Ray tracer designed (continued)

Now we are finally ready to build an image. This will be a very crude image because it will support ambient lighting only. Nevertheless this is a significant milestone because the 3-D geometry problem must be addressed.

Overview of the *make_image()* function

The *make_image()* function should live in a separate module named *image.c*

```
void make_image(
FILE *outFP,
model_t *model)
{
     unsigned char *pixmap;

     compute size of output image and malloc() pixmap.

     for y = 0 to  window size in pixels
     {
          for x = 0 to window size in pixels
          {
               make_pixel(model, x, y, pixmap_location);
          }
     }
     write .ppm P6 header to outFP
     write pixmap to outFP
}
```

The *make_pixel* function

This function is responsible for driving the construction of the *(r, g, b)* components of a single pixel. Within the ray tracing process pixel colors are represented as

1.double precision values in the range *[0.0, 1.0]* where
2.0.0 represents black and 1.0 the brightest level of the corresponding color.

However, depending upon input values its possible for the raytracing algorithm *to compute intensities that exceed 1.0.* When this happens this module must *clamp* them back to the allowable range [0.0, 1.0].

```
void make_pixel(
model_t *model,
int      x,                /* Pixel x coord             */
int      y,                /* Pixel y coord             */
unsigned char *pixval)     /* -> to (r, g, b) in pixmap */
{
   double *world = malloc(3 * sizeof(double));
   double *intensity = malloc(3 * sizeof(double));

   map_pix_to_world(x, y, world);
```

initilize intensity to (0.0, 0.0, 0.0)

compute unit vector dir in the direction from the *view_point* to *world;*

```
   ray_trace(model, model->proj->view_point, dir,  intensity,
                            0.0, NULL);
```

clamp each element of intensity to the range [0.0, 1.0]

*set (r, g, b) components of vector pointed to by pixval to 255 * corresponding intensity*

}

The *ray_trace* function

The *ray_trace* function is responsible for tracing a single ray. It should reside in *ray.c*

```
/**/
/* This function traces a single ray and returns the composite
*/
/* intensity of the light it encounters  It is recursive and */
/* so the start of the ray cannot be assumed to be the viewpt
*/
/* Recursion won't be involved until we take on specular light
*/

void ray_trace(
model_t *model,        /* pointer to model container         */
double base[3],        /* location of viewer or previous hit */
double dir[3],         /* unit vector in direction of object */
double intensity[3],   /* intensity return location          */
double total_dist,     /* distance ray has traveled so far   */
obj_t  *last_hit)      /* obj that reflected this ray or NULL*/
{
```

The ray trace function should rely upon *find_closest_object* to identify the nearest object that is hit by the ray. If none of the objects in the scene is hit, NULL is returned.

```
   closest = find_closest_obj(model->scene,
                              base, dir, NULL);

   if (closest == NULL)
      return;
```

add distance from base to closest object to total_dist
set intensity to the ambient reflectivity of closest
divide intensity by total_dist

}

Note: the "distance" to the closest object can be found in the "distance" field of the sceneobj_t structure associated with the closest object. The functions sphere_hits() and plane_hits() will set this field.

The *find_closest_obj* function

The prototype for the find_clostest_obj() function is:

```
obj_t *find_closest_obj(
   list_t *scene,      /* head-pointer of scene objects list */
   double base[],      /* starting coordinates of ray        */
   double unitDir[],   /* Direction of ray                   */
   void *lasthit,      /* Object last hit by ray, or NULL    */
);
```

The find_closest_obj() function processes each object in the scene list and uses the "hits" functions described in the following sections to determine if a ray that starts at `base` and goes in the direction `unitDir` hits the object. If one or more objects is hit by the ray the function determines which object is the closest to the base and returns a pointer to that object. If no object is hit, the function returns NULL.

For the initial version of the raytracer the lasthit parameter will always be NULL. We will use this parameter in later versions of the raytracer when a ray bounces from one object to another.