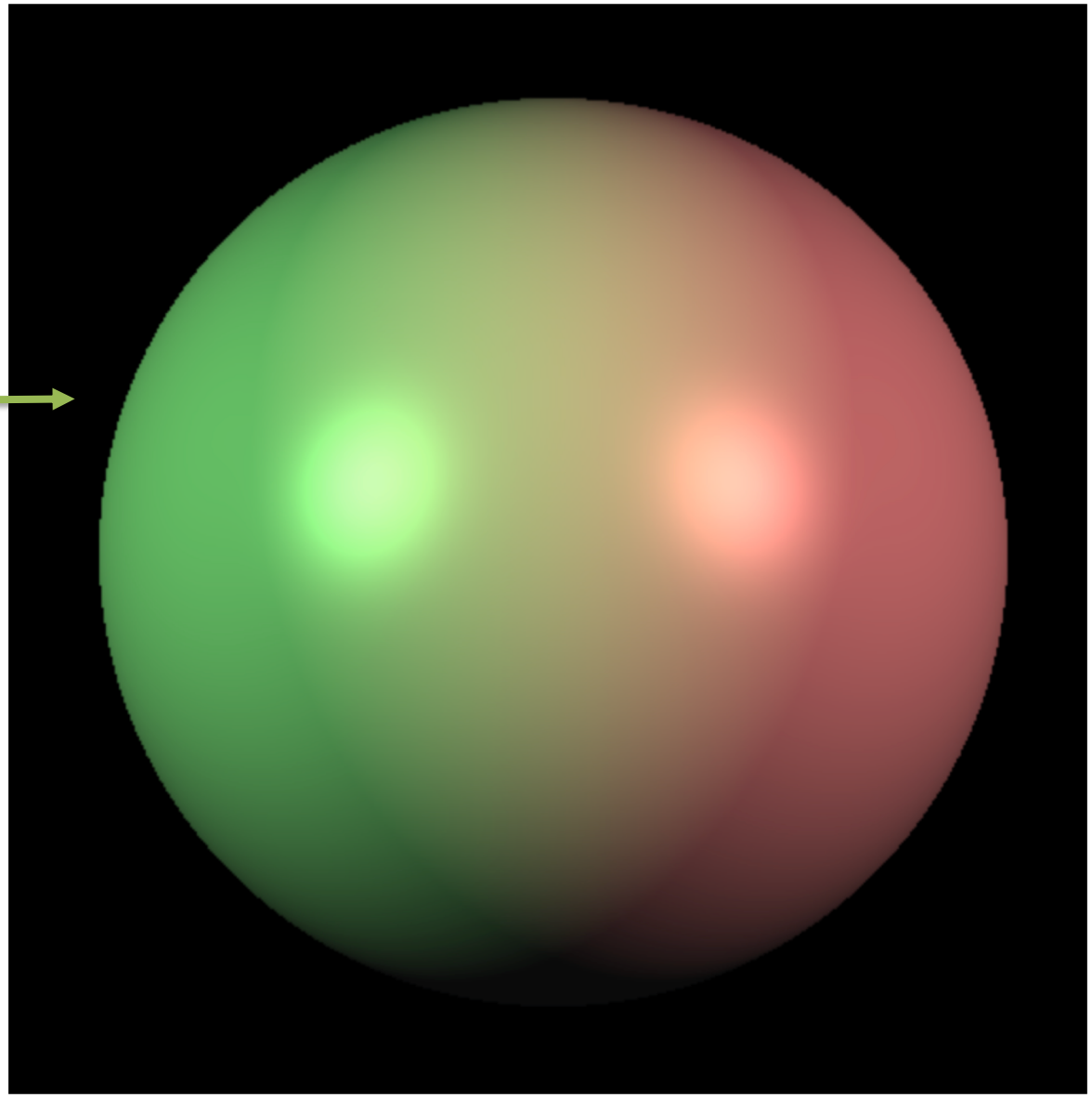
Lecture 14: Illumination SageMath Examples CS 410

