Lecture 20: All Together with Refraction

November 10, 2020

Translucence

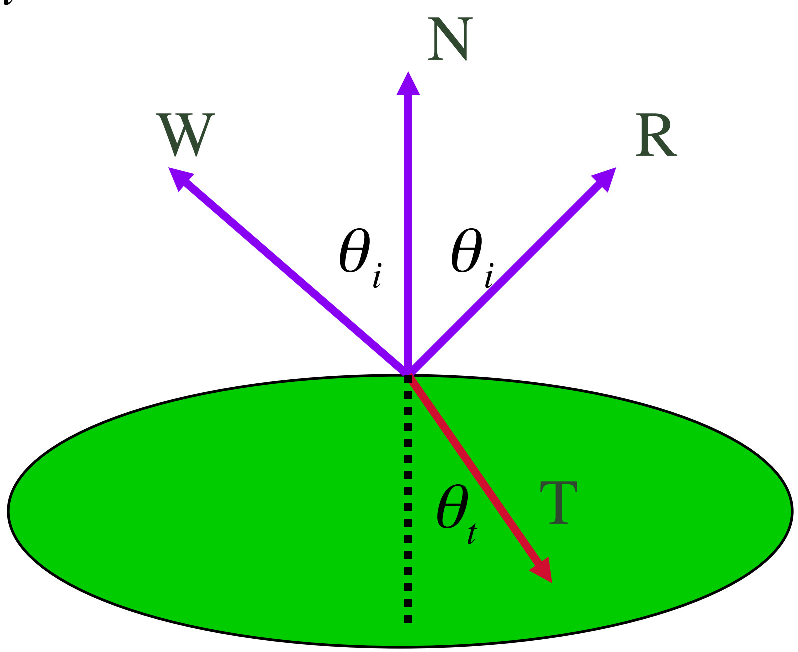
- Some light passes through the material.
  - Typically, “passed through” light gets the diffuse reflection properties of the surface, unless object is 100% translucent (i.e. transparent)
- Speed of light is a function of the medium
  - This causes light to bend at boundaries
  - example: looking at the bottom of a pool

Refraction - With Trigonometry

Key is Snell's law …

$$\sin(\theta_t) = \frac{\eta_i}{\eta_t}\sin(\theta_i)$$

θ_i	Angle of incidence
θ_t	Angle of refraction
η_i	Index of refraction material #1
η_t	Index of refraction material #2

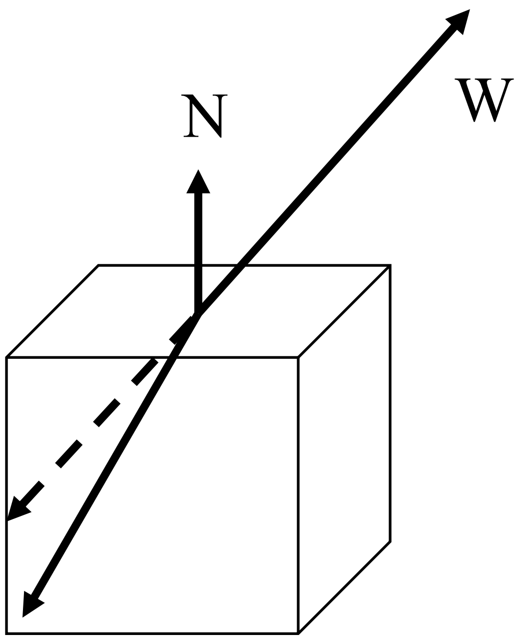


The refraction ray is:

$$T = \left(\frac{\eta_i}{\eta_t}\cos(\theta_i) - \cos(\theta_t)\right)N - \frac{\eta_i}{\eta_t}W$$

Practical Refraction: Solids

- When light enters a solid glass object?



$\eta_{air} = 1.0$

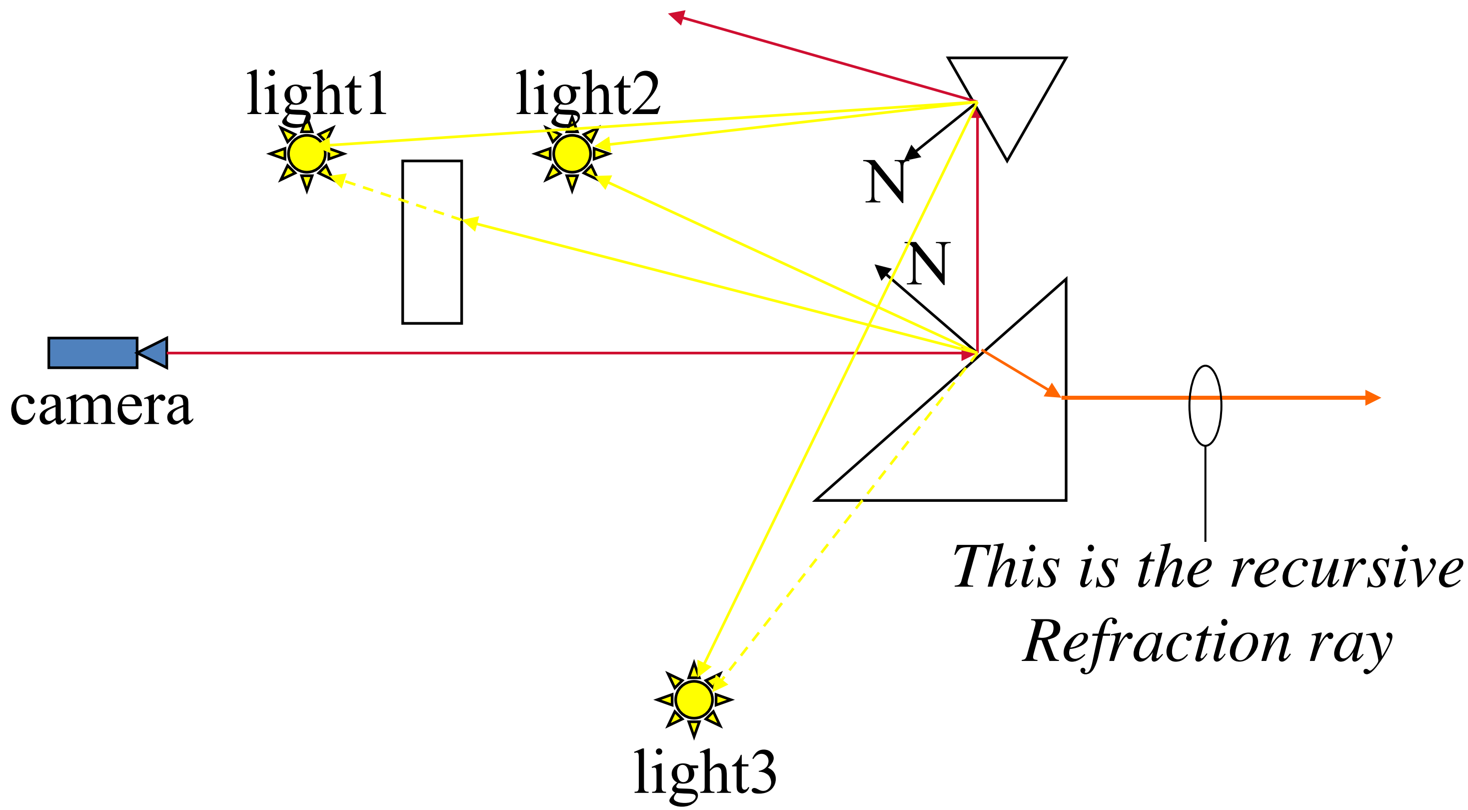
$\eta_{glass} = 1.5$

Theta i	Sin	mu	Theta r
0	0.00	0.67	0.00
10	0.17	0.67	6.67
20	0.34	0.67	13.33
30	0.50	0.67	20.00
40	0.64	0.67	26.67
50	0.77	0.67	33.33
60	0.87	0.67	40.00
70	0.94	0.67	46.67
80	0.98	0.67	53.33
90	1.00	0.67	60.00

$$\theta_r = \sin^{-1}\left(\frac{\eta_i}{\eta_r}\sin(\theta_i)\right) = \sin^{-1}\left(0.67 \cdot \sin(\theta_i)\right)$$

More Recursion

- This changes ray tracing from tail-recursion to double-recursion…



Practical Refraction: Surfaces

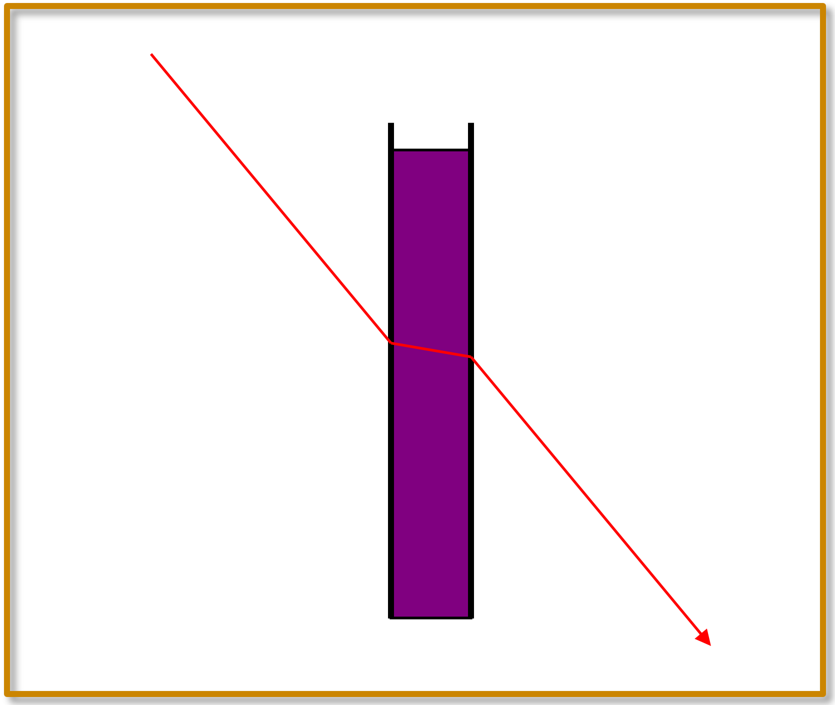
- What happens as it passes *through* a solid or surface?

$$\sin\theta_1 = \frac{\eta_i}{\eta_r}\sin\theta_i$$

$$\sin\theta_i = \frac{\eta_r}{\eta_i}\sin\theta_2$$

$$\sin\theta_1 = \frac{\eta_i}{\eta_r}\frac{\eta_r}{\eta_i}\sin\theta_2$$

$$= \sin\theta_2$$



➢ Overall effect: *displacement* of the incident vector

Note: this assumes the two surfaces of the solid are coplanar!

