Additional Material for: Efficient Monte Carlo Rendering with Realistic Lenses

Johannes Hanika and Carsten Dachsbacher,
Karlsruhe Institute of Technology, Germany

1 Adaptation of smallpt

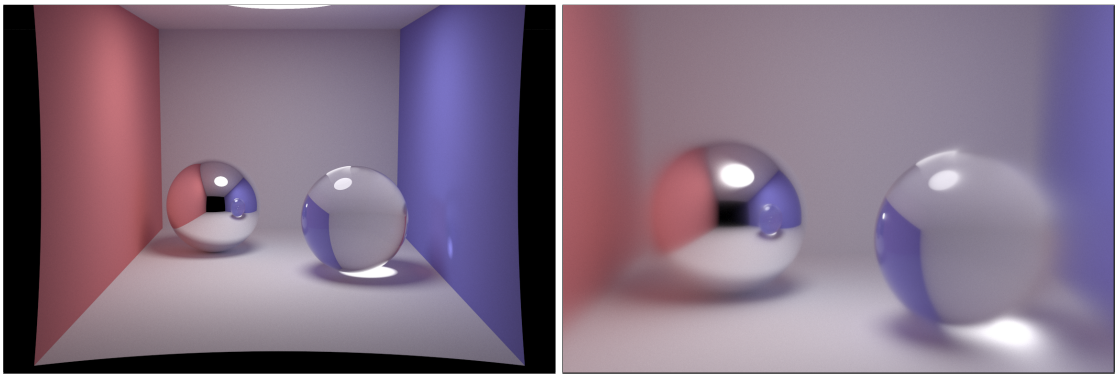

Figure 1: Smallpt without and with lens distortion. The lens system is a simple lensbaby-like lens to clearly show all aberrations. Since smallpt does not support spectral rendering, we used a grey transport version of the polynomials. We need to add 6 lines of code (not counting the preprocessor switches). The curved edges in the left image are due to the fact that the walls are actually modeled as spheres in this renderer.

Listing 1 shows modified smallpt source code with camera distortion. The block between `#if 1` and `#else` applies the transform. Sampling is done through the inner pupil and the transformation function is applied, i.e. no aperture importance sampling is done. The results with and without lens distortions can be seen in Figure 1.

Listing 1: Adapted smallpt code

```
#include <math.h>   // smallpt, a Path Tracer by Kevin Beason, 2008
#include <stdlib.h> // Make : g++ -O3 -fopenmp smallpt.cpp -o smallpt
#include <stdio.h>  //        Remove "-fopenmp" for g++ version < 4.2
struct Vec {        // Usage: time ./smallpt 5000 && xv image.ppm
  double x, y, z;                  // position, also color (r,g,b)
  Vec(double x_=0, double y_=0, double z_=0){ x=x_; y=y_; z=z_; }
  Vec operator+(const Vec &b) const { return Vec(x+b.x,y+b.y,z+b.z); }
  Vec operator-(const Vec &b) const { return Vec(x-b.x,y-b.y,z-b.z); }
  Vec operator*(double b) const { return Vec(x*b,y*b,z*b); }
  Vec mult(const Vec &b) const { return Vec(x*b.x,y*b.y,z*b.z); }
  Vec& norm(){ return *this = *this * (1/sqrt(x*x+y*y+z*z)); }
  double dot(const Vec &b) const { return x*b.x+y*b.y+z*b.z; } // cross:
  Vec operator%(Vec&b){return Vec(y*b.z-z*b.y,z*b.x-x*b.z,x*b.y-y*b.x);}
};
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)
  Sphere(double rad_, Vec p_, Vec e_, Vec c_, Refl_t refl_):
    rad(rad_), p(p_), e(e_), c(c_), refl(refl_) {}
  double intersect(const Ray &r) const { // returns distance, 0 if nohit
    Vec op = p-r.o; // Solve t^2*d.d + 2*t*(o-p).d + (o-p).(o-p)-R^2 = 0
    double t, eps=1e-4, b=op.dot(r.d), det=b*b-op.dot(op)+rad*rad;
    if (det<0) return 0; else det=sqrt(det);
    return (t=b-det)>eps ? t : ((t=b+det)>eps ? t : 0);
  }
};
Sphere spheres[] = {//Scene: radius, position, emission, color, material
  Sphere(1e5, Vec( 1e5+1,40.8,81.6), Vec(),Vec(.75,.25,.25),DIFF),//Left
  Sphere(1e5, Vec(-1e5+99,40.8,81.6),Vec(),Vec(.25,.25,.75),DIFF),//Rght
  Sphere(1e5, Vec(50,40.8, 1e5),     Vec(),Vec(.75,.75,.75),DIFF),//Back
  Sphere(1e5, Vec(50,40.8,-1e5+170), Vec(),Vec(),           DIFF),//Frnt
  Sphere(1e5, Vec(50, 1e5, 81.6),    Vec(),Vec(.75,.75,.75),DIFF),//Botm
  Sphere(1e5, Vec(50,-1e5+81.6,81.6),Vec(),Vec(.75,.75,.75),DIFF),//Top
  Sphere(16.5,Vec(27,16.5,47),       Vec(),Vec(1,1,1)*.999, SPEC),//Mirr
  Sphere(16.5,Vec(73,16.5,78),       Vec(),Vec(1,1,1)*.999, REFR),//Glas
  Sphere(600, Vec(50,681.6-.27,81.6),Vec(12,12,12),  Vec(), DIFF) //Lite
};
inline double clamp(double x){ return x<0 ? 0 : x>1 ? 1 : x; }
inline int toInt(double x){ return int(pow(clamp(x),1/2.2)*255+.5); }
inline bool intersect(const Ray &r, double &t, int &id){
  double n=sizeof(spheres)/sizeof(Sphere), d, inf=t=1e20;
  for(int i=int(n);i--;) if((d=spheres[i].intersect(r))&&d<t){t=d;id=i;}
  return t<inf;
}
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=1152, h=768, samps = argc==2 ? atoi(argv[1])/4 : 1; // # samples
  // Ray cam(Vec(50,52,295.6), Vec(0,-0.042612,-1).norm()); // cam pos, dir
  Ray cam(Vec(50,52,295.6), Vec(0,-0.12,-1).norm()); // cam pos, dir
  Vec cx=Vec(w*.5135/h), cx1=Vec(.5135), cy=(cx%cam.d).norm()*.5135, r, *c=new
       Vec[w*h]; // cx1 is forming a scaled ortho-nomal system with cy, no
      aspect ratio skew in there.
#pragma omp parallel for schedule(dynamic, 1) private(r)       // OpenMP
  for (int py=0; py<h; py++){                          // Loop over image rows
    fprintf(stderr,"\rRendering (%d spp) %5.2f%%",samps*4,100.*py/(h-1));
    for (unsigned short px=0, Xi[3]={0,0,py*py*py}; px<w; px++)   // Loop cols
      for (int sy=0, i=(h-py-1)*w+px; sy<2; sy++)     // 2x2 subpixel rows
        for (int sx=0; sx<2; sx++, r=Vec()){        // 2x2 subpixel cols
          for (int s=0; s<samps; s++){
```

```
            double r1=2*erand48(Xi), ddx=r1<1 ? sqrt(r1)-1: 1-sqrt(2-r1);
            double r2=2*erand48(Xi), ddy=r2<1 ? sqrt(r2)-1: 1-sqrt(2-r2);
#if 1 // simple lens element, gray transport
            float x = 36.0f * ((sx+.5f + px)/w - .5), y = 24.0f * ((sy+.5f +
                py)/h - .5); // 36x24mm film back
            const float dx = ddx * 5.f/30.f - x/30.f, dy = ddy * 5.f/30.f - y
                /30.f; // abuse pixel filter sampling for direction, scale by
                pupil radius / distance to sensor
            x += 0.355f * dx, y += 0.355f * dy; // need -0.355mm sensor offset
                to focus at 210dm world space
            const float out_x = 39.2597 * dx + -15.1427 * dx * dy * dy +
                -15.1427 * dx * dx * dx + -0.178326 * y * dx * dy + 0.00151706
                 * y * y * dx + 0.813123 * x + -0.456 * x * dy * dy +
                -0.634326 * x * dx * dx + -0.0121613 * x * y * dy + -7.55302e
                -05 * x * y * y + -0.0106442 * x * x * dx + -7.55302e-05 * x *
                 x * x;
            const float out_y = 39.2597 * dy + -15.1427 * dy * dy * dy +
                -15.1427 * dx * dx * dy + 0.813123 * y + -0.634326 * y * dy *
                dy + -0.456 * y * dx * dx + -0.0106442 * y * y * dy + -7.55302
                e-05 * y * y * y + -0.178326 * x * dx * dy + -0.0121613 * x *
                y * dx + 0.00151706 * x * x * dy + -7.55302e-05 * x * x * y;
            const float out_dx = 0.0143372 * dx + -1.24884 * dx * dy * dy +
                -1.24884 * dx * dx * dx + -0.0457492 * y * dx * dy +
                -0.000763789 * y * y * dx + -0.0251745 * x + -0.0230086 * x *
                dy * dy + -0.0687577 * x * dx * dx + -0.00105712 * x * y * dy
                + -2.29452e-05 * x * y * y + -0.00182091 * x * x * dx +
                -2.29452e-05 * x * x * x;
            const float out_dy = 0.0143372 * dy + -1.24884 * dy * dy * dy +
                -1.24884 * dx * dx * dy + -0.0251745 * y + -0.0687577 * y * dy
                 * dy + -0.0230086 * y * dx * dx + -0.00182091 * y * y * dy +
                -2.29452e-05 * y * y * y + -0.0457492 * x * dx * dy +
                -0.00105712 * x * y * dx + -0.000763789 * x * x * dy +
                -2.29452e-05 * x * x * y;
            Vec d = cx1*out_dx + cy*out_dy + cam.d, o = cam.o + cx1*(out_x
                /100.0) + cy*(out_y/100.0); // convert to world space from mm
            r = r + radiance(Ray(o + d*155,d.norm()),0,Xi)*(1./samps);
#else
            Vec d = cx*( ( (sx+.5 + ddx)/2 + px)/w - .5) +
                    cy*( ( (sy+.5 + ddy)/2 + py)/h - .5) + cam.d;
            r = r + radiance(Ray(cam.o+d*155,d.norm()),0,Xi)*(1./samps);
#endif
          } // Camera rays are pushed ^^^^^ forward to start in interior
          c[i] = c[i] + Vec(clamp(r.x),clamp(r.y),clamp(r.z))*.25;
        }
  }
  FILE *f = fopen("image.ppm", "w");          // Write image to PPM file.
  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[i].z));
}
```

2 Example Lens Polynomials

The first-order polynomial $P(x) = y$ of a simple bi-convex lens with focal distance $f = 35mm$ and sensor distance of $d = 40mm$ is

$$x' = 45.1547 \cdot u + 0.783978 \cdot x \tag{1}$$
$$y' = 45.1547 \cdot v + 0.783978 \cdot y \tag{2}$$
$$u' = -0.502663 \cdot u - 0.0307972 \cdot x \tag{3}$$
$$v' = -0.502663 \cdot v - 0.0307972 \cdot y \tag{4}$$
$$\lambda' = \lambda \tag{5}$$

The terms for a thin lens model would be

$$x' = d \cdot u + x \tag{6}$$
$$y' = d \cdot v + y \tag{7}$$
$$u' = d/o \cdot u + (d + o)/(d \cdot o) \cdot x \tag{8}$$
$$v' = d/o \cdot v + (d + o)/(d \cdot o) \cdot y \tag{9}$$

where $o = 1/(1/f - 1/d)$ is the object distance. In the given example this equates to $o = 280.0$ and

$$x' = 40.0 \cdot u + x \tag{10}$$
$$y' = 40.0 \cdot v + y \tag{11}$$
$$u' = 0.1428 \cdot u + 0.0286 \cdot x \tag{12}$$
$$v' = 0.1428 \cdot v + 0.0286 \cdot y. \tag{13}$$

The coefficients of first-order polynomial optics and the thin lens model do not line up exactly, since the parametrisation of the truncated Taylor series is different between the models (plane/plane parametrized light field vs. expansion of $\sin\theta$).