October 11, 2018

Part 1: What You See

Specification

- Camera
- Object / Globe
- Material
- Ambient Light
- Point Lights

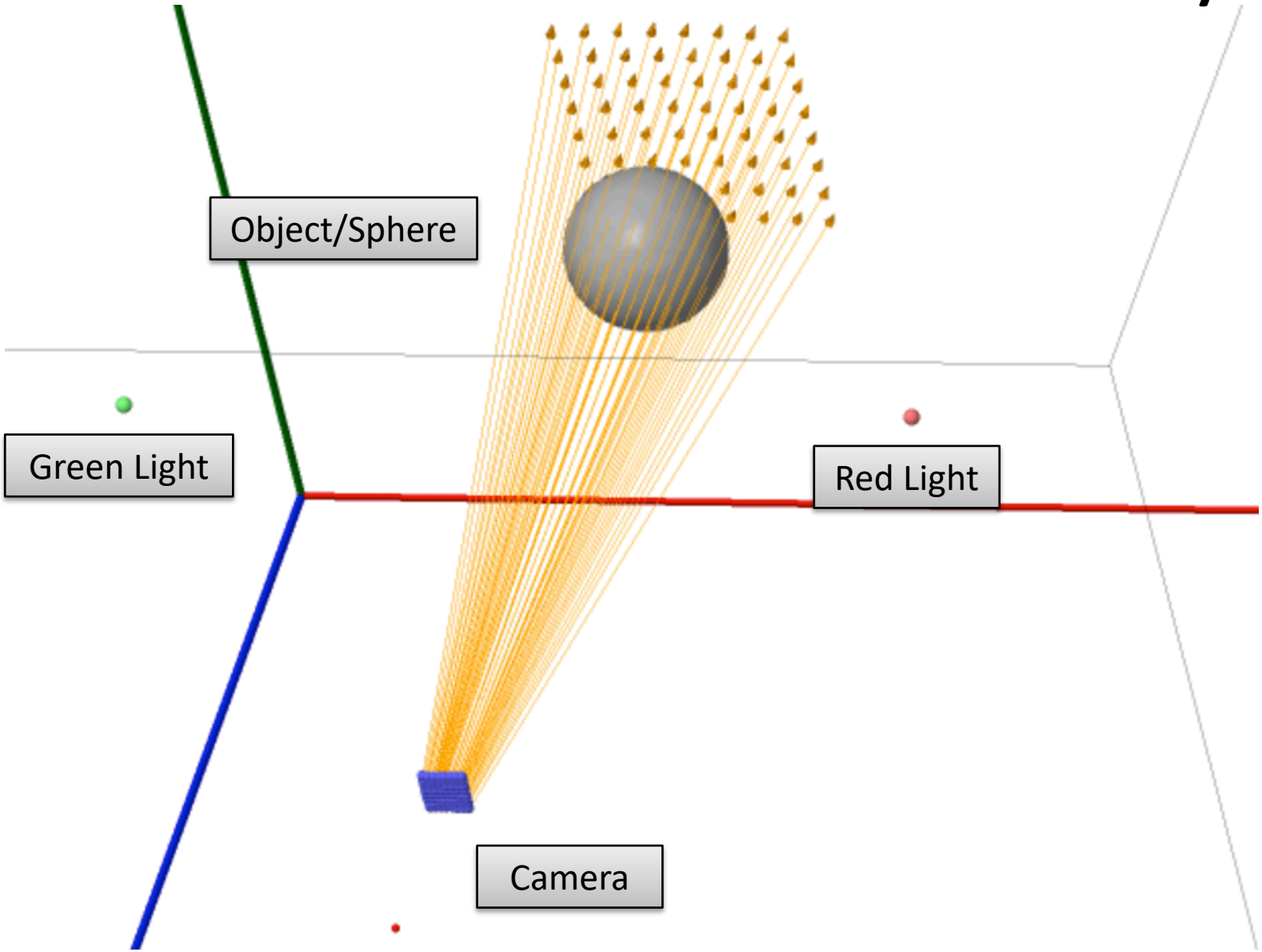


SageMath Notebook
cs410lec14n01

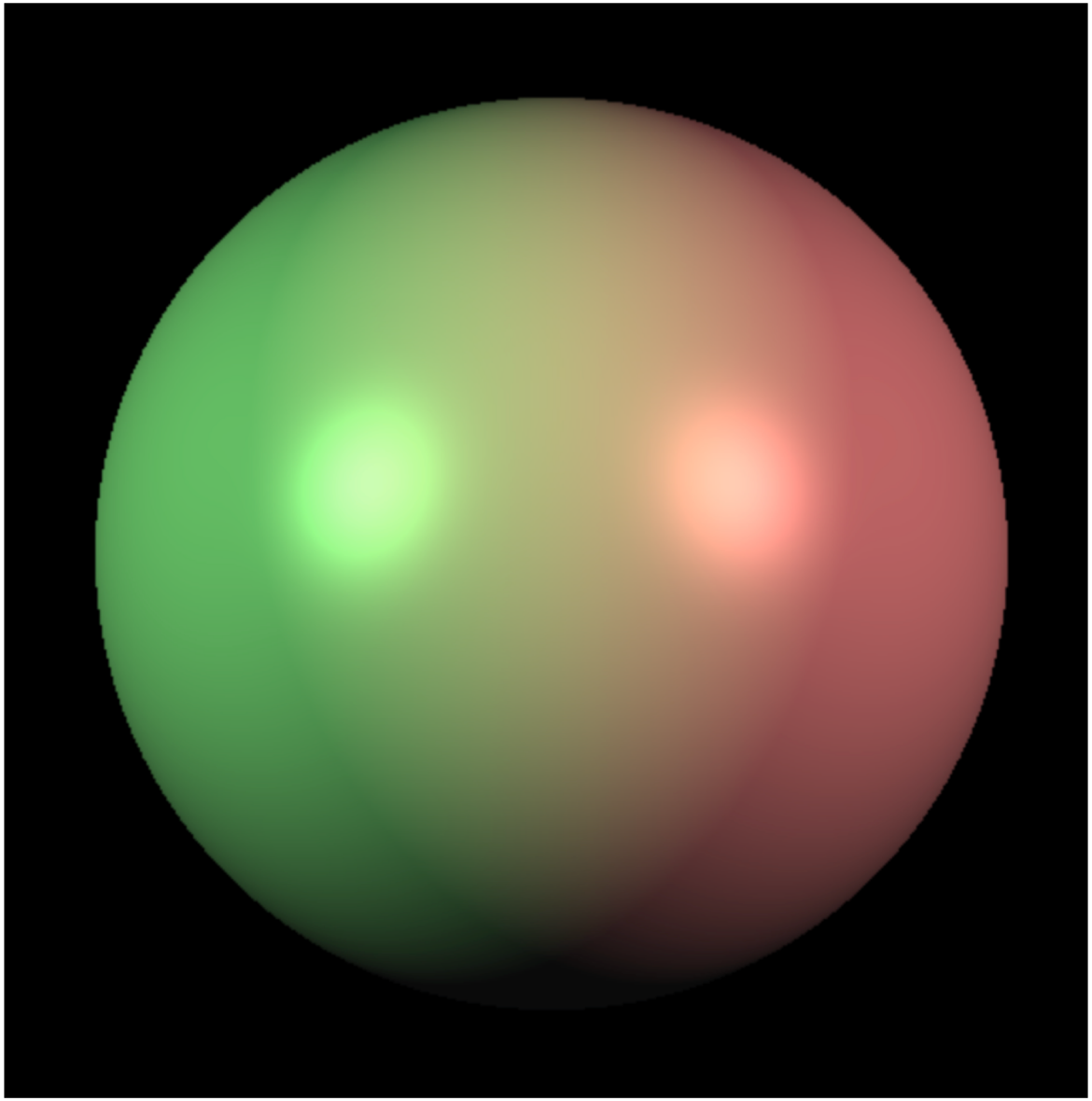
Scene Specification

```
cam = {'eye'  : vector(RR, 3, (25, 25, 60)),
       'look' : vector(RR, 3, (25, 25, 10)),
       'up'   : vector(RR, 3, (0, 1, 0)),
       'bnds' : { 'l' :-1, 'r' : 1, 'b' : -1, 't' : 1},
       'nefa' : { 'near' : -10, 'far' : -60},
       'resl' : { 'width' : 64, 'height' : 64} }
globe = {'c'  : vector(RR, 3, (25,25,10)), 'r' : 4.0 }
mat1  = {'ka' : vector(RR, 3, (0.2, 0.2, 0.2)),
         'kd' : vector(RR, 3, (0.7, 0.7, 0.7)),
         'ks' : vector(RR, 3, (0.5, 0.5, 0.5)) }
ambient = vector(RR, 3, (0.2, 0.2, 0.2))
lights  = [{'p' : vector(RR, 3, ( 5, 32, 30)),
            'e' : vector(RR, 3, (0.5, 1.0, 0.5))},
           {'p' : vector(RR, 3, (45, 32, 30)),
            'e' : vector(RR, 3, (1.0, 0.5, 0.5))} ]
```

