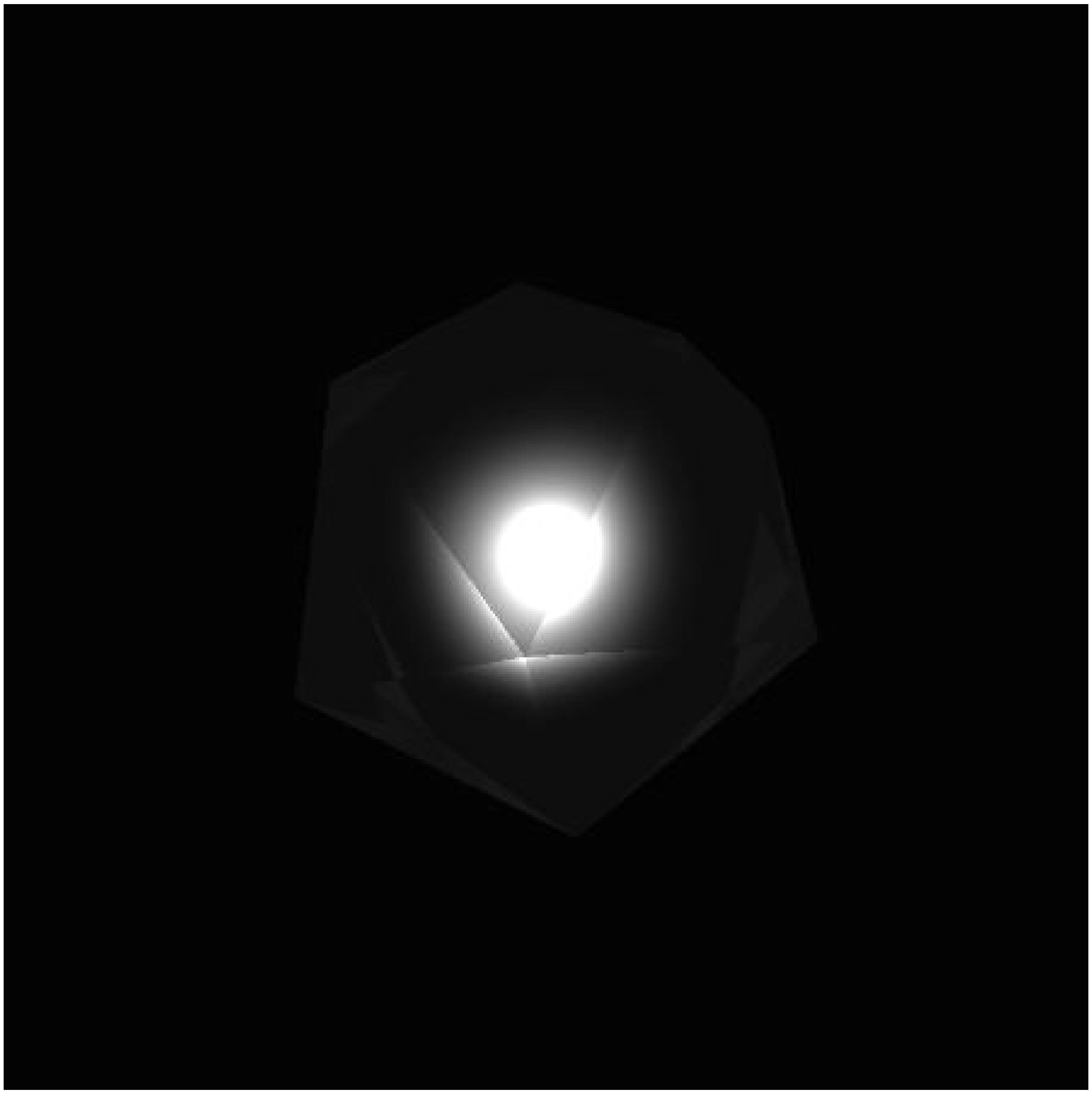
GRAPHICS LAB REPORT

Internal Illumination of a Translucent Object



Manikya Singh | 14CS10032

INTRODUCTION

Problem Statement

Given an 3d object mesh, render image output when a point light source is inside the mesh.

Input Format:

1. Maximum ray trace count (maximum number of times reflected/refracted rays off a transparent surface are traced further for calculation of color).
2. Opacity of object
3. RGB color of object, int in [0,255]
   R:
   G:
   B:
4. File name of output file
5. File name of .off file
6. Is entered .off file in color off format? (0 = No, 1 = Yes)
7. Scale Factor
8. X rotation
9. Y rotation
10. Z rotation
11. Refractive Index

Solution

The solution consist of following main steps-

1. Pre-processing step: the input from user is read and the object is stored in relevant data structures.
2. Rendering and Illumination: rendering and illumination step proceed hand in hand, i.e. for each pixel on screen the point of intersection to object is determined and then the color of that pixel is calculated using illumination model.
3. Post-processing step: The output of step 2 is a 2d array of colors which is written into a .ppm file in the final step.

Rendering Algorithm

Ray Tracing is used for rendering as refraction and reflection can be modeled easily when using ray tracing for rendering.

