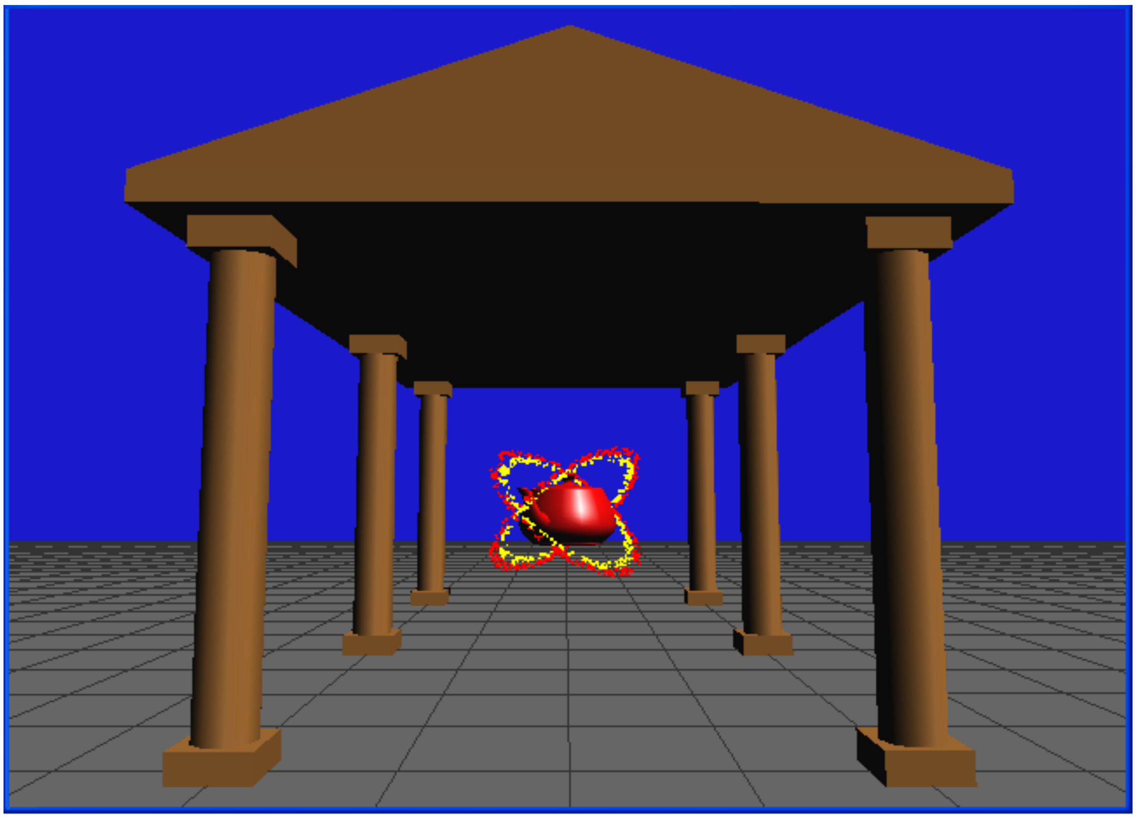
COMP27112 Laboratory Exercise 1

Navigating around a virtual world

Duration: 1 lab session; Marks: 10

Learning outcomes

1. To understand the use of 3D viewing and navigation in 3D
2. To understand the use of spherical coordinates
3. To understand simple interaction techniques



1 Preparation

Important: before you start this exercise, make sure you've read Chapter 9 of the OpenGL manual ("Viewing").

We're providing a skeleton program to get you started. So first take your own copy:

```
$ cd ~/COMP27112
$ mkdir ex1
$ cd ex1
$ cp /opt/info/courses/COMP27112/Lab/ex1/ex1.c .
```

Take a look at **`ex1.c`**, and check it compiles OK with **`cogl`**. Ponder the mystical object. Look at the code that implements it.

