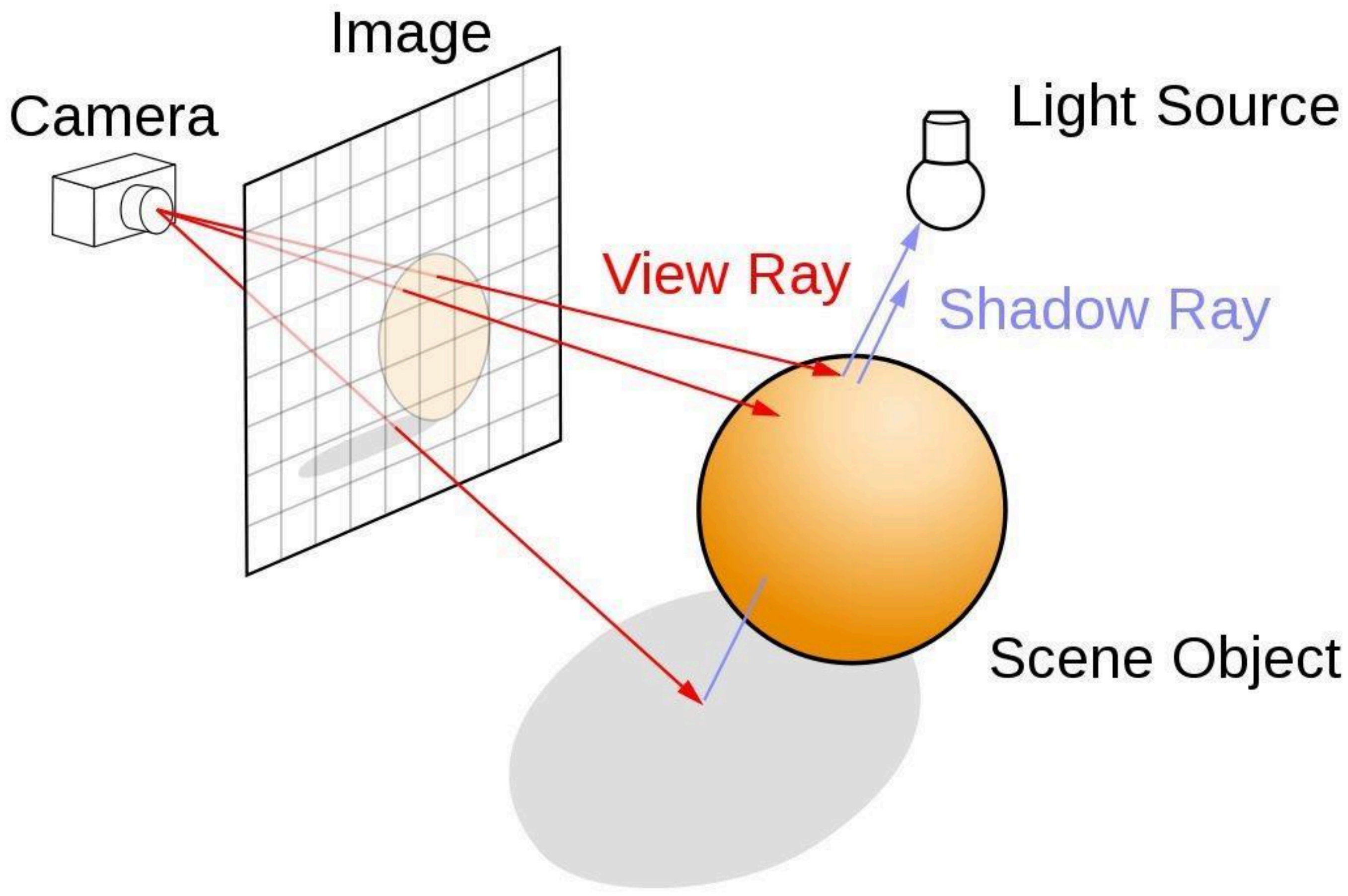
Assignment 2 – Ray Tracing



Overview:

The concept of ray tracing: a technique for generating an image by tracing the path of light through pixels in an image plane and simulating the effects of its encounters with virtual objects.
The technique can produce a very high degree of visual realism, usually higher than that of typical scan line rendering methods, but at a greater computational cost.

