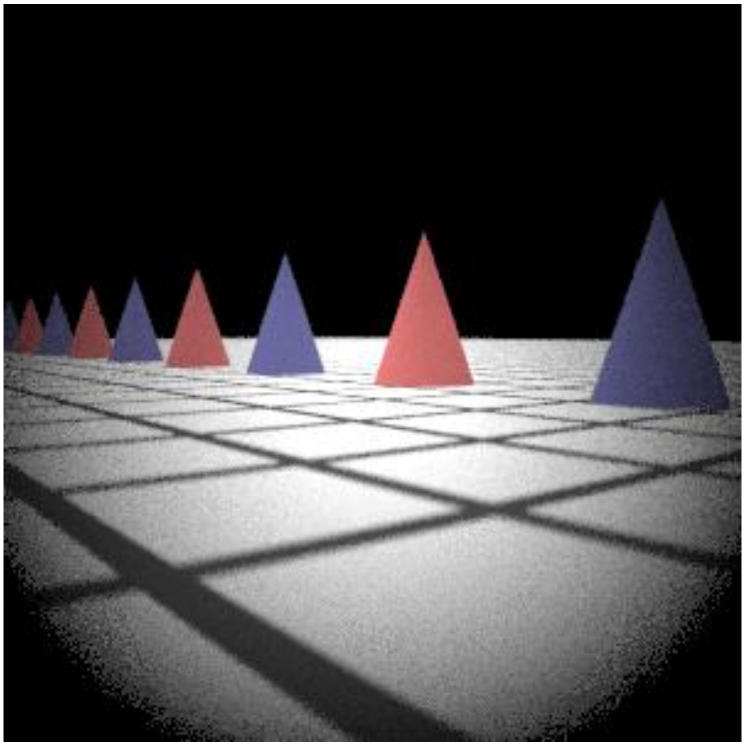
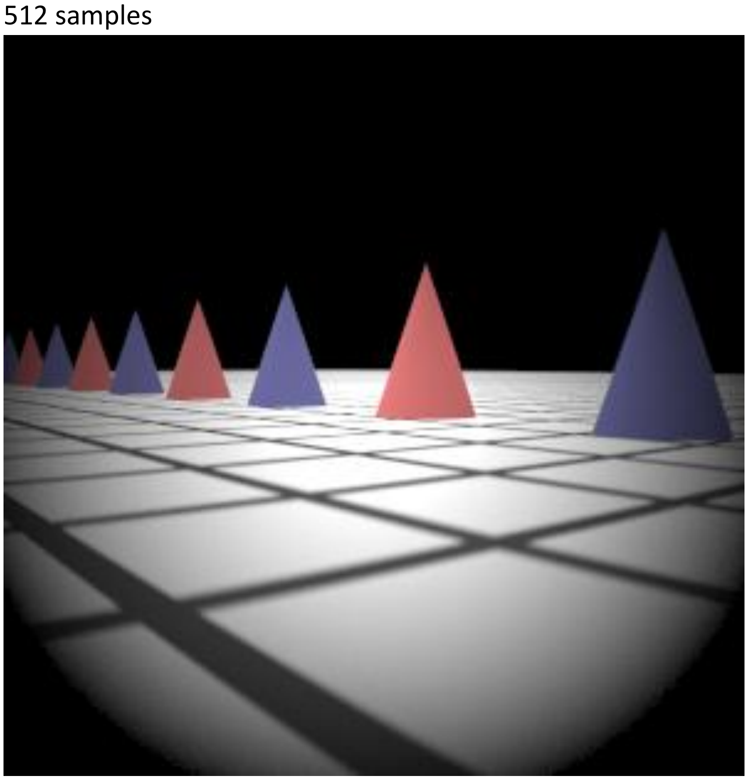
Rendering - HW2

Result:

