Lecture 21: Refraction Take Two

November 14, 2019

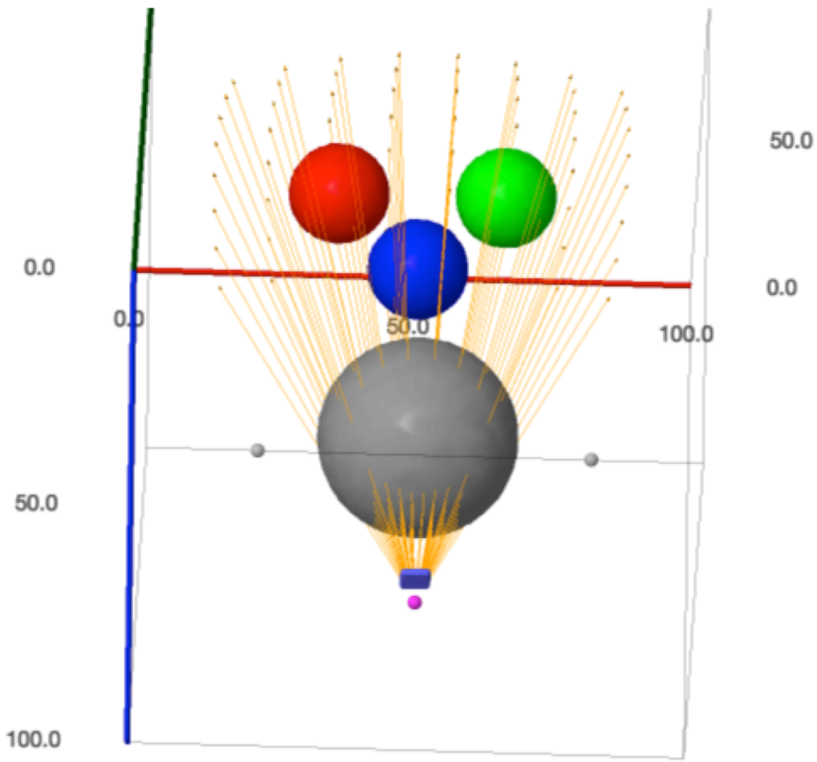
Note that with the instructor's apologies, due to being sick, the lecture notes here accompany a recorded CS 410 lecture given in 2017 which is largely the same as that presented here. The video link is on the CS 410 progress page.

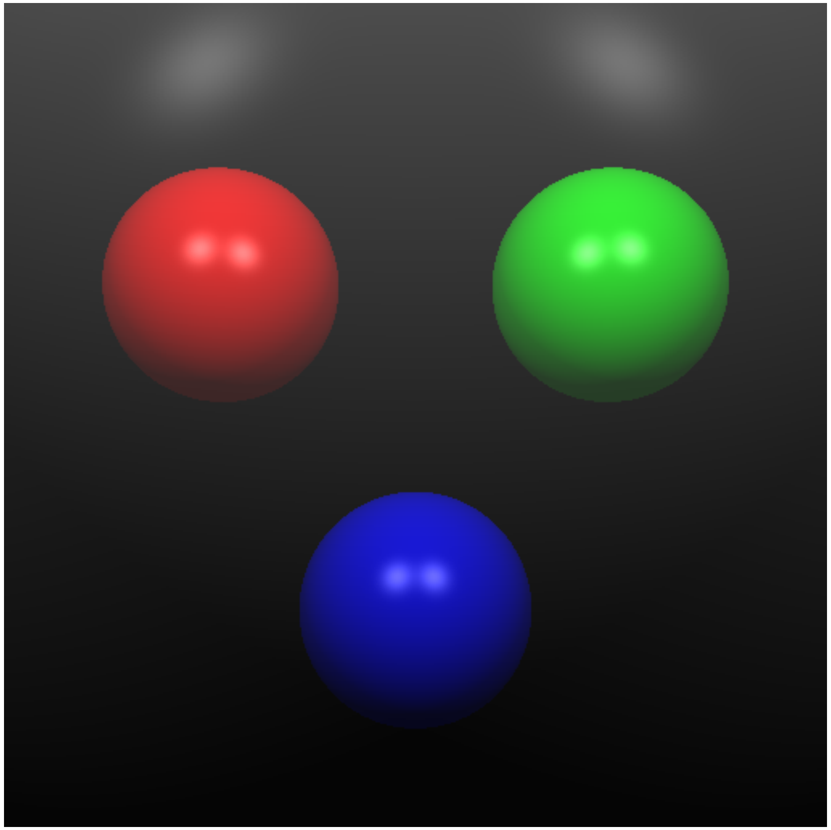
Strategy

- Take the Lecture 21 SageMath Notebook and start building up a scene adding complexity by using the configuration to add refraction, shadows, etc.
- Start with a limiting case where indices of refraction inside and outside are equal.
- Note in the code the outside (air) index of refraction is a global value set initially to 1.0

