Cameras

Digital Image Synthesis

Yung-Yu Chuang

10/25/2007

with slides by Pat Hanrahan and Matt Pharr

Camera

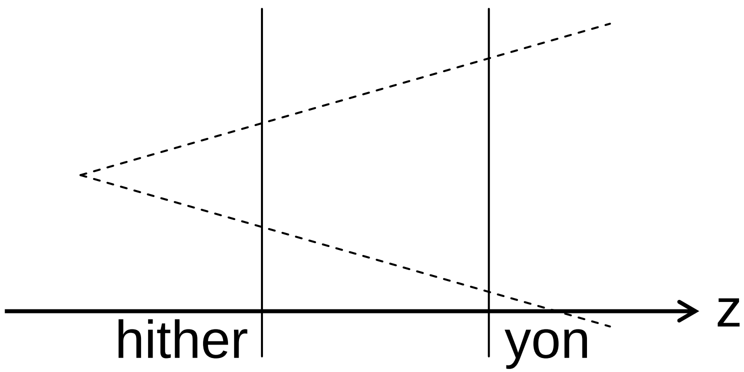
```
class Camera {
public:
  virtual float GenerateRay(const Sample
                &sample, Ray *ray) const = 0;
  ...
  Film *film;
protected:
  Transform WorldToCamera, CameraToWorld;
  float ClipHither, ClipYon;
  float ShutterOpen, ShutterClose;
};
```

return a weight, useful for simulating real lens

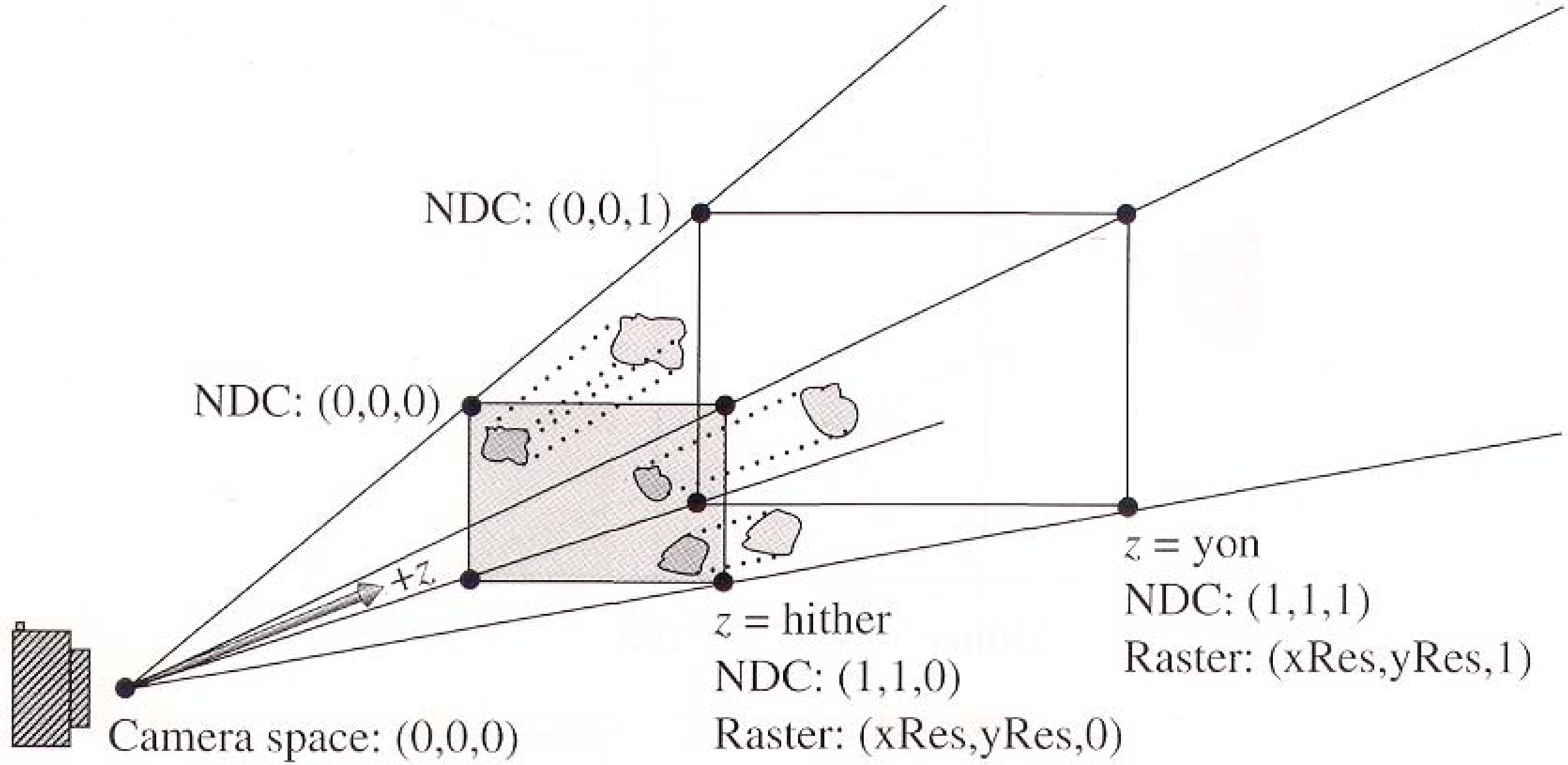
sample position at the image plane

corresponding normalized ray in the world space

for simulating motion blur, not Implemented yet



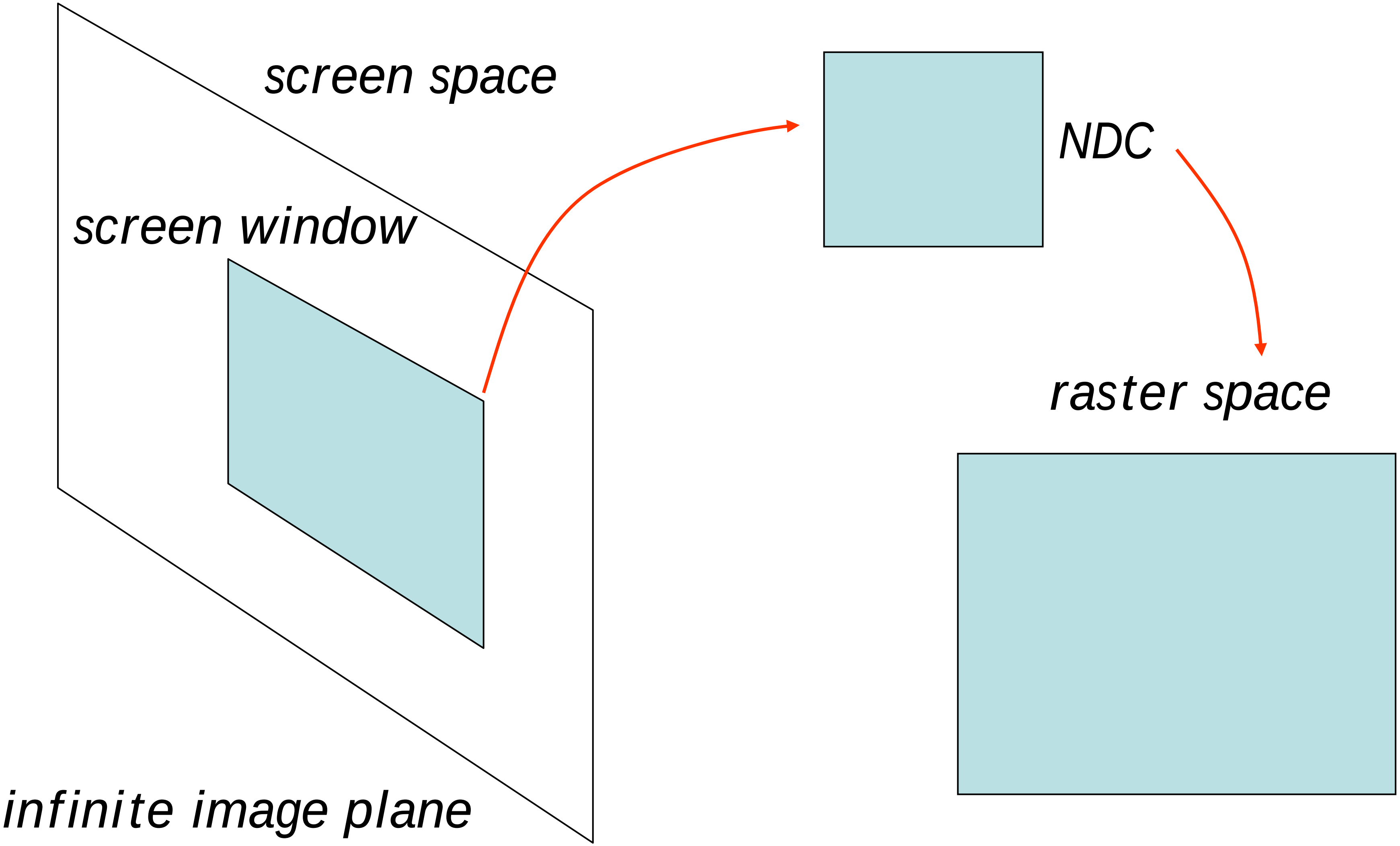
Camera space



Coordinate spaces

- world space
- object space
- camera space (origin: camera position, z: viewing direction, y: up direction)
- screen space: a 3D space defined on the image plane, z ranges from 0(near) to 1(far)
- normalized device space (NDC): (x, y) ranges from (0,0) to (1,1) for the rendered image, z is the same as the screen space
- raster space: similar to NDC, but the range of (x,y) is from (0,0) to (xRes, yRes)

Screen space



Projective camera models

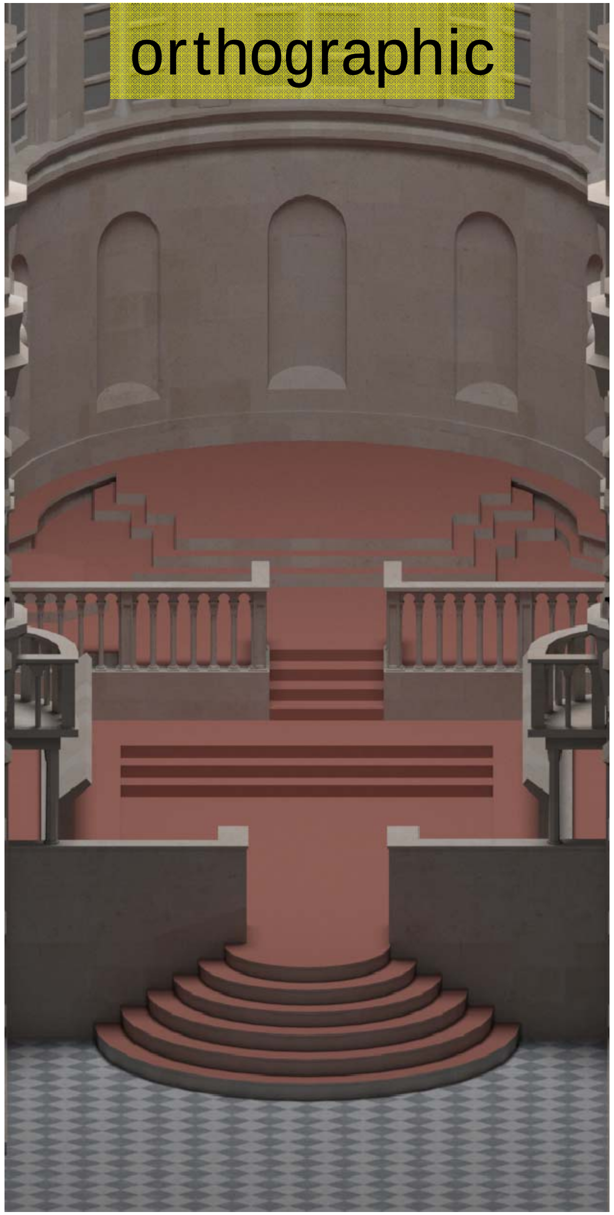
- Transform a 3D scene coordinate to a 2D image coordinate by a 4x4 projective matrix

```
class ProjectiveCamera : public Camera {
public: camera to screen projection (3D to 2D)
  ProjectiveCamera(Transform &world2cam,
    Transform &proj, float Screen[4],
    float hither, float yon, float sopen,
    float sclose, float lensr, float focald,
    Film *film);
protected:
  Transform CameraToScreen, WorldToScreen,
            RasterToCamera;
  Transform ScreenToRaster, RasterToScreen;
  float LensRadius, FocalDistance;
};
```

Projective camera models

```
ProjectiveCamera::ProjectiveCamera(...)
  :Camera(w2c, hither, yon, sopen, sclose, f) {
  ...
  CameraToScreen=proj;
  WorldToScreen=CameraToScreen*WorldToCamera;
  ScreenToRaster
   = Scale(float(film->xResolution),
           float(film->yResolution), 1.f)*
     Scale(1.f / (Screen[1] - Screen[0]),
           1.f / (Screen[2] - Screen[3]), 1.f)*
     Translate(Vector(-Screen[0],-Screen[3],0.f));
  RasterToScreen = ScreenToRaster.GetInverse();
  RasterToCamera =
      CameraToScreen.GetInverse() * RasterToScreen;
}
```

Projective camera models



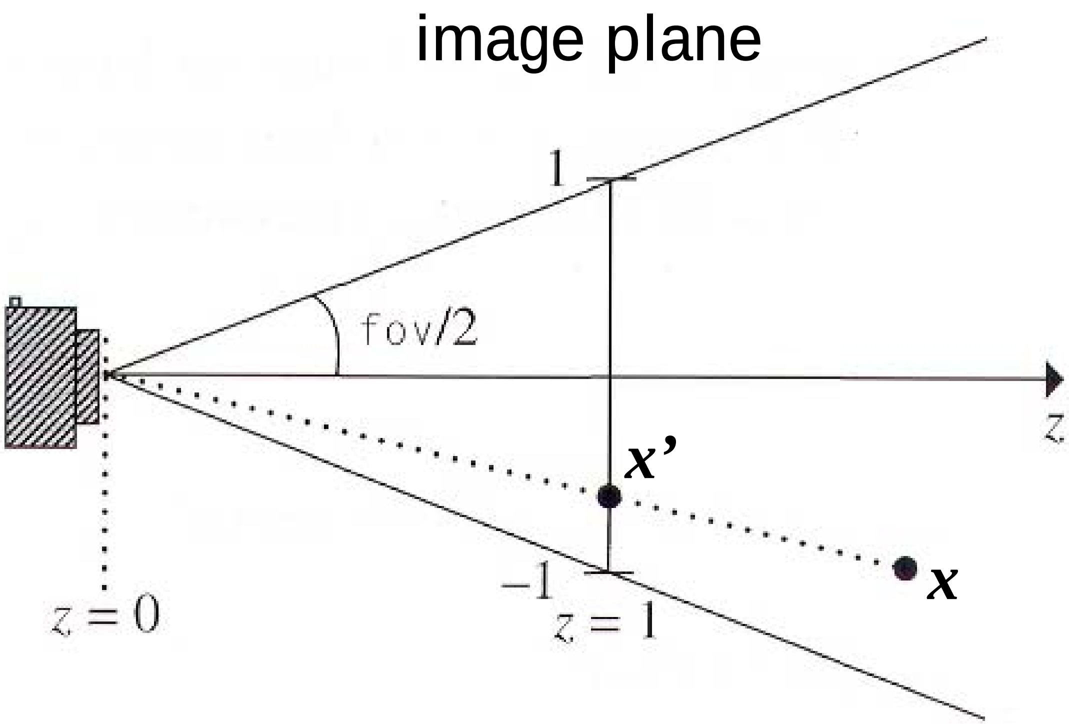
Orthographic camera

```
Transform Orthographic(float znear,
                       float zfar)
{
  return Scale(1.f, 1.f, 1.f/(zfar-znear))
    *Translate(Vector(0.f, 0.f, -znear));
}

OrthoCamera::OrthoCamera( ... )
  : ProjectiveCamera(world2cam,
      Orthographic(hither, yon),
      Screen, hither, yon, sopen, sclose,
      lensr, focald, f) {
}
```

OrthoCamera::GenerateRay

```
float OrthoCamera::GenerateRay
   (const Sample &sample, Ray *ray) const {
  Point Pras(sample.imageX,sample.imageY,0);
  Point Pcamera;
  RasterToCamera(Pras, &Pcamera);
  ray->o = Pcamera;
  ray->d = Vector(0,0,1);
  <Modify ray for depth of field>
  ray->mint = 0.;
  ray->maxt = ClipYon - ClipHither;
  ray->d = Normalize(ray->d);
  CameraToWorld(*ray, ray);
  return 1.f;
}
```

Perspective camera



$$x' = x / z$$

$$y' = y / z$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ ? & ? & ? & ? \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

f, n, z?

