**Viewing using the camera**

The idea of creating a 2D view of a 3D scene is simple: we “take a picture” of the scene using a

**camera**, and display the camera’s picture in the window on the display screen. For convenience,

OpenGL splits the process into three separate steps:

• **Step one**: First, we specify the position and orientation of the camera, using the function g**luLookAt()**;

• **Step two**: Second, we decide what kind of projection we’d like the camera to create. We can choose an **orthographic** projection (also known as a **parallel projection**) using the function **glOrtho()** or a **perspective** projection using the function **gluPerspective()** .

• **Step three**: Finally, we specify the size and shape of the camera’s image we wish to see in the window, using **glViewport()** . This last step is optional – by default the camera’s image is displayed using the whole window. In OpenGL, the camera model described above is always active – you can’t switch it off. It’s implemented using **transformation matrices.**



OpenGL keeps two transformation matrices: the **model view** matrix, M; and the **projection matrix**,

P. The model view matrix holds a transformation which composes the scene in world coordinates,

and then takes a view of the scene using the camera (step one, above). The projection matrix applies

the camera projection (step two, above).

**The window reshape function**

After creating the window, and registering the display and keyboard callbacks, we now register a new function, the reshape() callback:void **glutReshapeFunc** ( *void (\*func)(int width, int height)* );

**glutReshapeFunc()** registers the application callback to call when the window is first created, and also if the window manager subsequently informs OpenGL that the user has reshaped the window. The new height and width of the window, in pixels, are passed to the callback. Typically, the callback will use these values to define the way that OpenGL’s virtual camera projects its image onto the window, as we see in the next section.

**Specifying the projection**

We usually specify the projection in the reshape() callback function, because the projection will

often need to be adjusted if the user changes the shape of the window. In example ex4.c we use an

orthographic (also known as “parallel”) projection:

void reshape (int width, int height)

{ /\* Called when the window is created, moved or resized \*/

glViewport (0, 0, (GLsizei) width, (GLsizei) height);

glMatrixMode (GL\_PROJECTION); /\* Select the projection matrix \*/

glLoadIdentity ();

glOrtho(-1.0,1.0, -1.0,1.0, -1.0,1.0); /\* The unit cube \*/

glMatrixMode (GL\_MODELVIEW); /\* Select the modelview matrix \*/

}

**Modelling using transformations**

OpenGL maintains two separate 4 × 4 transformation matrices:

• the **model view** matrix; this is used to accumulate modelling transformations, and also to specify the position and orientation of the camera;

• the **projection** matrix; this is used to specify how the 3D OpenGL world is transformed into a

2D camera image for subsequent display. The projection matrix performs either a perspective

or othographic (parallel) projection.

At any time, one or the other of these matrices is selected for use, and is called the **current matrix**, or sometimes C for short. Most of the OpenGL functions for managing transformations affect the contents of C.



glMatrixMode (GL\_MODELVIEW); /\* Select the modelview matrix \*/

glLoadIdentity (); /\* Set the current matrix to identity \*/

glTranslatef (tx, ty, tz); /\* Post-multiply by (tx,ty,tz) \*/

glScalef (sx, sy, sz); /\* Post-multiply by (sx,sy,sz) \*/

**glMatrixMode()** selects the matrix stack, and makes the top matrix on the stack the “current matrix”(C). glMatrixMode selects the matrix stack, as follows:

• GL MODELVIEW: selects the modelview matrix stack;

• GL PROJECTION: selects the projection matrix stack.



Once a current matrix has been selected using **glMatrixMode()**, all subsequent matrix functions (such as **glRotatef()**, etc.) affect the current matrix. For example, to load a translation by (x, y, z) into the modelview matrix, the code would be:

glMatrixMode (GL\_MODELVIEW);

glLoadIdentity ();

glTranslatef (x, y, z);

Subsequent matrix operations will continue to affect the current modelview matrix, until **glMatrixMode()** is called again to select a different matrix.

There are a number of utility functions for changing the current matrix.

**glLoadIdentity()** sets the current matrix C to be the identity matrix I:

**Translation**

void **glTranslatef** ( GLfloat *x*,GLfloat *y*,GLfloat *z* );

**glTranslatef()** creates a matrix M which performs a translation by (x, y, z)

**Scaling**

void **glScalef** ( GLfloat *x*,GLfloat *y*,GLfloat *z* );

**glScalef()** creates a matrixM which performs a scale by (x, y, z)

**Rotation**

void **glRotatef** ( GLfloat *angle*,GLfloat *x*,GLfloat *y*,GLfloat *z* );

**glRotatef** creates a matrix M which performs a counter-clockwise rotation of angle degrees. The axis about which the rotation occurs is the vector from the origin (0, 0, 0) to the point (x, y, z)

There are two functions which operate on the current matrix stack: **glPushMatrix()** and **glPopMatrix()**.

void **glPushMatrix** ( void );

**glPushMatrix()** Pushes the current matrix stack down one level. The matrix on the top of the stack is copied into the next-to-top position

void **glPopMatrix** ( void );

**glPopMatrix()** pops the current matrix stack, moving each matrix in the stack one position towards the top of the stack.

Following fig shows the stack after executing glPushMatrix()



**glPushMatrix**();

– Save the state.

– Push a copy of the CTM onto the stack.

– The CTM itself is unchanged.

• **glPopMatrix**();

– Restore the state, as it was at the last Push.

– Overwrite the CTM with the matrix at the top of the stack.