The objective of this exercise is to implement a ray casting/tracing manually.
A ray tracer shoots rays from the observer's eye through a screen and into a scene of objects.
It calculates the ray's intersection with the scene objects, finds the nearest intersection and calculates the color of the surface according to its material and lighting conditions.
(This is the way you should think about it – this will help your implementation).

<u>Requirement:</u>

Read this entire explanation before starting.
Understand the slides taught in class especially.
The feature set you are required to implement in your ray tracer is as follows:

- Display Geometric data in the 3D space:
  - Spheres
  - Planes
  - Background
- Basic Light Sources:
  - Global Ambient light
  - Directional lights
  - Spotlights
- Basic Materials color (Ambient, Diffuse, Specular)
- Basic Hard Shadows
- Reflection, up to 5 recursive steps (mirror object)
- Transparency (Spheres Only), up to 5 recursive steps (crystal ball)
- Multi-sampling for anti-aliasing (5 bonus points)

<u>Scene Definition:</u>

The 3D scene for rendering will be defined in a scene definition text file.
The scene definition contains all the parameters required to render the scene and the objects in it.
The specific language used in the definition file is defined later.

<u>Screen:</u>

The screen is located on the **z=0** plane.
The right up corner of the screen is located at **(1,1,0)**,
and the left bottom corner of the screen is located at **(-1,-1,0)**.
All in the scene coordinates.

<u>Display Geometric data in the 3D space:</u>

Geometric Primitives:

- ***<u>Sphere:</u>***
  A sphere is defined by a center point **(x,y,z)** and scalar radius **r**. The normal of each point on the sphere's surface is easily computed as the normalized subtraction of the point and the center.
- ***<u>Plane:</u>***
  A plane is defined by an **un-normalized** normal vector **(a,b,c)** to the plane and a negative scalar which represents the **d** in the plane equation (**ax+by+cz+d=0**).
  <u>Notice</u>**:** In the plane equation you use the **un-normalized** normal vector **(a,b,c)**, and don't forget the intersection calculations to **normalize** it).
  Every plane is an infinite plane and will be divided into squares in checkerboard pattern.
  In dark squares the diffuse component of the Phong model has **0.5** coefficient.

Spheres and Planes may intersect with each other.

You may use the following code to get the checkerboard pattern:

```
vec3 checkerboardColor(vec3 rgbColor, vec3 hitPoint) {
    // Checkerboard pattern
    float scaleParameter = 0.5f;
    float checkerboard = 0;
    if (hitPoint.x < 0) {
       checkerboard += floor((0.5 - hitPoint.x) / scaleParameter);
    }
    else {
       checkerboard += floor(hitPoint.x / scaleParameter);
    }
    if (hitPoint.y < 0) {
       checkerboard += floor((0.5 - hitPoint.y) / scaleParameter);
    }
    else {
       checkerboard += floor(hitPoint.y / scaleParameter);
    }
    checkerboard = (checkerboard * 0.5) - int(checkerboard * 0.5);
    checkerboard *= 2;
    if (checkerboard > 0.5) {
       return 0.5f * rgbColor;
    }
    return rgbColor;
}
```

Background:

The background of the rendered scene is in flat black color **(0,0,0)**.

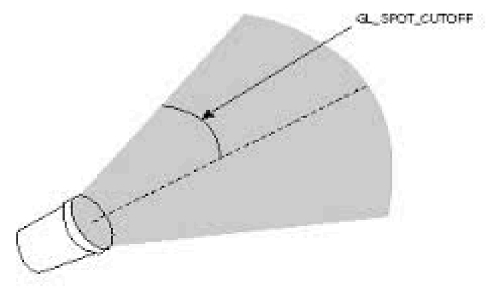
Camera:

The camera is a simple pinhole camera and will be located at **eye** coordinates (as specified in the given input file) and looks towards the center of the screen.

