DUBLIN CITY UNIVERSITY

SEMESTER TWO SOLUTIONS 2010

MODULE: EE563 Graphics and Visualisation

COURSE: MEN – M.Eng. in Electronic Systems
MEQ – Masters Engineering Qualifier Course
MTC – M.Eng. in Telecommunications Engineering

YEAR: Postgraduate (C)

EXAMINERS: Prof. Peter Ashburn (External Examiner)
Dr Robert Sadleir, Section A, Ext no. 8592
Dr Derek Molloy, Section B, Ext no. 5355

TIME ALLOWED: 3 Hours

INSTRUCTIONS: Please answer any TWO questions from Section A and any TWO questions from Section B. All questions carry equal marks.

Please do not turn over this page until you are instructed to do so

The use of programmable or text storing calculators is expressly forbidden. Please note that where a candidate answers more than the required number of questions, the examiner will mark all questions attempted and then select the highest scoring ones

Section A

Q 1 (a)

(i) A directed-acyclic graph is a directed graph in which there are no other cycles i.e. the beginning at one node in the graph, a path cannot be found to return to the same node. [2 marks]

(ii) The ViewSpecificGroup node is a Group whose descendants are rendered only on a specified set of views. It contains a list of views on which its descendants are rendered. [2 marks]

(iii) Many shapes share vertices between their different faces. Index geometry allows the list of vertices to be defined once and then the shape itself is defined by specifying indices into the vertex list. Each index is represented by and int (4 bytes) and each vertex coordinate is represented by 3 floats (12 bytes) so a lot of memory can be saved if there is significant vertex reuse. [2 marks]

(iv) This shading model interpolates the colour at each vertex across the primitive (e.g. line, polygon, quad) and consequently the primitive is drawn with many different colours. [2 marks]

(v) Magnification is where a single pixel of the rendered geometry corresponds to a small portion of a texel. Minification is where a single pixel of the rendered geometry corresponds to an area of the texture, i.e. several texels. [2 marks]

(vi) A bounding region is used to specify the region of the virtual world that a particular environment affects. If a bounding region is not specified for a particular environment node then it is considered to be inactive. [2 marks]

(vii) A maximum intensity projection (MIP) is a simple technique for transforming 3D data sets to a 2D image. It involves identifying the maximum valued pixel along each ray projected from the pixels of the 2D view plane. The resulting maximum value is ultimately assigned to the associated pixel. [2 marks]

Q1 (b) [2 marks]

(i) Non-uniform scale

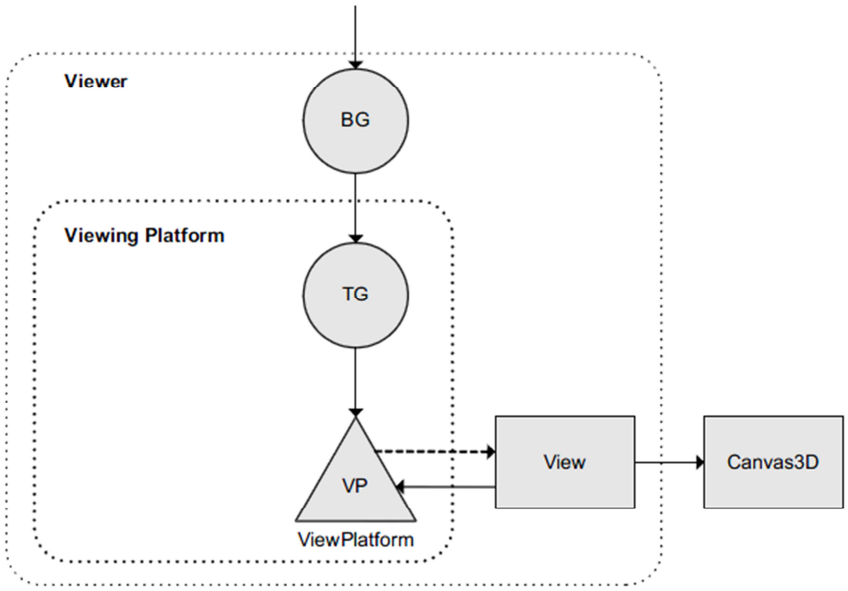
$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} sx & 0 & 0 & 0 \\ 0 & sy & 0 & 0 \\ 0 & 0 & sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

(ii)Translate [2 marks]

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & tx \\ 0 & 1 & 0 & ty \\ 0 & 0 & 1 & tz \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

Q1 (c)

The view branch created by the **`SimpleUniverse`** utility class is a BranchGroup routed scene graph consisting of a TransformGroup and a ViewPlatform. This is illustrated below



The ViewPlatform represents the location of the viewer within the scene. The scene is rendered from the perspective of the ViewPlatform to a canvas and the rendering process is facilitated by the View which defines all the parameters needed to render a 3D scene from a single viewpoint.