2 The exercise

You are to implement some code which will allow you to use the keyboard and mouse to "walk around" the 3D scene. The view specification model is based upon spherical polar coordinates, as described in Appendix 1 of this document (you might like to go and read that now).

Recall that in OpenGL, views may be specified using the **`gluLookAt()`** function:

```
gluLookat (eyex, eyey, eyez,           // eye point
           centerx, centery, centerz, // look-at point
           upx, upy, upz);            // up vector
```

Where the **eye point** is the location of the viewer in the scene, the **look point** is a point we're looking at, and the **up vector** says which way is up (we'll use the positive Y direction in this exercise).

In the spherical polar coordinate model we use two angles to record the current view direction: a **longitude**, which records our horizontal view angle (in the XZ plane) measured leftwards from the positive Z direction, and a latitude, which records our vertical view angle above or below the XZ plane.

Given a pair of view angles it's possible to compute a look point which is situated on the surface of a unit sphere centred on the eye point, and use this within the **`gluLookAt()`** function.

2.1 Task 1: View rotation

Complete the function **`calculate_lookpoint()`**, which uses the global view angles **`lat`** and **`lon`**, together with the current eye point (**`eyex`**, **`eyey`**, **`eyez`**) to calculate a look point (**`centerx`**, **`centery`**, **`centerz`**). Note that the **`sin()`** and **`cos()`** functions of the C library expect their arguments to be in radians not degrees, so you should always multiply any angle parameters by the program-defined constant **`DEG_TO_RAD`**. Now implement the **`cursor_keys()`** function to perform the following view modifications:

- **GLUT_KEY_LEFT**. Rotate your view to the left (increase longitude angle) by TURN_ANGLE degrees.
- **GLUT_KEY_RIGHT**. Rotate your view to the right (decrease longitude angle) by TURN_ANGLE degrees.
- **GLUT_KEY_PAGE_UP**. Tilt your view up (increase latitude angle) by TURN_ANGLE degrees.
- **GLUT_KEY_PAGE_DOWN.** Tilt your view down (decrease latitude angle) by TURN_ANGLE degrees.
- **GLUT_KEY_HOME**. Re-centre (set back to zero) your latitude angle.

When modifying the latitude angle you should ensure that it is never allowed to reach (or exceed) an angle of +90 or -90 degrees. This is because a 90 degree view angle would co-align the view direction vector and the fixed up vector (which points up the world coordinate Y axis), resulting in an ambiguous view specification. Going beyond

90 degrees will produce the effect of turning the viewer head-over-heels, which may be a little disorienting!

2.2 Task 2: Movement

Extend the cursor_keys and keyboard functions as appropriate to perform the following translations to the eye point:

- **`GLUT_KEY_UP`**. Step forwards in the XZ plane a distance **`RUN_SPEED`**. The direction of movement (the heading angle) is towards the look point, and therefore corresponds to the longitude angle.

- **`GLUT_KEY_DOWN`**. Step backwards in the XZ plane a distance **`RUN_SPEED`**. The heading is in the opposite direction to the longitude angle.

- **`Comma`**. Move (strafe) to your left in the XZ plane a distance **`RUN_SPEED`**. The heading is given by the longitide angle +90 degrees.

- **`Full-stop`**. Move (strafe) to your right in the XZ plane a distance **`RUN_SPEED`**. The heading is given by the longitide angle -90 degrees.

A movement of distance D in the direction given by the heading angle will have components Dx and Dz, as shown below. The Y component of the eye point should remain unchanged, as we're implementing a walking navigation model and the viewer cannot fly up or down in the scene.

