OpenGL and Computer Graphics

- OpenGL provides a Hardware Abstraction Layer, through its Applications Programmer Interface.
- It:
  - provides low level, platform independent, graphics.
  - allows non-standard extensions.
- It does not:
  - provide windowing facilities (we use GLUT for that).
  - contain high level modelling constructs, such as scene graphs.

OpenGL– it is a low level thing

- OpenGL should work on almost all systems.
- Most graphics cards support hardware implementations of OpenGL commands.
- OpenGL is largely an immediate mode graphics library (except display lists) – you specify what is to be drawn and it is sent to the display buffer.
- We would have to write our own higher level retained mode library – or use one of the existing ones.

OpenGL Primitives

- OpenGL uses only a very small number of primitives:
  - points,
  - lines,
  - polygons,
  - bitmaps / images.
- These primitives are then passed through:
  - the lighting / shading algorithms,
  - the 3D viewing algorithms,
  - and finally rasterisation (scan conversion).

GLUT

- GLUT is the OpenGL Utilities Toolkit – provides us with a basic window and interaction management system.
- `glutInit(&argc, argv);` – Initialise GLUT.
- `glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);` – allows us to set up the way OpenGL will run – other options `GLUT_DOUBLE`, `GLUT_DEPTH`. Can create and position, size and name a window.
- Callbacks provide event (interrupt) driven interaction with keyboard, mouse, display and resizing.
- Callback functions must receive the specified parameters.

Basic OpenGL

- The basic command is `glVertex*#`.
  `*` defines the number of coordinates we will give – generally 2, 3 or 4. `#` defines what type the arguments are. The most commonly used options are:
  - `i` for `int` (`GLint`), `f` for `float` (`GLfloat`),
  - `d` for `double` (`GLdouble`), `ub` for `unsigned char` (`GLubyte`).
- Other OpenGL commands such as `glColor*#`, `glRasterPos*#`, `glNormal3#` also have this syntax.
- An additional `v` may be specified at the end if we want to pass an array (vector).

Drawing with OpenGL

- The way the primitives are drawn on screen is determined by the drawing mode.
- `glBegin(`*mode*`);` and `glEnd();` must always enclose calls to `glVertex*#`.
- The drawing mode can be:
  - GL_POINTS, GL_LINES, GL_LINE_STRIP, GL_LINE_LOOP,
  - GL_TRIANGLES, GL_QUADS, GL_POLYGON.
- There are other options (we won't use them).

Colour and OpenGL

- Before drawing, clear the screen buffer using `glClear` with option `GL_COLOR_BUFFER_BIT`.
- Set the background colour using `glClearColor(0.0,0.0,0.0,0.0)`.
- Set the colour of the vertices using `glColor*#` – note each vertex can be a different colour.
- We always use RGB colours, for different types we have:
  - `f` and `d` take 0.0 to 1.0, `ub` takes 0 to 255.
- The fourth value specifies the alpha value, used in blending to mimic transparency.

Styles and OpenGL

- The way polygons are drawn can be set using `glPolygonMode` which applies to either face (`GL_FRONT` or `GL_BACK`) and can be:
  - `GL_POINT`,
  - `GL_LINE`,
  - `GL_FILL`.
- Lines can be styled using `glLineWidth(GLfloat width)`, and `glLineStipple(GLint factor, GLushort pattern)`.
- Points can be changed in size using `glPointSize(GLfloat size)`.
- This is really for basic 2D drawing.

Controlling OpenGL

- `glFlush()` causes OpenGL to flush to the screen buffer – draw the image.
- When animating, use `glutSwapBuffers()` and the double buffer mode, giving smoother animation.
- GLUT also provides us with a `glutIdleFunction` which contains the animation routine which typically calls `glutPostRedisplay()`.
- Sometimes we use global variables to control the animation (if we haven't produced a higher level scene graph).
- Make them static and use with care!

Transformations and OpenGL

- Basic commands are:
  - `glTranslate#(dx, dy, dz)`
  - `glScale#(sx, sy, sz)`
  - `glRotate#(angle, x, y, z)`
- We use `glPushMatrix()` and `glPopMatrix()` to 'save' the matrix stack.
- The matrices are applied to the vertices in the opposite order they are specified.
- Can define our own matrices: `glLoadMatrix` and `glMultMatrix`.

Viewing in OpenGL

- OpenGL viewing definition uses the camera analogy.
- Two matrices define the total projection:
- `GL_PROJECTION` defines the projection – the lense.
- `GL_MODELVIEW` controls both the objects and the view – the positioning.
- We will come back to this once we have looked at 3D → 2D projections.

Display Lists and Vertex Arrays in OpenGL

- Display lists allow precompiled objects, but these must be static. Can give a significant speed up, since the compiled OpenGL can be stored on the graphics card, and quickly drawn.
- Vertex arrays allow objects to be stored in arrays (of vertices, colours and normals). It is agreed that they do not always provide a speed up.
- Use display lists to define complete, static objects on which you want to apply transformations.

OpenGL

- We have covered the very basics of OpenGL.
- There remains a great deal of material which will be covered:
  - viewing and 3D graphics; lighting and materials;
- and this will not be covered:
  - texture, bump and environment mapping; NURBS curves.
- If in doubt consult the online manual, a reference book or me.