$$z' = \frac{f(z-n)}{z(f-n)}$$

Perspective camera

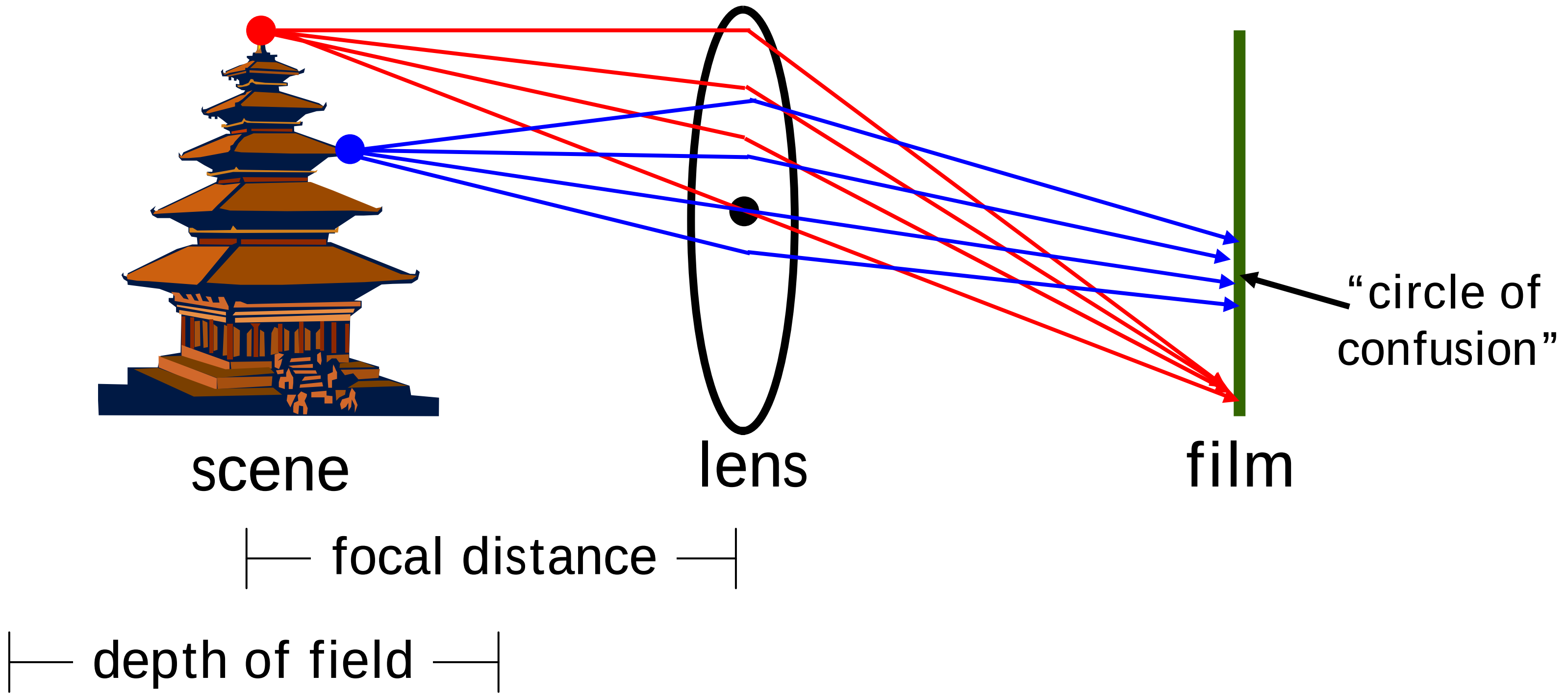
```
Transform Perspective(float fov,float n,float f)
{                                           near_z     far_z
  float inv_denom = 1.f/(f-n);
  Matrix4x4 *persp =
  new Matrix4x4(1, 0,        0,           0,
                0, 1,        0,           0,
                0, 0, f*inv_denom, -f*n*inv_denom,
                0, 0,        1,           0);

  float invTanAng= 1.f / tanf(Radians(fov)/2.f);
  return Scale(invTanAng, invTanAng, 1) *
         Transform(persp);
}
```

PerspectiveCamera::GenerateRay

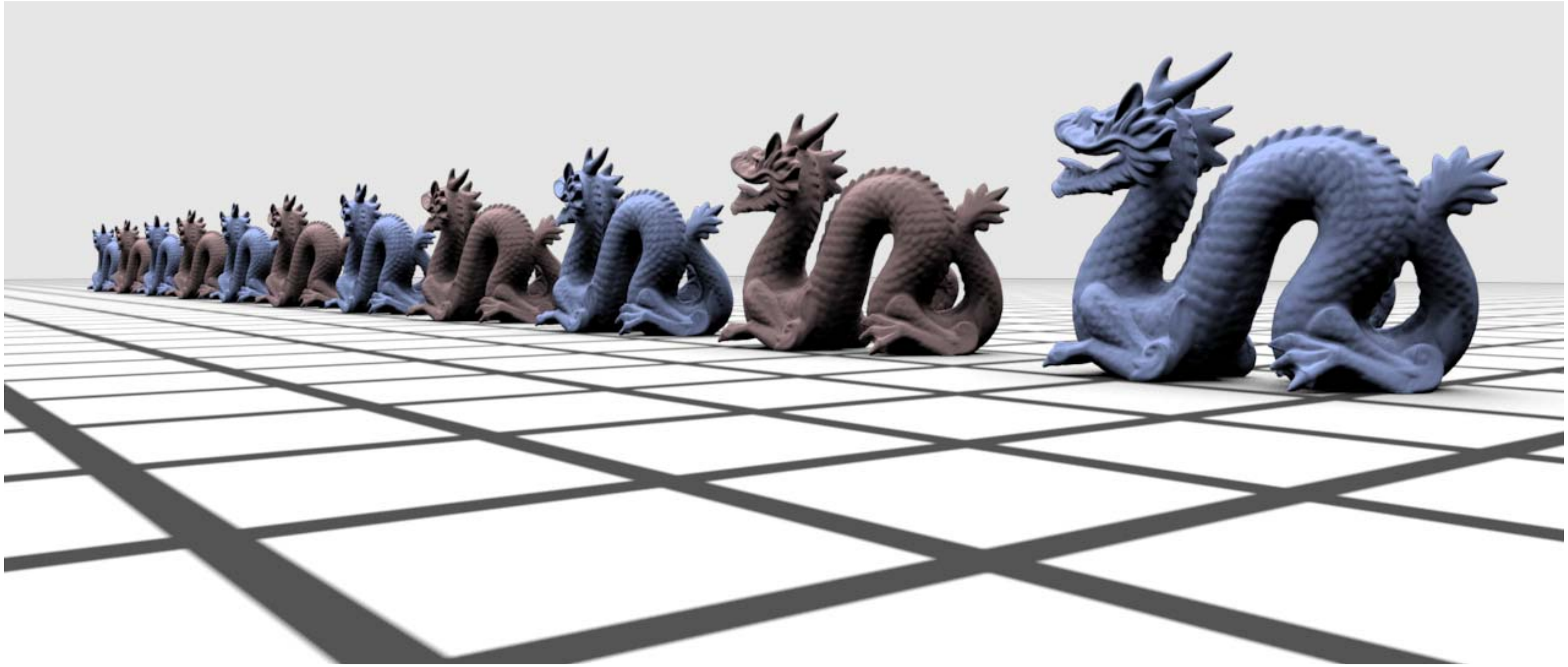
```
float PerspectiveCamera::GenerateRay
      (const Sample &sample, Ray *ray) const
{
  // Generate raster and camera samples
  Point Pras(sample.imageX, sample.imageY, 0);
  Point Pcamera;
  RasterToCamera(Pras, &Pcamera);
  ray->o = Pcamera;
  ray->d = Vector(Pcamera.x,Pcamera.y,Pcamera.z);
  <Modify ray for depth of field>
  ray->d = Normalize(ray->d);
  ray->mint = 0.;
  ray->maxt = (ClipYon-ClipHither)/ray->d.z;
  CameraToWorld(*ray, ray);
  return 1.f;
}
```

Depth of field

- Circle of confusion $\frac{1}{d_o} + \frac{1}{d_i} = \frac{1}{f}$
- Depth of field: the range of distances from the lens at which objects appear in focus (circle of confusion roughly smaller than a pixel)