Light rays are considered from viewpoint to each pixel on screen. For each light ray nearest point of intersection to 3d object is calculated. If the opacity of object at point of intersection is not 100% then refractive index of the object is considered and it is determined whether it is case of total internal reflection or not. In either case the resultant refracted/reflected ray is used to find next nearest point of intersection, the technique can be used in recursion to calculate many number of refractions/reflection and the resultant color of the pixel is calculated accordingly.

Running time however increases linearly with each increased reflection/refraction consideration.

Illumination Model

Phong's Model is extended to account for both internal and external illumination of any transparent/translucent object. Color is calculated by the following equation-

(Note that all colors mentioned in the discussion have magnitude of their R,G,B components in [0,1].

Color = Ka*Ambient_Color + (Diffuse Color + Specular Color)/(1 + constant*distance^2)

Where Ka is coefficient of ambient reflection of surface, Ambient Color is fixed while Diffuse and Specular color are calculated as follows-

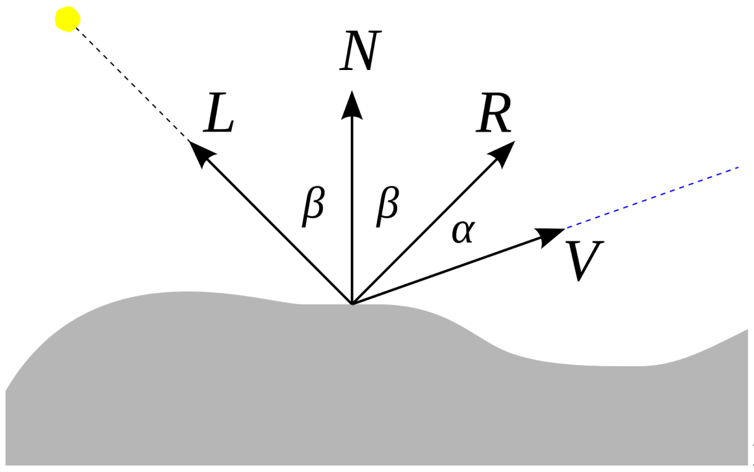


Figure 1

- For opaque objects, Phong's model is followed i.e.

    - If cos(β) <= 0 then Diffuse Color = 0
      Else Diffuse Color = Kd*cos(β)*Object Color*Light Color

    - If cos(α) < 0 Specular Color = 0
      Else Specular Color = Ks*cos^{n}(α)*Object Color*Light Color

  Where-

    - Kd and Ks are coefficient of diffuse and specular reflection of surface

    - n is shininess constant

    - Angles β and α are indicated in the figure 1

- For translucent objects,
  Reflected ray vector R(figure 1) becomes Refracted ray Vector and is calculated by finding direction of refracted ray (figure 2) (total internal refraction may also occur)

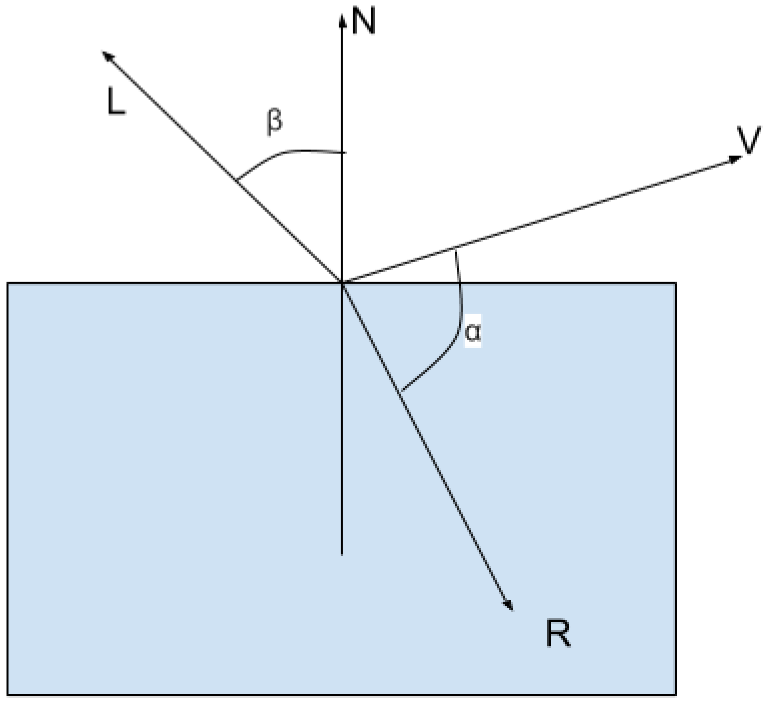


Figure 2

- Diffuse and Specular colors are calculated by-

  Diffuse Color = Kd*|cos(β)|*Object Color*Light Color

  If cos(α) < 0 Specular Color = 0
  Else Specular Color = Ks*$\cos^n$(α)*Object Color*Light Color

- However the reflected/refracted ray may be calculated if extra reflections and refractions are allowed in which case final color of the pixel associated with viewer light ray is calculated as-

  Color of pixel = transparency of point*color of point + (1-transparency of point)*(Background Color)

  Where Background Color is calculated by using reflected/refracted ray.

