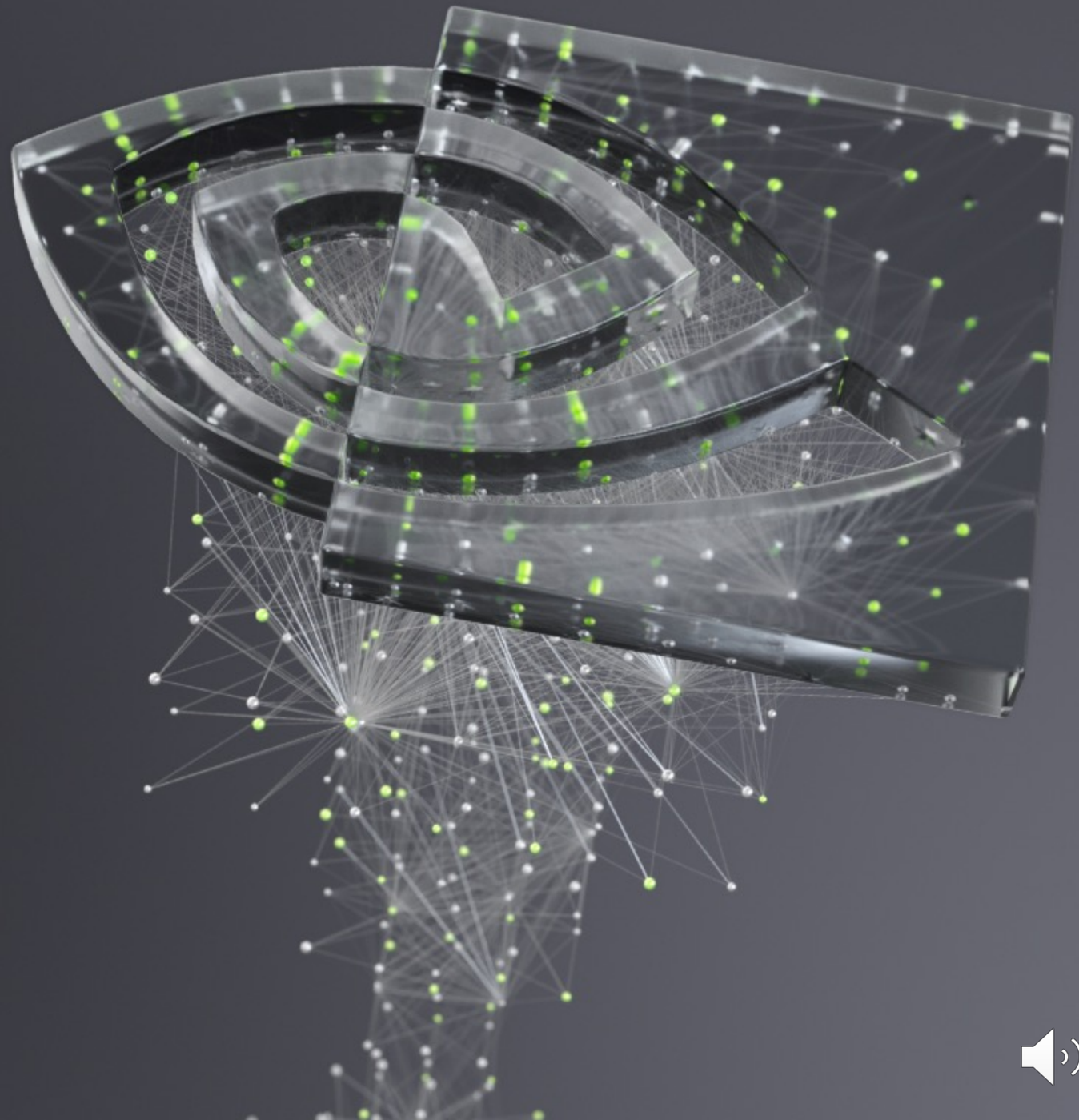


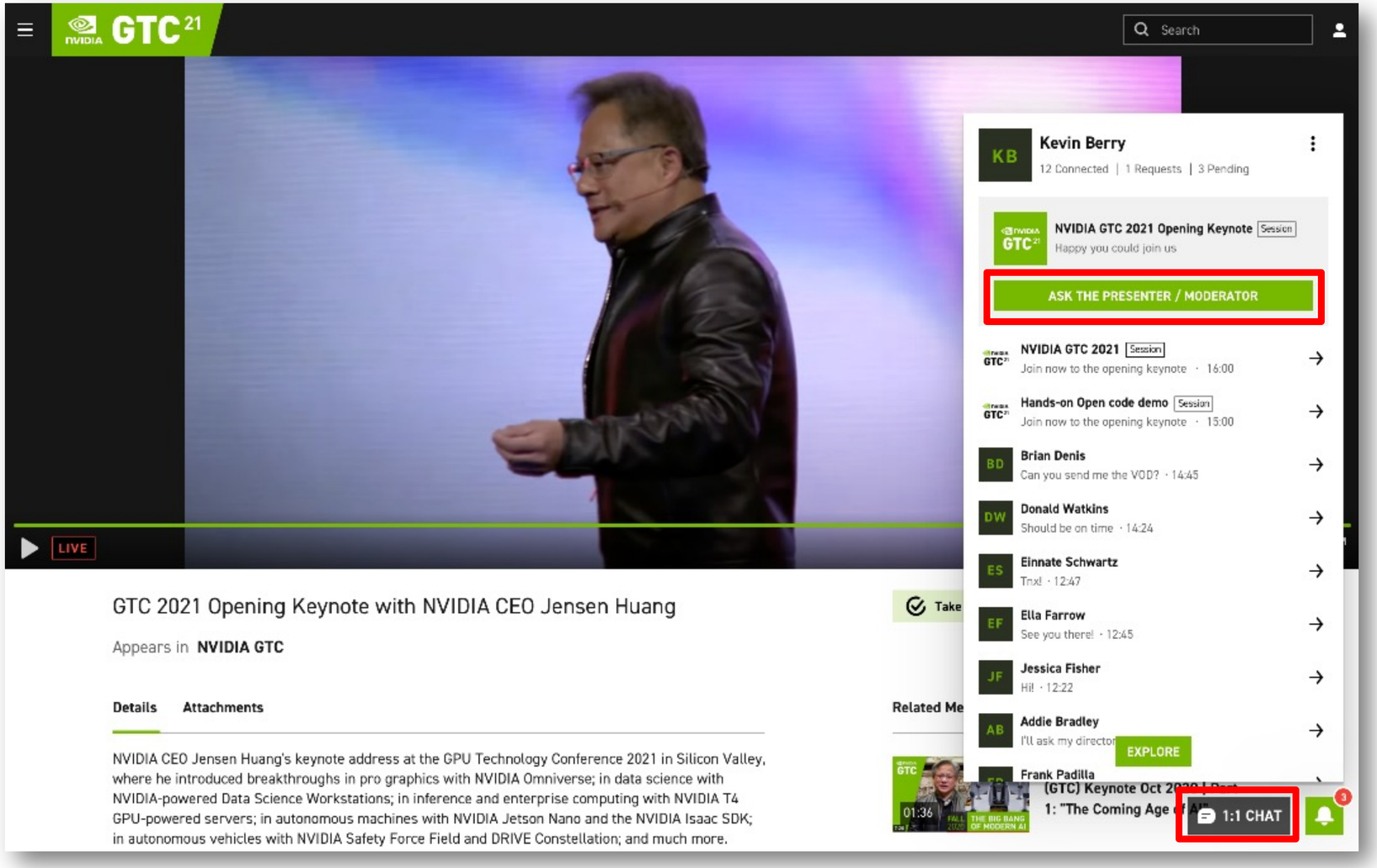
NVIDIA

RAY TRACING IN ONE WEEKEND

Peter Shirley, April 15, 2021

NVIDIA GTC 21

I'M AVAILABLE TO CHAT DURING THIS SESSION



Click on "1:1 Chat," then "Ask the Presenter/Moderator" button to submit your question.
After the session is over, connect with me via attendee chat by searching for my name.

