- Obj-c:
  - vec_t ✓
  - list_t ✓
  - material_t ✓
  - pixel_t ✓
  - object_t ✓
  - plane_t ✓

even
  - timer_t ✓

and
  - NSImageP, NSData ✓

- now, need to redo
  - model_t
  - hits (protocol)
  - find_closest

(lab 11, which is repeat of lab 05 but in Obj-C)

- lab 11 ⟶

- model_t class:
- as before, this object now holds (stores)

```
list_t *mats;
list_t *objs;
```

as internal data members (declared in @interface { })

- member functions:

```
// constructor
- (id) init;

// accessor
- (list_t *) objs;
- (list_t *) mats;

// i/o
- (void) read: (FILE *) _in;
- (void) write: (FILE *) _out;
```

- init method just allocs & inits the list objects (and returns self)
- accessors just return the list objects
- read function does what main() did before:
  - reads token string
  - gets material or object to read itself in
  - adds to list

- recall:

```
if(!strcmp(token,"plane")){
    obj=[(plane_t *)
        [plane_t alloc]init];
    str=[[NSMutableString alloc]
        initWithUTF8String:token];
    [obj setType:str];
    [obj read:_in:mats];
    [objs add:obj];
    [str release];
}
```

- write is even easier:

```
for([mats reset];
    ![mats end]; [mats next])
  [[mats data] write: _out];
```

- same for objs list

- find_closest is almost the same except for the pass-by-reference arguments of hit and N:

- (object_t *) find_closest:
  (vec_t *) pos:
  (vec_t *) dir:
  (double *) dis:
  (vec_t **) hit:
  (vec_t **) N

- inside the function, alloc and init both the "candidate" hit, N and the closest hit and N:

```
vec_t *c_hit = [(vec_t *)
        [vec_t alloc] init];
```

same for

```
vec_t *c_N
vec_t * closest_hit
vec_t * closest_N
```

- the rest of the logic is pretty much the same, e.g.:

```
for ([objs reset]; ![objs end];
     [objs next]) {
    :
  // check to make sure
  // c_obj conforms to
  // intersecting protocol
  if (![c_obj conformsToProtocol:
        @protocol(intersecting)])
    NSLog(@"warning...");
```

- then call [c_obj hits: ...]

- when setting the pass-by-ref arguments, don't forget to de-reference and autorelease:

```
[*hit autorelease];
*hit = [closest_hit copy];
```

- and then release local vars:

```
[closest_hit release];
```

- in main(), while reading the hits file,

```
vect *pos = [...4.0 : 3.0 : 5.0];
vect_t *dir = [(vect_t *)
  [vect_t alloc] init:
        0.0 : 0.0 : 0.0];
while([dir read: hits_file]){
  [dir write: stderr: "Ray
      direction"];
  [dir autorelease];
  dir = [[dir unit] retain];
```

↓

```
dis = 0.0;
obj = [model find_closest:
pos: dir: &dis: &hit: &N];
if (dis > 0) {
    fprintf(stderr,
"Hit: %-12s", [[obj getName]
                UTF8String]);
fprintf(stderr, "Dist = %8.3f",
                dis);
[hit write: stderr: "Loc = "];
}
```

- ASG 5 is very similar to lab 11
- in fact it "predates" lab 11 since it does not get use the model_t object
- ASG 5 just uses a test_hits function to call the hits function:

```
void test_hits(object *obj,
    vect *pos, vect *dir)
{
  double dis = 0.0;
  Vect *hit = [(vect*)
    [vect alloc] init];
  vec_t *N = [(vec_t *)
    [vect alloc] init];
[obj set last_hit: 0.0: 0.0: 0.0];
dis = [obj hits: pos: [dir unit]:
        &hit: &N];
```

```
fprintf(stderr, "dis to plane
    %s %8.3lf \n",
    [[obj getname] UTF8String],
      dis);
[hit write: stderr: "hit point"];
}
```

- Complete ASG 5 first
- ASG 6 is basically a redo of lab 5 and ASG 2
- introduces model_t object and camera_t object:

```
@interface camera_t: NSObject
{
  int cookie;
  NSString *name;
  int    pixel_dim[2];
  double pixel_dim[2];
  vec_t *view_point;
}
```

- camelact object needs these methods:

```
- (id) init;
- (void) read: (FILE *) _file;
- (void) write: (FILE *) _file;
```

- ASG7: a redo of ASG3 (the basic ray caster)
- the ray tracing & image writing loops are now split:

```
for(y=h-1; y>=0; y--){
  for(x=0; x<w; x++){
    wx = (double)x/(double)(w-1)*ww;
    wy = (double)y/(double)(h-1)*wh;
    [pix set: wx : wy : 1 z];
    dir = [pix minus: pos];
    dir = [[dir unit] retain];
```

```
Color = ray_trace(model,
        por, dir, 0.0);
imgloc = img + y*w + x;
for(i=0; i<3; i++)
  (*imgloc)[i] =
      [color get: i];
  }
}
```

- stop the timer after these loops

- now write the image to file:

```
fprintf(stdout, "P6 %d %d 255\n",
        w, h);
for(y=h-1; y>=0; y--) {
    for(x=0; x<w; x++) {
        fwrite(imgloc,
            sizeof(irgb_t), 1, stdout);
    }
}
```

- or dig using NSData

- the ray_trace function needs a few local variables:

```
pixel_t *color = nil;
```

what gets returned

```
double dis = 0.0;
Vec_t *hit = [... init: 0: 0: 0];
Vec_t *N = [... init: 0: 0: 0];
object_t *obj = nil;
pixel_t *ambient=nil,
        *diffuse=nil,
        *specular=nil;
```

- then basically as before:

```
if(!(obj = [model find_closest:
        pos: dir: &dis: &hit: &N]))
    return;
if(dis > 0) {
    ray dist += dis;
    ambient = [[obj mat] getAmb];
    diffuse = [[obj mat] getDiff];
    specular = [[obj mat] getSpec];
```

```
[color autorelease];
color = [ambient copy];
[color autorelease];
color = [color scale: 1.0/rayList];
} // ij statement
return [color scale: 255.0];
} // end of ray_trace
```

- this will change in A5 & A6
- need to return color $\in [0, 1]$