EBU5405
3D Graphics Programming Tools

OpenGL: Viewing

Dr. Marie-Luce Bourguet
(marie-luce.bourguet@qmul.ac.uk)

Slides adapted from Interactive Computer Graphics 4E © Addison-Wesley

Objectives

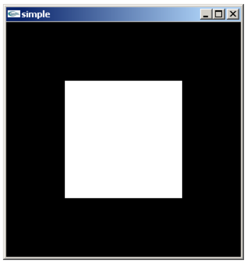
- Refine the first program
  - Alter the default values
  - Introduce a standard program structure
- Simple viewing
  - Two-dimensional viewing as a special case of three-dimensional viewing

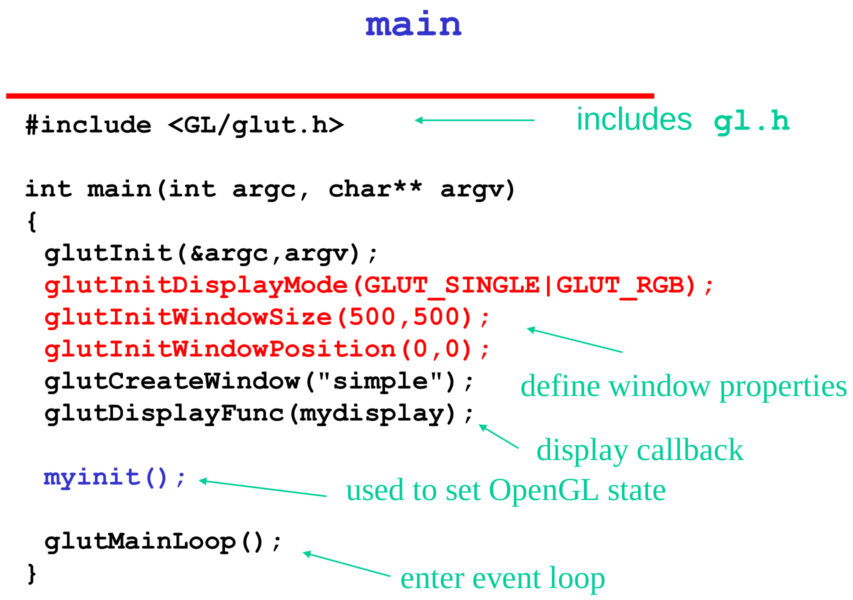
Program Structure

- Most OpenGL programs have a similar structure that consists of the following functions
  - `main()`:
    - defines the callback functions
    - opens one or more windows with the required properties
    - enters event loop (last executable statement)
  - `init()`: sets the state variables
    - Viewing
    - Attributes
  - callbacks
    - Display function
    - Input and window functions

simple.c revisited

- In this version, we shall see the same output but we have defined all the relevant state values through function calls using the default values.
- In particular, we set
  - Colors
  - Viewing conditions
  - Window properties





main

```
#include <GL/glut.h>

int main(int argc, char** argv)
{
  glutInit(&argc,argv);
  glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
  glutInitWindowSize(500,500);
  glutInitWindowPosition(0,0);
  glutCreateWindow("simple");
  glutDisplayFunc(mydisplay);

  myinit();

  glutMainLoop();
}
```

includes `gl.h`

define window properties

display callback

used to set OpenGL state

enter event loop

GLUT functions

- **`glutInit`** allows the application to get command line arguments and initializes system
- **`glutInitDisplayMode`** requests properties for the window (the *rendering context*)
  - RGB color
  - Single buffering
  - Properties logically ORed together
- **`glutInitWindowSize`** in pixels
- **`glutInitWindowPosition`** from top-left corner of display
- **`glutCreateWindow`** creates a window with title “simple”
- **`glutDisplayFunc`** declares the display callback
- **`glutMainLoop`** enters an infinite event loop

myinit

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

 glColor3f(1.0, 1.0, 1.0);