Visualize Scene Geometry

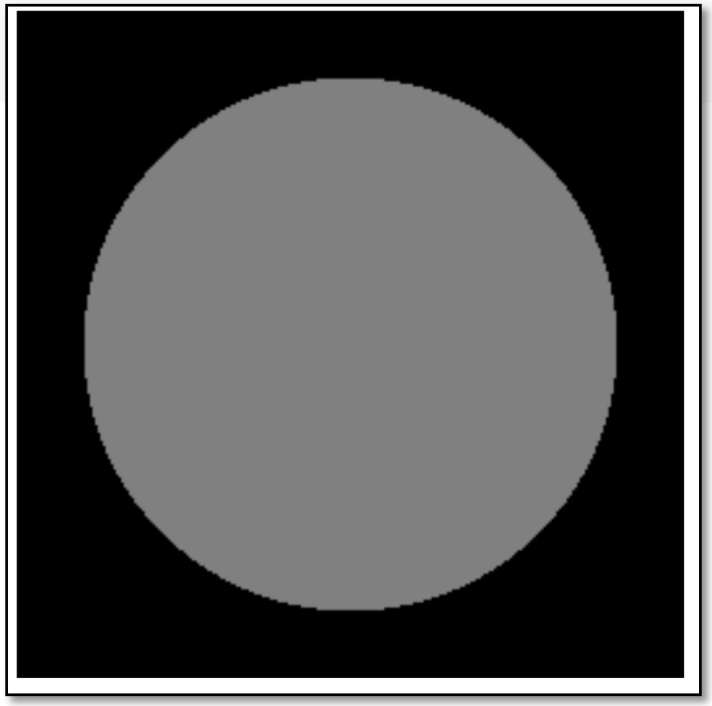


Two Lights Grey Sphere



Dial Back to Boring - Ambient

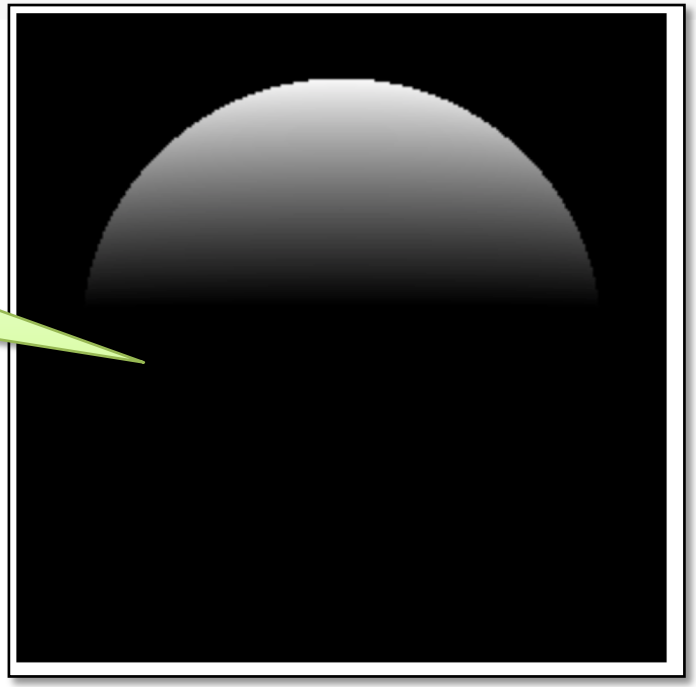
```
# White sphere and only ambient light
mat1['ka'] = vector(RR, 3, (0.5, 0.5, 0.5))
mat1['kd'] = vector(RR, 3, (0.0, 0.0, 0.0))
mat1['ks'] = vector(RR, 3, (0.0, 0.0, 0.0))
ambient    = vector(RR, 3, (1.0, 1.0, 1.0))
lights     = []
```



Diffuse Illumination Only

```
# White sphere with one light directly above
mat1['ka'] = vector(RR, 3, (0.5, 0.5, 0.5))
mat1['kd'] = vector(RR, 3, (1.0, 1.0, 1.0))
mat1['ks'] = vector(RR, 3, (0.0, 0.0, 0.0))
ambient    = vector(RR, 3, (0.0, 0.0, 0.0))
lights     = [{'p' : vector(RR, 3, ( 25, 60, 10)),
               'e' : vector(RR, 3, (1.0, 1.0, 1.0))}]
```

Dark side of the moon

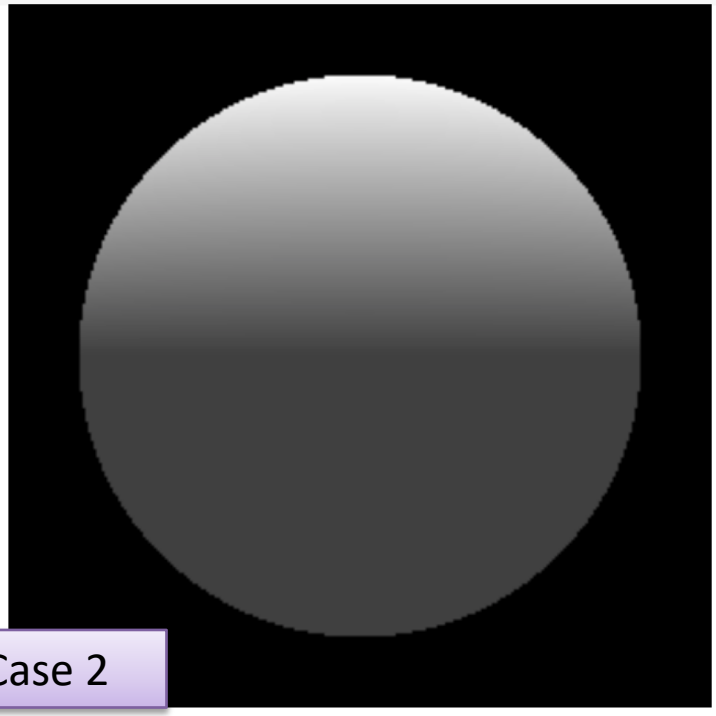


Diffuse Plus Ambient Case 1

```
# White sphere with one light directly above and also
# ambient 'background' illumination
mat1['ka'] = vector(RR, 3, (0.5, 0.5, 0.5))
mat1['kd'] = vector(RR, 3, (1.0, 1.0, 1.0))
mat1['ks'] = vector(RR, 3, (0.0, 0.0, 0.0))
ambient    = vector(RR, 3, (0.5, 0.5, 0.5))
lights     = [{'p' : vector(RR, 3, ( 25, 60, 10)),
               'e' : vector(RR, 3, (0.75, 0.75, 0.75))}]
```

Diffuse Plus Ambient Case 2

```
# White sphere with one light directly above and also
# ambient 'background' illumination
mat1['ka'] = vector(RR, 3, (0.5, 0.5, 0.5))
mat1['kd'] = vector(RR, 3, (1.0, 1.0, 1.0))
mat1['ks'] = vector(RR, 3, (0.0, 0.0, 0.0))
ambient    = vector(RR, 3, (0.5, 0.5, 0.5))
lights     = [{'p' : vector(RR, 3, ( 25, 60000, 10)),
               'e' : vector(RR, 3, (0.75, 0.75, 0.75))}]
```

