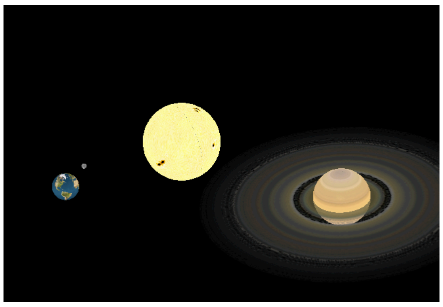
CS4102 CG Practical 5 - 2011 - Texture Mapping and Scene Graphs or Particle Simulation Modelling

Due Date: December 6th 23.59
If you are using OpenGL on a Mac please use XCode and zip up the project so it can be run and tested. If you are using the command line please describe how the code it to be compiled and tested. If some other environment please likewise explain. If you use Java please submit a JAR file. Along with this please submit a short yet descriptive 3 page report on it in PDF including screen grabs from your program and a video of your program in operation.



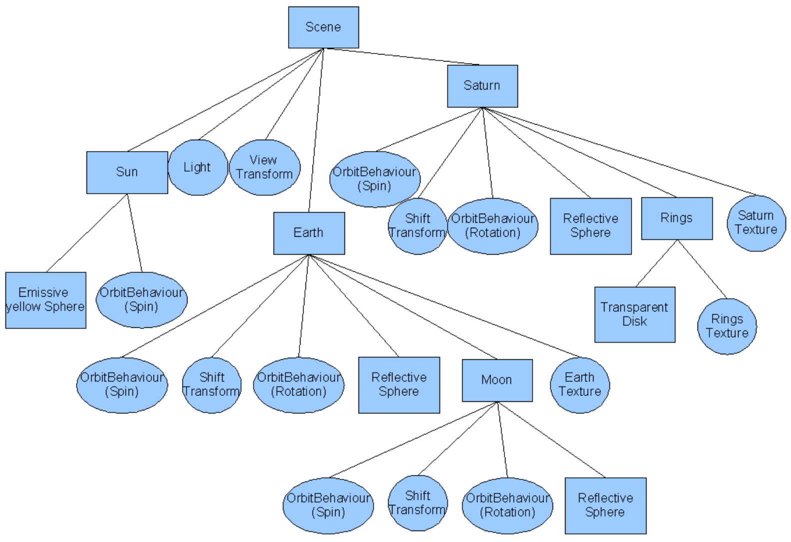
Aims and Objectives:
The aim of the practical is to gain familiarity with texture mapping in OpenGL building on your previous practical or by starting a new project on 3D graph layout. The aim is to demonstrate the use of texture mapping and a scene graph or particle simulation with texture mapped elements.

Resources
World images shown above can be obtained from http://planetpixelemporium.com (note copyright use)

Option 1:

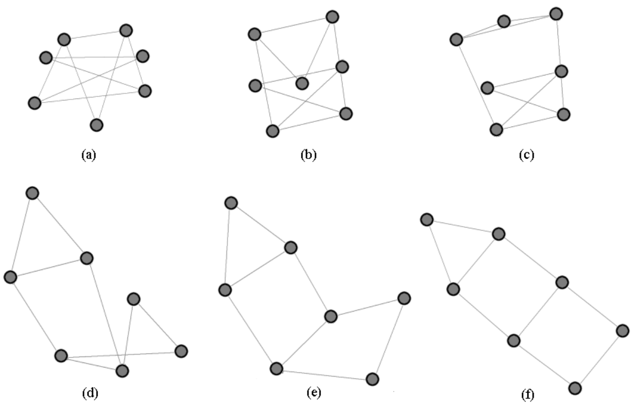
Write two applications:
(1) a basic texture-mapped planet world where you extend your practical 4 with a simplified (eg. single plane of motion), animated, sensibly **textured** and lit model of the solar system. Your model will also have textured rings around different planets as appropriate and at least the earth (and its texture) should be rotating as it moves along it path around the sun. (as shown in the image above).

(2) re-factor or re-code your world so that the elements are stored in a hierarchical model (data structure). The following shows what a Java Scene Graph might look like, a C structure (eg. struct) will need careful thought to ensure it's easy to traverse and hold the relevant details.

