Computer Graphics Problem Sheet 3

1.
   a.
      i. **Spatial Occupancy Enumeration**:
      A list of all the cells is enumerated which indicates for each cell whether it contains a bone or not. This could be done with an array of bits with a "1" representing "bone" and a "0" representing "not bone".
      ii. **Octree**:
      An Octree branches 8 times from the root and 8 times at every node thereafter. Each leaf is a bit or a boolean representing (as above) "bone" or "not bone". Each branch divides space into 8 segments (front up left, front up right, front down left, front down right, ditto for back up left &c.). If a segment of space is all bone it is represented by a "bone" leaf and if it has no bone it is represented by a "not bone" leaf. Otherwise it is divided up further.
      iii. **K-d tree**:
      For a K-d tree representation each node and leaf represents a "bone" voxel and any voxels not in the tree are "not bone" (or vice versa). Each node is annotated with a split-plane axis as well as the location of a voxel in space. This split-plane axis determines the structure of the tree according to the following rule: If a node is annotated with a point p and an axis a then all the left children of that node have a lower value on the a axis than p and all the right children have a greater value on the a axis than p.
      iv. **BSP tree**:
      As above, but generalized such that the split-panes need not be axis aligned.
      v. **Axis-aligned bounding volume hierarchy**:
      A tree is constructed where each node is annotated with an axis aligned cuboid. A node is a child of its parent if and only if the cuboid it represents is entirely contained within the cuboid represented by its parent. These cuboids may not partially intersect (that is to say they can only totally contain each other or not intersect at all). The leaves of this tree are voxels containing "bones" and any voxels not in the tree contain "not bone" (or vice versa).
2.
   a. A shader allows finer control over the GPU's rendering pipeline beyond parameterising fixed functions. A shader performs a given independent operation once for each element of some set (determined by the kind of shader). Types of shader include:
      i. **Pixel (Fragment) Shaders**:
      Describe a function performed once for each pixel. They are normally used to apply lighting values (e.g. shadows, specular highlights) and for colouring the pixels drawn to the screen. They can also be used for basic image processing techniques such as blurring, sharpening and edge detection
      ii. **Vertex Shaders:**

Describe a function performed once for each vertex. They are normally used to transform the 3D position of each vertex in virtual space to its 2D position. This shader can be modified to alter the kind of projection performed (e.g. parallel, fisheye, perspective) and to annotate vertices for later stages in the pipeline (e.g. depth values for the Z-Buffer).

iii. **Geometry Shaders**:
These are run once for each 2D primitive output by the vertex shader and are used to further refine the shapes output before they are rasterised. This can be used to subdivide faces, smoothing shapes out (geometry tessellation) or generate more complex scenes from simple descriptions (rendering voxels from pixel positions, or rendering blades of grass over a field).

b. A z-buffer stores an extra number for each pixel in the display as primitives are rasterized (it is initialized with a value of infinity). When these primitives are rasterised they are only drawn to the screen if their depth value is closer to the camera than that in the z-buffer (and when this happens the value of the z-buffer is overwritten). *I'm not sure how a z-buffer maintains the required variation in resolution but I'd guess its by allocating more values to closer points than to further points (e.g. by storing some function of the square root or the logarithm of the depth value).* Using a Z-buffer instead of the painter's algorithm introduces the added cost of having to keep track of the depth value of each vertex but removes the need to compute which shapes need to be drawn first, and makes drawing intersecting shapes more straightforward.

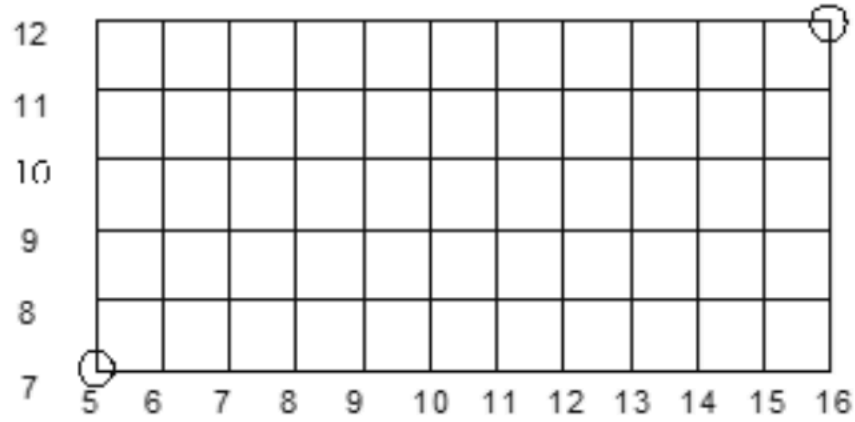
c. Both Display Lists and Vertex Buffer objects are interfaces opened up by OpenGL for storing data which is likely to be reused by the GPU.