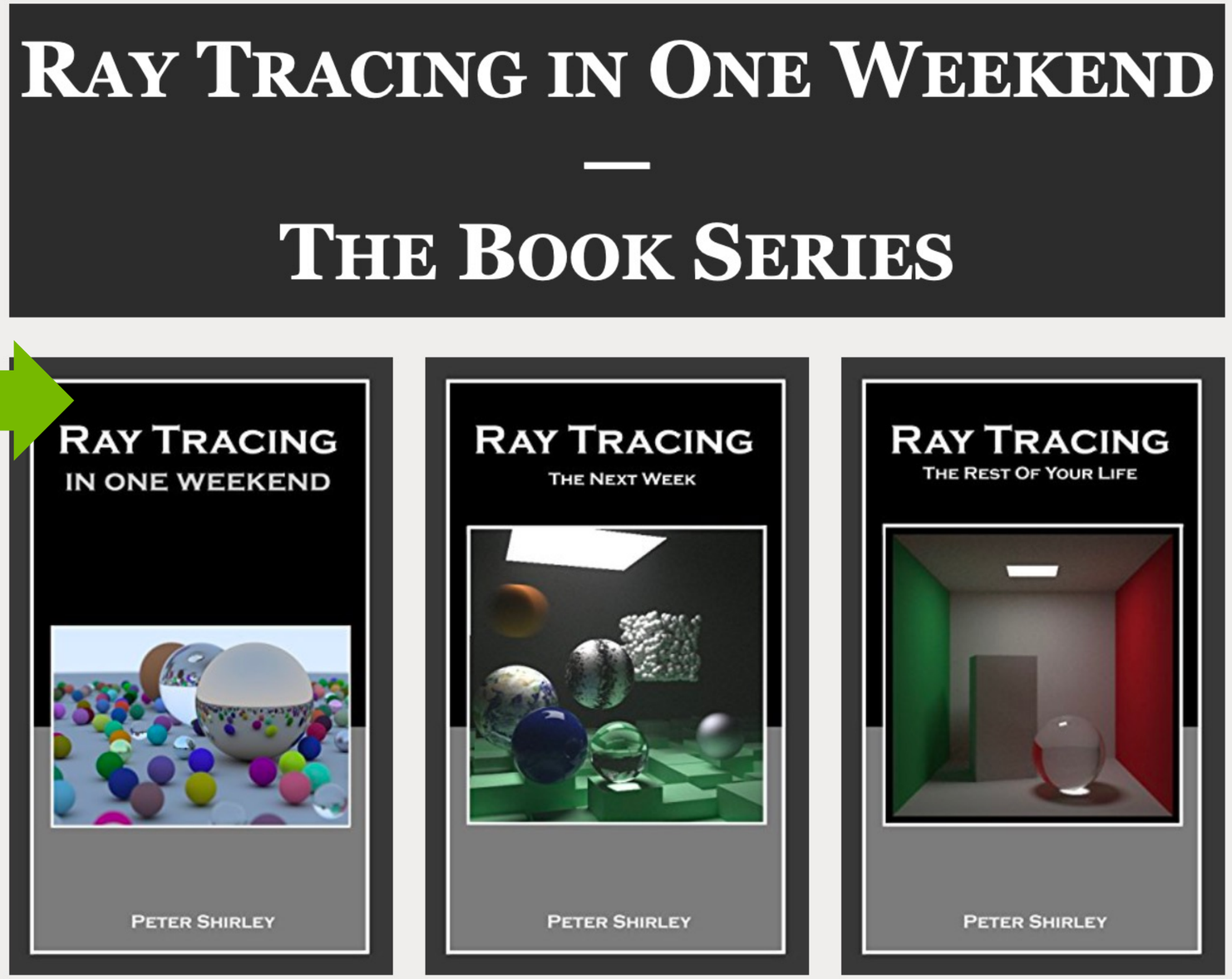
THIS TALK BASED ON

Edited by Steve Hollasch and Trevor Black

3 book C++ series at
https://raytracing.github.io/

This talk we do parts of Part 1, in Python

All the code is in a google colab : see Peter Shirley's **pinned tweet** for the location



SOME CONTEXT

Then we will get to coding!

Ray Tracing is a family of algorithms that mostly make pictures and use “rays”. They have been popular for 40 years

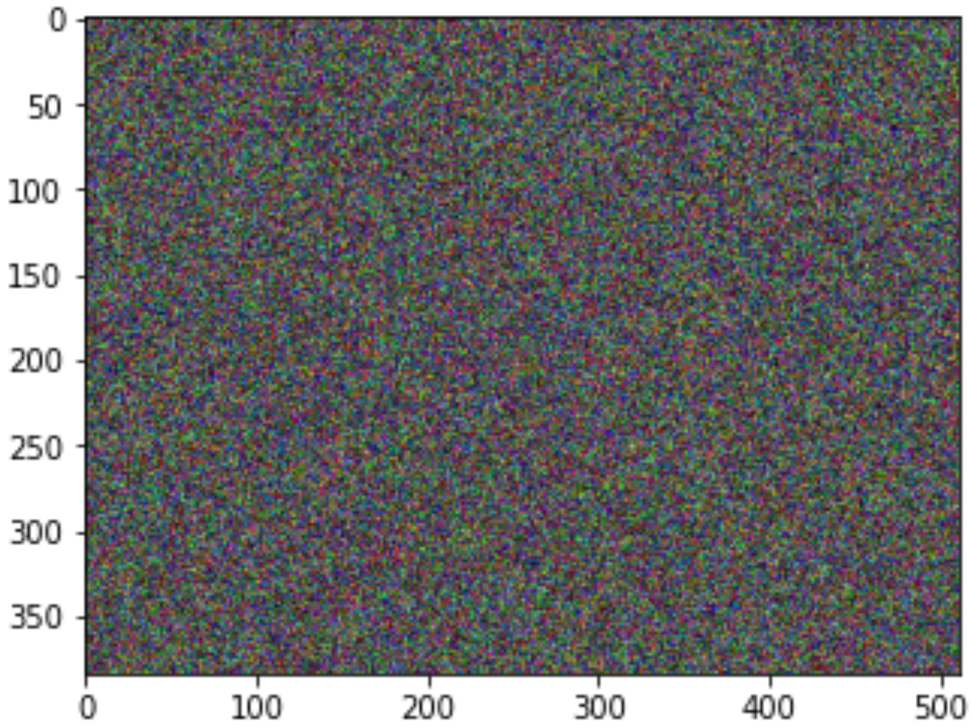
Ray tracing is possible to do interactively

Google colab is a free notebook Python environment. No installation needed: the code runs at google via a browser!

Questions on the code are more than welcome @me on Twitter (and elsewhere)

Program 0

Write to an output an image

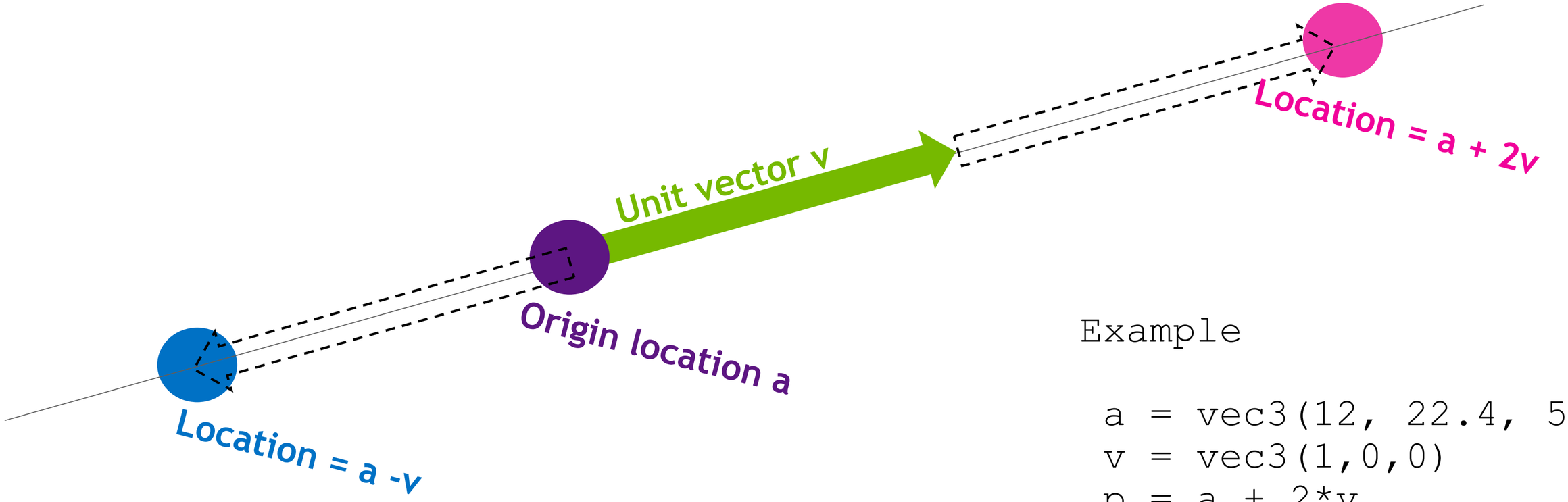



```
import numpy as np
import matplotlib.pyplot as plt
import math
import random

width = 512
height = 384
im = np.random.random(size=(height,width,3))
im = im*im # this is an approximate gamma
correction because imshow expects non-linear
intensities
plt.imshow(im)
plt.show()
```

WHAT IS A “RAY”

A directed half-line in 3D

As a computer science black box: give me a 1D coordinate and I return a 3D location: **`vec3 point_at(double t)`**