3 Comparisons and f-stops

In the following figures we provide a comparison of brute force path tracing, polynomial optics (Hullin et al., 2012), and our method.

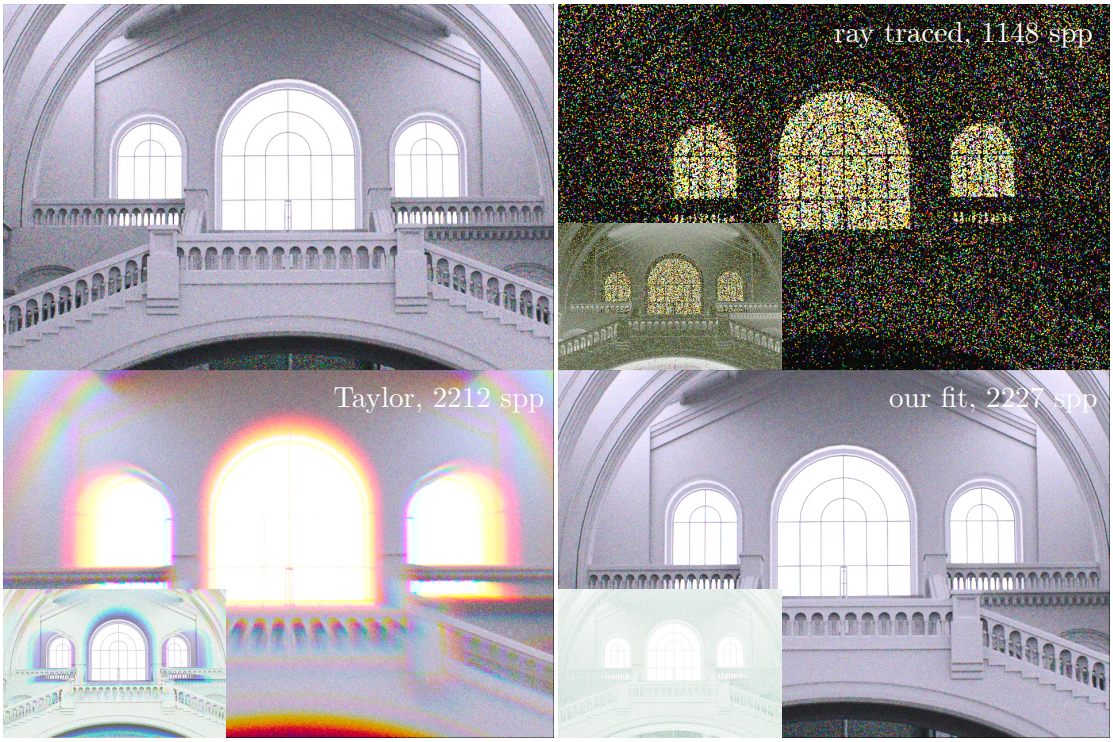

Figure 2: Renders taken through a virtual lens (Canon 70-200mm f/2.8L at f/16), using uniformly sampled path tracing. The first image is the raytraced ground truth reference with 1M samples per pixel. The other three are 20min equal-time comparison renders using the indicated method. The insets are difference images to the reference (inverted for print).

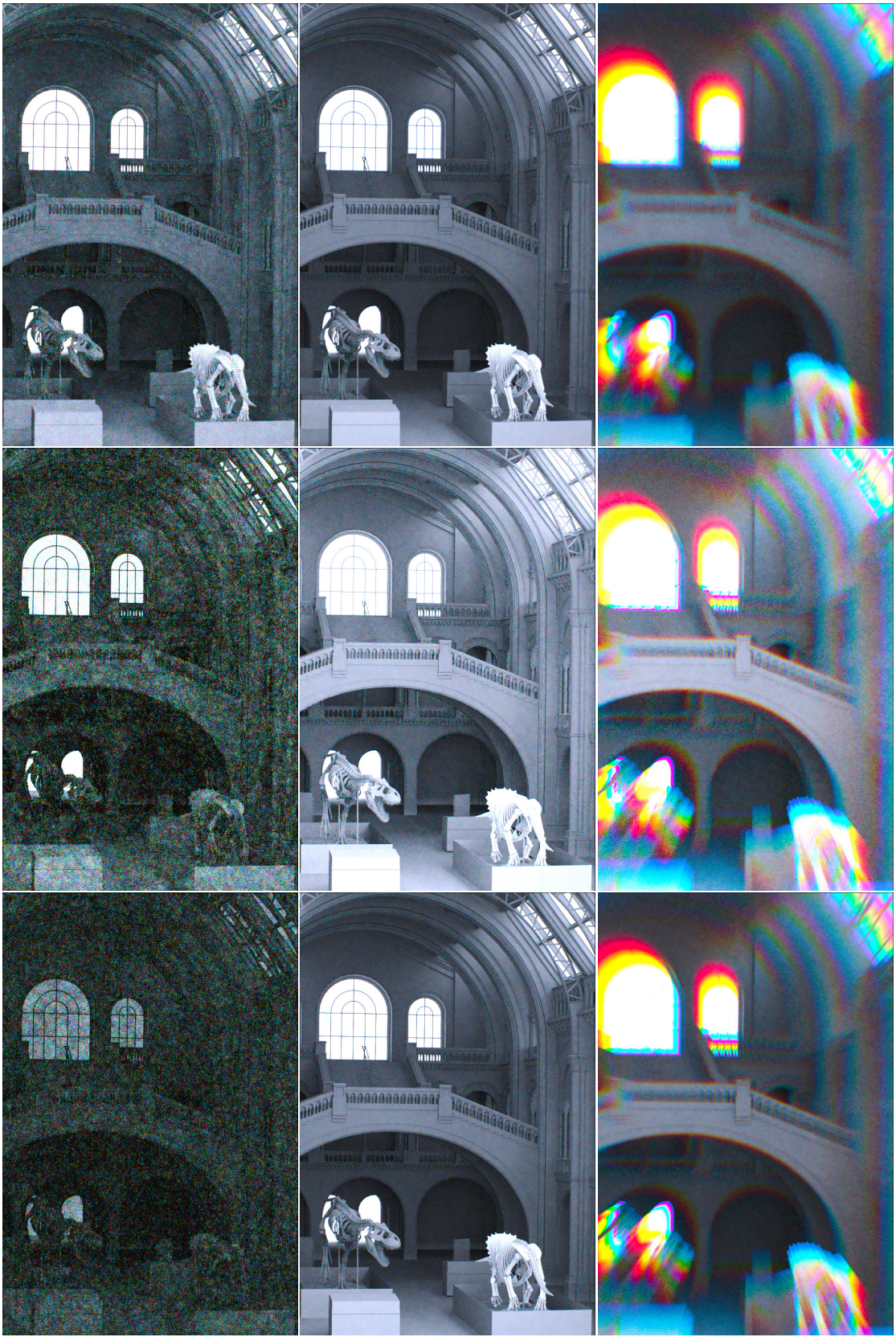

Figure 3: Same time comparisons rendering through a Canon 70-200mm 2.8L (20min). The rendering algorithm was a path tracer with next event estimation and primary sample space Metropolis light transport. From left to right: brute force ray tracing through the lenses, our method, polynomial optics with Taylor expansion. Aperture sizes: top row f/2.8, middle f/8.0, bottom f/16. The renders have (left to right, top to bottom) (1891, 2307, 2427), (1816, 2292, 2393), (1725, 2328, 2411) samples per pixel. Note that the top row is the best case for brute force ray tracing, as the aperture is wide open.

