Interactive Computer Graphics Mid-term Exam, May 3, 2016

1. BSP Tree (15%)
   A. Construct the Binary Space Partitioning (BSP) tree of the model in Fig.1 below. Please use the node "a" as the root. Split any line segment as you wish, and mark them as b.1, b.2, b.3, b.4 etc. Please choose smaller numbers/alphabets as the sub-tree root node.
   B. From the BSP tree in (a), derive the display sequence in terms of the given viewing position in this figure.

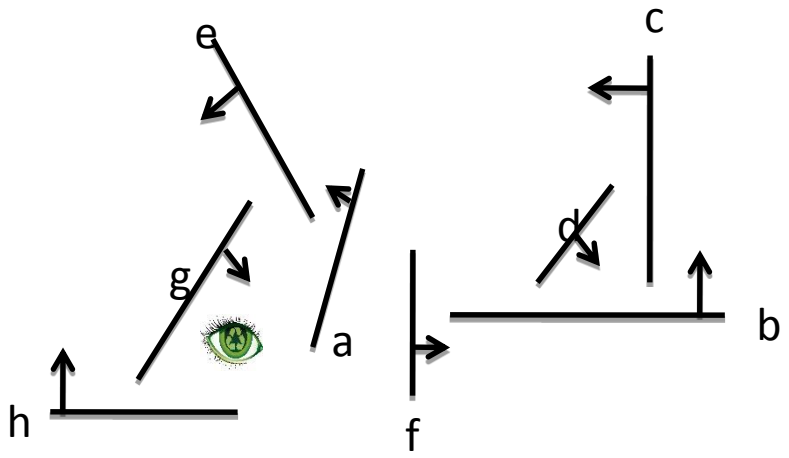


Figure 1.

2. (10%) When using the Cohen-Sutherland line clipping algorithm, how do we check the outcodes to see if a line can be trivially accepted or rejected?

Each coordinate of a line is assigned a 4-bit outcode according to its location relative to the left, right, top, and bottom screen edges.
First bit set to 0 if: ycoord < yscreen-min
Second bit set to 0 if: ycoord > yscreen-max
Third bit set to 0 if: xcoord < xscreen-min
Fourth bit set to 0 if: xcoord > xscreen-max
If both of a line's outcodes are 0000 (logical OR), trivially accept. If a logical AND of the lines outcodes is not zero, trivially reject.

3. (3 %) (OpenGL) What does `glFlush` do?
4. (4 %) (OpenGL) Consider this sequence of calls:

```
glColor3f(1.,1.,1.);
glColor3f(0.,1.,0.);
glVertex3f(1.,1.,1.);
glVertex3f(2.,2.,2.);
```

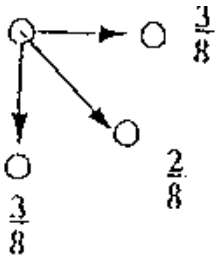
- What color is the vertex (1, 1, 1)?
- What color is the vertex (2, 2, 2)?

5. (5%) (OpenGL) Suppose that you have 1000 triangles that can be arranged into a triangle strip. How many vertices would you have to specify to OpenGL if you use the fact that they can be arranged into a triangle strip?
6. (5%) (OpenGL) What would look different in the resulting image if you changed the *1.0* to *-1.0* in the following line.

```
glm::lookAt(eye, center, vec3(0 , 1.0, 0))
```

7. (10%) The following is the pseudocode of Floyd–Steinberg error diffusion dithering

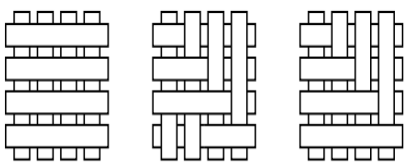
```
for each y from top to bottom
 for each x from left to right
     oldpixel = pixel[x][y]
     newpixel = floor(oldpixel / quant_number) * quant_number
     pixel[x][y] = newpixel
     quant_error = oldpixel - newpixel
     pixel[x+1][y] := pixel[x+1][y] + quant_error * 3/8
     pixel[x][y+1] := pixel[x ][y+1] + quant_error * 3/8
     pixel[x+1][y+1] := pixel[x+1][y+1] + quant_error * 2/8
```