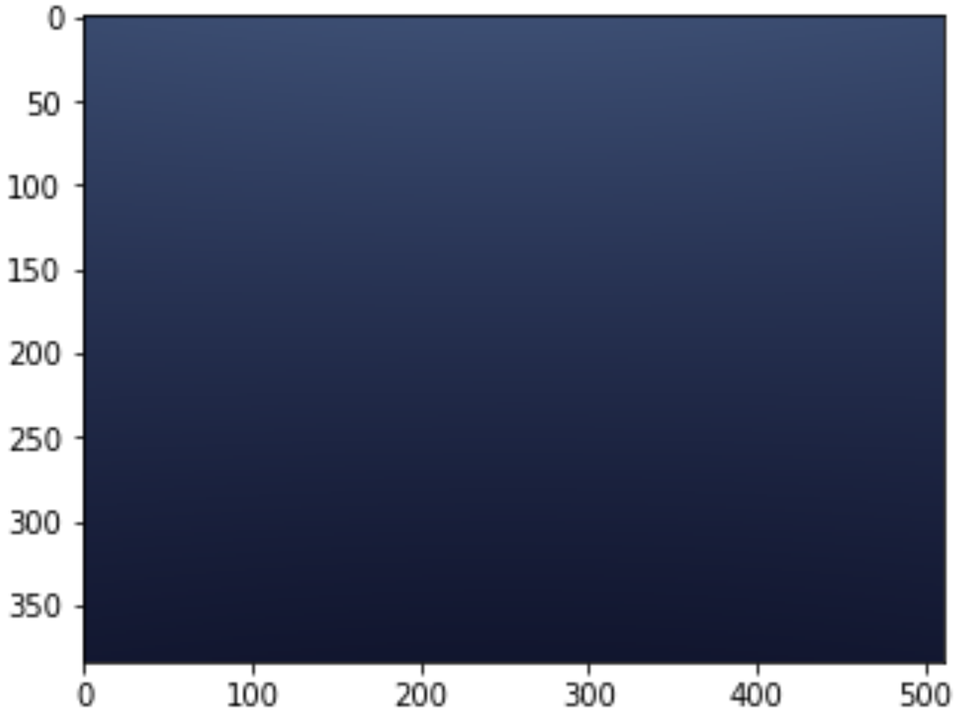


Example

```
a = vec3(12, 22.4, 5)
v = vec3(1,0,0)
p = a + 2*v
```

Program 1

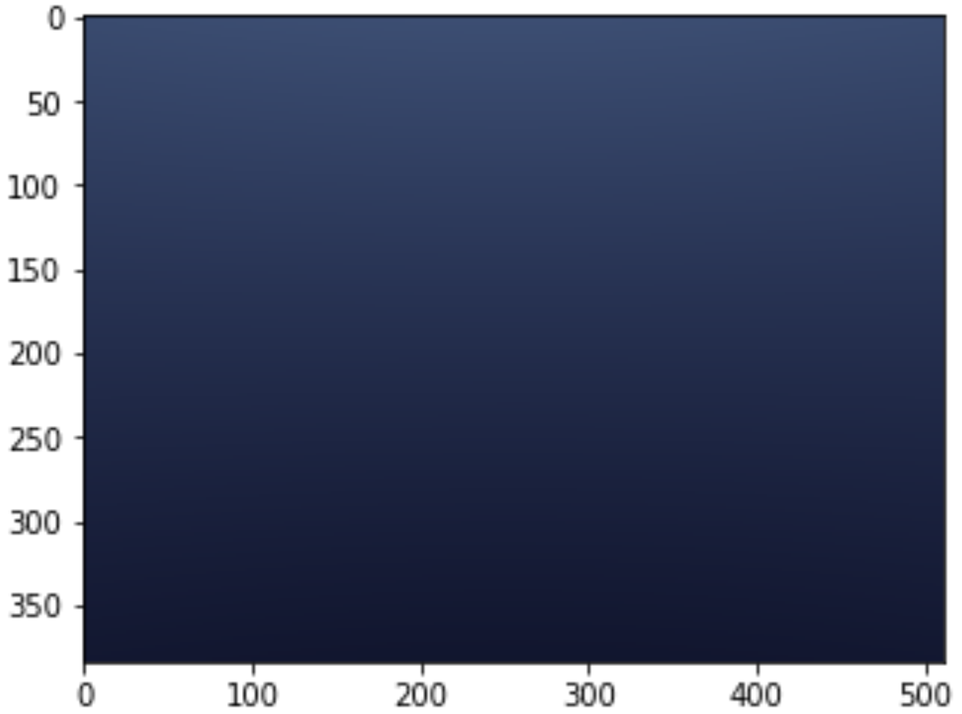
A ray trace of empty space



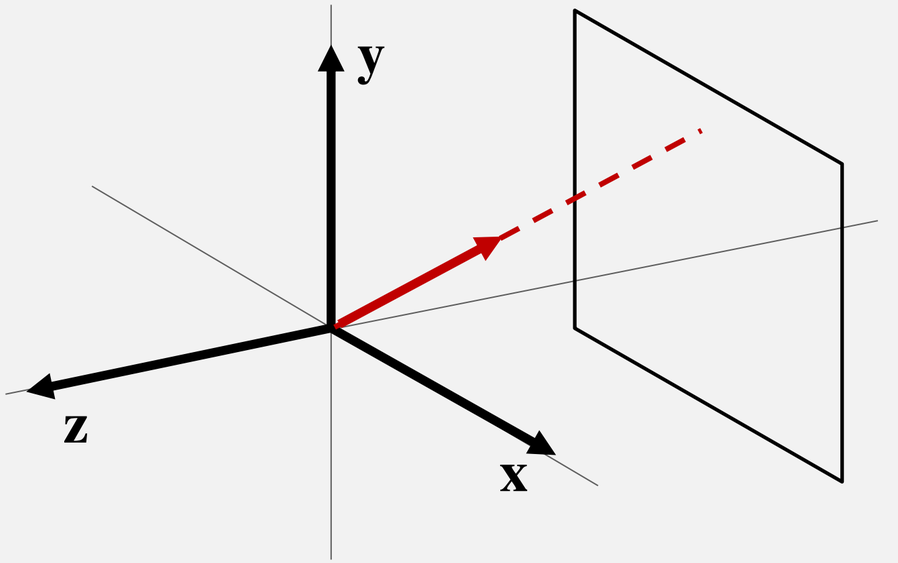

```
def vec3(x,y,z):
    return np.array((x,y,z))

def unit_vector(v):
    len = math.sqrt(np.dot(v,v))
    return v / len

def background_color(v):
    u = 0.5*(1.0 + v[1])
    return      u*vec3(0.7, 0.8, 0.9) +
           (1.0-u)*vec3(0.05, 0.05, 0.2)
```

Program 1 (cont'd)

A ray trace of empty space

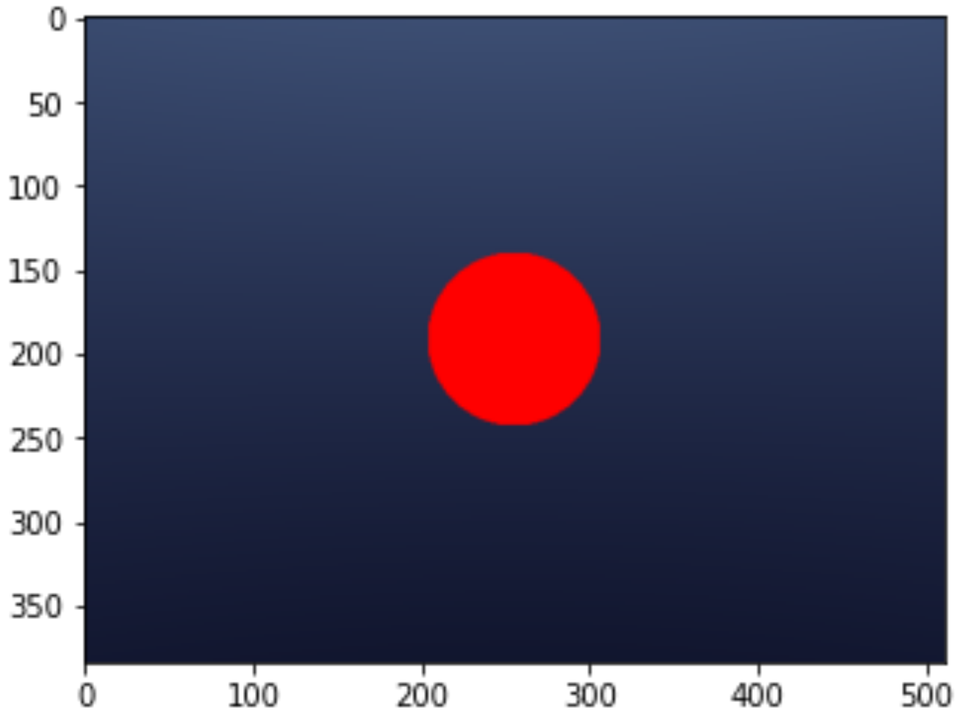



```
ray_origin = vec3(0,0,0)
aspect = height/width
window_depth = 2.0
for row in range(height):
    for column in range(width):
        u = (column + 0.5) / width
        v = (row + 0.5) / height
        ray_direction = unit_vector(vec3(2.0*u-1.0,
        aspect*(2.0*v-1.0), -window_depth))
        im[height-row-1,column,:] =
            background_color( ray_direction )
```