Building A Scene Example 1

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





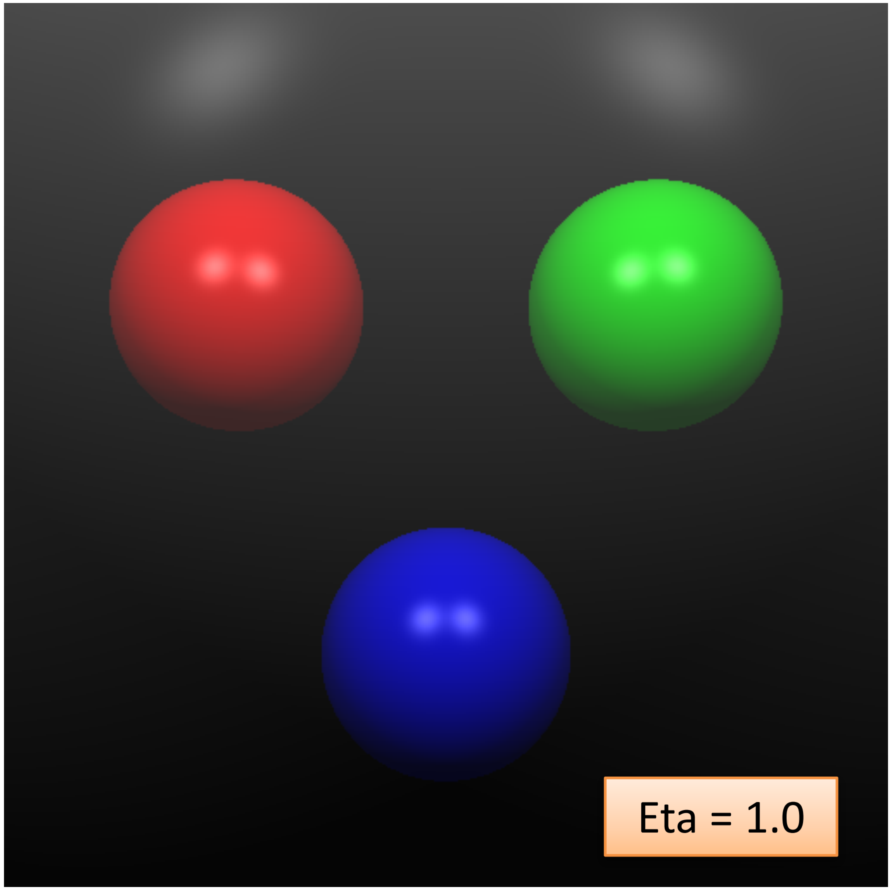
About Materials

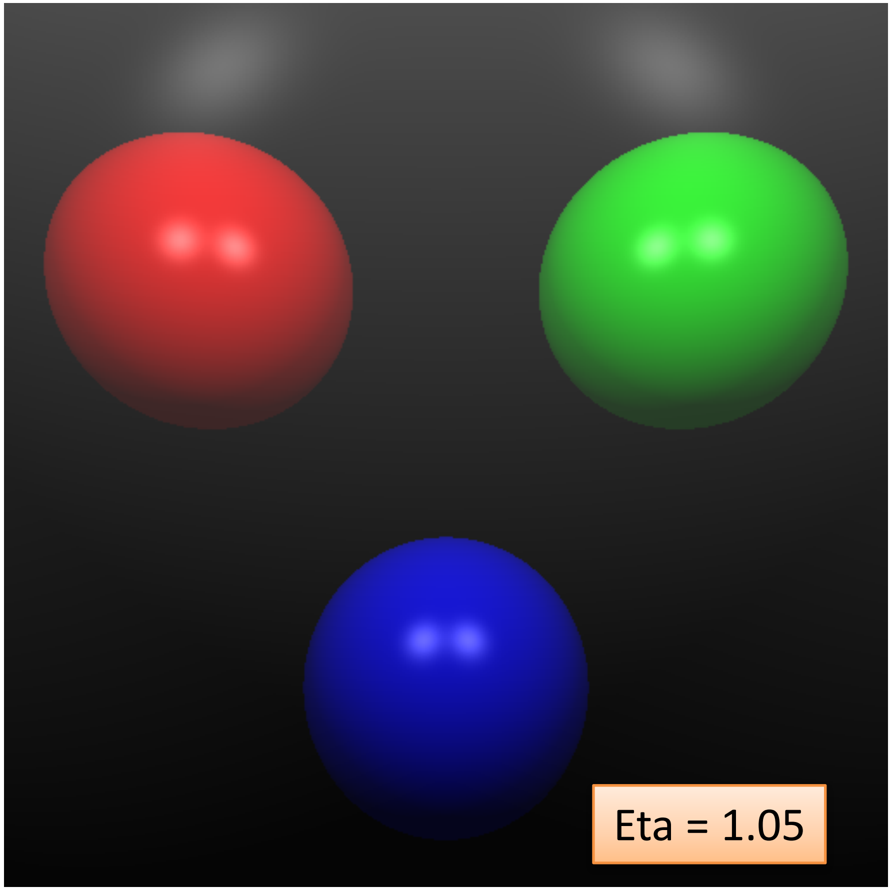
```
class Material :
    def __init__(self, a, d, s, r, o, spow, eta) :
        self.ka   = np.array(a)
        self.kd   = np.array(d)
        self.ks   = np.array(s)
        self.kr   = np.array(r)
        self.ko   = np.array(o)
        self.spow = spow
        self.eta  = eta
```

- 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

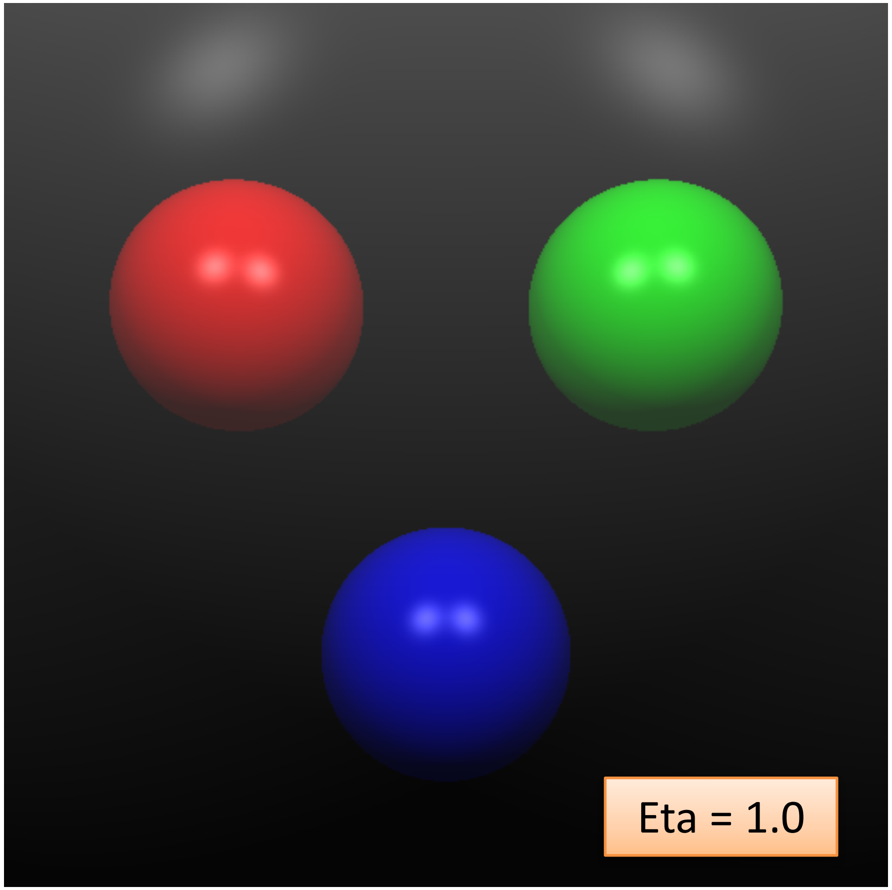
- To see a minor change based upon the index of refraction being set to 1.05 instead of 1.0