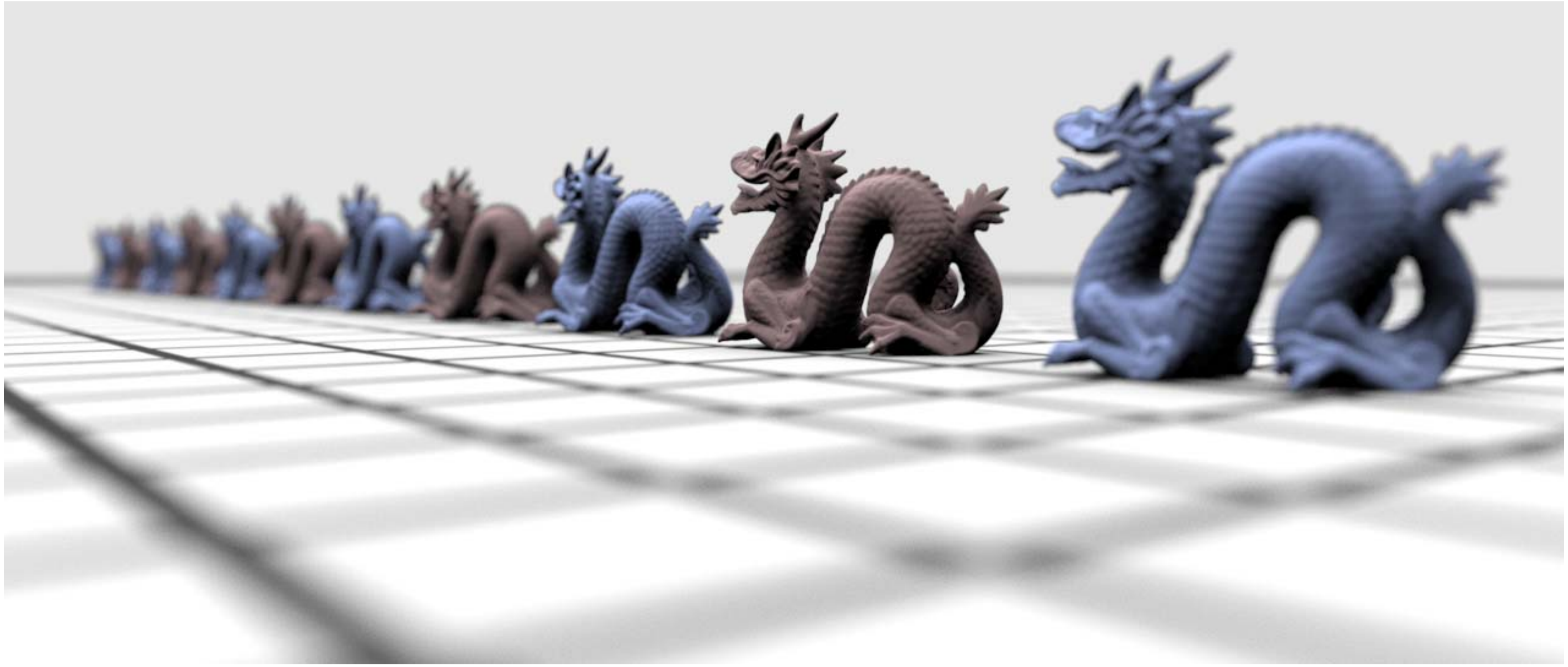


Depth of field



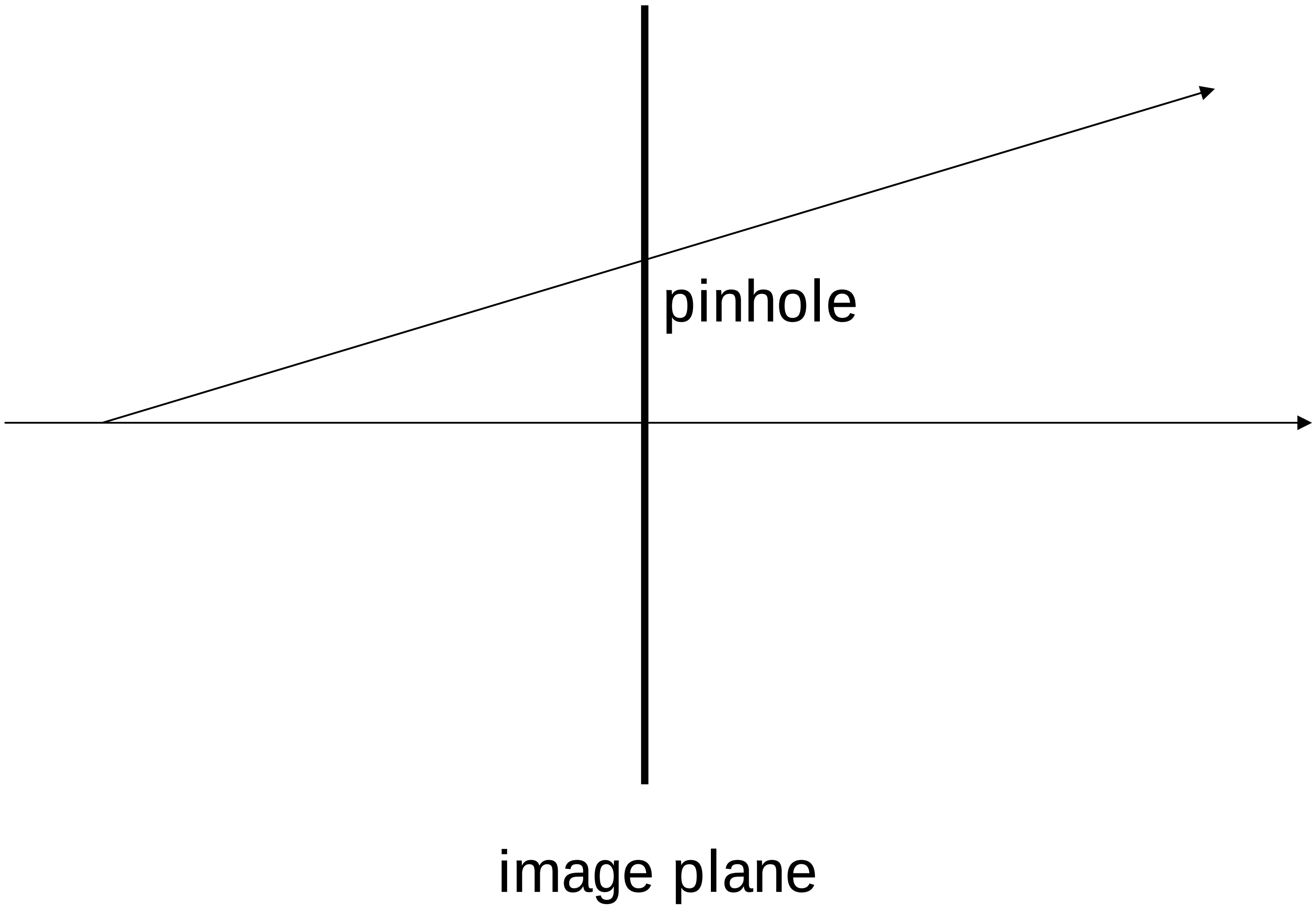
without depth of field

Depth of field

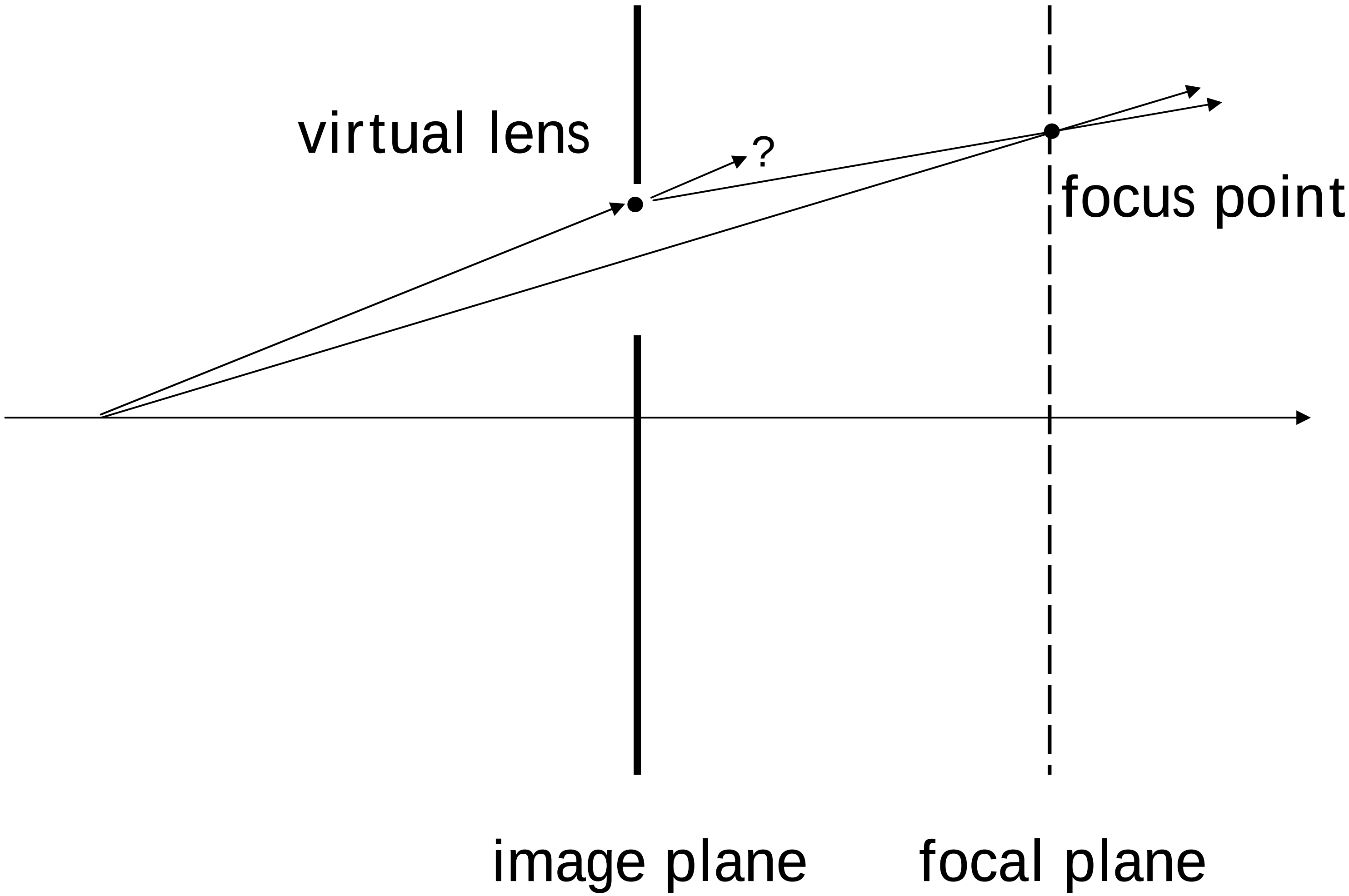


with depth of field

Sample the lens



Sample the lens



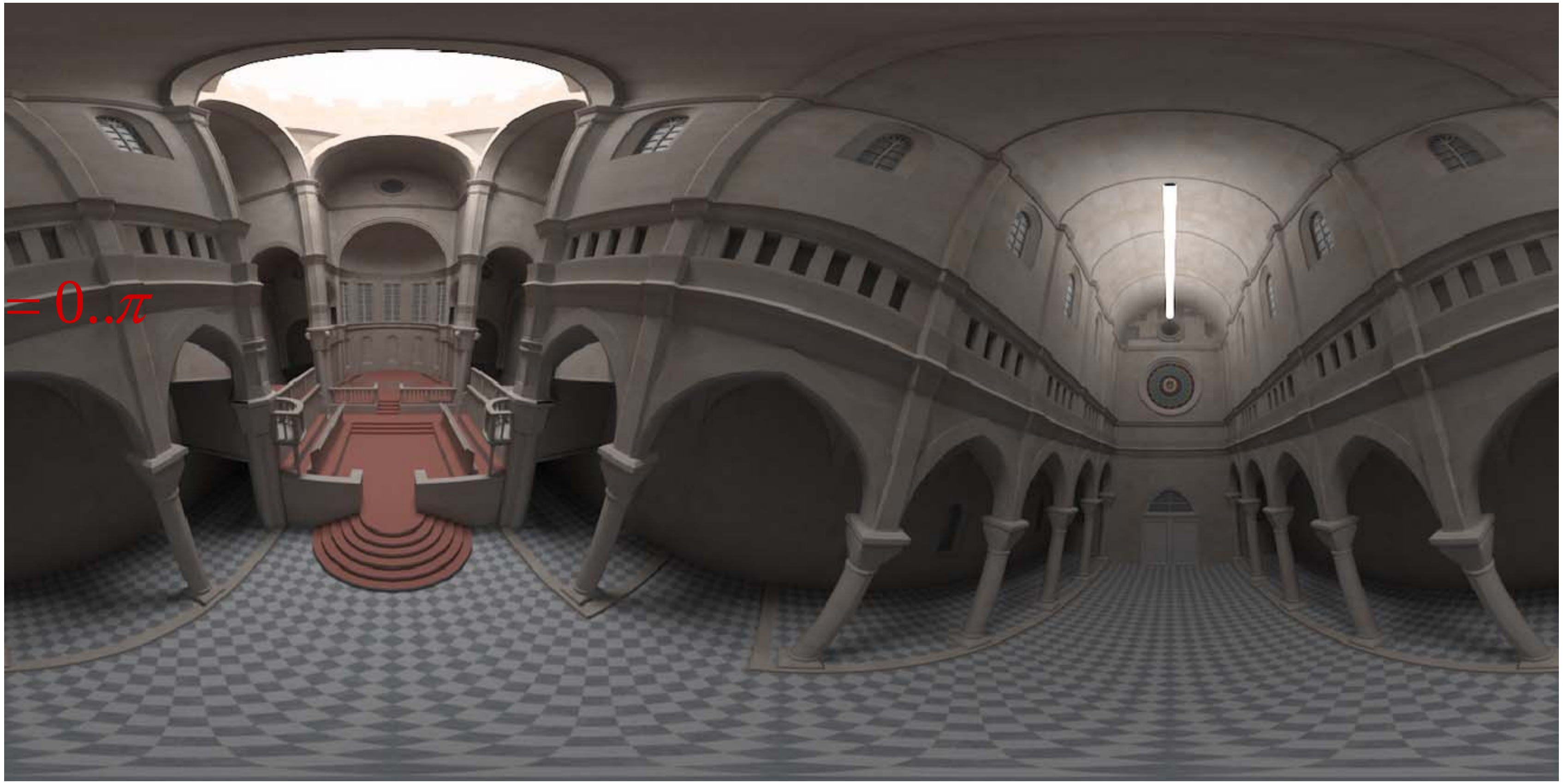
In GenerateRay(…)

```
if (LensRadius > 0.) {
  // Sample point on lens
   float lensU, lensV;
   ConcentricSampleDisk(sample.lensU, sample.lensV,
                        &lensU, &lensV);
   lensU *= LensRadius;
   lensV *= LensRadius;
   // Compute point on plane of focus
  float ft = (FocalDistance - ClipHither) / ray->d.z;
   Point Pfocus = (*ray)(ft);
   // Update ray for effect of lens
   ray->o.x += lensU;
   ray->o.y += lensV;
   ray->d = Pfocus - ray->o;
}
```

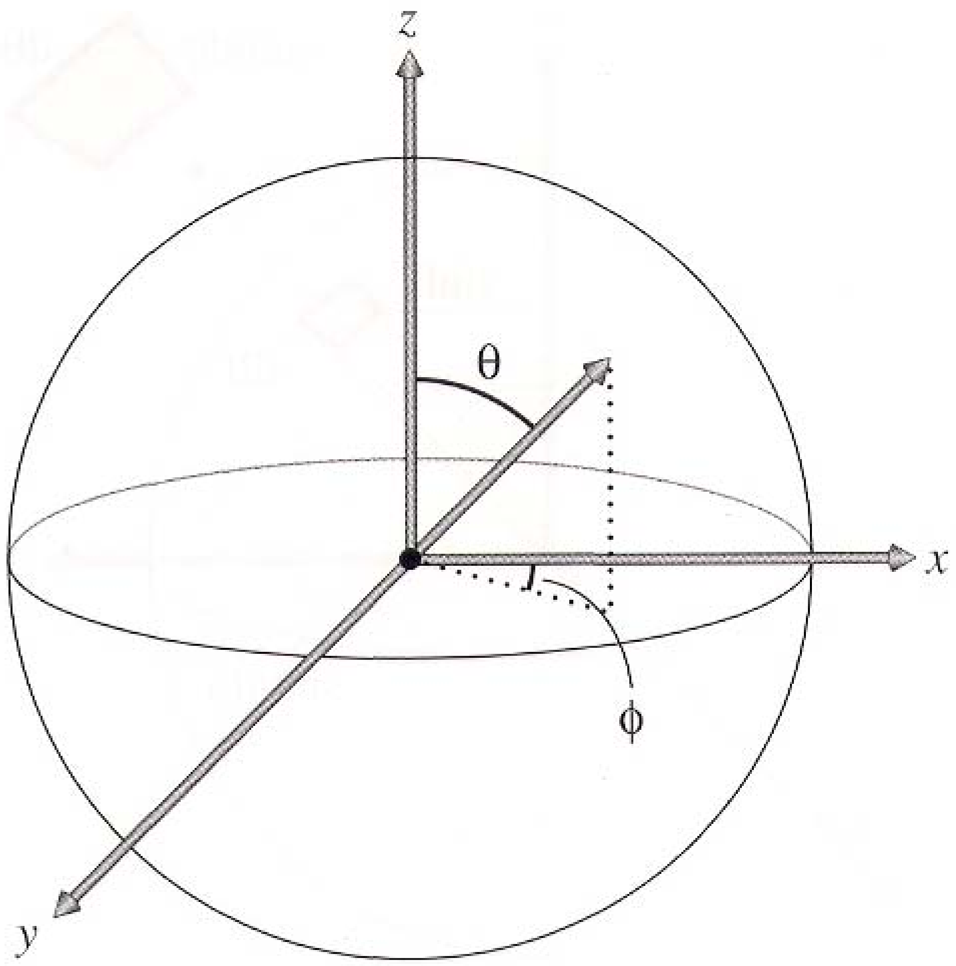
Environment camera

$\phi = 0..2\pi$

$\theta = 0..\pi$



Environment camera



$x = \sin\theta\cos\phi$
$y = \sin\theta\sin\phi$
$z = \cos\theta$

EnvironmentCamera

```
EnvironmentCamera::
  EnvironmentCamera(const Transform &world2cam,
                    float hither, float yon,
                    float sopen, float sclose,
                    Film *film)
  : Camera(world2cam, hither, yon,
           sopen, sclose, film)
{
  rayOrigin = CameraToWorld(Point(0,0,0));
}
```

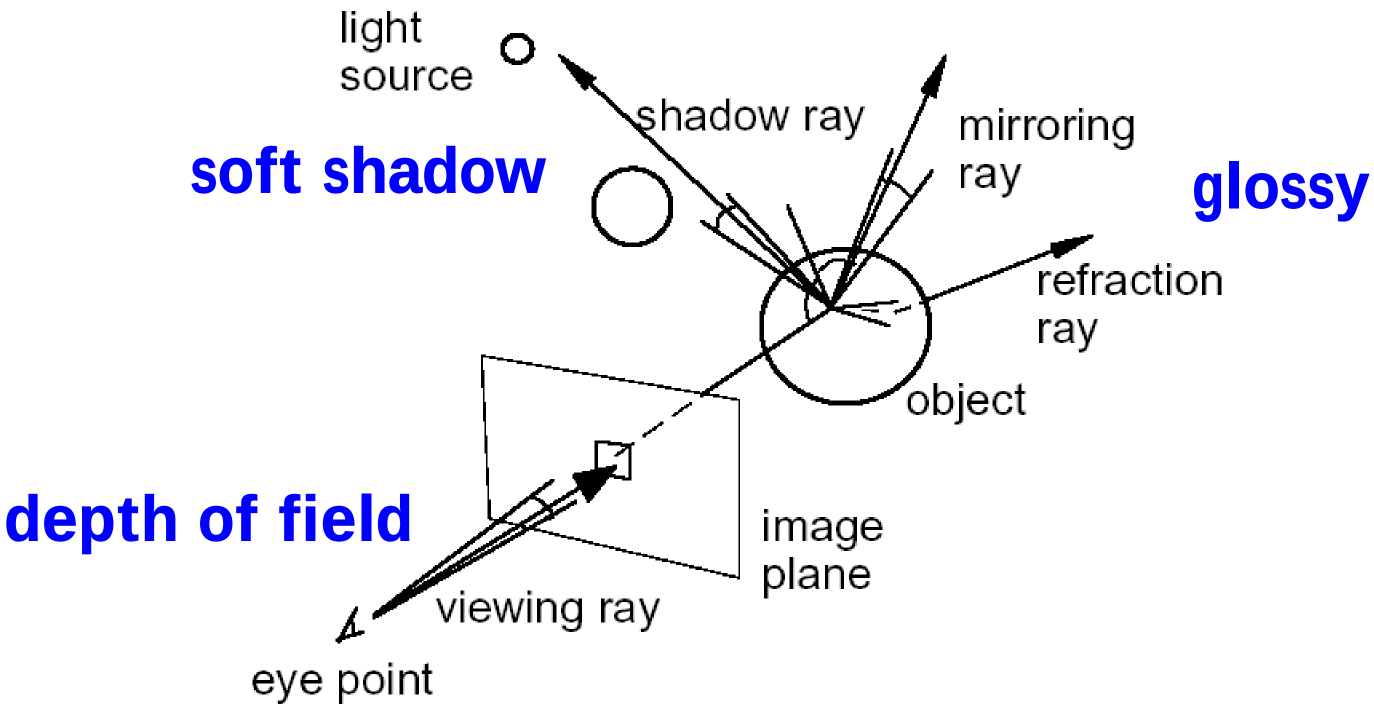
in world space

EnvironmentCamera::GenerateRay

```
float EnvironmentCamera::GenerateRay
      (const Sample &sample, Ray *ray) const
{
  ray->o = rayOrigin;
  float theta=M_PI*sample.imageY/film->yResolution;
  float phi=2*M_PI*sample.imageX/film->xResolution;
  Vector dir(sinf(theta)*cosf(phi), cosf(theta),
             sinf(theta)*sinf(phi));
  CameraToWorld(dir, &ray->d);
  ray->mint = ClipHither;
  ray->maxt = ClipYon;
  return 1.f;
}
```

Distributed ray tracing

- *SIGGRAPH 1984, by Robert L. Cook, Thomas Porter and Loren Carpenter from LucasFilm.*
- Apply distribution-based sampling to many parts of the ray-tracing algorithm.



