Last time:

plane_hits() [sphere_hits()]

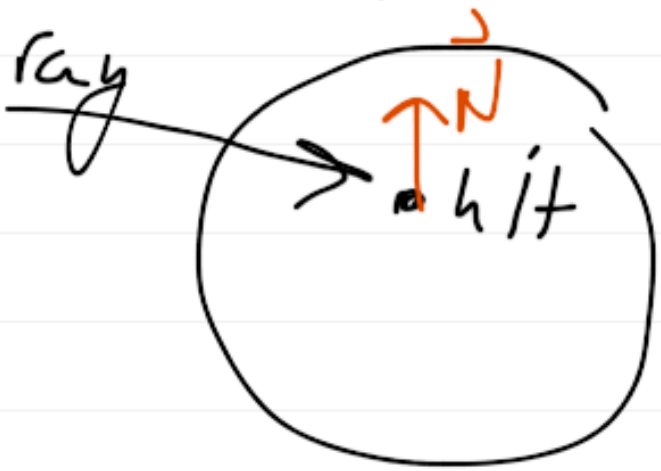
obj->hits() (calling function)

obj->hits() in turn called by object_find_closest()

```
for each obj or objs list:
    if(obj == last_hit ||
       dist = obj->hits() < 0)
        continue;
    if(closest == NULL ||
       dist < min_dist) {
        min_dist = dist;
        closest = obj.
    }
return (closest);
```

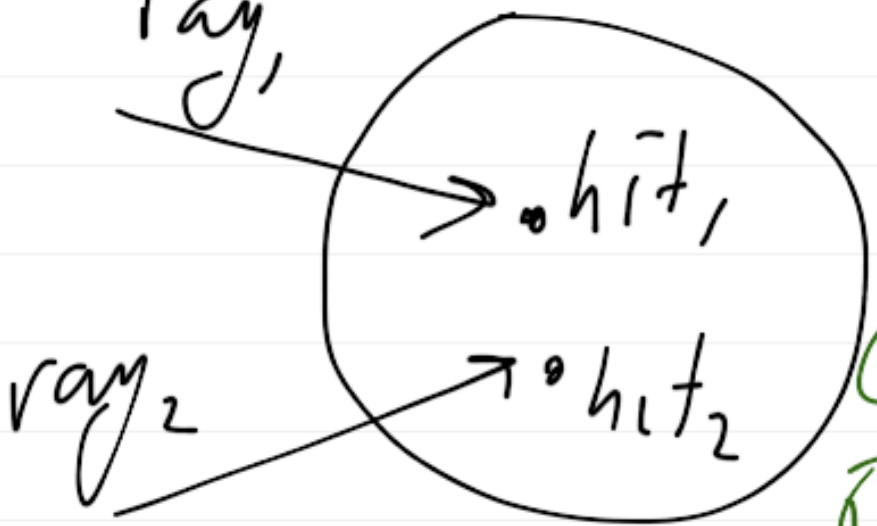
- What is this last_hit ??
- Why is mindist initialized to -1?
- Why is object_find_closest() part of object? should be part of model

- last_hit (& last_normal) were part of object because we wanted each object to keep track of last hit point



- idea was:
  1. ray hits obj
  2. obj is queried for hit & normal

- this fails when ray tracer runs in parallel



- can happen that we get:
  1. obj → hits()
  2. obj → hits()
  3. obj → lasthit

ray1 "sees" hit2 — ERROR [RACE CONDITION]

- need to rewrite find_closest so that it sets hit & N vectors

---

- lab 5 ok with old function
- ASG3 should use new one (ok if use old one)

- BENEFITS:

- allows ray to store hit & N (eventually) & allows many rays [PARALLEL PROCESSING!!]

- alg. more elegant & intuitive

- new function signature:

```
object_t * object_find_closest(
    list_t *objs,
    vec_t pos, vec_t dir,
    double *dis,
    vec_t hit, vec_t n)
```

- two sets of local variables:

```
double c_dis;
Vec_t c_hit, c_N;
Object_t *c_obj = NULL;
```

"candidate" object & related data — used when iterating through list ✓

```
double closest_dis = INFINITY;
Vec_t closest_hit, closest_N;
Object_t *closest_obj = NULL;
```