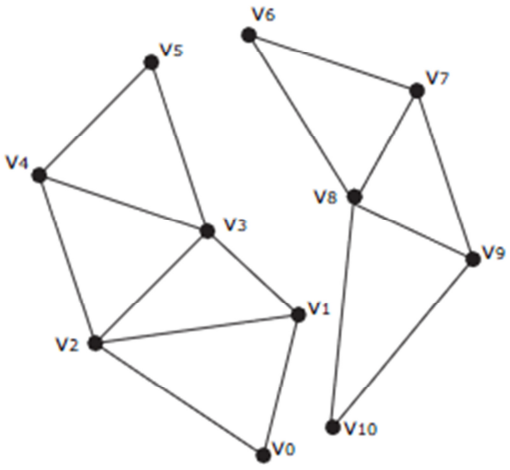
[3 marks] for scene graph
[4 marks] for description

Q2 (a)

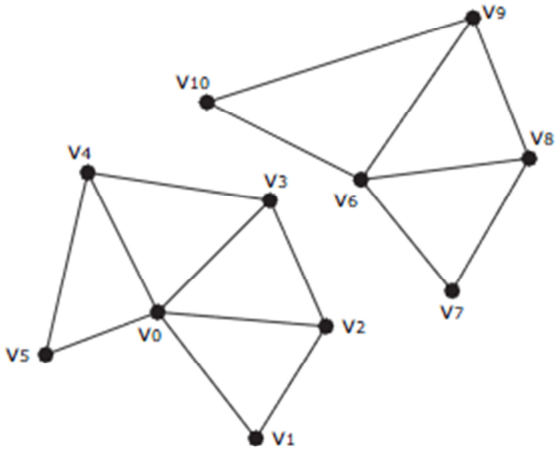
Strip Geometry reduces the need to repeatedly specify the same vertices when defining continuous pieces of geometry. Additional pieces of geometry are created using by specifying a single additional vertex to be used in conjunction with the previous vertices of the geometry
[2 marks]

TriangleStripArray draws its array of vertices as a set of connected triangle strips. An array of per-strip vertex counts specifies where the separate strips appear in the vertex array. For every strip in the set, each vertex, beginning with the third vertex in the array, defines a triangle to be drawn using the current vertex and the two previous vertices.
[2.5 marks]

TriangleFanArray draws its array of vertices as a set of connected triangle fans. An array of per-strip vertex counts specify where the separate fans appear in the vertex array. For every strip in the set, each vertex, beginning with the third vertex in the array defines a triangle to be drawn using the current vertex, the previous vertex and the first vertex.
[2.5 marks]



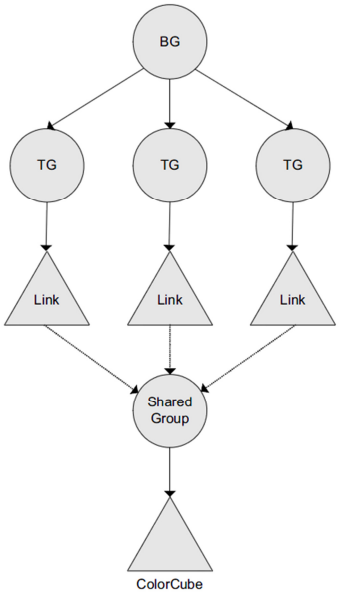
Triangle Strip Array



Triangle Fan Array

Q 2(b)

A SharedGroup enables a subgraph to be shared between different groups via Link leaf nodes. The essentially allows the same content to be replicated several times within a single scene. A SharedGroup may be referenced by one or more Link leaf nodes. Any runtime changes to a node or component object in a shared subgraph affect all graphs that refer to that subgraph. Only Link leaf nodes may refer to SharedGroup nodes. A SharedGroup node cannot have parents or be attached to a Locale object. A shared subgraph may contain any group node, except an embedded SharedGroup node as SharedGroup nodes cannot have parents.