Distributed ray tracing

Gloss/Translucency

- Perturb directions reflection/transmission, with distribution based on angle from ideal ray

Depth of field

- Perturb eye position on lens

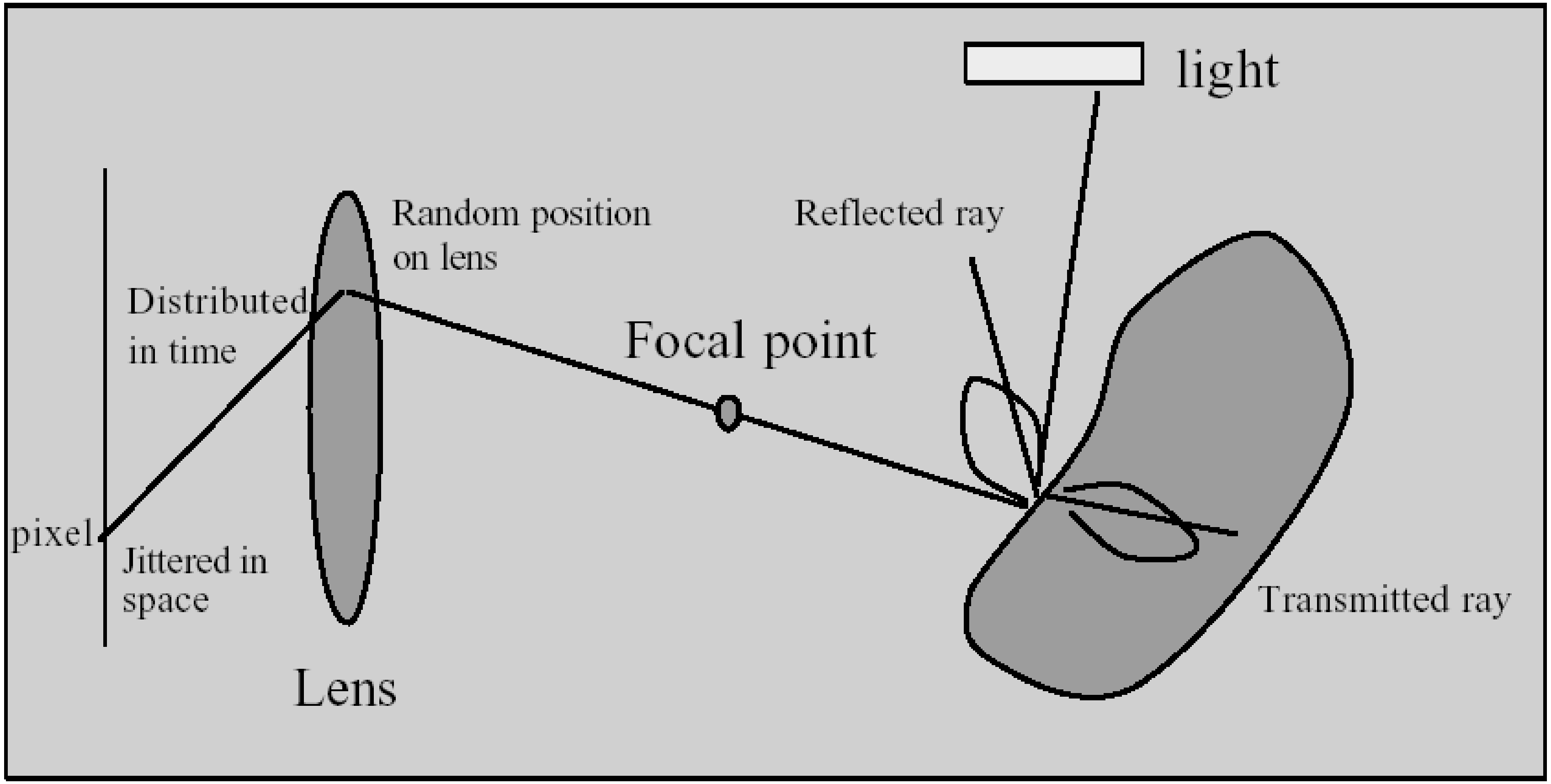
Soft shadow

- Perturb illumination rays across area light

Motion blur

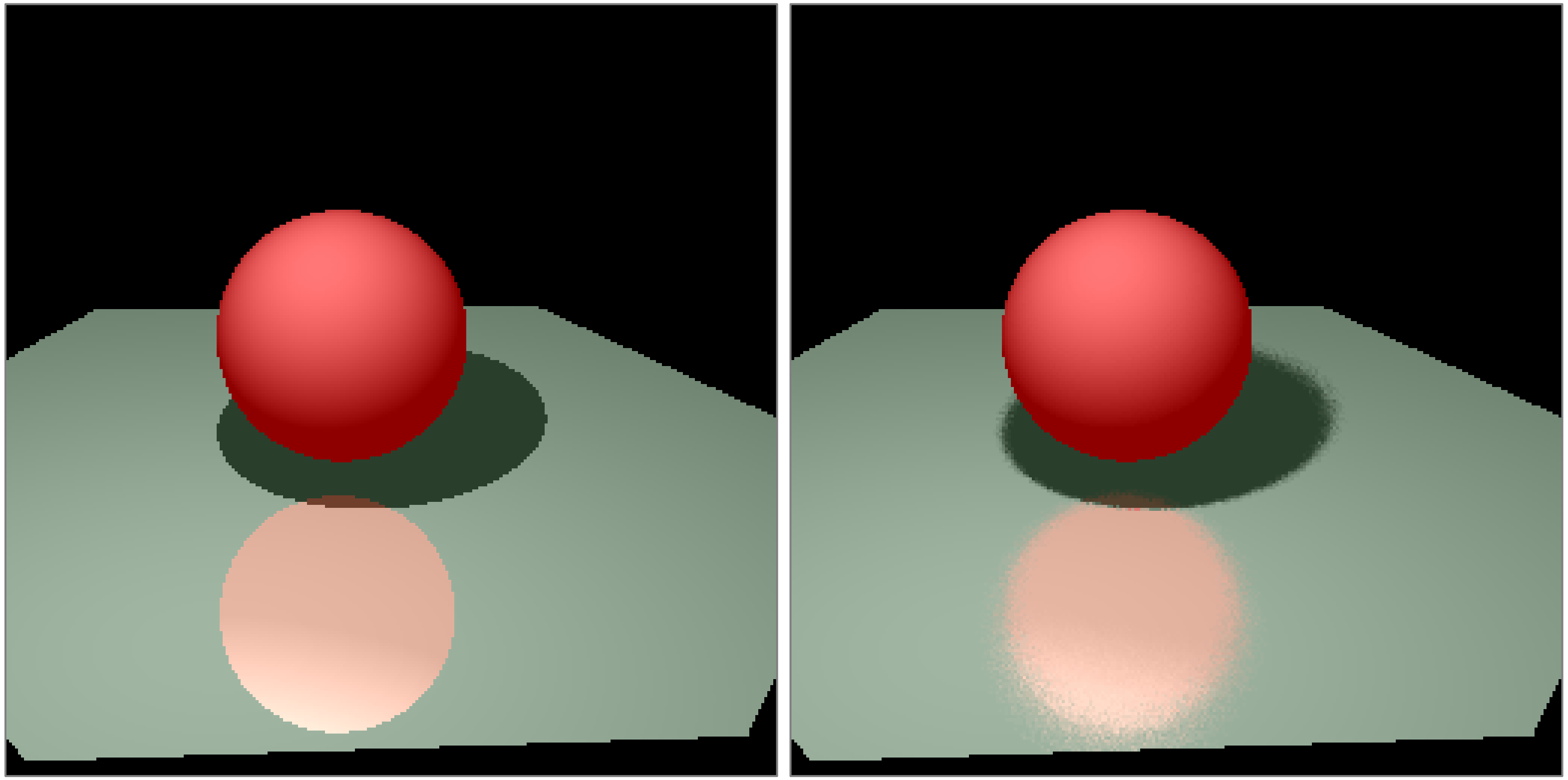
- Perturb eye ray samples in time

Distributed ray tracing



DRT: Gloss/Translucency

- Blurry reflections and refractions are produced by randomly perturbing the reflection and refraction rays from their "true" directions.