IMPLEMENTATION DETAILS

Helper Functions

- int compare(double a, double b)
    - Used for comparing doubles
    - Return value is 1 if a>b, 0 if a == b, and -1 if a<b
- int roots(double a, double b, double c, double *r1, double *r2)
    - Find solution of quadratic equation $ax^2 + bx + c = 0$ and saves the root in *r1 and *r2
    - Return value is number of real valued solutions

Classes

- **Matrix**
    - Implement Matrix , used mainly for transformations
- **Vector**
    - Implement Vectors, contains functions for finding dot product, cross product, reflection, refraction etc
- **Ray**
    - Implement Ray, (start point vector and direction vector), used in rendering
- **Color**
    - Implement Color(r, g, b, a) where r,g,b lie in [0,255] and a lie in [0,100]
- **Point**
    - Point is a Vector with a Color
- **Polygon**
    - Implement Polygon, contains function for finding intersection with a Ray
    - Each polygon has it's own ka, kd, ks (coefficients of ambient, diffuse and specular reflection)
- **Light**
    - Implemen point source light
    - Contains function for finding color of a point based on the illumination model described above
- **Object**
    - Contains polygons and other objects
    - Contains function project, to render object and use Light class to find color based on illumination model

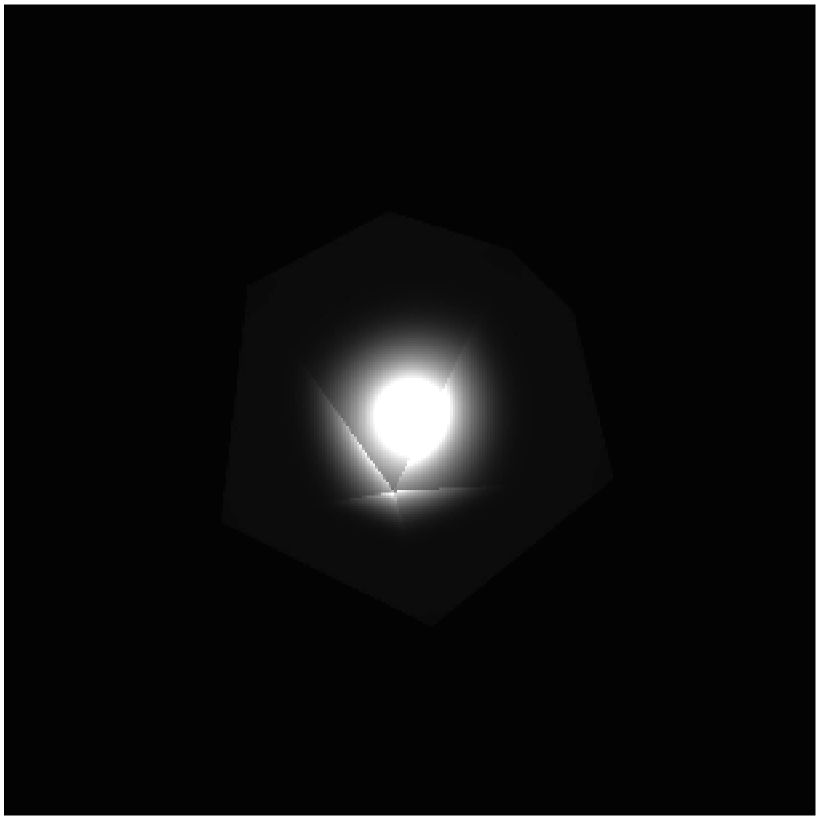
Psedu Code for Rendering and Illuminating an Object