[5 marks]

One example of a node that can't be replicated using a SharedGroup node is a Background node. The reason for this is because a there can only be one background in a scene.

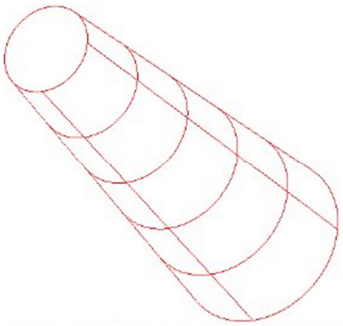
[1 marks]

Q2(c)

The depth values of all pixels generated by polygon rasterization can be offset by a value that is computed for that polygon. Two values are used to specify the offset.

- **Offset bias** - the constant polygon offset that is added to the final device coordinate Z value for the polygon primitive. [1 mark]
- **Offset factor** - the factor to be multiplied by the slope of the polygon and then added to the final device coordinate Z value of the polygon. [1 mark]

Polygon offset is used to solve a specific problem: drawing lines on top of polygons. The typical usage of this is in drawing a "hidden line" display. This is a form of wire-frame display in which a solid object is drawn as lines so that the lines hidden by the foreground are removed. An example of a hidden line display is illustrated below.[1 mark]



[1 mark]

Q2(d)

(i) **Texture2D**: specifies a 2D image that is to be mapped to the exterior of a particular geometry. The Texture2D class also specified the state of the texture, the boundary mode i.e. how the texture appears for texture coordinates outside the range [0, 1] and the texture filtering mode specifies how the texture is drawn when it is larger or smaller than its original size. [2 marks]

(ii) **TextureAttributes**: defines the attributes that apply to texture mapping, such as the texture mode, texture transform, blend colour, and perspective correction mode. [2 marks]

(iii) **TexCoordGeneration**: defines the attributes that apply to texture coordinate generation, such as whether coordinate generation is enabled, coordinate format (2D or 3D coordinates), coordinate generation mode (object linear, eye linear, or spherical reflection mapping), and the R, S and T coordinate plane equations. [2 marks]

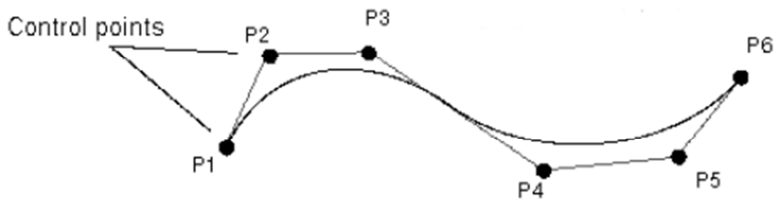
(iv) **TextureUnitState**: an array that defines the texture state for each of N separate texture units. This allows multiple textures to be applied to a geometry. Each TextureUnitState object contains a Texture object, a TextureAttributes object and a TexCoordGeneration object. [2 marks]

Q 3(a)

A B-spline curve is a smooth path that is defined by a series of control points and blending functions. A B-spline curve does not pass through its control points. It is a complete piecewise cubic polynomial consisting of any number of curve segments.

[3 marks]

Qi is the ith B-spline segment and Pi is a set of four points in a sequence of control points. The value for u over a single curve segment is $0 \leq u \leq 1$. Using this notation u represents a local parameter, locally varying over the parametric range 0 to 1 to define a single B-spline curve segment. It is clear that a B-spline curve is a series of m¡ 2 curve segments that are labelled Q3, Q4, ..., Qm defines or determined by m + 1 control points P0, P1, ..., Pm; $m \geq 3$. Each curve segment is defined by four control points and each control point influence four and only four curve segments. An example of a B-spline curve is illustrated below:



[4 marks]

Java 3D provides support for a variation on B-splines known as Kochanek-Bartels cubic splines. These types of spline are also known as a TCB splines as they have configurable

tension, continuity and bias characteristics. Unlike B-splines, TCB splines do go through the control points.