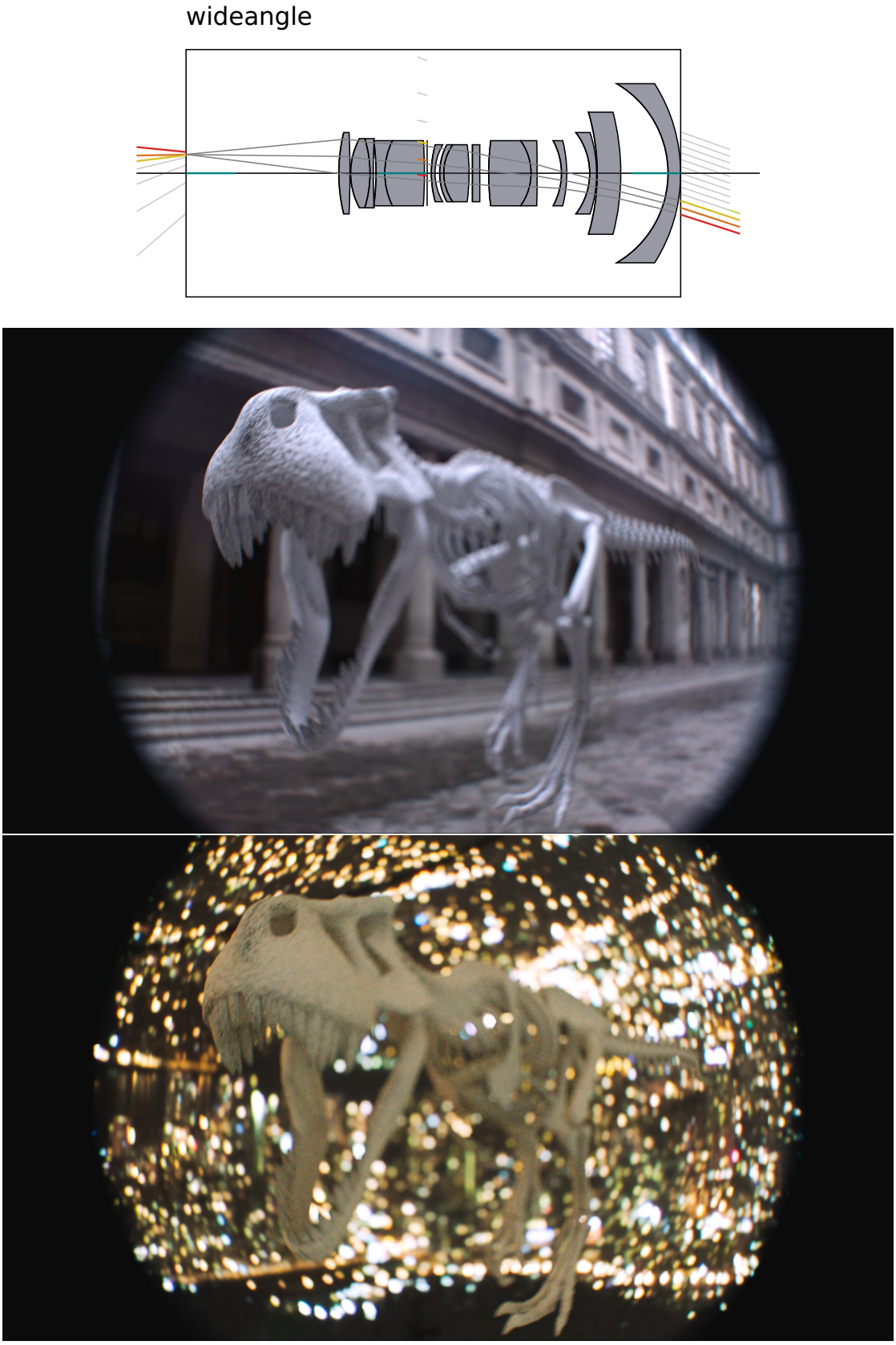

Figure 4: A wide angle lens (US Patent 4,310,222, 14mm f/2.8) at f/8.0, apparently designed for crop sensors. Even though fisheye lenses are not supported, our method handles wide angle lenses reasonably well.

4 A More Complex Example

The following lists code for the degree-4 fit of the Canon Zoom 70-200mm f/2.8L at 70mm suitable for path tracing from the eye using aperture sampling (Listing 2, Figure 8 in the paper) and the corresponding Jacobian (Listing 3, $J_{x \to a}$ in Sec. 4.1 in the paper). To just do simple forward evaluation (path tracing), Listing 4 provides the code. For reverse evaluation (light tracing) an iteration is required (Listing 5, Figure 7 in the paper). All units are in millimeters and the wavelength λ is in micrometers.

Listing 2: Aperture Sampling

```
float pred_x;
float pred_y;
float pred_dx;
float pred_dy;
for(int k=0;k<4;k++)
{
  const float begin_x = x + dist * dx;
  const float begin_y = y + dist * dy;
  const float begin_dx = dx;
  const float begin_dy = dy;
  const float begin_lambda = lambda;
  pred_x = 100.843 * begin_dx + 9.60414 * begin_dx * begin_lambda + -18.8958 *
       begin_dx * begin_lambda * begin_lambda + 12.2068 * begin_dx *
      begin_lambda * begin_lambda * begin_lambda + 112.349 * begin_dx *
      begin_dy * begin_dy + -32.3766 * begin_dx * begin_dy * begin_dy *
      begin_lambda + 112.053 * begin_dx * begin_dx * begin_dx + -31.4918 *
      begin_dx * begin_dx * begin_dx * begin_lambda + 3.09757 * begin_y *
      begin_dx * begin_dy + -0.502456 * begin_y * begin_dx * begin_dy *
      begin_lambda + 0.0143842 * begin_y * begin_y * begin_dx + -0.000232369 *
       begin_y * begin_y * begin_dx * begin_lambda + 0.854018 * begin_x +
      0.0168942 * begin_x * begin_lambda + -0.0795893 * begin_x * begin_lambda
       * begin_lambda + 0.0660324 * begin_x * begin_lambda * begin_lambda *
      begin_lambda + 1.47562 * begin_x * begin_dy * begin_dy + -0.291411 *
      begin_x * begin_dy * begin_dy * begin_lambda + 4.65531 * begin_x *
      begin_dx * begin_dx + -0.931064 * begin_x * begin_dx * begin_dx *
      begin_lambda + 0.0287845 * begin_x * begin_y * begin_dy + -0.00228194 *
      begin_x * begin_y * begin_dy * begin_lambda + 1.27873e-05 * begin_x *
      begin_y * begin_y + 3.69551e-05 * begin_x * begin_y * begin_y *
      begin_lambda + 0.0457001 * begin_x * begin_x * begin_dx + -0.00672622 *
      begin_x * begin_x * begin_dx * begin_lambda + 3.23436e-05 * begin_x *
      begin_x * begin_x + 3.73746e-06 * begin_x * begin_x * begin_x *
      begin_lambda;
  pred_y = 103.55 * begin_dy + -5.2896 * begin_dy * begin_lambda + 8.58575 *
      begin_dy * begin_lambda * begin_lambda + -4.638 * begin_dy *
      begin_lambda * begin_lambda * begin_lambda + 108.383 * begin_dy *
      begin_dy * begin_dy + -27.0975 * begin_dy * begin_dy * begin_dy *
      begin_lambda + 100.358 * begin_dx * begin_dx * begin_dy + -13.394 *
      begin_dx * begin_dx * begin_dy * begin_lambda + 0.860714 * begin_y +
      -0.0151031 * begin_y * begin_lambda + -0.0277736 * begin_y *
      begin_lambda * begin_lambda + 0.0369927 * begin_y * begin_lambda *
      begin_lambda * begin_lambda + 4.54678 * begin_y * begin_dy * begin_dy +
      -0.787859 * begin_y * begin_dy * begin_dy * begin_lambda + 1.40017 *
      begin_y * begin_dx * begin_dx + -0.167978 * begin_y * begin_dx *
      begin_dx * begin_lambda + 0.0448829 * begin_y * begin_y * begin_dy +
      -0.00537016 * begin_y * begin_y * begin_dy * begin_lambda + 2.955e-05 *
      begin_y * begin_y * begin_y + 1.14243e-05 * begin_y * begin_y * begin_y
      * begin_lambda + 2.90727 * begin_x * begin_dx * begin_dy + -0.223117 *
      begin_x * begin_dx * begin_dy * begin_lambda + 0.0280037 * begin_x *
      begin_y * begin_dx + -0.00116075 * begin_x * begin_y * begin_dx *
      begin_lambda + 0.0140134 * begin_x * begin_x * begin_dy + 0.000187161 *
      begin_x * begin_x * begin_dy * begin_lambda + 1.60201e-05 * begin_x *
      begin_x * begin_y + 3.03695e-05 * begin_x * begin_x * begin_y *
      begin_lambda;
  pred_dx = 0.759703 * begin_dx + 0.0190482 * begin_dx * begin_lambda +
      -0.69393 * begin_dx * begin_lambda * begin_lambda + 0.673603 * begin_dx
      * begin_lambda * begin_lambda * begin_lambda + 2.46174 * begin_dx *
      begin_dy * begin_dy + -1.28829 * begin_dx * begin_dy * begin_dy *
      begin_lambda + 2.46753 * begin_dx * begin_dx * begin_dx + -1.2861 *
```

```
        begin_dx * begin_dx * begin_dx * begin_lambda + 0.0718457 * begin_y *
        begin_dx * begin_dy + -0.0233206 * begin_y * begin_dx * begin_dy *
        begin_lambda + 0.000437736 * begin_y * begin_y * begin_dx + -9.1986e-05
        * begin_y * begin_y * begin_dx * begin_lambda + -0.00315683 * begin_x +
        -0.00123079 * begin_x * begin_lambda + -0.00348602 * begin_x *
        begin_lambda * begin_lambda + 0.00430269 * begin_x * begin_lambda *
        begin_lambda * begin_lambda + 0.0362201 * begin_x * begin_dy * begin_dy
        + -0.0126474 * begin_x * begin_dy * begin_dy * begin_lambda + 0.111417 *
         begin_x * begin_dx * begin_dx + -0.0413785 * begin_x * begin_dx *
        begin_dx * begin_lambda + 0.000908059 * begin_x * begin_y * begin_dy +
        -0.00020389 * begin_x * begin_y * begin_dy * begin_lambda + 3.91009e-06
        * begin_x * begin_y * begin_y + -2.55481e-07 * begin_x * begin_y *
        begin_y * begin_lambda + 0.00142093 * begin_x * begin_x * begin_dx +
        -0.000418869 * begin_x * begin_x * begin_dx * begin_lambda + 4.39321e-06
         * begin_x * begin_x * begin_x + -1.07291e-06 * begin_x * begin_x *
        begin_x * begin_lambda;
    pred_dy = 1.0321 * begin_dy + -1.49666 * begin_dy * begin_lambda + 2.08734 *
         begin_dy * begin_lambda * begin_lambda + -1.0087 * begin_dy *
        begin_lambda * begin_lambda * begin_lambda + 2.33764 * begin_dy *
        begin_dy * begin_dy + -1.08664 * begin_dy * begin_dy * begin_dy *
        begin_lambda + 2.26501 * begin_dx * begin_dx * begin_dy + -0.964831 *
        begin_dx * begin_dx * begin_dy * begin_lambda + -0.00244446 * begin_y +
        -0.00497688 * begin_y * begin_lambda + 0.00308954 * begin_y *
        begin_lambda * begin_lambda + 0.000423841 * begin_y * begin_lambda *
        begin_lambda * begin_lambda + 0.107072 * begin_y * begin_dy * begin_dy +
         -0.0343315 * begin_y * begin_dy * begin_dy * begin_lambda + 0.033781 *
        begin_y * begin_dx * begin_dx + -0.00854399 * begin_y * begin_dx *
        begin_dx * begin_lambda + 0.00138251 * begin_y * begin_y * begin_dy +
        -0.000348079 * begin_y * begin_y * begin_dy * begin_lambda + 4.31541e-06
         * begin_y * begin_y * begin_y + -8.49656e-07 * begin_y * begin_y *
        begin_y * begin_lambda + 0.0699557 * begin_x * begin_dx * begin_dy +
        -0.0204927 * begin_x * begin_dx * begin_dy * begin_lambda + 0.0008824 *
        begin_x * begin_y * begin_dx + -0.000162373 * begin_x * begin_y *
        begin_dx * begin_lambda + 0.000437797 * begin_x * begin_x * begin_dy +
        -9.25942e-05 * begin_x * begin_x * begin_dy * begin_lambda + 3.88775e-06
         * begin_x * begin_x * begin_y + -2.14283e-07 * begin_x * begin_x *
        begin_y * begin_lambda;
    float dx1_domega0[2][2];
    dx1_domega0[0][0] = 100.843 + 9.60414 * begin_lambda + -18.8958 *
        begin_lambda * begin_lambda + 12.2068 * begin_lambda * begin_lambda *
        begin_lambda + 112.349 * begin_dy * begin_dy + -32.3766 * begin_dy *
        begin_dy * begin_lambda + 336.16 * begin_dx * begin_dx + -94.4753 *
        begin_dx * begin_dx * begin_lambda + 3.09757 * begin_y * begin_dy +
        -0.502456 * begin_y * begin_dy * begin_lambda + 0.0143842 * begin_y *
        begin_y + -0.000232369 * begin_y * begin_y * begin_lambda + 9.31061 *
        begin_x * begin_dx + -1.86213 * begin_x * begin_dx * begin_lambda +
        0.0457001 * begin_x * begin_x + -0.00672622 * begin_x * begin_x *
        begin_lambda+0.0f;
    dx1_domega0[0][1] = 224.698 * begin_dx * begin_dy + -64.7532 * begin_dx *
        begin_dy * begin_lambda + 3.09757 * begin_y * begin_dx + -0.502456 *
        begin_y * begin_dx * begin_lambda + 2.95124 * begin_x * begin_dy +
        -0.582822 * begin_x * begin_dy * begin_lambda + 0.0287845 * begin_x *
        begin_y + -0.00228194 * begin_x * begin_y * begin_lambda+0.0f;
    dx1_domega0[1][0] = 200.717 * begin_dx * begin_dy + -26.7881 * begin_dx *
        begin_dy * begin_lambda + 2.80035 * begin_y * begin_dx + -0.335956 *
        begin_y * begin_dx * begin_lambda + 2.90727 * begin_x * begin_dy +
        -0.223117 * begin_x * begin_dy * begin_lambda + 0.0280037 * begin_x *
        begin_y + -0.00116075 * begin_x * begin_y * begin_lambda+0.0f;
    dx1_domega0[1][1] = 103.55 + -5.2896 * begin_lambda + 8.58575 * begin_lambda
         * begin_lambda + -4.638 * begin_lambda * begin_lambda * begin_lambda +
        325.15 * begin_dy * begin_dy + -81.2925 * begin_dy * begin_dy *
        begin_lambda + 100.358 * begin_dx * begin_dx + -13.394 * begin_dx *
        begin_dx * begin_lambda + 9.09356 * begin_y * begin_dy + -1.57572 *
        begin_y * begin_dy * begin_lambda + 0.0448829 * begin_y * begin_y +
        -0.00537016 * begin_y * begin_y * begin_lambda + 2.90727 * begin_x *
        begin_dx + -0.223117 * begin_x * begin_dx * begin_lambda + 0.0140134 *
        begin_x * begin_x + 0.000187161 * begin_x * begin_x * begin_lambda+0.0f;
    float invJ[2][2];
    const float invdet = 1.0f/(dx1_domega0[0][0]*dx1_domega0[1][1] - dx1_domega0
        [0][1]*dx1_domega0[1][0]);
    invJ[0][0] =  dx1_domega0[1][1]*invdet;
    invJ[1][1] =  dx1_domega0[0][0]*invdet;
    invJ[0][1] = -dx1_domega0[0][1]*invdet;
```

```
    invJ[1][0] = -dx1_domega0[1][0]*invdet;
    const float dx1[2] = {out_x - pred_x, out_y - pred_y};
    for(int i=0;i<2;i++)
    {
      dx += invJ[0][i]*dx1[i];
      dy += invJ[1][i]*dx1[i];
    }
  }
  out_dx = pred_dx;
  out_dy = pred_dy;
```