Program 2

A ray trace of a red sphere

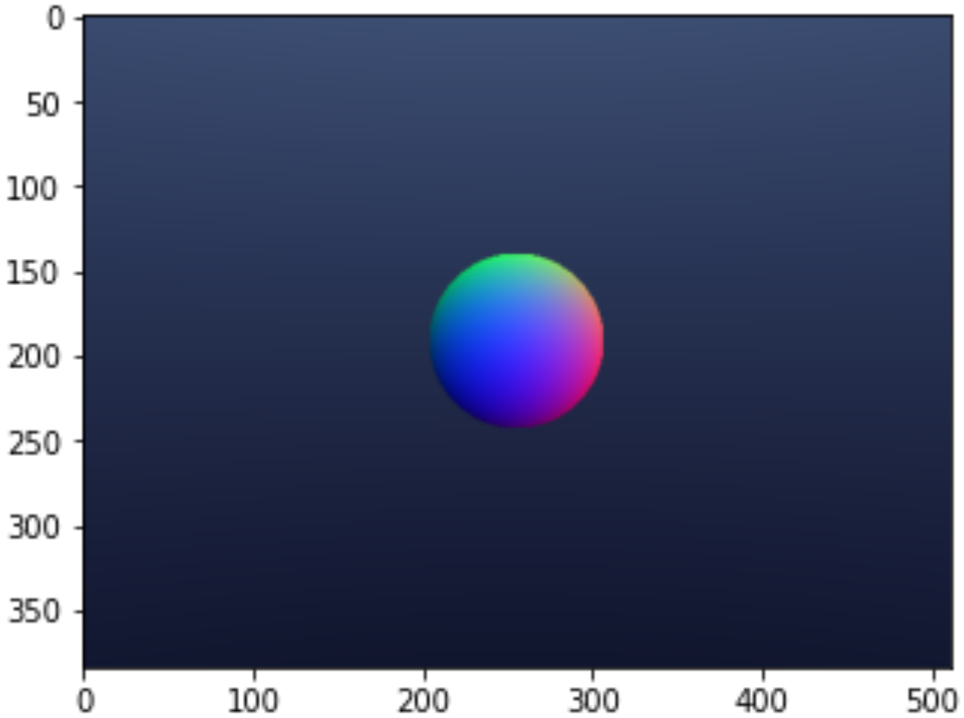



```
def hit_sphere(ray_origin, ray_direction,
                center, radius):
    oc = ray_origin - center
    qb = np.dot(oc,ray_direction)
    qc = np.dot(oc,oc)-radius*radius
    discriminant = qb*qb-qc
    if (discriminant > 0):
        t = (-qb - math.sqrt(discriminant))
    if t > 0.00001:
        return t
    t = (-qb + math.sqrt(discriminant))
    if t > 0.0001:
        return t
    return 1.0e9
```

```
t = hit_sphere(ray_origin, ray_direction,
                center, radius)
if (t < 1.0e8):
    im[height-row-1,column,:] = vec3(1,0,0)
else:
    im[height-row-1,column,:] =
            background_color( ray_direction )
```

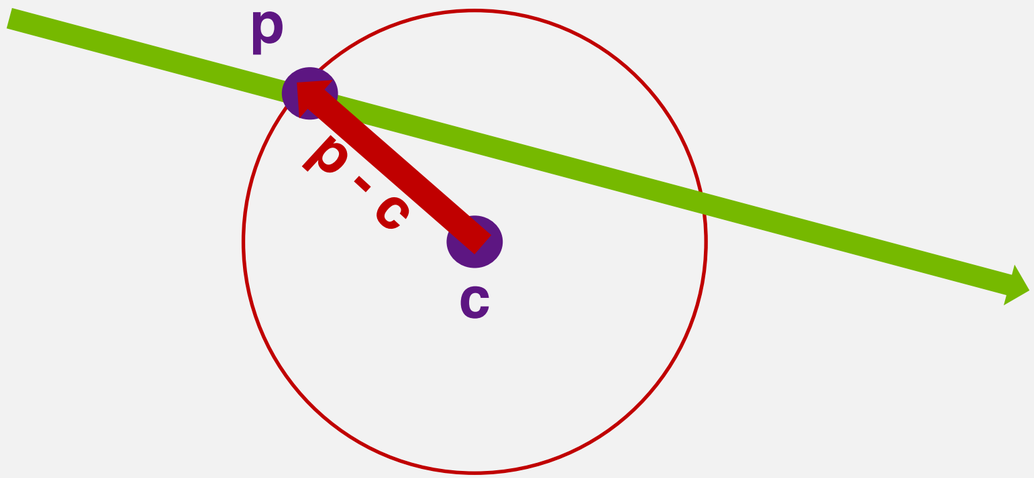
Program 3

Visualize the normal vectors



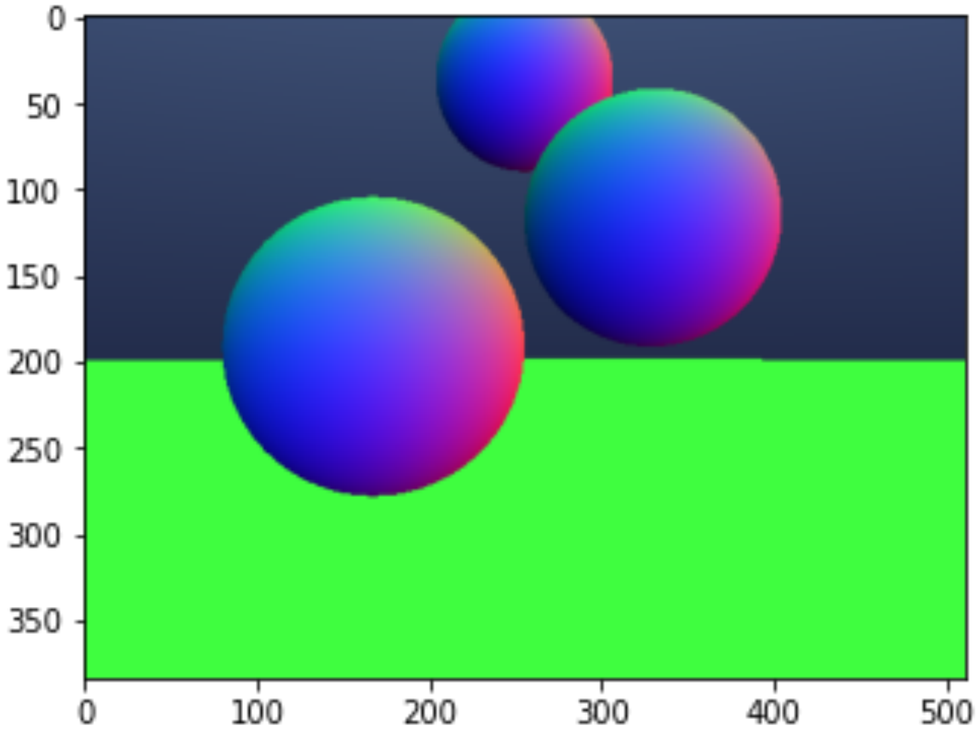

```
def pseudocolor(v):
    return 0.5*(vec3(1,1,1) + v)
```

```
hit_point = ray_origin + t*ray_direction
surface_normal = (1/radius)*(hit_point - center)
im[height-row-1,column,:] =
                    pseudocolor(surface_normal)
```



