Short Programming Assignment #5: Diffuse Illumination

5 things you have to do:
1. in model.cpp - uncomment the lights list
2. in ray.h - nothing to do here but remember this is where the light_t class is defined; location & emissivity are the two new attributes that will be in the input file (see pg 3 of Unit 13 notes)
3. light.cpp - VERY similar to material.cpp (except for illuminate method)
   a. constructor: is almost exactly the same as in material.cpp, except the attributes (and mask value) are different
   b. two getter methods so can access the light_t data members:
      • getemiss (drgb_t *emiss) -- requires 1 line of code (using pix_copy)
      • getloc (vec_t *loc) -- requires 1 line of code (using vec_copy)
   c. printer class method - similar to other printer methods
   d. light_list_print function  *C code * almost exactly like material_list_print except you’re dealing with lights instead of materials (add to rayhdrs.h file)
   e. illuminate class method
      • *** explained below ***
4. in raytrace.c, write add_illuminate function (see pg 5 of Unit 13 notes)
   • *** explained below ***
   • NOTE: the notes call this function add_illumination( ...) so there is a naming discrepancy between the notes and the actual files that I have - it may very well work fine if you name it add_illumination rather than add_illuminate (I didn’t check to see what else would be affected, if anything)
5. in the raytrace(...) function, add the call to the add_illuminate function right after the getambient( ) function call

From the above to-do list, #3e and #4 require more explanation:

Starting with the easiest - #4 - see pg 5 of the Unit 13 notes - most of that add_illuminate function is shown right there in the notes, with pseudo code for where the while loop goes - it’s the only part that’s missing, and it’s very similar to the other while loops (missing parts are in blue):

```c
while (not at end of list) {
    use get_entity( ) to get the next light off of the list
    light->illuminate(model, base, hitobj, pixel); - - - given in the notes
    traverse list to the next item
}
```
Lastly, #3e from above - the illuminate method from the light_class (pg 7 Unit 13 notes):

```cpp
void light_t::illuminate(model_t *model, vec_t *base, object_t *hitobj, drgb_t *pixel) {
    vec_t dir;       // unit direction to light from hitpt
    object_t *obj;  // closest object in dir to light
    double close;   // dist to closest object in dir to light
    double cos;     // of angle between normal and dir to light
    double dist;    // to the light from hitpoint
    drgb_t diffuse = {0.0, 0.0, 0.0};

    /* Compute the distance from the hit to the light (dist) and */
    /* and a unit vector in the direction of the light from hitpt */
    /* (dir) */
    3. use the getlast_hit method (from the object class) to get the last_hit from the hitobj and
    store into the lasthit vector declared from step 1 above
    4. find the distance between the hit (lasthit) and the light (location - - the protected data
    member coming from the light_t object that you can access directly since this is a class
    method)
    a. this requires first getting the vector between those two points (vec_diff) - gets put
    into dir (declared above)
    b. then finding the length of that vector which will be assigned to dist (variable declared
    above)
    c. then get the unit vector (of dir - declared above)
    5. use getlast_normal method (from the object class) to get the last_normal from the hitobj;
    put into lastnorm variable from step 2
    /* Test the object for self-occlusion and return if occluded */
    6. check if the angle is > 90
    i.e. if the dot product of the vectors is <= 0 then it is self-occluded (dot product stored
    into cos - variable declared above)
    ... if it is self-occluded, then return
    /* Ask find_closest_object( ) if a ray fired toward the light */
    /* hits another object. Pass "hitobj" as the "lasthit" */
    /* parameter so it won't be hit again at distance 0. */
    7. one line of code here for find_closest_object using
    a. &lasthit for base because it’s firing another ray from the hitpoint toward the light
    hits another object
    b. hitobj for lasthit (we previously had been using NULL for this argument because we
    were just shooting a ray towards an object and that’s it; now we are shooting another
    ray from the object back towards the light to see if it hits anything before it gets
    to the light)
    c. close (declared above) for retdist to hold the dist to the closest object in dir to
    light
    /* If an object is hit and the distance to the hit (close)is */
    /* closer to the hitpoint than the light, return */
    8. If an object is hit (i.e. if (obj != NULL)), and
    the distance to the hit (close) is closer to the hitpoint (dist) (i.e. if (close < dist)
    ... then return
    /* Arriving at this point means the light does illuminate */
    /* object. Ask hitobj->getdiff( ) to return diffuse */
    /* reflectivity */
    9. if the program gets this far, then step 6 was false, and step 8 was false (there wasn’t
    anything in between the light and the hitobj) -- so use getdiffuse on hitobj (put result into
    diffuse, the variable declared above)
```
/* Multiply component-wise the diffuse reflectivity by */
/* the emissivity of the light. */
10. use pixProd to multiply component-wise

/* Scale the resulting diffuse reflectivity by cos/dist */
11. use pix_scale

/* Add scaled value to pixel. */
12. use pix_sum

}