"current" closest object & data

- start alg as before:

loop thru objects:

```
for(list_reset(objs);
    list_not_end(objs);
    list_next_link(objs)) {
  //get candidate obj
  c_obj = (object_t *)
          list_get_data(objs);
  //get intersection
  if((c_dis = c_obj->hits(
        c_obj, pos, dir, c_hit
        c_N)) < 0)
    continue;
```

```
else if ((0.000001 < c_dis)
    && (c_dis < closest_dis))
{
    closest_dis = c_dis;
    closest_obj = c_obj;
    VEC_COPY(c_hit, closest_hit);
    VEC_COPY(c_N, closest_N);
}
else
    continue;
} //end for
```

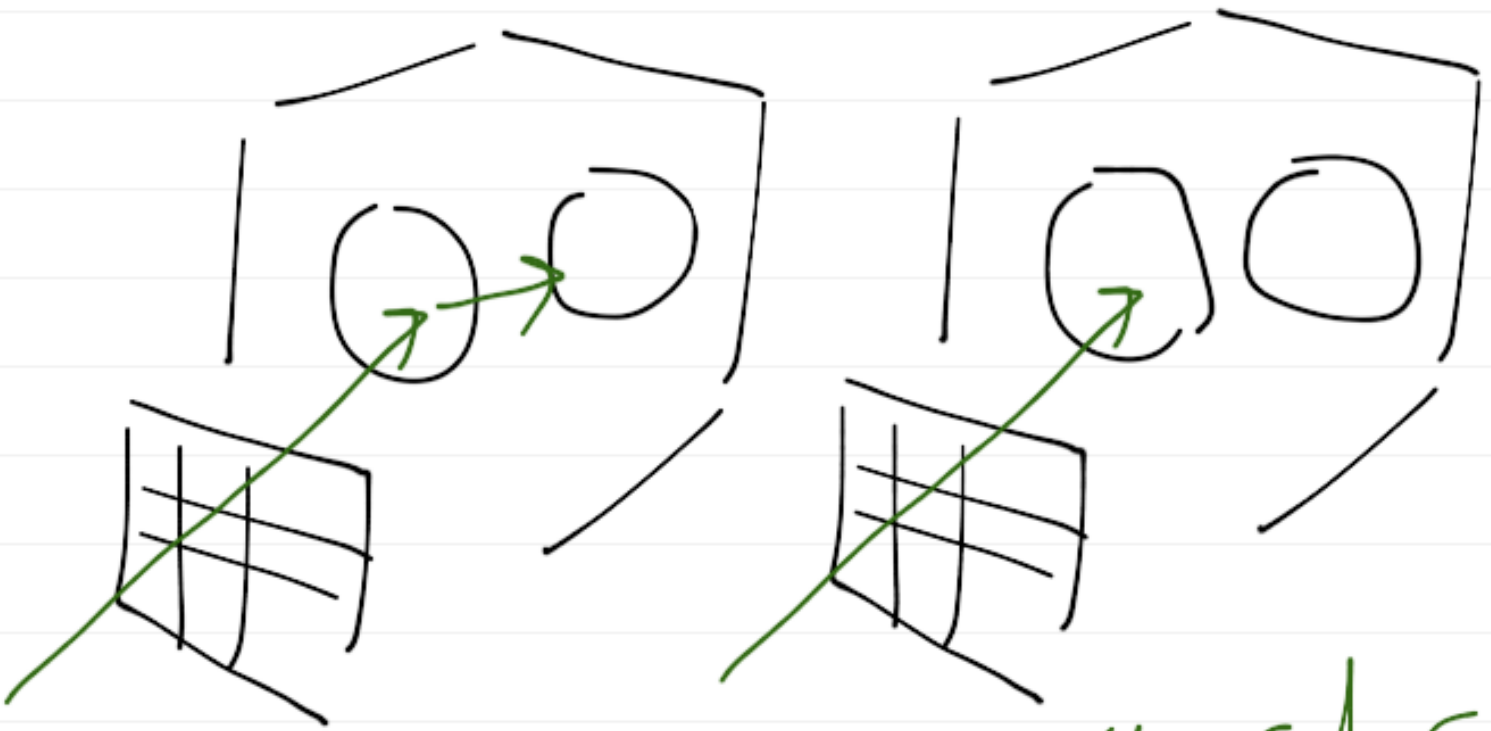
```
if (closest_obj) {
    vec_copy(closest_hit, hit)
    vec_copy(closest_n, n)
  (+dir) += closest_dir
}
return (closest_obj);
}
```