Listing 3: Evaluate Aperture Jacobian

```
  const float dx00 = 0.854018 + 0.0168942 * lambda + -0.0795893 * lambda *
      lambda + 0.0660324 * lambda * lambda * lambda + 1.47562 * dy * dy +
      -0.291411 * dy * dy * lambda + 4.65531 * dx * dx + -0.931064 * dx * dx *
      lambda + 0.0287845 * y * dy + -0.00228194 * y * dy * lambda + 1.27873e-05
      * y * y + 3.69551e-05 * y * y * lambda + 0.0914003 * x * dx + -0.0134524 *
       x * dx * lambda + 9.70309e-05 * x * x + 1.12124e-05 * x * x * lambda+0.0f
      ;
  const float dx01 = 3.09757 * dx * dy + -0.502456 * dx * dy * lambda +
      0.0287684 * y * dx + -0.000464737 * y * dx * lambda + 0.0287845 * x * dy +
       -0.00228194 * x * dy * lambda + 2.55747e-05 * x * y + 7.39101e-05 * x * y
       * lambda+0.0f;
  const float dx02 = 100.843 + 9.60414 * lambda + -18.8958 * lambda * lambda +
      12.2068 * lambda * lambda * lambda + 112.349 * dy * dy + -32.3766 * dy *
      dy * lambda + 336.16 * dx * dx + -94.4753 * dx * dx * lambda + 3.09757 * y
       * dy + -0.502456 * y * dy * lambda + 0.0143842 * y * y + -0.000232369 * y
       * y * lambda + 9.31061 * x * dx + -1.86213 * x * dx * lambda + 0.0457001
      * x * x + -0.00672622 * x * x * lambda+0.0f;
  const float dx03 = 224.698 * dx * dy + -64.7532 * dx * dy * lambda + 3.09757 *
       y * dx + -0.502456 * y * dx * lambda + 2.95124 * x * dy + -0.582822 * x *
       dy * lambda + 0.0287845 * x * y + -0.00228194 * x * y * lambda+0.0f;
  const float dx04 = 9.60414 * dx + -37.7915 * dx * lambda + 36.6204 * dx *
      lambda * lambda + -32.3766 * dx * dy * dy + -31.4918 * dx * dx * dx +
      -0.502456 * y * dx * dy + -0.000232369 * y * y * dx + 0.0168942 * x +
      -0.159179 * x * lambda + 0.198097 * x * lambda * lambda + -0.291411 * x *
      dy * dy + -0.931064 * x * dx * dx + -0.00228194 * x * y * dy + 3.69551e-05
       * x * y * y + -0.00672622 * x * x * dx + 3.73746e-06 * x * x * x+0.0f;
  const float dx10 = 2.90727 * dx * dy + -0.223117 * dx * dy * lambda +
      0.0280037 * y * dx + -0.00116075 * y * dx * lambda + 0.0280268 * x * dy +
      0.000374322 * x * dy * lambda + 3.20402e-05 * x * y + 6.07389e-05 * x * y
      * lambda+0.0f;
  const float dx11 = 0.860714 + -0.0151031 * lambda + -0.0277736 * lambda *
      lambda + 0.0369927 * lambda * lambda * lambda + 4.54678 * dy * dy +
      -0.787859 * dy * dy * lambda + 1.40017 * dx * dx + -0.167978 * dx * dx *
      lambda + 0.0897658 * y * dy + -0.0107403 * y * dy * lambda + 8.865e-05 * y
       * y + 3.42728e-05 * y * y * lambda + 0.0280037 * x * dx + -0.00116075 * x
       * dx * lambda + 1.60201e-05 * x * x + 3.03695e-05 * x * x * lambda+0.0f;
  const float dx12 = 200.717 * dx * dy + -26.7881 * dx * dy * lambda + 2.80035 *
       y * dx + -0.335956 * y * dx * lambda + 2.90727 * x * dy + -0.223117 * x *
       dy * lambda + 0.0280037 * x * y + -0.00116075 * x * y * lambda+0.0f;
  const float dx13 = 103.55 + -5.2896 * lambda + 8.58575 * lambda * lambda +
      -4.638 * lambda * lambda * lambda + 325.15 * dy * dy + -81.2925 * dy * dy
      * lambda + 100.358 * dx * dx + -13.394 * dx * dx * lambda + 9.09356 * y *
      dy + -1.57572 * y * dy * lambda + 0.0448829 * y * y + -0.00537016 * y * y
      * lambda + 2.90727 * x * dx + -0.223117 * x * dx * lambda + 0.0140134 * x
      * x + 0.000187161 * x * x * lambda+0.0f;
  const float dx14 = -5.2896 * dy + 17.1715 * dy * lambda + -13.914 * dy *
      lambda * lambda + -27.0975 * dy * dy * dy + -13.394 * dx * dx * dy +
      -0.0151031 * y + -0.0555471 * y * lambda + 0.110978 * y * lambda * lambda
      + -0.787859 * y * dy * dy + -0.167978 * y * dx * dx + -0.00537016 * y * y
      * dy + 1.14243e-05 * y * y * y + -0.223117 * x * dx * dy + -0.00116075 * x
       * y * dx + 0.000187161 * x * x * dy + 3.03695e-05 * x * x * y+0.0f;
  const float dx20 = -0.00315683 + -0.00123079 * lambda + -0.00348602 * lambda *
       lambda + 0.00430269 * lambda * lambda * lambda + 0.0362201 * dy * dy +
      -0.0126474 * dy * dy * lambda + 0.111417 * dx * dx + -0.0413785 * dx * dx
      * lambda + 0.000908059 * y * dy + -0.00020389 * y * dy * lambda + 3.91009e
      -06 * y * y + -2.55481e-07 * y * y * lambda + 0.00284187 * x * dx +
      -0.000837738 * x * dx * lambda + 1.31796e-05 * x * x * x + -3.21873e-06 * x *
      x * lambda+0.0f;
```

```
const float dx21 = 0.0718457 * dx * dy + -0.0233206 * dx * dy * lambda +
    0.000875472 * y * dx + -0.000183972 * y * dx * lambda + 0.000908059 * x *
    dy + -0.00020389 * x * dy * lambda + 7.82017e-06 * x * y + -5.10961e-07 *
    x * y * lambda+0.0f;
const float dx22 = 0.759703 + 0.0190482 * lambda + -0.69393 * lambda * lambda
    + 0.673603 * lambda * lambda * lambda + 2.46174 * dy * dy + -1.28829 * dy
    * dy * lambda + 7.4026 * dx * dx + -3.8583 * dx * dx * lambda + 0.0718457
    * y * dy + -0.0233206 * y * dy * lambda + 0.000437736 * y * y + -9.1986e
    -05 * y * y * lambda + 0.222833 * x * dx + -0.0827569 * x * dx * lambda +
    0.00142093 * x * x + -0.000418869 * x * x * lambda+0.0f;
const float dx23 = 4.92349 * dx * dy + -2.57659 * dx * dy * lambda + 0.0718457
     * y * dx + -0.0233206 * y * dx * lambda + 0.0724401 * x * dy + -0.0252949
     * x * dy * lambda + 0.000908059 * x * y + -0.00020389 * x * y * lambda
    +0.0f;
const float dx24 = 0.0190482 * dx + -1.38786 * dx * lambda + 2.02081 * dx *
    lambda * lambda + -1.28829 * dx * dy * dy + -1.2861 * dx * dx * dx +
    -0.0233206 * y * dx * dy + -9.1986e-05 * y * y * dx + -0.00123079 * x +
    -0.00697204 * x * lambda + 0.0129081 * x * lambda * lambda + -0.0126474 *
    x * dy * dy + -0.0413785 * x * dx * dx + -0.00020389 * x * y * dy +
    -2.55481e-07 * x * y * y + -0.000418869 * x * x * dx + -1.07291e-06 * x *
    x * x+0.0f;
const float dx30 = 0.0699557 * dx * dy + -0.0204927 * dx * dy * lambda +
    0.0008824 * y * dx + -0.000162373 * y * dx * lambda + 0.000875595 * x * dy
     + -0.000185188 * x * dy * lambda + 7.7755e-06 * x * y + -4.28567e-07 * x
    * y * lambda+0.0f;
const float dx31 = -0.00244446 + -0.00497688 * lambda + 0.00308954 * lambda *
    lambda + 0.000423841 * lambda * lambda * lambda + 0.107072 * dy * dy +
    -0.0343315 * dy * dy * lambda + 0.033781 * dx * dx + -0.00854399 * dx * dx
     * lambda + 0.00276502 * y * dy + -0.000696158 * y * dy * lambda + 1.29462
    e-05 * y * y + -2.54897e-06 * y * y * lambda + 0.0008824 * x * dx +
    -0.000162373 * x * dx * lambda + 3.88775e-06 * x * x + -2.14283e-07 * x *
    x * lambda+0.0f;
const float dx32 = 4.53003 * dx * dy + -1.92966 * dx * dy * lambda + 0.067562
    * y * dx + -0.017088 * y * dx * lambda + 0.0699557 * x * dy + -0.0204927 *
     x * dy * lambda + 0.0008824 * x * y + -0.000162373 * x * y * lambda+0.0f;
const float dx33 = 1.0321 + -1.49666 * lambda + 2.08734 * lambda * lambda +
    -1.0087 * lambda * lambda * lambda + 7.01291 * dy * dy + -3.25993 * dy *
    dy * lambda + 2.26501 * dx * dx + -0.964831 * dx * dx * lambda + 0.214145
    * y * dy + -0.0686631 * y * dy * lambda + 0.00138251 * y * y +
    -0.000348079 * y * y * lambda + 0.0699557 * x * dx + -0.0204927 * x * dx *
     lambda + 0.000437797 * x * x + -9.25942e-05 * x * x * lambda+0.0f;
const float dx34 = -1.49666 * dy + 4.17469 * dy * lambda + -3.02609 * dy *
    lambda * lambda + -1.08664 * dy * dy * dy + -0.964831 * dx * dx * dy +
    -0.00497688 * y + 0.00617909 * y * lambda + 0.00127152 * y * lambda *
    lambda + -0.0343315 * y * dy * dy + -0.00854399 * y * dx * dx +
    -0.000348079 * y * y * dy + -8.49656e-07 * y * y * y + -0.0204927 * x * dx
     * dy + -0.000162373 * x * y * dx + -9.25942e-05 * x * x * dy + -2.14283e
    -07 * x * x * y+0.0f;
const float dx40 = +0.0f;
const float dx41 = +0.0f;
const float dx42 = +0.0f;
const float dx43 = +0.0f;
const float dx44 = 1+0.0f;
```