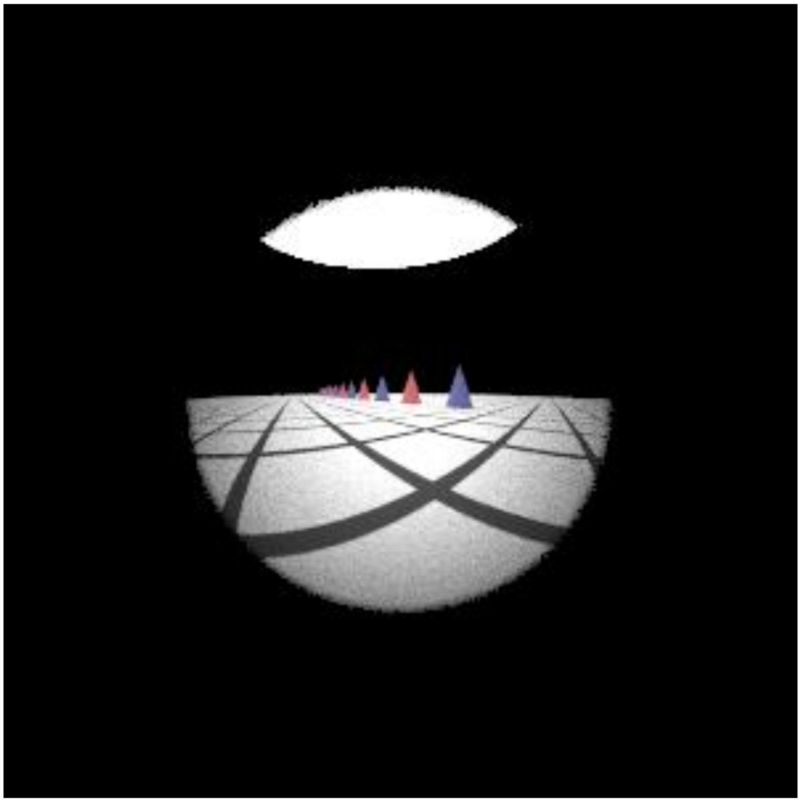
1. dof-dragons.dgauss.pbrt
    a. 32 samples