Basic Light Sources:

For basic lighting you need to implement:

1. **Global Ambient light** – A light that reflects from all surfaces with equal intensity.
2. **Directional light** – A light source like the sun, which lies at infinity and has just a direction.
3. **Spotlight** – A point light source that illuminates in a direction given by a direction vector.
   This light source has a cutoff angle as described in the following image.



Every light source has its own intensity (color) and there can be multiple light sources in a scene.

Basic Materials:

You need to implement the lighting formula of the Phone model.
The material of a surface should be flat with Material **Ambient**, **Diffuse**, **Specular** colors, and Material **Shininess** (the power of $\mathbf{V}\cdot\mathbf{R}$ when **V** and **R** are unit vectors).

Basic Hard Shadows:

You will add a shadow effect to your scene using parameters in the Phong model.
Shadow appears when some object is located **between** the **light source** and **another object** which can be observed by the viewer (or outside the cutoff angle of spotlight sources).
In that case you should ignore the covered light source part in the Phong model.
Some common mistakes may cause spurious shadows to appear.
Make sure you understand the vector math involved and all the edge-cases.

Notice: Spotlight sources have position, and objects can appear behind them.
Make sure that in your calculations to find if there is another object between the intersection point you found, and a spotlight source, you ignore objects that appear behind the spotlight.

In the equation shadows are expressed in the $\mathbf{S}_i$ term.
To know if a point **p** in space (usually on a surface) lies in a shadow of a light source,
you need to shoot a ray from **p** in the direction of the light and check if it hits another object.
If it does, make sure that it really hits it before reaching the light source and that this object is not the same one.

Reflection:

Each ray which hits a mirror object breaks symmetrically to the normal.
The color of the pixel in a mirror object is calculated according to the breaking ray.
For the reflective objects, ignore the material parameter (Ambient, Diffuse, Specular) and use the color from the **reflected** light.

Transparency (Spheres Only):

Use Snell's law with a refractive index of 1.5 to determine the direction of the light rays that intersect with the sphere (air has refractive index of 1).
For the transparent objects, ignore the material parameter (Ambient, Diffuse, Specular) and use the color from the **refracted** light.
Notice: You may assume that there will be no other objects that intersect or appear inside the **transparent** objects (Since it will require handling a lot of edge cases), and raise the recursive count after the ray gets out of the sphere.

Multi-Sampling for anti-aliasing (5 bonus points):

Shoot 4 or more rays from each pixel and combine the color values you get to determine the pixel color.

Input:

You will get scene text files (like in the example), where the first parameters meaning are as follows:

- **"e"** (eye) – Represents the **Camera position** coordinates **(x,y,z)**.
  The 4th coordinate can be used as a mode flag for the **multi sampling for anti-aliasing** bonus.
  (if you choose to not implement the bonus, you can ignore this coordinate value).
- **"a"** (ambient) – Represents the **Global Ambient Intensity (r,g,b)**.
  The 4th coordinate will always be 1.0 and can be ignored.