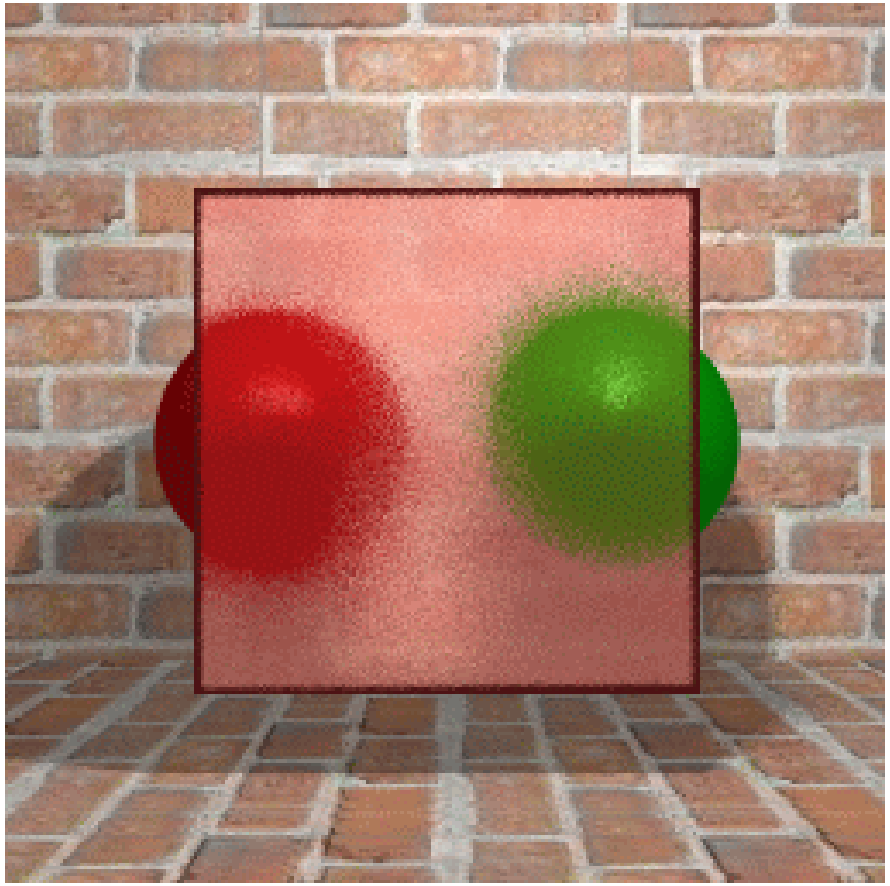


Glossy reflection

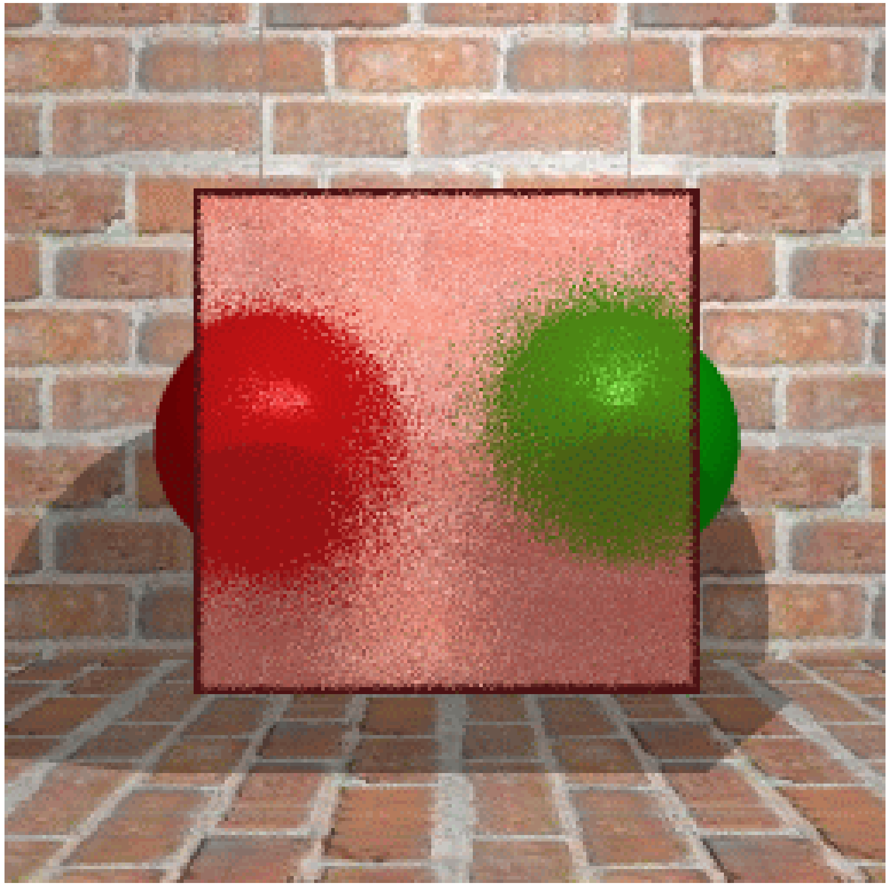
4 rays

64 rays

Translucency

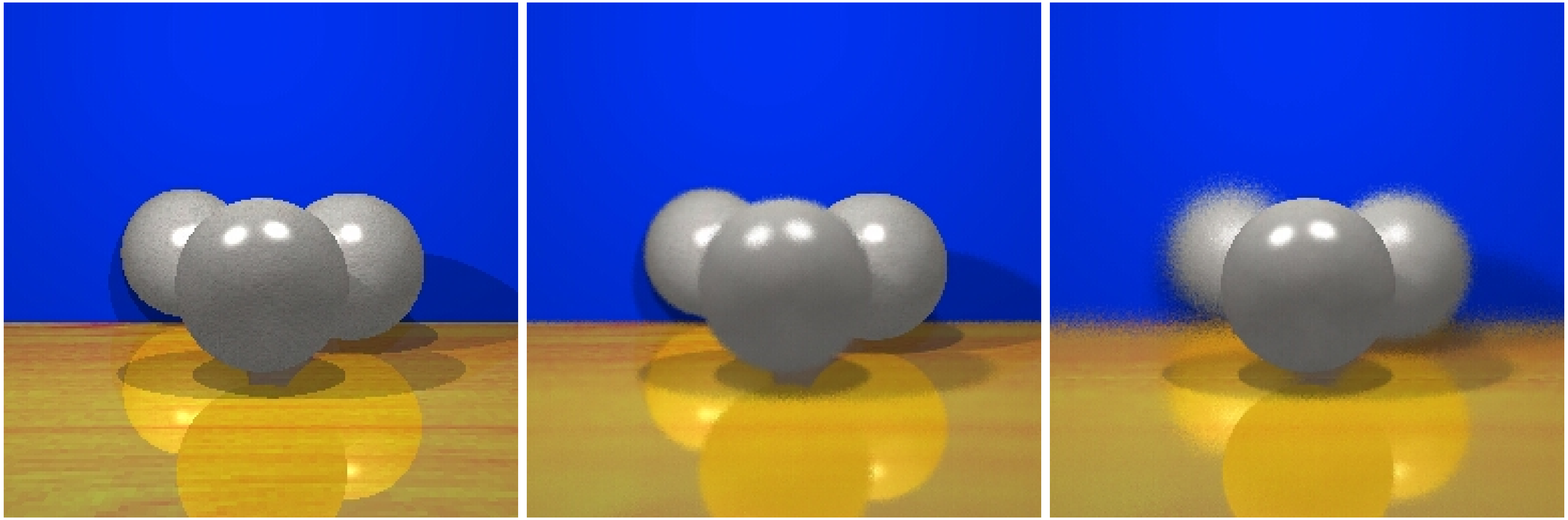


4 rays

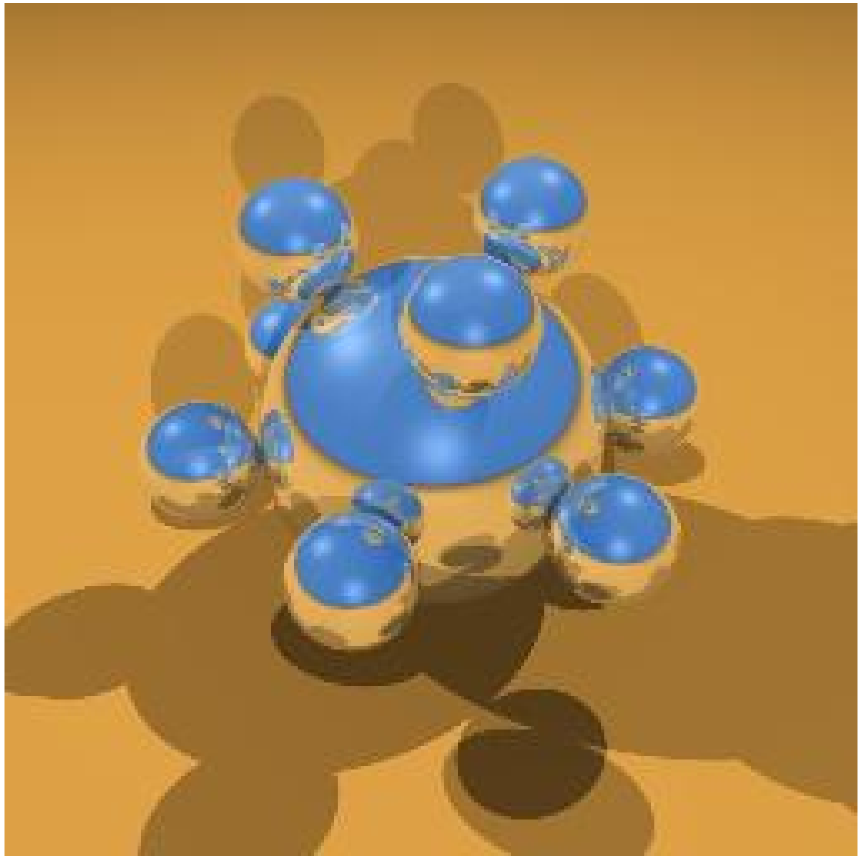


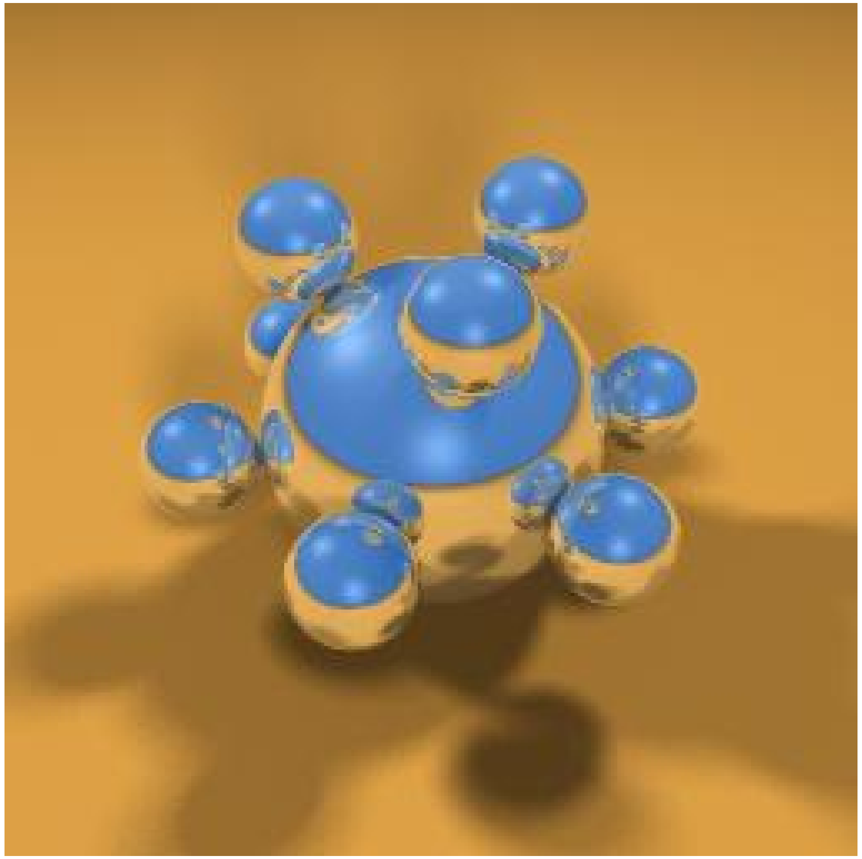
16 rays

Depth of field

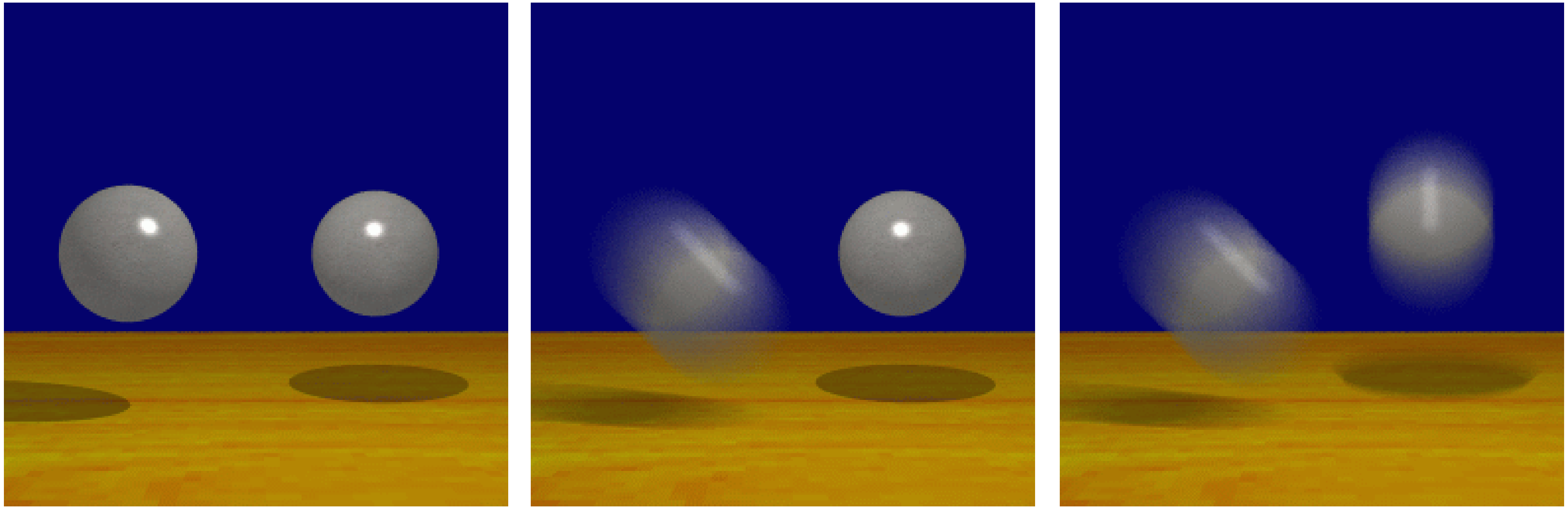


Soft shadows

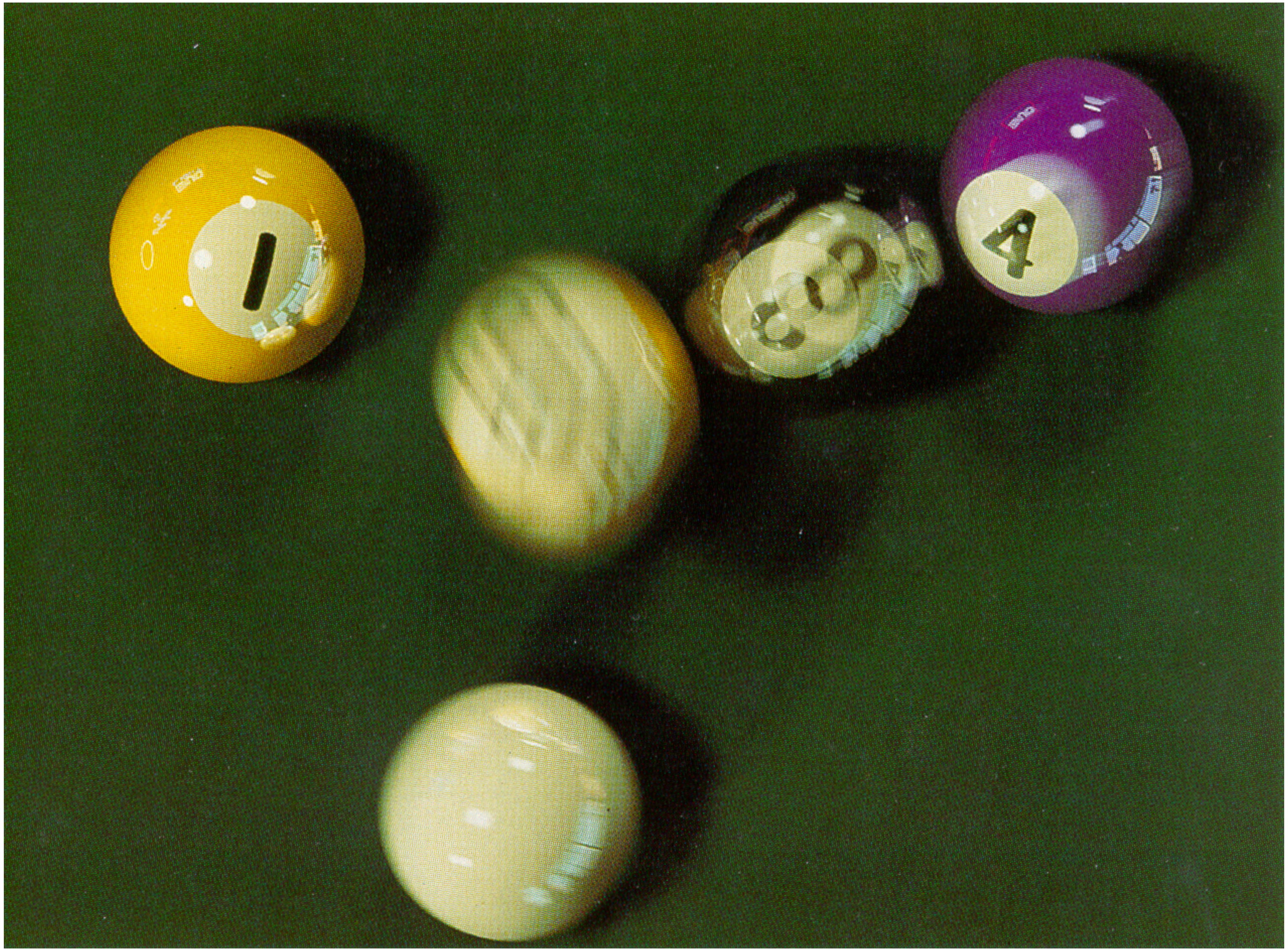




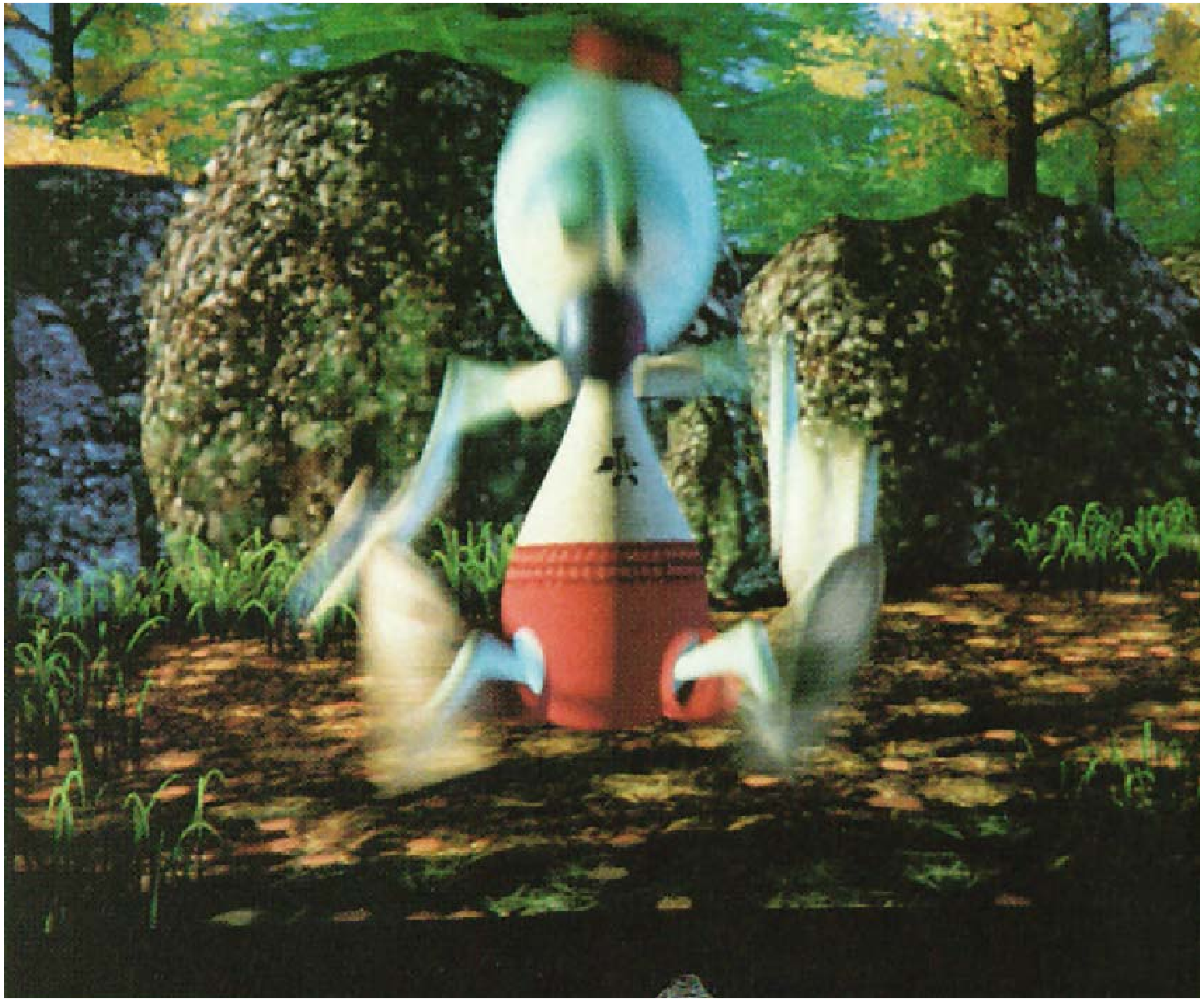
Motion blur



Results



Adventures of Andre & Wally B (1986)



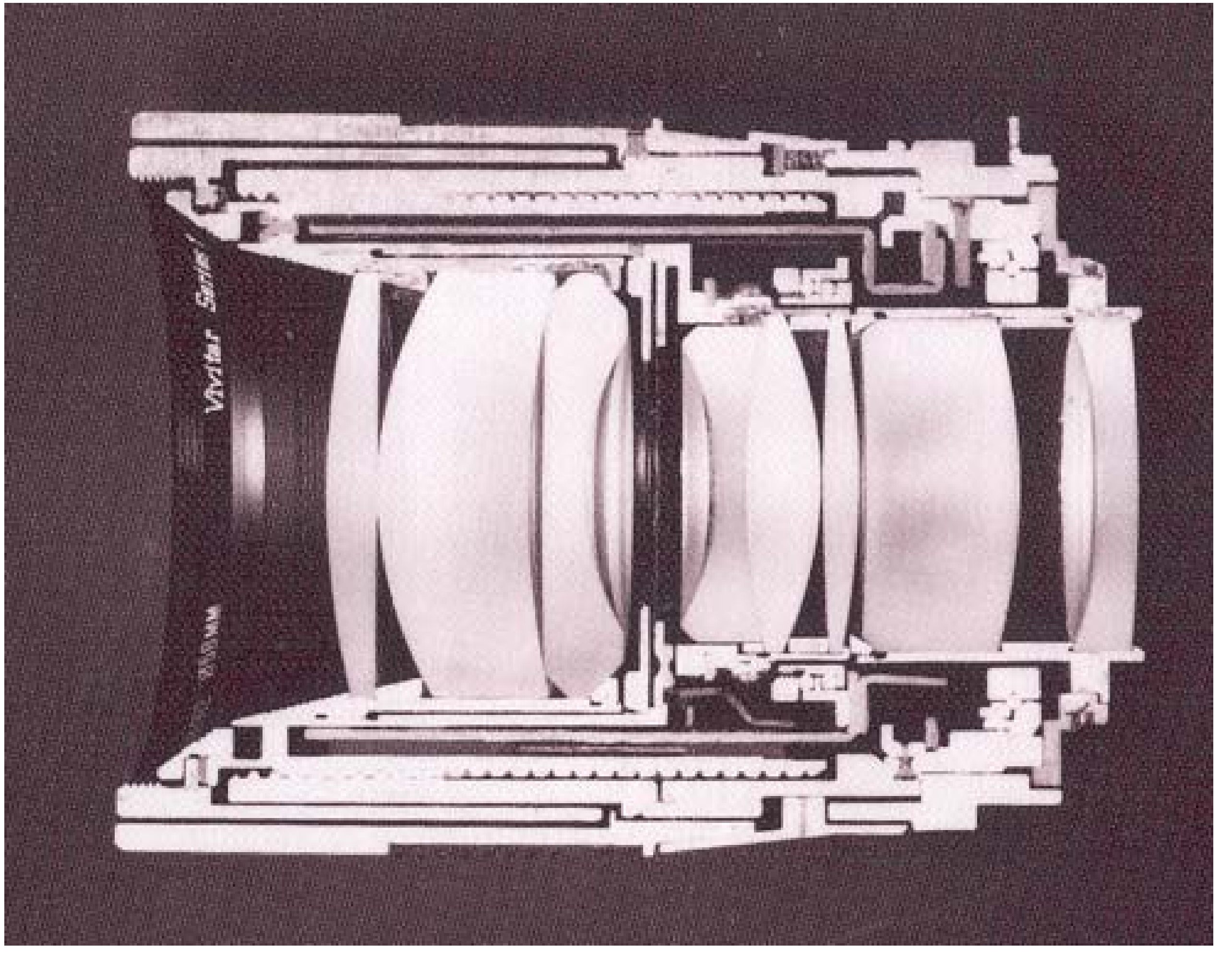
Realistic camera model

- Most camera models in graphics are not geometrically or radiometrically correct.
- Model a camera with a lens system and a film backplane. A lens system consists of a sequence of simple lens elements, stops and apertures.

Why a realistic camera model?

- Physically-based rendering. For more accurate comparison to empirical data.
- Seamlessly merge CGI and real scene, for example, VFX.
- For vision and scientific applications.
- The camera metaphor is familiar to most 3d graphics system users.

