Computer Graphics Final Exam (2017 Spring Term)

1. Fill blanks (2 pts each, 10 pts in total)

1) Under linear perspective projection, straight lines always appear as _straight line_.

2) When two curve segments join at a point and both curves approach that point with the same derivative, the joining is said to be _parametric _ continuous.

3) _parallel projection_ is a special case of perspective projection where the viewer is infinitely far away.

4) In the context of a scan-line renderer, Z-buffers are used for _hidden surfaces removal_ .

5) _quaternions_ encode 3D rotations as point in 4D space.

2. Data Structure (8 pts)

Give the names and brief explanations of four of the most common spatial partition data structures within computer graphics.

octree: 利用三个相互垂直平面将三维立体空间一分为八个子空间，判断每个子空间是否需要继续划分，如果需要，则利用同样的规则进一步细分，直到每个子空间不再需要细分。

bsp tree: 空间二分树：选择一个平面将空间分为平面的左、右及平面上三部分，对平面左或右两个子空间，如需要划分则按同样规则进行，直到每个子空间不再需要划分。

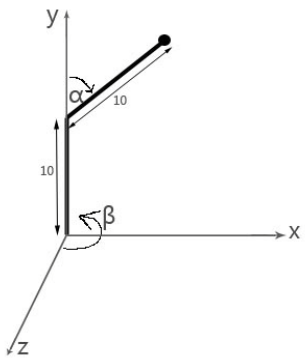
quad tree: 利用两相互垂直直线将二维平面空间一分为四个子空间，判断每个子空间是否需要继续划分，如果需要，则利用同样的规则进一步细分，直到每个子空间不再需要细分。

scene graph: 表达3D场景的一个层次图结构，图的每个顶点可以是一个几何物体、也可以表达一个变换、一个光源，图的边表达两个顶点间的定义关系。

3. Robotic Arm (12 pts)

Consider the robotic arm given in the diagram. It has two parts, both 10 units in length. The bottom part can rotate by β around the y axis (counter clockwise) and

the top part can move α away from the y axis. We are interested in the final position of the arm, marked by point •.



Suppose robotic arm takes the following commands:
reset () - brings the arm initial position where α = 0 , β=0 and p = (0,20,0);

move(α, β) - moves the arm down from y-axis α, and rotates around y-axis β.

1) Please write two code fragments using OpenGL to implement two commands above, to achieve the end point of the robot arm in a right position. (8 pts)

A: suppose the end point of robot arm, we called p, has (xp, yp, zp) as its current coordinates, then β = arcsin(z/10); α=arccos((y-10)/10).

reset:

glMatrixMode(GL_MODELVIEW);

glPushMatrix();

glRotate(arccos((y-10)/10)*180.0/3.14, 0, 0, 1);

glRotate (arcsin(z/10)*180.0/3.14, 0,1,0).

glPopMatrix();

move:

glMatrixMode(GL_MODELVIEW);

glPushMatrix();

glRotate(α, 0, 0, 1);

glRotate (β, 0,1,0).

glPopMatrix();

2) Give out the accessible range of the robotic arm and explain why. (4 pts)

The accessible range of the robotic arm is a sphere with its center at the point(0, 10, 0). Since the end of the robotic arm has two freedoms rotating around y and z axis, and the length of the second segment of the arm are invariable. That's apparently a sphere.

4. Texture mapping (20 pts)

Texture mapping is used to make simple polygon models look more realistic without having to create millions of polygons with different colors. Assume that you are given the following image of the diet coke logo, and your task is to texture map this image onto a cylinder, so you can have diet coke cans in your graphics scene.

a) How would you break your cylinder model into N polygons? Draw a diagram that shows a side view of your cylinder to illustrate your polygon model. (5 pts)

A:

b) How would you map the texture onto these polygons? Be specific about how you would assign polygon vertices with texture coordinates. Write a fragment of source code to specify this mapping in a typical OpenGL application. (8 pts)

A: We could use Texture mapping method to map the texture onto these polygongs. Suppose the center of cylinder bottom disc is the origin point, the radius of the cylinder is r, the height of the cylinder is h, and the texture is a 512*256 image. Then,