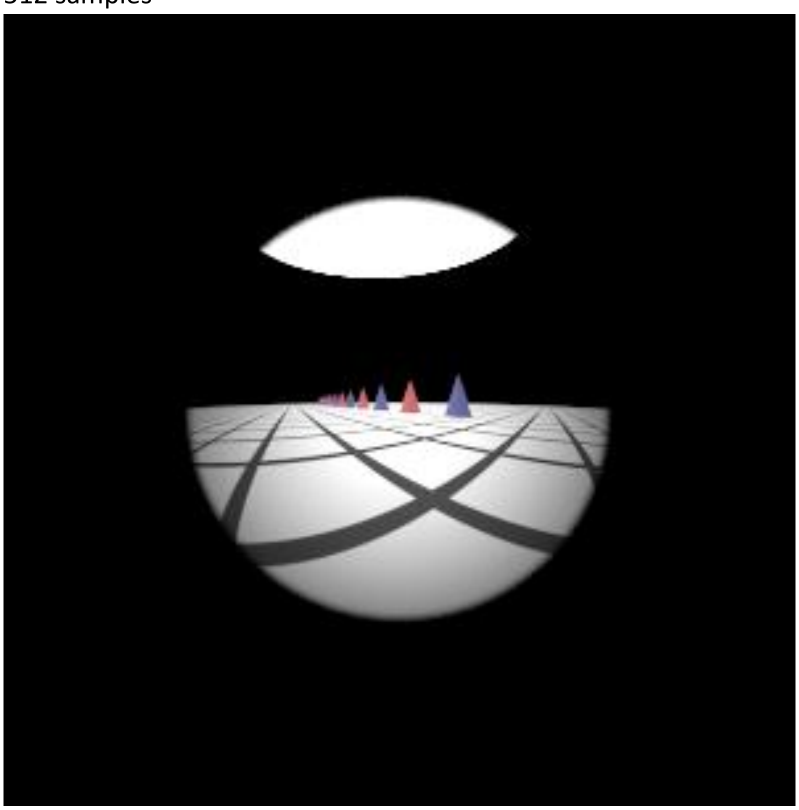
    b. 512 samples



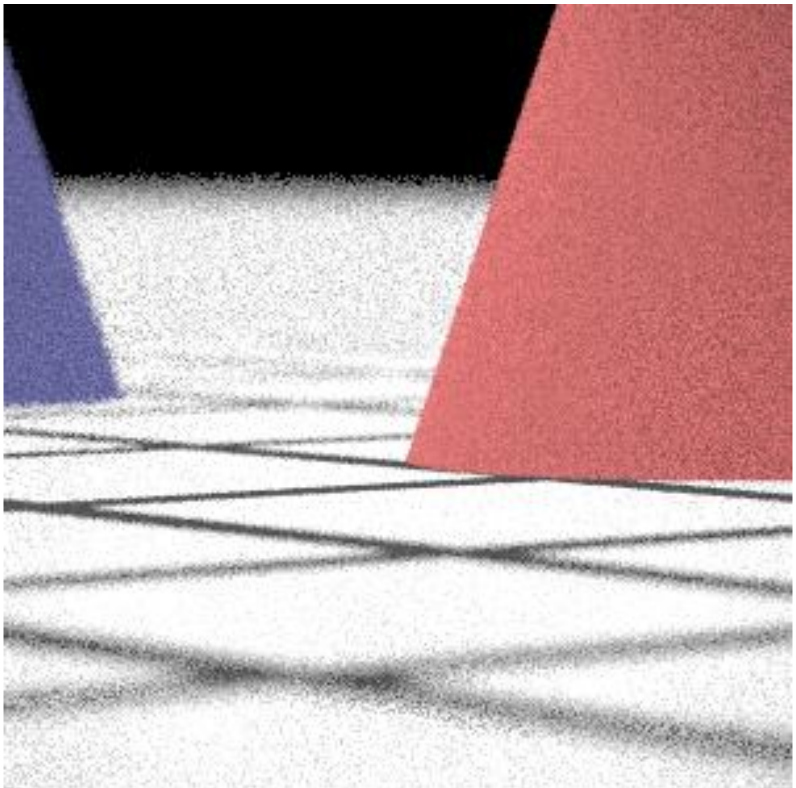
2. dof-dragons.fisheye.pbrt
    a. 32 samples