Listing 4: Forward Evaluation

```
const float out_x = 63.2264 * dx + 32.6497 * dx * lambda + -54.944 * dx *
    lambda * lambda + 33.3292 * dx * lambda * lambda * lambda + 57.7471 * dx *
     dy * dy + -138.072 * dx * dy * dy * lambda + 61.3699 * dx * dx * dx +
    -143.399 * dx * dx * dx * lambda + 1.76121 * y * dx * dy + -2.6537 * y *
    dx * dy * lambda + 0.0111823 * y * y * dx + -0.0129575 * y * y * dx *
    lambda + -0.626276 * x + -0.125257 * x * lambda + 0.057206 * x * lambda *
    lambda + 0.0502796 * x * lambda * lambda * lambda + 1.02119 * x * dy * dy
    + -1.49366 * x * dy * dy * lambda + 2.91904 * x * dx * dx + -4.30553 * x *
     dx * dx * lambda + 0.0240656 * x * y * dy + -0.0297037 * x * y * dy *
    lambda + -7.90387e-05 * x * y * y + -0.000138471 * x * y * y * lambda +
    0.0358763 * x * x * dx + -0.0428982 * x * x * dx * lambda + -7.63429e-05 *
     x * x * x + -0.000136509 * x * x * x * lambda;
const float out_y = 67.99 * dy + 5.96952 * dy * lambda + -5.37242 * dy *
    lambda * lambda + 2.95346 * dy * lambda * lambda * lambda + 53.1854 * dy *
     dy * dy + -129.231 * dy * dy * dy * lambda + 58.2154 * dx * dx * dy +
```

```
    −139.881 * dx * dx * dy * lambda + −0.578446 * y + −0.399169 * y * lambda
    + 0.573051 * y * lambda * lambda + −0.269919 * y * lambda * lambda *
    lambda + 2.6983 * y * dy * dy + −3.96884 * y * dy * dy * lambda + 0.995415
     * y * dx * dx + −1.49655 * y * dx * dx * lambda + 0.0340459 * y * y * dy
    + −0.0408553 * y * y * dy * lambda + −7.76228e−05 * y * y * y +
    −0.00013218 * y * y * y * lambda + 1.84152 * x * dx * dy + −2.74272 * x *
    dx * dy * lambda + 0.0235201 * x * y * dx + −0.0288369 * x * y * dx *
    lambda + 0.0119761 * x * x * dy + −0.0142142 * x * x * dy * lambda +
    −7.65261e−05 * x * x * y + −0.000137851 * x * x * y * lambda;
const float out_dx = −0.171409 * dx + 0.713318 * dx * lambda + −1.26916 * dx *
     lambda * lambda + 0.772566 * dx * lambda * lambda * lambda + 1.01376 * dx
     * dy * dy + −1.53791 * dx * dy * dy * lambda + 1.03026 * dx * dx * dx +
    −1.55944 * dx * dx * dx * lambda + 0.020078 * y * dx * dy + −0.0316224 * y
     * dx * dy * lambda + 0.000103109 * y * y * dx + −0.000158942 * y * y * dx
     * lambda + −0.0147988 * x + 0.00409983 * x * lambda + −0.00685401 * x *
    lambda * lambda + 0.00414031 * x * lambda * lambda * lambda + 0.0101327 *
    x * dy * dy + −0.0155023 * x * dy * dy * lambda + 0.0294299 * x * dx * dx
    + −0.0448993 * x * dx * dx * lambda + 0.000196838 * x * y * dy +
    −0.000312778 * x * y * dy * lambda + 1.58732e−07 * x * y * y + −1.55616e
    −06 * x * y * y * lambda + 0.000288688 * x * x * dx + −0.000440471 * x * x
     * dx * lambda + 1.1481e−07 * x * x * x + −1.44709e−06 * x * x * x *
    lambda;
const float out_dy = −0.116479 * dy + 0.434559 * dy * lambda + −0.780914 * dy
    * lambda * lambda + 0.47692 * dy * lambda * lambda * lambda + 0.716233 *
    dy * dy * dy + −0.991301 * dy * dy * dy * lambda + 0.525664 * dx * dx * dy
     + −0.658006 * dx * dx * dy * lambda + −0.0158343 * y + 0.0100773 * y *
    lambda + −0.0179733 * y * lambda * lambda + 0.0108352 * y * lambda *
    lambda * lambda + 0.0231207 * y * dy * dy + −0.0335459 * y * dy * dy *
    lambda + 0.00712624 * y * dx * dx + −0.0103961 * y * dx * dx * lambda +
    0.000239532 * y * y * dy + −0.000354859 * y * y * dy * lambda + −3.03081e
    −08 * y * y * y + −1.19451e−06 * y * y * y * lambda + 0.0131775 * x * dx *
     dy + −0.0182776 * x * dx * dy * lambda + 0.000149963 * x * y * dx +
    −0.000226161 * x * y * dx * lambda + 7.8347e−05 * x * x * dy + −0.00010865
     * x * x * dy * lambda + −4.37435e−08 * x * x * y + −1.17216e−06 * x * x *
     y * lambda;
const float out_lambda = 1 * lambda;
```

