Qt questions?

ASG 6: photon mapper

Idea:

1. shoot photons (ASG4)
   → emit photons from light source(s)
2. trace ~~shoot~~ rays
   → each ray accumulates light "power" from k-nn photons

Step 1:

shoot photons "points"

— result is list of points in 3D each with power & direction, scattered on surfaces in 3D world

3 light sources — model.txt

airball.txt

1 light source

MAKE YOUR OWN

Step 1a:
loop thru list of n photons, send power by $1/n$ (n will be smaller than what you started out with, i.e. 500,000)

Step 1b:

loop thru n photons

find min, max (x,y,z) pos

bounding volume for kd-tree

Step 1c:

Insert photons into kd-tree:

```
kdtree_t<photon_t,
         photon_t*,
         photon_c>
         kdtree;
```

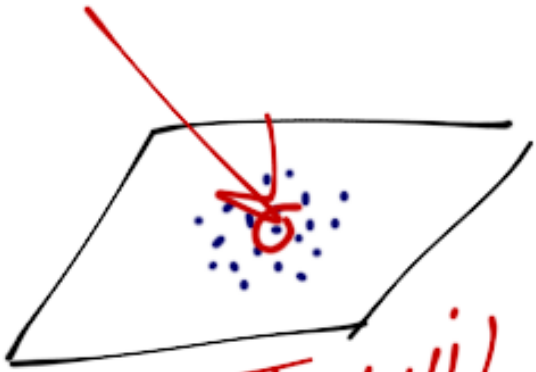
comparator functor

```
kdtree.insert(photons.
              min, max)
```

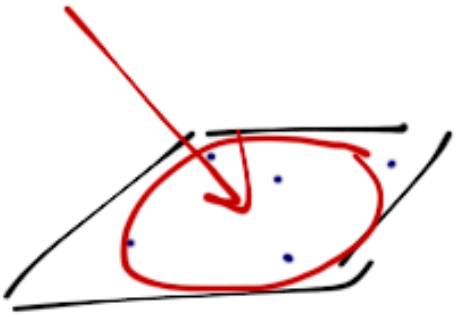
Step 2 : ray trace

— each ray will query for K nearest photons (K = 20)

— estimate power density (flux) at ray intersection (surf. point)



5-nearest will be within small r (1/r is large)



5-nearest will be within large r (1/r is small)

- Go (in main.cpp)

use

ray → trace(model, Kdtree, color, ∅)

instead of

ray → trace(model, color, ∅)

overloaded trace() fcn: just copy old fcn, add in argument

need to include Kdtree into your #pragma omp statement (shared or private)

```
ray_t::trace(...)
{
  // pretty much same
     as before but now
     add "contribution"
     from photon power flux
  .
  :
  obj = model.find_closest(...)
  if (dis > 0) {
```

```
if (dis > 0) {
    // get material properties
    // ambient color
    color += 1.0 / dis * ambient
    color.clamp(...)
    if (!diffuse.isZero()) {
        color += (I_d * diffuse)
        color.clamp()
    }
```

```
if (!specular.isZero)
    // shoot reflection ray
    color = (diffuse * color) +
        (specular * ref color)
    color.clamp(...)
if (alpha > 0)
    // shoot refraction ray
color = (1 - alpha) * color +
alpha * ((1-k) * tcol + k * rcol))
```

// specular highlights

// photon power

(see Henrik Wann Jensen)

---

a) kun gjeng ($K = 20$), gut $\underline{\underline{r}}$

b) for each of k nn photons

surf normal

if(N · - photon.dir > 0)

d_p = distance from
hit point to photon

(use photon_t :: distance)

$$w_{pc} = \frac{1 - \frac{d_p}{kr}}{1 - \frac{2}{3k}}$$

cone filter

→ See A56 web page

flux += photon.pwr * upc

//end of for loop

c) color += flux * $\left(\frac{1}{\pi r^2}\right)$

color.clamp(...)