Ans. to Tut 7

Qn 1

a)

i) For Z buffering, the number of comparison and update operations is proportional to the total number of polygons pq in the scene. Hence the computational complexity is $O(pq)$.

ii) When the objects are simple, the intersection test can be done easily for each object. For each pixel, there are p intersection test.

This is repeated for each of the $n \times m$ pixels. Hence the computational complexity is $O(nmp)$.

iii) When the objects are complicated, an intersection test involves root finding which is complicated and may not have an analytical solution. In this case, intersection tests can be done easily with each of the polygon of each of the object. This involves or each pixel, pq intersection test.

Hence the computational complexity is $O(nmpq)$.

b) For Z buffering, we need an additional Z buffer which is the same size as the image but each pixel stores a double. Hence the memory complexity is $O(nm)$

For ray casting, there are many intersection tests, but they are independent and each intersection test involves constant resources. Hence the memory complexity is constant, denoted by $O(1)$.

Qn 2

a) Pixel ray equation

$$(X, Y, Z) = (0,0,2) + s\left[(2,4,0) - (0,0,2)\right]$$

Note: One can normalize $(2, 4, -2)$ to a unit vector, but this would only affect the magnitude of s and will not affect finding the intersection point. So we skip this step.

Normalizing to get unit vector u is useful if you wish to find v, v = -u.

b) Put into

$$(X-10)^2 + (Y-24)^2 + (Z+10)^2 = 84$$

$$\Rightarrow (2s-10)^2 + (4s-24)^2 + (12-2s)^2 = 84$$

$\Rightarrow \quad 24s^2 - 280s + 736 = 0$

$\Rightarrow$ s = 4 and 7.666666667

Choosing the smaller s,

Intersection point **I** = (8, 16, -6)

c) Since the shape is a sphere, outward normal $\mathbf{N} = |\mathbf{I} - \mathbf{C}|$

where **C** is the center of the sphere.

$\mathbf{N} = |(8,16,-6) - (10,24,-10)| = (-0.21821789,\ -0.87287156,\ 0.43643578)$

$\mathbf{L} = (0,0,1)$

$\mathbf{R} = (2\mathbf{N} \cdot \mathbf{L})\mathbf{N} - \mathbf{L}$
= (0.87287156)(-0.21821789, -0.87287156, 0.43643578) – (0, 0, 1)

= (-0.19047619, -0.761904761, -0.619047619) [Check: **R** is a unit vector]

The viewer direction is the vector from the intersection point to **PRP**, which is

$\mathbf{V} = |(0,0,2) - (8,16,-6)|$ = (-0.40824829, -0.81649658, 0.40824829)

Intensity of pixel

$= k_s I_l (\mathbf{V} \cdot \mathbf{R})^{n_s} = (\mathbf{V} \cdot \mathbf{R})^2 = (0.447129078)^2 = 0.199924412$

Qn 3

```
GLfloat  light1PosType [ ] = {0, 200, 200, 1.0};
                    :
GLfloat  M[16];
for (i=0; i<16; i++)
    M[i] = 0;

M[0]=M[5]=M[10]=1;
M[11]=-1.0/200;                 //  caution: 1/200 will return 0

 <C code>                       // draw the object

glPushMatrix ( );               //  save state

glMatrixMode (GL_MODELVIEW);
glTranslatef (0, 200, 200);     //  M_wc←s
```

```
glMultMatrixf (M);          //   perspective projection
glTranslatef (0, -200, -200);   //   M_{s←wc}

glColor3fv (shadowcolour);

< C code>                    // draw the shadow

glPopMatrix ( );             //   restore state
```

Qn 4

Change the light source to a light direction:

GLfloat light1PosType [] = {1, 1, 1, 0};

Modify the perspective projection to a parallel projection with direction of projection (1, 1, 1).

$(x, y, 0) = (X, Y, Z) + t(1, 1, 1) \Rightarrow t = -Z$

The homogeneous transformation is

$$\begin{pmatrix} x \\ y \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

The code is modified to

GLfloat M[16] = {1, 0, 0, 0, 0, 1, 0, 0, -1, -1, 0, 0, 0, 0, 0, 1};

The program is

```
GLfloat  light1PosType [ ] = {1, 1, 1, 0};
                    :
GLfloat  M[16] = {1, 0, 0, 0, 0, 1, 0, 0, -1, -1, 0, 0, 0, 0, 0, 1};

< C code >                   // draw the objects

glPushMatrix ( );            //  save state

glMatrixMode (GL_MODELVIEW);
glMultMatrixf (M);           //   parallel projection

glColor3fv (shadowcolour);

< C code >                    // draw the shadow
```

```
glPopMatrix ( );                    //  restore state
```