- Need to rewrite hits()
  functions (plane, sphere)
- Need to rewrite calling function
(one calling object_find_closest)

On to basic ray tracer (ray caster)

ray spawned at surface intersection

ray spawned as well, but only cast once



rays bounce (reflect) — RECURSIVE

return color at surface if intersection NO RECURSION

A56.3: all code should be there, just need to write "driver" code in main.c

main:

1. open & parse (read) model.txt as input (via argv[1])
   (don't forget to close file)

2. print model to stderr

3. output PPM image (stdout)

fprintf(stdout, "P6 %d %d 255\n",
w, h);

decimals, or int's
not doubles

image
header
(magic bytes)

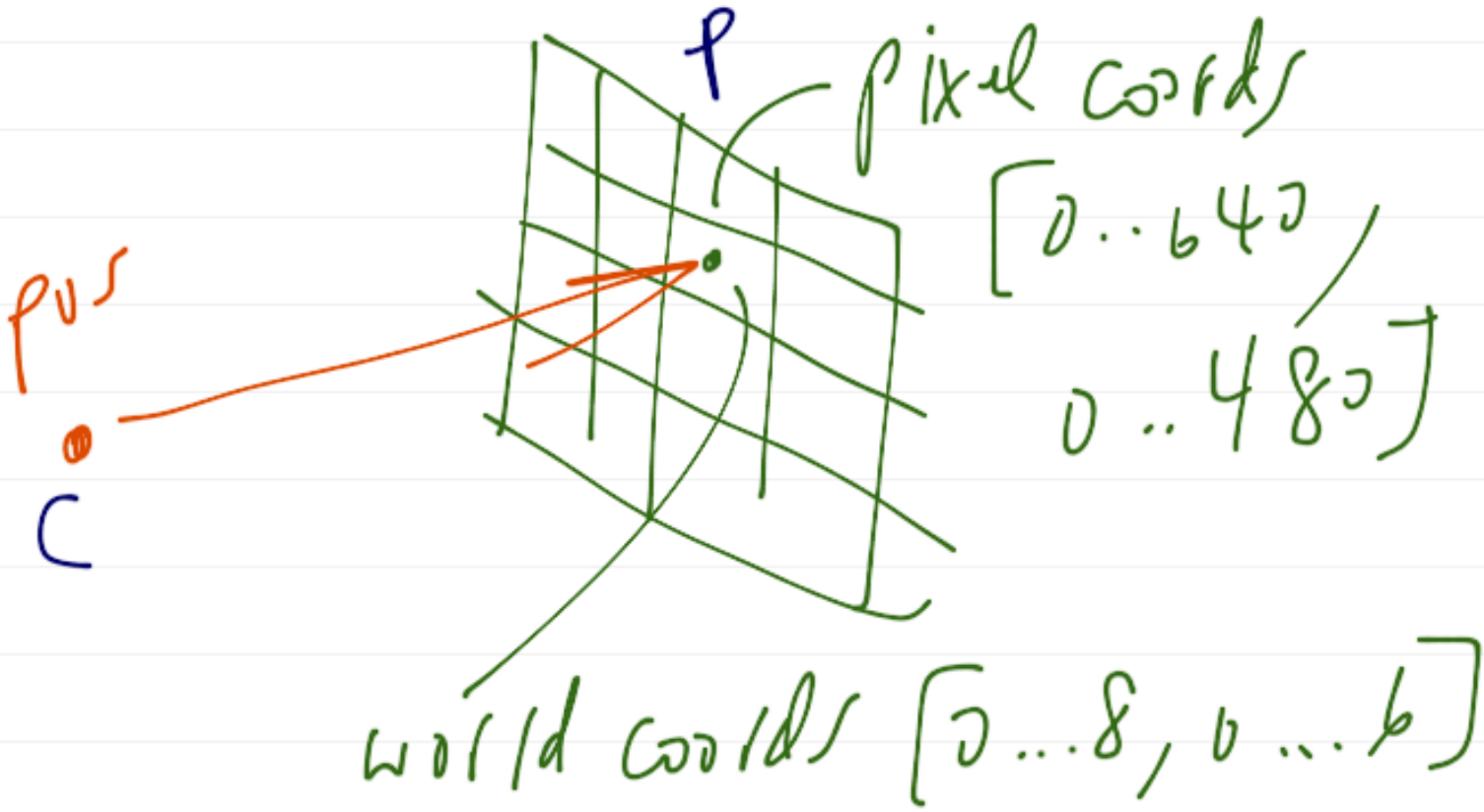
int w =
model → cam →
pixel_dim[0];

int h = model → cam → pixel_dim[1];

```
int x, y; // pixel coords
int ww = cam->world_dim[0]
int wh =        "          " [1]

for(y=0; y<h; y++){
  for(x=0; x<w; x++){
    wx = (double)x /
         (double)(w-1) * ww;
    wy = ...
```

conversion of 640×480 image coords to world coords



C: camera pos, also ray pos (orig)