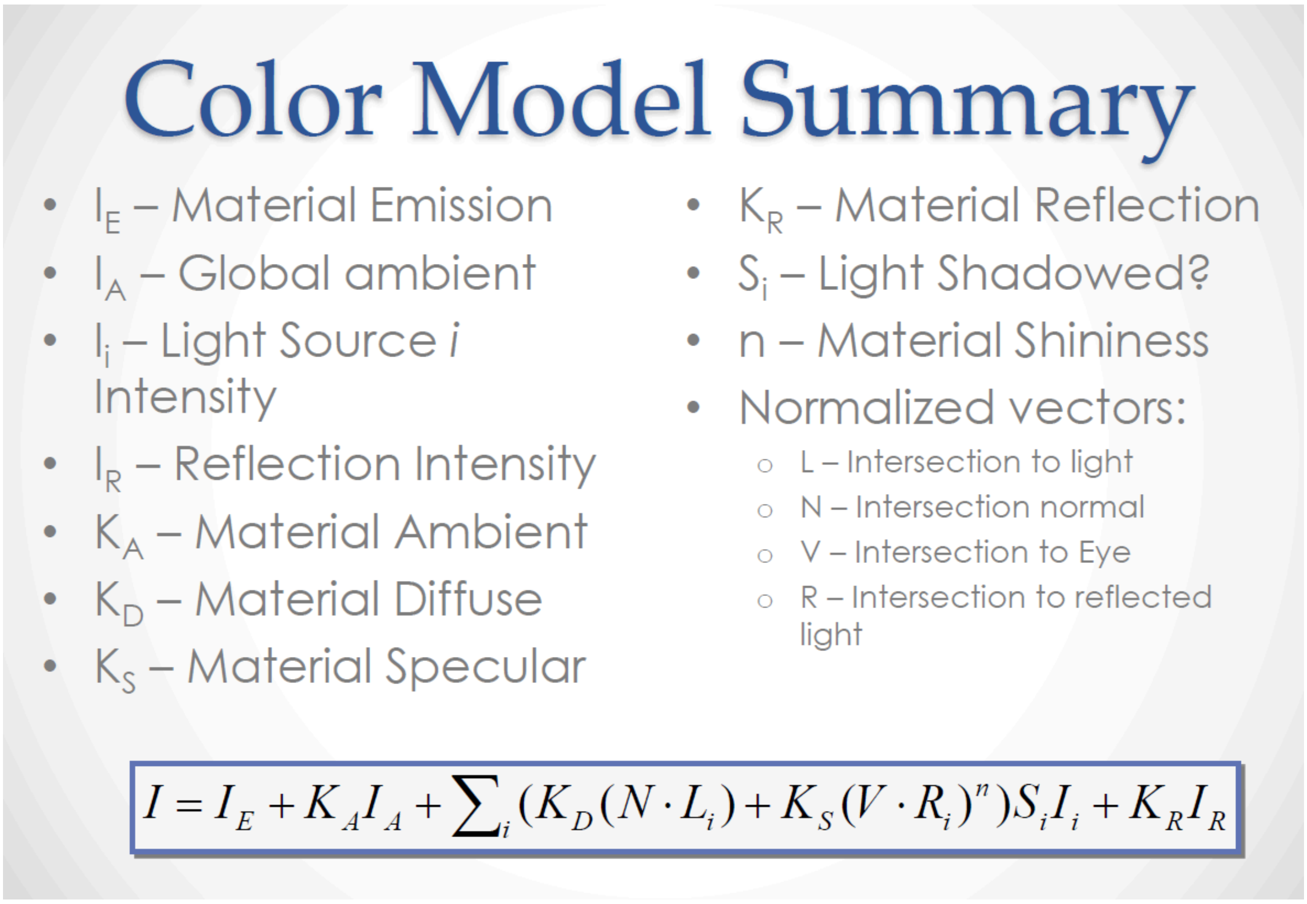
The next parameters represent the information about the **light sources** and **objects**:

- **"d"** (direction) – Represents the **Light source direction (x, y, z)**.
  The 4th coordinate value will be 0.0 for **Directional light** and 1.0 for **Spotlight**.
- **"p"** (position) – (Only for spotlights) Represents the **Spotlight** position coordinates **(x,y,z)**.
  The 4th coordinate value represents the **cutoff angle cosine value**.
  (**"p"** order corresponds to the **"d"** spotlights order)
- **"i"** (intensity) – Represents the **Light source Intensity (r, g, b)**.
  The 4th coordinate will always be 1.0 and can be ignored.
  (**"i"** order corresponds to the **"d"** order)
- **"o"** (object), **"r"** (reflective) or **"t"** (transparent) – Represents Spheres and Planes, where:
  - o Spheres **(x,y,z,r)** – where **(x,y,z)** is the center position and **r** is the radius (**r>0**).
  - o Planes **(a,b,c,d)** – where **(a,b,c,d)** represents the coefficients of the plane equation (**d<=0**).