Listing 5: Reverse Evaluation

```
// evaluate approximate reverse polynomial
float sensor_x = 69.0791 * dx + −0.186176 * dx * lambda + 5.33476 * dx *
    lambda * lambda + −2.96981 * dx * lambda * lambda * lambda + 153.216 * dx
    * dy * dy + −263.973 * dx * dy * dy * lambda + 156.59 * dx * dx * dx +
    −266.756 * dx * dx * dx * lambda + 3.44211 * y * dx * dy + −5.19988 * y *
    dx * dy * lambda + 0.017323 * y * y * dx + −0.0259331 * y * y * dx *
    lambda + −0.037005 * x + −0.0347439 * x * lambda + 0.103966 * x * lambda *
     lambda + −0.0589274 * x * lambda * lambda * lambda + 1.71688 * x * dy *
    dy + −2.58662 * x * dy * dy * lambda + 5.27385 * x * dx * dx + −7.9057 * x
     * dx * dx * lambda + 0.0335471 * x * y * dy + −0.0509601 * x * y * dy *
    lambda + 0.000169438 * x * y * y + −0.000253996 * x * y * y * lambda +
    0.0519726 * x * x * dx + −0.0785288 * x * x * dx * lambda + 0.000172629 *
    x * x * x * x + −0.000261073 * x * x * x * x * lambda;
float sensor_y = 66.9386 * dy + 11.9344 * dy * lambda + −16.582 * dy * lambda
    * lambda + 9.90149 * dy * lambda * lambda * lambda + 147.295 * dy * dy *
    dy + −251.886 * dy * dy * dy * lambda + 153.308 * dx * dx * dy + −263.223
    * dx * dx * dy * lambda + −0.0590123 * y + 0.0906889 * y * lambda +
    −0.126835 * y * lambda * lambda + 0.079004 * y * lambda * lambda * lambda
    + 5.04943 * y * dy * dy + −7.50973 * y * dy * dy * lambda + 1.75802 * y *
    dx * dx + −2.60507 * y * dx * dx * lambda + 0.0498901 * y * y * dy +
    −0.0746927 * y * y * dy * lambda + 0.000165057 * y * y * y + −0.000246911
    * y * y * y * lambda + 3.41155 * x * dx * dy + −5.16826 * x * dx * dy *
    lambda + 0.0339111 * x * y * dx + −0.0510839 * x * y * dx * lambda +
    0.0168654 * x * x * dy + −0.0256828 * x * x * dy * lambda + 0.000167937 *
    x * x * y + −0.000253264 * x * x * y * lambda;
float sensor_dx = −0.621146 * dx + −0.156809 * dx * lambda + 0.124455 * dx *
    lambda * lambda + −0.00355763 * dx * lambda * lambda * lambda + 1.07296 *
    dx * dy * dy + −2.08177 * dx * dy * dy * lambda + 1.07348 * dx * dx * dx +
     −2.11456 * dx * dx * dx * lambda + 0.0289146 * y * dx * dy + −0.0448984 *
     y * dx * dy * lambda + 9.28794e−05 * y * y * dx + −0.00023664 * y * y *
    dx * lambda + −0.0147291 * x + 0.00375943 * x * lambda + −0.00617813 * x *
     lambda * lambda + 0.00360175 * x * lambda * lambda * lambda + 0.0136682 *
```

```
      x * dy * dy + -0.0208031 * x * dy * dy * lambda + 0.0416379 * x * dx * dx
      + -0.0648596 * x * dx * dx * lambda + 0.000152194 * x * y * dy +
      -0.000434364 * x * y * dy * lambda + 1.50633e-07 * x * y * y + -2.28425e
      -06 * x * y * y * lambda + 0.000225401 * x * x * dx + -0.000642212 * x * x
      * dx * lambda + 1.04311e-08 * x * x * x * x + -2.06439e-06 * x * x * x *
      lambda;
  float sensor_dy = -3.25268 * dy + 15.338 * dy * lambda + -29.0938 * dy *
      lambda * lambda + 17.7271 * dy * lambda * lambda * lambda + -7.36823 * dy
      * dy * dy + 13.0197 * dy * dy * dy * lambda + -18.2576 * dx * dx * dy +
      32.7679 * dx * dx * dy * lambda + -0.0327873 * y + 0.110976 * y * lambda +
       -0.209009 * y * lambda * lambda + 0.126656 * y * lambda * lambda * lambda
       + -0.192236 * y * dy * dy + 0.351767 * y * dy * dy * lambda + -0.156402 *
       y * dx * dx + 0.284889 * y * dx * dx * lambda + -0.00213022 * y * y * dy
      + 0.00355905 * y * y * dy * lambda + -8.0021e-06 * y * y * y + 1.22456e-05
       * y * y * y * lambda + -0.372944 * x * dx * dy + 0.677914 * x * dx * dy *
       lambda + -0.00348422 * x * y * dx + 0.00608412 * x * y * dx * lambda +
      -0.00210257 * x * x * dy + 0.00369773 * x * x * dy * lambda + -2.00502e-05
       * x * x * x * y + 3.3751e-05 * x * x * x * y * lambda;
  float sensor_lambda = 1 * lambda;
  // do newton iterations to get precise answer
  for(int it=0;it<8;it++)
  {
    float sensor_r_x =  sensor_x;
    float sensor_r_y =  sensor_y;
    float sensor_r_dx = - sensor_dx;
    float sensor_r_dy = - sensor_dy;
    float sensor_r_lambda =  sensor_lambda;
    // evaluate adjoint to get error vector
    float light_r_x = 63.2264 * sensor_r_dx + 32.6497 * sensor_r_dx *
        sensor_r_lambda + -54.944 * sensor_r_dx * sensor_r_lambda *
        sensor_r_lambda + 33.3292 * sensor_r_dx * sensor_r_lambda *
        sensor_r_lambda * sensor_r_lambda + 57.7471 * sensor_r_dx * sensor_r_dy
        * sensor_r_dy + -138.072 * sensor_r_dx * sensor_r_dy * sensor_r_dy *
        sensor_r_lambda + 61.3699 * sensor_r_dx * sensor_r_dx * sensor_r_dx +
        -143.399 * sensor_r_dx * sensor_r_dx * sensor_r_dx * sensor_r_lambda +
        1.76121 * sensor_r_y * sensor_r_dx * sensor_r_dy + -2.6537 * sensor_r_y
        * sensor_r_dx * sensor_r_dy * sensor_r_lambda + 0.0111823 * sensor_r_y *
         sensor_r_y * sensor_r_dx + -0.0129575 * sensor_r_y * sensor_r_y *
        sensor_r_dx * sensor_r_lambda + -0.626276 * sensor_r_x + -0.125257 *
        sensor_r_x * sensor_r_lambda + 0.057206 * sensor_r_x * sensor_r_lambda *
         sensor_r_lambda + 0.0502796 * sensor_r_x * sensor_r_lambda *
        sensor_r_lambda * sensor_r_lambda + 1.02119 * sensor_r_x * sensor_r_dy *
         sensor_r_dy + -1.49366 * sensor_r_x * sensor_r_dy * sensor_r_dy *
        sensor_r_lambda + 2.91904 * sensor_r_x * sensor_r_dx * sensor_r_dx +
        -4.30553 * sensor_r_x * sensor_r_dx * sensor_r_dx * sensor_r_lambda +
        0.0240656 * sensor_r_x * sensor_r_y * sensor_r_dy + -0.0297037 *
        sensor_r_x * sensor_r_y * sensor_r_dy * sensor_r_lambda + -7.90387e-05 *
         sensor_r_x * sensor_r_y * sensor_r_y + -0.000138471 * sensor_r_x *
        sensor_r_y * sensor_r_y * sensor_r_lambda + 0.0358763 * sensor_r_x *
        sensor_r_x * sensor_r_dx + -0.0428982 * sensor_r_x * sensor_r_x *
        sensor_r_dx * sensor_r_lambda + -7.63429e-05 * sensor_r_x * sensor_r_x *
         sensor_r_x + -0.000136509 * sensor_r_x * sensor_r_x * sensor_r_x *
        sensor_r_lambda;
    float light_r_y = 67.99 * sensor_r_dy + 5.96952 * sensor_r_dy *
        sensor_r_lambda + -5.37242 * sensor_r_dy * sensor_r_lambda *
        sensor_r_lambda + 2.95346 * sensor_r_dy * sensor_r_lambda *
        sensor_r_lambda * sensor_r_lambda + 53.1854 * sensor_r_dy * sensor_r_dy
        * sensor_r_dy + -129.231 * sensor_r_dy * sensor_r_dy * sensor_r_dy *
        sensor_r_lambda + 58.2154 * sensor_r_dx * sensor_r_dx * sensor_r_dy +
        -139.881 * sensor_r_dx * sensor_r_dx * sensor_r_dy * sensor_r_lambda +
        -0.578446 * sensor_r_y + -0.399169 * sensor_r_y * sensor_r_lambda +
        0.573051 * sensor_r_y * sensor_r_lambda * sensor_r_lambda + -0.269919 *
        sensor_r_y * sensor_r_lambda * sensor_r_lambda * sensor_r_lambda +
        2.6983 * sensor_r_y * sensor_r_dy * sensor_r_dy + -3.96884 * sensor_r_y
        * sensor_r_dy * sensor_r_dy * sensor_r_lambda + 0.995415 * sensor_r_y *
        sensor_r_dx * sensor_r_dx + -1.49655 * sensor_r_y * sensor_r_dx *
        sensor_r_dx * sensor_r_lambda + 0.0340459 * sensor_r_y * sensor_r_y *
        sensor_r_dy + -0.0408553 * sensor_r_y * sensor_r_y * sensor_r_dy *
        sensor_r_lambda + -7.76228e-05 * sensor_r_y * sensor_r_y * sensor_r_y +
        -0.00013218 * sensor_r_y * sensor_r_y * sensor_r_y * sensor_r_lambda +
        1.84152 * sensor_r_x * sensor_r_dx * sensor_r_dy + -2.74272 * sensor_r_x
         * sensor_r_dx * sensor_r_dy * sensor_r_lambda + 0.0235201 * sensor_r_x
        * sensor_r_y * sensor_r_dx + -0.0288369 * sensor_r_x * sensor_r_y *
```

```
        sensor_r_dx * sensor_r_lambda + 0.0119761 * sensor_r_x * sensor_r_x *
        sensor_r_dy + -0.0142142 * sensor_r_x * sensor_r_x * sensor_r_dy *
        sensor_r_lambda + -7.65261e-05 * sensor_r_x * sensor_r_x * sensor_r_y +
        -0.000137851 * sensor_r_x * sensor_r_x * sensor_r_y * sensor_r_lambda;
    float light_r_dx = -0.171409 * sensor_r_dx + 0.713318 * sensor_r_dx *
        sensor_r_lambda + -1.26916 * sensor_r_dx * sensor_r_lambda *
        sensor_r_lambda + 0.772566 * sensor_r_dx * sensor_r_lambda *
        sensor_r_lambda * sensor_r_lambda + 1.01376 * sensor_r_dx * sensor_r_dy
        * sensor_r_dy + -1.53791 * sensor_r_dx * sensor_r_dy * sensor_r_dy *
        sensor_r_lambda + 1.03026 * sensor_r_dx * sensor_r_dx * sensor_r_dx +
        -1.55944 * sensor_r_dx * sensor_r_dx * sensor_r_dx * sensor_r_lambda +
        0.020078 * sensor_r_y * sensor_r_dx * sensor_r_dy + -0.0316224 *
        sensor_r_y * sensor_r_dx * sensor_r_dy * sensor_r_lambda + 0.000103109 *
         sensor_r_y * sensor_r_y * sensor_r_dx + -0.000158942 * sensor_r_y *
        sensor_r_y * sensor_r_dx * sensor_r_lambda + -0.0147988 * sensor_r_x +