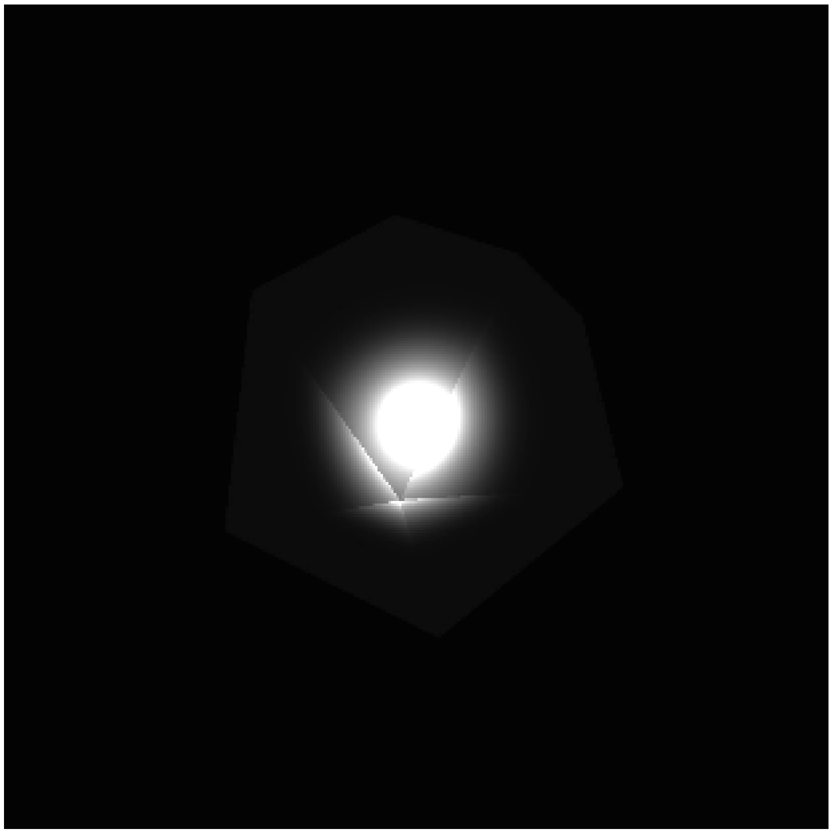
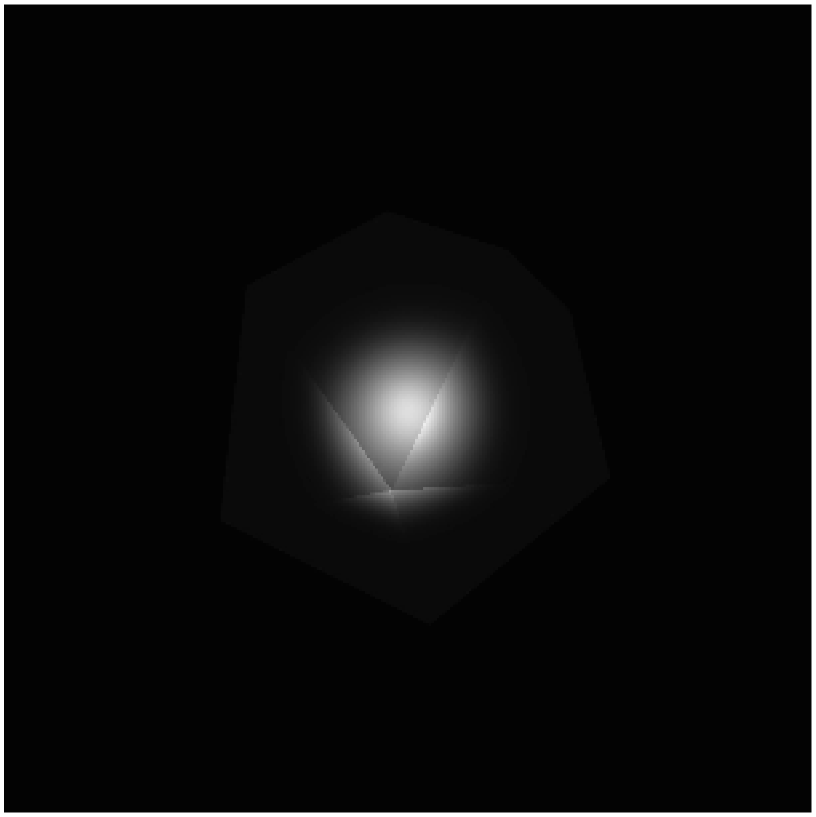
```
Color ** project(Object o, Light *l, int lc, int w, int h, Vector vp){
    //returns 2d array of color,
    // l = light sources array
    // lc = length of l
    // vp = Position of View Point
    // Image plane is XY plane , Image screen is centered at origin and sides parallel to
    axis , width = 2*w+1 and height 2*h+1 pixels
    Color ret[2*w+1][2*h+1];
    For (each point p on Image screen){
        Ray r = Ray starting from p in direction of (p-vp)
        Point pi = Nearest_intersection(r, o);
        Color tmp = shade(pi, l)
        Int trace_count = 0;
        while(trace_count < max_trace_count && tmp.a < 100){
            r' = refraction of ray r at pi
            pi = Nearest_intersection(r', o)
            Color tmp2 = shade(pi, l)
            tmp2.color.a = (100 - tmp.color.a)/100;
            tmp.r = (tmp.r*tmp.a + tmp2.r*tmp2.a)/(tmp.a+tmp2.a)
            tmp.g = (tmp.g*tmp.a + tmp2.g*tmp2.a)/(tmp.a+tmp2.a)
            tmp.b = (tmp.b*tmp.a + tmp2.b*tmp2.a)/(tmp.a+tmp2.a)
            tmp.a += tmp2.a;
        }
        if(cc == max_ray_trace_count){
            tmp.a = 100;
        }

        ret[i][j] = tmp;

    }

    return ret;
}

Color shade(Point p, Light *l){
    Color ret = rgba(0,0,0,p.color.a);
    For(each light source ls in l){
        ret += (determine color using illumination model discussed above);
    }
    Return ret;
}
```