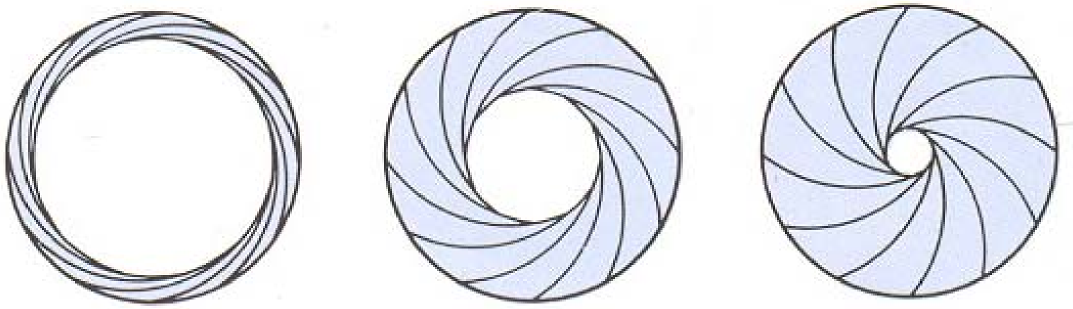
Real Lens



Cutaway section of a Vivitar Series 1 90mm f/2.5 lens
Cover photo, Kingslake, *Optics in Photography*

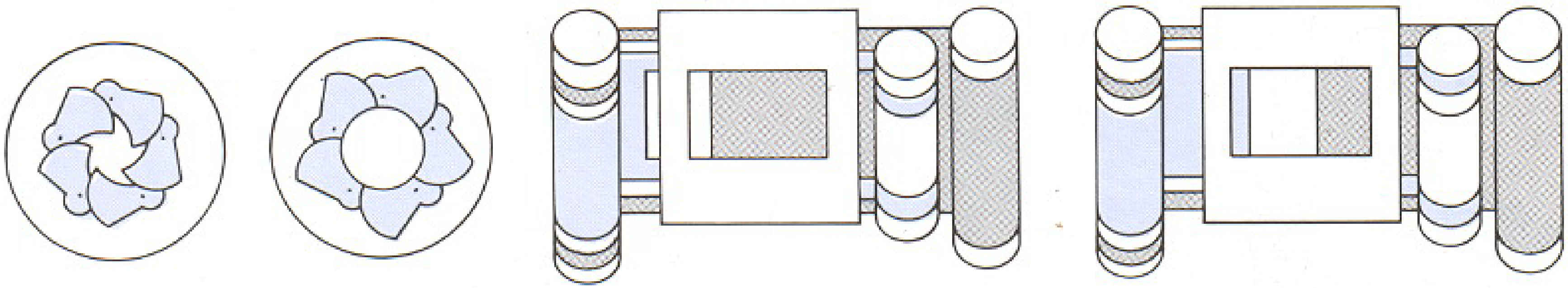
Exposure

- Two main parameters:
  - Aperture (in f stop)



Full aperture | Medium aperture | Stopped down

  - Shutter speed (in fraction of a second)

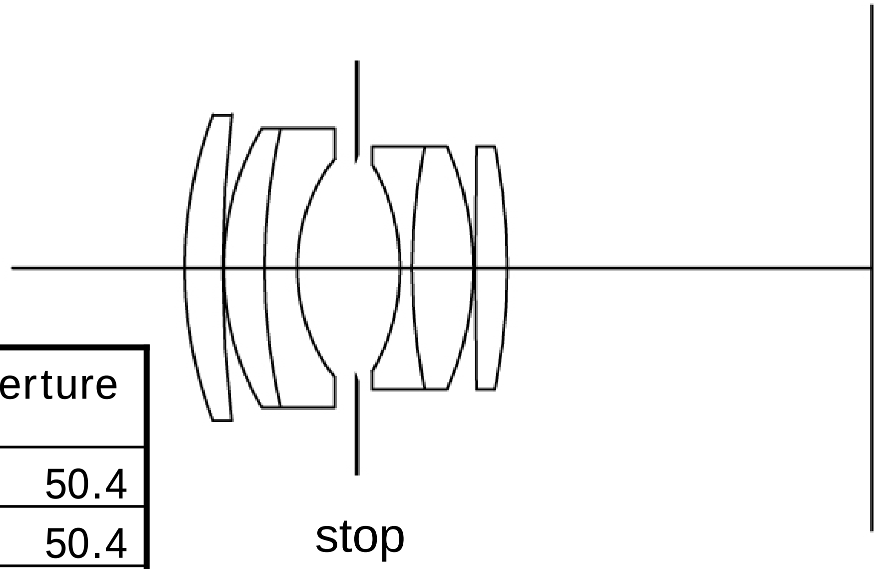


Blade (closing) | Blade (open) | Focal plane (closed) | Focal plane (open)

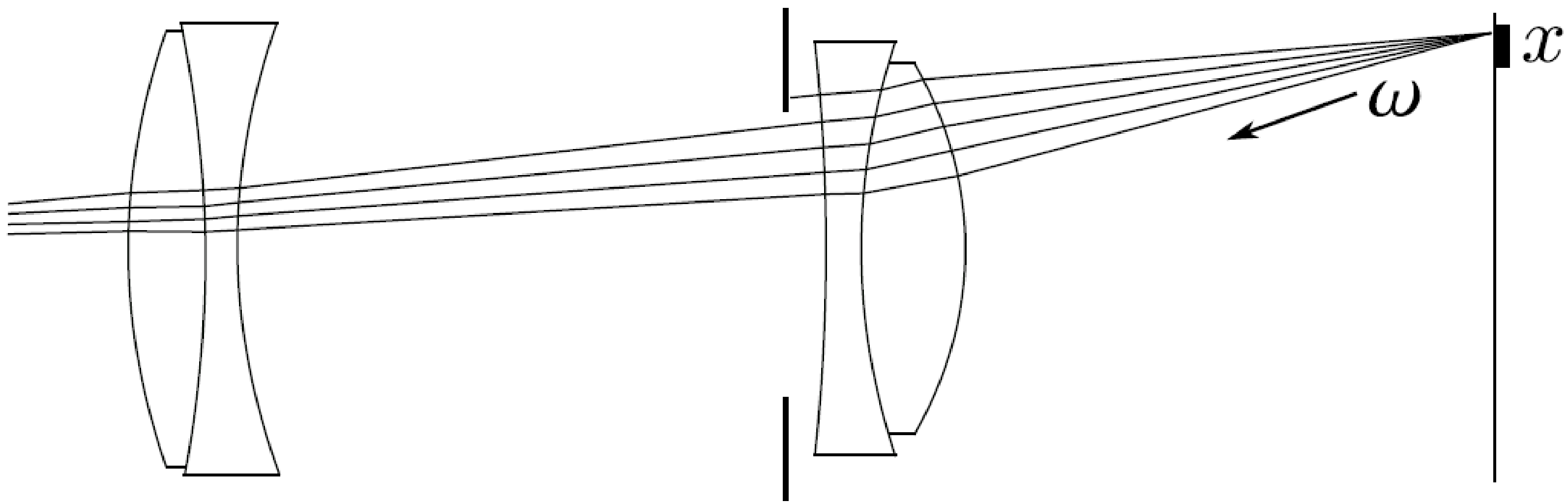
Double Gauss

**Data from W. Smith,
Modern Lens Design, p 312**

Radius (mm)	Thick (mm)	n_d	V-no	aperture
58.950	7.520	1.670	47.1	50.4
169.660	0.240			50.4
38.550	8.050	1.670	47.1	46.0
81.540	6.550	1.699	30.1	46.0
25.500	11.410			36.0
	9.000			34.2
-28.990	2.360	1.603	38.0	34.0
81.540	12.130	1.658	57.3	40.0
-40.770	0.380			40.0
874.130	6.440	1.717	48.0	40.0
-79.460	72.228			40.0



Measurement equation



$$R = \int\int\int\int L(T(x,\omega,\lambda);\lambda) S(x,t) P(x,\lambda) \cos\theta \; dx \; d\omega \; dt \; d\lambda$$

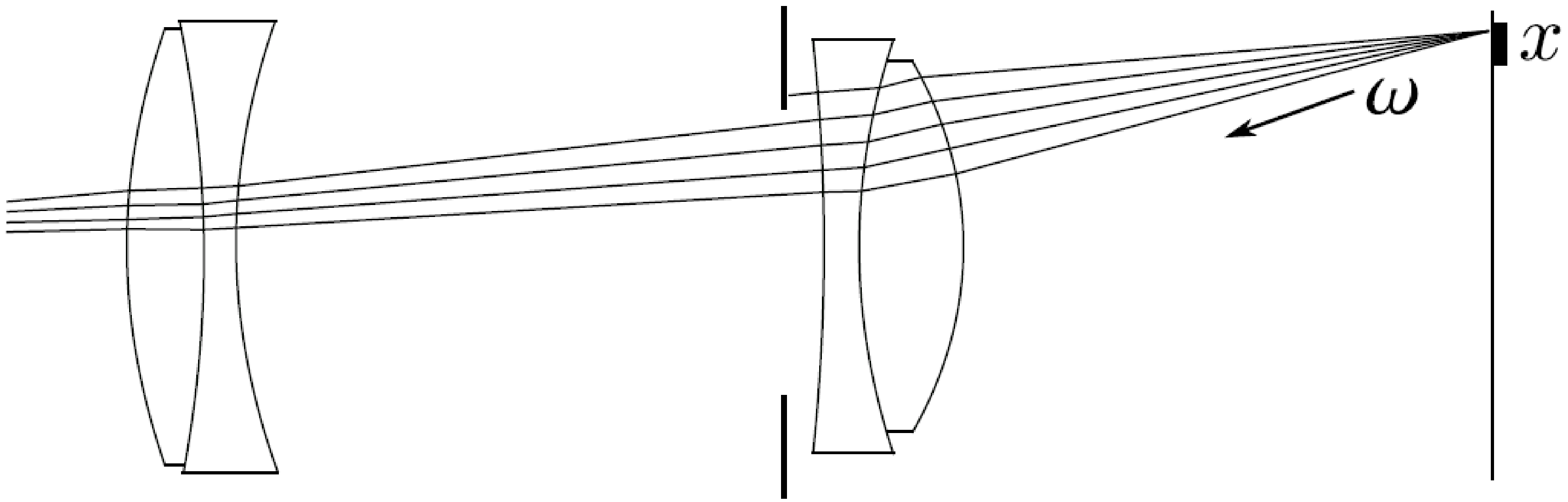
L: radiance

T: image to object space transformation

S: shutter function

P: sensor response characteristics

Measurement equation



$$R = \Delta t \cdot \int \int L(T(x,\omega)) \cos\theta \; dx \; d\omega$$

L: radiance T: image to object space transformation

Solving the integral

Problem: given a function f and domain Ω, how to calculate

$$\int_\Omega f(x)dx$$

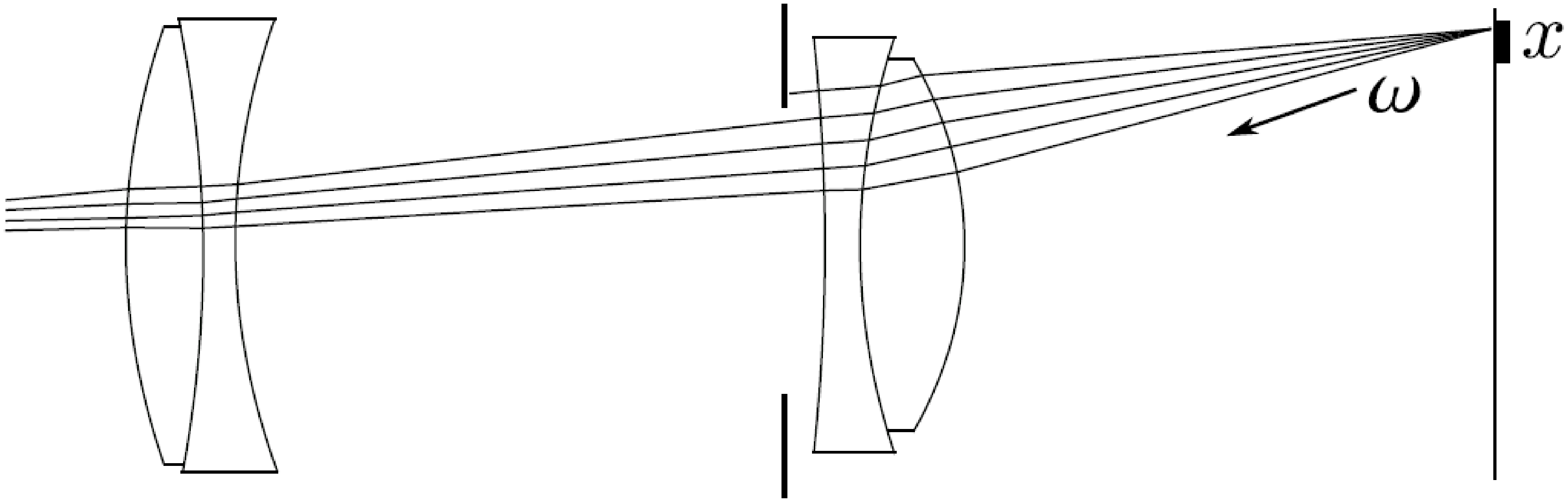
Solution: Monte Carlo method:

$$\int_\Omega f(x)dx \approx [\frac{1}{N}\sum_{i=1}^{N} f(x_i)] \cdot \int_\Omega dx$$

where $x_1, x_2, \ldots, x_N$ are uniform distributed random samples in Ω.

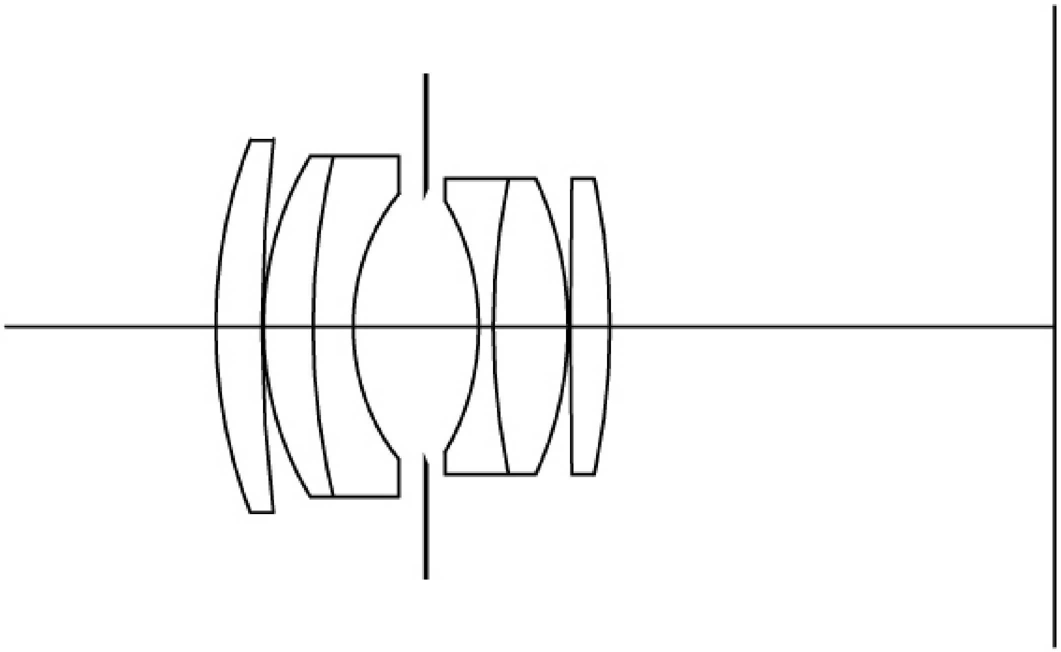
Algorithm

1. For each pixel on the image, generate some random samples x_i and ω_i uniformly.
2. For each x_i and ω_i, calculate $T(x_i, \omega_i)$.
3. Shoot the ray according to the result of $T(x_i, \omega_i)$ into the scene, and calculate the radiance.
4. Set the pixel value to the average of radiance.



Tracing rays through lens system

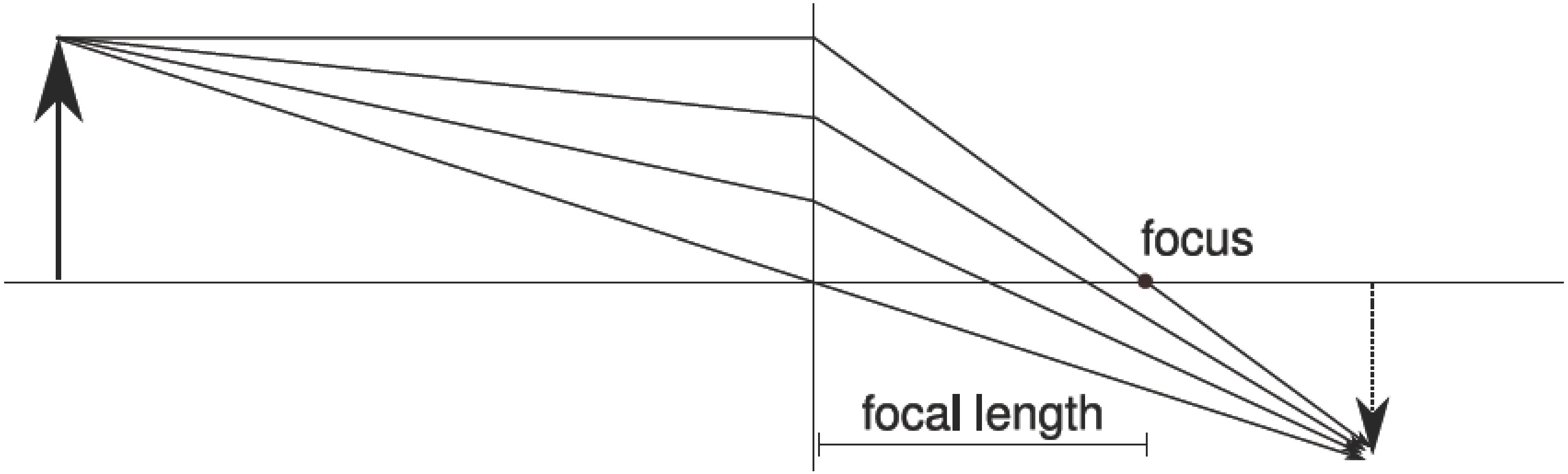
1. $R = Ray(x_i, \omega_i)$
2. Calculate the intersection point p for each lens element E_i from rear to front.
    1. Return zero if p is outside the aperture of E_i.
    2. Compute the new direction by Snell's law if the medium is different.

