Computer Science 102
Lab 5

Objectives: (1) Understand how to statically define a model so that parallel development of project components can take place. **(2)** Design and implement a function that will identify the first (closest) object hit by a ray.

In real world software development it is common for teams of programmers to work together on a large system.   In our raytracing example one (unlucky) team might work on the parsing problem while another luckier team worked on the the mechanics of the raytracing.

In that situation it is desirable for both teams to be able to proceed in parallel and not have the raytracer team wait for the parser team to finish. To accomplish this objective it is often necessary to develop "throw away" components that allow development and testing on the ray tracer to proceed even though the parser is not ready.

The model.c that you will use this week is an example of such a component. It contains a complete, statically initialized, model structure so that raytracing could occur without a single _init function being available.

Structured data types may be initialized by giving the name of structure elements followed by the value the element is to take on. When the value is an array or sub-structure then the elements must be enclosed in { } ,

```
#include "ray.h"

material_t mat1 =
{
   cookie: MAT_COOKIE,
   name:  "green",
   ambient: {0, 5, 0},
};


plane_t plane1 =
{
   normal: {3, 0, 1},
   point:  {0, 0, 0},
};
```

One structure can be made to point to another as shown below, but the usual rule applies that a name must be defined before it can be referenced. Hence the object definition must follow the plane and material definitions.

```
object_t object1 =
{
   cookie:   OBJ_COOKIE,
   objname:  "leftwall",
   hits:     plane_hits,
   priv:     (void *)&plane1,
   mat:      &mat1,
};
```

Lists can be statically created as well. However, *extreme* care must be taken in setting the link pointers and the list header to ensure that the structure is correct and consistent.

What is the order of the objects in this list??

```
link_t link1 =
{
   next: NULL,
   item: (void *)&object2,
};

link_t link2 =
{
   next: &link1,
   item: (void *)&object1,
};

link_t link3 =
{
   next: &link2,
   item: (void *)&object3,
};


list_t list1 =
{
   head: &link3,
   tail: &link1,
};
```

For todays project we don't need a camera and we don't need to process the material list, so we put only the object list in the model structure.

```
model_t model =
{
   objs: &list1,
};

model_t *model_init(
FILE *in)
{
   return(&model);
};
```

In this lab you will implement the *find_closest_object()* function. It should live in *ray.c*

Components that are provided for you include: *main.c, ray.h, rayfuns.h, rayhdrs.h, model.c* and a *makefile.* You must provide your own *vector.h,* and *plane.c.* You do not need camera.c or material.c. You MUST NOT supply your own model.c For this lab to work you must a have a functional *plane_hits()* function. If yours is not yet working, see your TA by e-mail or in person *before lab* so that it *will be working* by lab time.

The *find_closest_object()* function

This function must process the entire object list, calling

```
     dist = obj->hits(base, dir, obj);
```

for every object in the list.

If no object is hit by the ray the function should set *retdist to -1 and return NULL. If one or more objects is hit by the ray the function should return a pointer to the object with the smallest hit distance and set *retdist to the distance from the base of the ray to the hit point.

```
object_t *find_closest_object(
model_t  *model,       /* Model structure              */
vec_t    *base,        /* Base of ray (viewpoint)      */
vec_t    *dir,         /* unit direction of ray        */
object_t *last_hit,    /* object last hit (ignore)     */
double   *retdist)     /* return dist to hit point here */
{
   object_t *minobj  = NULL;
   double    mindist = -1.0;
    :
    :
    :

   return(minobj);
}
```

Algorithm:

```
   For each object in the object list
      if the ray hits the object at a smaller distance than any
          previous hit

          remember the object
          remember the distance

   When done, store mindist in *ret_dist and return minobj;
```

Remember the nesting rules!! *A maximum nesting of a single if within a single loop is permitted. The if may have a compound condition though.*

In this lab you will submit a compressed tar file named closest.tar.gz containing all the components needed to build your program.

```
     sendlab.102.labsection#  lab#  closest.tar.gz
```

Since this is lab5 and if you are in section 1 the command you should use is (remember to cd .. because that is where you put your tarfile!)

```
     sendlab.102.1 5 closest.tar.gz
```