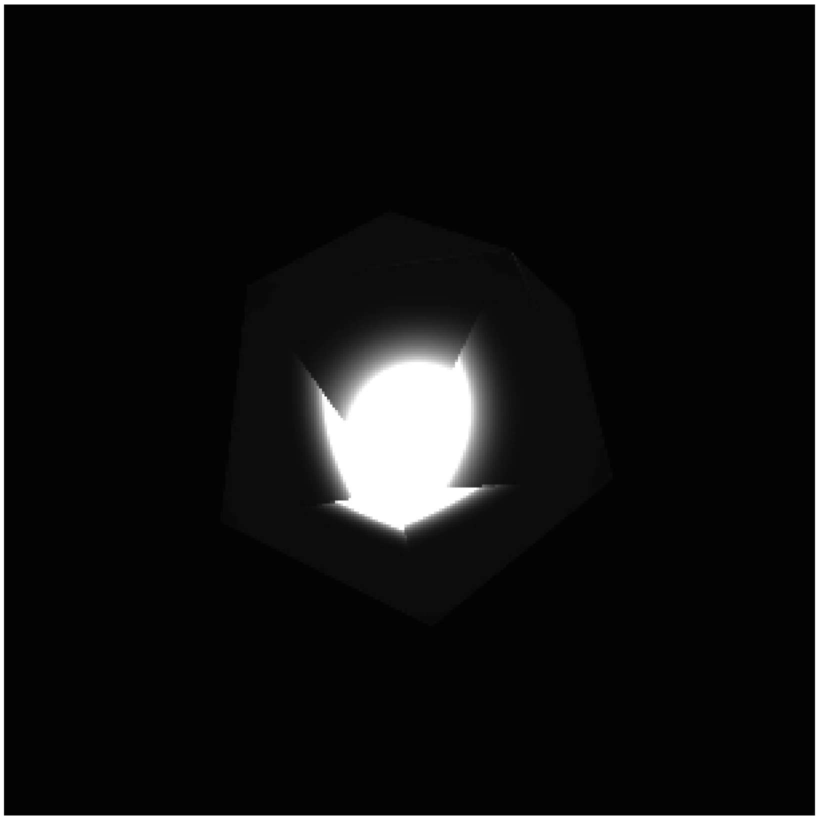
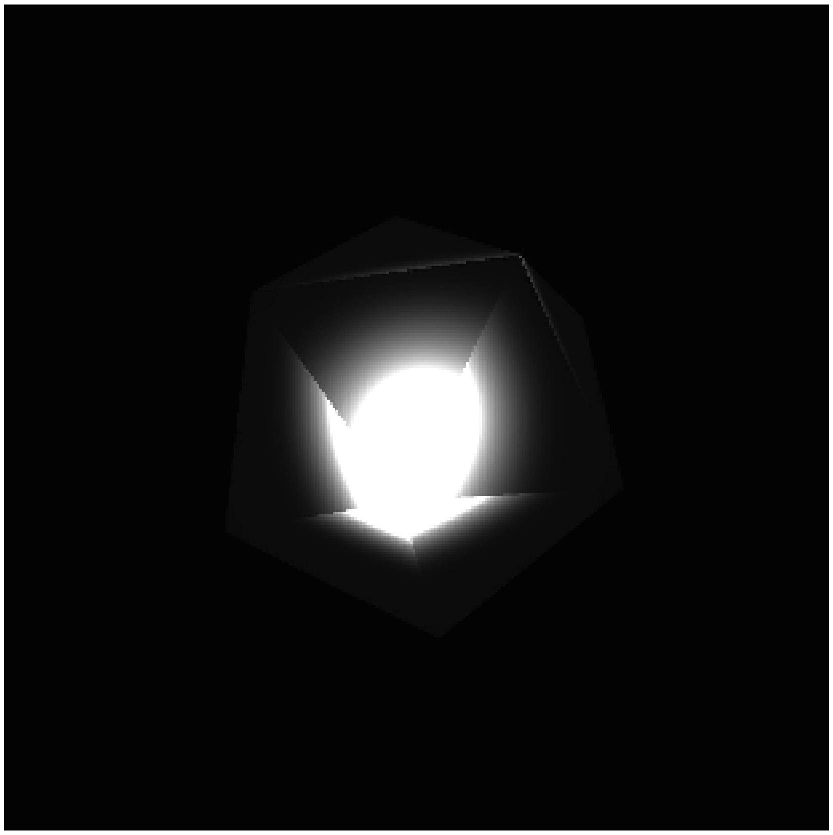
RESULTS