GLubyte my_texels[512][256];

glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, 512, 256, 0, GL_RGB, GL_UNSIGHED_BYTE, my_texels);

glEnable(GL_TEXTURE_2D);

```
glBegin(GL_QUAD_STRIP);
for (i=0; i<=N; i++){
glTexCoord2f(i/N, 0);
glVertex3f(r*cos(i*2*3.14/N), r*sin(i*2*3.14/N), h);
glTexCoord2f(i/N, 1);
glVertex3f(r*cos(i*2*3.14/N), r*sin(i*2*3.14/N), h);
}
glEnd();
```

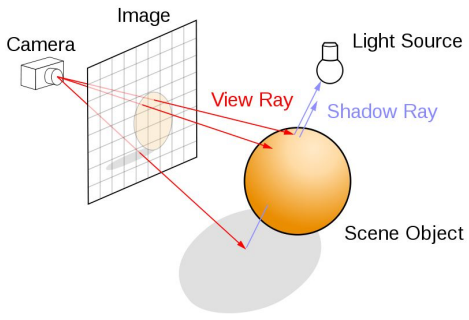
c) What happens if the screen area (a polygon) of the projected 3D object is larger than your texture map image? How are the pixel color values calculated for the polygon? (7 pts)

A: The texture will be scaled up to fit the projected region. That means one pixel of the original texture will become to more than one pixels on the screen. To avoid the aliasing effect, the color of one pixel 'p' should be the averaged color of a region in the texture image, the center of the region in texture image is the corresponding pixel of 'p'.

5. Ray tracing (20 pts)

Ray tracing was invented in the early 1980s to create realistic images, and now they are widely used in computer graphics rendering. Assume that you have a scene made up of N spheres with different sizes, locations, colors and reflection properties.

1) Draw a small diagram that shows the focus point of the camera, the imaging plane, and one ray being traced into the scene that hits a sphere. (2 pts)



A: The diagram is like above, may have no shadow ray.

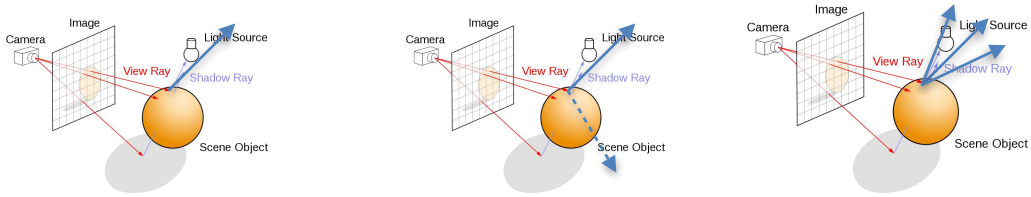
2) If we have L light sources in the scene, how do we decide how much light from each light source is directly illuminating the sphere where the ray intersected? (6 pts)

A: we could use phong illumination model to compute the light intensity that illuminating the hitting point of the sphere. It comprises to 2 kind of light. One is the diffuse light and the other are specular light. The computations of these two lights are as follows:

$I_d = k_d * L * cos(L \cdot n))$, in which k_d is the diffuse reflection coefficient, n is the normal of the hitting point.

$I_s = K_s * L * cos(L \cdot v)^{\alpha}$ in which K_s is the specular reflection coefficient, v is the view direction from the view point to the hitting point, α is the shining coefficient to adjust the shiness strength of various materials.

3) Draw a diagram that shows what happens to your ray when it intersects a sphere made of shiny metal, clear glass, or coarse wood, draw a diagram for each for the three situations. How do we respectively decide what the pixel color should be for this ray in such three different situations? (12 pts)



A: The above three diagrams from left to right show the three situations that the sphere is made of shiny metal, clear glass, and coarse wood respectively. That is, when the sphere is shiny metal, only the light from ideal reflection direction could go into the camera, when the sphere is clear glass, the light from ideal reflection and refraction could go into the camera, while the sphere is coarse wood, the diffused reflection lights will go into the camera.

6. Intersection Test (30 pts)

1) Describe an algorithm for computing the intersection between a ray $l(t)=a+tb$, and a sphere $||c-x||-r = 0$, in which c and r are the center and radius of the sphere respectively. (10 pts)

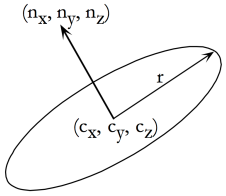
A: 1) substitute the equation of the ray into the sphere equation, we get

||c-(a+tb)||-r = 0;

2) solve the above equation, in which t is variable, c is a point, a, b, r are real number, we could get a quadratic equation.