Refraction - No Trigonometry.

First Constraint: Snells Law

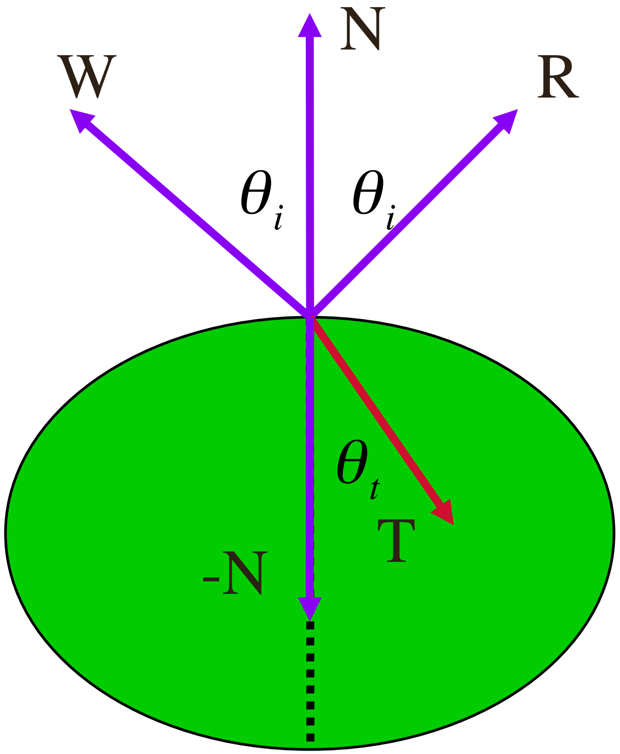
$$T = \alpha W + \beta N$$

$$\sin(\theta_i)^2 \mu^2 = \sin(\theta_t)^2 \quad \mu = \frac{\mu_i}{\mu_t}$$

$$\left(1 - \cos(\theta_i)^2\right)\mu^2 = 1 - \cos(\theta_t)^2$$

$$\left(1 - (W \cdot N)^2\right)\mu^2 = 1 - (-N \cdot T)^2$$

$$\left(1 - (W \cdot N)^2\right)\mu^2 = 1 - \left(-N \cdot (\alpha W + \beta N)\right)^2$$



Refraction - No Trigonometry

Second Constraint: Refraction ray is unit length.

$$T \cdot T = \left(\alpha W + \beta N\right) \cdot \left(\alpha W + \beta N\right) = 1$$

$$= \alpha^2 + 2\alpha\beta\left(W \cdot N\right) + \beta^2 = 1$$

Two quadratic equations in two unknowns.

Solving is a bit involved, …

Here is the answer.

$$\alpha = -\mu \quad \beta = \mu\left(W \cdot N\right) - \sqrt{1 - \mu^2 + \mu^2\left(W \cdot N\right)^2}$$

A Wonderful Real Example

AAPT High School Physics Photo Contest (sample picture)

First Place - Contrived (2009)

Title: **Where Sand Meets Sea**

Student: **Kelsey Rose Weber**

School: **Wildwood School, Los Angeles, California**

Teacher: **Tengiz Bibilashvili**

This photo was contrived by placing a transparent sphere against the beach horizon. By matching the refraction from the sphere with the point where the shoreline and skyline meet, this photo demonstrates the physics of refraction. By means of refraction, lenses form an image. The glass sphere in this photo acted as a lens causing the inverted image. This photo was taken at the Venice beach in Los Angeles, California and shows the beauty of combining physics with ones own natural surroundings.

https://physicsb-2009-10.wikispaces.com

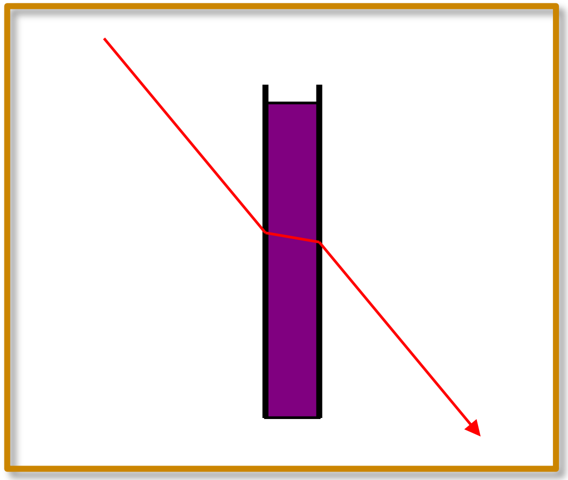
Yes, refraction typically makes everthing upside down and backwards.

Refraction and Polygons

- It is entirely possible to implement refraction through complex solid models defined by polygons.
- But! Doing so requires the following:
  - Models must be complete: no holes!
  - All faces (triangles) must be tagged to a solid.
    - Needed to find where refraction ray exits the solid.
- There is a simpler special case
  - Thin faces with parallel sides (next slide).

Special Case: Thin Faces

- Consider entrance and exit
  - The are parallel (see picture)
- Refraction vectors
  - Pass through at a shifted angle
  - But exit in the same direction
- Result is an offset only
  - Offset depends on index of refraction
  - Offset depends on the thickness of the face



Illuminated Spheres with Relfection and Refraction: Scene 1

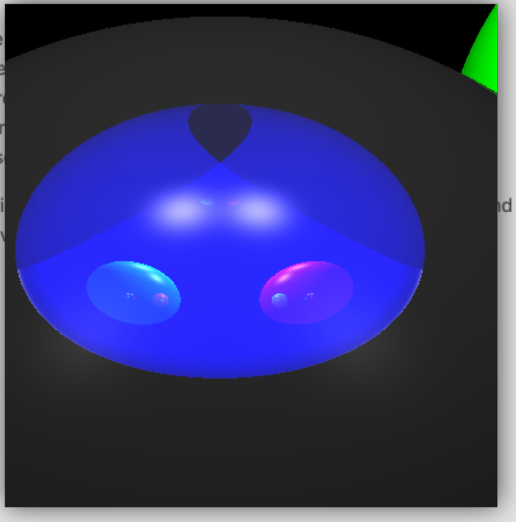
Ross Beveridge, November 10, 2020

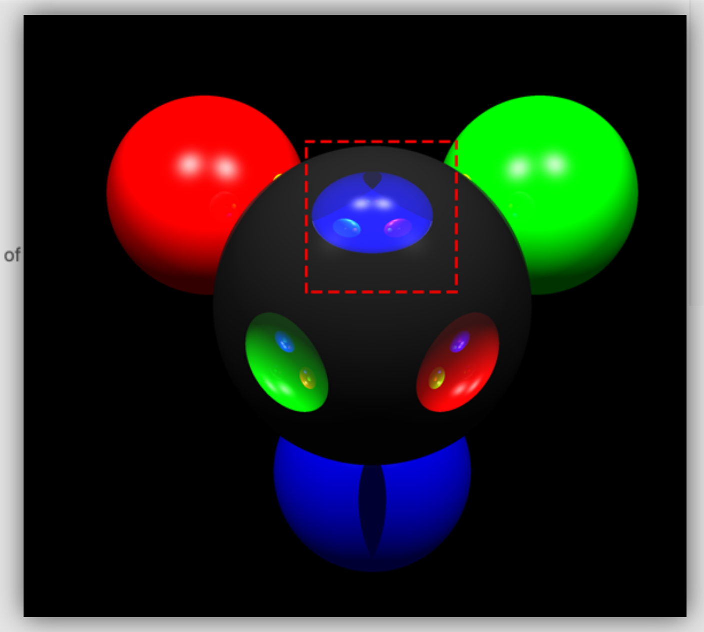
This notebook is a rather complete illustration of many key concepts in CS 410 pertaining to Ray Tracing. This example consists of one semi-transparent sphere partially occluding 3 brightly colored spheres forming a triangle pattern.

