25);
    glEnd();
    glFlush();
}
```

Answer: Output background color is Green

```
void display(void)
{
     glClearColor (1.0, 0.0, 0.0, 0.0);          // Red
     glClearColor (0.0, 0.0, 1.0, 0.0);          // Blue
     glClearColor (0.0, 1.0, 1.0, 0.0);          // Cyan

/* clear frontground color buffer */
     glClear(GL_COLOR_BUFFER_BIT);

/* draw a bigger yellow polygon */
    glColor3f(1.0, 1.0, 0.0);
    //glClearColor (1.0, 1.0, 0.0, 0.0);
    glBegin(GL_POLYGON);
            glVertex2f (-0.5, -0.5);
            glVertex2f (0.5, -0.5);
            glVertex2f (0.5, 0.5);
            glVertex2f (-0.5, 0.5);
    glEnd();

/* draw a smaller red polygon */
    glColor3f(1.0, 0.0, 0.0);
    glBegin(GL_POLYGON);
            glVertex2f (-0.25, -0.25);
            glVertex2f (0.25, -0.25);
            glVertex2f (0.25, 0.25);
            glVertex2f (-0.25, 0.25);
    glEnd();
    glFlush();
}
```



Because glColor() only grabs the last color defined by glClearColor() to set the background color, therefore, in this case background is green.

We can write our own set background function

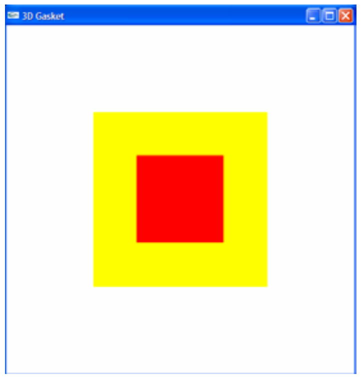
```
Void SetBackground(GLclampf red,
                   GLclampf green,
                   GLclampf blue,
                   GLclampf alpha)
   {
   /* set the clear value for color buffer */
   glClearColor (red, green, blue, alpha);

   /* clear color buffer with clear value set by
   glClearColor */
   glClear(GL_COLOR_BUFFER_BIT);
   }
```

(4) Set object color

- OpenGL uses glColor*() to set drawing color.

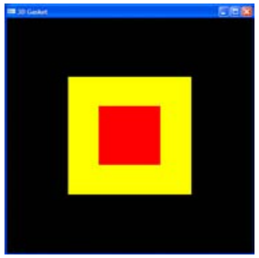
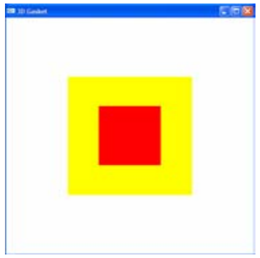
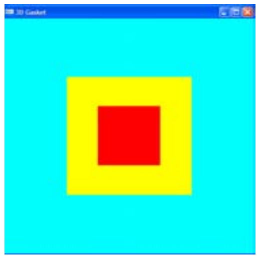
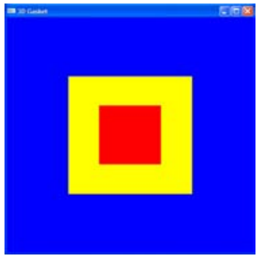
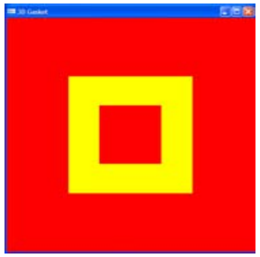
Example of how to use OpenGL RGB Colors

```
void display(void)
{

/* Set the buffer clear value to white for the background color */
     glClearColor (1.0, 1.0, 1.0, 1.0);          // white
/* clear frontground color buffer */
     glClear(GL_COLOR_BUFFER_BIT);

/* draw a bigger yellow polygon */
    glColor3f(1.0, 1.0, 0.0);
    //glClearColor (1.0, 1.0, 0.0, 0.0);
    glBegin(GL_POLYGON);
            glVertex2f (-0.5, -0.5);
            glVertex2f (0.5, -0.5);
            glVertex2f (0.5, 0.5);
            glVertex2f (-0.5, 0.5);
    glEnd();

/* draw a smaller red polygon */
    glColor3f(1.0, 0.0, 0.0);
    glBegin(GL_POLYGON);
            glVertex2f (-0.25, -0.25);
            glVertex2f (0.25, -0.25);
            glVertex2f (0.25, 0.25);
            glVertex2f (-0.25, 0.25);
    glEnd();
    glFlush();
}
```



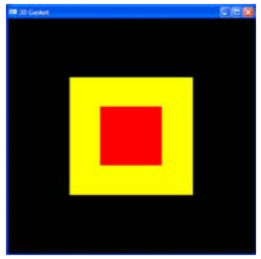
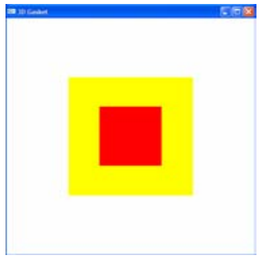
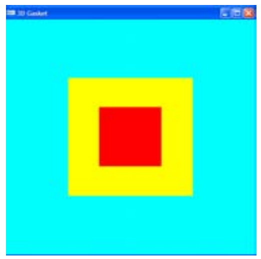
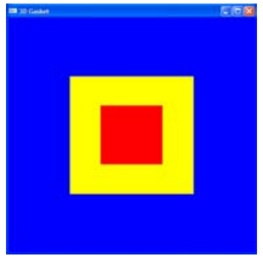
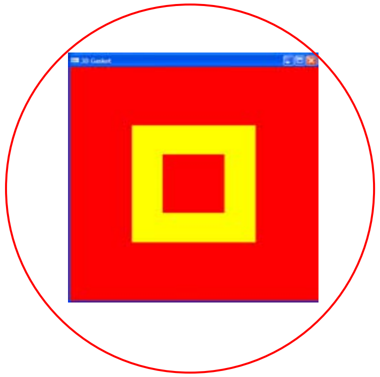
Guess what's the output of this display() function…

