Ray Tracing Graphics Offline tips and common issues

Prerequisite: Read the 17 batch's specification first, then ours.

Changes between the two specifications:

1) We need to handle pyramid and cube objects.
2) We need to make an infinite floor
3) We need to use the falloff parameter in color computation
4) All the light sources have color white and the shininess of the floor is 0

I've followed Zarif vai's github to get the basic flow of the offline. So I'll give reference according to that.

Camera Setup:

```
500     pos.x = 0; pos.y = -160, pos.z = 60;
501     l.x = 0; l.y = 1; l.z = 0;
502     r.x = 1; r.y = 0; r.z = 0;
503     u.x = 0; u.y = 0; u.z = 1;
```

This is the approximate camera position that I've followed. You can adjust it as per your need.

```
gluPerspective(fovY, aspectRatio, nearDistance, farDistance);
//fovY,aspect,zNear,zFar
```

You need to set this in initGL. The values will come from the input file

Camera Control:

Same as our offline 1 task 3

Incremental Approach to progress the offline:

Manipulating Inputs:

In your objects vector, the objects should be: triangles, floor and spheres. That is after taking input, there will be no existence of pyramid or cube, everything will be kept as triangles only. You need to write the basic classes: object, floor, triangle, sphere before this.

Drawing the infinite floor: must be drawn in XY plane

Take the camera position as reference point initially (or any other point near the camera position)

Choose how many tiles you want to draw (The more you draw, the slower it will be. You can choose between 50* 50 to 100*100 tiles)

Now the tricky part, when we are moving along -ve X axis, we need to increase the tiles of X axis in negative direction. We can do this by shifting the starting point of drawing tiles towards the -ve X axis.

As we have alternating white and black color, so we need to shift 2 tiles so that our eyes cannot catch the redrawing effect.

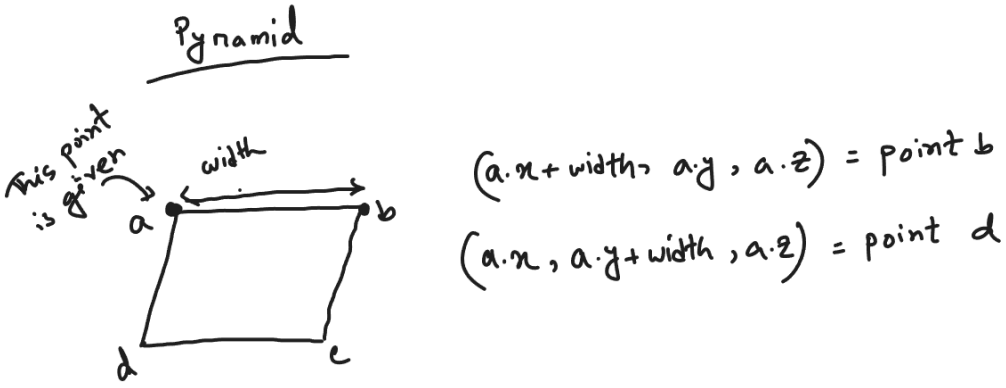
So if you have moved two tiles left, then shift the starting point two tiles left, else keep the starting point as previous.

Same goes for right, front and back.

Drawing Sphere: Same as our offline 1

Drawing Pyramid: You can do it by converting the pyramid surfaces to 4 triangles and 1 square or 6 triangles all together (converting the lower square surface to two triangles)

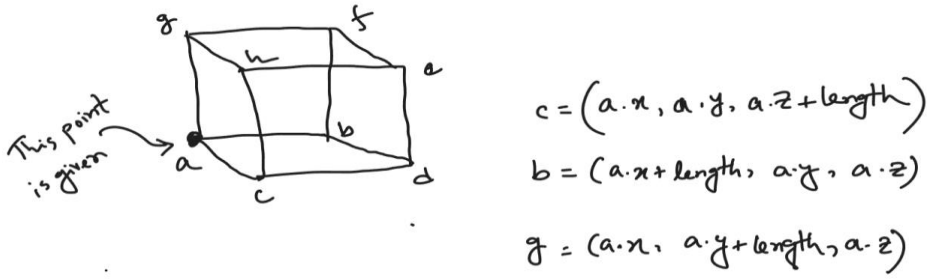
The input is lowest point of the pyramid and width and height of the pyramid. A demo of how you can find other points is shown below.