   Display lists work by prebuilding objects and copying them into the display-list memory of the graphics hardware. This means that if several identical objects are being drawn, the compilation and memory writing steps only need to take place once. The major drawback of this is that the compiled objects allow very little modification before being drawn (beyond a basic translation or rotation).

   Vertex buffer objects create immutable areas of memory in the graphics memory of the GPU for storing an object which will be rendered multiple times. These stored objects can be updated and changed when required (at a small performance cost), unlike display lists that are immutable once stored in memory.

d. The Sutherland-Hodgman algorithm for line clipping works as follows: For each line segment in the polygon, extend that line segment infinitely and remove any portion of the line being clipped which is on the out side of the infinitely extended line segment. To generalize for a polygon the boundary of the polygon needs to be redrawn when part of the polygon falls outside the line. Special attention must be paid to edge cases such as when the polygon being clipped has one of the lines of the window as a tangent.

e.



3.

a. The definition of "realism" in computer graphics varies depending on the application of the image being drawn.
   "Functional realism" describes the level of realism required to convey some information about the real world to a user. The image shown however, does not need to be convincing as a real word scene nor even remotely accurate (functional realism can be conveyed through symbols or cartoony diagrams).
   "Physical realism" describes the level of realism required to correctly model physical phenomena – it should provide the same visual stimulation as a real world scene without using "cheats" or "hacks" which merely provide the right visual response in our brains.
   "Photo realism" describes the level of realism required to look real to human eyes. The actual image as rendered does not need to by physically accurate, so long as it is convincing.

b. A straightforward way to validate synthetically generated scenes would be to compare the generated scenes with measurements taken of analogous real world scenes. This however leads to the following challenges:
   i. Constructing a real world scene which is the same as a synthetic one (and vice versa) is difficult and requires taking and recording accurate measurements.
   ii. Taking and recording accurate measurements of real scenes is hard. Furthermore we need to ensure that the measurements we take to compare the two scenes are not the measurements we took for constructing them (as this would invalidate the methodology).
   iii. Given two scenes to compare, there are a variety of criteria on which to compare them (see more below). It is not inconceivable (for example) that a synthetic image could offer a greater level of believable realism than a photograph, even though the photograph offers a greater level of predictive realism (the key difference between the two is that predictive realism is specifically about looking like the real world, rather than about convincing a viewer that a scene is real).

   The three key areas to be considered when validating an image (by comparison) are:
   i. Goniometric comparison (namely, whether the measurement of the scene items is correct)
   ii. Radiometric comparison (whether the lighting model is correct)
   iii. Perceptual comparison (whether the output displayed is correct)

c. The key difference between believable and predictive image synthesis is that the former is validated in terms of whether it is convincingly real to human eyes (that is as compared to our mental models of how the world looks), whereas the latter is validated in terms of whether it is indistinguishable from some set of real world measurements (that is as compared to measurements of the real world).

d. For a static scene, the irradiance volume algorithm allows us to precompute the global illumination of a scene. This works by dividing space up into a series of zones and precomputing the global lighting for a

point in each of those zones. At render time this data is used to light surfaces in each zone.

e. Ambient occlusion is a technique for computing self shadowing effects by modelling the local blocking of light as constant regardless of the direction of the light source. This technique can be further sped up by performing these calculations in screen-space (that is on the GPU after the scene has been converted to a raster during the fragment shader stage) and using the depth information stored during this stage to add the local shadows.

4.

a.

b. (Assuming that by "supporting the Lambertian and Phong illumination model" the question means that these are all that is used to compute the illumination of any given point.

```
Type Pixel: {int x, int y, float rayAngle, rgb color}
Type Screen = [Pixel]

for each (pixel in screen):
        line = constructLine(pixel.x, pixel.y, rayAngle);

        for each (shape in scene):
            (collides?, angle, point) = LMC(line, shape)
            if(collides):
                for each (light in scene):
                    pixel.color = Phong(
                        new Vector(angle, 1),
                        vectorBetween(point, light.position),
                        point.normal,
                        light.brightness,
                        shape.material.brightness
                    )
            else:
                pixel.color = backgroundcolor
```

c. When raytracing, if a ray hits an object we then need to decide if there is an unobstructed line between the light and that object. For this we cast a ray towards the light sources (called the shadow ray) and see if it intersects any shadow casting objects before reaching the light source.

d. If we model a light as a point source, all the shadows cast by objects illuminated by the light will have hard edges. In order to get more realistic shadows we can cat several rays from a light source to generate penumbra and antumbra effects.

e.