Program 4

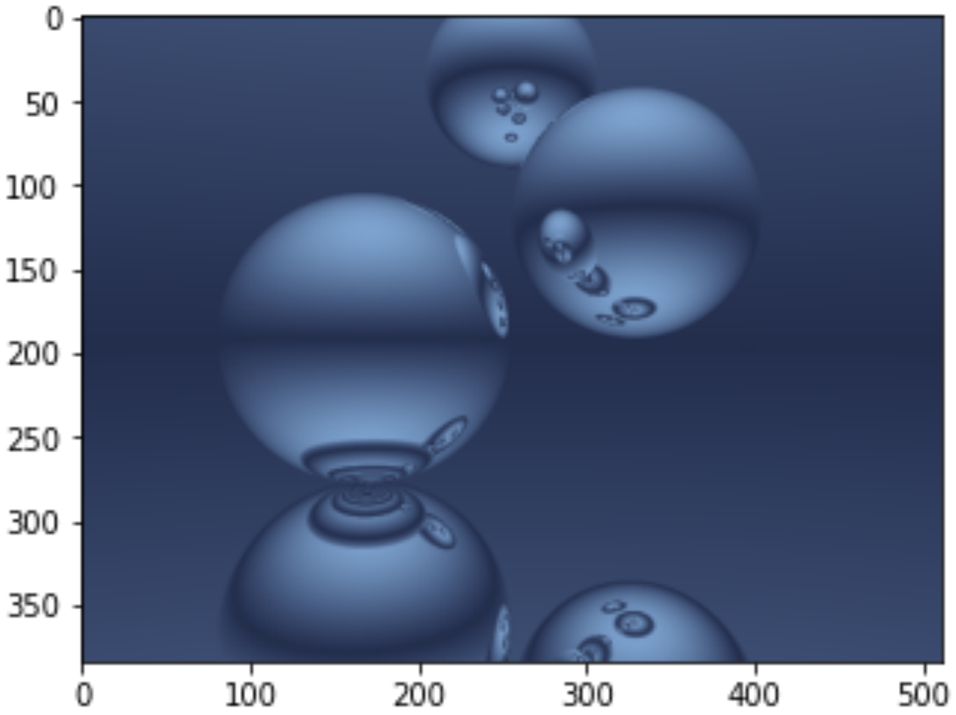
Multiple spheres



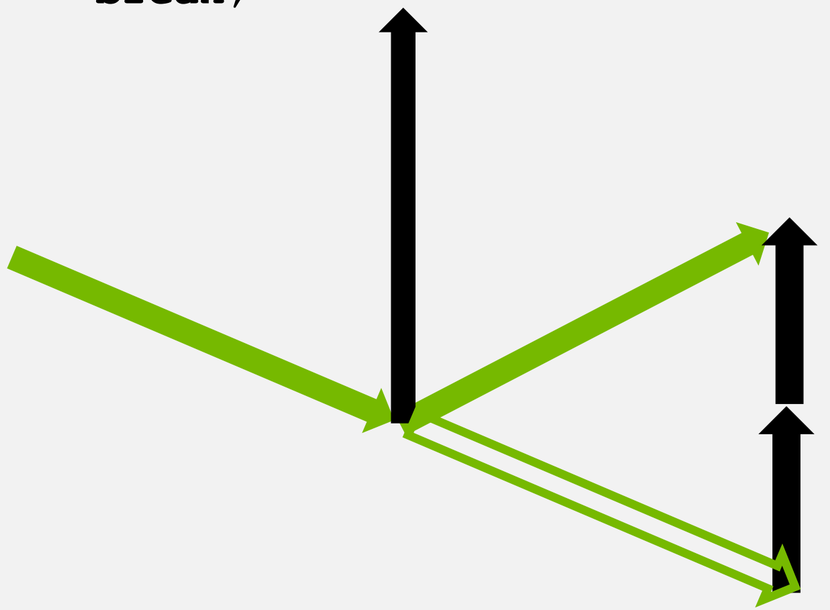

```
def hit_array(ray_origin, ray_direction, centers,
              radii):
    t_min = 9.0e8
    t_min = -1
    hit_something = False
    for i in range(len(centers)):
        t = hit_sphere(ray_origin, ray_direction,
                       centers[i], radii[i])
        if (t >= 0.0001 and t < t_min):
            t_min = t
            i_min = i
             hit_something = True
    return (hit_something, t_min, i_min)
```

Program 5

Specular (mirror) reflections

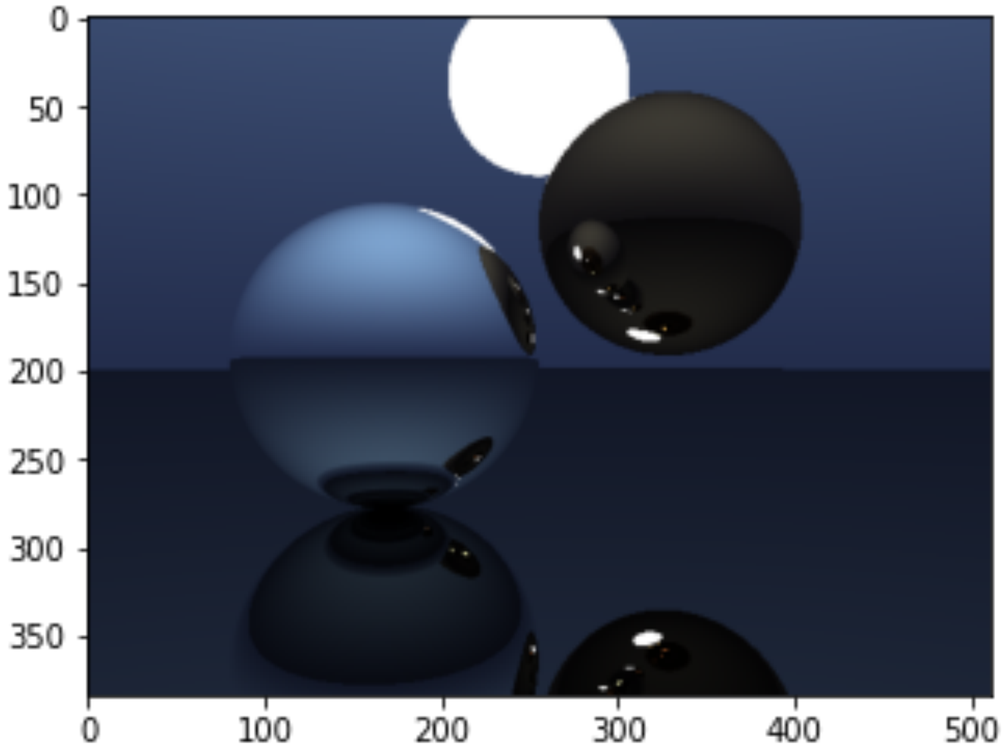



```
def reflect(v, n):
    return v - 2.0*np.dot(v,n)*n

while True:
   if (hit_something):
      ray_origin = ray_origin + t*ray_direction
      surface_normal = (1.0/radii[i])*(ray_origin -
                                centers[i])
      ray_direction = reflect(ray_direction,
                                    surface_normal)
    else:
      im[height-row-1,column,:] =
          background_color( ray_direction )
      break;
```



Program 6

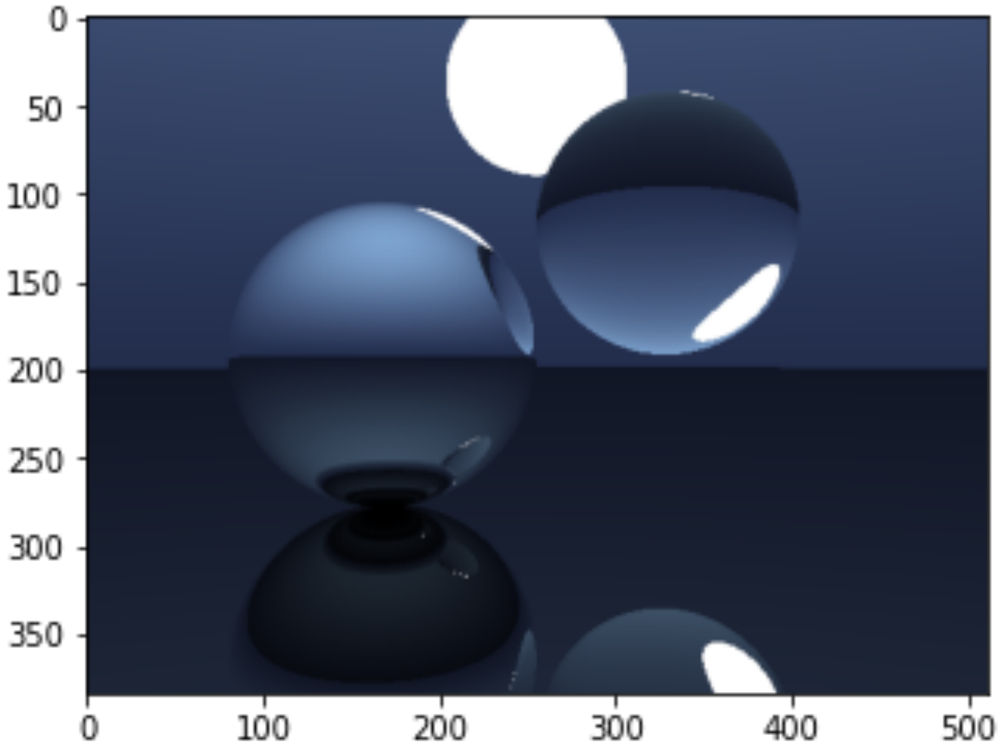
Add a light and attenuation




```
while True:
    (hit_something, t, i) =
        hit_array(ray_origin, ray_direction,
                  centers, radii)
     if (hit_something):
        if (not_zero(emitted_colors[i])):
           im[height-row-1,column,:] =
                  multiplier*emitted_colors[i]
           break
        ray_origin = ray_origin + t*ray_direction
        surface_normal = (1.0/radii[i])*
            (ray_origin - centers[i])
        ray_direction = reflect(ray_direction,
                                surface_normal)
        multiplier *= specular_reflectivity[i]
    else:
        im[height-row-1,column,:] =
               multiplier*
               background_color( ray_direction )
        break;
```