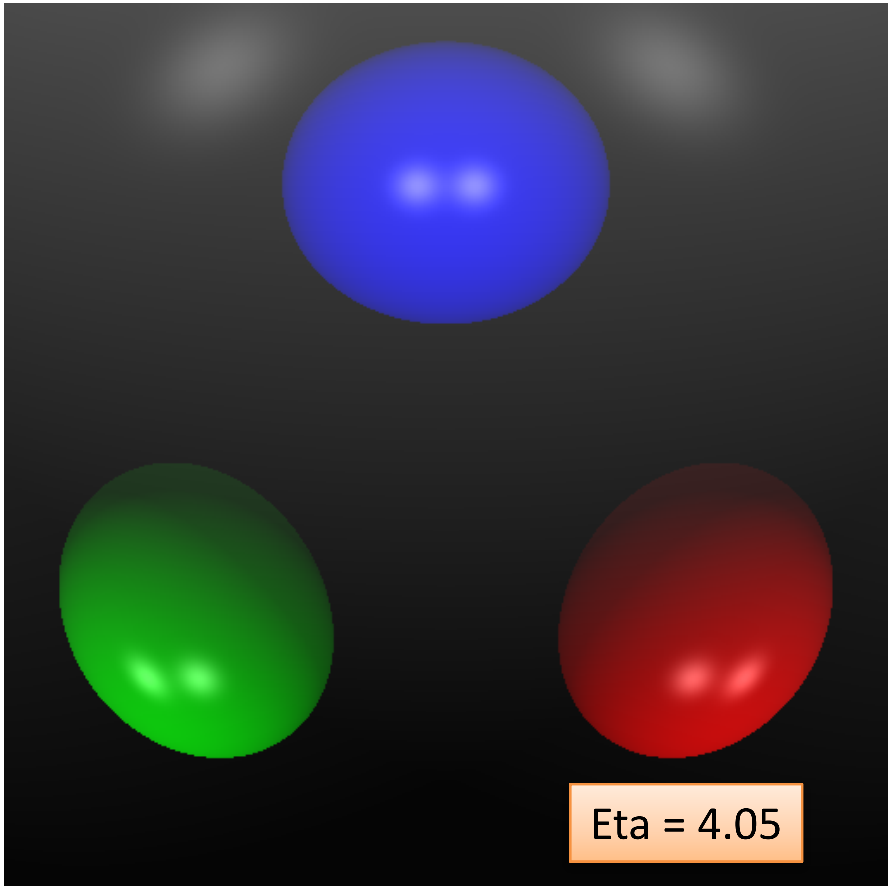




A Large Change in Eta

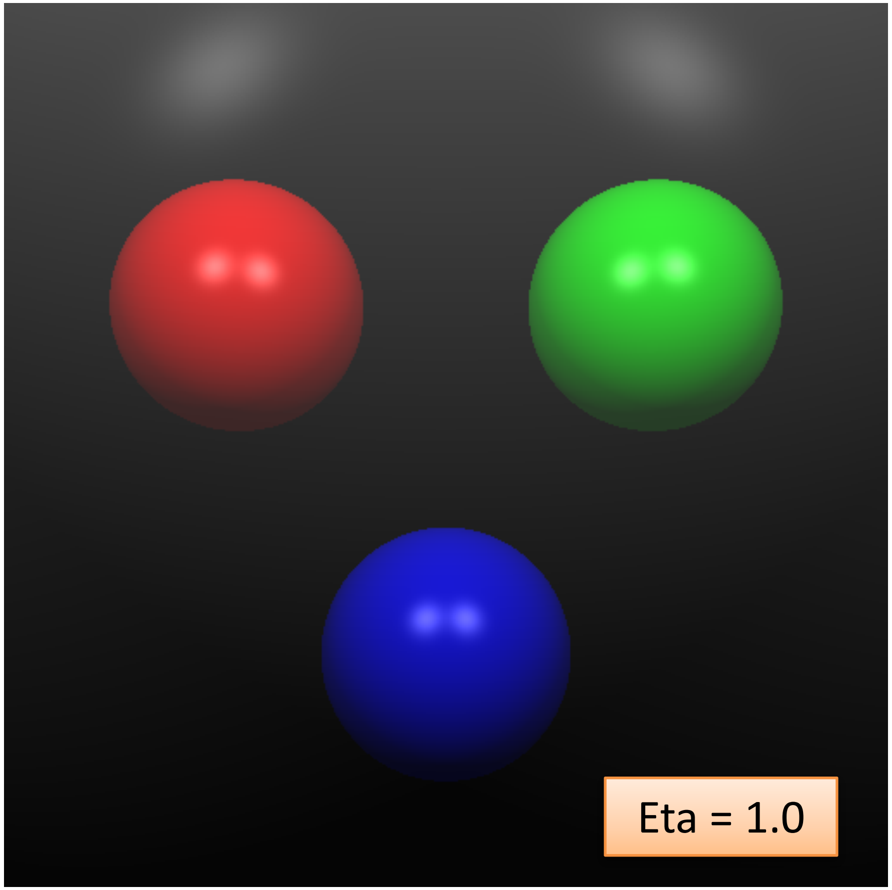
- A bit of graphics science fiction, here is a Germanium sphere with a very high eta.