The general concpets illustrated here include:

- A camera object/model fully specifying how a camera views a 3D scene.
- A ray object defined by a point of origination and a direction.
- A scene consisting of multiple 3D objects, more specifically spheres.
- Materials used to specify how light interacts with an object's surface.
- Point light sources
- A SageMath enabled 3D visualization of the
- Code to efficiently detect ray sphere interse
- Code to support recursive ray tracing with r
- Code to shoot a refractioh ray through a ser
- Code to render scenes at user specified res

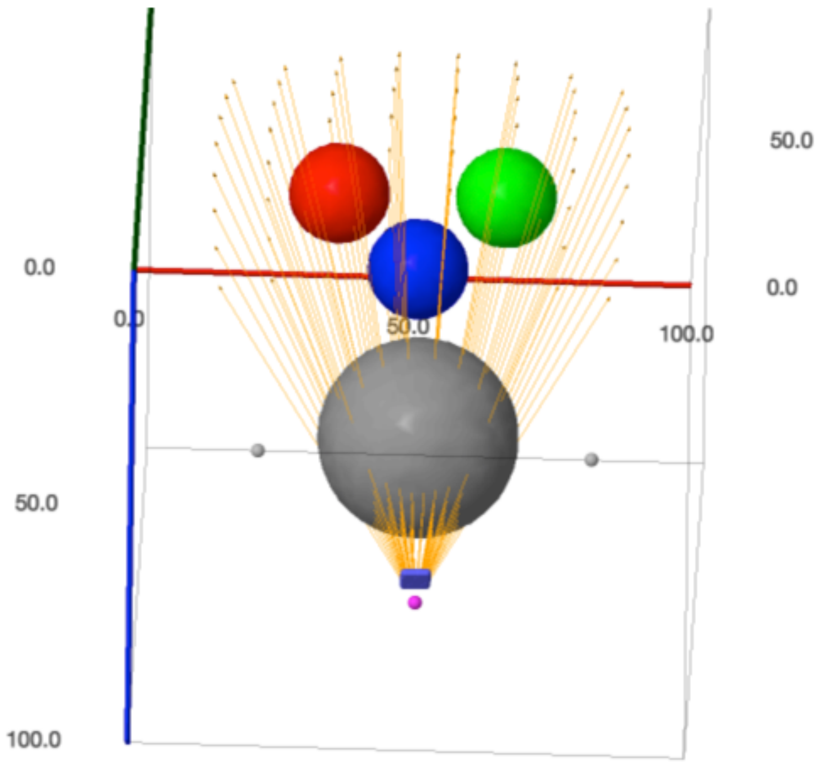
This notebook should be used to study and experi ... nd how all of
optimized. However, it is compact enough to fit ev

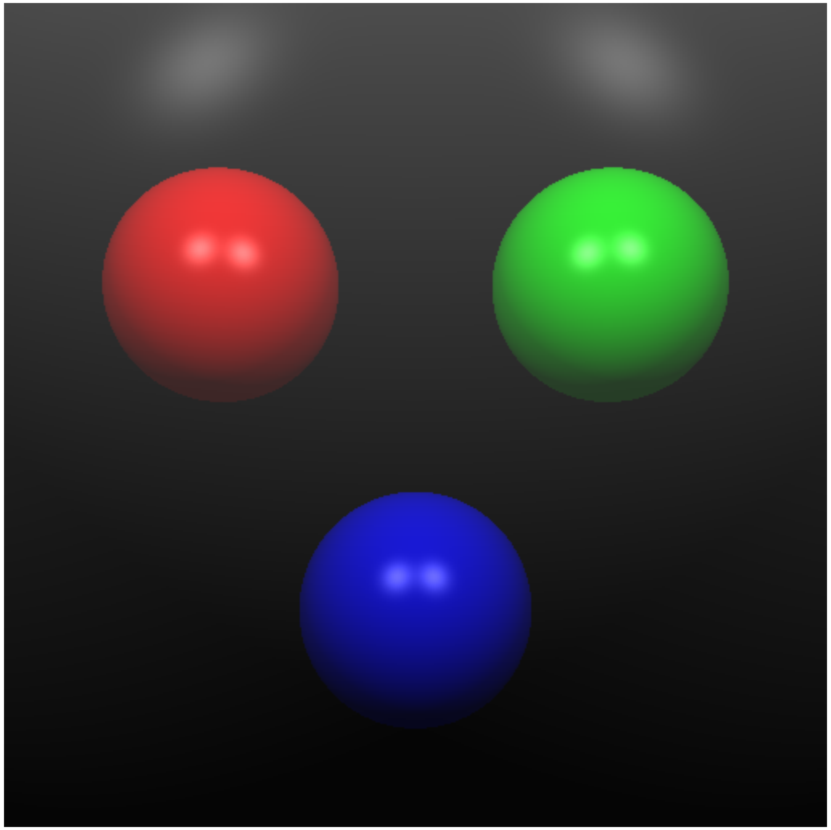




Building A Scene Example 1

- One semi-transparent sphere with eta 1.0
- View three colored spheres behind.





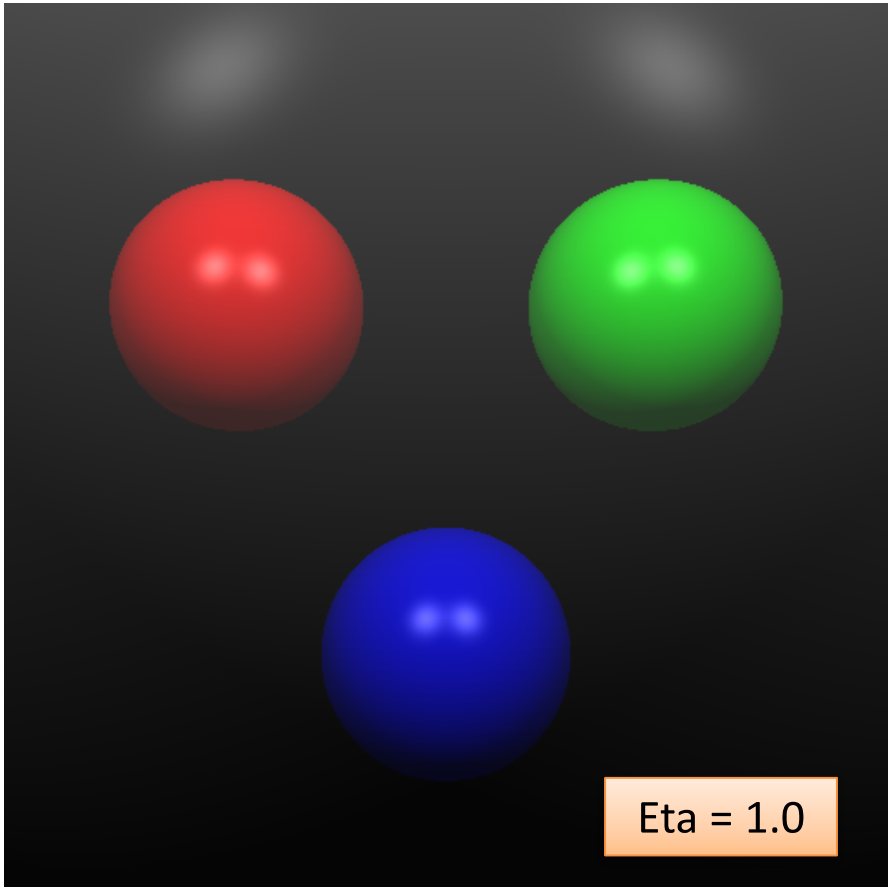
About Materials

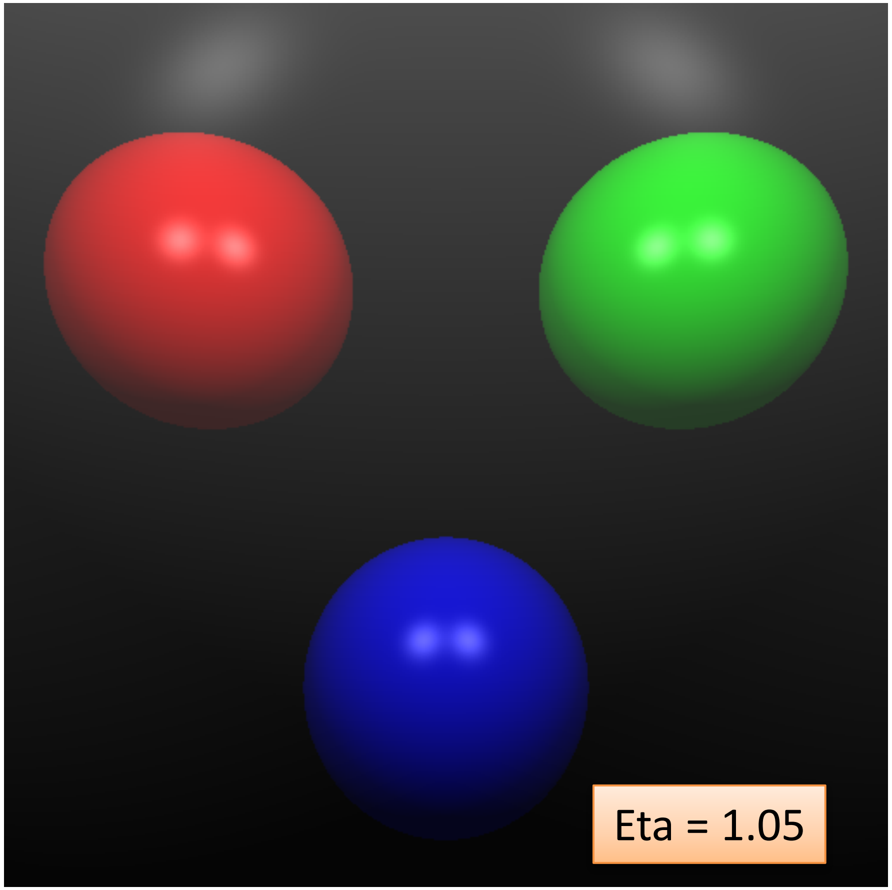
```
class Material :
    def __init__(self, a, d, s, r, o, spow, eta) :
        self.ka   = np.array(a)
        self.kd   = np.array(d)
        self.ks   = np.array(s)
        self.kr   = np.array(r)
        self.ko   = np.array(o)
        self.spow = spow
        self.eta  = eta
```