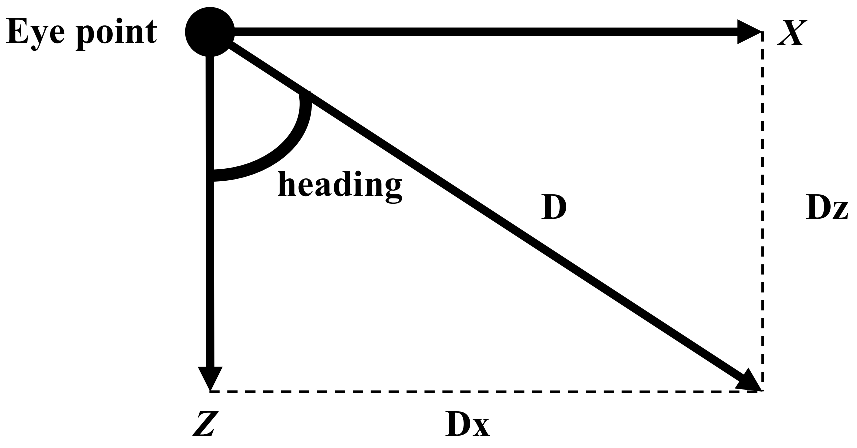


Figure 1: Components of the heading angle (plan view).

2.3 Task 3: Mouse look

It's convenient to allow mouse movements to be used as a way of applying fine adjustments to the specified view direction. This can be achieved by modifying the mouse_motion function which is called by **`glutPassiveMotionFunc()`**. You should compute a pair of additional view angles **`mlon`** and **`mlat`** which are linear functions of mouse coordinates **x** and **y**, as shown in Figure 2. The (xy) displacement of the mouse from the window centre therefore corresponds to an offset of up to 50 degrees in the view longitude and latitude angles.

Now, rather than using the mouse independent view angle (**`lon`**, **`lat`**) in your view calculations, you should use the base view angle with the mouse offset angle applied: (**`lon + mlon`**, **`lat + mlat`**). Once again you should respect the latitude angle constraints described in Task 1.

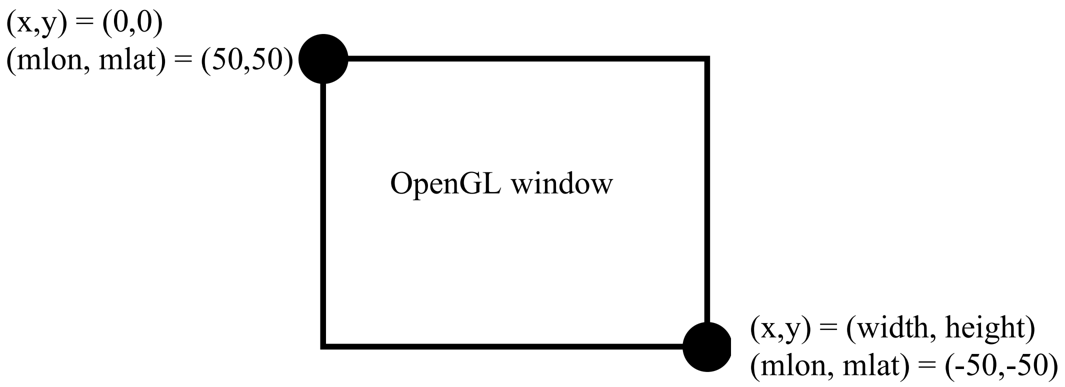


Figure 2: Relationship between mouse position and view offset angles.

2.4 Task 4: Crouching and jumping

Implement crouching, so that pressing the **C** key makes the eye point drop by 0.5 (say). Hitting **C** again restores the eye point to its original height. Implement jumping, so that hitting the **`SPACE`** key initiates a short vertical jump animation. The eye point is raised over a sequence of frames, and is then lowered again. This extension requires a bit more thought, and is likely to involve modifications to the function called by **`glutIdleFunc()`**.

3 Deliverables

You must **submit** your completed **`ex1.c`**. There are 10 marks available:

Task	**Marking criteria**	**marks**
1 View rotation	Rotation working	1
	Keyboard key control	1
	Latitude angle constraints working	1
2 Movement	Movement working	1
	Movement direction correct	1
	Keyboard key control	1
3 Mouse look	Mouse look working	1
	Angle constraints working	1
4 Crouching and jumping	Crouch, jump	2

4 Other ideas (optional, educational, and fun, but not marked)

- Shoot ‘em up. Yes, you guessed, it – implement a weapon and animate it firing projectiles of some kind, or an energy beam. No gory graphics, though, please.
- Make the mystical object even more mystical by making it pulsate with light, or be held within a semi-transparent energy beam, or anything you like.