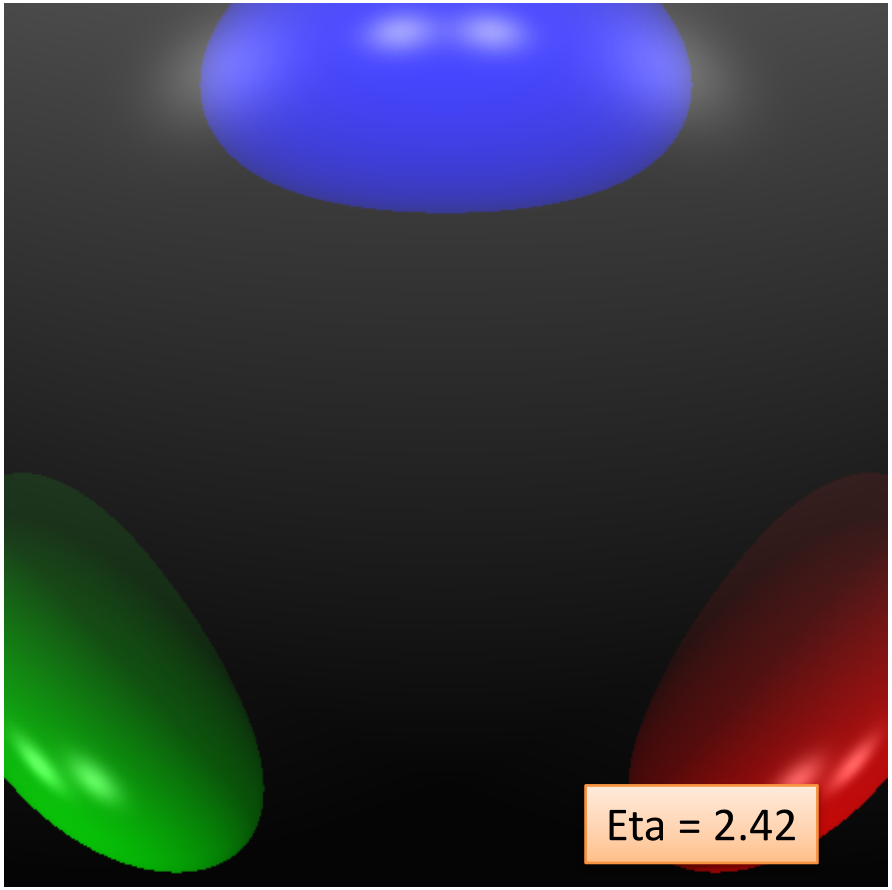




And a Diamond Sphere

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





Refraction Review 1

First Constraint: Snells Law

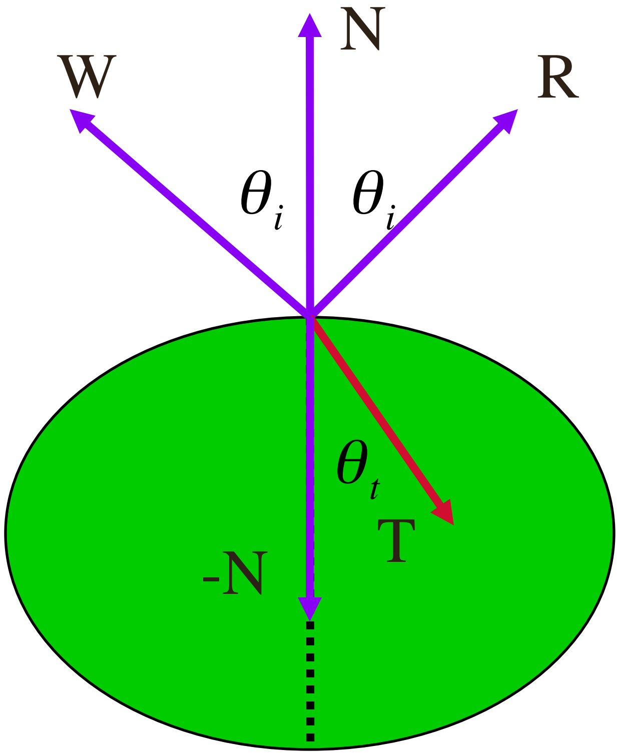
$$T = \alpha W + \beta N$$

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

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

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

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



Refraction Review 2

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.