```
void display(void)
{
     glClearColor (1.0, 0.0, 0.0, 0.0);


/* draw a bigger yellow polygon */
    glColor3f(1.0, 1.0, 0.0);

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

/* draw a smaller red polygon */
    glColor3f(1.0, 0.0, 0.0);
    glBegin(GL_POLYGON);
            glVertex2f (-0.25, -0.25);
            glVertex2f (0.25, -0.25);
            glVertex2f (0.25, 0.25);
            glVertex2f (-0.25, 0.25);
    glEnd();
    glFlush();
}
```

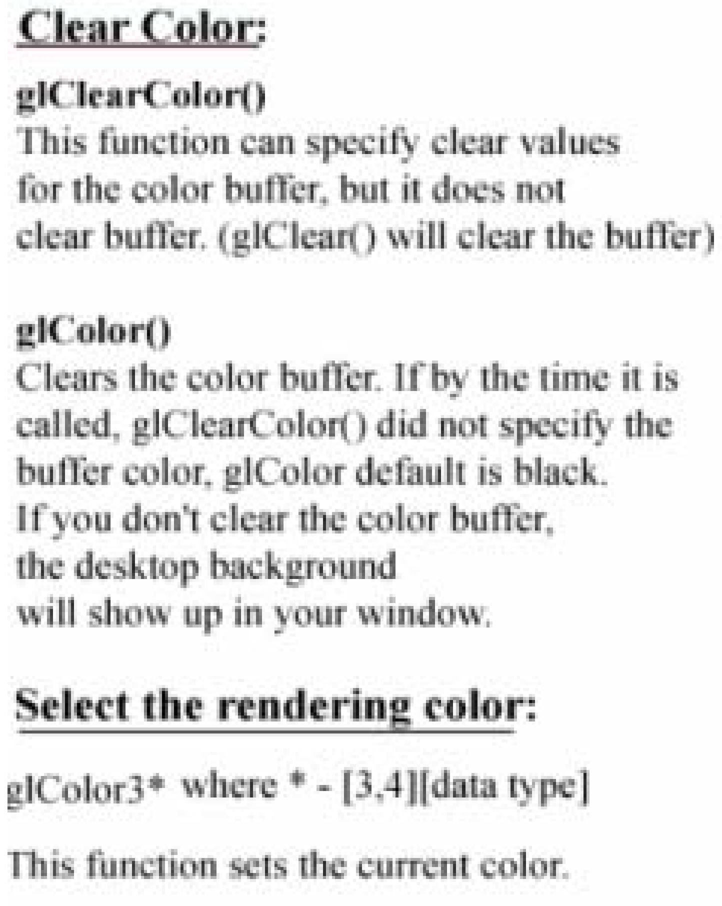






The output is …

```
void display(void)
{
     glClearColor (1.0, 0.0, 0.0, 0.0);

/* draw a bigger yellow polygon */
    glColor3f(1.0, 1.0, 0.0);

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

/* draw a smaller red polygon */
    glColor3f(1.0, 0.0, 0.0);
    glBegin(GL_POLYGON);
            glVertex2f (-0.25, -0.25);
            glVertex2f (0.25, -0.25);
            glVertex2f (0.25, 0.25);
            glVertex2f (-0.25, 0.25);
    glEnd();
    glFlush();
}
```

OpenGL Color Function Summary



Clear Color:

glClearColor()
This function can specify clear values
for the color buffer, but it does not
clear buffer. (glClear() will clear the buffer)

glColor()
Clears the color buffer. If by the time it is
called, glClearColor() did not specify the
buffer color, glColor default is black.
If you don't clear the color buffer,
the desktop background
will show up in your window.

Select the rendering color:

glColor3* where * - [3,4][data type]

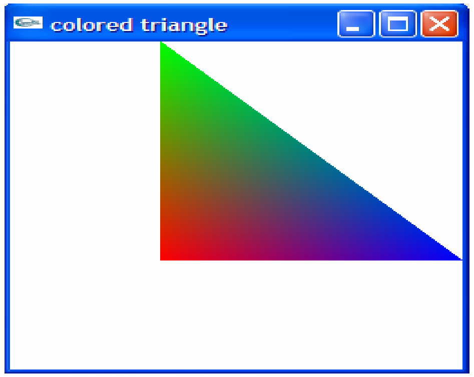
This function sets the current color.

```
void glClearColor(
            GLclampred,
            GLclampgreen,        [0,1]
            GLclampblue,
            GLclampalpha};

void glClear(GLbitfield mask);

void glColor3f(
            GLbyte red,
            GLbyte green,

            GLbyte blue);
```

Color Interpolation

```
void display(void)
{
/* clear all pixels  */
   glClear (GL_COLOR_BUFFER_BIT |
     GL_DEPTH_BUFFER_BIT);

/* a color is defined for each Vertex*/
  glBegin(GL_TRIANGLES);
       glColor3d (1.0, 0.0, 0.0);
       glVertex3f (0.0, 0.0, 0.0);

       glColor3f (0.0, 1.0, 0.0);
       glVertex3f (0.0, 1.0, 0.0);

       glColor3f (0.0, 0.0, 1.0);
       glVertex3f (1.0, 0.0, 0.0);
    glEnd();

glFlush ();
}
```

- color_inter.c


colored triangle