- ka: the red, green and blue coefficients for ambient illumination
- kd: the red, green and blue coefficients for diffuse illumination
- ks: the red, green and blue coefficients for specular illumination
- spow: the exponent used to control the apparent size of specular highlights
- kr: the red, green and blue attenuation for reflection
- ko: the red, greeen and blue opacity of the material
- eta: the index of refraction for the material: 1.0 for air and typically 1.5 for glass

Small Change to Eta

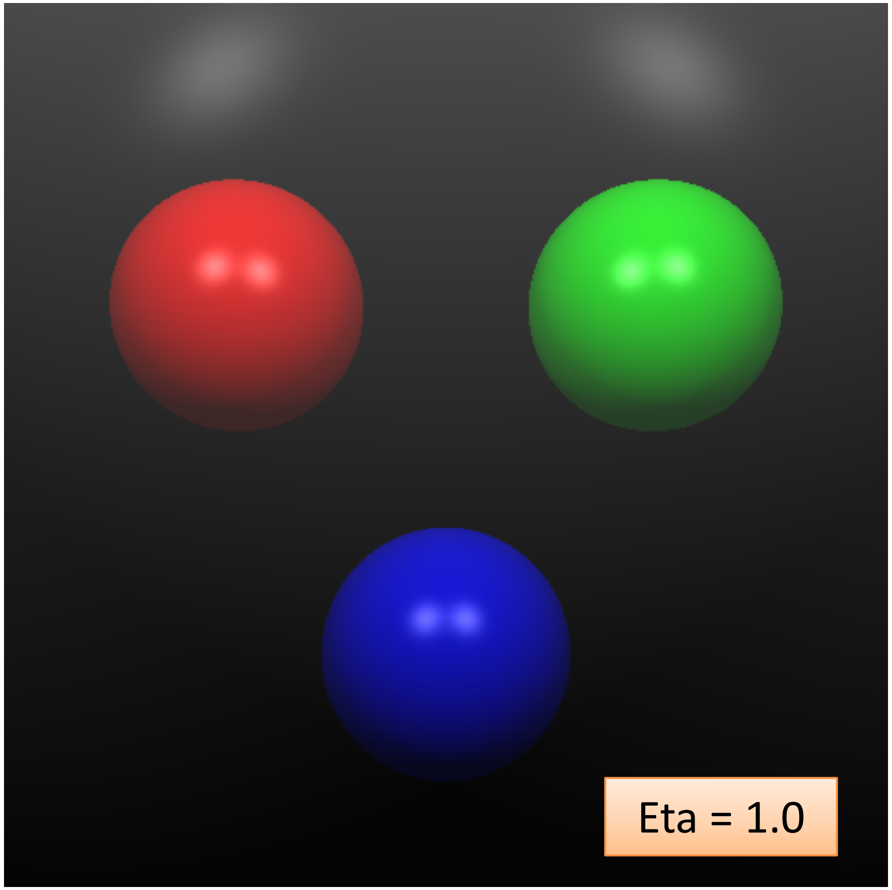
- To see a minor change based upon the index of refraction being set to 1.05 instead of 1.0

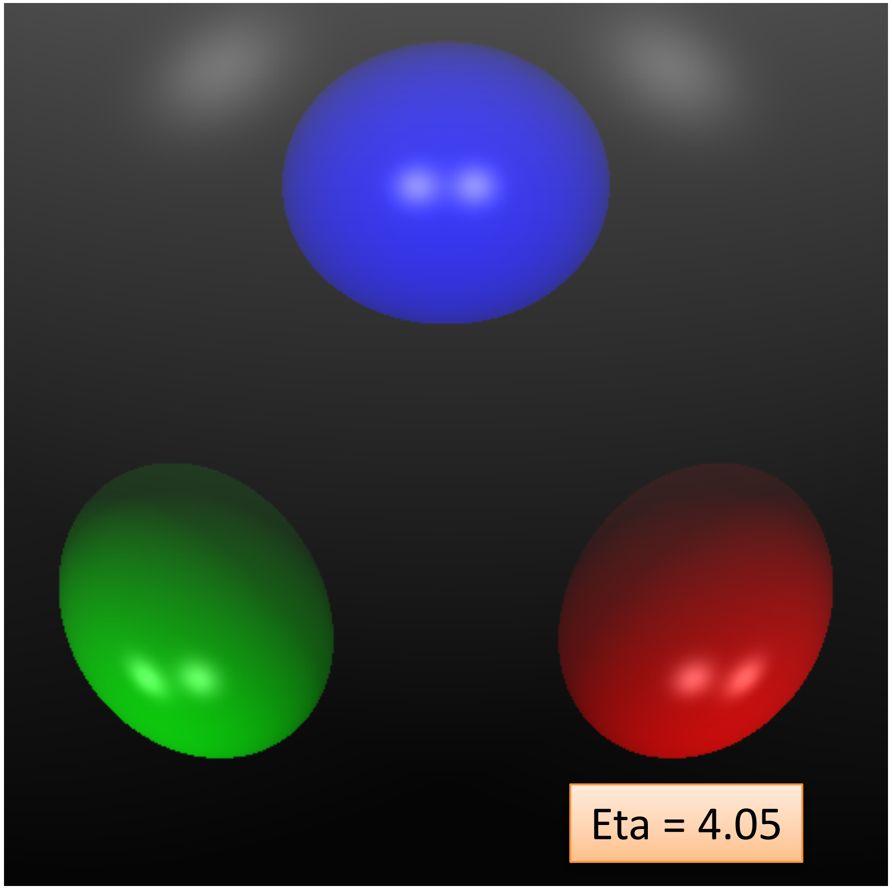




A Large Change in Eta

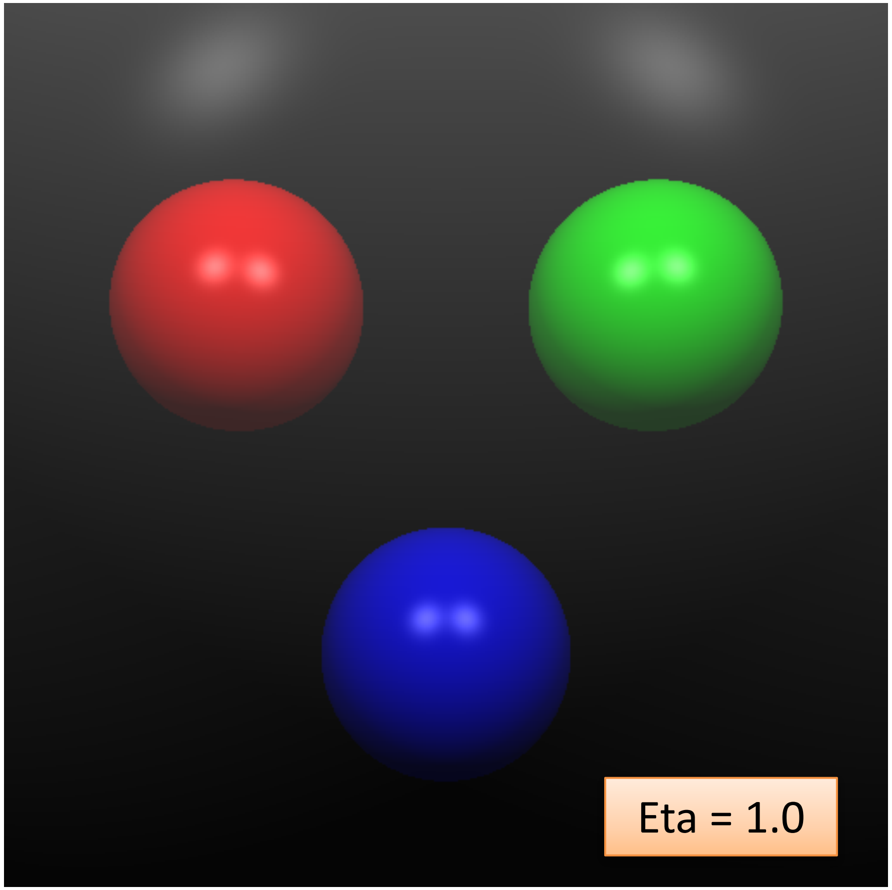
- A bit of graphics science fiction, here is a Germanium sphere with a very high eta.