P: pixel coord

dir = P - C

vec_t vec_t vec_t

} all in world coords

↳ normalize!!

vec_diff(pos, pix, dir)

c p

vec_unit(dir, dir) // normalize dir

now we have pos, dir } ray

these get sent to

ray_trace( )

ray_trace() returns color,
a drgb_t pixel $\in [0, 1]$

pix_scale(255.0, color, color)
(upon return)

// now write color to
image ← an array of
irgb_t type

→

```
irgb_t  color;
irgb_t *imgloc;
irgb_t *img = NULL;
```

the image matrix, as a 1D array

a pointer to image pixel

```
img = (rgb_t *) malloc(
        w*h*sizeof(rgb_t));

memset(img,
    w*h*sizeof(rgb_t));
```

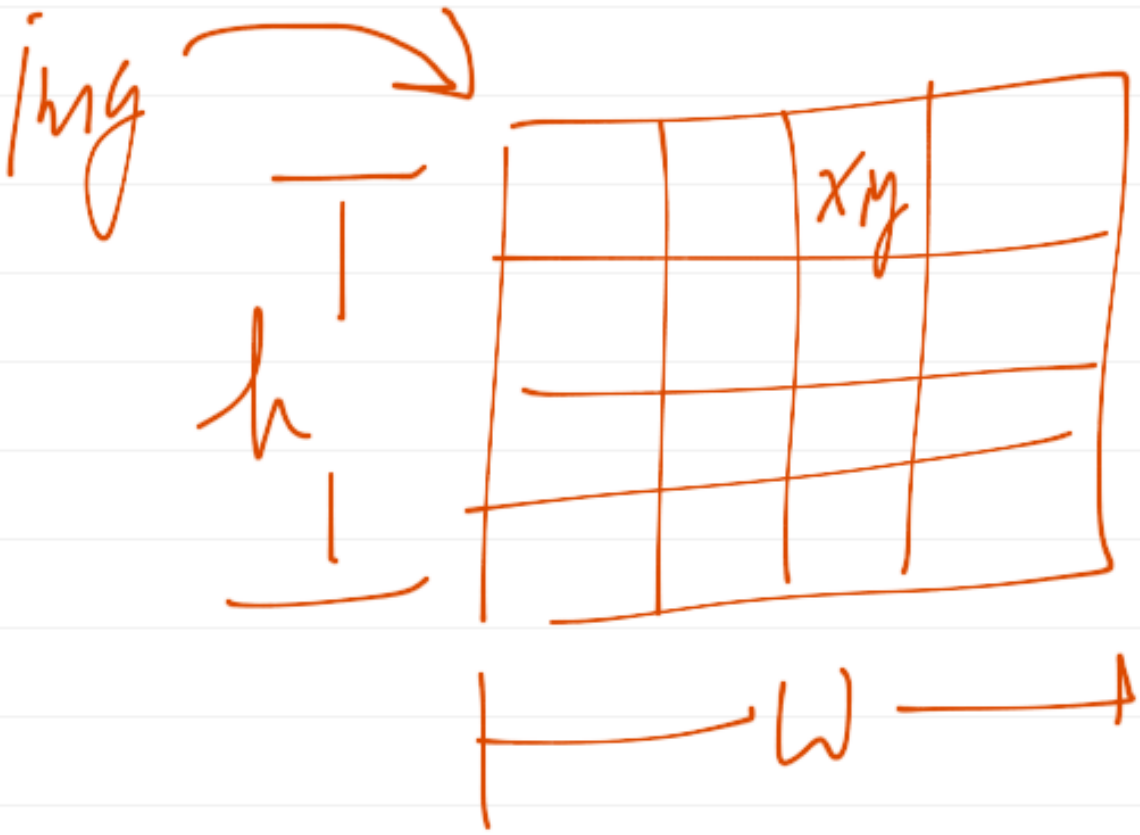
(do this before the doubly-nested for-loop)

now, at bottom of for loop
after ray-trace &
after pix_scale

1D index to 2D

```
imgloc = img + y*w + x;
for(i=0; i<3; i++)
  (*imgloc)[i] = color[i]
```

conversion from
double to unsigned char



img + y * W + x

how many rows to skip

which column

finally, one more conversion
(not strictly necessary)
& write to file

```
for(i=0; i<3; i++)
    icolor[i] = color[i];
fwrite(icolor, sizeof(irgb_t),
       1, stdout);
```

should cast here

---

that's the main loop.