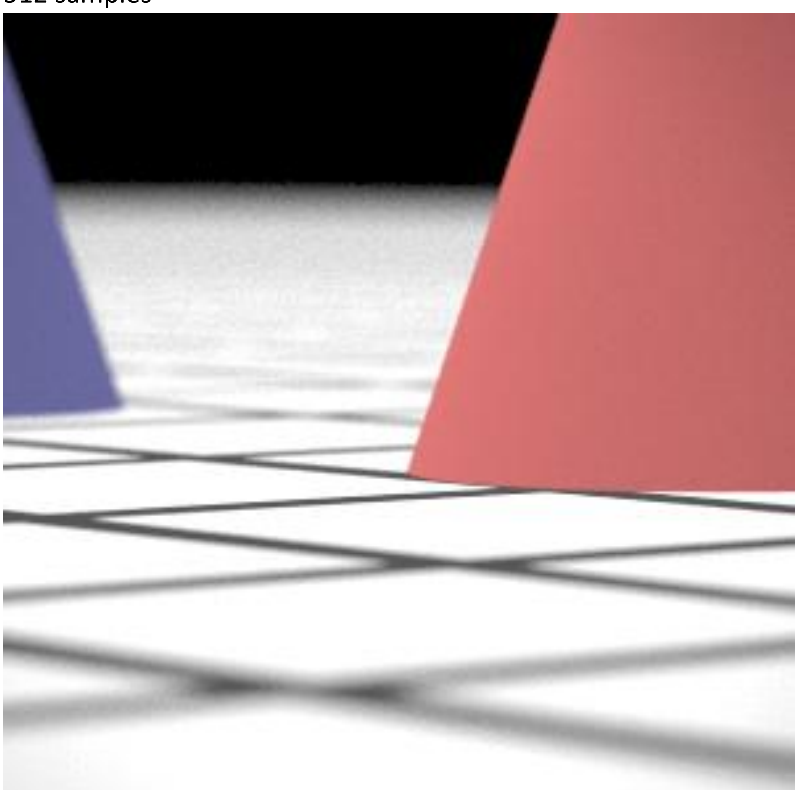
    b. 512 samples



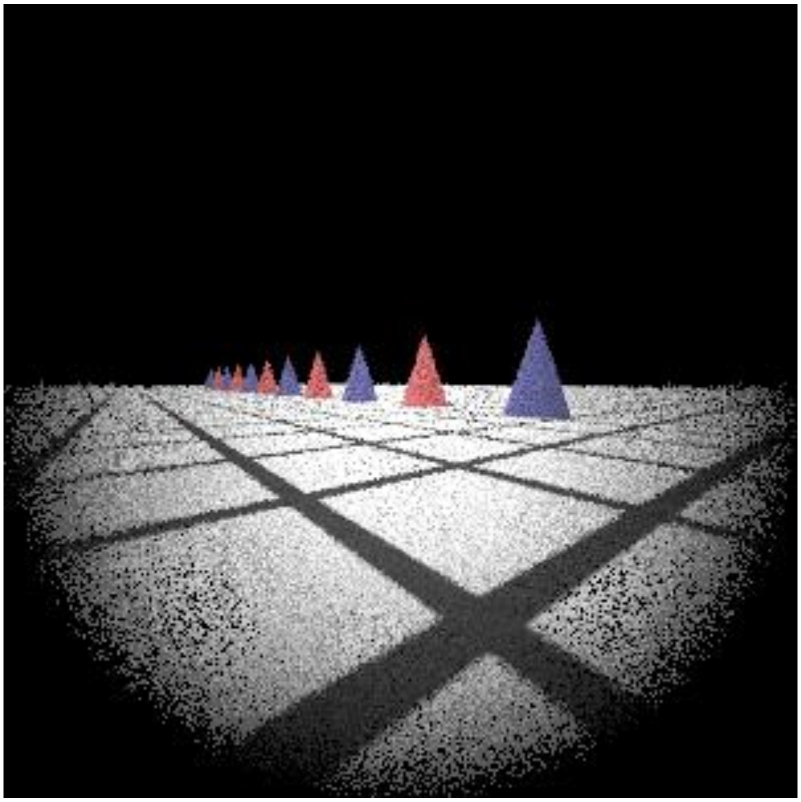
3. dof-dragons.telephoto.pbrt
    a. 32 samples



