ETGG1803	Lab7: **RayTracer part II / II**	Points: **63**[1]
Assigned: **3/27/2017 (MW)**	Due: **Before noon on 4/7/2017** (don't procrastinate – I'm making it a bit longer than normal so you can attempt the bonus -- we'll cover new material and start a new lab before this due date)	

Overview:

In this lab, we'll finish our solid-color ray-tracer to include the standard-lighting equation (phong) and shadows. This will be the end of the RayTracer!

Tasks:

1. Get a working copy of Lab6 (preferably your own, but mine should be on the web page if your lab6 wasn't fully finished)
2. (**6 points**) Add these items to the VectorN class:
    a. A new "pairwise-mult" method in our VectorN class. This is the ⊗ symbol I used on the slides.
    b. A new "clamp" method which takes a low and high scalar value. If any one of the entries of the VectorN are below the low or above the high, move them within range. Default to 0.0 and 1.0 for low, high.
    c. Don't forget docstrings and type-checking (with ValueErrors).
3. (**5 points**) Modify objects3d such that each object takes a **material dictionary** in place of the single Vector3 color we had in Lab6.
    a. This dictionary should include at least an "ambient", "diffuse", "specular" key, where each value is either:
        i. a Vector3 for a solid color
        ii. (If doing the bonus features, *also* allow these to be a filename of an image)
    b. Also include a scalar "shininess" (some places in the slides it might be called "hardness") key
4. (**4 points**) Modify the main program to create a scene using the new material scheme and also create some light sources.
5. (**5 points**) Make a new (Point) Light class and make the raytracer store a list of such lights.
6. (**8 points**) When a hit occurs, we need some additional information about the hit point (in the RayCollision class)
    a. The normal vector at that point
    b. (if doing the uv-mapping bonus or the mirror-reflections, you'll need these as well)
    c. (**+4 points**) collect normals from the Box and TriangleMesh classes – if you don't attempt this, just remove these objects from the scene)
    d. *I suggest making sure your ray-tracer still works after adding this bit of code (before attempting the rest of the lab)*
7. (**20 points**) Integrate the standard lighting equation
    a. Make sure your code works with multiple light sources!
8. (**15 points**) Integrate shadows

[1] Max of 100 (there are 135 points total, but I'll only allow the total score to go up to 100 points, though).

Sample Results

a. Camera: Pos = (-15, 19, -30) Coi = (2, 5, 3) Up = (0, 1, 0)
Fov = 60.0 Near = 1.5 Aspect = 1.5

b. Ambient Light: (1, 1, 1)

c. Scene:

i. Sphere: Center = (2, 8, 5) Radius = 7.0
Material = { amb = (0.3, 0, 0), diff = (1, 0, 0), spec = (1, 1, 1), shiny = 10.0 }

ii. Green plane:
Normal = (0, 1, 0) Point = (0, 5, 0)
Material = { amb = (0, 0.5, 0), diff = (0, 1, 0), spec = (1, 0, 0), shiny = 2.0 }

iii. Blue plane: Normal = (0.1, 1, 0).normalized() Point = (0, 4, 0)
Material = { amb = (0, 0, 0.1), diff = (0, 0, 1), spec = (1, 0, 1), shiny = 6.0 }

iv. Box: Center = (15, 12, -5) Hext = (9, 5, 2)
Rotate 45 degrees around the z axis, then 15 degrees around the x axis.
Material = { amb = (0.5, 0.3, 0.1), diff = (1, 1, 0), spec = (0.5, 1, 0.5), shiny = 30.0 }

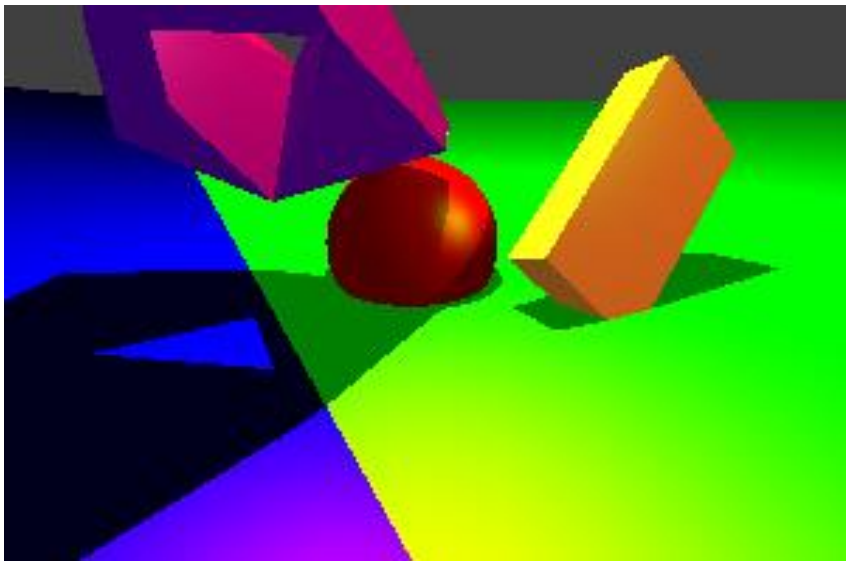
v. Polymesh: Mesh = “tri.obj” (on ssugames) Offset = (-10, 22, 0)
Material = { amb = (0.2, 0, 0.4), diff = (0.7, 0, 1), spec = (1, 1, 1), shiny = 50.0 }

