Part A

Question 1

We need to use the equation

Id = Kd * Ld * s.m

Where Kd are the diffuse co-efficents of the material (0.4,1,0) and Ld are the diffuse co-efficients of the light source (0.9,0,0.2).

S is the normalized vector TO the light source so is

s = (0,1,4) – (0,1,2) = (0,0,2)

Normalised we get:

s = (0,0,1)

m is the normal which we also need to normalize so

m = (0,4/sqrt(41),5/sqrt(41)) = (0,0.625,0.781)

Id = (0.4,1,0) * (0.9,0,0.2) * (0,0,1) . (0,0.625,0.781)

= (0.4,1,0) * (0.9,0,0.2) * 0.781

= (0.281,0,0)

Question 2

a) We need to provide the up vector.

b) Suppose we provide an up vector of (0,1,0).

```
gl.glMatrixMode(GL2.GL_MODELVIEW);
gl.glLoadIdentity();
GLU glu = new GLU();
glu.gluLookAt(3,2,1,1,0,-1,0,1,0);
```

c)

First we need to find our **k** vector.
This will be the eye vector – the look vector
k = (3,2,1) – (1,0,-1) = (2,2,2)
i = up x **k**
= (0,1,0) x (2,2,2)
= (2,0,-2)
j = k x **i**
= (2,2,2) x (2,0,-2)
= (-4,8,-4)

We need to normalise these

i = (2/sqrt(8),0,-2/sqrt(8))
  = (1/sqrt(2),0,-1(sqrt(2))
j = (-4/sqrt(96),8/sqrt(96),-4/sqrt(96))
  = (-1/sqrt(6),2/sqrt(6),-1/sqrt(6))
k = (2/sqrt(12),2/sqrt(12),2/sqrt(12))
  = (1/sqrt(3),1/sqrt(3),1/sqrt(3))

d) Putting these directly in a matrix would give us the inverse matrix of what we need. Eg

```
[1/sqrt(2), -1/sqrt(6), 1/sqrt(3), 3]
[    0,      2/sqrt(6), 1/sqrt(3), 2]
[-1/sqrt(2), -1/sqrt(6),1/sqrt(3), 1]
[    0,       0,            0,     1]
```

We could get the inverse of that. Or it may be easier to think of the matrix above as the result of

T * R where

```
T = [1 0 0 3]
    [0 1 0 2]
    [0 0 1 1]
    [0 0 0 1]
```

And R is

```
[1/sqrt(2), -1/sqrt(6), 1/sqrt(3), 0]
[    0,      2/sqrt(6), 1/sqrt(3), 0]
[-1/sqrt(2), -1/sqrt(6),1/sqrt(3), 0]
[    0,       0,            0,     1]
```

Then we need to find the inverse which would be

$R^{-1} * T^{-1}$

The inverse of a rotation matrix is just the transpose.

```
R-1 =  [1/sqrt(2) ,    0 ,     -1/sqrt(2), 0]
       [-1/sqrt(6), 2/sqrt(6), -1/sqrt(6), 0]
       [1/sqrt(3) ,  1/sqrt(3), 1/sqrt(3) , 0]
       [    0,         0,           0,     1]
```

```
T-1 = [1 0 0 -3]
      [0 1 0 -2]
      [0 0 1 -1]
      [0 0 0 1]
```

Which gives us

```
[1/sqrt(2) ,    0 ,     -1/sqrt(2), -2/sqrt(2)]
[-1/sqrt(6), 2/sqrt(6), -1/sqrt(6),        0  ]
[1/sqrt(3),  1/sqrt(3), 1/sqrt(3),  -6/sqrt(3)]
[    0,         0,           0,            1  ]
```

```
e) Multiplying the matrix above by
[-1]   = [-4.242]
[1 ]     [ 0    ]
[3]      [-1.733]
[1]      [ 1    ]
```

Part B

These are just possible answers I quickly wrote. Yours may be different yet still correct or may be much better ☺

Question 3

BSP trees are data structures that are used to partition objects within a space. BSP trees can be used for hidden surface removal and for ray tracing. When used for hidden surface removal a tree is created and traversed in its entirety with the goal of rendering the scene in back to front order with respect to the camera. When used in ray tracing, the whole point is that we do not have to traverse the whole tree and that we can just visit leaves that contain the parts of the world the ray we are tracing actually passes through.

Question 4

A fragment shader is a piece of code that runs on the GPU and operates on every fragment that is produced by the rasterisation process. The fragment shader must compute the color of the fragment. Each fragment shader will have a corresponding vertex shader. In OpenGL shaders are created by passing in the shader code as a string and are compiled and linked together at runtime.

Question 5

Trilinear filtering is a form of texture filtering. It performs a texture lookup and bilinear filtering on the two closest mipmap levels and then linearly interpolates the results. This helps to prevent abrupt changes in quality at boundaries where one mip-map level is switched to the next.

Part C

Question 6

I would model the shape of the teapot using Bezier patches. Alternatively, the base and lid could be modeled as surfaces of revolution. The spout could possibly be modeled using extrusion.(The standard teapot is actually modeled using Bezier patches). Assuming we want the teapot to be a metallic surface we would model it as being highly specular. We could possibly use environment mapping to get reflections on the teapot. Since we are modeling this for a real-time game, we probably would not use ray-tracing techniques that would make the teapot look more realistic, as we would be concerned about efficiency.

Question 7

I would use the global illumination approach called radiosity. The advantages of the approach is that it models indirect diffuse lighting and as a result can render soft shadows and realistic diffuse lighting .The disadvantages of radiosity is that it does not handle translucent, transparent or specular surfaces in the calculations (although there is work that combines radiosity with raytracing). It is calculation intensive and too slow for real-time applications. It can be used in real-time applications to pre-compute lighting for portions of geometry that are static and rendered offline into textures. However pre-computed lighting is not suitable for dynamic lights of moving objects.