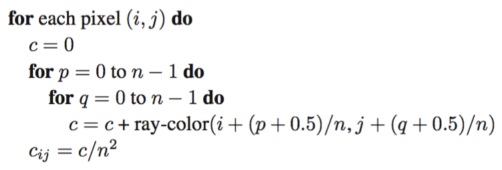
OVERALL SUBMISSION

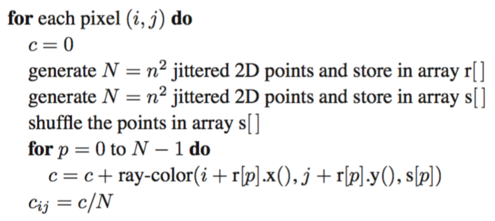
I designed six features aside from the shadows and reflection, shows as following.

1. Reflection: When the hits a specular object, then I shoot a reflection ray from it and add that contribution to pixel. More over the reflection coefficient decreasing exponentially when recursive depth increases.
2. Hard Shadow: To produce shadow I cast then ray from the intersection point back to light source, if it intersect some other objects before it reaches light source, then it is in shadow.
3. HandLing a non-trivial compound object: A cylinder centred at origin of height and width both equal to 1 is created for this part. And to dealing with with cylinder, basic is checking if height of the intersection is between $[-0.5, 0.5]$, if it is then we need determined whether it intersects with top and bottom unit disks or the body part, then solve some quadratic equation to determined the exact intersection.
4. Anti-aliasing: A simple way to antialias an image is to compute the average colour for the area of the pixel rather than the colour at the centre point; hence, we sample some ray for each pixel and get the average value. Following is the algorithm I used from the textbook one page 310.



```
for each pixel $(i, j)$ do
  $c = 0$
  for $p = 0$ to $n - 1$ do
    for $q = 0$ to $n - 1$ do
      $c = c +$ ray-color$(i + (p + 0.5)/n, j + (q + 0.5)/n)$
  $c_{ij} = c/n^2$
```

5. Glossy reflections: Some discernible image is visible in the reflection but it is blurred. I simulate this by randomly perturbing ideal specular reflection rays, using the Following is the algorithm I used from the textbook one page 314.
6. Soft shadow (extended light source): The reason shadows are hard to handle in standard ray tracing is that lights are infinitesimal points or directions and are thus either visible or invisible. So in order to create soft shadow, I created a parallelogram light source, then shadows is accounted for the light being an area rather than a point. following is the algorithm I used from the textbook one page 311.



```
for each pixel $(i, j)$ do
  $c = 0$
  generate $N = n^2$ jittered 2D points and store in array r[ ]
  generate $N = n^2$ jittered 2D points and store in array s[ ]
  shuffle the points in array s[ ]
  for $p = 0$ to $N - 1$ do
    $c = c +$ ray-color$(i +$ r$[p]$.x()$, j +$ r$[p]$.y(), s$[p])$
  $c_{ij} = c/N$
```

7. Texture-mapping: I did a text mapping for sphere using spherical coordinates, map every point on a sphere to a pixel on a image and get the colour of that pixel then shade the sphere, and I used a on online algorithm from from *https://github.com/ellisroberts* to do so.

8. Refraction: Using the algorithm provided in textbook page (305), I calculated the refractive ray, and decide whether the it is total internal reflection or not, as using some online source from $https : //github.com/ellisroberts$ and $https : //github.com/dat2$ to implement this part.

DEGREE OF THE FREEDOM

1. $bool\ _{s}hadows$: control whether have shadow or not.
2. $int\ _{m}axdepth$: control the recursive depth for both reflection and refraction.
3. $int\ _{a}ntiAlis$: control the number of ray sampled for Anti-aliasing.
4. $double\ _{b}ulrDegree$: control the blur degree for glossy reflections.

EXTERNAL RESOURCES

1. Fundamentals of Computer Graphics by Shirley (Course textbook): which us used of how to implement majority of the parts of this program.
2. $https : //github.com/ellisroberts$
3. $https : //github.com/dat2$