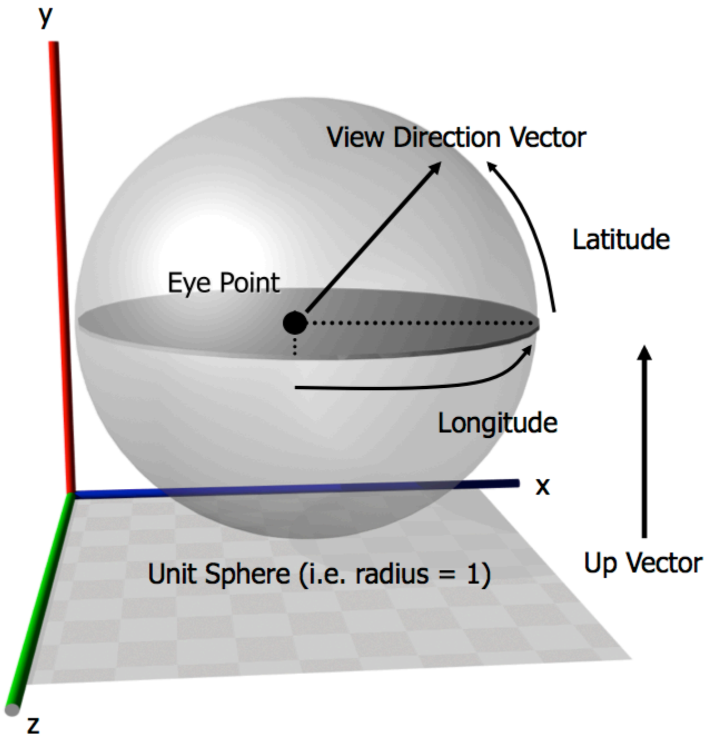
- Bring a friend. The exercise as it stands assumes there is only one inhabitant of the virtual world – you. Implement another inhabitant, and use a key to switch between being "you" and the other inhabitant, controlling the view and movement of whoever is currently selected, as before. Each inhabitant should also be able to see the other in the environment. Maybe use GLUT's 3D primitives to give yourself and the other inhabitant platonic-solid "bodies".

Appendix 1: Spherical Polar Coordinates

One way of specifying viewing parameters is to use "Spherical Polar Coordinates". This is commonly used in 3D games.

The method uses an eye point and view up vector as with **`gluLookAt()`**, but instead of a "centre of interest" point, we describe the direction of view in terms of:

- a longitude angle for the horizontal plane (say, with respect to the world Z-axis)
- a latitude angle for the vertical plane (say, with respect to the XZ plane)



This gives a viewing scheme in which the view is in a 'general direction' rather than being aimed 'at a specific object'. This is a more natural way of specifying head movement for 'lookaround' in a 3D world.

Converting between spherical coordinates and Cartesian coordinates

There are two steps.

First, calculate the view direction based on the latitude and longitude. From simple trigonometry, think of 2 triangles, one in the plane YZ, the other in the plane XZ. So we have:

$$\begin{aligned} dir\_x &= \cos(lat) \,.\, \sin(lon) \\ dir\_y &= \sin(lat) \\ dir\_z &= \cos(lat) \,.\, \cos(lon) \end{aligned}$$

Next, the look point is then the eye point with the view direction applied as an offset:

$$\begin{aligned} centre\_x &= eye\_x + dir\_x \\ centre\_y &= eye\_y + dir\_y \\ centre\_z &= eye\_z + dir\_z \end{aligned}$$

Or, if we're representing centre, eye and direction as vectors **C**, **D** and **E**, we have:

$$\mathbf{C} = \mathbf{E} + \mathbf{D}$$

[ends]