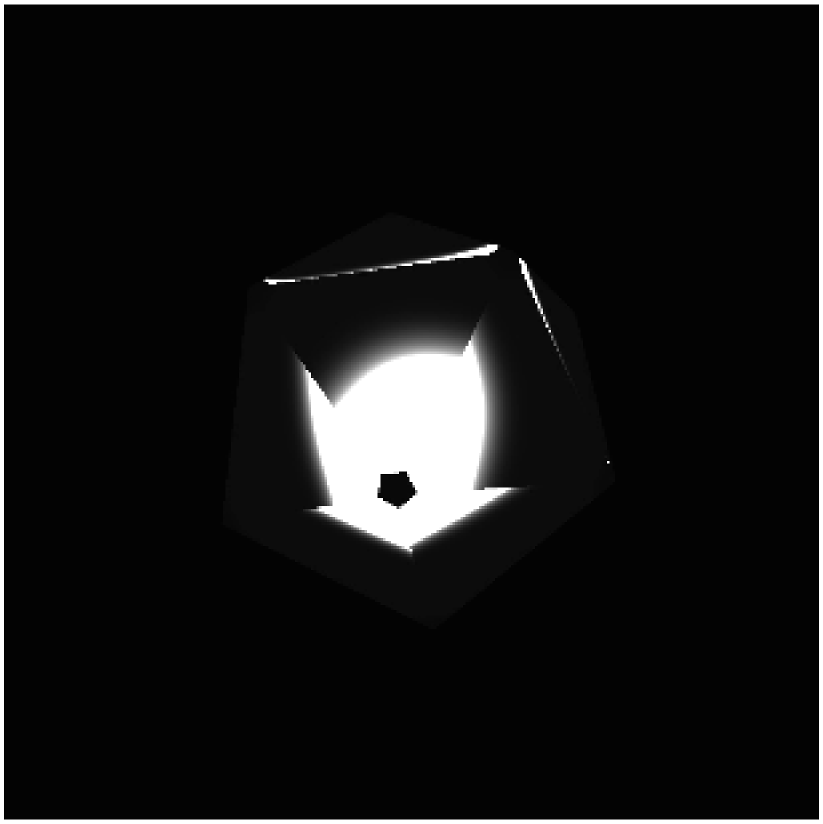
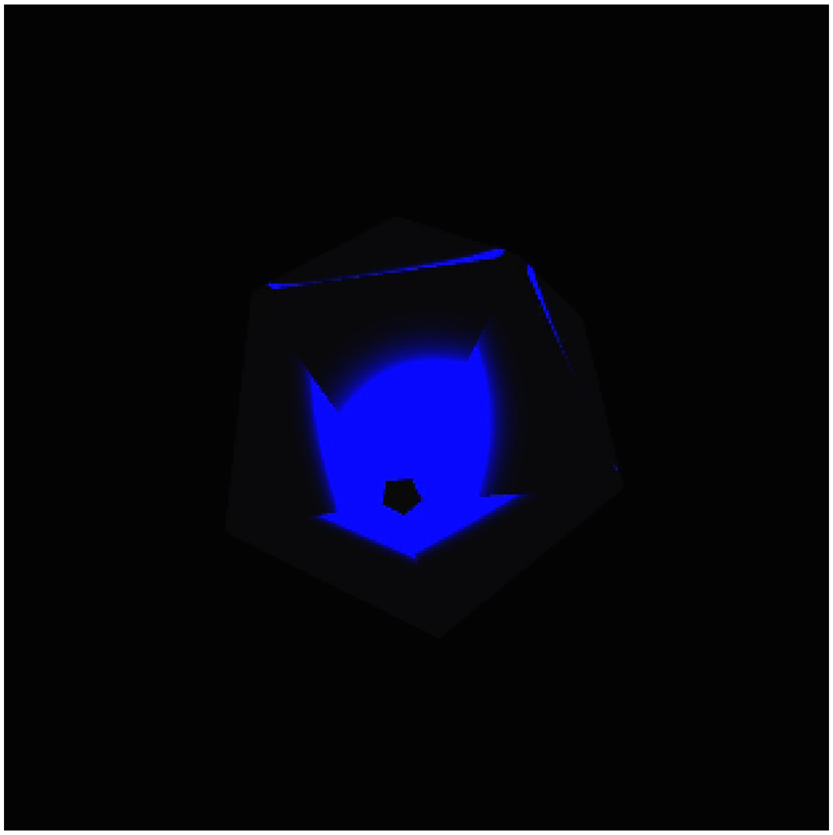
The values of various parameters for every output are as follows unless stated with the output:

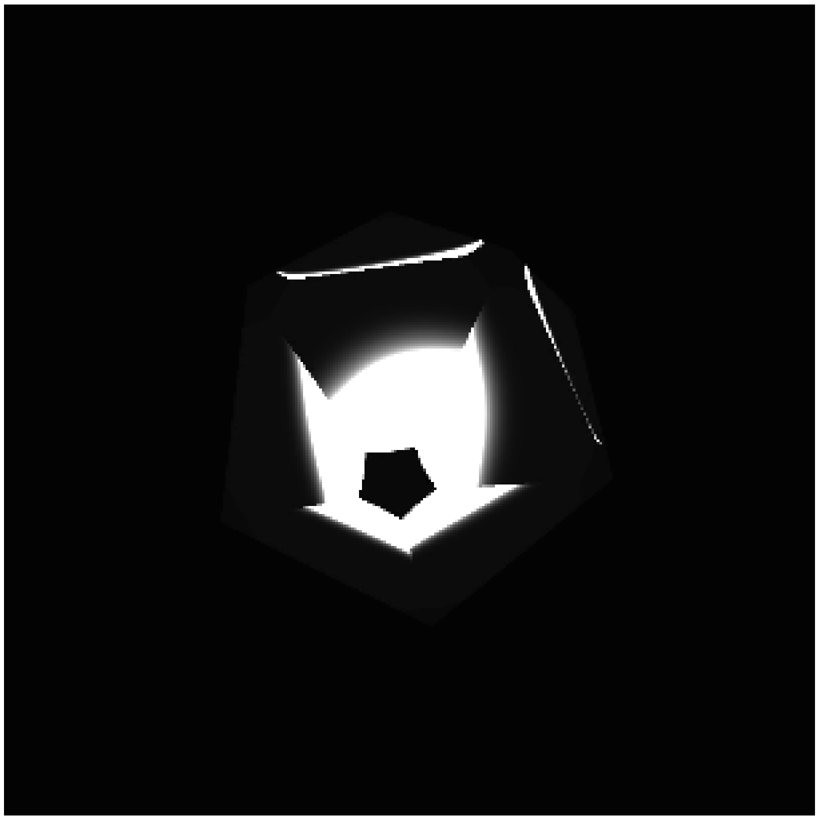
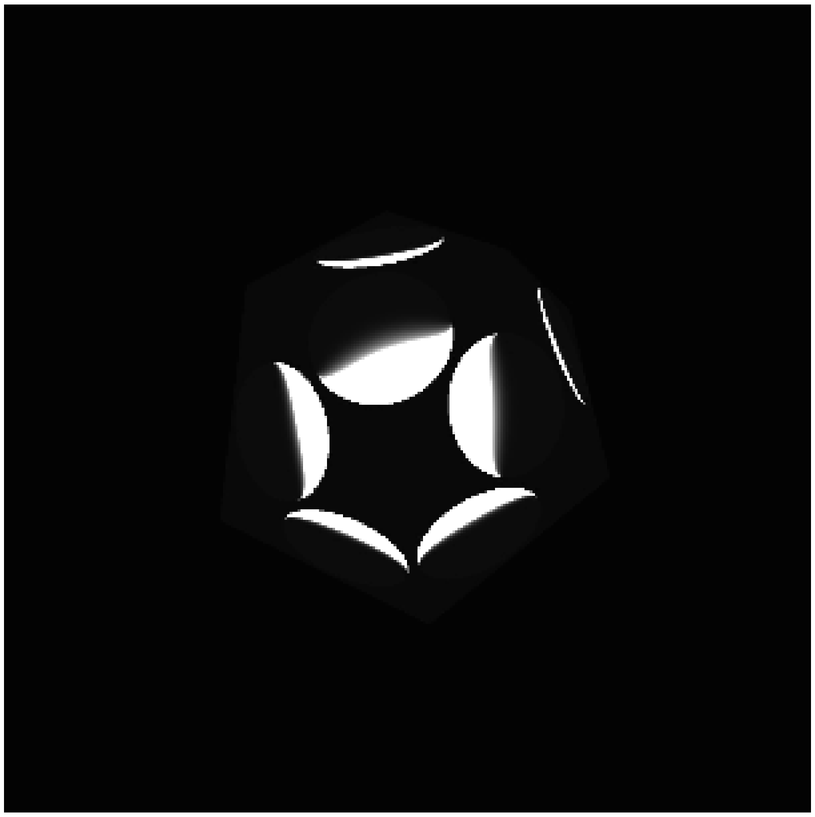
- Transparency of mesh: 25
- Color of mesh faces: rgb(255,255,255)
- n (shininess constant): 16
- Max Trace Count: 0
- Color of light source light: rgb(255,255,255)