 glMatrixMode (GL_PROJECTION);
 glLoadIdentity ();
 glOrtho(-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);
}
```

black clear color

opaque colour

fill/draw with white

viewing volume

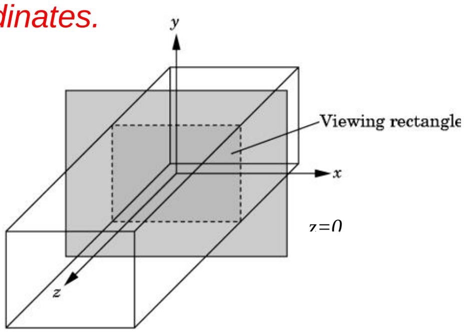
Objectives

- Refine the first program
  - Alter the default values
  - Introduce a standard program structure
- Simple viewing
  - Two-dimensional viewing as a special case of three-dimensional viewing

Coordinate Systems

- The units in **`glVertex`** are determined by the application and are called *object* or *world coordinates.*

  e.g. glVertex2f(-0.5, -0.5);



- The viewing specifications are also in object coordinates and define the size of the viewing volume that determines what will appear in the image.

  e.g. glOrtho(-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);

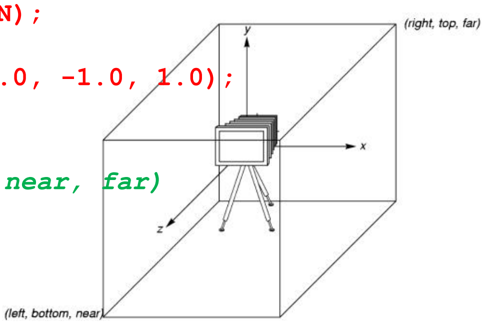
- Internally, OpenGL will convert to *camera (eye) coordinates* and later to *screen coordinates.*

OpenGL Camera

- OpenGL places a default camera at the origin in object space pointing in the negative *z* direction
- The default viewing volume is a box centered at the origin with a side of length 2

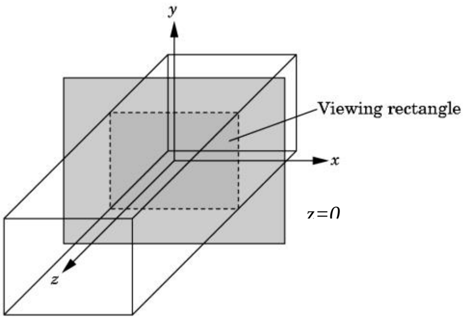
```
glMatrixMode (GL_PROJECTION);
glLoadIdentity ();
glOrtho(-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);
```

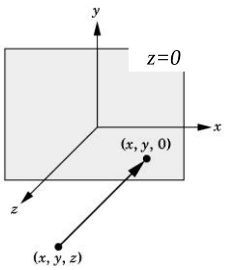
(left, right, bottom, top, near, far)



Orthographic Viewing

In the default orthographic view, points are projected forward along the *z* axis onto the plane *z=0*





Transformations and Viewing

- In OpenGL, projection is carried out by a projection matrix (transformation)
- There is only one set of transformation functions so we must set the matrix mode first

  ```
  glMatrixMode (GL_PROJECTION)
  ```

- Transformation functions are incremental so we start with an identity matrix and alter it with a projection matrix that gives the view volume

```
glLoadIdentity();
glOrtho(-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);
```

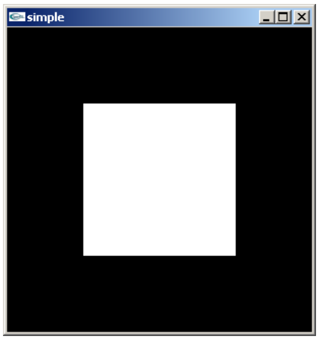
Two- and three-dimensional viewing

- In **glOrtho(left, right, bottom, top, near, far)** the near and far distances are measured <u>from</u> the camera
- Two-dimensional vertex commands place all vertices in the plane z=0
- If the application is in two dimensions, we can use the function

```
gluOrtho2D(left, right, bottom, top)
```

- In two dimensions, the view (or clipping) volume becomes a *view (or clipping) rectangle (or window)*

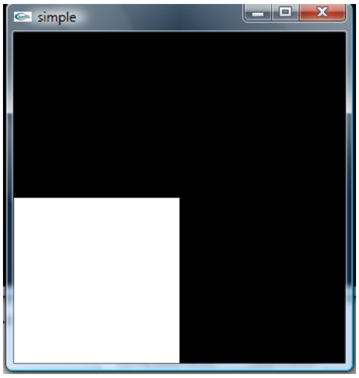
Simple Program



```
glOrtho(-1.0, 1.0, -1.0, 1.0,
           -1.0, 1.0);


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

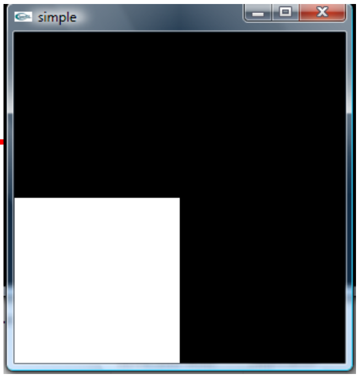
Viewing

What would you change in the Simple.c program to obtain this?



```
glOrtho(-1.0, 1.0, -1.0,
         1.0, -1.0, 1.0);

glBegin(GL_POLYGON);




glEnd();
```

