Optimizing smallpt

Daven Scies, Siddharth Bhat

Haskell Exchange

November 4th, 2020

What is smallpt anyway?

```
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
struct Vec {
  double x, y, z; // position, also color (r,g,b)
  ... methods...
};
struct Ray { Vec o, d; Ray(Vec o_, Vec d_) : o(o_), d(d_) {} };
enum Refl_t { DIFF, SPEC, REFR };  // material types, used in radiance()
struct Sphere {
  double rad;   // radius
  Vec p, e, c;  // position, emission, color
  Refl_t refl;  // reflection type (DIFFuse, SPECular, REFRactive)
  ... methods ...
  double intersect(const Ray &r) const // returns distance, 0 if nohit
};
Sphere spheres[] = {//Scene: radius, position, emission, color, material
  Sphere(1e5, Vec( 1e5+1,40.8,81.6), Vec(),Vec(.75,.25,.25),DIFF),//Left
  ... initialization ...
};
inline bool intersect(const Ray &r, double &t, int &id)
```

What is smallpt anyway?

```
Vec radiance(const Ray &r, int depth, unsigned short *Xi){
  double t;                               // distance to intersection
  int id=0;                               // id of intersected object
  if (!intersect(r, t, id)) return Vec(); // if miss, return black
  const Sphere &obj = spheres[id];        // the hit object
  Vec x=r.o+r.d*t, n=(x-obj.p).norm(), nl=n.dot(r.d)<0?n:n*-1, f=obj.c;
  double p = f.x>f.y && f.x>f.z ? f.x : f.y>f.z ? f.y : f.z; // max refl
  if (++depth>5) if (erand48(Xi)<p) f=f*(1/p); else return obj.e; //R.R.
  if (obj.refl == DIFF){                  // Ideal DIFFUSE reflection
    double r1=2*M_PI*erand48(Xi), r2=erand48(Xi), r2s=sqrt(r2);
    Vec w=nl, u=((fabs(w.x)>.1?Vec(0,1):Vec(1))%w).norm(), v=w%u;
    Vec d = (u*cos(r1)*r2s + v*sin(r1)*r2s + w*sqrt(1-r2)).norm();
    return obj.e + f.mult(radiance(Ray(x,d),depth,Xi));
  } else if (obj.refl == SPEC)            // Ideal SPECULAR reflection
    return obj.e + f.mult(radiance(Ray(x,r.d-n*2*n.dot(r.d)),depth,Xi));
  Ray reflRay(x, r.d-n*2*n.dot(r.d));     // Ideal dielectric REFRACTION
  bool into = n.dot(nl)>0;                // Ray from outside going in?
  double nc=1, nt=1.5, nnt=into?nc/nt:nt/nc, ddn=r.d.dot(nl), cos2t;
  if ((cos2t=1-nnt*nnt*(1-ddn*ddn))<0)    // Total internal reflection
    return obj.e + f.mult(radiance(reflRay,depth,Xi));
  Vec tdir = (r.d*nnt - n*((into?1:-1)*(ddn*nnt+sqrt(cos2t)))).norm();
  double a=nt-nc, b=nt+nc, R0=a*a/(b*b), c = 1-(into?-ddn:tdir.dot(n));
  double Re=R0+(1-R0)*c*c*c*c*c,Tr=1-Re,P=.25+.5*Re,RP=Re/P,TP=Tr/(1-P);
  return obj.e + f.mult(depth>2 ? (erand48(Xi)<P ?   // Russian roulette
    radiance(reflRay,depth,Xi)*RP:radiance(Ray(x,tdir),depth,Xi)*TP) :
    radiance(reflRay,depth,Xi)*Re+radiance(Ray(x,tdir),depth,Xi)*Tr);
}
```

```
int main(int argc, char *argv[]){
  int w=1024, h=768, samps = argc==2 ? atoi(argv[1])/4 : 1; // #
  Ray cam(Vec(50,52,295.6), Vec(0,-0.042612,-1).norm()); // cam
  Vec cx=Vec(w*.5135/h), cy=(cx%cam.d).norm()*.5135, r, *c=new V
#pragma omp parallel for schedule(dynamic, 1) private(r)       /
  for (int y=0; y<h; y++){                       // Loop over ima
    fprintf(stderr,"\rRendering (%d spp) %5.2f%%",samps*4,100.*y
    for (unsigned short x=0, Xi[3]={0,0,y*y*y}; x<w; x++)   // L
      for (int sy=0, i=(h-y-1)*w+x; sy<2; sy++)     // 2x2 subpi
        for (int sx=0; sx<2; sx++, r=Vec()){        // 2x2 subpi
          for (int s=0; s<samps; s++){
            double r1=2*erand48(Xi), dx=r1<1 ? sqrt(r1)-1: 1-sqr
            double r2=2*erand48(Xi), dy=r2<1 ? sqrt(r2)-1: 1-sqr
            Vec d = cx*( ( (sx+.5 + dx)/2 + x)/w - .5) +
                    cy*( ( (sy+.5 + dy)/2 + y)/h - .5) + cam.d;
            r = r + radiance(Ray(cam.o+d*140,d.norm()),0,Xi)*(1.
          } // Camera rays are pushed ^^^^^ forward to start in
          c[i] = c[i] + Vec(clamp(r.x),clamp(r.y),clamp(r.z))*.25
    }
  }
  FILE *f = fopen("image.ppm", "w");          // Write image to PP
  fprintf(f, "P3\n%d %d\n%d\n", w, h, 255);
  for (int i=0; i<w*h; i++)
    fprintf(f,"%d %d %d ", toInt(c[i].x), toInt(c[i].y), toInt(c
}
```