[2 marks]

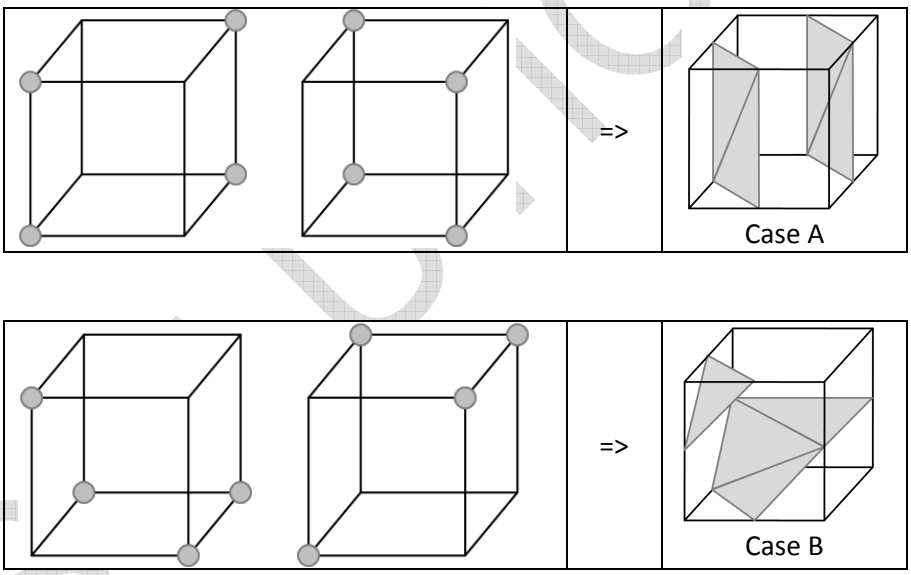
Q3(b)

The marching cubes algorithm begins by thresholding the data set, assigning a 1 to voxels ≥ diso (inside the isosurface) and a 0 to voxels < diso (outside the isosurface). A cubic mask of size 2 x 2 x 2 is then passed through the volume and at each mask location the configuration of the eight underlying voxels is examined and the relevant surface patches are generated.

[3 marks]

There are 256 (2^8) possible configurations of eight binary voxels and although possible, the task of manually specifying the surface patches associated with each configuration is both tedious and prone to error. This task can be greatly simplified by considering all complementary cases (& rotations) for each configuration.

[2 marks]



[2 marks]

The use of complementary cases as outlined above can cause holes in the mesh generated by the marching cubes algorithm. The holes are due to ambiguous cases resulting from mismatches between the surface patches of adjoining cubes. The ambiguous cases that result in unwanted holes are a direct result of the use of complementary cases in the standard MCA to reduce the number of core cube configurations that must be specified. By disregarding complementary cases and using only rotation to identify equivalent cube configurations the number of core configurations increases from 15 to 23.

[2marks]

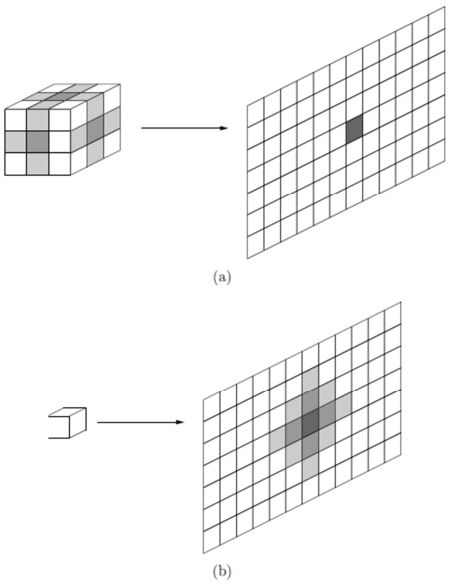
Q3(c)

The term splatting is used to describe the effect that one voxel has on the image plane. This algorithm considers how the contribution of a voxel should be spread or splatted in the image plane. The two possible ways that splatting can be used are:

[3 marks]

(a) A point in the data set at the centre of a particular voxel projects onto a single pixel. The three dimensional region surrounding the sampled voxel is filtered to determine its contribution to the pixel. [2 marks]

(b) A single pixel value can be spread over a number of pixels in the image plane. [2 marks]



Section B

4(a) There are many different unique solutions to this problem. One possible solution is that you would create classes as follows:

Scene Object
- Camera
  - Viewer View – with overlay car controls
  - Possible out of car view
- Geometric Scene Object -> would have textures and bounding boxes
  - Other cars
  - Race track scenery
  - Bullet
  - Particles? - Smoke, rain etc.
  - Ground Plane
- Light
  - Realistic lighting from the sun, glare in windscreen, mirrors
  - Ambient lights for global illumination
- Physics Properties
  - Collisions
  - Movement model, inertia, acceleration, friction etc.