  Notice the following things:
  - o The 4th coordinate determines if the object is a sphere or a plane.
  - o Spheres and Planes can be either **normal** objects, **reflective** objects or **transparent** objects and will require different handling based on their type.
  - o For "r" (reflective) and "t" (transparent) and next "c" parameter values can be ignored.
- **"c"** (color) - Represents the **Ambient Material** and **Diffuse Material** color **(r,g,b)**.
  The 4th coordinate represents the **Shininess** value.
  (**"c"** order corresponds to the **"o"**, **"r"**, **"t"** order)

The input parameters explanation according to the Phong model equation:

You have seen the following Phong equation:



Color Model Summary

- I_E – Material Emission
- I_A – Global ambient
- I_i – Light Source *i* Intensity
- I_R – Reflection Intensity
- K_A – Material Ambient
- K_D – Material Diffuse
- K_S – Material Specular
- K_R – Material Reflection
- S_i – Light Shadowed?
- n – Material Shininess
- Normalized vectors:
  - L – Intersection to light
  - N – Intersection normal
  - V – Intersection to Eye
  - R – Intersection to reflected light

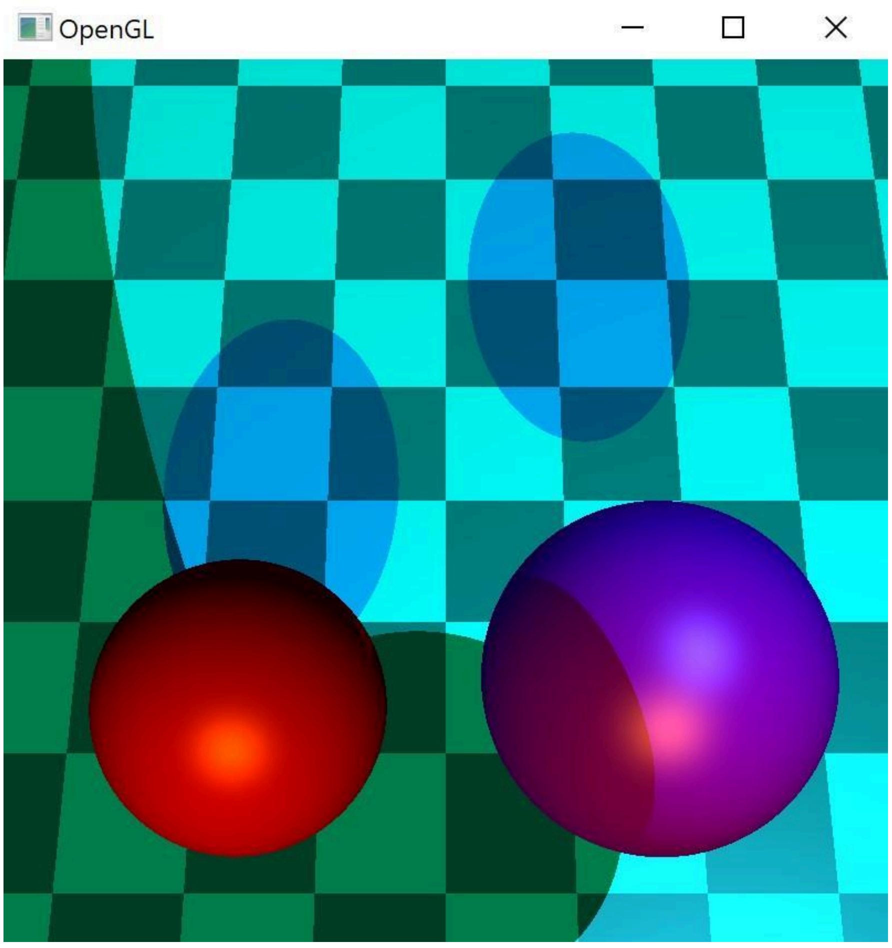
$$\boxed{I = I_E + K_A I_A + \sum_i (K_D (N \cdot L_i) + K_S (V \cdot R_i)^n) S_i I_i + K_R I_R}$$