3) solve the quadratic equation, we could get either two real root, or one double root, or no real root, corresponding two intersection points, one tangent point or no intersection point. The coordinate of the intersection points or tangent point could be computed by the value of the corresponding roots.

2) Consider a circular disc which is defined by the coordinate of it's center, a direction normal to the disk, and a radius. All points **x** on the disk satisfy a plane equation, as well as a distance constraint.



a) Write down the plane equation and the distance constraint. (5 pts)

A: plane equation is $(x - c_x) \cdot n_x + (y - c_y) \cdot n_y + (z - c_z) \cdot n_z = 0$, in which (x,y,z) is the coordinate of point **x** on the disk.

Distance constraint is: $(x - c_x)^2 + (y - c_y)^2 + (z - c_z)^2 \leq r^2$.

b) Suppose you are given the following classes, write a code fragment for the intersect( ) method of the Disc object. The method should return true in the case of an intersection and update the relevant fields of the Ray object that it is passed. Otherwise, it should return false. Also, assume that a ray can intersect the disc from either side. (10 pts)

```
class Vector3D {
        public float x, y, z;
// constructors
public Vector3D( );
public Vector3D(float x, float y, float z);
// methods
public static float dot(Vector3D A, Vector3D B);
public static Vector3D cross(Vector3D A, Vector3D B);
public static Vector3D normalize(Vector3D A);
public static Vector3D scale(Vector3D A, float s);
public static Vector3D plus(Vector3D A, Vector3D B);
public static Vector3D minus(Vector3D A, Vector3D B);
}
```

```
class Ray {
public static final float MAX_T = Float.MAX_VALUE;
public Vector3D origin;
public Vector3D direction;
public float t;
public Renderable object;
}
class Disc implements Renderable { Surface surface;
Vector3D center;
Vector3D normal;
 float radius;
public Disc(Surface s, Vector3D c, Vector3D n, float r) {
surface = s; center = c; normal = Vector3D.normalize(n); radius = r; }
public boolean intersect(Ray ray) { // returns false if ray does not intersect
//write your code here
}
}
```

A: Algorithm 1: check distance constraint first

```
public boolean intersect(Ray ray) {
// check distance constraint first
Vector3D x = center.minus(ray.origin);
float t = Vector3D.dot(x, ray.direction);
if (Vector3D.dot(x, x) - t*t >= radius*radius)
return false;
// find intersection with plane
t = Vector3D.dot(normal, ray.direction);
if (t == 0)
return false; // ray is parallel to disc
t = Vector3D.dot(normal, x) / t;
if ((t < 0) || (t > ray.t))
return false;
x = x.minus(ray.direction.scale(t));
if (Vector3D.dot(x, x) >= radius*radius)
return false;
ray.t = t;
ray.object = this;
return true;
}
```

Algorithm 2: compute the intersection first

```
public boolean intersect(Ray ray){
// compute intersection with plane
float t = Vector3D.dot(normal, ray.direction);
if (t == 0)
return false; // ray is parallel to disc
Vector3D x = center.minus(ray.origin);
t = Vector3D.dot(normal, x) / t;
if ((t < 0) || (t > ray.t))
return false;
// then compute distance constraint
```

```
x = x.minus(ray.direction.scale(t));
if (Vector3D.dot(x, x) >= radius*radius)
return false;
ray.t = t;
ray.object = this;
return true;
}
```

c) When you were implementing the disc intersection test, you had the choice of testing for intersection with the infinite plane, or testing the distance constraint first. Discuss the relative merits of each approach. (5 pts)

A: As I showed above, there are two alternative solution when computing the intersections. One is check the distance constraint first, the other is computing the intersection points with the infinite plane first. For the first solution, the advantage is when so many rays are not intersected with the sphere, when doing distance constraint, the algorithm will return back quickly. And for the second solution, the advantage is that the algorithm is clear and easy reading, moreover, if some rays are parallel to the plane, it will return back very early.