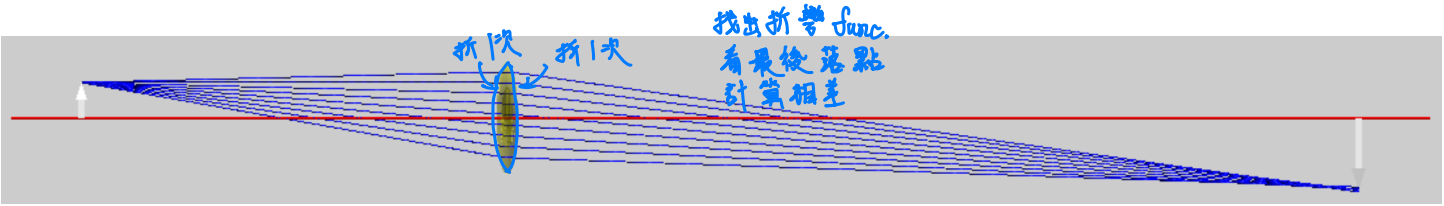
VP11 Simulation for Imaging by a thick lens



When we draw the imaging diagram for a lens, we usually draw three optical rays:

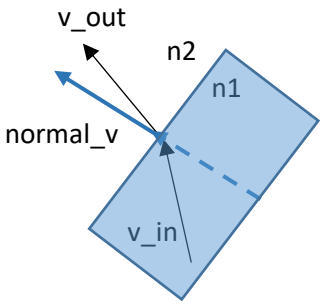
(1) The ray goes parallel to the optical axis, and after the lens, it is refracted and goes straight to pass through the back focal point;

(2) The ray goes straight to pass the center of the lens, and after the lens, it keeps its original direction;

(3) The ray goes straight and pass the front focal point, and after the lens, it goes parallel to the optical axis;

By geometric optics, the three rays will meet together at a point and form the image. The question is how we know that all other rays will all go through the same point behind the lens to form the image. We want to simulate this.

To simulate an optical ray, we use a moving object with a unit velocity vector that leaves a trace to represent the optical ray. When there is no interface, the optical ray goes straight. When the ray hits an interface, it is refracted. We need to establish a function, applying Snell's law, to calculate the **unit vector for the outgoing ray**, by the given **unit vector of the incoming ray**, the **unit normal vector of the interface**, and the refractive indices n1 and n2.




```
def refraction_vector(n1, n2, v_in, normal_v) :
    (code here)
    return v_out
```

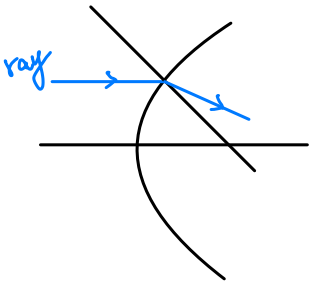
After you have the correct function to find the refracting ray, you will use the following parameters to do the imaging simulation. As shown in the title figure, an arrow object is at (-6, 0, 0) with the arrow length = 0.5. A glass (n_glass = 1.5) with two spherical surfaces form the lens. The central thickness of the lens is 0.3 cm. The radius of curvature of both surfaces are 4 cm. The lens center is at (0, 0, 0). Finish the following code to do the simulation for at least 9 rays (already specified in the given codes) to see if they meet at the same point. Also print the ray's y position when it reaches the theoretical imaging plane (at x = 12) obtained by image formula for an ideal thin lens.

```
from vpython import *

scene = canvas(background=vec(0.8, 0.8, 0.8), width=1200, height=300, center = vec(3,0,10), fov = 0.004)

lens_surface1 = shapes.arc(radius=0.15, angle1=0, angle2=pi)
circle1 = paths.arc(pos=vec(0, 0, 0), radius=0.0000001, angle2=2*pi, up = vec(1,0,0))
lens_surface2 = shapes.arc(radius=0.15, angle1=-pi, angle2=0)
circle2 = paths.arc(pos=vec(0, 0, 0), radius=0.0000001, angle2=2*pi, up = vec(1,0,0))
extrusion(path=circle1, shape=lens_surface1, color=color.yellow, opacity = 0.6)
extrusion(path=circle2, shape=lens_surface2, color=color.yellow, opacity = 0.6)
curve(pos=[vec(-7,0,0),vec(13,0,0)], color=color.red, radius = 0.02)
```

```
arrow(pos=vec(-6,0,0), axis=vec(0,0.5,0), shaftwidth=0.1)
arrow(pos=vec(12, 0, 0), axis=vec(0, -1, 0), shaftwidth = 0.1)

def refraction_vector(n1, n2, v_in, normal_v):
    # find the unit vector of velocity of the outgoing ray
    return v_out

R = 4.0
thickness = 0.3
g1center = vec(-R + thickness/2, 0, 0)
g2center = vec(R - thickness/2, 0, 0)
nair = 1
nglass = 1.5

for angle in range(-7, 2):
    ray = sphere (pos=vec(-6, 0.5, 0), color = color.blue, radius = 0.01, make_trail=True)
    ray.v = vector (cos(angle/40.0), sin(angle/40.0), 0)

    dt = 0.002

    while True:
        rate(1000)
        ray.pos = ray.pos + ray.v*dt

        # your code here

        if ray.pos.x >= 12:
            print(ray.pos.y)
            break
```

粒子出發

留下軌跡



用 Snell's Law

改變 ray.v 的方向、大小