Now the intersection of ac and bd will be the center of the abcd surface. If you add the height to the z-coordinate of the center of abcd, then you'll get the topmost point of the pyramid.

Now draw triangles using the points and it will be done.

Drawing Cube: The lowest left point is given. The length of the cube's each side is given.



Milestone 1

After writing the basic Object class and others for floor, sphere, cube and pyramid, check the draw() method. If everything is drawn appropriately and if the infinite floor is working.

If everything is good, we are ready to go for next step: **Intersection**

Intersections and Normals

Write code for finding normal and intersection point for sphere, floor, pyramid(basically triangle) and cube (Basically triangle).

You can take help from zarif vai's code or our theory slides or schaum's book. Zarif vai has followed the theory slides and schaum's book in parallel.

Now write the capture method according to 17 batch's specs or our specs, the one you understand.

I've followed the 17 batch's specs for this. Now check if your intersections are working properly or not. To do this, return the color of the object here in zarif vai's code.

```
186         virtual Ray getNormal(PT point, Ray incidentRay) = 0;
187         virtual double intersect(Ray ray, Color &color, int level)
188         {
189             double t = intersectHelper(ray, color, level);
190 
191             if(t < 0) return -1;
192             if(level == 0) return t;
193 
194             // find intersection point and it's color
195             PT intersectionPoint = ray.origin + ray.dir*t;
196             Color colorAtIntersection = getColorAt(intersectionPoint);
197 
```

Get the colorAtIntersection after calculating t. Then if the level is 0, then return set the color to colorAtIntersection.

Milestone 2

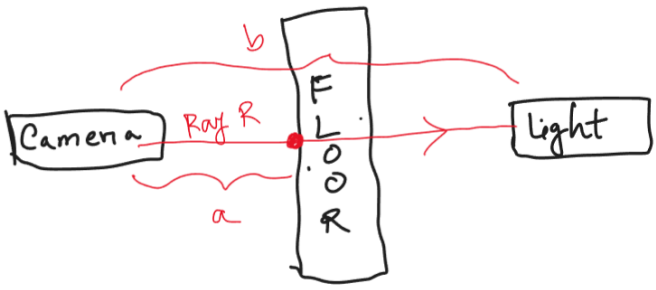
Run your code, if the what you see after running the exe file and what you get after generating the bmp image, then your intersections and normals are working properly.

Calculating Colors

You can follow zarif vai's color calculation but we need to add the falloff effect . See the formula of our specs and multiply the scaling factor with lambert and phong value.

Some issues that are not handled in zarif vai's code:

1) If the light sources are above the floor and your camera is below the floor, then only ambient color will be seen. No diffuse or specular color. For this you need to check if the light sources and camera are to the opposite sides of the floor or not. If they are in opposite, their color calculation must be ignored. A simple concept which I have followed is given here. You may use your own ideas.



if Ray R intersects floor and a < b, color calculation of that light must be ignored.

2) If the camera is inside an object, then the ambient color of that object should be shown. No other objects color should come here. You need to check if your camera is inside an object.

Milestone 3

Now do the above things for point lights at first. If everything goes fine, do the same for spot lights. Then do the recursive reflection.

Some issues may arise according to different bugs in your code:

1) If you see that the diffuse or specular light effects are not as expected or after calculating reflections, there are black spots or the shades are blackish, you may have errors in normal or getting intersection points. Debug those areas. If the issues are not fixed, then check for the color calculations again.