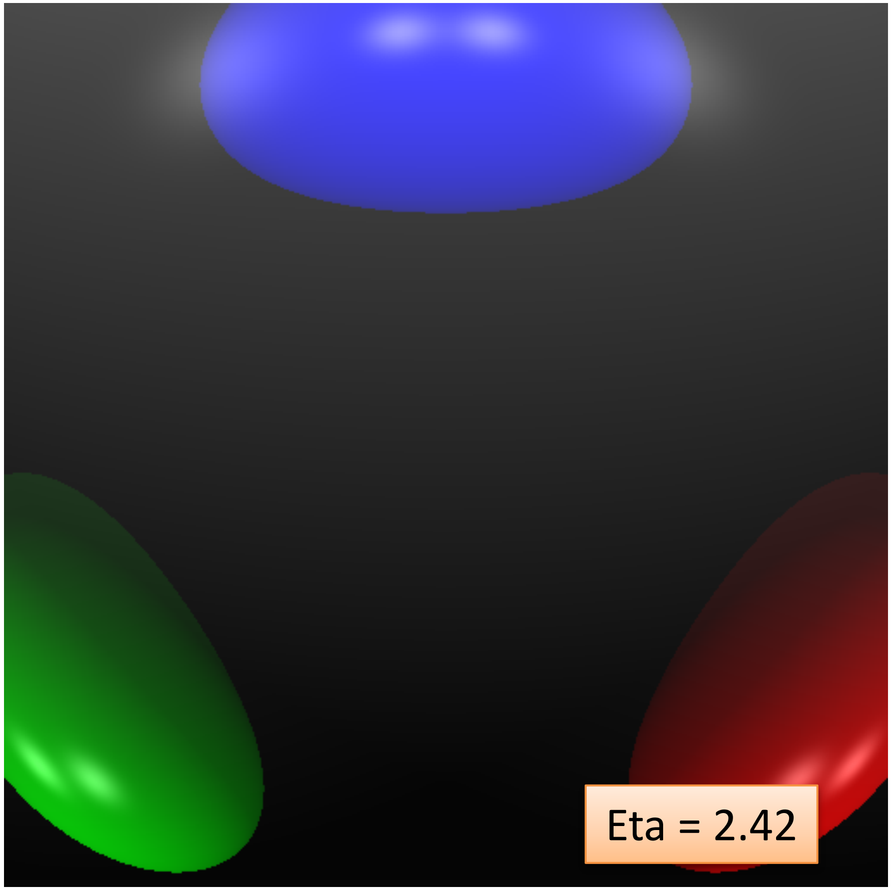




And a Diamond Sphere

- The index of refraction for diamond is higher than glass at 2.42.





Refraction SageMath Code

$$\alpha = -\mu \quad \beta = \mu\left(W \cdot N\right) - \sqrt{1 - \mu^2 + \mu^2\left(W \cdot N\right)^2}$$

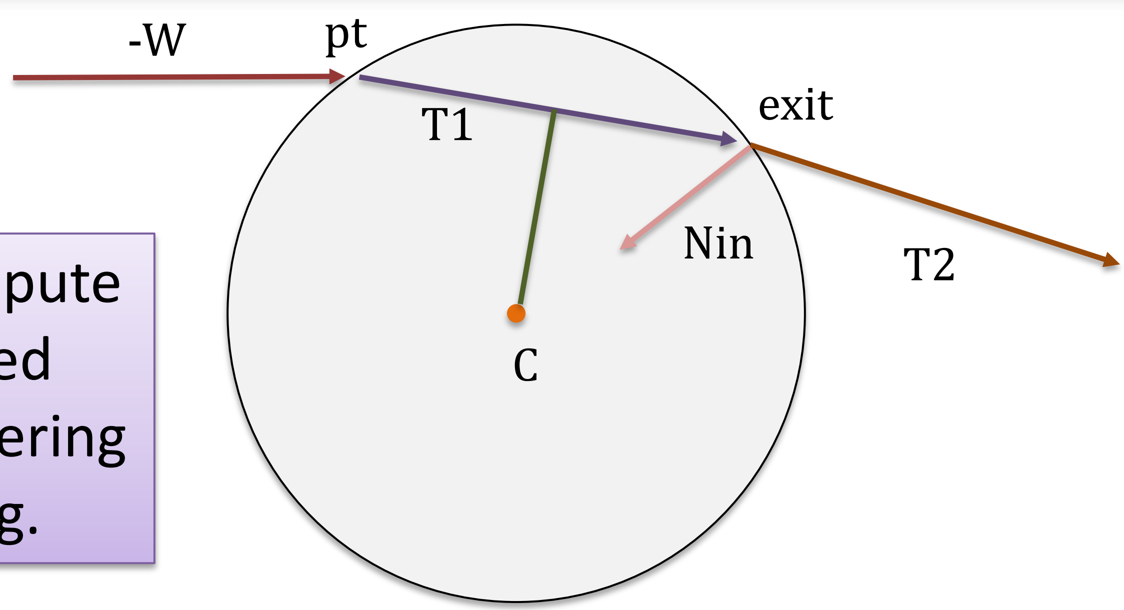
```
def refract_tray(self, W, pt, N, eta1, eta2) :
    etar  = eta1 / eta2
    a     = - etar
    wn    = np.dot(W,N)
    radsq = etar**2 * (wn**2 - 1) + 1
    if (radsq < 0.0) :
        T = np.array([0.0,0.0,0.0])
    else :
        b = (etar * wn) - sqrt(radsq)
        T = a * W + b * N
    return(T)
```

Refraction Code – Exiting the Sphere

```
def refract_exit(self, W, pt, eta_in, eta_out) :
    T1 = self.refract_tray(W, pt, make_unit(pt - self.C), eta_out, eta_in)
    if (sum(T1) == 0.0) :
        return None
    else :
        exit = pt + 2 * np.dot((self.C - pt),T1) * T1
        Nin = make_unit(self.C - exit)
        T2 = self.refract_tray(-T1, exit, Nin, eta_in, eta_out)
        refR = Ray(exit, T2)
        return refR
```

Here is code to find the exit point on the sphere.

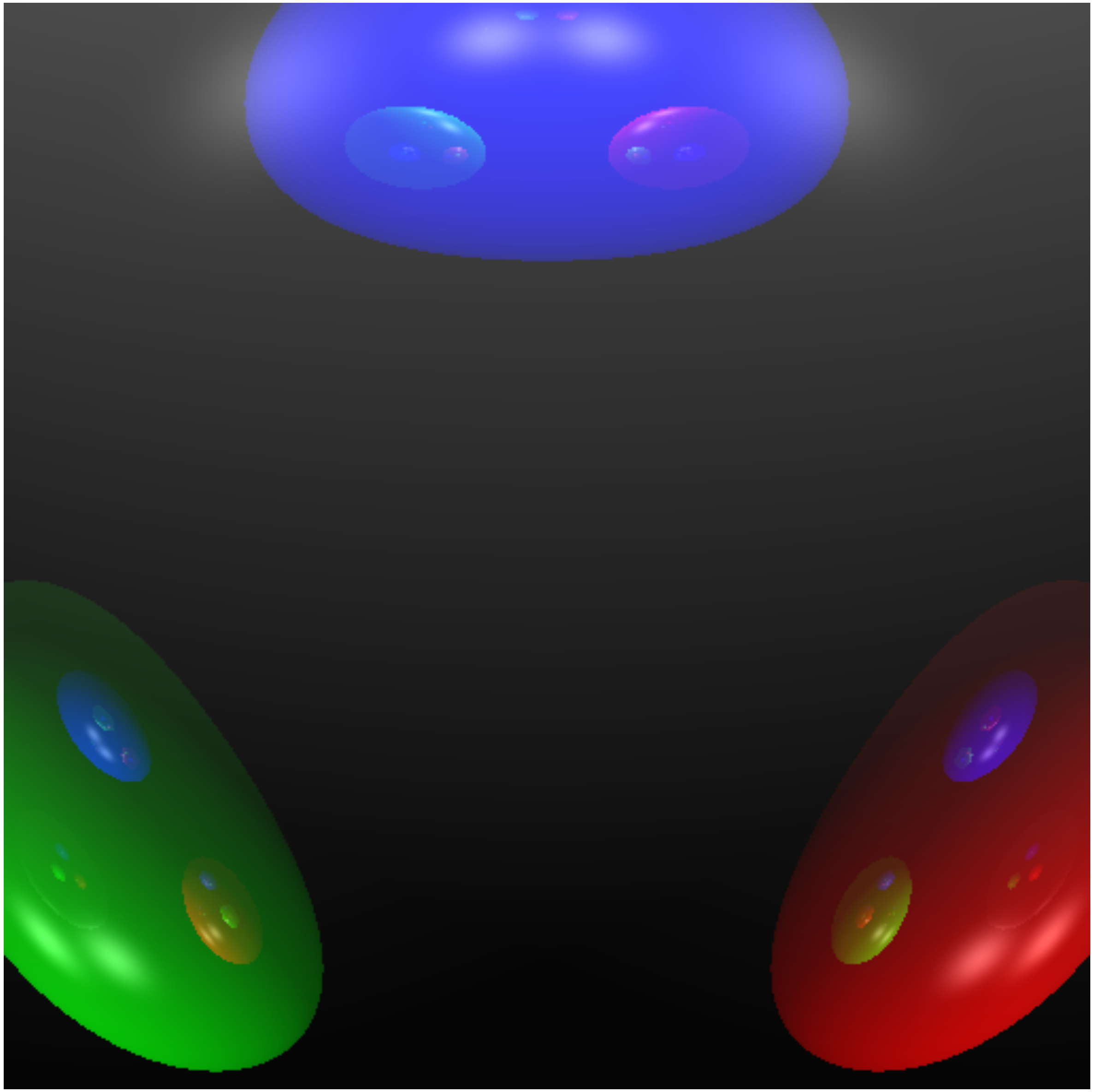
Note the code to compute a refraction ray is called twice. Once upon entering and once upon leaving.



Now With Recursion at 6

This image is created using the same configuration (Diamond) as the previous.