	Refractice Index: 1.1

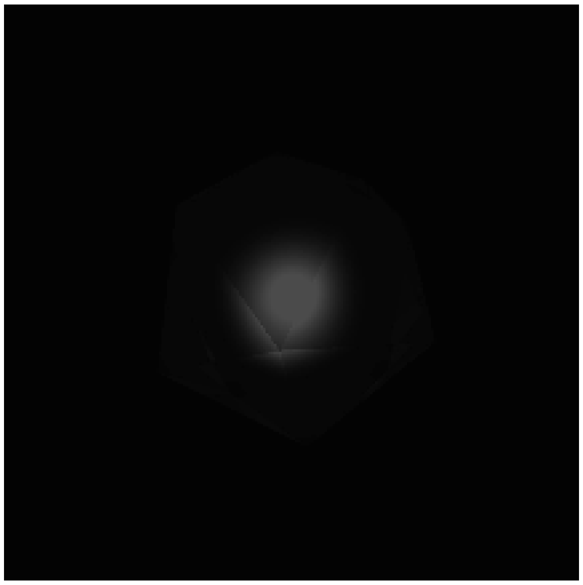
	Refractive Index: 1.1 Without distance attenuation
	Refractive Index: 1.1 By decreasing value of constant in distance attuenation function 1/(1 + d*d*constant)

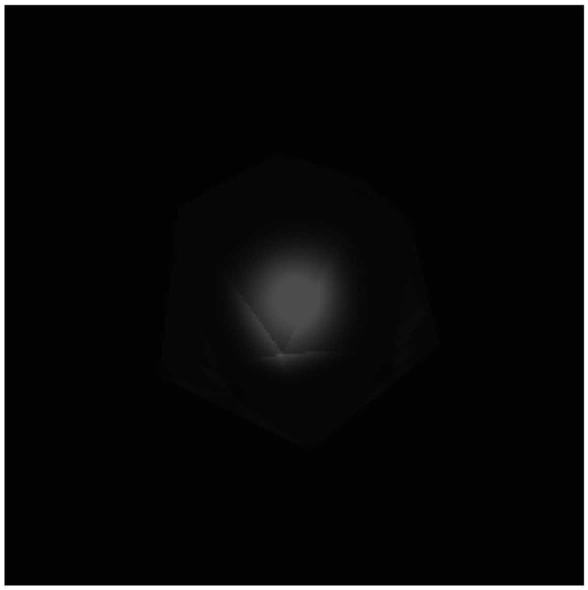
	Refractice Index: 1.4
	Refractive index: 1.4 Shineness Coefficient: 8

	Refractice Index: 1.8
	Refractice Index: 1.8 Object Color: rgb(0,0,255)

	Refractice Index: 2.0
	Refractice Index: 3.0 (We can ovserve the circle of outcomming light formed due to TIR.