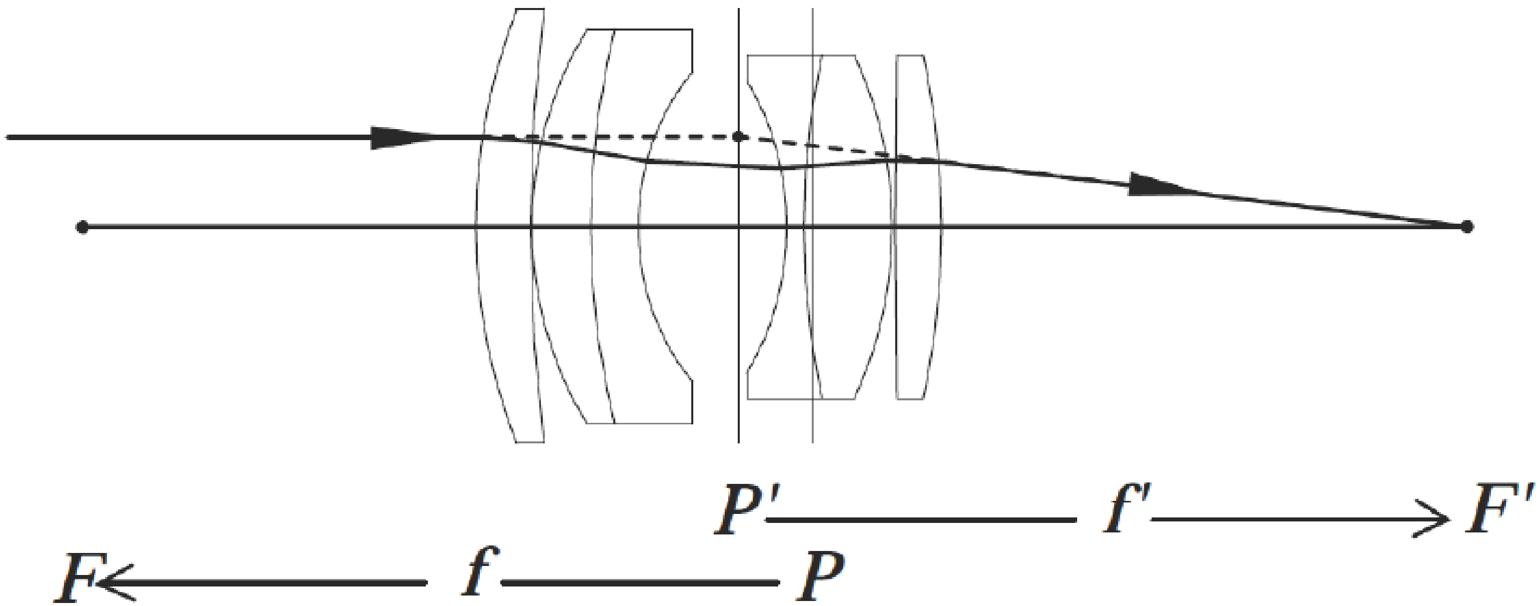


Ideal lens approximation

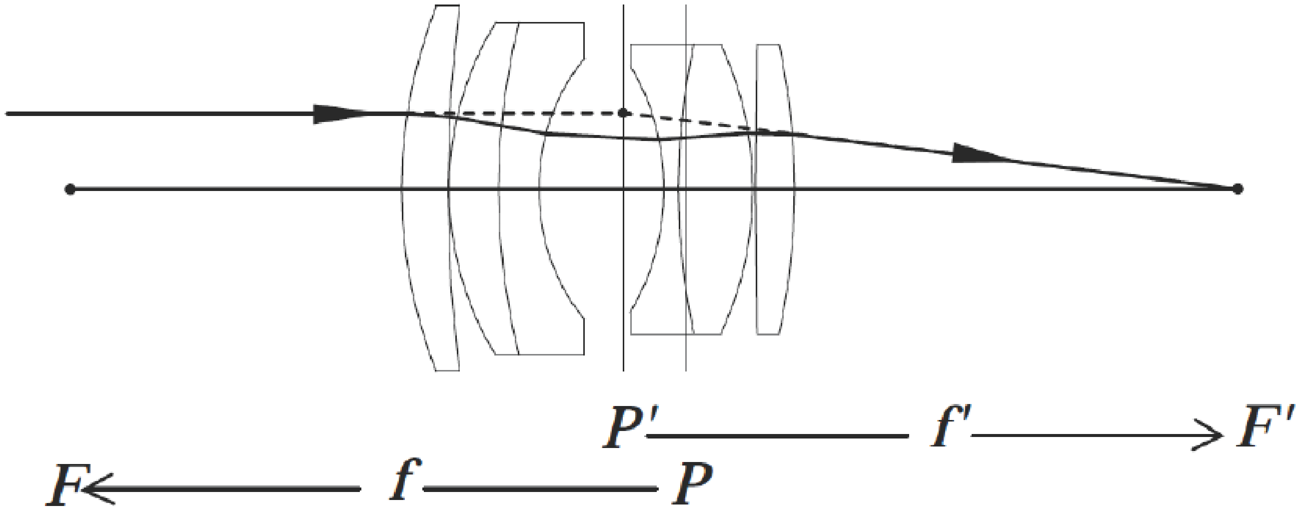
- In some situations we need an ideal lens approximation.
  - Ideal lens: each point in object space is imaged onto a single point in the image space.
  - All points on the **plane of focus** map onto the image plane.
- Thin lens approximation assumes that the thickness of lens is zero.
- Thick lens approximation has additional parameter of thickness.

Thin lens and thick lens





Finding thick lens approximation

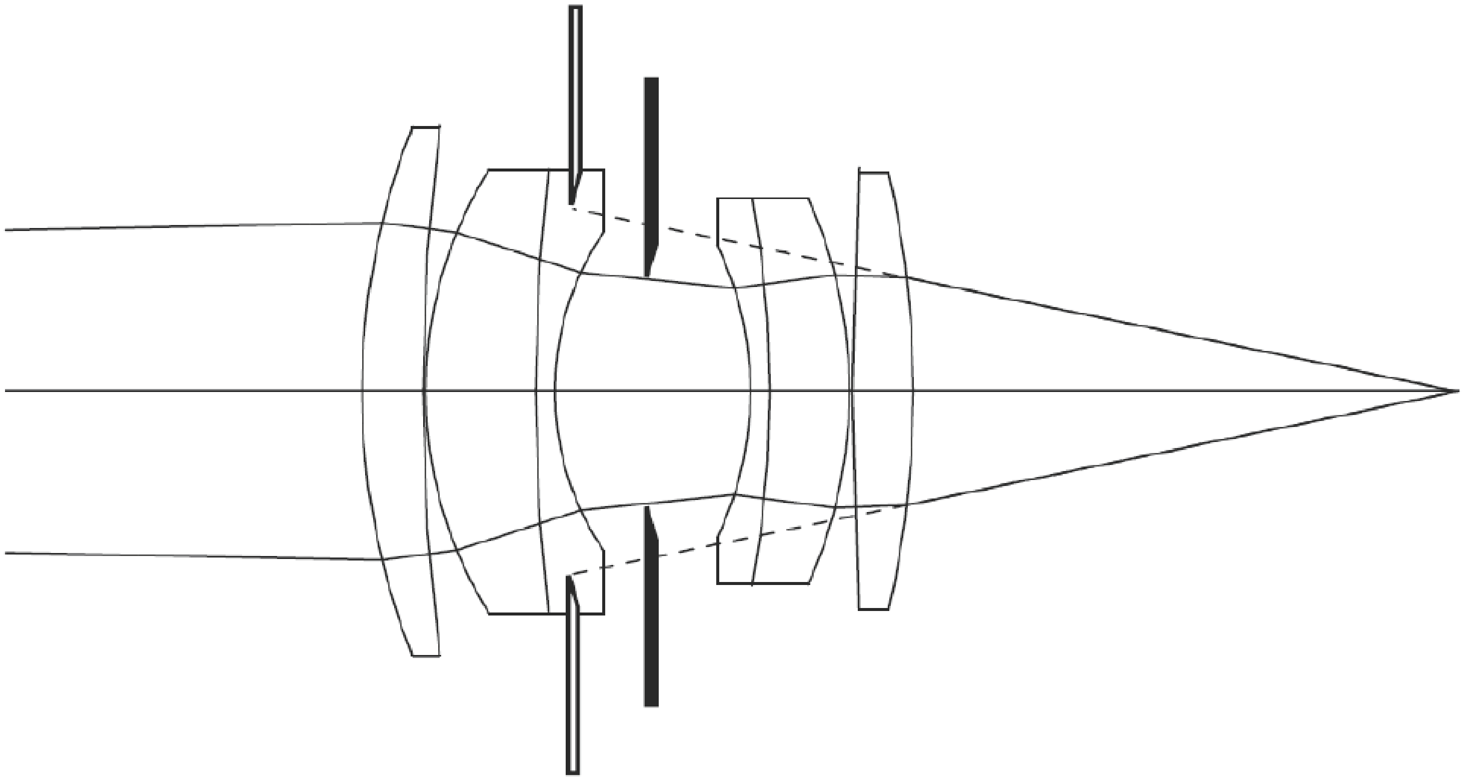


1. Shoot a ray parallel to the axis to find the focus.
2. Find the principal plane by intersecting the refracted ray and parallel one.
3. Find the secondary principal plane by tracing from another side.

Applications of thick lens approximation

- Faster way to calculate the transform
- Autofocus
- Calculate the exit pupil

Exit pupil

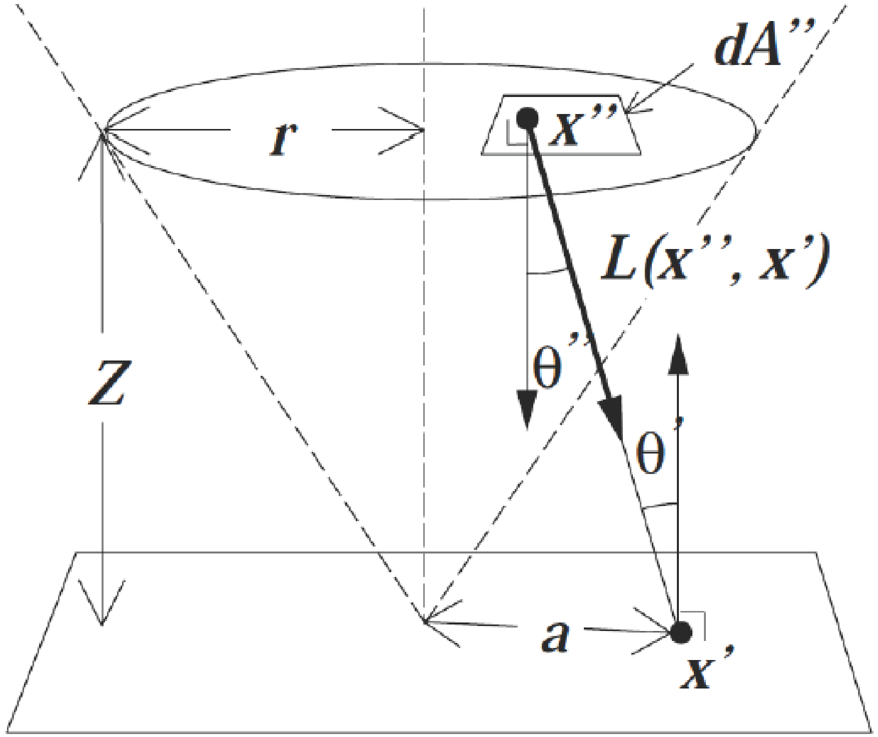


The exit pupil is the effective aperture stop in the image space which allows ray incindence.

Finding exit pupil

- Finding the exit pupil:
  1. For each aperture stop, calculate its image by thick lens approximation.
  2. Find the aperture stop whose image subtends the smallest solid angle.
- You may also use the aperture of the nearest lens as the exit pupil.

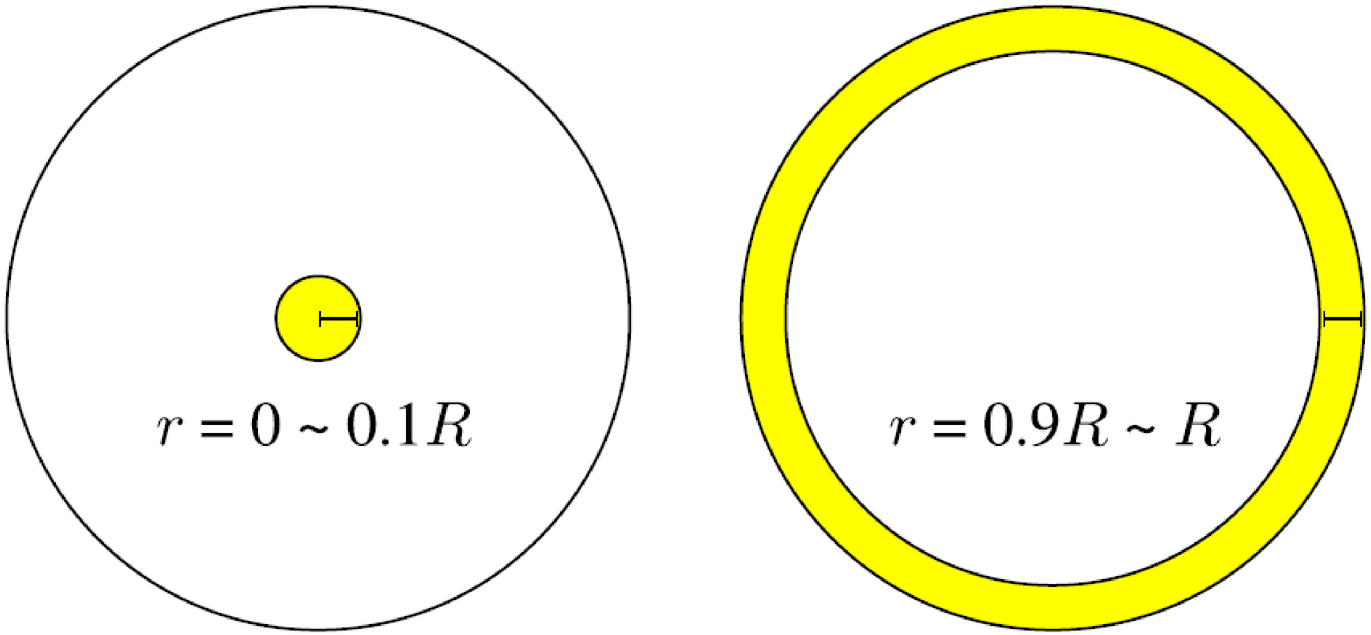
Integral over the exit pupil



$$E(x') = \frac{1}{Z^2} \int_{x'' \in D} L(x'', x') \cos^4 \theta' \, dA''$$

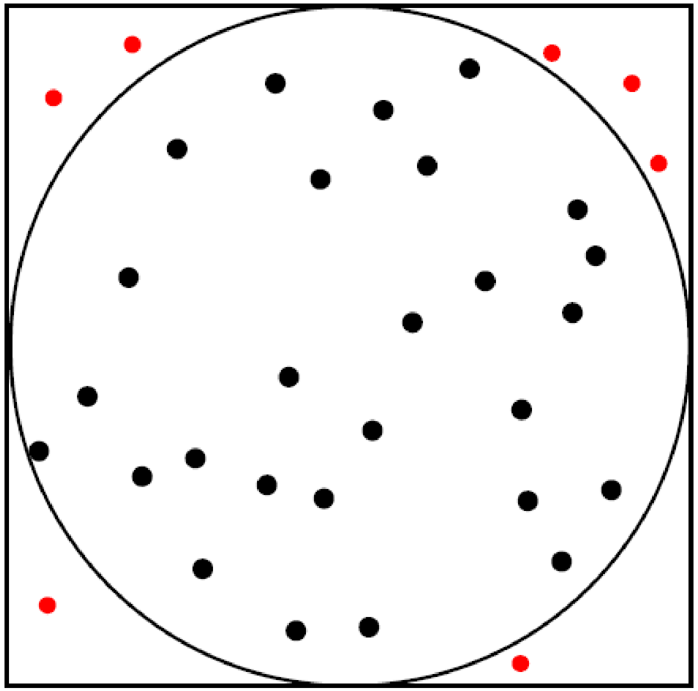
Sampling a disk uniformly

- Now we need to obtain random samples on a disk uniformly.
- How about uniformly sample r in $[0, R]$ and θ in $[0, 2\pi]$ and let $x = r\cos\theta, y = r\sin\theta$?
  - The result is not uniform due to coordinate transformation.