now, ray_trace(model_t *model,
vec_t pos, vec_t dir,
dbgb_t color,
double ray_dist)

what gets returned

for recursion (later)

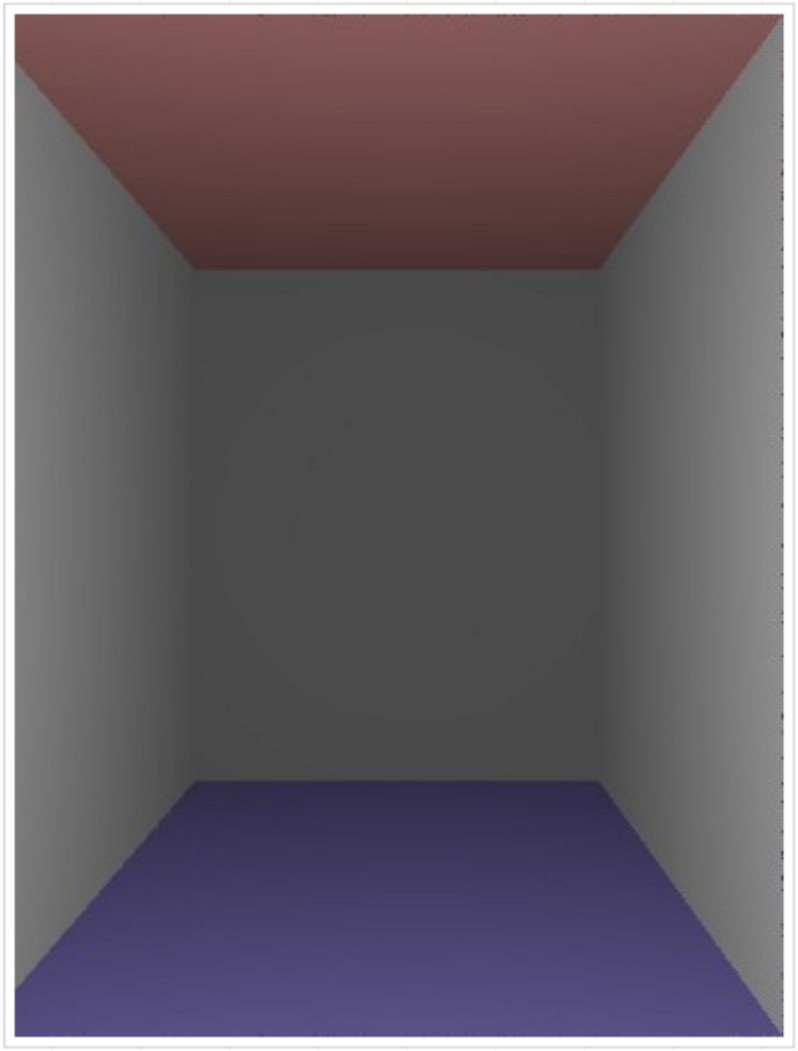
```
local vars;
    double dis = 0.0;
object_t *obj = NULL;
vec_t hit, N;
rgb_t thiscolor =
        {0.0, 0.0, 0.0};
rgb_t ambient,
      diffuse,
      specular;
```

```
if (!obj = object_find_
    closest(model->objs,
        pos, dir, &dis,
        hit, N))
    return NULL;
```

```
if (dis > 0) {
    raydist += dis;
    material_getamb(
            obj->mat, ambient)
    material_getdiff(
            obj->mat, diffuse)
    material_getspec(
            obj->mat, specular)
```

```
        pix_copy(ambient,
                 thiscolor)
        pix_scale(1.0/raydist,
                  thiscolor, thiscolor)
        pix_sum(thiscolor,
                color, color)
    }
}
```

cbox.txt



model.txt