d. Light Sources:

i. Light #1: pos = (0, 50, 0), diff = (1, 1, 1), spec = (1, 1, 1)

ii. Light #2: pos = (50, 50, -50), diff = (0.3, 0, 0), spec = (0, 0.6, 0)

e. Rendering (300 x 200 screen) [enlarged a bit to show detail]



Bonus Features

a. (**+20 points**) specular-based **Reflections**

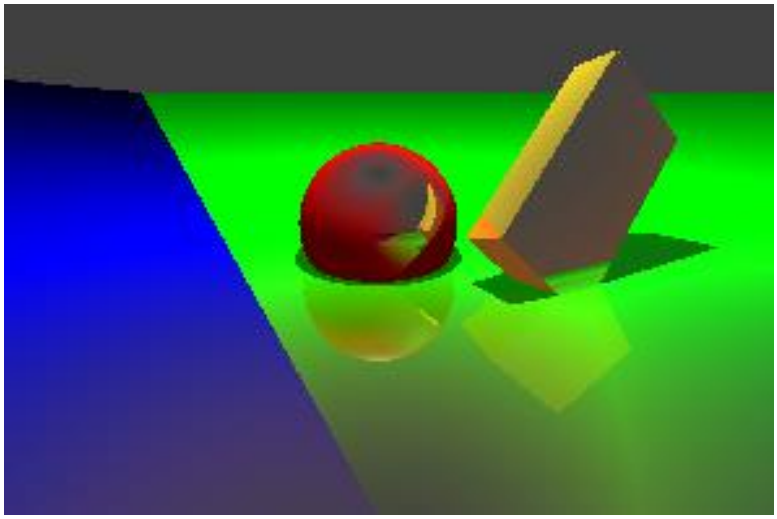
i. When spawning new rays (perhaps in render_one_line), add a weight attribute (1.0) to the ray.

ii. Calculate the lit color as in the base part of the lab, but…

iii. If the weight on the ray is above some threshold (e.g. 0.1), do the following:

1. Calculate the maximum specular strength (without raising to the shininess power)
2. Create a ray in which the direction of the ray is the reflection of the negation of the initial ray direction about the normal (hint: use the same technique we did when calculating the specular lighting)
3. Send this ray out (just as we did initially) recursively to get a reflection color. But…when you get this color mix it with the lit color of the object, using the weight of the ray times the base color and (1 – weight) on the reflection color.

iv. Sample Results:



b. (**+12 points**) **Anti-Aliasing**

i. Instead of sending out one ray per pixel, send out many, where each ray origin on the view plane is jittered by a random *float* between -1.0 and +1.0 pygame pixel.

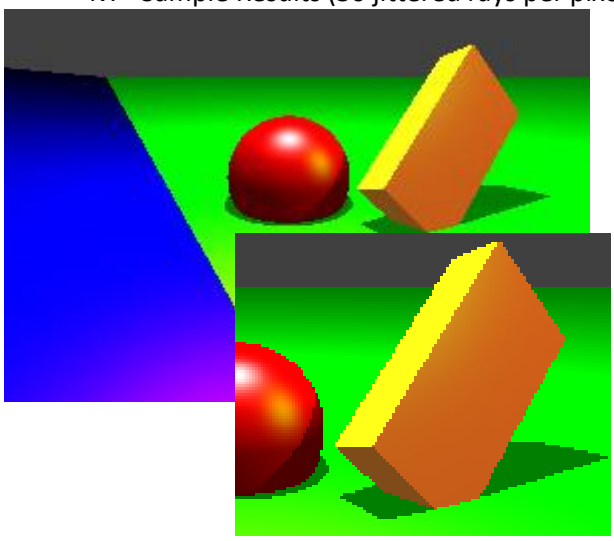
ii. Average the colors together using a weighting system (commonly a Gaussian function like this)