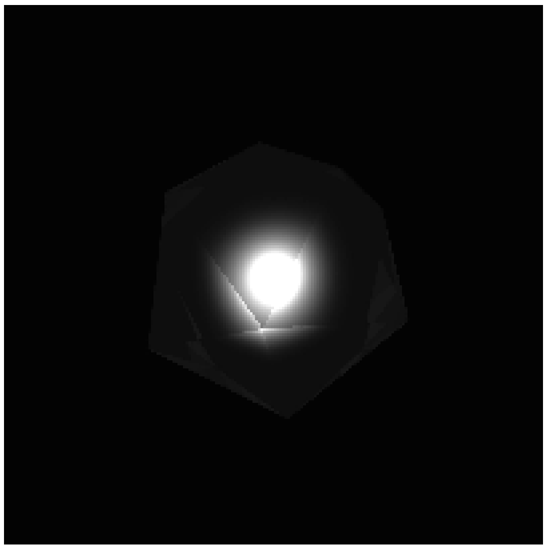
Increasing max trace count


Max trace count = 5


Max trace count = 10

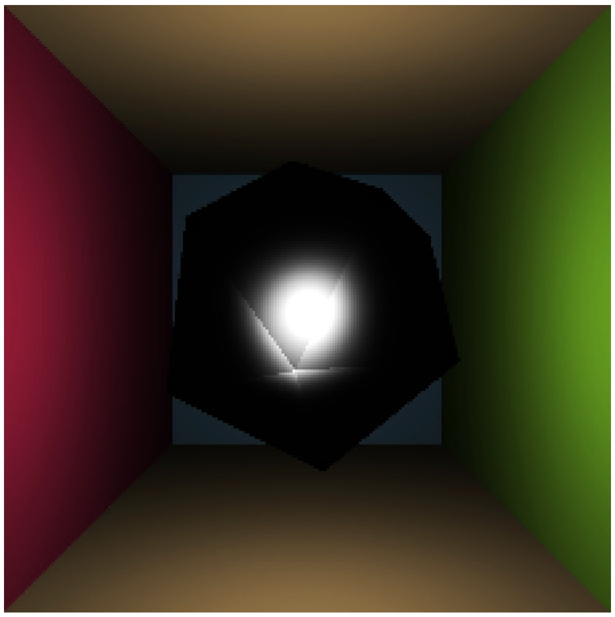
When we increase max trace count to 5 or 10 from 0, we observe an unsual drop in intensity of color at surface of our mesh.

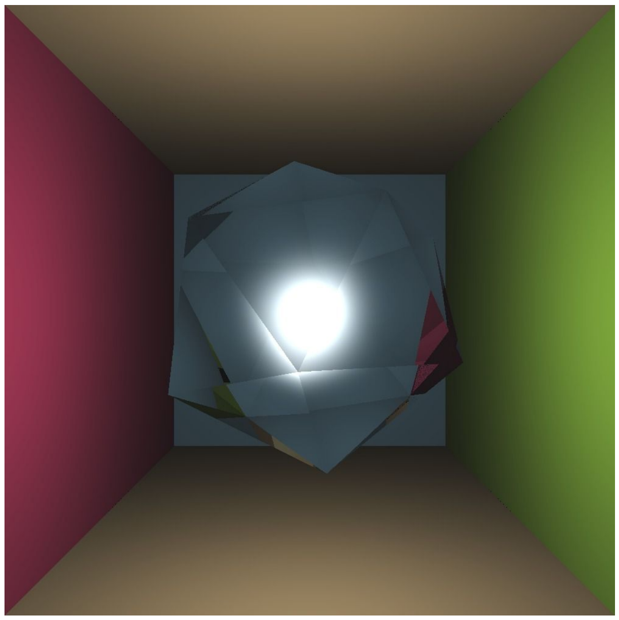
This is because untill now, we did not take in consideration the fact that specular reflection will have same intensity even if opacity of surface is less. Making this change in shading algorithim we get fair output.



Output in Cornell Box

We can observe that if we want to account for refraction and reflection with surroundings we need to keep max trace count above 1.

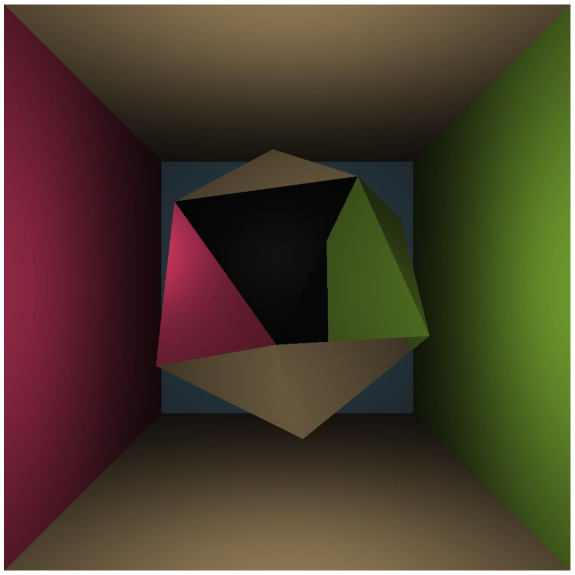




Max trace count = 0 | time taken: 23.47s | Max trace count = 10 | time taken: 24.12s

Low refractive indices

We can observe that very low refractive index object act like mirror as would be expected.



LIMITATIONS

One of the major limitation of implementation is that it is very slow. Reason being the fact that rendering time increases linearily with

1. Number of Pixel on image screen
2. Number of polygons in the mesh
3. Number of reflections/refractions if max trace count is not zero.