Rejection

1. Uniformly sample a point in the bounding square of the disk.
2. If the sample lies outside the disk, reject it and sample another one.



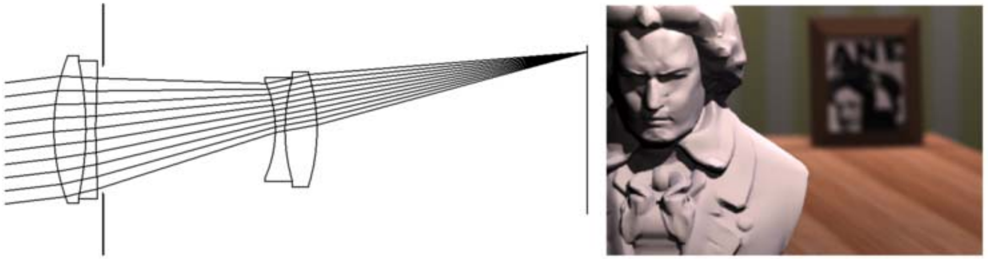
Another method

- Sample r and θ in a specific way so that the result is uniform after coordinate transformation.
- Let

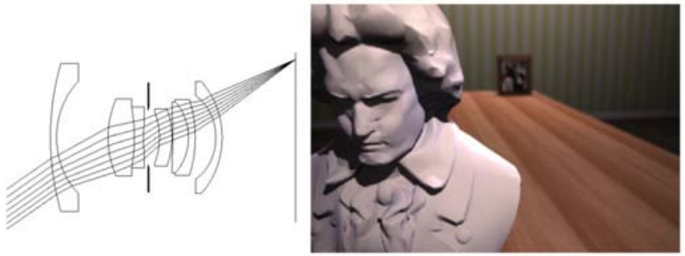
$$r = \sqrt{\xi_1},\ \theta = 2\pi\xi_2$$

  where ξ_1 and ξ_2 are random samples distributed in $[0, 1]$ uniforml uniformly.
- This produce uniform samples on a disk after coordinate transformation. We will prove it later in chapter 14 "Monte Carlo integration".

Ray Tracing Through Lenses



200 mm telephoto



35 mm wide-angle

50 mm double-gauss

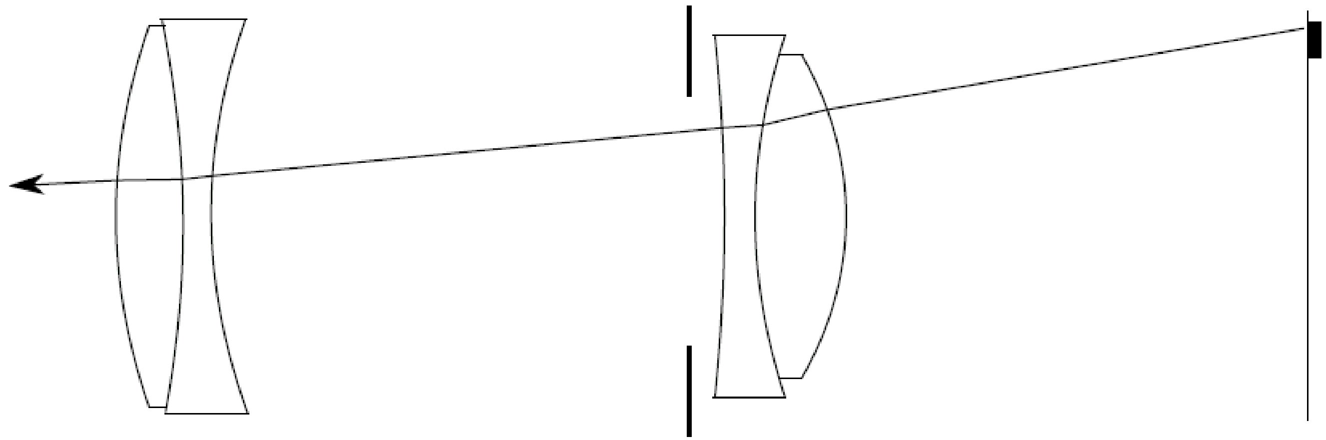
16 mm fisheye

From Kolb, Mitchell and Hanrahan (1995)

Assignment #2

- Write the “realistic” camera plugin for PBRT which implements the realistic camera model.
- The description of lens system will be provided.
- `GenerateRay(const Sample &sample, Ray *ray)`
  - PBRT generate rays by calling `GenerateRay()`, which is a virtual function of `Camera`.
  - PBRT will give you pixel location in `sample`.
  - You need to fill the content of `ray` and return a value for its weight.

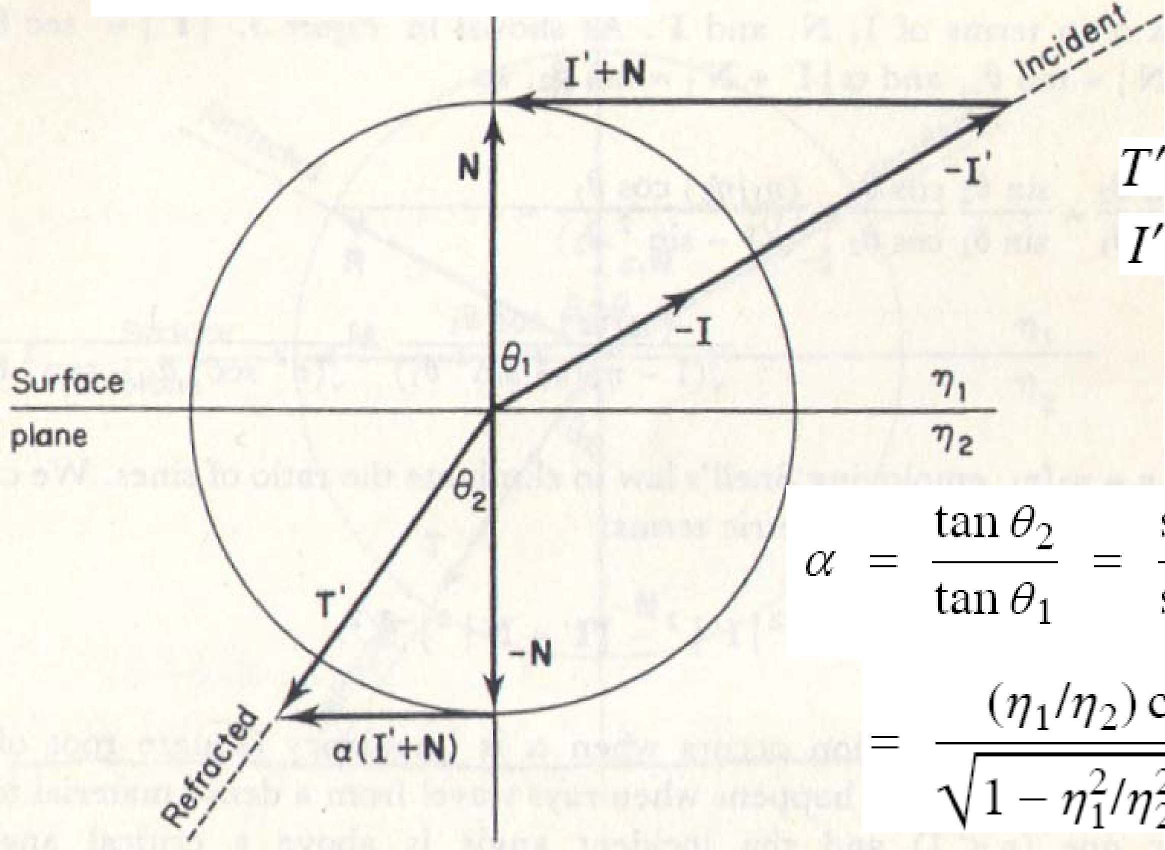
Assignment #2



1. Sample a point on the exit pupil uniformly.
   - Hint: `sample.lensU` and `sample.lensV` are two random samples distributed in $[0,1]$ uniformly.
2. Trace this ray through the lens system. You can return zero if this ray is blocked by an aperture stop.
3. Fill `ray` with the result and return $\frac{\cos^4\theta'}{Z^2}$ as its weight.

Whitted's method

$$\eta_1 \sin\theta_1 = \eta_2 \sin\theta_2$$



$$T' = \alpha(I' + N) - N \text{ for some } \alpha$$

$$I' = I/(-I \cdot N)$$

$$|I' + N| = \tan\theta_1$$

$$\alpha|I' + N| = \tan\theta_2$$

$$\alpha = \frac{\tan\theta_2}{\tan\theta_1} = \frac{\sin\theta_2}{\sin\theta_1}\frac{\cos\theta_1}{\cos\theta_2} = \frac{(\eta_1/\eta_2)\cos\theta_1}{\sqrt{1-\sin^2\theta_2}}$$

$$= \frac{(\eta_1/\eta_2)\cos\theta_1}{\sqrt{1-\eta_1^2/\eta_2^2\sin^2\theta_1}} = \frac{1}{\sqrt{n^2\sec^2\theta_1 - \tan^2\theta_1}}$$

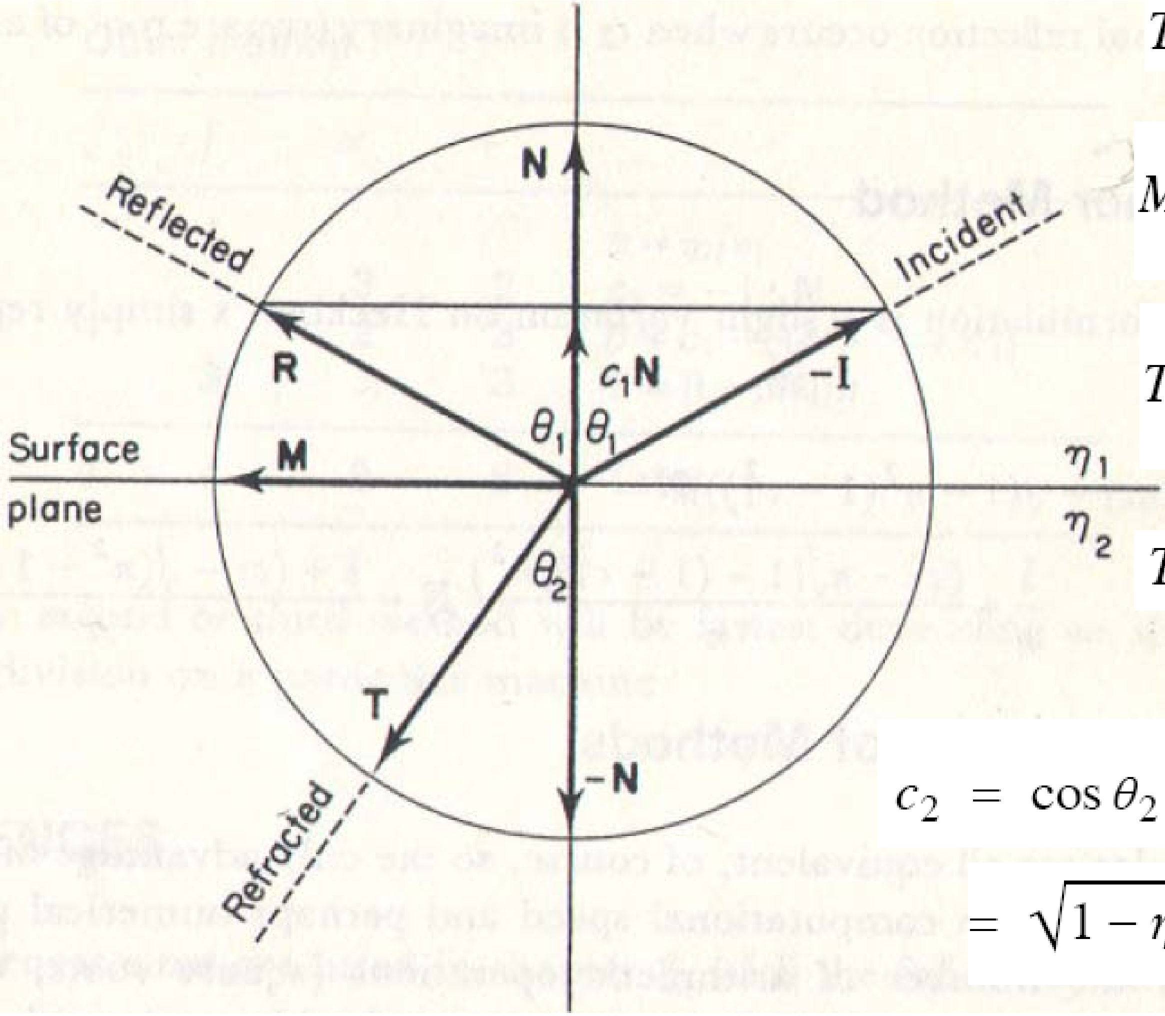
$$|I'| = \sec\theta_1$$

$$\alpha = (n^2|I'|^2 - |I' + N|^2)^{-1/2}$$

Whitted's method

Whitted's Method				
$\sqrt{\ }$	/	$\times$	+	
	1			$n = \eta_2/\eta_1$
	3	3	2	$I' = I/(-I \cdot N)$
			3	$J = I' + N$
1	1	8	5	$\alpha = 1/\sqrt{n^2(I' \cdot I') - (J \cdot J)}$
		3	3	$T' = \alpha J - N$
1	3	3	2	$T = T'/\|T'\|$
2	8	17	15	TOTAL

Heckber's method



$$T = \sin\theta_2 M - \cos\theta_2 N$$

$$M = \frac{I_{perp}}{|I_{perp}|} = \frac{I + c_1 N}{\sin\theta_1}$$

$$T = \frac{\sin\theta_2}{\sin\theta_1}(I + c_1 N) - \cos\theta_2 N$$

$$T = \eta I + (\eta c_1 - c_2) N$$

$$c_2 = \cos\theta_2 = \sqrt{1 - \sin^2\theta_2}$$

$$= \sqrt{1 - \eta^2 \sin^2\theta_1} = \sqrt{1 - \eta^2(1 - c_1^2)}$$

Heckbert's method

Heckbert's Method				
$\sqrt{\ }$	/	$\times$	+	
	1			$\eta = \eta_1 / \eta_2$
		3	2	$c_1 = -I \cdot N$
1		3	2	$c_2 = \sqrt{1 - \eta^2 (1 - c_1^2)}$
		7	4	$T = \eta I + (\eta c_1 - c_2) N$
1	1	13	8	TOTAL

Other method

$$
\begin{aligned}
T &= \eta I + (\eta c_1 - \sqrt{1 - \eta^2(1 - c_1^2)})N \\
&= \frac{I}{n} + \frac{c_1 - n\sqrt{1 - (1 - c_1^2)/n^2}}{n} N \\
&= \frac{I + (c_1 - \sqrt{n^2 - 1 + c_1^2})N}{n}
\end{aligned}
$$

Other Method				
$\sqrt{\ }$	/	$\times$	+	
	1			$n = \eta_2/\eta_1$
		3	2	$c_1 = -I \cdot N$
1		2	3	$\beta = c_1 - \sqrt{n^2 - 1 + c_1^2}$
	3	3	3	$T = (I + \beta N)/n$
1	4	8	8	TOTAL

Comparisons

Whitted's Method

$\sqrt{\ }$	/	$\times$	+	
	1			$n = \eta_2/\eta_1$
	3	3	2	$I' = I/(-I \cdot N)$
			3	$J = I' + N$
1	1	8	5	$\alpha = 1/\sqrt{n^2(I' \cdot I') - (J \cdot J)}$
		3	3	$T' = \alpha J - N$
1	3	3	2	$T = T'/\lvert T' \rvert$
2	8	17	15	TOTAL

Heckbert's Method

$\sqrt{\ }$	/	$\times$	+	
	1			$\eta = \eta_1/\eta_2$
		3	2	$c_1 = -I \cdot N$
1		3	2	$c_2 = \sqrt{1 - \eta^2(1 - c_1^2)}$
		7	4	$T = \eta I + (\eta c_1 - c_2)N$
1	1	13	8	TOTAL

Other Method

$\sqrt{\ }$	/	$\times$	+	
	1			$n = \eta_2/\eta_1$
		3	2	$c_1 = -I \cdot N$
1		2	3	$\beta = c_1 - \sqrt{n^2 - 1 + c_1^2}$
	3	3	3	$T = (I + \beta N)/n$
1	4	8	8	TOTAL