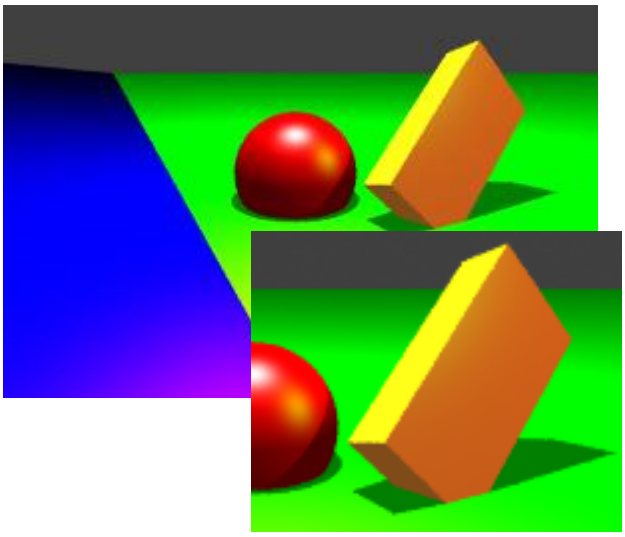
$$weight(\Delta x, \Delta y) = e^{-(\frac{\Delta x^2}{2\sigma^2}+\frac{\Delta y^2}{2\sigma^2})}$$

$where\ \Delta x\ and\ \Delta y\ are\ the\ offsets\ from\ above\ and\ \sigma\ is\ the\ standard\ deviation\ (I\ used\ 0.3)$

iii. Make sure to divide the final color by the total of all weights used (so the colors aren't brightened or darkened)

iv. Sample Results (50 jittered rays per pixel)





c. (**+30 points**) **Texture mapping**

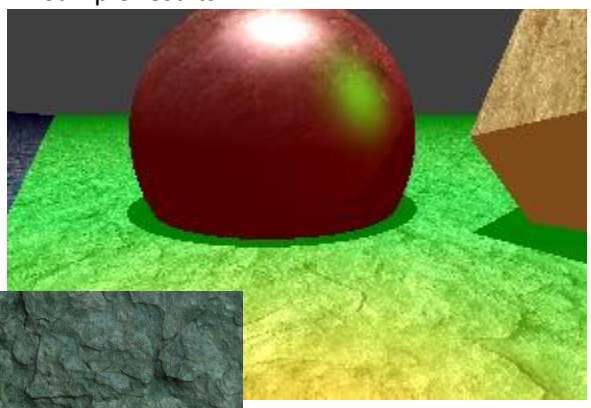
i. Instead of only allowing solide-colors for ambient, specular, and diffuse colors (and the shininess value), allow a texture to be used in its place.

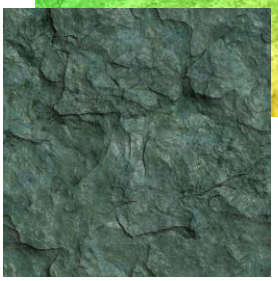
ii. A part of this process involves calculating a uv coordinate for each ray-object collision point. These uv coordinates normally range from 0…1 and correspond to a percentage of the way across an image in the horizontal and vertical direction. Each primitive will calculate the uv coordinate slightly differently.

1. For sphere, this is a good reference: https://www.mvps.org/directx/articles/spheremap.htm
2. For boxes, use the local coordinate we used in hit detection, and stretch the image across each side of the box.
3. For planes, you'll need to create (any) two perpendicular vectors that are perpendicular to the plane's normal. Project the hit point onto these two vectors and modulo the projection distance to get the uv coordinate.
4. I would recommend ignoring Triangles.
5. For TriangleMeshes, in the blender exporter, enable uv coordinates (and look at the vt lines in the text file).

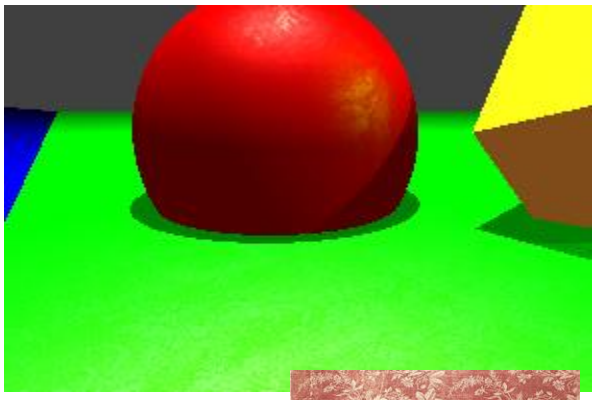
iii. Once you have the uv coordinates, use surf.get_at to get a color value – use that instead of a solid-color material color for diffuse, specular, and ambient. For shininess, you can pick out just the red component and use that…

iv. Sample results:





all diffuse material colors from this texture.



all specular material colors from this texture.

d. (**+10 points**) **Raytraced "movie"**

i. Rotate the camera and generate a large-ish number (100+) of still images

1. Hint1: use sin and cos on the z plane to adjust the camera position
2. Hint2: use pygame.image.save to save the contents of the screen to an image file.
3. You might want to let this run over-night☺

ii. Use Blender (https://www.youtube.com/watch?v=OGapXgrLOnI) to create a video file based on your images (you can use another movie creator if you like).

iii. Sample results: **https://youtu.be/Ut7PKiPrdHo** (360 still images 600 x 400)