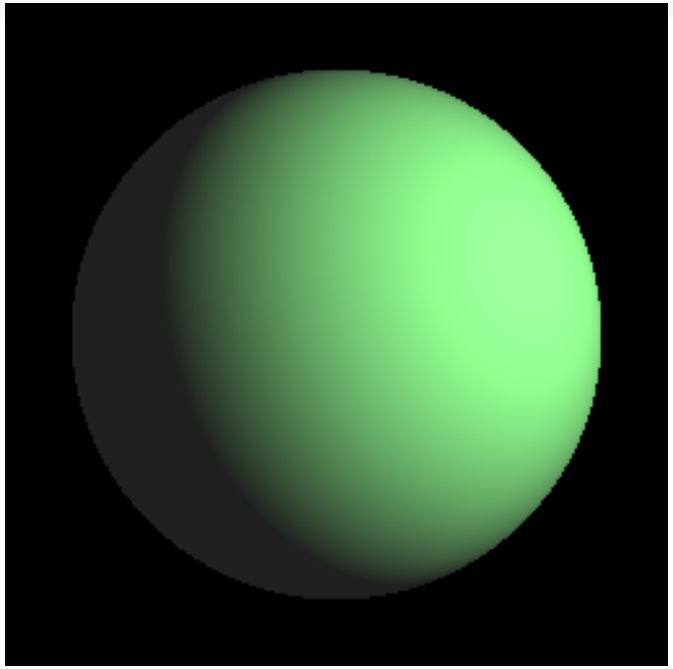


Case 2

Case 1

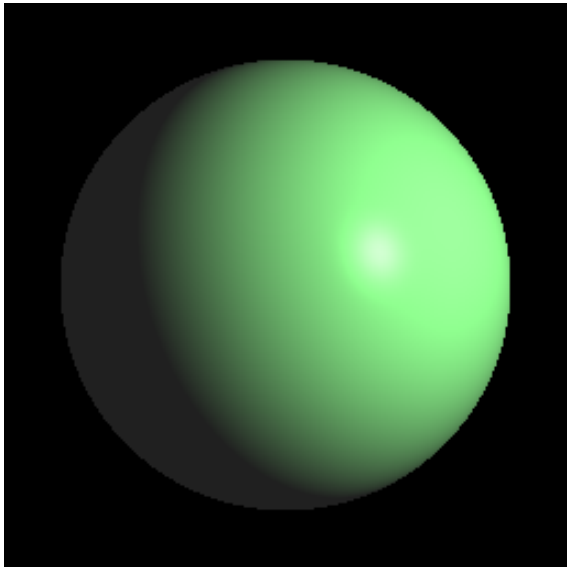
Colored Light Source

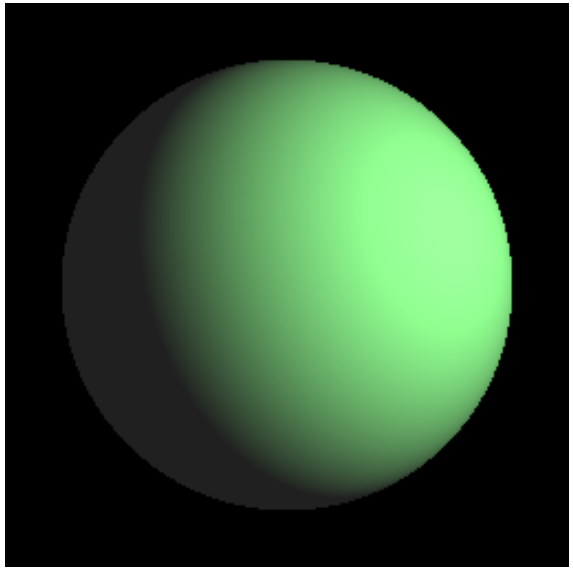
```
# White sphere with one green light off to the right
mat1['ka'] = vector(RR, 3, (0.5, 0.5, 0.5))
mat1['kd'] = vector(RR, 3, (1.0, 1.0, 1.0))
mat1['ks'] = vector(RR, 3, (0.0, 0.0, 0.0))
ambient    = vector(RR, 3, (0.25, 0.25, 0.25))
lights     = [{'p' : vector(RR, 3, (45, 32, 30)),
              'e' : vector(RR, 3, (0.5, 1.0, 0.5))}]
```



Colored Light With Highlight: Case 7

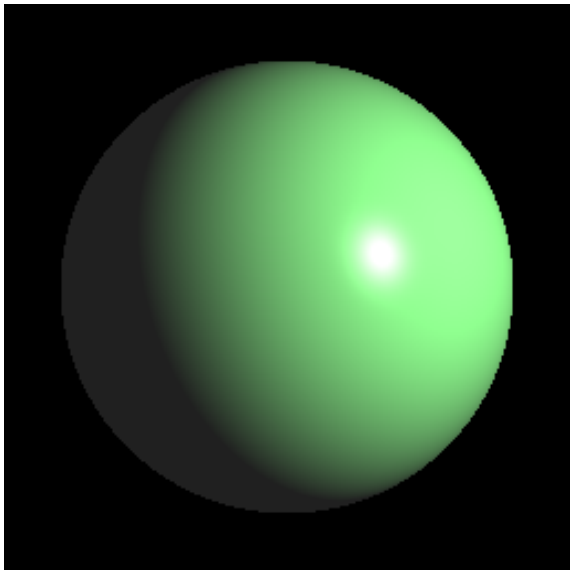
```
# White sphere with one green light off to the right
# and with specular reflection
mat1['ka'] = vector(RR, 3, (0.5, 0.5, 0.5))
mat1['kd'] = vector(RR, 3, (1.0, 1.0, 1.0))
mat1['ks'] = vector(RR, 3, (0.5, 0.5, 0.5))
ambient    = vector(RR, 3, (0.25, 0.25, 0.25))
lights     = [{'p' : vector(RR, 3, (45, 32, 30)),
              'e' : vector(RR, 3, (0.5, 1.0, 0.5))}]
```