These relationships could exist as inheritance relationships – e.g. a Smoke Cloud IS-A Geometric Scene Object IS-A Scene Object. The Scene Graph would be designed to have a root node and all of the scene objects could be added to this node in a tree structure. The scenegraph would allow the objects to be related to each other for undergoing translations, rotations etc. E.g. if there was a geometric model representing a driver in a car – if the car translates then so should the driver. Given the huge amount of geometric information and the inter-relationships that must occur between all the objects in the scene it is vital that certain efficiencies are employed, e.g.:
- Binary Space Partitioning to establish the spatial relationship between different scene objects efficiently.
- Culling and Clipping – to only display the polygons that are visible to the viewer. The clipping, culling should be complex, to reduce cost, where it should be based on the overlay from the controls also.
- The use of convex hulls and bounding boxes to help with the spatial relationships of the object – e.g. have two cars collided? – first evaluate against the convex hull/ bounding box.
- Use mip-mapping and billboarding to reduce the number of polygons in the scene and replace their geometries with textures/imposters, e.g. cars/scenery objects (e.g. Trees) in the distance.

[8 marks]

4(b) This SceneObject container will work for all of the scene objects in (a)

```
#include<vector>  // Use the STL Vector as our container

enum RTTI_OBJECT_TYPE
{
        RTTI_CAMERA,     //does not exist yet
        RTTI_LIGHT,      //does not exist yet
        RTTI_DUMMY,
};

class SceneObject
{

  protected:
        SceneObject* parentObject;
        std::vector<SceneObject*> childrenObjects;
```

```
    public:
        SceneObject();
        virtual ~SceneObject();

        // assessors/mutators
        void setParent(SceneObject* parent)      { parentObject = parent; }
        void addChild(SceneObject* child);
        std::vector<SceneObject*>* getChildrenObjects() { return &childrenObjects; }
        virtual RTTI_OBJECT_TYPE getType() const = 0;

        // Force every child to have a render and update methods
        virtual void render(float timeElapsed) = 0;
        virtual void update(float timeElapsed) = 0;
};

void SceneObject::addChild(SceneObject *child)
{
        if (!child) { std::cerr << "Attempt to add invalid child to scene graph."; }

        child->setParent(this);
        childrenObjects.push_back(child);
}
```

Need to describe how it works.

[9 marks]

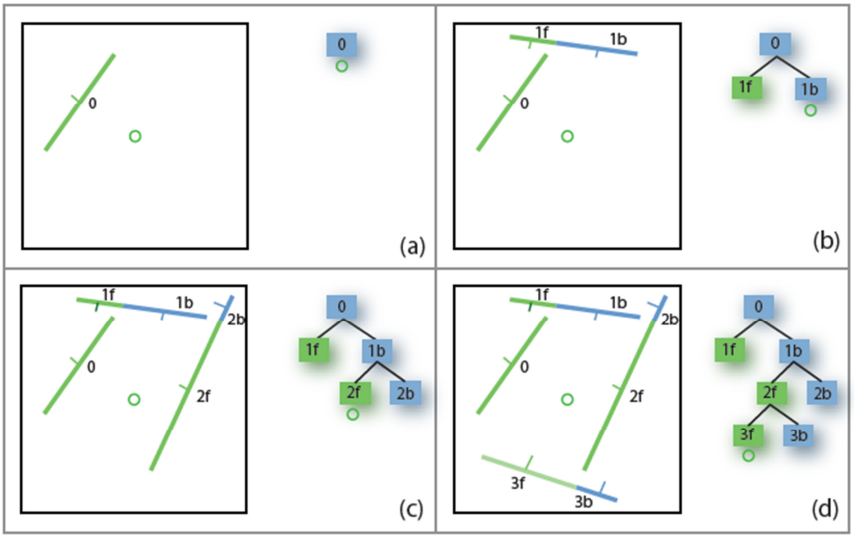
4(c) This is the recursive depth first traversal code:

```
class SceneObject;  //avoid a circular definition

class SceneGraph
{
  public:

        SceneGraph();
        virtual ~SceneGraph();

        void updateScene(SceneObject* sceneObject, float timeElapsed);
};

void SceneGraph::updateScene(SceneObject* sceneObject, float timeElapsed)
{
        if (!sceneObject)
        {
                std::cerr << "Attempt update of object not on the scene graph.";
                return;
        }
        else
        {
                sceneObject->update(timeElapsed);
                // and call all the children
                std::vector<SceneObject*>::iterator it = sceneObject->getChildrenObjects()-
>begin();
                for (; it!=sceneObject->getChildrenObjects()->end(); ++it)
                {
                        updateScene(*it, timeElapsed);
                }
        }
}
```