And in this task, this is the parameters corresponding values (per **object** on the intersection point):

I_E – Assume as **(0,0,0)** for all objects.
I_A – The **(r,g,b)** values of **"a"**.
I_i – The **(r,g,b)** values of the i-th **"i"**.
I_R – Assumed as **(0,0,0)** for **normal** object **"o"** and (1,1,1) for **reflective "r"** and **transparent "t"** objects.
K_A – The **(r,g,b)** values of **"c"**.
K_D – The **(r,g,b)** values of **"c"**.
K_S – Assume as **(0.7,0.7,0.7)** for all objects.
K_R – The final color from the recursive calculations of **reflective** or **transparent** objects.
S_i – The shadow term binary value (0 if the i-th light source is blocked, 1 otherwise)
n – The 4th value of **"c"**.
Normalized vectors – Use the values of the remaining parameters to calculate them as required.

Input example:

```
e 0.0 0.0 4.0 1.0

a 0.1 0.2 0.3 1.0

o 0.0 -0.5 -1.0 -3.5
o -0.7 -0.7 -2.0 0.5
o 0.6 -0.5 -1.0 0.5

c 0.0 1.0 1.0 10.0
c 1.0 0.0 0.0 10.0
c 0.6 0.0 0.8 10.0

d 0.5 0.0 -1.0 1.0
d 0.0 0.5 -1.0 0.0

p 2.0 1.0 3.0 0.6

i 0.2 0.5 0.7 1.0
i 0.7 0.5 0.0 1.0
```



Notes:

Design Before you start coding!
Think about how you will represent the scene information in a way that will be easy for you to work with.
Start checking from a small scene with one or two objects.

Validation:

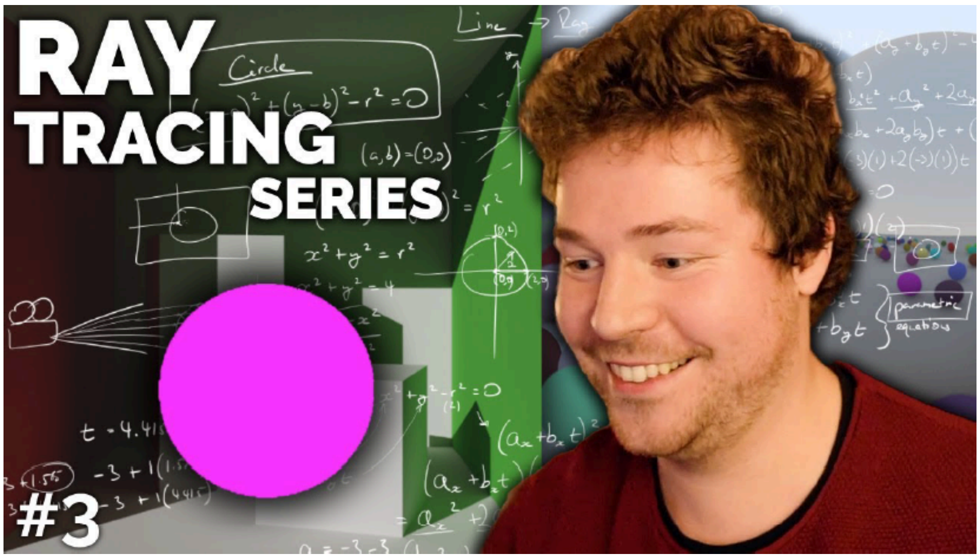
We will provide you with sample scenes for validation and the way they are supposed to be rendered in your ray tracing algorithm.
Your displayed result may vary from the supplied image in small details, but in general the scenes should look the same.