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

Refraction SageMath Code

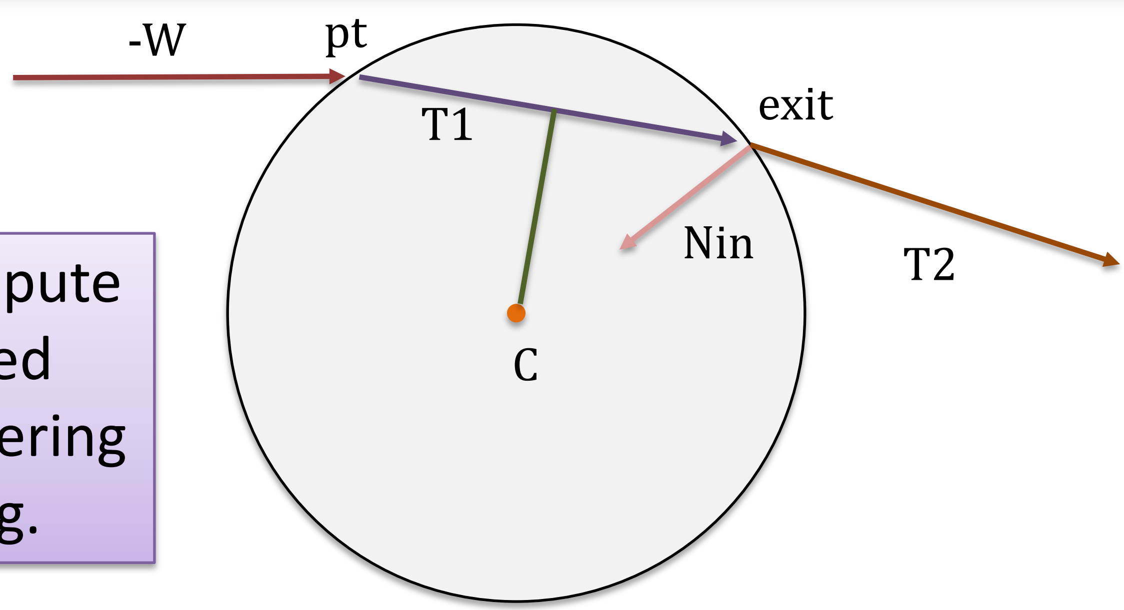
$$\alpha = -\mu \quad \beta = \mu(W \cdot N) - \sqrt{1 - \mu^2 + \mu^2 (W \cdot N)^2}$$

```
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.

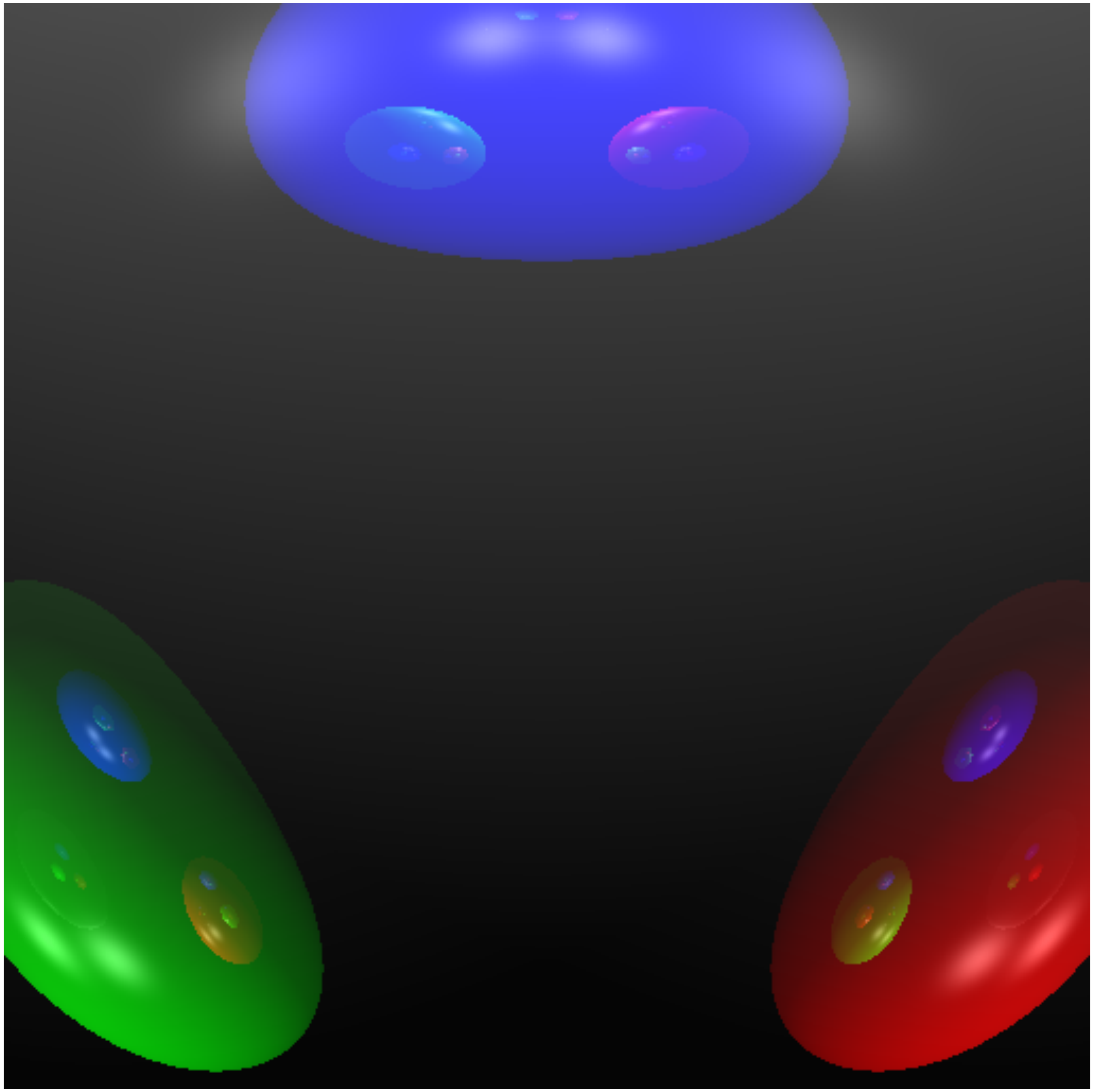


Note the code to compute a refraction ray is called twice. Once upon entering and once upon leaving.

Now With Recursion at 6

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

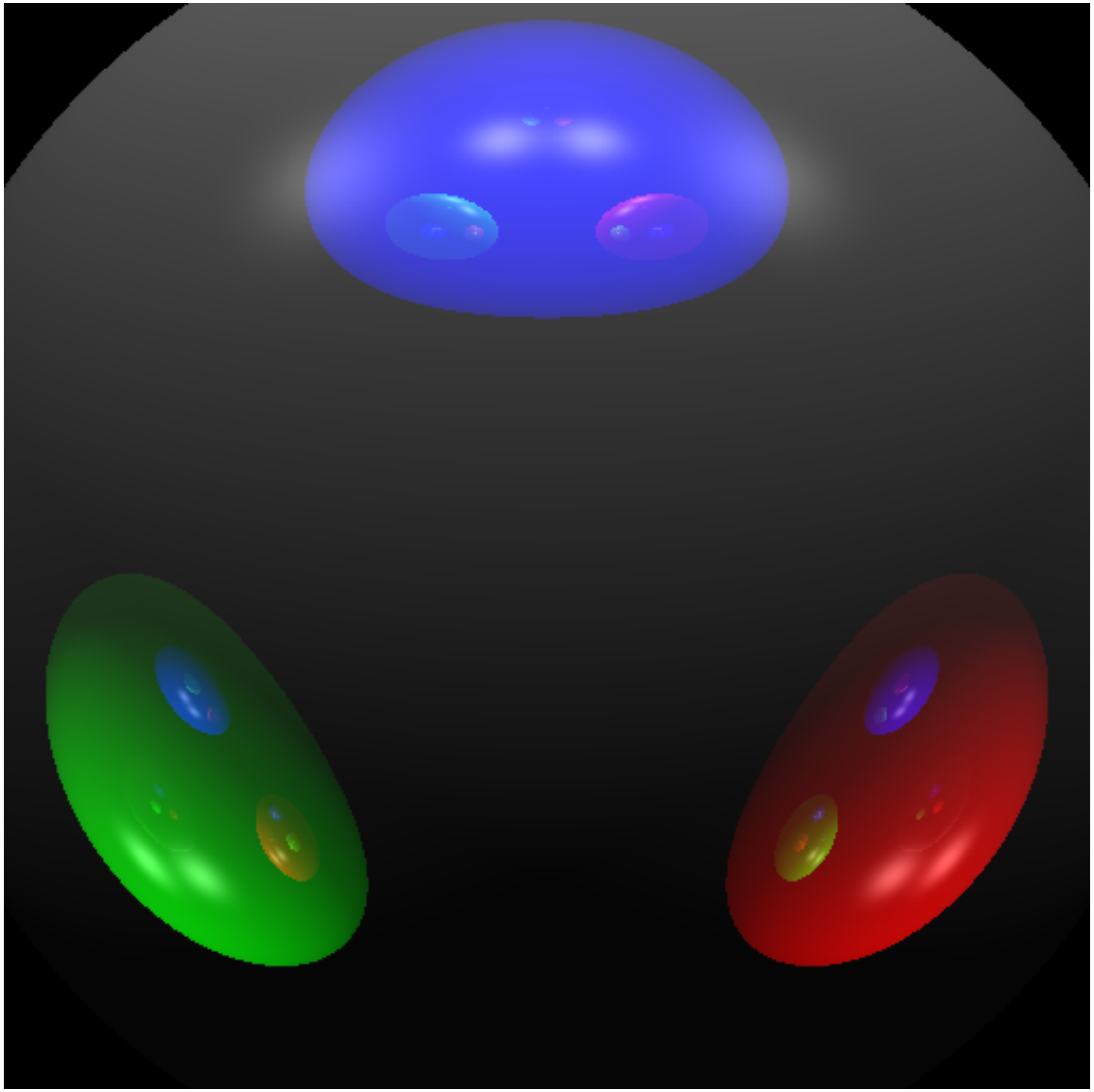
The only change is recursion level is now set to 6



.. and expanding field of view

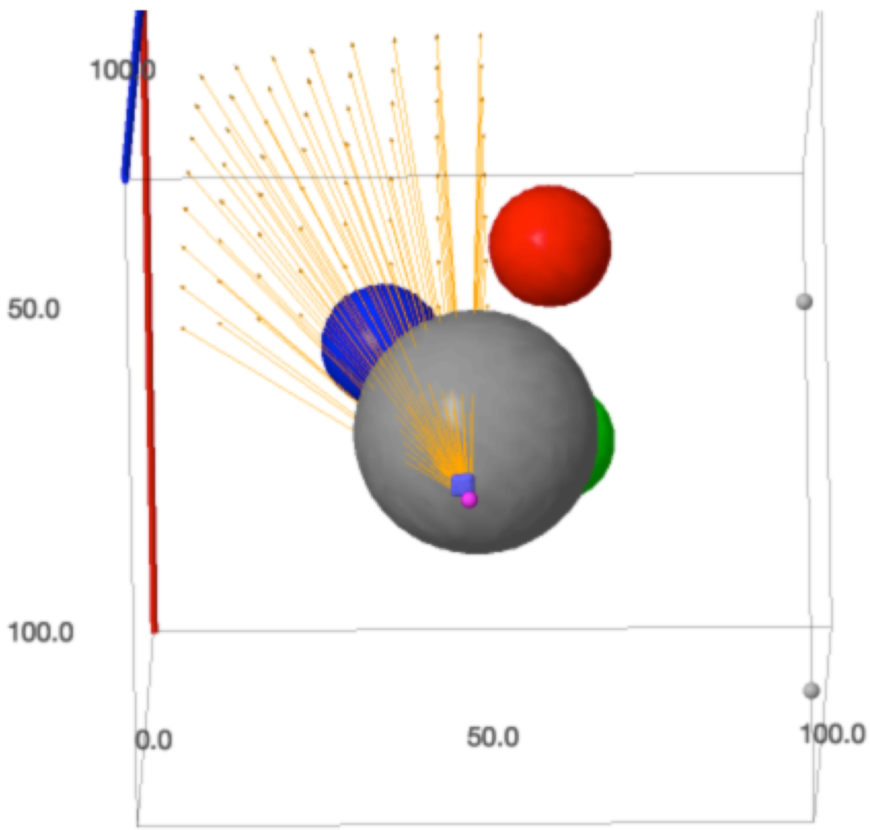
This image is created using the same configuration (Diamond) as the previous.

