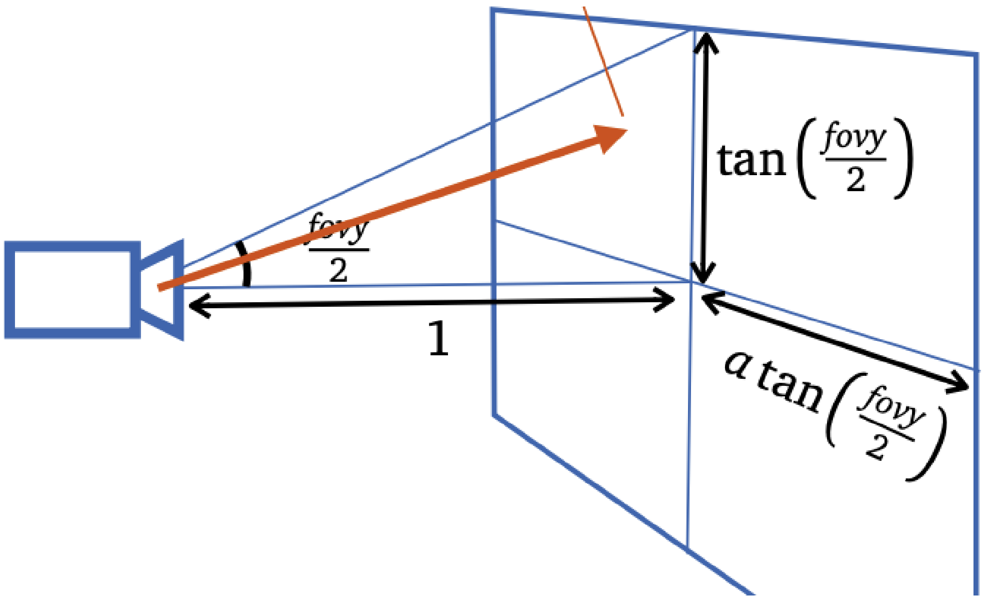
Partner 1: Chih-Lin Wang
Partner 2: Yaohui Chen
Topic: Raytracing
Objective of the project: Raytracing is a very impressive rendering technique with the goal to achieve photorealism. And the objective of this project is to implement a ray tracing framework so we can use that framework to render our object to achieve photorealism.

Steps:

1. Construct object with triangles
    a. We construct triangles by using the vertex of the geometric, for example a square with vertex A,B,C,D(four corners) containing 2 triangles. The first triangle has 3 corners(A,B and C) the second triangle has the corners(A C and D).
    b. By using this method we can use many triangles to create a complex shape like the teapot.
2. Build triangles soups (which contain all the triangles from all the objects in the scene)
    a. By multiplying the triangles with a corresponding transformation matrix we can transform the triangles into many different shapes. Like the table we transform the unit cube into a table top and table leg.
3. Construct a ray that traverse in the scene dimension
    a. The ray is made out of 2 parts the position where the ray starting from and the direction of the ray


$\frac{fovy}{2}$
1
$\tan\left(\frac{fovy}{2}\right)$
$a\tan\left(\frac{fovy}{2}\right)$

4. Check if the ray has any intersection with triangle in the triangle_soup if there is any retreat to the closest intersection.

a. The way we check if the ray intersects any triangle is to find the barycentric coordinate representation of the intersect position by using the triangle vertex (which means we need to run the pixel against all the triangle we have in the triangle soup) if barycentric coordinate coefficient (P1, P2 P3 and d) is all positive then intersection is inside the triangle.

use the barycentric coordinate (what we just solved)