Program 7

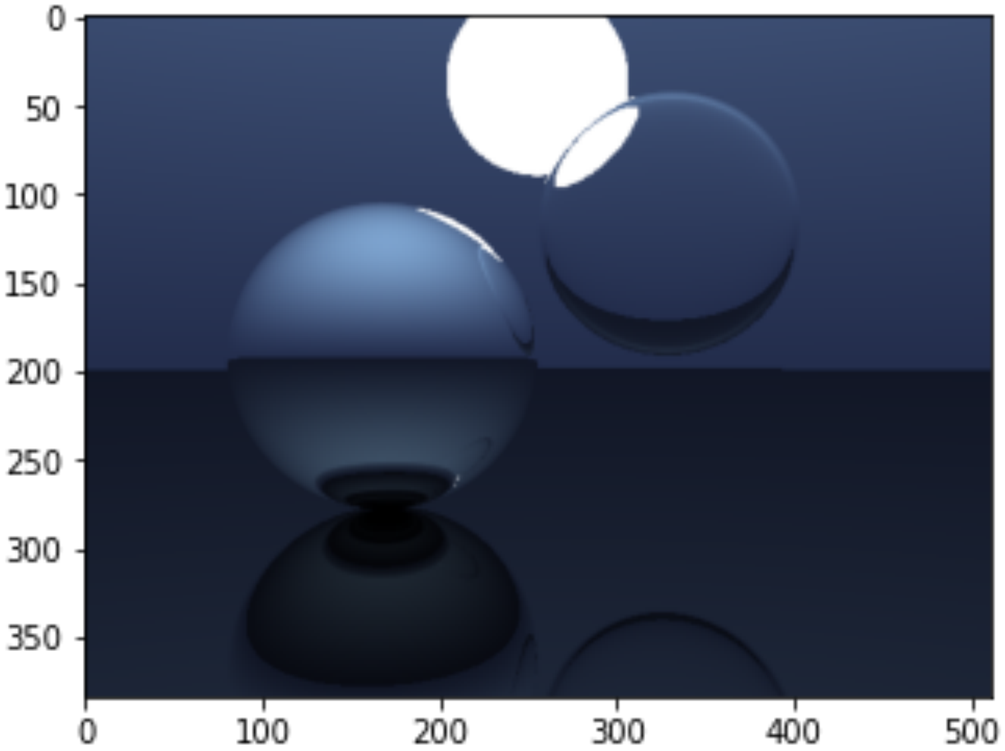
Add simple glass




```
def refract(v, normal, ni, nt):  ….



if (refractive_indices[i] > 1.0):
    cos_incident =
         np.dot(ray_direction,surface_normal)
    if (cos_incident < 0):
         ni = 1.0
         nt = refractive_indices[i]
    else:
         ni = refractive_indices[i]
         nt = 1.0
         surface_normal = -surface_normal
    (can_refract, refracted_direction) =
      refract(ray_direction, surface_normal, ni, nt)
    if (can_refract):
      ray_direction = refracted_direction
    else:
      ray_direction = reflect(ray_direction,
                              surface_normal)
```

Program 8

Hollow glass spheres

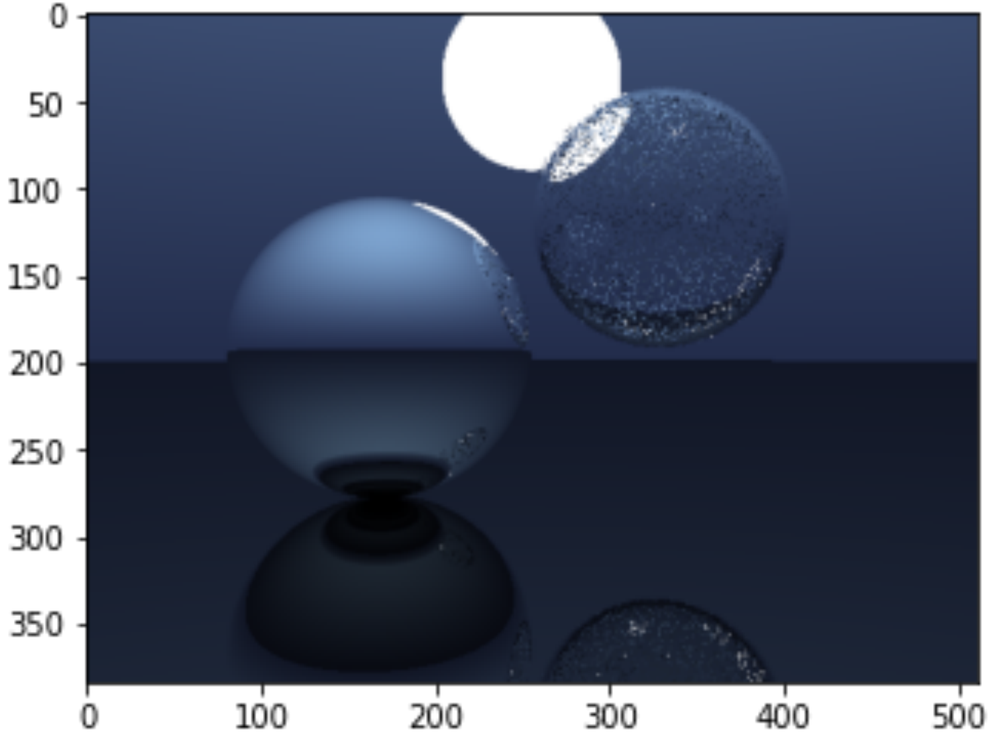


HACK!

```
refractive_indices.append(1.5)
emitted_colors.append(vec3(0.0,0.0,0.0))
centers.append(vec3(1.0, 1.0, -7.0))
radii.append(-0.95)
specular_reflectivity.append(vec3(0.0,0.0,0.0))
```

Program 9

Reflection and refraction

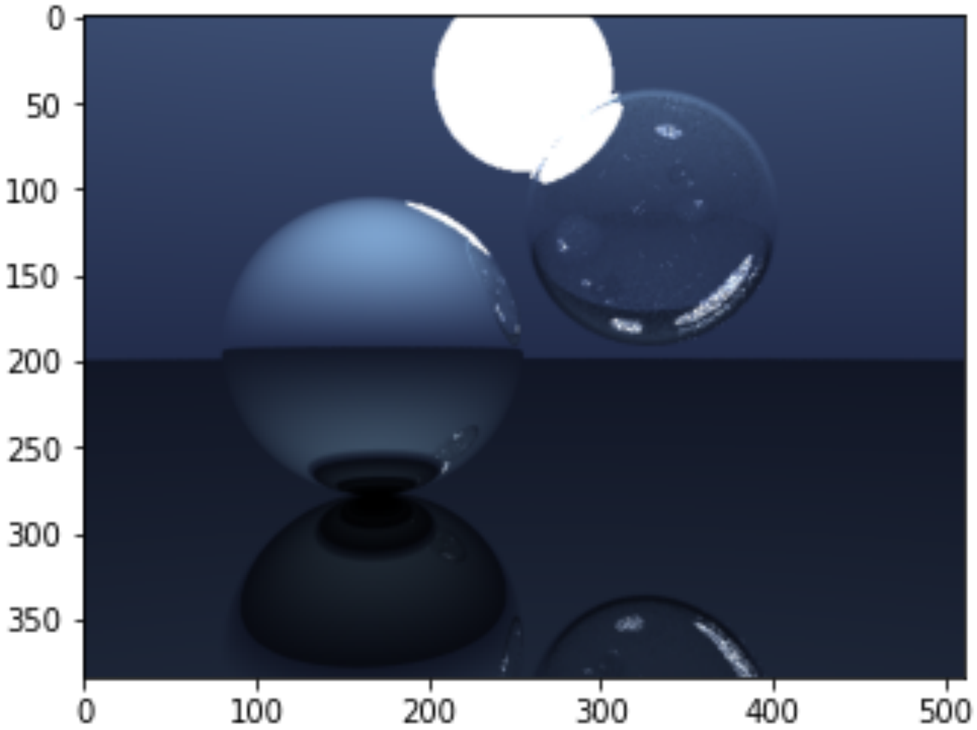



```
def glass_fresnel(c):
    c5 = math.pow(1-c, 5)
return 0.05*(1-c5) + 0.95*c5
```

```
prob_reflect = glass_fresnel(cosine)
if (random.uniform(0.0,1.0) < prob_reflect):
    ray_direction = reflect(ray_direction,
          surface_normal)
else:
    ray_direction = refracted_direction
```