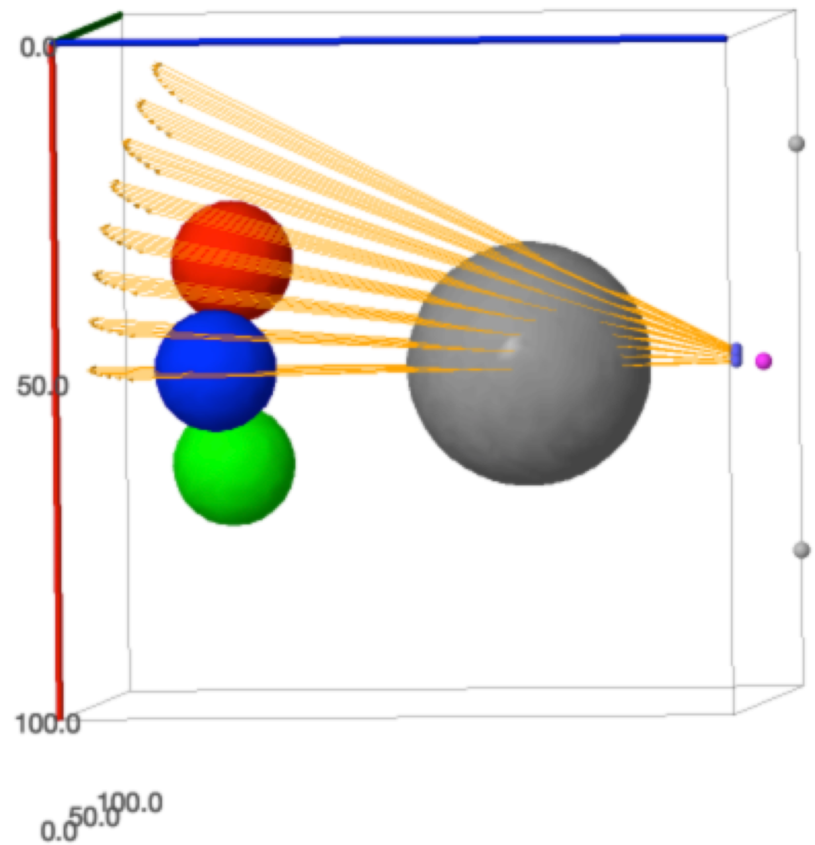
The only change is distance to the near clipping plane is 4 instead of 5



To Show a Quarter of the Image

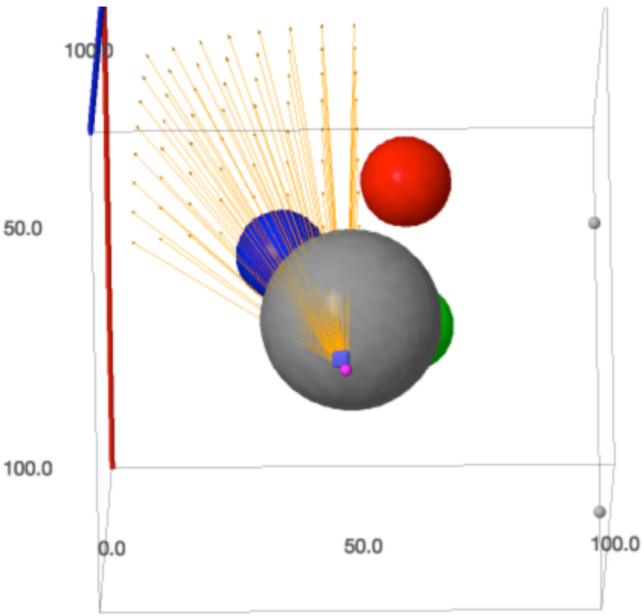
For this example the bounds run -2 to 0 on both horizontal and vertical.

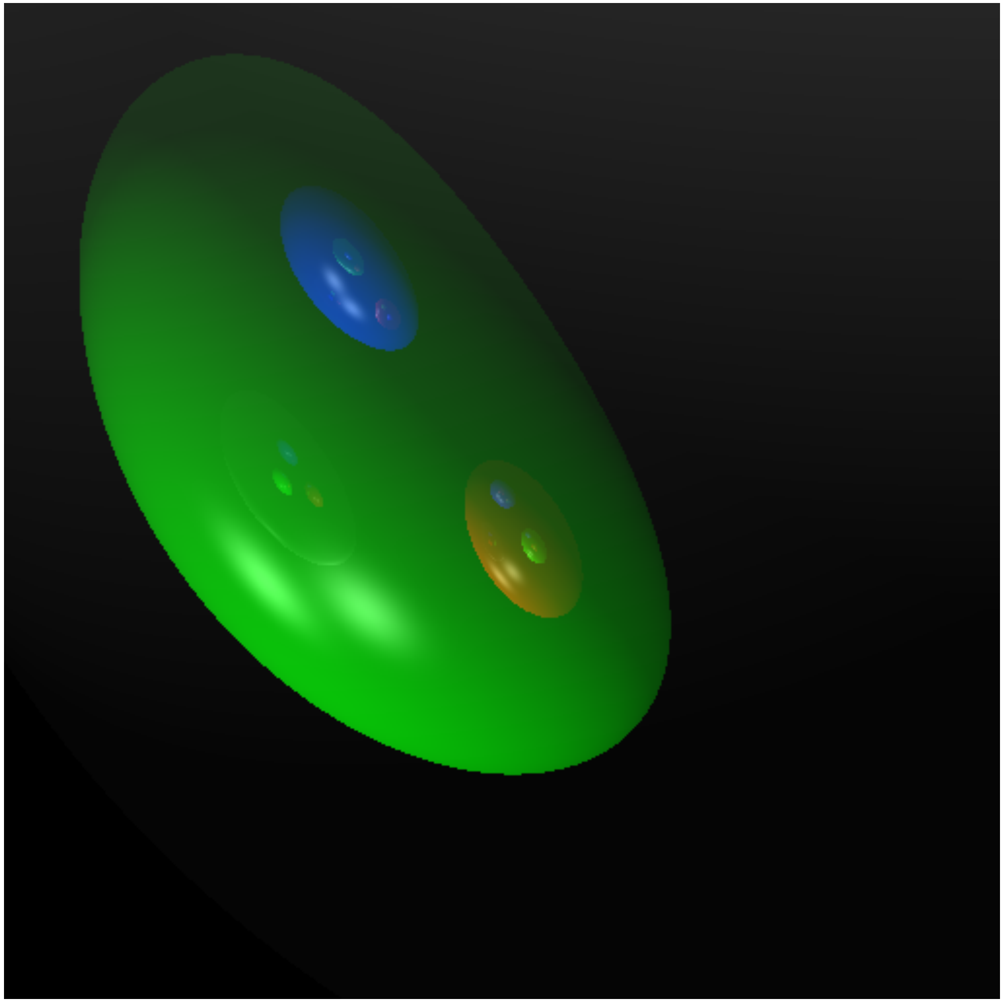




To Show a Quarter of the Image

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

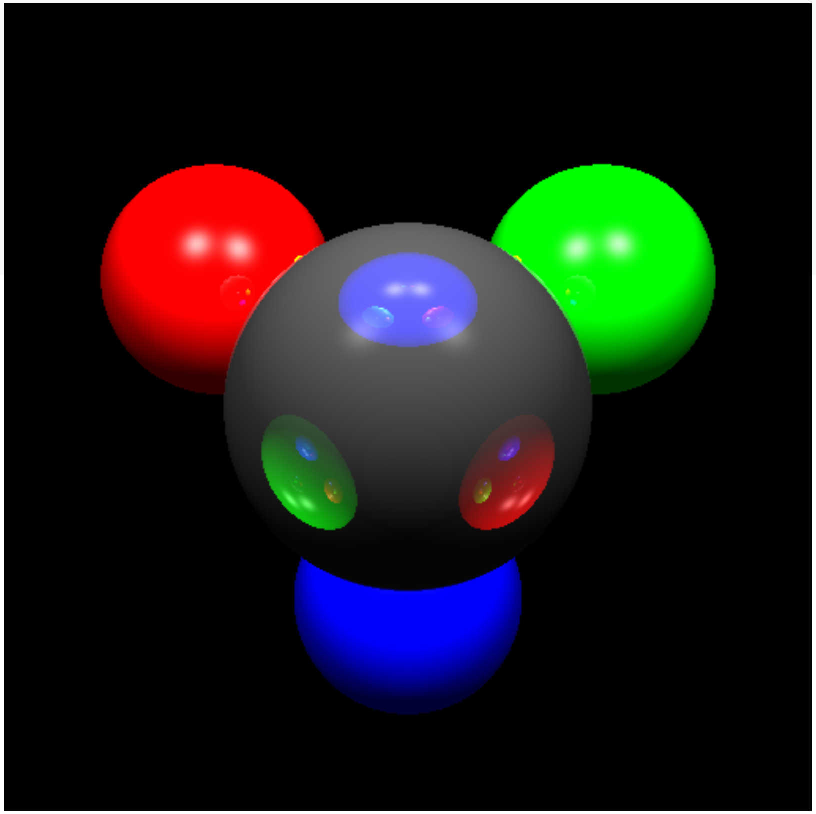




If you understand why the green sphere is being rendered in this view then you are a long way towards understanding refraction.

Now to the “default” scene

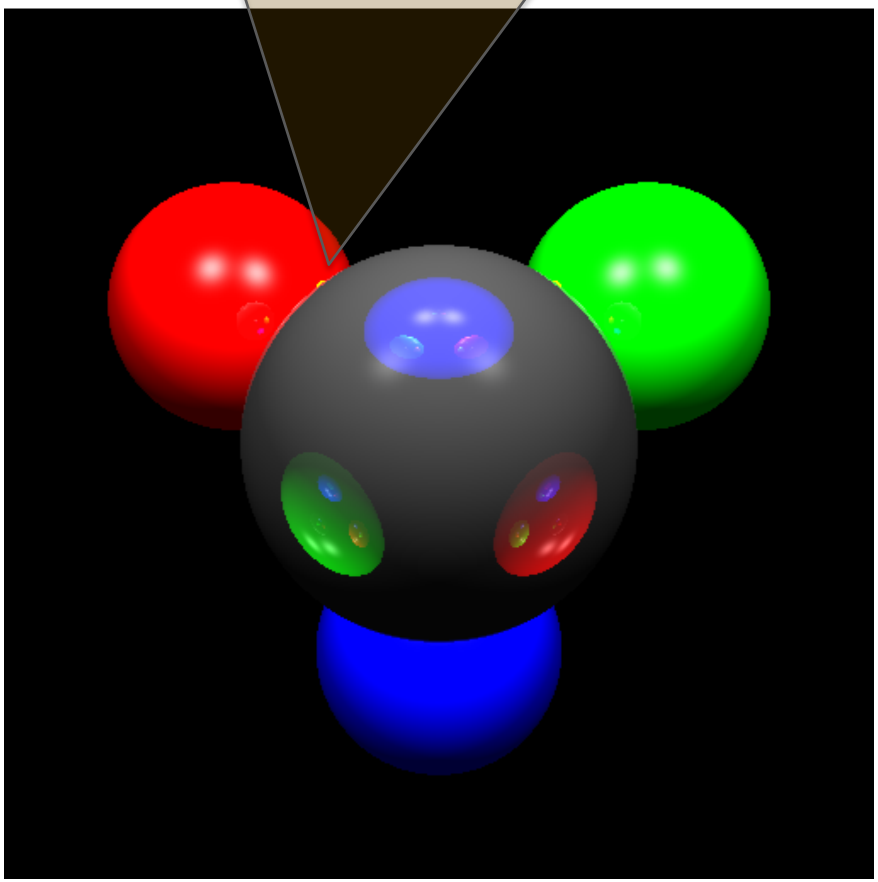
```
cam1 = Camera((50,50,100),(50,50,10),(0,1,0),(-2.0,2.0,-2.0,2.0),-5,-100,8,8)
cam2 = copy(cam1);
cam2.width  = 512
cam2.height = 512