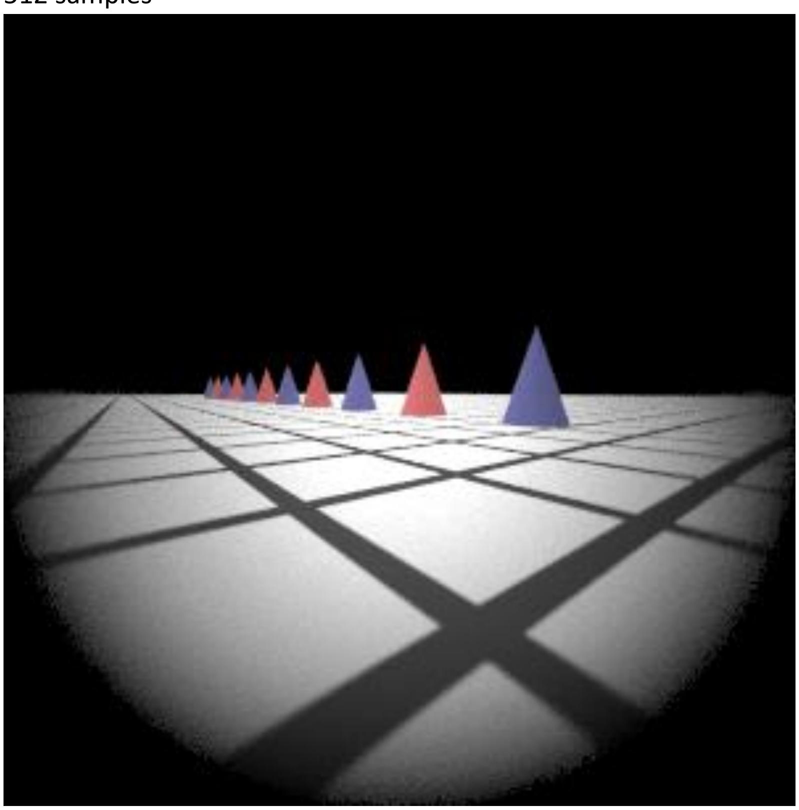
    b. 512 samples



4. dof-dragons.wide.pbrt
    a. 32 samples

    b. 512 samples

Method:

1. Collect relation data – initial()
   a. Read XXXX.pbrt data
      Get by CreateRealisticCamera() method for system arthitecture.

```
// Extract common camera parameters from \use{ParamSet}
float hither = params.FindOneFloat("hither", -1);
float yon = params.FindOneFloat("yon", -1);
float shutteropen = params.FindOneFloat("shutteropen", -1);
float shutterclose = params.FindOneFloat("shutterclose", -1);

// Realistic camera-specific parameters
string specfile = params.FindOneString("specfile", "");
float filmdistance = params.FindOneFloat("filmdistance", 70.0);
float fstop = params.FindOneFloat("aperture_diameter", 1.0);
float filmdiag = params.FindOneFloat("filmdiag", 35.0);

Assert(hither != -1 && yon != -1 && shutteropen != -1 &&
    shutterclose != -1 && filmdistance != -1);
if (specfile == "") {
    Severe("No lens spec file supplied!\n");
}
return new RealisticCamera(cam2world, hither, yon,
    shutteropen, shutterclose, filmdistance, fstop,
    specfile, filmdiag, film);
```

   b. Read XXXX.dm file
      In initial method - RealisticCamera()
      Read file and split by ‘tab’ to get float data.
      All store in a vector by lensData data structure.

```
//Now read in lens information from the spec file
ifstream inFile(specfilename.c_str());
std::string line;
while (std::getline(inFile, line))
{
    if (line.at(0) == '#')
        continue;

    std::vector<std::string> elems;
    split(line, '\t', elems);

    lensData temp;
    // 1
    temp.lensRadius = std::stof(elems[0]);
    temp.lensRadiusSquare = temp.lensRadius * temp.lensRadius;
    // 2
    temp.axisPos = std::stof(elems[1]);
    // 3
    temp.refractiveIndex = std::stof(elems[2]);
    if(temp.refractiveIndex == 0.f)
        temp.refractiveIndex = 1.f;
    // 4
    float tempf = std::stof(elems[3]);
    temp.apertureRadius = tempf * 0.5;
    temp.apertureRadiusSquare = temp.apertureRadius * temp.apertureRadius;

    lens.push_back(temp);
    numLens++;
}
inFile.close();

lens.back().axisPos = filmDistance;
```