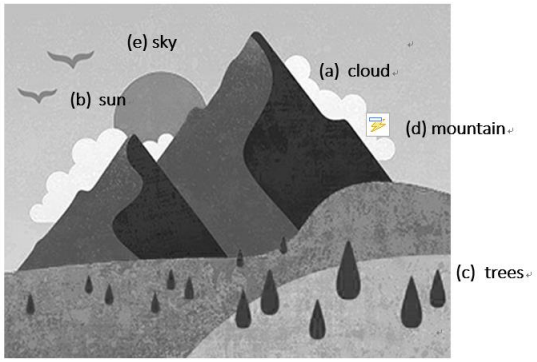
Approximate such a matrix using Floyd-Steinberg error diffusion if we have elements of the form 2xN only (i.e. 0,2,4,6,8,10, etc), give the result of the 2x2 upper left matrix.

$$\begin{bmatrix} 3 & 5 & * \\ 7 & 1 & * \\ * & * & * \end{bmatrix}$$

8. (10%) Painter's Algorithm
   (1) (5%) Which of the following scenes would have troubles in the Painter's Algorithm? Explain your answer briefly. (Each rectangle is a single primitive.)



(2) (5%) Determine the painting order in Painter's Algorithm.



9. (20%) There are nice properties about Ray Tracing.

A. (7%) Please describe the strength and weakness of Ray Tracing, using the theory of "The Rendering Equation"

$$L_o(\mathbf{x}, \omega_o, \lambda, t) = L_e(\mathbf{x}, \omega_o, \lambda, t) + \int_\Omega f_r(\mathbf{x}, \omega_i, \omega_o, \lambda, t)\, L_i(\mathbf{x}, \omega_i, \lambda, t)\, (\omega_i \cdot \mathbf{n})\, \mathrm{d}\,\omega_i$$

B. (5%) Using the rendering equation, please explain the weakness of Phong shading.

C. (8%) Modified Distribution Ray tracing. In the photo below, there are very bright spots under the glass sphere, called caustics. One way to improve the previous ray-tracing is to combine the rays (light packets) shooting from the lights, and store the (location, direction) information at the light-surface intersection points, called photon maps.

For caustics, we only store the photon map when rays hit the highly specular surface or pass through a transparent object and finally reach a diffuse surface. After the photon maps are created, we can use ray-tracing to shoot rays from the eye until it hits the surfaces with the photon maps. Why is this method successful in solving the Rendering Equation? Please give your own algorithm describing the previous solution.

10. (5 %) What is your term project for this semester? What are the technical

difficulties involved in the project? (You can refer to the project listing).

11. (13%) Illumination Models (20%)

A popular illumination model is called Phong model, which is developed in 1975. The formula of this model for each color channel is (please refer to textbook):

$$I = I_a k_a O_d + f_{att} I_P [k_d O_d (\overline{N} \cdot \overline{L}) + k_s (\overline{R} \cdot \overline{V})^n] .$$

where L is light direction, V is viewing direction.

(a) (4%) Could you give two examples of materials that <u>definitely can not be approximated</u> by the Phong model? And simply explain why.

(b) (4%) A general depiction of illumination model is called bidirectional reflectance distribution function (BRDF) (bidirectional reflectivity):

$$I = f_{BRDF}(L,V) .$$

The idea of this function is that given a viewing and lighting (single light source) direction, the BRDF function returns its reflectance (or precisely, the ratio between the incident flux and reflected intensity). Please design a platform to capture BRDF of materials. You will have the following equipments:

1. Spheres of different materials
2. A digital camera
3. A light source with a 1DOF rotational arm that allows you to move the light direction horizontally.

(c) (5%) The function *I*, if stored as a database, can be quite big (much larger than 1.0 Gaga bytes). This can be a problem for ordinary PC graphics card in real-time graphics. Can you think of a way to compress the data of function I? (Assume viewing and lighting direction is in division of one degree, say in spherical coordinate system, per sample in theta and phi direction.).