

ETGG1803

Lab9: The Raytracer Completed!

Points: 43

Assigned: 2/23/2010 (MW), 2/24/2010 (TR)

Due: 3/5/2010 (Friday) by 11:59pm

Objectives:

- Implement Phong Shading with multiple lights
- Implement Shadows

Tasks:

1. math3d.py modifications:
 - a. (2 points) Include a “tensor” product function in math3d.py:

$$\vec{A} \otimes \vec{B} = \begin{bmatrix} A_x * B_x \\ A_y * B_y \\ A_z * B_z \end{bmatrix}$$

- b. (1 point) Add a new clamp method in the Vector3 class, which clamps any component which is over the passed scalar. For example:

```
v = math3d.Vector3(0.5, 1.5, 2.5)
w = v.clamp(1.0)
print w # Prints 'Vector3<0.5, 1.0, 1.0>'
```
2. materials.py modifications:
 - a. (2 points) Convert all “simple” color attributes to:
 - i. kAmbient: a Vector3 which represents the ambient *reflective* color.
 - ii. kDiffuse: a Vector3 which represents the diffuse *reflective* color.
 - iii. kSpecular: a Vector3 which represents the specular *reflective* color.
 - iv. kShiny: a scalar which represents the “focus” of the specular hotspot (1.0 = unfocused, 50.0 = very focused)
 - b. (1 point) modify the getColor methods of SolidColor and CheckerboardMaterial so they return a tuple of (kAmbient, kDiffuse, kSpecular, kShiny).
 3. (2 points) raytracer_main modifications:
 - a. You’ll now need to change the way you create materials. Example:

```
redMat = materials.SolidColor(math3d.Vector3(1,0,0))
redMat = materials.SolidColor(math3d.Vector3(0.4,0,0), \# kA
                               math3d.Vector3(1,0,0), \# kD
                               math3d.Vector3(1,1,1), \# kS
                               45.0) \# kShiny
```
 - b. Create a light and add it to the scene (see step 4 and 5a-5c)
 4. (2 points) objects3d.py modifications:
 - a. Add a **Light** class. The attributes should be:
 - i. pos: the position of the light
 - ii. LDiffuse: The diffuse light color emitted by this light.
 - iii. LSpecular: The specular light color emitted by this light.

5. scene.py modifications:

- a. Add a **LAmbient** attribute (you can initialize it to `Vector3(1,1,1)`)
- b. Add an attribute for a list of lights, initialize it to an empty list.
- c. **(1 point)** Add an **addLight** method (similar to `addObject`).
- d. Modify the end of `castRay` like this:

```
if closestHit == None:    # R hit nothing in the scene.
    return self.backgroundColor
else:
    # Get the actual hit point
    P = R.getPoint(closestHit.t)
    # Get the color of the hit object at P.
    color = closestHit.O.material.getColor(P, None)
    return color
    return self.litColor(closestHit)
```

- e. **(16 points)** Add a new method, **litColor**:
 - i. Takes a `HitInfo` object as an argument.
 - ii. Calculate the hit point being illuminated, `P`, using the `Ray` and `t` value stored within the `HitInfo` object.
 - iii. Extracts the `kA`, `kD`, `kS`, and `kShiny` values for the material at `P` (using the `.O` field of the `HitInfo` object, `P`, and the normal of the `HitInfo` object).
 - iv. Calculates the `cAmbient` color at point `P`.
 - v. Loops through all lights in the scene, adding their `cDiff` and `cSpec` contributions.
 - vi. Returns the `cTotal` (`cAmb + cDiff + cSpec`) color, clamped to 1.0.
- f. **(7 points)** Add a new method, **shadowTest**:
 - i. Takes two arguments:
 1. A `Ray`
 2. An object to ignore
 3. The distance from the hit point to the light (call it `Ldist`).
 - ii. Loops through the list of objects, similar to `castRay` with two differences:
 1. If `o` is the object to ignore, skip it.
 2. If *any* object is hit *and* that object has a `t`-value smaller than `Ldist`, return `True`.
 3. Otherwise, if no objects are hit within `Ldist`, return `False`.
- g. **(9 points)** Test for shadows in **litColor**.
 - i. Inside the loop in which you calculate `cDiff` and `cSpec` additions for each light...
 - ii. Create a `Ray`:
 1. The origin is `P`, the hit point.
 2. The direction is the direction towards the light (you probably already have this as part of your `cDiff` calculation)
 3. The `Ldist` value is the distance between `P` and the light source.
 - iii. Call `shadowTest`, passing it the ray you just created (NOT the ray in the `HitInfo` object)
 - iv. If it comes back true, don't add anything to `cDiff` or `cSpec`.
 - v. If it comes back False, continue with the loop as normal.

Sample Image:

Camera: (0,5,20) pointing at (0,0,0)

Ball: Pos=(0,3,0), radius=2.5, material=SolidColor(kA=(0.4,0,0), kD=(1,0,0), kS=(1,1,1), kShiny=50)

Plane: Normal=(0,1,0), d=0, material=Checker(kA1=(0,0.4,0), kD1=(0,1,0), kS1=(1,1,1), kShiny1=50,
kA2=(0,0,0.4), kD2=(0,0,1), kS2=(1,1,1), kShiny2=50),
spacing=1.0)

Light: Pos=(0,10,0), LD=(1,1,1), LS=(1,1,1)