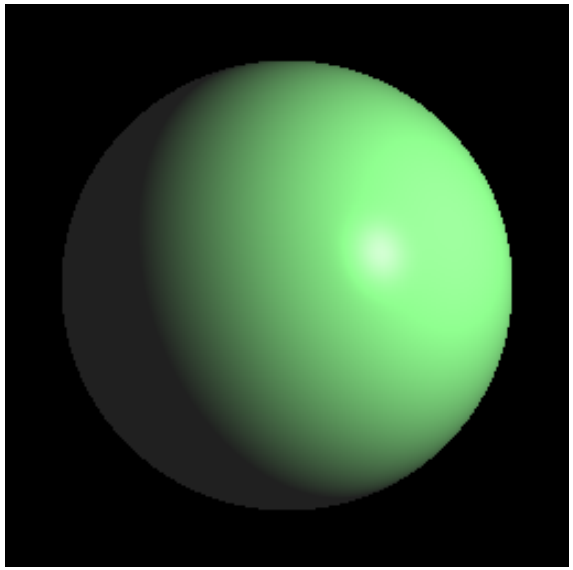




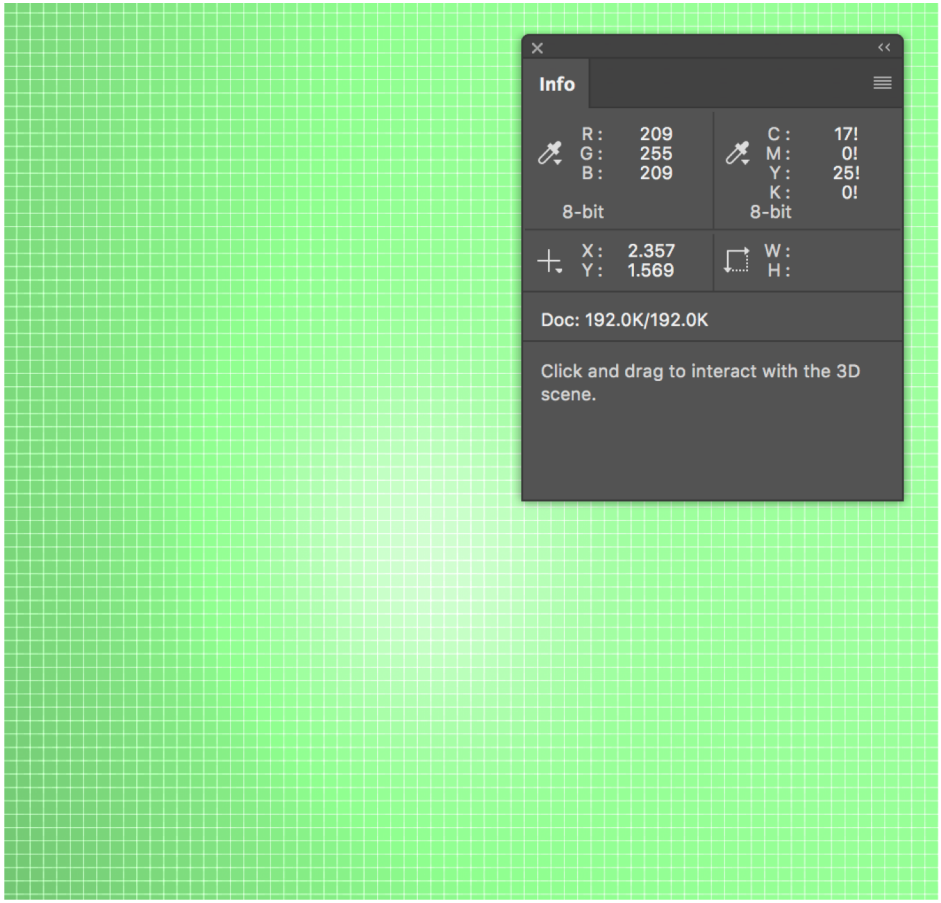
White Light with Green Sphere: Case 8

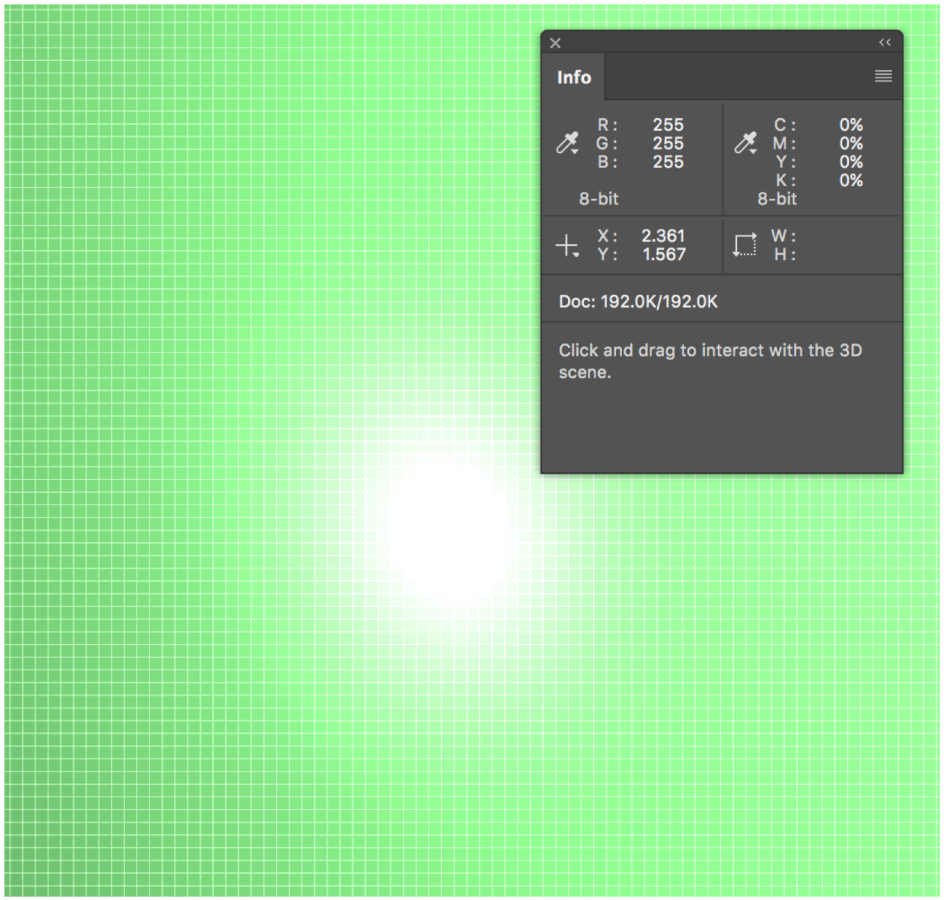
```
# Green sphere with one white light off to the right
# and with specular reflection
mat1['ka'] = vector(RR, 3, (0.5, 0.5, 0.5))
mat1['kd'] = vector(RR, 3, (0.5, 1.0, 0.5))
mat1['ks'] = vector(RR, 3, (0.5, 0.5, 0.5))
ambient    = vector(RR, 3, (0.25, 0.25, 0.25))
lights     = [{'p' : vector(RR, 3, (45, 32, 30)),
             'e' : vector(RR, 3, (1.0, 1.0, 1.0))}]
```





Highlights Zoomed: Case 7 & 8





Highlight takes on the color of the light source.