        0.00409983 * sensor_r_x * sensor_r_lambda + -0.00685401 * sensor_r_x *
        sensor_r_lambda * sensor_r_lambda + 0.00414031 * sensor_r_x *
        sensor_r_lambda * sensor_r_lambda + 0.0101327 *
        sensor_r_x * sensor_r_dy * sensor_r_dy + -0.0155023 * sensor_r_x *
        sensor_r_dy * sensor_r_dy * sensor_r_lambda + 0.0294299 * sensor_r_x *
        sensor_r_dx * sensor_r_dx + -0.0448993 * sensor_r_x * sensor_r_dx *
        sensor_r_dx * sensor_r_lambda + 0.000196838 * sensor_r_x * sensor_r_y *
        sensor_r_dy + -0.000312778 * sensor_r_x * sensor_r_y * sensor_r_dy *
        sensor_r_lambda + 1.58732e-07 * sensor_r_x * sensor_r_y * sensor_r_y +
        -1.55616e-06 * sensor_r_x * sensor_r_y * sensor_r_y * sensor_r_lambda +
        0.000288688 * sensor_r_x * sensor_r_x * sensor_r_dx + -0.000440471 *
        sensor_r_x * sensor_r_x * sensor_r_dx * sensor_r_lambda + 1.1481e-07 *
        sensor_r_x * sensor_r_x * sensor_r_x + -1.44709e-06 * sensor_r_x *
        sensor_r_x * sensor_r_x * sensor_r_lambda;
    float light_r_dy = -0.116479 * sensor_r_dy + 0.434559 * sensor_r_dy *
        sensor_r_lambda + -0.780914 * sensor_r_dy * sensor_r_lambda *
        sensor_r_lambda + 0.47692 * sensor_r_dy * sensor_r_lambda *
        sensor_r_lambda * sensor_r_lambda + 0.716233 * sensor_r_dy * sensor_r_dy
         * sensor_r_dy + -0.991301 * sensor_r_dy * sensor_r_dy * sensor_r_dy *
        sensor_r_lambda + 0.525664 * sensor_r_dx * sensor_r_dx * sensor_r_dy +
        -0.658006 * sensor_r_dx * sensor_r_dx * sensor_r_dy * sensor_r_lambda +
        -0.0158343 * sensor_r_y + 0.0100773 * sensor_r_y * sensor_r_lambda +
        -0.0179733 * sensor_r_y * sensor_r_lambda * sensor_r_lambda + 0.0108352
        * sensor_r_y * sensor_r_lambda * sensor_r_lambda * sensor_r_lambda +
        0.0231207 * sensor_r_y * sensor_r_dy * sensor_r_dy + -0.0335459 *
        sensor_r_y * sensor_r_dy * sensor_r_dy * sensor_r_lambda + 0.00712624 *
        sensor_r_y * sensor_r_dx * sensor_r_dx + -0.0103961 * sensor_r_y *
        sensor_r_dx * sensor_r_dx * sensor_r_lambda + 0.000239532 * sensor_r_y *
         sensor_r_y * sensor_r_dy + -0.000354859 * sensor_r_y * sensor_r_y *
        sensor_r_dy * sensor_r_lambda + -3.03081e-08 * sensor_r_y * sensor_r_y *
         sensor_r_y + -1.19451e-06 * sensor_r_y * sensor_r_y * sensor_r_y *
        sensor_r_lambda + 0.0131775 * sensor_r_x * sensor_r_dx * sensor_r_dy +
        -0.0182776 * sensor_r_x * sensor_r_dx * sensor_r_dy * sensor_r_lambda +
        0.000149963 * sensor_r_x * sensor_r_y * sensor_r_dx + -0.000226161 *
        sensor_r_x * sensor_r_y * sensor_r_dx * sensor_r_lambda + 7.8347e-05 *
        sensor_r_x * sensor_r_x * sensor_r_dy + -0.00010865 * sensor_r_x *
        sensor_r_x * sensor_r_dy * sensor_r_lambda + -4.37435e-08 * sensor_r_x *
         sensor_r_x * sensor_r_y + -1.17216e-06 * sensor_r_x * sensor_r_x *
        sensor_r_y * sensor_r_lambda;
    float light_r_lambda = 1 * sensor_r_lambda;
    light_r_dx = - light_r_dx;
    light_r_dy = - light_r_dy;
    const float d_light_0 = (x - light_r_x);
    const float d_light_1 = (y - light_r_y);
    const float d_light_2 = (dx - light_r_dx);
    const float d_light_3 = (dy - light_r_dy);
    const float d_light_4 = (lambda - light_r_lambda);
    // move through jacobian and correct sensor point
    float JI[5][5] = {{0.0f}};
    JI[4][4] = 1.0f;
    JI[0][0] = -0.037005 + -0.0347439 * light_r_lambda + 0.103966 *
        light_r_lambda * light_r_lambda + -0.0589274 * light_r_lambda *
        light_r_lambda * light_r_lambda + 1.71688 * light_r_dy * light_r_dy +
        -2.58662 * light_r_dy * light_r_dy * light_r_lambda + 5.27385 *
        light_r_dx * light_r_dx + -7.9057 * light_r_dx * light_r_dx *
        light_r_lambda + 0.0335471 * light_r_y * light_r_dy + -0.0509601 *
        light_r_y * light_r_dy * light_r_lambda + 0.000169438 * light_r_y *
        light_r_y + -0.000253996 * light_r_y * light_r_y * light_r_lambda +
```

```
    0.103945 * light_r_x * light_r_dx + -0.157058 * light_r_x * light_r_dx *
     light_r_lambda + 0.000517886 * light_r_x * light_r_x + -0.000783218 *
    light_r_x * light_r_x * light_r_lambda+0.0f;
JI[0][1] = 3.44211 * light_r_dx * light_r_dy + -5.19988 * light_r_dx *
    light_r_dy * light_r_lambda + 0.0346461 * light_r_y * light_r_dx +
    -0.0518662 * light_r_y * light_r_dx * light_r_lambda + 0.0335471 *
    light_r_x * light_r_dy + -0.0509601 * light_r_x * light_r_dy *
    light_r_lambda + 0.000338876 * light_r_x * light_r_y + -0.000507992 *
    light_r_x * light_r_y * light_r_lambda+0.0f;
JI[0][2] = 69.0791 + -0.186176 * light_r_lambda + 5.33476 * light_r_lambda *
     light_r_lambda + -2.96981 * light_r_lambda * light_r_lambda *
    light_r_lambda + 153.216 * light_r_dy * light_r_dy + -263.973 *
    light_r_dy * light_r_dy * light_r_lambda + 469.77 * light_r_dx *
    light_r_dx + -800.267 * light_r_dx * light_r_dx * light_r_lambda +
    3.44211 * light_r_y * light_r_dy + -5.19988 * light_r_y * light_r_dy *
    light_r_lambda + 0.017323 * light_r_y * light_r_y + -0.0259331 *
    light_r_y * light_r_y * light_r_lambda + 10.5477 * light_r_x *
    light_r_dx + -15.8114 * light_r_x * light_r_dx * light_r_lambda +
    0.0519726 * light_r_x * light_r_x + -0.0785288 * light_r_x * light_r_x *
     light_r_lambda+0.0f;
JI[0][3] = 306.432 * light_r_dx * light_r_dy + -527.945 * light_r_dx *
    light_r_dy * light_r_lambda + 3.44211 * light_r_y * light_r_dx +
    -5.19988 * light_r_y * light_r_dx * light_r_lambda + 3.43376 * light_r_x
     * light_r_dy + -5.17323 * light_r_x * light_r_dy * light_r_lambda +
    0.0335471 * light_r_x * light_r_y + -0.0509601 * light_r_x * light_r_y *
     light_r_lambda+0.0f;
JI[0][4] = -0.186176 * light_r_dx + 10.6695 * light_r_dx * light_r_lambda +
    -8.90944 * light_r_dx * light_r_lambda * light_r_lambda + -263.973 *
    light_r_dx * light_r_dy * light_r_dy + -266.756 * light_r_dx *
    light_r_dx * light_r_dx + -5.19988 * light_r_y * light_r_dx * light_r_dy
     + -0.0259331 * light_r_y * light_r_y * light_r_dx + -0.0347439 *
    light_r_x + 0.207931 * light_r_x * light_r_lambda + -0.176782 *
    light_r_x * light_r_lambda * light_r_lambda + -2.58662 * light_r_x *
    light_r_dy * light_r_dy + -7.9057 * light_r_x * light_r_dx * light_r_dx
    + -0.0509601 * light_r_x * light_r_y * light_r_dy + -0.000253996 *
    light_r_x * light_r_y * light_r_y + -0.0785288 * light_r_x * light_r_x *
     light_r_dx + -0.000261073 * light_r_x * light_r_x * light_r_x+0.0f;
JI[1][0] = 3.41155 * light_r_dx * light_r_dy + -5.16826 * light_r_dx *
    light_r_dy * light_r_lambda + 0.0339111 * light_r_y * light_r_dx +
    -0.0510839 * light_r_y * light_r_dx * light_r_lambda + 0.0337307 *
    light_r_x * light_r_dy + -0.0513656 * light_r_x * light_r_dy *
    light_r_lambda + 0.000335875 * light_r_x * light_r_y + -0.000506528 *
    light_r_x * light_r_y * light_r_lambda+0.0f;
JI[1][1] = -0.0590123 + 0.0906889 * light_r_lambda + -0.126835 *
    light_r_lambda * light_r_lambda + 0.079004 * light_r_lambda *
    light_r_lambda * light_r_lambda + 5.04943 * light_r_dy * light_r_dy +
    -7.50973 * light_r_dy * light_r_dy * light_r_lambda + 1.75802 *
    light_r_dx * light_r_dx + -2.60507 * light_r_dx * light_r_dx *
    light_r_lambda + 0.0997802 * light_r_y * light_r_dy + -0.149385 *
    light_r_y * light_r_dy * light_r_lambda + 0.00049517 * light_r_y *
    light_r_y + -0.000740733 * light_r_y * light_r_y * light_r_lambda +
    0.0339111 * light_r_x * light_r_dx + -0.0510839 * light_r_x * light_r_dx
     * light_r_lambda + 0.000167937 * light_r_x * light_r_x + -0.000253264 *
     light_r_x * light_r_x * light_r_lambda+0.0f;
JI[1][2] = 306.616 * light_r_dx * light_r_dy + -526.446 * light_r_dx *
    light_r_dy * light_r_lambda + 3.51604 * light_r_y * light_r_dx +
    -5.21014 * light_r_y * light_r_dx * light_r_lambda + 3.41155 * light_r_x
     * light_r_dy + -5.16826 * light_r_x * light_r_dy * light_r_lambda +
    0.0339111 * light_r_x * light_r_y + -0.0510839 * light_r_x * light_r_y *
     light_r_lambda+0.0f;
JI[1][3] = 66.9386 + 11.9344 * light_r_lambda + -16.582 * light_r_lambda *
    light_r_lambda + 9.90149 * light_r_lambda * light_r_lambda *
    light_r_lambda + 441.884 * light_r_dy * light_r_dy + -755.657 *
    light_r_dy * light_r_dy * light_r_lambda + 153.308 * light_r_dx *
    light_r_dx + -263.223 * light_r_dx * light_r_dx * light_r_lambda +
    10.0989 * light_r_y * light_r_dy + -15.0195 * light_r_y * light_r_dy *
    light_r_lambda + 0.0498901 * light_r_y * light_r_y + -0.0746927 *
    light_r_y * light_r_y * light_r_lambda + 3.41155 * light_r_x *
    light_r_dx + -5.16826 * light_r_x * light_r_dx * light_r_lambda +