There would be many reasons to traverse the scenegraph in the game:
- Check that a bullet object has collided with any object after a certain number of milli-seconds. This would be a non-display test that would call the updateScene() method (possibly using a thread with a delay).
- Establish the viewers view volume and determine the objects (and therefore polygons) that are present in that view.
- Determine an updated BSP tree for the scene as bullets and other actors move independently of the viewer.
- Display the scene through the eyes of the viewer.

[8 marks]

5(a) Binary Space Partitioning (BSP) is a technique for recursively subdividing a 3-D space into two non-overlapping regions using a plane, referred to as a hyperplane. Any point in 3-D space lies within only one of these regions. BSP is a hierarchal approach where the space that is divided can be further subdivided using the same space partitioning approach until some condition is met, resulting in a space-partitioning tree, which is particularly useful when building techniques for dealing with hidden surface removal. The basic properties of BSPs are that objects on one side of a hyperplane cannot intercept an object on the other side; and given a particular view point objects on the same side of the hyperplane are closer than objects on the other side.



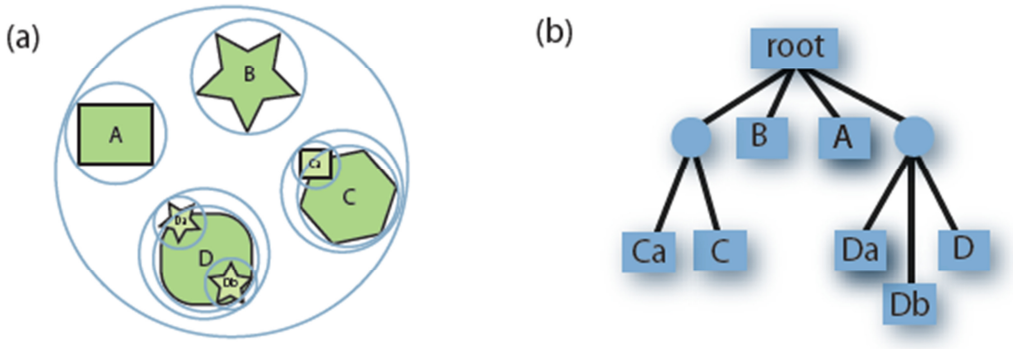
This 2-D example illustrates the sample creation of a BSP Tree using lines to create the tree. Each line represents a partition plane (a) creates the partition plane and thus the root node, (b) through (d) illustrates the addition of further planes, with f representing the front side and b representing the back side.

[6 marks]

5(b) The algorithm to build a BSP tree is:
• Select a partition plane - The choice of planes is application dependent, but often axis aligned. In an ideal situation this will result in a balanced tree, but a poor choice will result in a large number of splits and an increase in the number of polygons. There is usually a trade-off between a well-balanced tree and a large number of splits.
• Segment the current set of polygons using the chosen plane - If a polygon lies entirely to one side or other of the plane then it is not modified and is added to the partition set for the side that it is on. If the polygon spans the partition plane then it is split into two pieces, which are added to the set on the correct side of the plane.
• Repeat again using the new sets of polygons - The termination condition is a application specific, often based on a maximum number of nodes in a leaf node, or maximum tree depth.