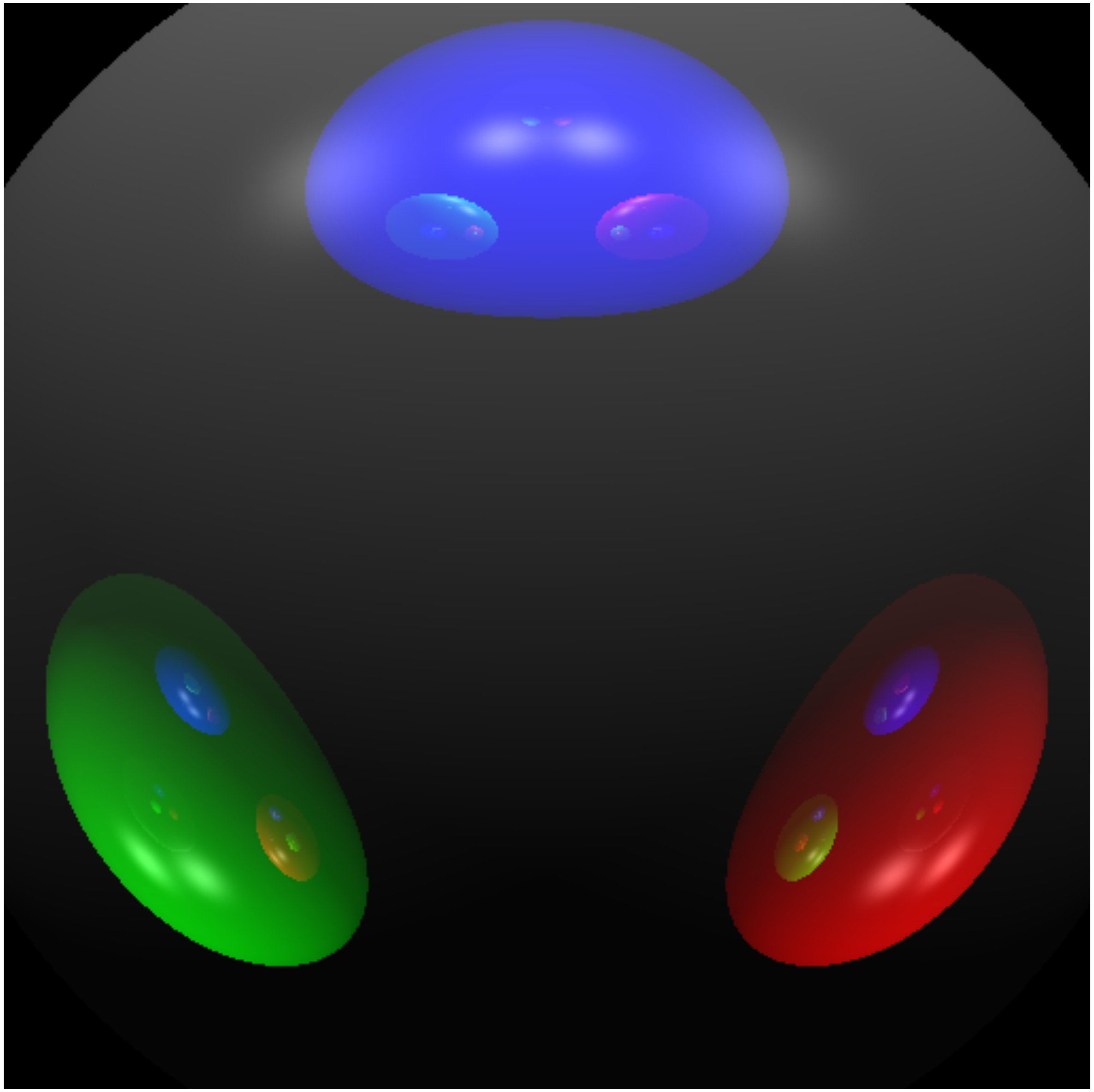
The only change is recursion level is now set to 6



.. and expanding field of view

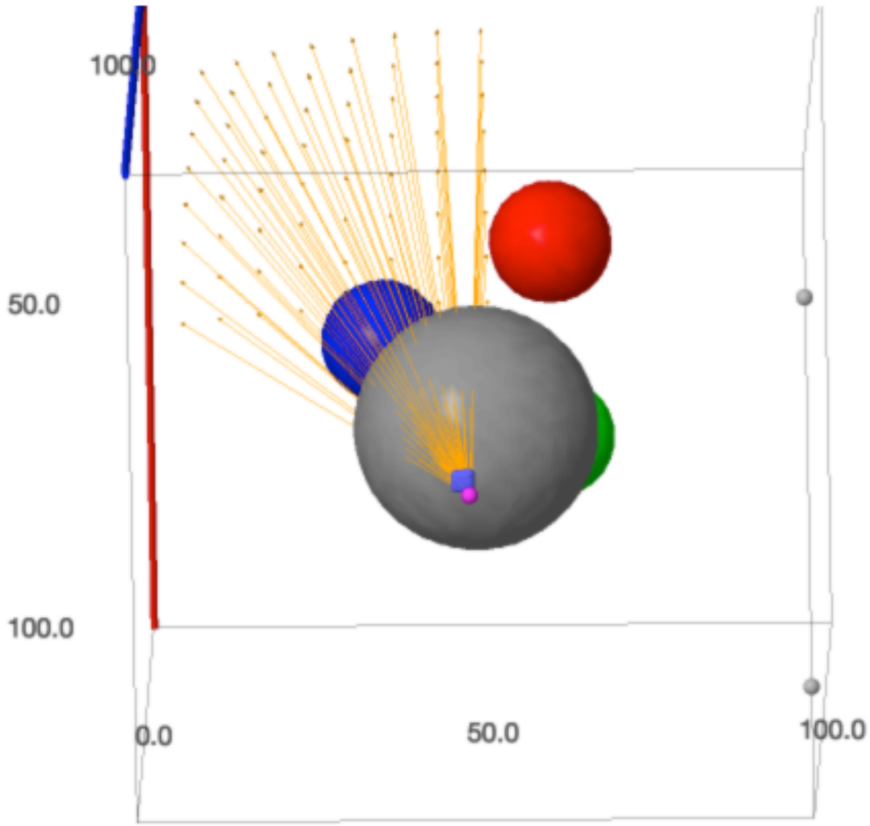
This image is created using the same configuration (Diamond) as the previous.

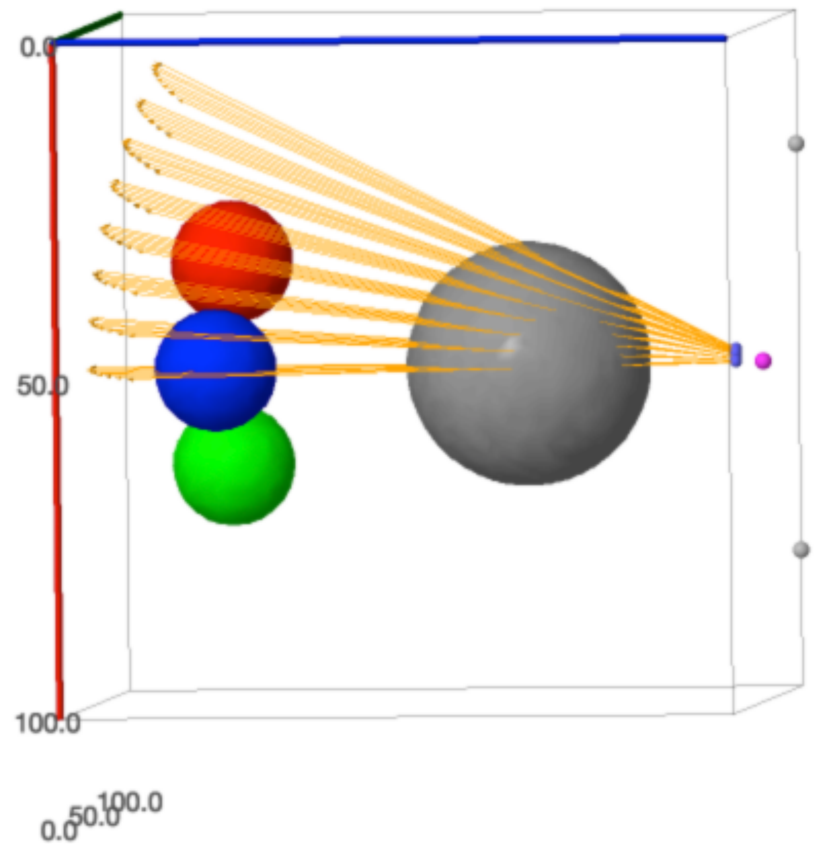
The only change is distance to the near clipping plane is 4 instead of 5



To Show a Quarter of the Image

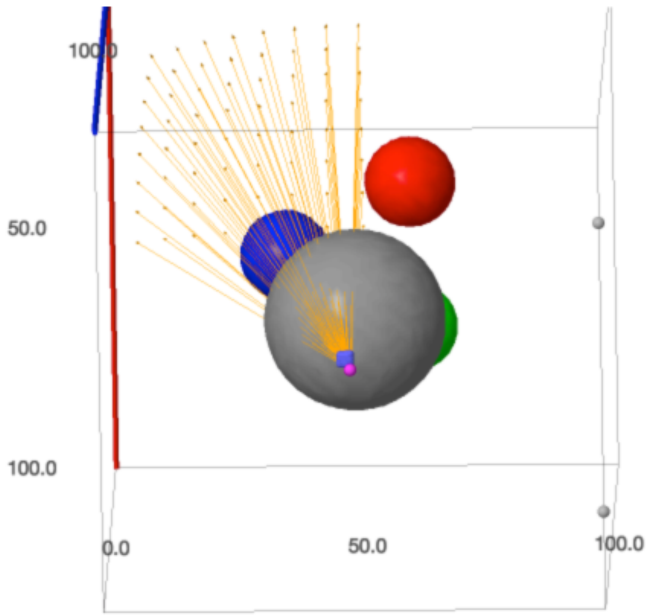
For this example the bounds run -2 to 0 on both horizontal and vertical.