Helpful links:

https://en.wikipedia.org/wiki/Ray_tracing_(graphics)

https://www.cl.cam.ac.uk/teaching/1999/AGraphHCI/SMAG/node2.html

You can watch the ray tracing series as well (The image is clickable).



How to Start:

Start implementing your solution in the following order, and validate in each step that you get the expected results:

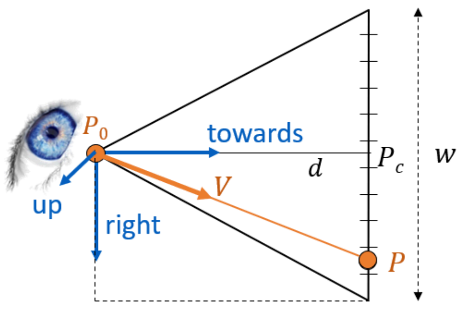
1. Define the data structures you will use to store the relevant data from the text file.
   You may find it helpful to use Object Oriented Programming and build classes and hierarchies for the **Objects** and **Light sources** (For example: A Plane is an Object; Directional light is a Light source), and also for the **Rays** and **Intersection Points** to hold your current data during a recursive call.
2. Build the reader of the text scene data, that loads the information to the data structures you have defined on step 1.
3. Build the double **for** loops for running over all the pixels on the screen and finding the rays that pass through them.
   To do that, find the sizes of a single pixel and calculate a pixel center to find the ray that passes through it.
4. **(The important part)** Define a simple sphere or a box of your own and check that with the rays that you have built for each pixel you can get the sphere on the screen like in the provided video above.
5. Replace the sphere you have built with the spheres data from the text file you have read and check that you can see them on the scene.
6. Now you can move to implement the full Phong equation. Start with calculating the **Ambient**, **Diffuse** and **Specular** color of a **Normal** object, then implement the **Hard Shadows** and finally move to the **Reflective** and **Transparent** objects.

General tips:

- Before plotting a pixel, make sure that its color does not exceed the range of 0-1 (or 0-255) for every color channel.
- In the Transparency part, when trying to find the second intersection point inside the sphere you might get that there are 2 intersection points on the ray.
  In that case, always take the further intersection point (This might be happening when due to numerical errors, the closer point will have very small positive **t** value instead of absolute zero).
- Right vector of the screen will be orthogonal to up vector and toward vector:

Image Space → View Plane Space

- Image center: $P_c = P_0 + dV_{to}$
- $V_{right} = V_{to} \times V_{up}$ (and normalize)
- $\tilde{V}_{up} = V_{right} \times V_{to}$ (and normalize)
- Ratio (pixel width): $R = \frac{w}{R_x}$



$$P = P_c + \boxed{\left(x - \left\lfloor \frac{R_x}{2} \right\rfloor\right) R \, V_{right} - \left(y - \left\lfloor \frac{R_y}{2} \right\rfloor\right) R \, \tilde{V}_{up}}$$

Assignment Score:

- **Sanity** - Working Ray Tracer with Ambient Material Color and Regular Objects: **40 points**
- Diffuse Material Color: **5 points**
- Specular Material Color: **5 points**
- Shadow Term for all light sources: **10 point**
- Reflective Objects: **10 points**
- Transparent Objects: **10 points**
- Performance: **20 points**
- Bonus: Up to **5 points**

Submission:

Submit zip file with the following files:

1. Link to your GitHub repository.
2. **(Optional)** Text, Doc or PDF file with short explanation about the changes you did in the engine (files you change and functions you modify).
   This file will help you and us to understand what changes you have made in the engine to complete the assignment 😊

The zip file name must include your ID numbers as follows: **ID1_ID2.zip**.