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
```

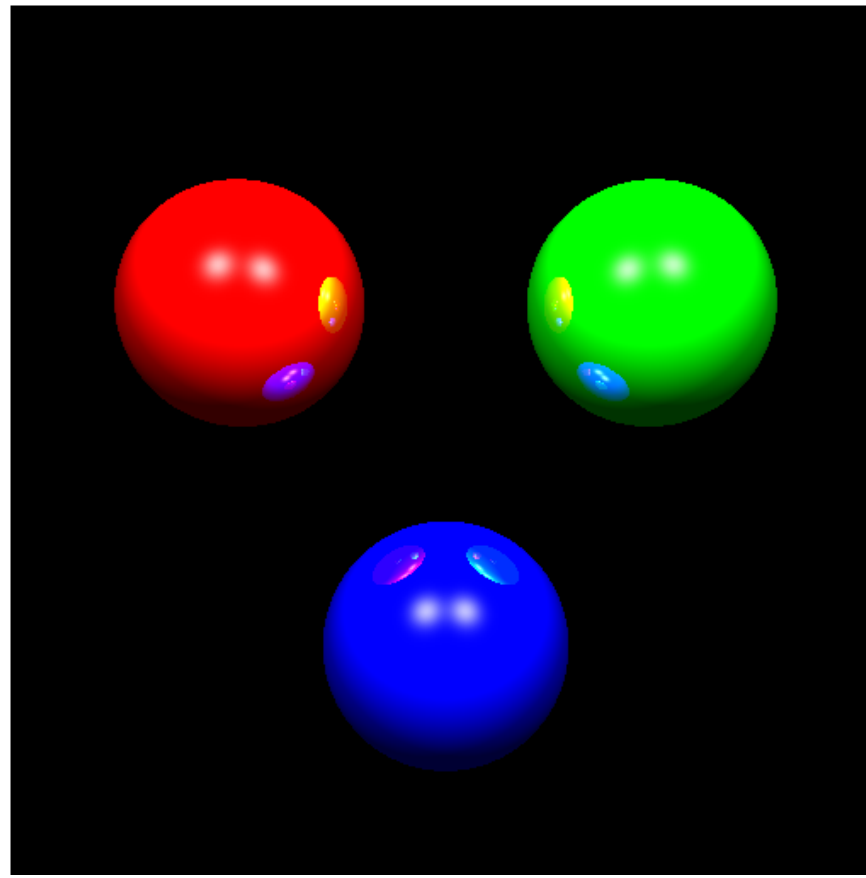


Detail: About The Yellow Pixel

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
```