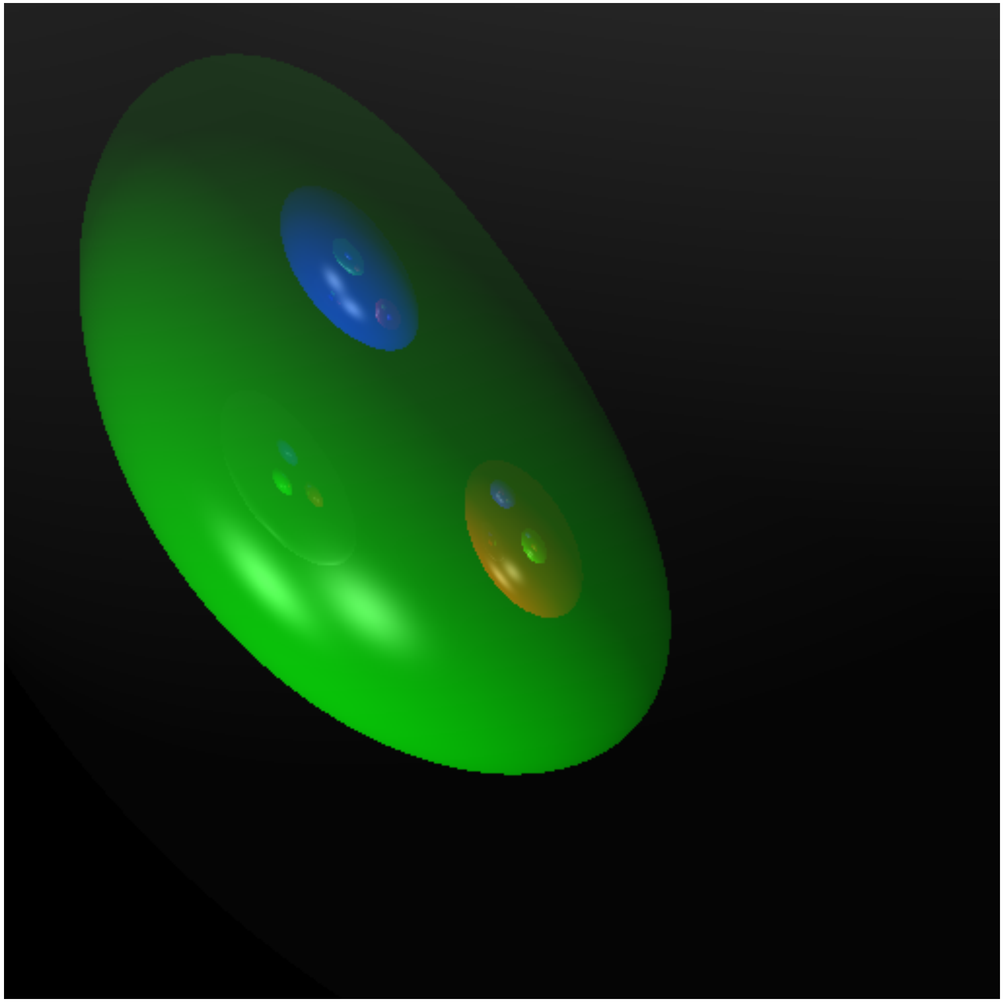




To Show a Quarter of the Image

For this example the bounds run -2 to 0 on both horizontal and vertical.

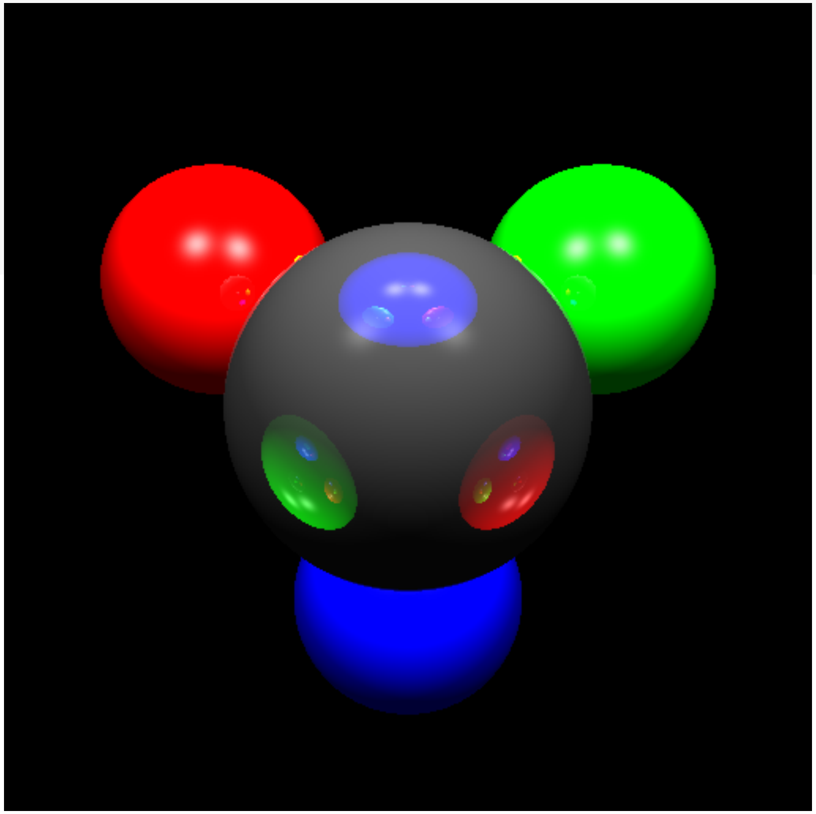




If you understand why the green sphere is being rendered in this view then you are a long way towards understanding refraction.

Now to the “default” scene

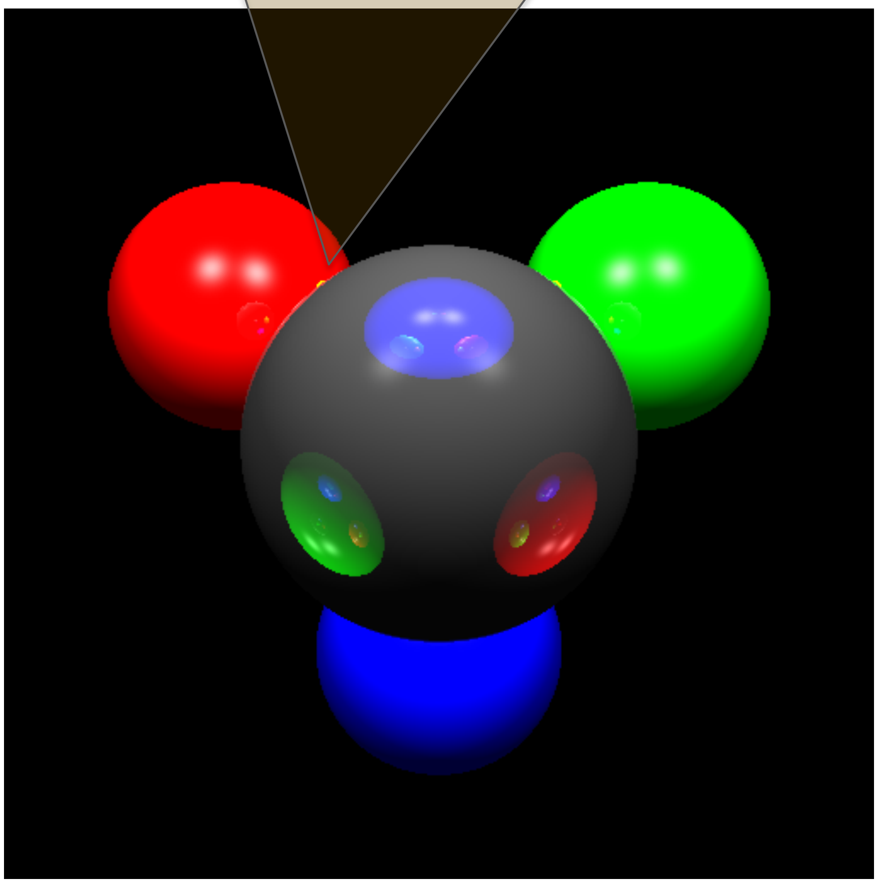
```
cam1 = Camera((50,50,100),(50,50,10),(0,1,0),(-2.0,2.0,-2.0,2.0),-5,-100,8,8)
cam2 = copy(cam1);
cam2.width  = 512
cam2.height = 512

mats = [Material((0.2, 0.2, 0.2),(0.6, 0.6, 0.6),(0.5, 0.5, 0.5),(0.9, 0.9, 0.9),(0.5, 0.5, 0.5), 64, 2.0),
        Material((1.0, 0.0, 0.0),(1.0, 0.0, 0.0),(1.0, 1.0, 1.0),(0.9, 0.9, 0.9),(1.0, 1.0, 1.0), 32, 1.3),
        Material((0.0, 1.0, 0.0),(0.0, 1.0, 0.0),(1.0, 1.0, 1.0),(0.9, 0.9, 0.9),(1.0, 1.0, 1.0), 32, 1.3),
        Material((0.0, 0.0, 1.0),(0.0, 0.0, 1.0),(1.0, 1.0, 1.0),(0.9, 0.9, 0.9),(1.0, 1.0, 1.0), 32, 1.3)]

lgts = [Light((20,100,100),(0.75, 0.75, 0.75)),Light((80,100,100),(0.75, 0.75, 0.75))]
ambi = vector(RR, 3, (0.2, 0.2, 0.2))

objs = [Globe((50,50,50),  9, 0),
        Globe((35,60,20),  9, 1),
        Globe((65,60,20),  9, 2),
        Globe((50,35,20),  9, 3)]

eta_outside = 1.0
trace_depth = 6
```

