Introduction/Background, Implementation, Challenges, Conclusion, References

The above has to be provided according to Brandon's Report specifications

Introduction/Background:

Note: please don't say its to build a raytracer, probably every team who chose raytracer will say that

Rephrase something like we had many ideas: provide samples, and may be say we wanted to see how to create a realistic and pretty image and hence we chose raytracer

Or say something like how raytracing and raycasting is the concept behind all powerful renderers which produce photo realistic images or some modified version of that

Also include a gist of what projects we came up with and why we decided to do this particular one

Implementation: Just an overview feel free to enhance it with more details and images References included in the description

1. Basic frame work
    - Display class
        - save tga/ppm images to disk
    - Camera
        - from given lookat,up(0,1,0) and position calculate X,Y,Z,
    - Material
        - class for specifying color, diffuse, spec, ambient coefficients, reflection,refractive index
        - also extra parameter for texture files
    - Geometry
        - super class from which is inherited by all implicit and parsed objects
    - Vector operations
        - dot product, cross product, normalize, magnitude calculation
    - Color check
        - checks rgb values to be between 0-1

2. Ray from camera to object(**ray direction calculation**)

    http://www.macwright.org/literate-raytracer/

    Send a ray from from the camera for every pixel(center)
    Convert the raw pixel value to NDC space -1to1 based on aspect ratio and 1/d(tan(FOV/2)) (for including perspective correction, this gives us the distance of camera from the image plane)
    you can add the image from class for 1/d calculation
    Transform each ray from the camera such that it fits in the NDC space(that's done by scaling the X and Y vector of camera based on the above values), finally the ray from camera in CamZ direction is shifted horizontally and vertically by necessary amount which you compute by scaling X and Y.
    ray origin is just the camera position

```
horizOffset     = camera->camX.scalarMult(horizScaleCeoff);
vertOffset = camera->camY.scalarMult(vertScaleCoeff);
primaryRay.origin = camera->position;
primaryRay.direction = (camera->camZ.addVector(horizOffset)).addVector(vertOffset);
primaryRay.direction = primaryRay.direction.normalize();
```

Use that ray to find intersection
Send the ray to calculate object intersection, once you get the intersection value for all objects that fall on the path of the ray, compare the values and object that wins is the one with lowest intersection value.
Means, thats the object that will be visible in the final image, for which we will be doing shading calculations(Note: intersection value has to be > 0.0000001)

3. Shading
   - Diffuse & Shadow
     For given lights, loop through the objects:
     Check if NdotL > 0 , means light hits the object
     Send a new Ray with the Current Object Intersection point as Origin and Light Direction as direction(`Ray shadowRay = Ray(currectObjectIntersection,L);`)
     For the above shadowRay send it to calculate the intersection with every object(other that the object for which lighting is being calculated), set a flag shadow = true
     So we calculate color for a point on the object only if shadow is false(ie the light ray hitting a object is not intersected by any other object.

     Calculation of the diffuse color is done by adding Ka,Kd * NdotL * LightColor
   - Specular
     Calculate Specular(reflected ray: specular is nothing but reflection of light)
     origin: currentOnjectIntersection vector
     position: Here we take light direction, since we are sending back as -L
     specular ray direction: 2(NDotL)*N – L (Note: normalize the vector)
     The above specular ray is R.
     If RdotL > 0 : calculate specular as Ks*LightColor*(RdotL^spec coeff)
     add the calculated specular color to the above diffuse value.