    0.0168654 * light_r_x * light_r_x + -0.0256828 * light_r_x * light_r_x *
     light_r_lambda+0.0f;
JI[1][4] = 11.9344 * light_r_dy + -33.164 * light_r_dy * light_r_lambda +
    29.7045 * light_r_dy * light_r_lambda * light_r_lambda + -251.886 *
```

```
        light_r_dy * light_r_dy * light_r_dy + -263.223 * light_r_dx *
        light_r_dx * light_r_dy + 0.0906889 * light_r_y + -0.253671 * light_r_y
        * light_r_lambda + 0.237012 * light_r_y * light_r_lambda *
        light_r_lambda + -7.50973 * light_r_y * light_r_dy * light_r_dy +
        -2.60507 * light_r_y * light_r_dx * light_r_dx + -0.0746927 * light_r_y
        * light_r_y * light_r_dy + -0.000246911 * light_r_y * light_r_y *
        light_r_y + -5.16826 * light_r_x * light_r_dx * light_r_dy + -0.0510839
        * light_r_x * light_r_y * light_r_dx + -0.0256828 * light_r_x *
        light_r_x * light_r_dy + -0.000253264 * light_r_x * light_r_x *
        light_r_y+0.0f;
    JI[2][0] = -0.0147291 + 0.00375943 * light_r_lambda + -0.00617813 *
        light_r_lambda * light_r_lambda + 0.00360175 * light_r_lambda *
        light_r_lambda * light_r_lambda + 0.0136682 * light_r_dy * light_r_dy +
        -0.0208031 * light_r_dy * light_r_dy * light_r_lambda + 0.0416379 *
        light_r_dx * light_r_dx + -0.0648596 * light_r_dx * light_r_dx *
        light_r_lambda + 0.000152194 * light_r_y * light_r_dy + -0.000434364 *
        light_r_y * light_r_dy * light_r_lambda + 1.50633e-07 * light_r_y *
        light_r_y + -2.28425e-06 * light_r_y * light_r_y * light_r_lambda +
        0.000450802 * light_r_x * light_r_dx + -0.00128442 * light_r_x *
        light_r_dx * light_r_lambda + 3.12934e-08 * light_r_x * light_r_x +
        -6.19318e-06 * light_r_x * light_r_x * light_r_lambda+0.0f;
    JI[2][1] = 0.0289146 * light_r_dx * light_r_dy + -0.0448984 * light_r_dx *
        light_r_dy * light_r_lambda + 0.000185759 * light_r_y * light_r_dx +
        -0.000473279 * light_r_y * light_r_dx * light_r_lambda + 0.000152194 *
        light_r_x * light_r_dy + -0.000434364 * light_r_x * light_r_dy *
        light_r_lambda + 3.01267e-07 * light_r_x * light_r_y + -4.56851e-06 *
        light_r_x * light_r_y * light_r_lambda+0.0f;
    JI[2][2] = -0.621146 + -0.156809 * light_r_lambda + 0.124455 *
        light_r_lambda * light_r_lambda + -0.00355763 * light_r_lambda *
        light_r_lambda * light_r_lambda + 1.07296 * light_r_dy * light_r_dy +
        -2.08177 * light_r_dy * light_r_dy * light_r_lambda + 3.22045 *
        light_r_dx * light_r_dx + -6.34368 * light_r_dx * light_r_dx *
        light_r_lambda + 0.0289146 * light_r_y * light_r_dy + -0.0448984 *
        light_r_y * light_r_dy * light_r_lambda + 9.28794e-05 * light_r_y *
        light_r_y + -0.00023664 * light_r_y * light_r_y * light_r_lambda +
        0.0832757 * light_r_x * light_r_dx + -0.129719 * light_r_x * light_r_dx
        * light_r_lambda + 0.000225401 * light_r_x * light_r_x + -0.000642212 *
        light_r_x * light_r_x * light_r_lambda+0.0f;
    JI[2][3] = 2.14593 * light_r_dx * light_r_dy + -4.16355 * light_r_dx *
        light_r_dy * light_r_lambda + 0.0289146 * light_r_y * light_r_dx +
        -0.0448984 * light_r_y * light_r_dx * light_r_lambda + 0.0273365 *
        light_r_x * light_r_dy + -0.0416063 * light_r_x * light_r_dy *
        light_r_lambda + 0.000152194 * light_r_x * light_r_y + -0.000434364 *
        light_r_x * light_r_y * light_r_lambda+0.0f;
    JI[2][4] = -0.156809 * light_r_dx + 0.248909 * light_r_dx * light_r_lambda +
         -0.0106729 * light_r_dx * light_r_lambda * light_r_lambda + -2.08177 *
        light_r_dx * light_r_dy * light_r_dy + -2.11456 * light_r_dx *
        light_r_dx + -0.0448984 * light_r_y * light_r_dx *
        light_r_dy + -0.00023664 * light_r_y * light_r_y * light_r_dx +
        0.00375943 * light_r_x + -0.0123563 * light_r_x * light_r_lambda +
        0.0108053 * light_r_x * light_r_lambda * light_r_lambda + -0.0208031 *
        light_r_x * light_r_dy * light_r_dy + -0.0648596 * light_r_x *
        light_r_dx * light_r_dx + -0.000434364 * light_r_x * light_r_y *
        light_r_dy + -2.28425e-06 * light_r_x * light_r_y * light_r_y +
        -0.000642212 * light_r_x * light_r_x * light_r_dx + -2.06439e-06 *
        light_r_x * light_r_x * light_r_x+0.0f;
    JI[3][0] = -0.372944 * light_r_dx * light_r_dy + 0.677914 * light_r_dx *
        light_r_dy * light_r_lambda + -0.00348422 * light_r_y * light_r_dx +
        0.00608412 * light_r_y * light_r_dx * light_r_lambda + -0.00420514 *
        light_r_x * light_r_dy + 0.00739545 * light_r_x * light_r_dy *
        light_r_lambda + -4.01004e-05 * light_r_x * light_r_y + 6.7502e-05 *
        light_r_x * light_r_y * light_r_lambda+0.0f;
    JI[3][1] = -0.0327873 + 0.110976 * light_r_lambda + -0.209009 *
        light_r_lambda * light_r_lambda + 0.126656 * light_r_lambda *
        light_r_lambda * light_r_lambda + -0.192236 * light_r_dy * light_r_dy +
        0.351767 * light_r_dy * light_r_dy * light_r_lambda + -0.156402 *
        light_r_dx * light_r_dx + 0.284889 * light_r_dx * light_r_dx *
        light_r_lambda + -0.00426044 * light_r_y * light_r_dy + 0.0071181 *
        light_r_y * light_r_dy * light_r_lambda + -2.40063e-05 * light_r_y *
        light_r_y + 3.67368e-05 * light_r_y * light_r_y * light_r_lambda +
        -0.00348422 * light_r_x * light_r_dx + 0.00608412 * light_r_x *
        light_r_dx * light_r_lambda + -2.00502e-05 * light_r_x * light_r_x +
        3.3751e-05 * light_r_x * light_r_x * light_r_lambda+0.0f;
```

```
    JI[3][2] = -36.5152 * light_r_dx * light_r_dy + 65.5358 * light_r_dx *
        light_r_dy * light_r_lambda + -0.312804 * light_r_y * light_r_dx +
        0.569778 * light_r_y * light_r_dx * light_r_lambda + -0.372944 *
        light_r_x * light_r_dy + 0.677914 * light_r_x * light_r_dy *
        light_r_lambda + -0.00348422 * light_r_x * light_r_y + 0.00608412 *
        light_r_x * light_r_y * light_r_lambda+0.0f;
    JI[3][3] = -3.25268 + 15.338 * light_r_lambda + -29.0938 * light_r_lambda *
        light_r_lambda + 17.7271 * light_r_lambda * light_r_lambda *
        light_r_lambda + -22.1047 * light_r_dy * light_r_dy + 39.059 *
        light_r_dy * light_r_dy * light_r_lambda + -18.2576 * light_r_dx *
        light_r_dx + 32.7679 * light_r_dx * light_r_dx * light_r_lambda +
        -0.384472 * light_r_y * light_r_dy + 0.703534 * light_r_y * light_r_dy *
         light_r_lambda + -0.00213022 * light_r_y * light_r_y + 0.00355905 *
        light_r_y * light_r_y * light_r_lambda + -0.372944 * light_r_x *
        light_r_dx + 0.677914 * light_r_x * light_r_dx * light_r_lambda +
        -0.00210257 * light_r_x * light_r_x + 0.00369773 * light_r_x * light_r_x
         * light_r_lambda+0.0f;
    JI[3][4] = 15.338 * light_r_dy + -58.1876 * light_r_dy * light_r_lambda +
        53.1814 * light_r_dy * light_r_lambda * light_r_lambda + 13.0197 *
        light_r_dy * light_r_dy * light_r_dy + 32.7679 * light_r_dx * light_r_dx
         * light_r_dy + 0.110976 * light_r_y + -0.418018 * light_r_y *
        light_r_lambda + 0.379968 * light_r_y * light_r_lambda * light_r_lambda
       + 0.351767 * light_r_y * light_r_dy * light_r_dy + 0.284889 * light_r_y
        * light_r_dx * light_r_dx + 0.00355905 * light_r_y * light_r_y *
        light_r_dy + 1.22456e-05 * light_r_y * light_r_y * light_r_y + 0.677914
        * light_r_x * light_r_dx * light_r_dy + 0.00608412 * light_r_x *
        light_r_y * light_r_dx + 0.00369773 * light_r_x * light_r_x * light_r_dy
         + 3.3751e-05 * light_r_x * light_r_x * light_r_y+0.0f;
    JI[4][0] = +0.0f;
    JI[4][1] = +0.0f;
    JI[4][2] = +0.0f;
    JI[4][3] = +0.0f;
    JI[4][4] = 1+0.0f;
    sensor_x += JI[0][0] * d_light_0;
    sensor_x += JI[0][1] * d_light_1;
    sensor_x += JI[0][2] * d_light_2;
    sensor_x += JI[0][3] * d_light_3;
    sensor_x += JI[0][4] * d_light_4;
    sensor_y += JI[1][0] * d_light_0;
    sensor_y += JI[1][1] * d_light_1;
    sensor_y += JI[1][2] * d_light_2;
    sensor_y += JI[1][3] * d_light_3;
    sensor_y += JI[1][4] * d_light_4;
    sensor_dx += JI[2][0] * d_light_0;
    sensor_dx += JI[2][1] * d_light_1;
    sensor_dx += JI[2][2] * d_light_2;
    sensor_dx += JI[2][3] * d_light_3;
    sensor_dx += JI[2][4] * d_light_4;
    sensor_dy += JI[3][0] * d_light_0;
    sensor_dy += JI[3][1] * d_light_1;
    sensor_dy += JI[3][2] * d_light_2;
    sensor_dy += JI[3][3] * d_light_3;
    sensor_dy += JI[3][4] * d_light_4;
    sensor_lambda += JI[4][0] * d_light_0;
    sensor_lambda += JI[4][1] * d_light_1;
    sensor_lambda += JI[4][2] * d_light_2;
    sensor_lambda += JI[4][3] * d_light_3;
    sensor_lambda += JI[4][4] * d_light_4;
  }
```