The material object

Each visible object must be associated with at least one material_type which defines the way the surface of the object interacts with light in the scene. At the simplest level, the material definition can be thought of as specifying the color of the object in drgb_t (r, g, b) units.

typedef struct material_type
{
    int     cookie;          /* material_t cookie            */
    char    name[NAME_LEN];  /* light_blue for example      */
    drgb_t  ambient;         /* Reflectivity for materials */
    drgb_t  diffuse;
    double  specular;
} material_t;

There are three components to the light interaction model:

- **ambient** – specifies how the object reflects light that is present in the scene but is not emanating from any particular light source. This is how we will initially illuminate our scenes. The visible color of a pixel will be the ambient reflectivity divided by the distance from the viewpoint to the location in 3D space where the ray hits the object.

- **diffuse** – specifies how the object reflects light that does emanate from specific light sources. This will have no effect until we implement light sources. As with ambient lighting, diffuse lighting simulates the physical process by which a photon is absorbed by the material and a new photon having energy (color) dependent upon the atomic structure of the material is emitted.

- **specular** – specifies the degree to which the object acts like a mirror (incoming light is precisely reflected (instead of being diffused) with the angle of incidence being equal to the angle of reflection).

It is possible to create models that are physically unrealizable. We can define an object that reflects ambient light as red and diffuse light as green! But no physical object exists that operates in such a way.
Material functions (methods)

material_init() -

The material_init() function is responsible for

(1) allocating a material_t structure with malloc() and initializing the cookie element. Unlike the camera, material attributes are all optional. So the memset() function should be used to initialize the entire structure to 0 before initializing the cookie.

(2) reading in the material attributes into the material_t structure.

(3) adding the address of the material_t structure to the mats list of the model_t structure

/**/  
/* Create a new material description */

void material_init(FILE *in, model_t *model, int attrmax)
{
    material_t *mat;

    /* malloc() a material_t structure, use memset() to */
    /* initialize it to 0 and store the MAT_COOKIE */

    /* Load attributes as in camera.c */
    /* Unlike the camera the number of attributes is */
    /* optional. Attributes should be initialized to 0.0 */

    material_load_attributes(in, mat);

    /* Ask list_add to add the material entity to the end */
    /* of the mats list in the model structure. */

}
This function must search the list of materials looking for a material for which `mat->name` matches the color specified in the name parameter:

```c
/**
 * Try to locate a material by name */

/* char *name is the requested material name (e.g. yellow) */
material_t *material_getbyname(model_t *model, char *name)
{
    material_t *mat;

    for each mat in the model->mats list
    {
        assert(mat->cookie == MAT_COOKIE);
        if (mat->name matches name) // use strcmp here
            return(mat)
    }
    return(NULL);
}
material_list_print() -

The material_list_print() function processes the entire material list. It should call material_item_print() to print each item.

/**/
/* Produce a formatted dump of the material list */

void material_list_print(model_t *model, FILE *out)
{
    for each mat in the model->mats list
    {
        assert(mat->cookie == MAT_COOKIE);
        material_print(mat, out);
    }
}

material_print() -

The material_print() function should print a formatted version of the material structure. The format should be consistent with that produced by camera_print(). Similar to camera_print().

void material_print(material_t *mat, FILE *out)
{
}
The *material_getters()*

These functions simply copy reflectivity of the *material_t* structure that is passed in to the *dest* *drgb_t* parameter. You can accomplish this with a single call to the *pix_copy()* function (found in *pixel.h*).

Since the caller of this function must already have a pointer to the *material_t* structure, it may seem like useless overhead to call *material_getambient()* instead of just referencing *mat->ambient[R]* directly.

Nevertheless, there are two good reasons for doing this:

1. In a true O-O language an *external user* of the *material* class will not be permitted to directly access its private data attributes. That is, trying to directly reference *mat->ambient[R]* will cause a compile time error!

2. These functions also provide the basis for what is called *polymorphic* behavior. Polymorphism allows us to provide a "default" behavior that can be replaced with a specialized behavior as required. These are the functions that provide the default behavior.

```c
/* drgb_t is where ambient reflectivity gets filled in */
void material_getambient(object_t *obj, material_t *mat, drgb_t *dest) {
}

/* drgb_t *dtest is where diffuse goes */
void material_getdiffuse(object_t *obj, material_t *mat, drgb_t *dest) {
}

/* drgb_t *dtest is where specular goes */
double material_getspecular(object_t *obj, material_t *mat) {
}
```
Testing the material module

As with the camera module, we will build a test driver to test our material functions in isolation.

/* mattest.c */
#include "ray.h"

int main(int argc, char *argv[])
{
    model_t    mod;
    model_t    *model = &mod;
    material_t *mat;
    object_t obj1;
    object_t *obj = &obj1;
    char       entity[16];
    drgb_t     dpix;
    int        count;
    FILE *infile;
    FILE *outfile;

    assert(argc > 2);
    infile = fopen(argv[1], "r");
    assert(infile != NULL);

    outfile = fopen(argv[2], "w");
    assert(outfile != NULL);

    /* Create a material list */
    model->mats = list_init();

    /* Input should consist only of material definitions */
    count = fscanf(stdin, "%s", entity);

    /* but there can be any number of material defs in the file */
    while (count == 1)
    {
        /* create material_t structure and read attributes */
        material_init(stdin, model, 0);

        /* this test is designed to ensure that list_add */
        /* pointed current to the material just loaded */
        mat = (material_t *)list_get_entity(model->mats);
        assert(mat->cookie == MAT_COOKIE);
        fprintf(stderr, "loaded %s \n", mat->name);
        count = fscanf(stdin, "%s", entity);
    }
/ * Have read them all in .. now try to print them */
material_list_print(model, stderr);

/* See if we can find the first in the list */
mat = material_getbyname(model, "blue");
assert(mat->cookie == MAT_COOKIE);
fprintf(stderr, "found %s \n", mat->name);
material_getambient(obj, mat, &dpix);
pix_print(stderr, "ambient is: ", &dpix);

/* See if we can find the last one */
mat = material_getbyname(model, "yellow");
assert(mat->cookie == MAT_COOKIE);
fprintf(stderr, "found %s \n", mat->name);
material_getambient(obj, mat, &dpix);
pix_print(stderr, "ambient is: ", &dpix);

/* See what happens if we try to find a non-existent element */
mat = material_getbyname(model, "chartreuse");
assert(mat == NULL);

return(0);
}