Viewing

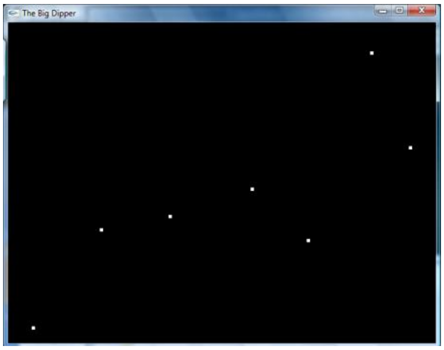


```
glOrtho

Or
gluOrtho2D


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

Viewing

What parameters would you use for the gluOrtho2D(left, right, bottom, top) function ?

GLint vertices[7][2] = {{20, 10}, {74, 74}, {129, 83}, {194, 101}, {239, 67}, {320, 128}, {289, 190}};



Viewing

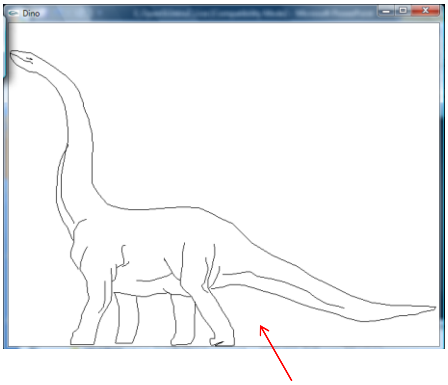
What would the result be with the following viewing parameters ?

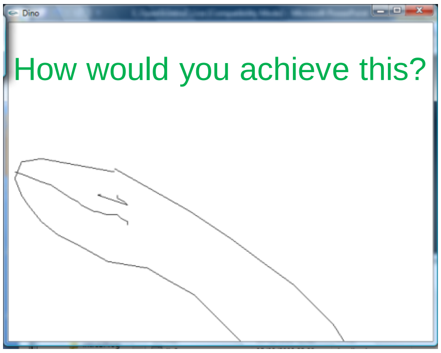
gluOrtho2D(0.0, 100.0, 0.0, 100.0);

GLint vertices[7][2] = {{20, 10}, {74, 74}, {129, 83}, {194, 101}, {239, 67}, {320, 128}, {289, 190}};

?

Using viewing windows for zooming



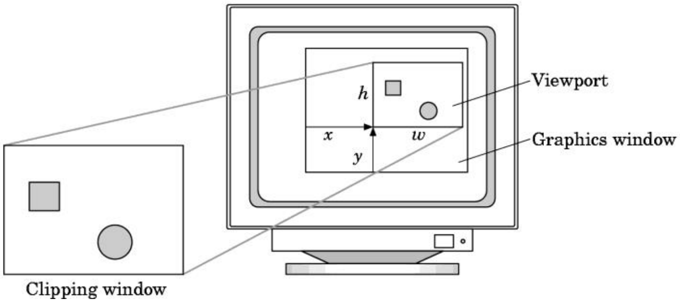


How would you achieve this?

```
gluOrtho2D(0.0, 640.0, 0.0, 480.0);
```

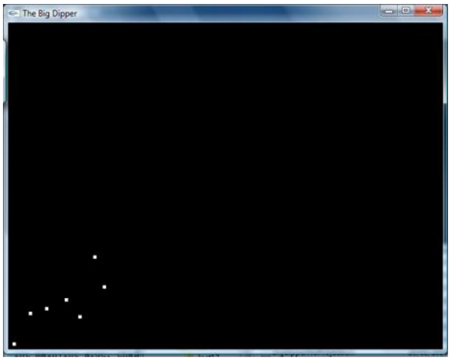
Viewports

- We do not have to use the entire window for the image: **`glViewport(x,y,w,h)`**
- Values in pixels (screen coordinates)

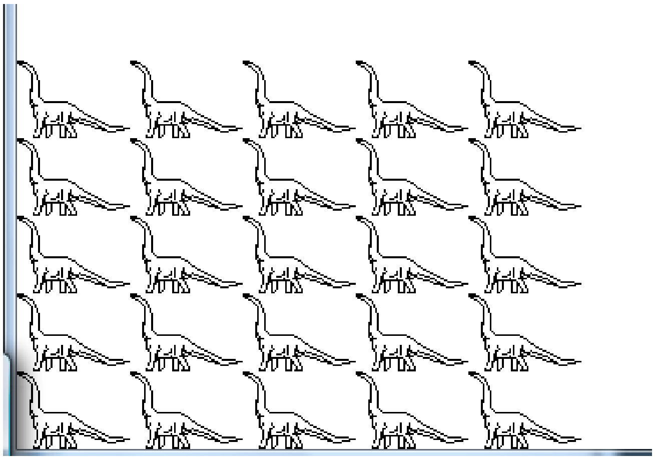


Viewports

```
glutInitWindowSize(640, 480);
glViewport (0, 0, 150, 150);
gluOrtho2D(0, 340, 0, 210);
```



Using viewports to tile a screen



Using viewports to tile a screen

```
glutInitWindowSize(640, 440);

void mydisplay(){
   int k, l;
   gluOrtho2D (0.0, 640.0, 0.0, 440.0);
   glClear(GL_COLOR_BUFFER_BIT);

   for (k=0; k < 10; k++) {
      for (l=0; l < 10; l++) {
         glViewport (k*64, l*44, 64, 44);
         drawDinosaur ……
      }
   }
   glFlush();
}
```

Using viewports to tile a screen

How would you achieve this?