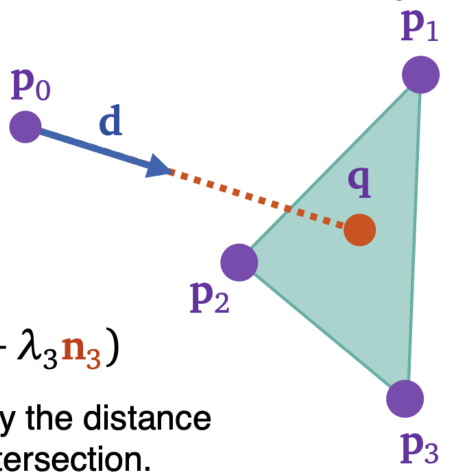
$$\lambda_1, \lambda_2, \lambda_3$$

to interpolate position and vertex attributes, such as normals

$$\mathbf{q} = \lambda_1 \mathbf{p}_1 + \lambda_2 \mathbf{p}_2 + \lambda_3 \mathbf{p}_3$$

$$\mathbf{n} = \text{normalize}(\lambda_1 \mathbf{n}_1 + \lambda_2 \mathbf{n}_2 + \lambda_3 \mathbf{n}_3)$$

The variable t we solved is exactly the distance between the ray source and the intersection.



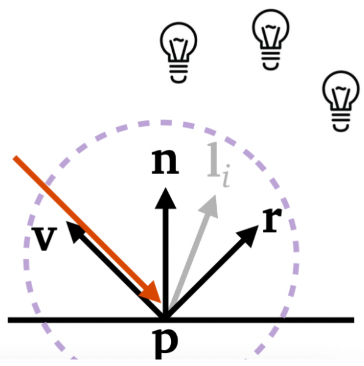
5. After retreating the intersection from the closest triangle (If there is any), create a new ray at the position of where the intersection occurs and shoot that ray toward the light position.
6. If the new ray intersects some triangle before it reaches the light that means we know that light is blocked by that triangle(object) and it will create a shadow on that position and we will assign the color of that pixel to the ambient of the object plus the reflection from the new ray multiple with the specular. That will create a more realistic shadow than just set the pixel to black.
7. If the ray is able to reach the light we assign the color of the light on the object. By using Ambient + Lambertian-diffuse + Blinn-Phong formula to calculate the color from the light.

$$\mathbf{L}_{\text{seen}} = \sum_{i \in \text{lights}} \mathbf{C}_{\text{diffuse}} \mathbf{L}_{\substack{\text{light} \\ \text{source}_i}} \max(\mathbf{n} \cdot \mathbf{l}_i, 0)\ \text{visibility}_i$$

$$+\ \mathbf{C}_{\text{specular}} \boxed{\mathbf{L}_{(\mathbf{p},\mathbf{r})}}$$

color seen by
ray (**p**, **r**)

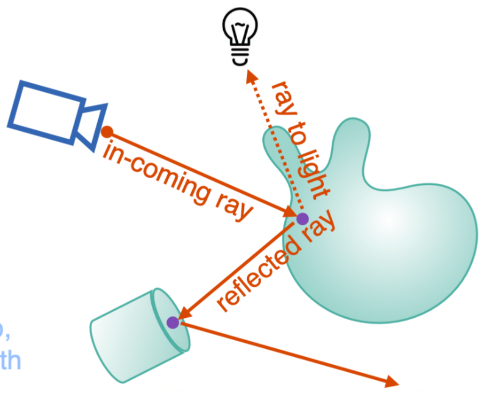
- mirror reflection direction
  $\mathbf{r} = 2(\mathbf{n} \cdot \mathbf{v})\mathbf{n} - \mathbf{v}$



8. After that we shoot another ray called reflect_ray that is at the position where the original ray interests the triangle. That reflect_ray is going to return another closest intersected triangle and we are going to pass the intersects to repeat step 4 to find the color and multiply with the original triangle's material specular. And add it to the color of the pixel. (Like displaying the reflection on the pixel)

```
Color FindColor( hit ){
  • Generate secondary rays to all lights
      ▸ color = Visible? ShadingModel : 0;
  • ray2 = Generate mirror-reflected ray
      ▸ hit2 = Intersect( ray2, scene );
      ▸ color += specular * FindColor( hit2 );
  • return color;
}
```

▸ Recursion might never stop, so set a max recursion depth



9. But If the initial ray didn't intersect any triangle which means that ray has not hit any object and we will set color to the background color.

10. Repeat the step 3 to step 9 for every pixel on the screen. (recursive steps until reach the max bounce MAX_RECURSION_DEPTH)