2. Ray tracing – GenerateRay()
    a. Generate a ray

    Create a ray by sample a point in the near lens and the point in the film in camera space

```
// STEP 1 - generate first ray
// use sample->imageX and sample->imageY to get raster-space coordinates of the samp
Point Pras(sample.imageX, sample.imageY, 0.0);
Point Pcamera;
Pcamera = RasterToCamera(Pras);

// use sample->lensU and sample->lensV to get a sample position on the lens
float lensU, lensV;
ConcentricSampleDisk(sample.lensU,sample.lensV,&lensU,&lensV);
lensU *= lens.back().apertureRadius;
lensV *= lens.back().apertureRadius;

float d = sqrtf(lens.back().lensRadiusSquare - lens.back().apertureRadiusSquare);
if (lens.back().lensRadius < 0.f)
    d = -d;
float lensZ = lens.back().centerInCameraAxis + d;

// the sample point in lens
Point Plens(lensU,lensV,lensZ);
Vector dir = Normalize(Plens - Pcamera);
Ray tempRay(Pcamera, dir, 0.f, INFINITY);
```

b. Ray-lens interaction

For each lens face element, from near to far, calculate the intersection between ray and lens, and change the ray origin and direction.

```
// 1. intersect test and calculate t
float t;
lensData templen = lens[i];
if (templen.lensRadius != 0.f){ // if surface is planar or not

    Vector D = tempRay.o - Point(0, 0, templen.centerInCameraAxis);

    // check intersect or not
    float a = Dot(tempRay.d, tempRay.d);
    float b = 2 * Dot(D, tempRay.d);
    float c = Dot(D, D) - (templen.lensRadius*templen.lensRadius);
    float discriminant = (b*b) - (4 * a*c);
    if (discriminant < 0){
        ray = NULL;
        return 0.f;
    }

    // get intersect point
    float distSqrt = sqrt(discriminant);
    float t0 = (-b - distSqrt) / (2.0* a);
    float t1 = (-b + distSqrt) / (2.0* a);
    if (t0 > t1){   // make sure t0 is smaller than t1
        float temp = t0;
        t0 = t1;
        t1 = temp;
    }
    if (templen.lensRadius < 0)
        t = t0; // Lens
    else
        t = t1; // Concave lens
}
else
{
    Vector normal = Vector(0.f, 0.f, -1.f);
    t = -(Dot(Vector(tempRay.o), normal) + templen.centerInCameraAxis) / Dot(temp
}
```

```
// 2. aperture test
Point pointOnLens = tempRay(t);
float lenghtofV = sqrt(pointOnLens.x*pointOnLens.x + pointOnLens.y*pointOnLens.y);
if (lenghtofV > lens.at(i).apertureRadius){
    ray = NULL;
    return 0.0f;
}
```

```
// 3. prepare data for refraction - indexRatio, I, N
// refractiveIndex ratio
Point centerOfLens(0, 0, lens.at(i).centerInCameraAxis);
float curIndex = (lens.at(i).refractiveIndex);
float nextIndex;
if (i != 0)
    nextIndex = (lens.at(i - 1).refractiveIndex);
else
    nextIndex = 1; //Air after that
float indexRadio = curIndex / nextIndex;

// normal
Vector I = Normalize(tempRay.d);
Vector N;
if (templen.lensRadius != 0.f)
    N = Normalize(pointOnLens - Point(0, 0, templen.centerInCameraAxis));
else
    N = Vector(0.f, 0.f, -1.f);
if (Dot(-I, N) < 0)
    N = -N;
```

```
// 4. Calculate Refraction
float c1 = -Dot(I, N);
float c2Squared = 1.f - ((indexRadio*indexRadio)*(1.f - (c1*c1)));
if (c2Squared < 0){
    ray = NULL;
    return 0.0f;
}
float c2 = sqrt(c2Squared);
Vector T = indexRadio*I + (indexRadio*c1 - c2)*N;

tempRay.o = pointOnLens;
tempRay.d = T;
```

c. Return ray and weight

```
// STEP 3: Calculate the ray weights
//Use the back disc approximation for exit pupil
Vector filmNormal(0.f, 0.f, 1.f);
Vector diskToFilm = Normalize(Plens - Pcamera);
float cosTheta = Dot(filmNormal, diskToFilm);
return ((lens.back().apertureRadiusSquare * M_PI) / pow(fabs(filmPlaneZInCameraAxis -
```