Part 2: The Code

```
def raySphereRGB(ray, sph) :
    hitp = raySphereTest(ray, sph)
    if hitp[0] :
        ptos = hitp[2]
        snrm = ptos - sph['c'] ; snrm = snrm / snrm.norm()
        color = ambient.pairwise_product(mat1['ka'])
        for lt in lights :
            ptL = lt['p']
            emL = lt['e']
            toL = ptL - ptos; toL = toL / toL.norm()
            if (snrm.dot_product(toL) > 0.0) :
                color += mat1['kd'].pairwise_product(emL) * snrm.dot_product(toL)
                toC    = ray['L'] - ptos; toC = toC / toC.norm()
                spR    = (2 * snrm.dot_product(toL) * snrm) - toL
                CdR    = toC.dot_product(spR)
                if (CdR > 0.0) :
                    color += mat1['ks'].pairwise_product(emL) * CdR**phong
        return [True, color]
    else :
        return [False]
```

Ray Sphere Test

```
def raySphereTest(ray, sph) :
    r  = sph['r']
    Cv = sph['c']
    Lv = ray['L']
    Uv = ray['U']
    Tv = vector(RR, 3, (Cv - Lv))
    v    = Tv.dot_product(Uv)
    csq  = Tv.dot_product(Tv)
    disc = r^2 - (csq - v^2)
    if (disc < 0) :
        return [False]
    else :
        d  = sqrt(disc)
        t  = v - d
        pt = Lv + t * Uv
        return [True, t, pt]
```

Code Implements fast algorithm

Note intermediate variable we have discussed previously 'disc'.

Note that this function returns a list and the first element indicates whether an intersection exists.

Ray Sphere RGB

- This code does all three
  - Ambient, diffuse and specular

```
def raySphereRGB(ray, sph) :
    hitp = raySphereTest(ray, sph)
    if hitp[0] :
        ptos = hitp[2]
        snrm = ptos - sph['c'] ; snrm = snrm / snrm.norm()
        color = ambient.pairwise_product(mat1['ka'])
        for lt in lights :
            ptL = lt['p']
            emL = lt['e']
            toL = ptL - ptos; toL = toL / toL.norm()
            if (snrm.dot_product(toL) > 0.0) :
                color += mat1['kd'].pairwise_product(emL) * snrm.dot_product(toL)
                toC    = ray['L'] - ptos; toC = toC / toC.norm()
                spR    = (2 * snrm.dot_product(toL) * snrm) - toL
                color += mat1['ks'].pairwise_product(emL) * toC.dot_product(spR)**16
                # print(ray['L'], toC.dot_product(spR)**4)
        return [True, color]
    else :
        return [False]
```

Once Through Carefully 1

First, pass ray and sphere to intersection code and if the response returns true, then compute and return illumination, otherwise return false

```
def raySphereRGB(ray, sph) :
    hitp = raySphereTest(ray, sph)
    if hitp[0] :
        ptos = hitp[2]
        snrm = ptos - sph['c'] ; snrm = snrm / snrm.norm()
        color = ambient.pairwise_product(mat1['ka'])
        for lt in lights :
            ptL = lt['p']
            emL = lt['e']
            toL = ptL - ptos; toL = toL / toL.norm()
            if (snrm.dot_product(toL) > 0.0) :
                color += mat1['kd'].pairwise_product(emL) * snrm.dot_product(toL)
                toC    = ray['L'] - ptos; toC = toC / toC.norm()
                spR    = (2 * snrm.dot_product(toL) * snrm) - toL
                color += mat1['ks'].pairwise_product(emL) * toC.dot_product(spR)**16
                # print(ray['L'], toC.dot_product(spR)**4)
        return [True, color]
    else :
        return [False]
```

Once Through Carefully 2

Retrieve the 3D position of the point of intersection on the sphere and then use that position to computer the surface normal. Recall that it is particularly easy to computer sphere surface normals.

```
def raySphereRGB(ray, sph) :
    hitp = raySphereTest(ray, sph)
    if hitp[0] :
        ptos = hitp[2]
        snrm = ptos - sph['c'] ; snrm = snrm / snrm.norm()
        color = ambient.pairwise_product(mat1['ka'])
        for lt in lights :
            ptL = lt['p']
            emL = lt['e']
            toL = ptL - ptos; toL = toL / toL.norm()
            if (snrm.dot_product(toL) > 0.0) :
                color += mat1['kd'].pairwise_product(emL) * snrm.dot_product(toL)
                toC    = ray['L'] - ptos; toC = toC / toC.norm()
                spR    = (2 * snrm.dot_product(toL) * snrm) - toL
                color += mat1['ks'].pairwise_product(emL) * toC.dot_product(spR)**16
                # print(ray['L'], toC.dot_product(spR)**4)
        return [True, color]
    else :
        return [False]
```

Once Through Carefully 3

Create the color starting with the ambient illumination computation.
That is the pairwise product of the material property 'ka' and strength of the ambient light source.
Recall 'ambient' is a vector and hence the ability to do a pairwise product in one method call.

```
def
    if hitp[0] :
        ptos = hitp[2]
        snrm = ptos - sph['c'] ; snrm = snrm / snrm.norm()
        color = ambient.pairwise_product(mat1['ka'])
        for lt in li
            ptL = lt
            emL = lt
            toL = pt
            if (snrm
                colo                                    oL)
                toC
                spR
                colo                                    R)**16
                # pr
        return [True,
    else :
        return [False]
```

```
pretty_print(ambient)
pretty_print(mat1['ka'])
color = ambient.pairwise_product(mat1['ka']) ;
pretty_print(color)
```

```
(0.200000000000000, 0.200000000000000, 0.200000000000000)
(0.200000000000000, 0.200000000000000, 0.200000000000000)
(0.0400000000000000, 0.0400000000000000, 0.0400000000000000)
```

Once Through Carefully 4

The next step iterates over all light sources.

'toL' is a unit length vector pointing toward the light source.

If and only if dot product of 'toL' and 'snrm' is positive should illumination be computed (think through why this is true).

The illumination is a combination of diffuse and specular (next slides).

```
def r
    h
    i
        snrm = ptos - sph['c'] ; snrm = snrm / snrm.norm()
        color = ambient.pairwise_product(mat1['ka'])
        for lt in lights :
            ptL = lt['p']
            emL = lt['e']
            toL = ptL - ptos; toL = toL / toL.norm()
            if (snrm.dot_product(toL) > 0.0) :
                color += mat1['kd'].pairwise_product(emL) * snrm.dot_product(toL)
                toC    = ray['L'] - ptos; toC = toC / toC.norm()
                spR    = (2 * snrm.dot_product(toL) * snrm) - toL
                color += mat1['ks'].pairwise_product(emL) * toC.dot_product(spR)**16
                # print(ray['L'], toC.dot_product(spR)**4)
        return [True, color]
    else :
        return [False]
```