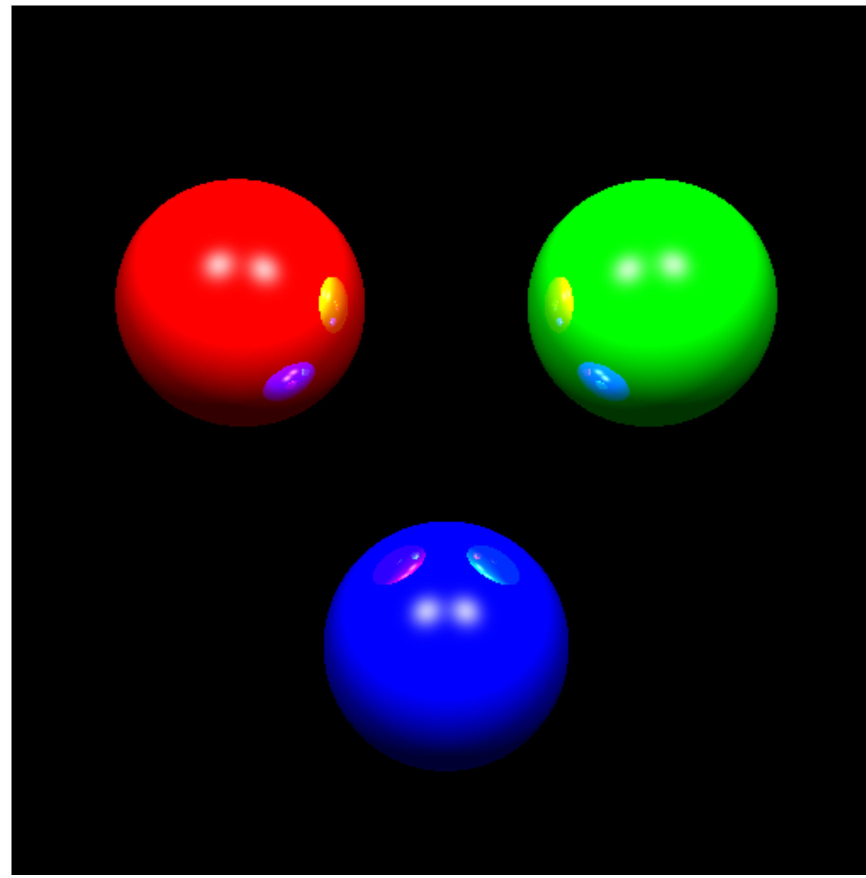


Detail: About The Yellow Pixel

Here is the default scene with the semi-transparent sphere removed.

In the last lecture I was asked about the bit of yellow at the edge of the semi-transparent sphere.





Double Recursion Code

```
def ray_trace(ray, accum, refatt, level) :
    if (ray_find(ray) != None) :
        N   = make_unit(ray.best_pt - ray.best_sph.C)
        mat = mats[ray.best_sph.m]
        pt_illum(ray, N, mat, accum, refatt)
        if (level > 0) :
            flec = np.array([0.0,0.0,0.0])
            Uinv = (-1 * ray.D)
            refR = make_unit((2 * np.dot(N, Uinv) * N) - Uinv)
            ray_trace(Ray(ray.best_pt, refR), flec, mat.kr * refatt, (level - 1))
            for i in range(3) : accum[i] += refatt[i] * mat.ko[i] * flec[i]
        if (level > 0) and (sum(mat.ko) < 3.0) :
            thru = np.array([0.0, 0.0, 0.0])
            fraR = ray.best_sph.refract_exit(-1 * ray.D, ray.best_pt, mat.eta, eta_outside)
            if fraR != None :
                ray_trace(fraR, thru, mat.kr * refatt, (level - 1))
                for i in range(3) : accum[i] += refatt[i] * (1.0 - mat.ko[i]) * thru[i]
    return accum
```

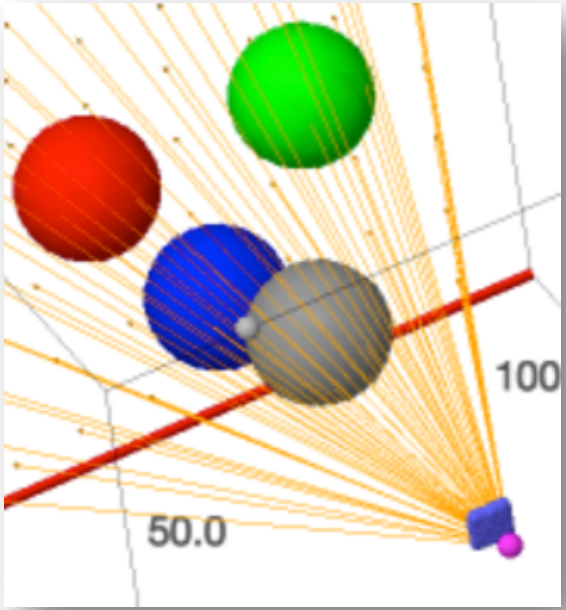
- There are two calls to ray trace
- There are two intermediate accumulation vectors for colors
- The sphere object finds the exit refraction ray
- Transparency is modulated by the `mat.ko` property.

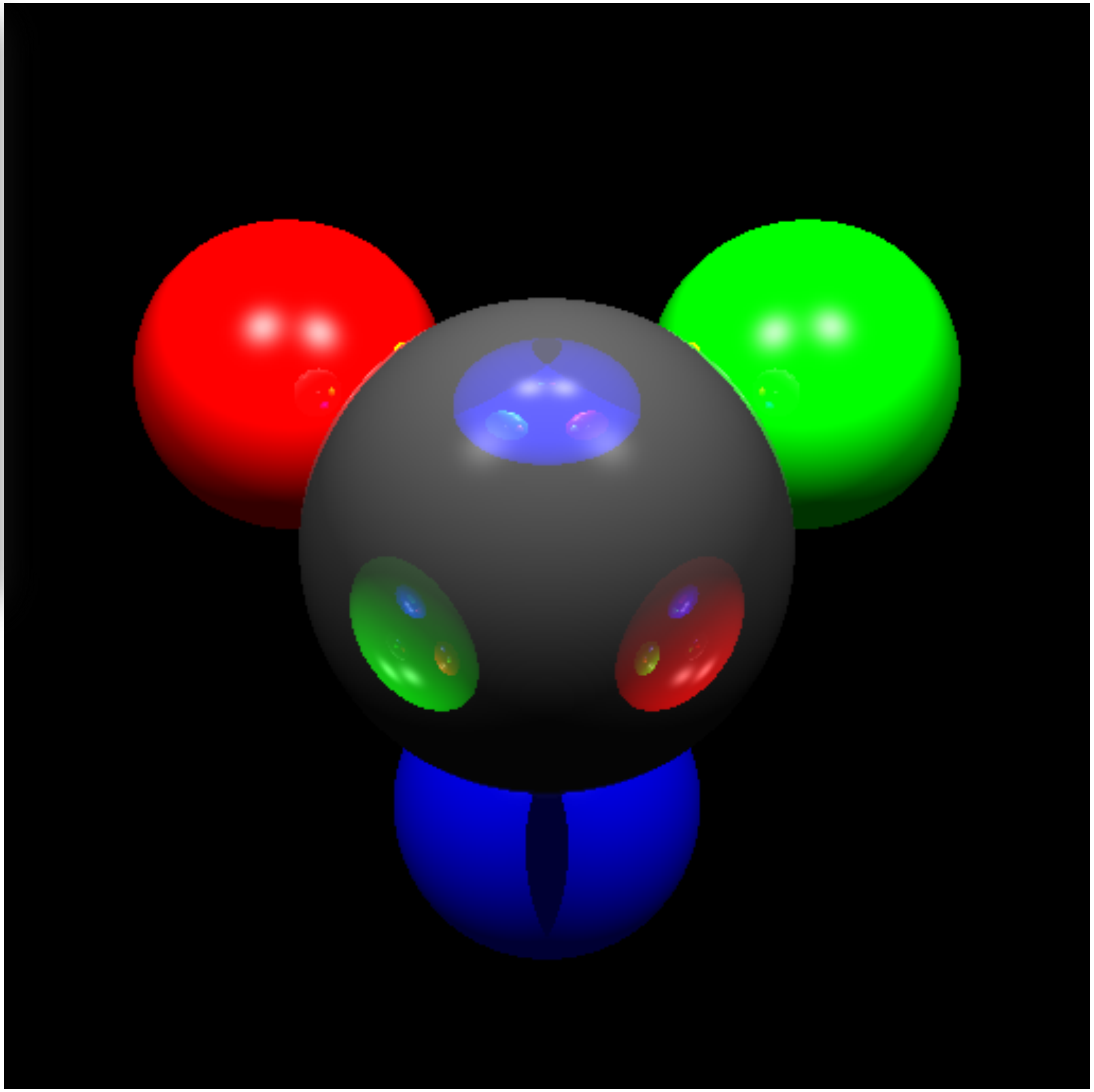
What About Shadows

- It is easy to test whether and object is between the point of interest and light.
- It is harder to ‘dim’ a light – not done here.

```
def shadow(pt, lt) :
    L   = lt.P - pt
    ray = Ray(pt, L)
    dtl = np.dot(L, ray.D)
    for s in objs :
        if ray.sphere_test(s) and ray.best_t < dtl :
            return True
    return False
```

The "default" scene with Shadows





The 3D view above shows how the light source 'sees' the semi-transparent and then blue sphere.

The Complete Package

When you understand every line of code in the Sage Notebook creating this image you are will be in a position to write a truly compelling ray tracer.