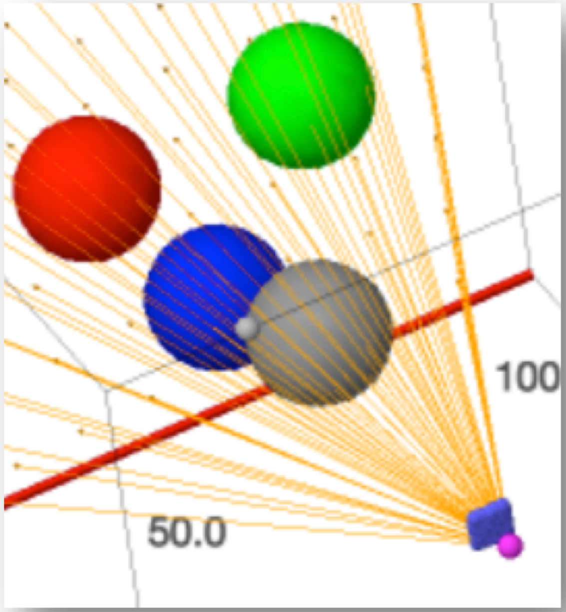
- There are two calls to ray trace
- There are two intermediate accumulation vectors for colors
- The sphere object finds the exit refraction ray
- Transparency is modulated by the `mat.ko` property.

What About Shadows

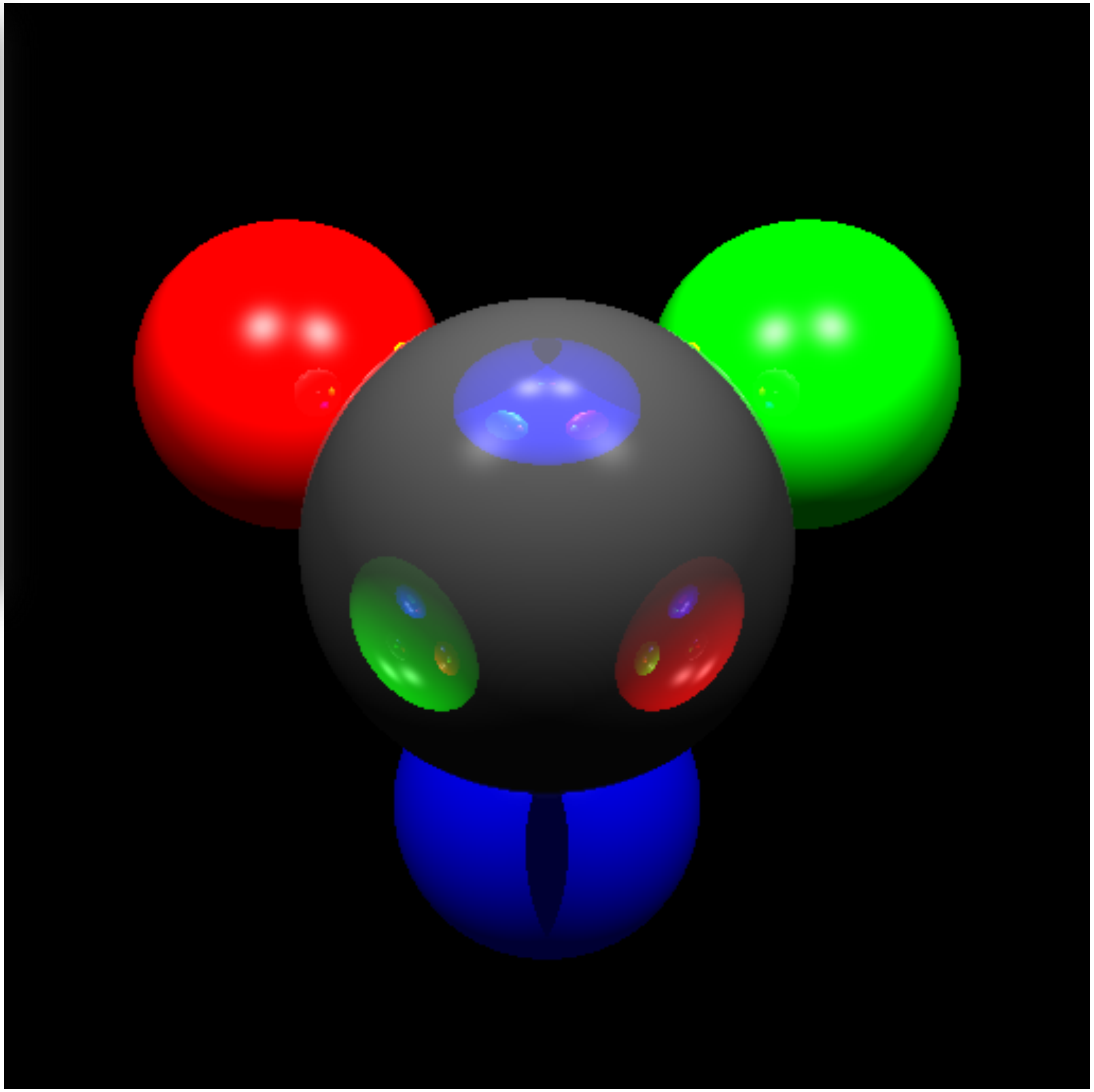
- It is easy to test whether and object is between the point of interest and light.
- It is harder to 'dim' a light – not done here.

```
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.



Second SageMath Notebook

Pay particular attention to the ring of planets and their order of appearance as reflected on the surface versus as they appear refracted through the sphere.