[3 marks]

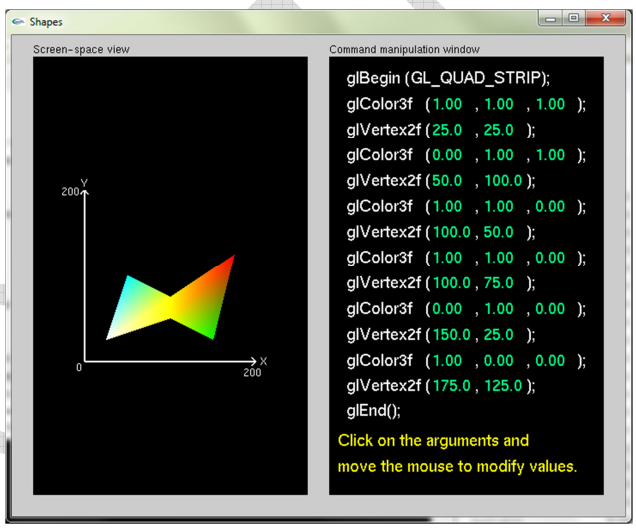
5(c) A Bounding Volume Hierarchy (BVH) is a tree of bounding volumes where the root node includes every object in the scene and at the leaf nodes each bounding volume is just large enough to contain each scene object. The tree takes on the same hierarchical shape as the scene graph. Once again, we can quickly determine if an object is in a particular region of space using its bounding volume, but the hierarchy also allows us to determine if the segmented objects' bounding volumes contained in the child nodes are also within this region of space. The tree like structure allows all these tests to be performed very quickly. It is common practice for us to use the same object-oriented tree structure for the scene graph and also for the bounding volume hierarchy. This figure illustrates the BVH approach.



The BVH Approach is a hierarchial approach, which may require a priori knowledge of the structure of the objects in the scene. BSP does not rely on any higher a priori knowledge, rather it partitions space entirely. BSP can segment individual objects into collections of polygons depending on the way that the space is segmented.

[3 marks]

5(d)



Output of code on RHS on LHS.

[8 marks]

5(e) Clipping against the view volume removes objects which are not within the view volume, where typically polygons outside of the view volume are discarded and polygons that intersect the view volume boundary are 'clipped'. Culling is a similar operation that takes account of the viewer's position to determine if part of an object cannot be seen; for example, in 3-D (the real world!) most objects are only 50% visible from any one point, as the back side of the object is occluded by the front side. A very simple test to see if we are looking at the front-side or back-side of a polygon is to use the view vector ~n and the polygon's normal vector ~np, where a particular polygon is visible if: $\sim np \cdot \sim n > 0$. Similarly if the polygon is outside the bounding box of the view volume then it can be clipped.

[5 marks]

6(a)

- Specular surfaces - These surfaces appear shiny as the light that is reflected is maintained within a narrow range of angles, close to the angle of reflection. Mirrors are perfect specular surfaces.
- Translucent surfaces - These surfaces allow some of the light to penetrate the surface and to emerge from some other location on the object. For example, refraction in glass or water would cause the light to emerge from another location on the object.
- Diffuse surfaces - These surfaces are characterised by having light scattered in all directions; for example, walls painted with a matte paint are diffuse reflectors.

[3 marks]

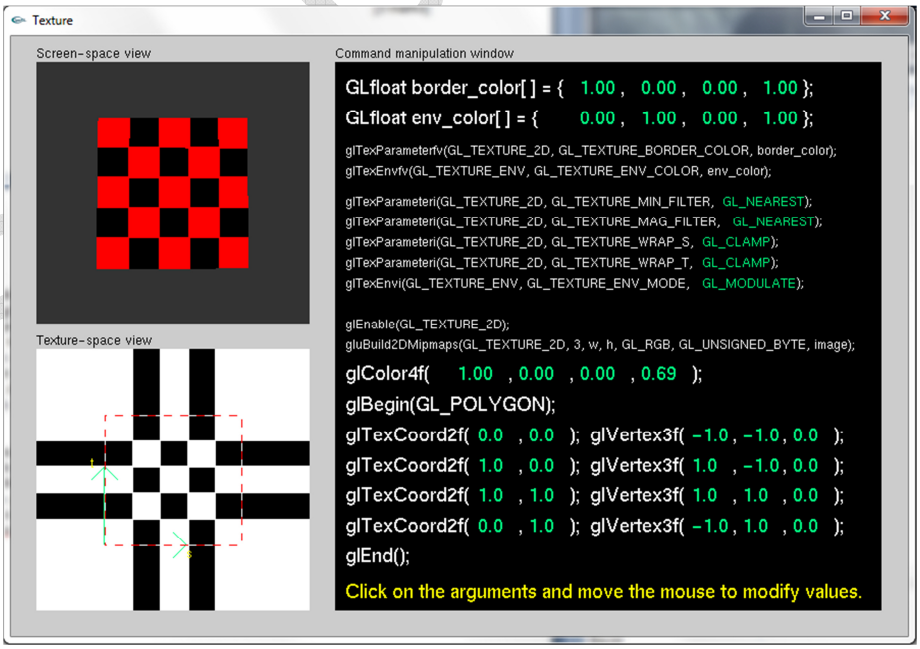
6(b)

