

main.cpp :

```

#include <iostream>
#include <fstream> — for C++ file I/O
    {
        provider
        std::ifstream : like cin
        std::ofstream : like cout
        (file printf)
    }
int main(int argc,
char *argv[])
{
    std::ifstream model_ifs; (model input
    std::ofstream hits_ofs; (hits file)
                                file stream)

// assume model file name is " argv[1]
" " hits " " " argv[2]
    { was just a sequence of raw directions
      (vec-t's)
}

// check for argc == 3
model_ifs.open(argv[1], std::ifstream::in);
// should check for 'good' status
model_ifs >> model; // model reads itself
    { ifstream operator >> (model_t & model)
      ↑
      your model_t object
      has to have this operator
      overloaded
}
model_ifs.close(); // don't forget to close file

// debug

```

```
std::cout << model; // get model to print itself
```

```
// same for hits file
```

```
// open hits file, then loop for each dir vector found therein
```

```
while (hits_ifs >> dir) {
```

```
    ; // a vector  
    ; // ifstream hits_ifs  
    ; // return found (return s)  
    ; // vector
```

vector pointer that returns vector

find_closest can alter contents

```
    obj = model.find_closest(pos, dir.norm(), (obj_t *)NULL, dist);
```

```
    if (dist > 0) {
```

```
        // should probably check for obj != NULL
```

```
        std::cerr << "Loc = " << obj -> get_last_hit() << std::endl;
```

has to return vector that find_closest would have set

double <

```
    } // read write
```

```
    hits_ifs.close();
```

```
} // main
```

material_t
also has lights list, camera

- model_t :

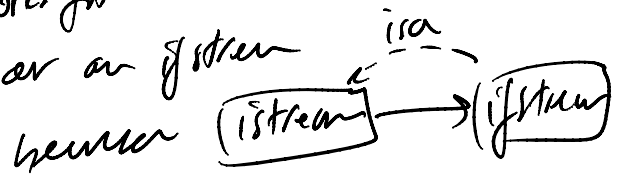
- private data members: mats, obj's lists

- std::istream & operator >> (std::istream & s, model_t & (hs))
{
 std::string

 :
 while (!s.eof()) {

works for cin

or an ifstream



 member (istream) >> model;

 if (token == "material") {

 s >> (mat = new material_t());

 ↑
 pointer to material_t

 ↑
 allocates mem for mat

 ↑
 material_t constructor

 // add mat to mats list

 }

 if (token == "plane") {

 s >> (obj_i = new plane_t(token));

 ↑
 obj_t *
 // add obj_i to obj's list

 ↑
 plane_t constructor

 if (token == "sphere") {

 ↑
 result of the

is a plane + #

}

if (folken == "cannon") {

}

if (folken == "light") {

}

} // while
return s,
}

return obj (ptr to object_t)

object_t * model_t :: find_closest (vec_t & pos,
vec_t & dir,
object_t * (const list_t,
double & ret_dlist)

{

dlist.reset();

// basic list iteration

while (dlist.not_end()) {

obj_i = (object_t * & dlist.get_dart(i);

;

if ((dist = obj_i -> hits(pos, dir)) < 0)

;

↑
invoke polymorphic hits fn

...

return(closest);

plane_t would have one
sphere_t " " " "

}

... and one

3

space

direct-t now have virtual one