Program 10

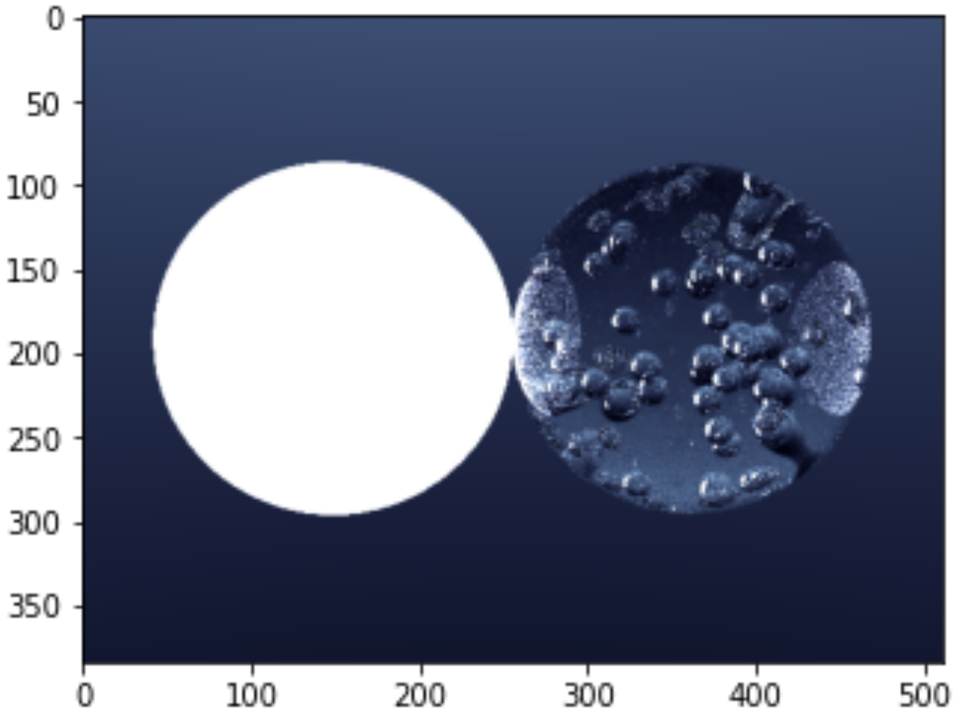
Multiple samples per pixel




```
for row in range(height):
 for column in range(width):
  accum = vec3(0.0,0.0,0.0)
  for sample in range(num_samples):
    u = (column + random.uniform(0.0,1.0)) / width
    v = (row + random.uniform(0.0,1.0)) / height


  im[height-row-1,column,:] = accum / num_samples
```

Program 11

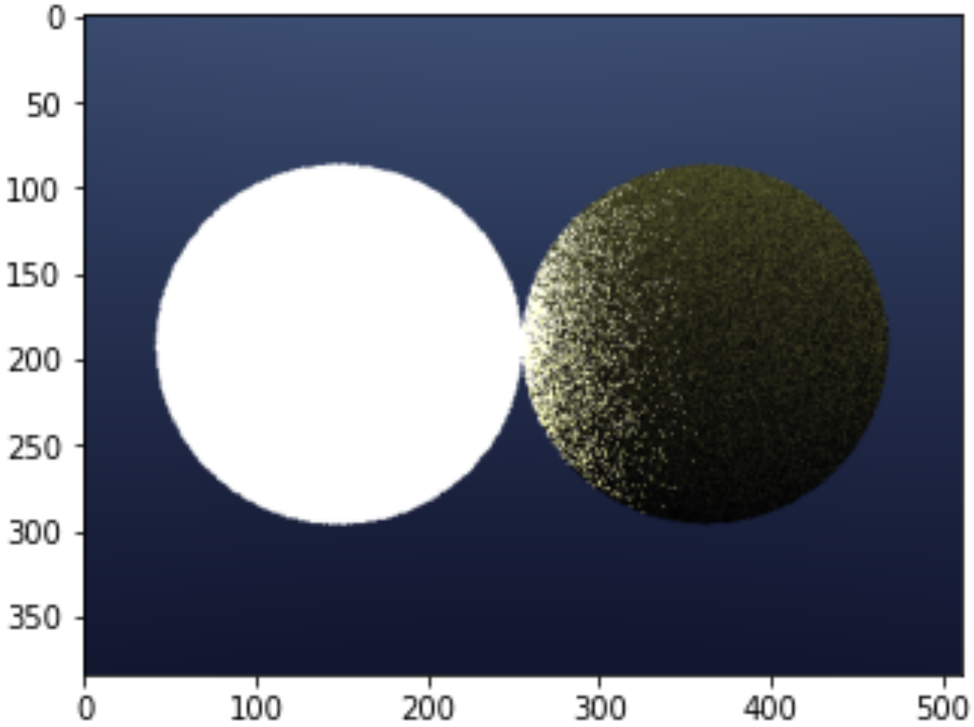
Bubbles in glass



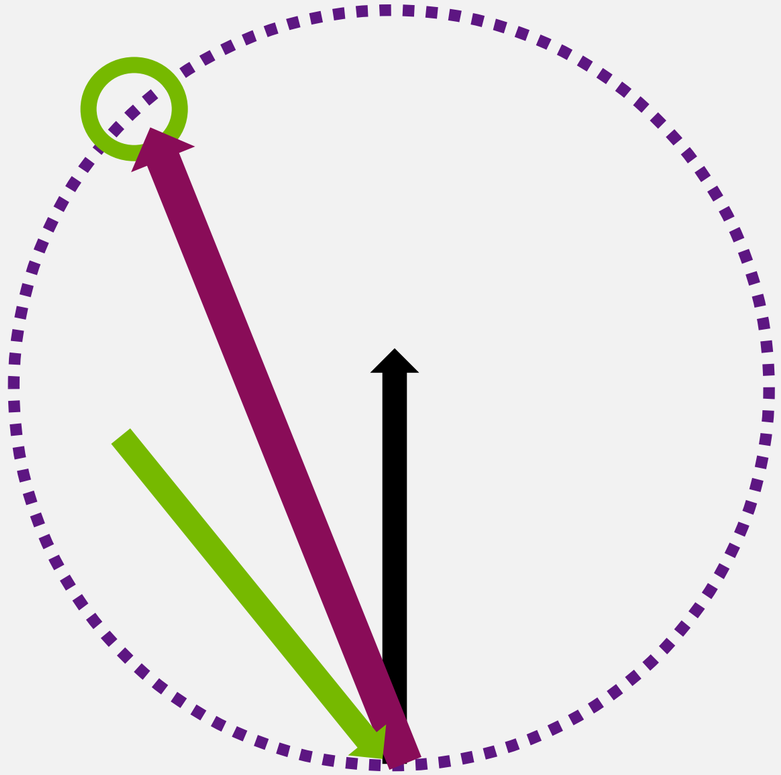
Source: Wikipedia

Program 12

Diffuse sphere

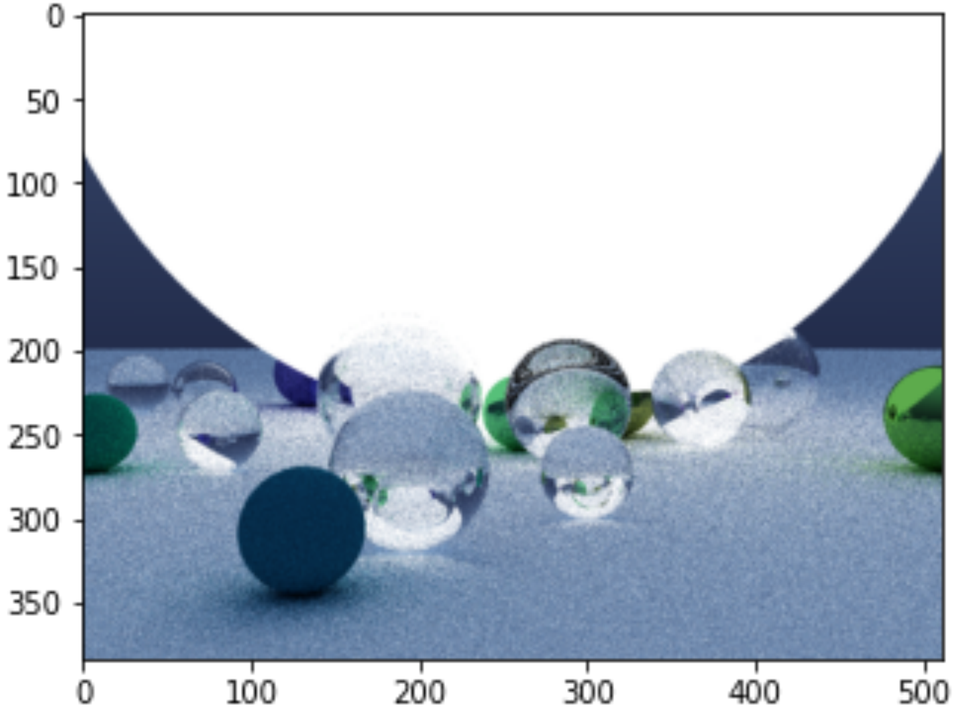



```
if (not_zero(diffuse_reflectivity[i])):
    target = random_on_sphere(ray_origin +
                              surface_normal, 0.99)
     ray_direction = unit_vector(target - ray_origin)
     multiplier *= diffuse_reflectivity[i]
```



Program 13

Do lots of random spheres



Just procedural object creation!