The Phong Reflection model provides a good approximation to physical reality, producing good renderings under varying lighting conditions and materials. The Phong model uses four vectors to calculate the colour at a particular point P on a surface; these are n, the normal vector at that point on the surface; v, which is in the direction from point P to the viewer (or centre of projection); l, the direction of a line from P to a point light source; and r is the direction that a perfectly reflected ray from l would take. The Phong model supports the three types of material-light interaction of ambient, diffuse and specular. OpenGL works by assuming that if there is a set of point sources that each source can have separate red, green and blue ambient, diffuse and specular components.

Diffuse relections are characterised by rough surfaces, where rays of light that strike the surface are reflected back at quite different angles. Perfectly diffuse surfaces are called Lambertian Surfaces and can be modelled by Lambert's law, which states that: , where theta is the angle between the normal at the point of interest n and the direction of the light source l. If only a fraction of incoming light is reflected we can add in a reflection coefficient kd (where 0 < kd < 1) we can write: Rd = kdLd cos theta.

[7 marks]

6(c)

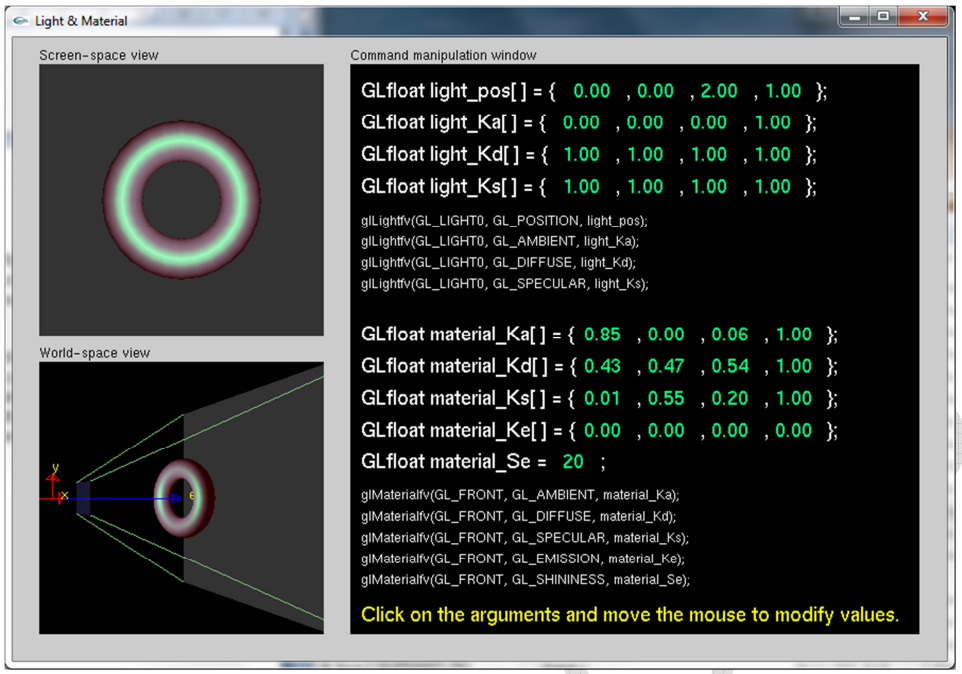


The code required is on the RHS.

[3 marks for the texture space - inc. use of GL_CLAMP]
[3 marks for the mapping of TexCoord to glVertex]
[2 marks for use of GL_MODULATE]

[8 marks Total]

6(d)



It should be clear that the torus will be red with the centre of the ring accented by a green highlight. The solution should describe each one of the arguments and what they mean.

[7 marks Total]