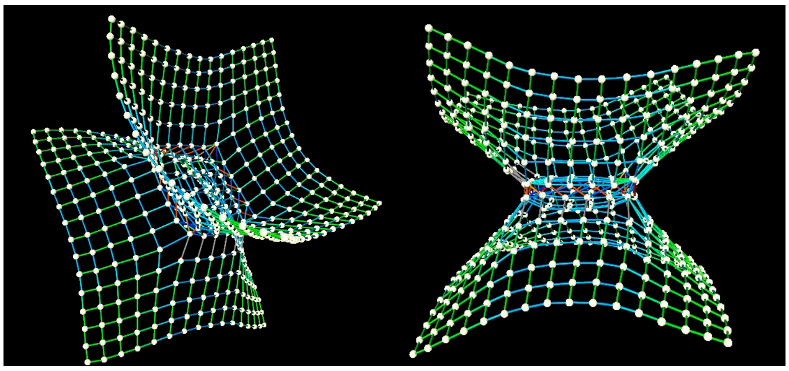


Option 2:

Force directed graph drawing algorithms, also know as spring algorithms or spring embedders are popular methods for drawing graphs (consisting of nodes and edges) in 2D or 3D. Force directed algorithms tend to emphasise symmetry, maximise edge length uniformity, maximise the distance between non-adjacent nodes, and as by-product tend to minimise the number of edge crossings. Force directed algorithms view the graph as a virtual physical system, where the nodes of the graph are bodies of the system. These bodies have forces acting on or between them. Often the forces are physics-based, and therefore have a natural analogy, such as magnetic repulsion or gravitational attraction.



An example of a 7 node and 9 edge graph can be seen above. Starting from a random layout in 2D of the nodes the forces in a force-directed layout iterate and in effect push the unconnected nodes apart while holding the connected one together. Here the edges can be modelled as gravitational attraction and all nodes have an electrical repulsion between them. It is also possible for the system to simulate unnatural forces acting on the bodies, which have no direct physical analogy, for example the use of a logarithmic distance measure rather than Euclidean. Regardless of the exact nature of the forces in the virtual physical system, force directed algorithms aim to compute a locally minimum energy layout of the nodes. This is usually achieved by computing the forces on each node and iterating the system in discrete time steps. The forces are applied to each node and the positions are updated accordingly. Force-directed algorithms are often used in graph drawing due to their flexibility, ease of implementation, and the aesthetically pleasant drawings they produce (as shown below or above). However, classical force directed algorithms are unable to handle larger graphs due the inherent O(n^2) cost at each timestep, where **n** is the number of bodies (nodes) in the system.



Resources
You are provided the original spring layout paper by Eades and the CS4102_Spring_Layout.pde and node.pde and edge.pde code to study (you are not asked to use this code).

Write one application:
(1) either by reading a graph defined in a file into your program or by hard-coding a simple graph model into your code (as shown in the example processing code), computer a force-directed layout (in 3D for all the nodes). Either iterate for a fixed number of timesteps or under the overall movement falls below a threshold you decide. Following this, draw a texture mapped sphere for every node in its 3D location and a line for every edge (simple lines are fine). Your model should be lit so that the nodes and edges can be seen.

A simple file format for a graph might be

```
#My graph file name
Number_of_nodes Number_of_edges
1 2
2 3
3 4
4 5
5 1
```

(2) for a better grade extend this program so that when started a random x,y,z location is given to each node, and when a particular key is pressed the force directed layout starts iterating and the movements of the nodes are displayed on screen (from random to final layout). An exceptional answer should allow a user to click on a node and drag it in 3D (using a key press + mouse movement to allow for X-Z versus X-Y movement of the node), while the force-directed layout is being computed.

Learning Outcomes
An answer getting a grade of **Merit** will have a simplified, basic motion, sensibly **textured** and lit model of the solar system. For option 1 your model will also have textured rings around different planets as appropriate and at least the earth (and its texture) should be rotating as it moves along it path around the sun. For **first rank** and above your program or programs should have a clear and well thought out hierarchical model which is used by your code to traverse and display as appropriate. An **exceptional** answer should allow a user to pick a planet and swap its texture.

For option 2, a grade of Merit will have correctly lit texture mapped nodes positioned in 3D (*according to a force-directed layout*) with edges (using simple lines) connecting the nodes as appropriate. For a basic grade no motion of the nodes and edges is required (ie. A static 3D layout with texture mapping). An answer of **first rank** should start the graph node positions in a random 3D layout and then move the nodes and edges overtime *according to a force-directed layout.* An **exceptional** answer should allow a user to click on node and drag it in 3D (using a key press + mouse movement to allow for X-Z versus X-Y movement of the node), while the force-directed layout is being computed!