4. Sphere and Plane using implicit equations
   Reference: Andrew Glassner Intro to ray tracing
   Rorigin = R0 = [X0,Y0,Z0]
   Rdirection = Rd = [Xd,Yd,Zd] normalized vector
   X = X0 + Xd*t,Y = Y0 + Yd*t,Z = Z0 + Zd*t (t is the intersection value)
   Sphere:
   equation: (Xs-Xc)^2+(Ys-Yc)^2+(Zs-Zc)^2 = radius^2
   Substituting X,Y,Z we get an equation of the form At^2+Bt+C=0
   A = Xd^2+Yd^2+Zd^2 = 1(unit vector)
   B = 2 * (Xd*(X0-Xc)+Yd*(Y0-Yc)+Zd*(Z0-Zc))
   C = (X0-Xc)^2+(Y0-Yc)^2+(Z0-Zc)^2 – radius^2
   Find roots solve for t, if the determinant is negative(intersection is negative) hence ray misses the object, or the object is behind camera.
   From there calculate intersection point and normal
   Ri = [X,Y,Z](the variables are defined above)
   Rn = [(X-Xc)/Radius,(Z-Zc)/Radius,(Z-Zc)/Radius]
   Plane:
   Ax+By+Cz+D = 0
   Given plane normal Pn vector and distance D(magnitude) from origin
   t(intersection value) = -(Pn.R0+D)/Pn.Rd
   intersection point: Ri = [X,Y,Z]

5. trackRay() function: seperate section because this is the most important recursive function of the raytracer which does primary intersection, reflection and refraction calculations:
This function is recursively called until INTERACTION_DEPTH is reached(depth for reflection and refraction calculations)
    1. Calculates color by calculating intersection for primary ray and finding the shading value from getColor()
    2. Reflection/Refraction calls this function recursively till it reaches depth limit

6. Reflection:
http://www.cs.jhu.edu/~cohen/RendTech99/Lectures/Ray_Tracing.bw.pdf
`R = (-I.N)N + I + (-I.N)N = I - 2(I.N)N :` **`Include the diagram from website`**
If currentDepth < `INTERACTION_DEPTH` and if reflection is enabled:
calculate reflected ray:
reflected ray origin: objectIntersection point
previous ray(can be primary or another reflected ray): Rp
normal of object calling reflection N
reflected ray direction: Rp.direction – 2(N.Rp)*N(normalize the direction)
Send the ray to calculate object intersection, getcolor, multiply with reflective amount for the object, add it to the current object color.

7. Refraction:
http://graphics.stanford.edu/courses/cs148-10-summer/docs/2006--degreve--reflection_refraction.pdf

8. Interpolating multiple objects: For every object in the path of a ray, find the object with lowest intersection value, and higher than threshold value, return the intersection value, intersection position and normal of that object.
9. Obj Parser
    - Octree optimization
10. Normal and s-t interpolation for parsed objects
11. Texturing
    - Procedural textures
      Checker Board: Find x, y intersection of plane add it.
      If value %2 =0 then put first color, else use a different color
    - Mapping image files
12. Antialiasing
Instead of sending a single ray per pixel, send multiple rays and average the color based on some weight, will increase render time, but will add more detail hence reducing aliasing articafts
13. Scene Setup
    - Testing various camera positions
    - Implementing object transformation
    - testing multiple obj files back and forth between maya and our raytracer
    - fixing the position of the objects and deciding the look and feel for the final render.

Challenges:

Biggest challenge:
Ray tracing objects from obj files:

The first time we rendered a teapot it took 45 minutes to get a 640X480 image, we realized it was unusable, so we calculated bounding box, implemented Octree optimization on top of it and got a significant improvement to almost 3-4 minutes.

Getting the first raytracer build:

All the features of raytracer were incremental, like shading was the base of all, then ray intersection, reflection then refraction(its multilayered, and not independent)

Getting the first built took a lot of time.

Even then,it needed a lot of back and forth, and we had to keep changing equations to perfect them as we moved on, every additional layer revealed a new issue we were not aware of.

TGA files and visual studio compiler issue

Modularizing code:

Since so many parts of the ray tracer are reused in multiple calculations, we had to make sure the code is modularized, and hence easy to debug.

The main thing that saved us in this project was building a git repo, and that helped every one to have the latest version of code to build on top of each other.

Lower priority:
Reflection:

Reflection ray calculation was a challenge to figure out(what to use as origin and what to use as direction, took multiple shots to figure it out.

Shading calculation:

We tried like how it was taught in class but eventually realized we calculate by sending out ray from the object, than the light so once we understood that it was a piece of cake.

Extra References:

http://www.flipcode.com/archives/Raytracing_Topics_Techniques-Part_2_Phong_Mirrors_and_Shadows.shtml

http://www.scratchapixel.com/