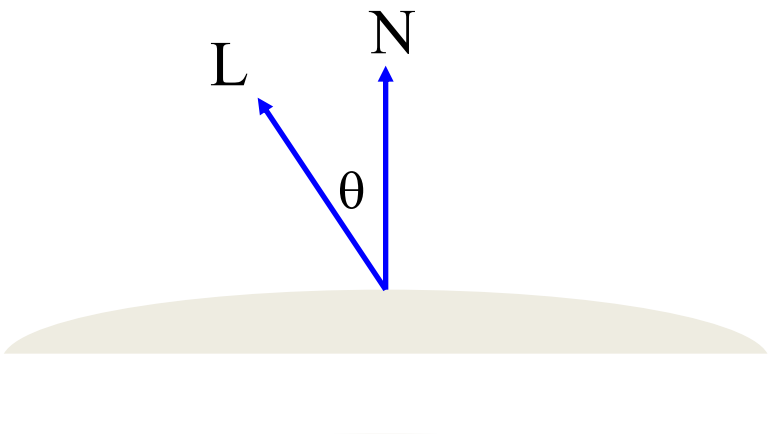
Recall: Diffuse Reflection

- Illumination/area drops by θ

$$i_r = b_r \cos\theta \quad i_g = b_g \cos\theta \quad i_b = b_b \cos\theta$$

- Where light L emits light with brightness B:

$$B = \begin{vmatrix} b_r \\ b_g \\ b_b \end{vmatrix}$$



- And as SageMath code

```
color += mat1['kd'].pairwise_product(emL) * snrm.dot_product(toL)
```

Once Through Carefully 5

Last but not least, specular illumination.

Superficial observation, three lines of code compared with one.

Substantive observation, there are three steps

1. Determine vector indicating direction to Camera, called ‘toC’
2. Determine vector that is reflection ray, called ‘spR’
3. Mudulate specular illumination by Phong Shading formula

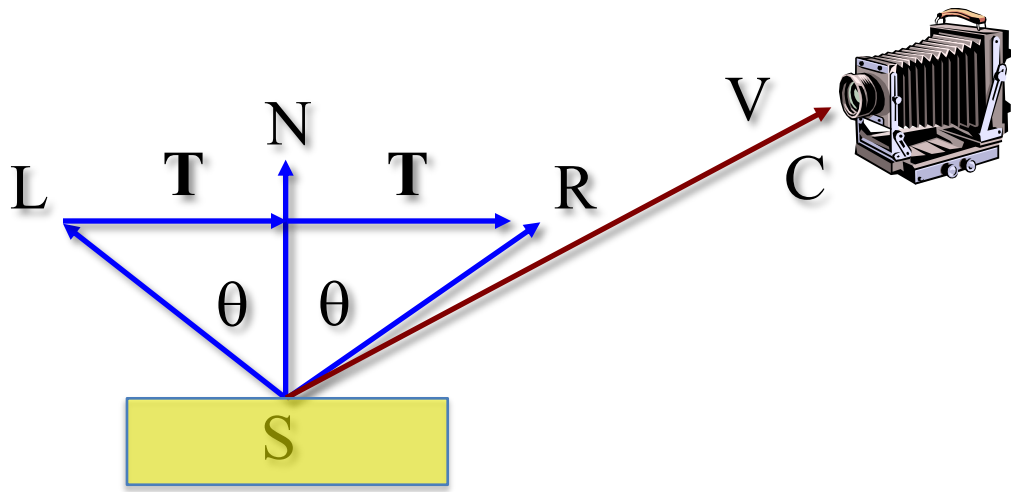
```
def r
    h
    i
        for lt in lights :
            ptL = lt['p']
            emL = lt['e']
            toL = ptL - ptos; toL = toL / toL.norm()
            if (snrm.dot_product(toL) > 0.0) :
                color += mat1['kd'].pairwise_product(emL) * snrm.dot_product(toL)
                toC    = ray['L'] - ptos; toC = toC / toC.norm()
                spR    = (2 * snrm.dot_product(toL) * snrm) - toL
                color += mat1['ks'].pairwise_product(emL) * toC.dot_product(spR)**16
                # print(ray['L'], toC.dot_product(spR)**4)
        return [True, color]
    else :
        return [False]
```

Review: Reflection and Viewer

- V is the unit vector from the surface point to the camera:

  V = (C – S)/|C – S|

- Calculating R is more involved



Review: Reflection Vector

- Let NL be the projection of L onto N
  - Assume N is a unit vector

    NL = (L·N)N
- T is defined as

    T = NL – L = (L·N)N – L
- R is L + 2T

$$R = L + 2\left(\left(L \cdot N\right)N - L\right)$$

$$= 2\left(L \cdot N\right)N - L$$

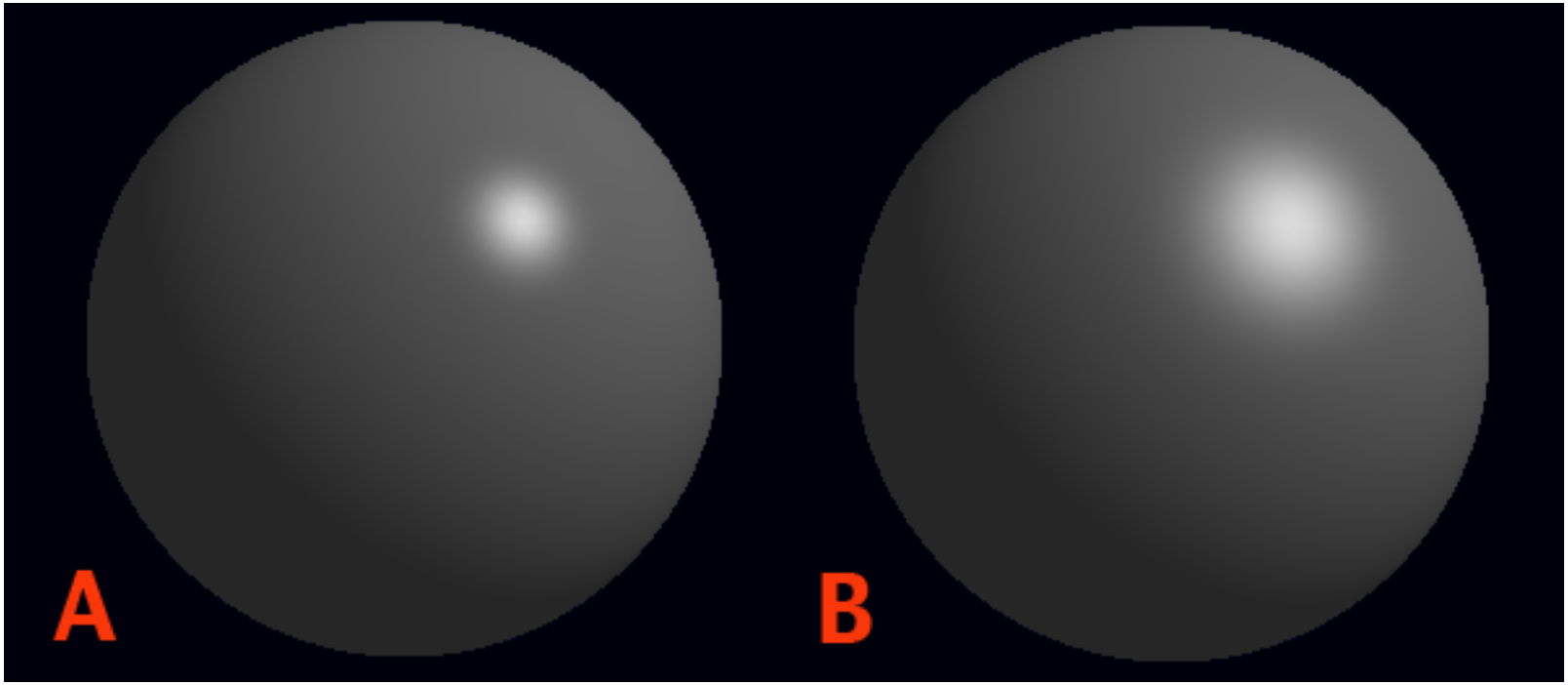
Review: Phong Shading

$$I = k_s B\left(\cos\Phi\right)^{\alpha}$$

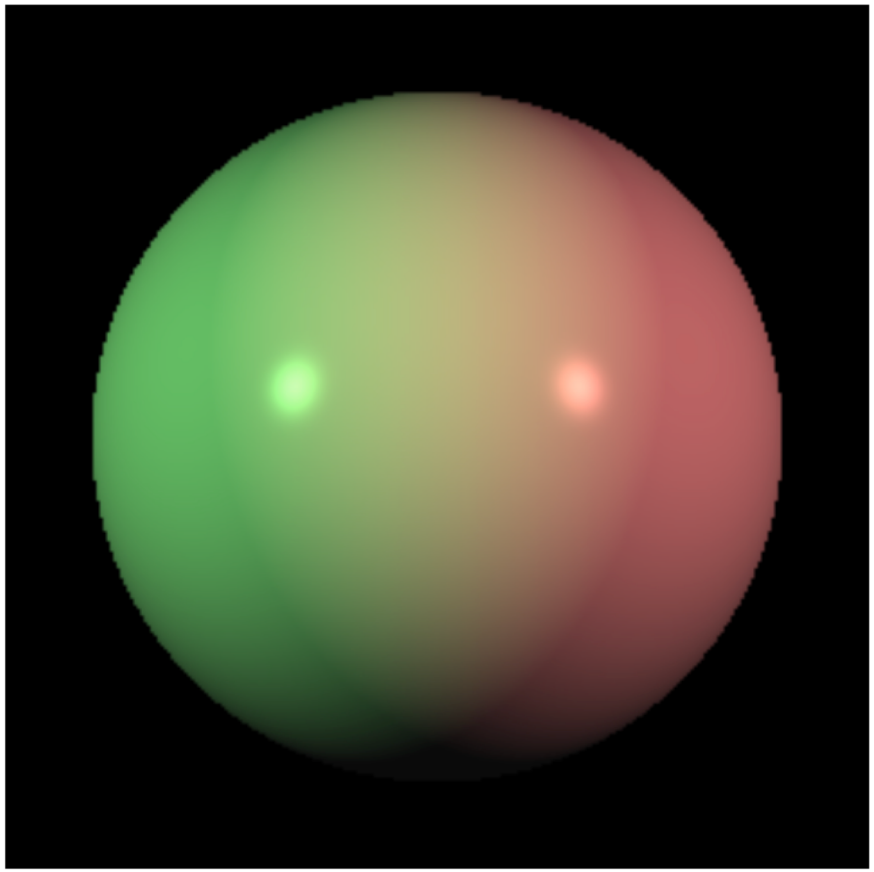
- Φ is the angle between the viewing ray and the angle of reflection.
- α is the so-called Phong constant, expressing how "shiny" an object is
  - Mirrors: α = 200
  - Dull objects: α = 5 … 50

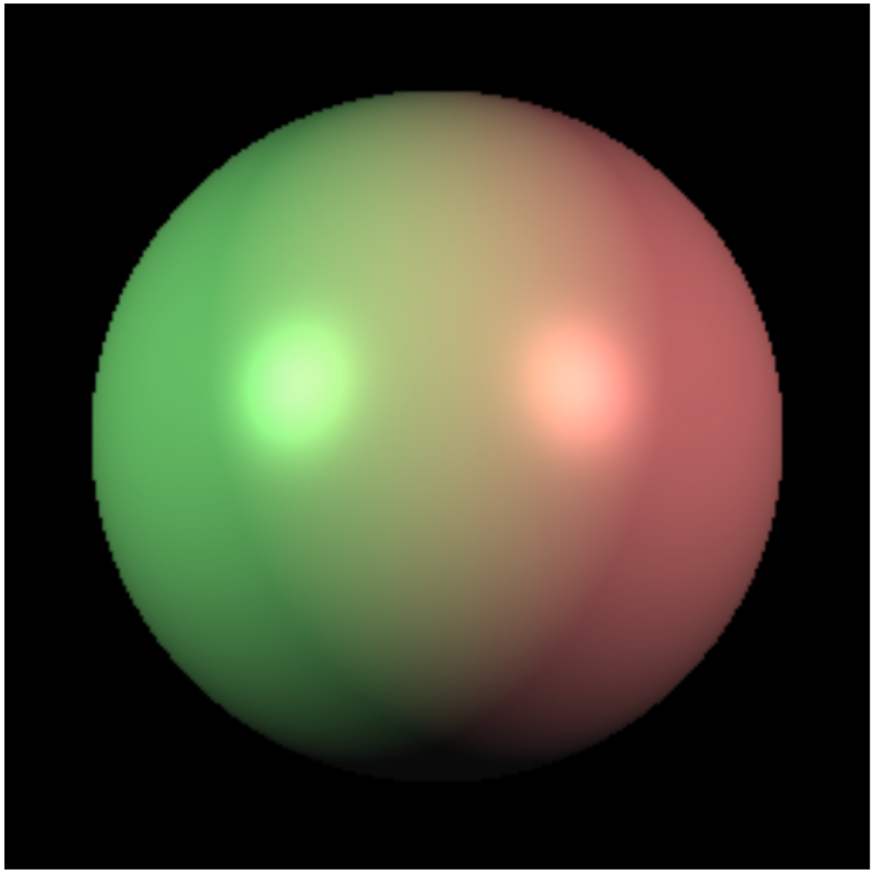
Review: Changing α

- Consider which has the larger alpha?



Phong 16 vs 200

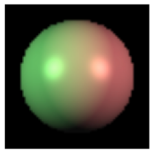




Back to Ray Tracing

- Put it all together and generate an image

```
img = Image('RGB', (width, height), 'black')
pix = img.pixels()
for i in range(width) :
    for j in range(height) :
        res = raySphereRGB(pixelRay(i,(height - j -1)), globe)
        if res[0] :
            pix[i,j] = tuple(map(lambda(x) : ZZ(max(0,min(255,round(255.0 * x)))), res[1]))
img
```



- Note the code is simple at this level of abstraction – this is as it should be.