WHERE NOW?!

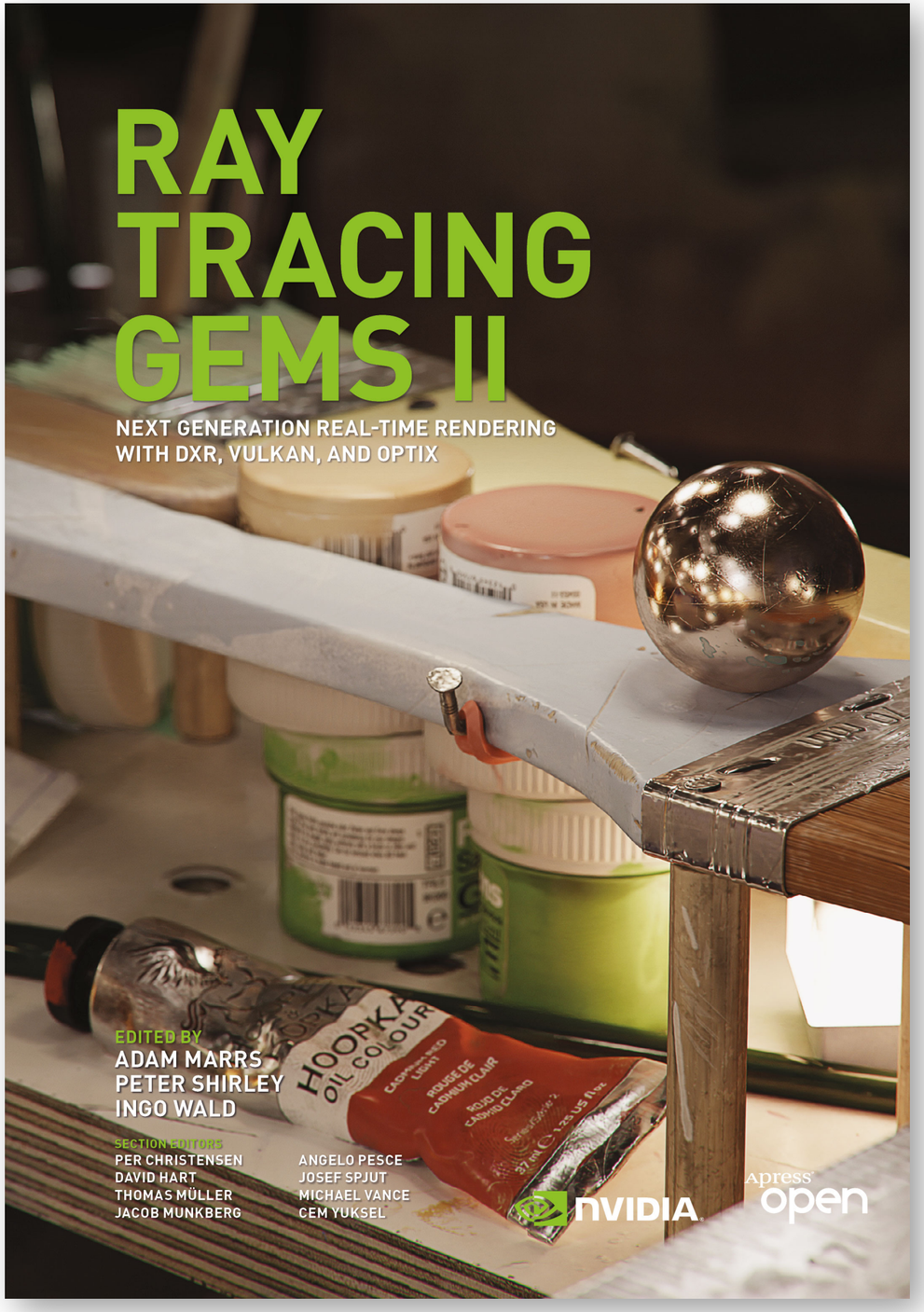
So many options!

Speed

Function

(eventually both)

Send me your progress on Twitter!

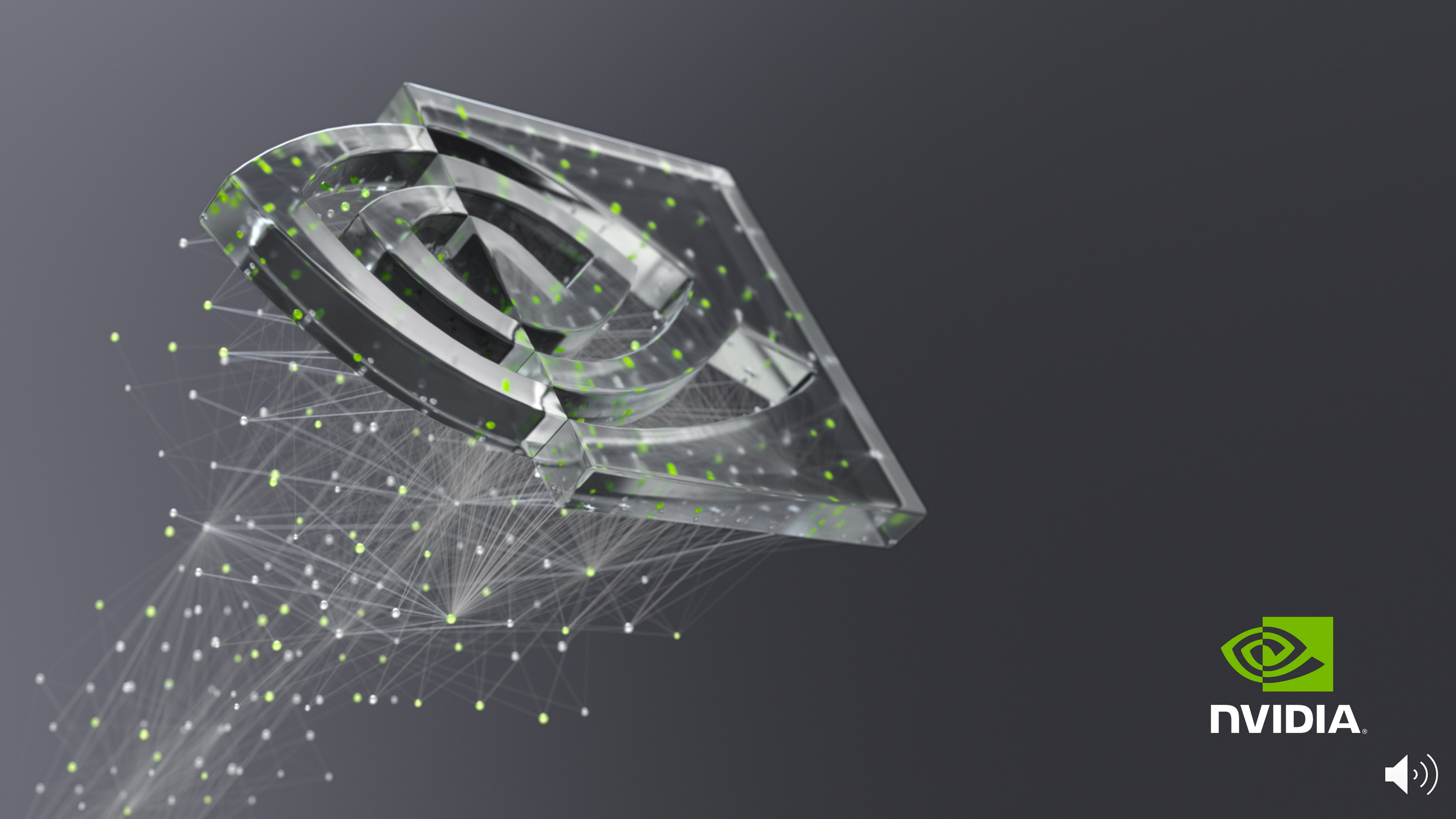


Available August 4th, 2021

Free Digital Edition

Print Edition from Apress and Amazon

Ray Tracing Gems II brings the community of rendering experts back together and unearths true "gems" for developers of games, architectural applications, visualizations, and more in this exciting new era of real-time rendering.


NVIDIA