

# INFOGR – Computer Graphics

J. Bikker - April-July 2015 - Lecture 10: “Ground Truth”

# Welcome!



# Today's Agenda:

- Limitations of Whitted-style Ray Tracing
- Monte Carlo
- Path Tracing

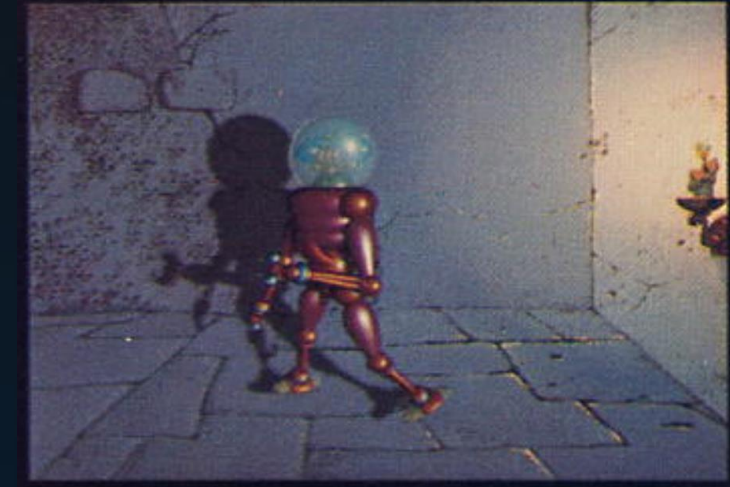


# Whitted Recap

## Whitted-style Ray Tracing

In 1980, “State of the Art” consisted of:

- Rasterization
- Shading: either diffuse ( $N \cdot L$ ) or specular ( $(N \cdot H)^n$ ), both not taking into account fall-off (Phong)
- Reflection, using environment maps (Blinn & Newell \*)
- Stencil shadows (Williams \*\*)



Goal:

- Solve reflection and refraction

Improved model:

- Based on classical ray optics

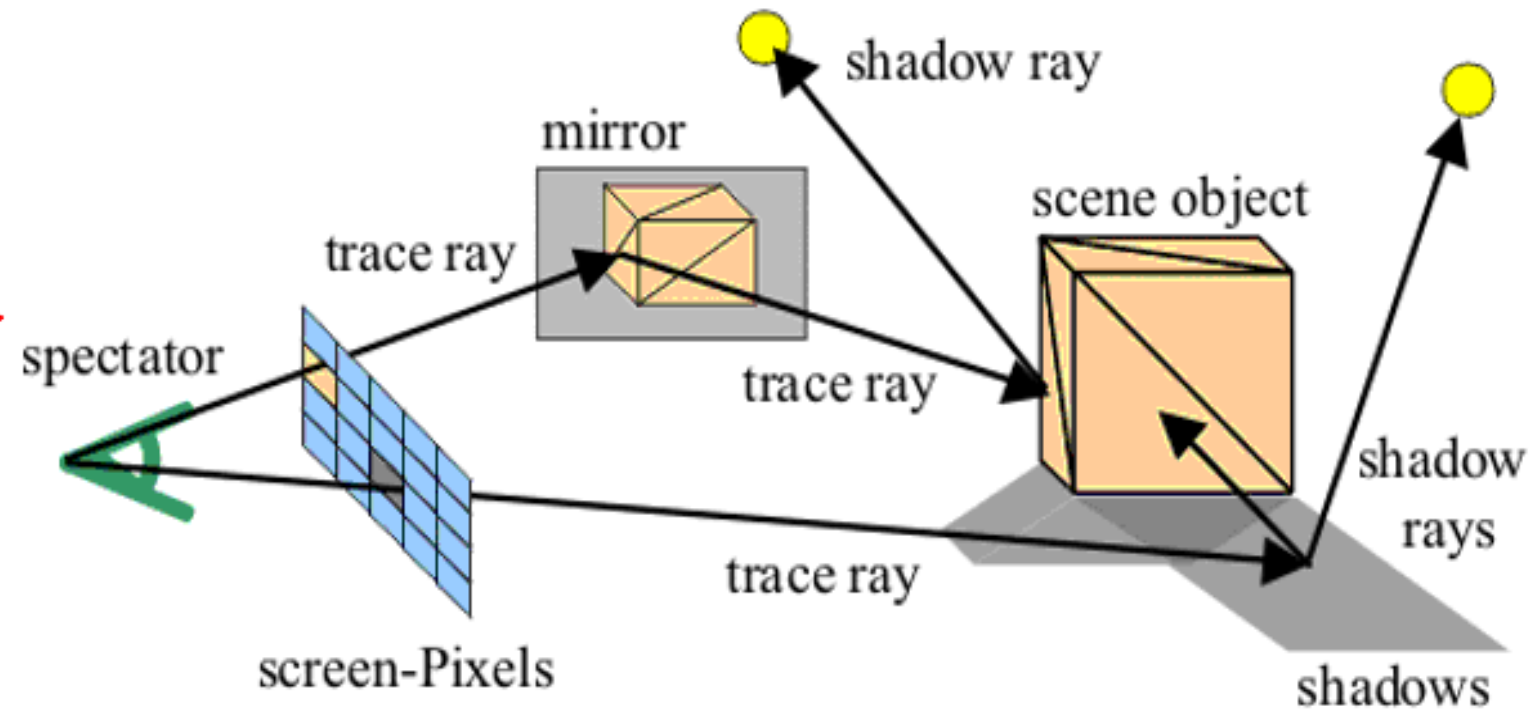
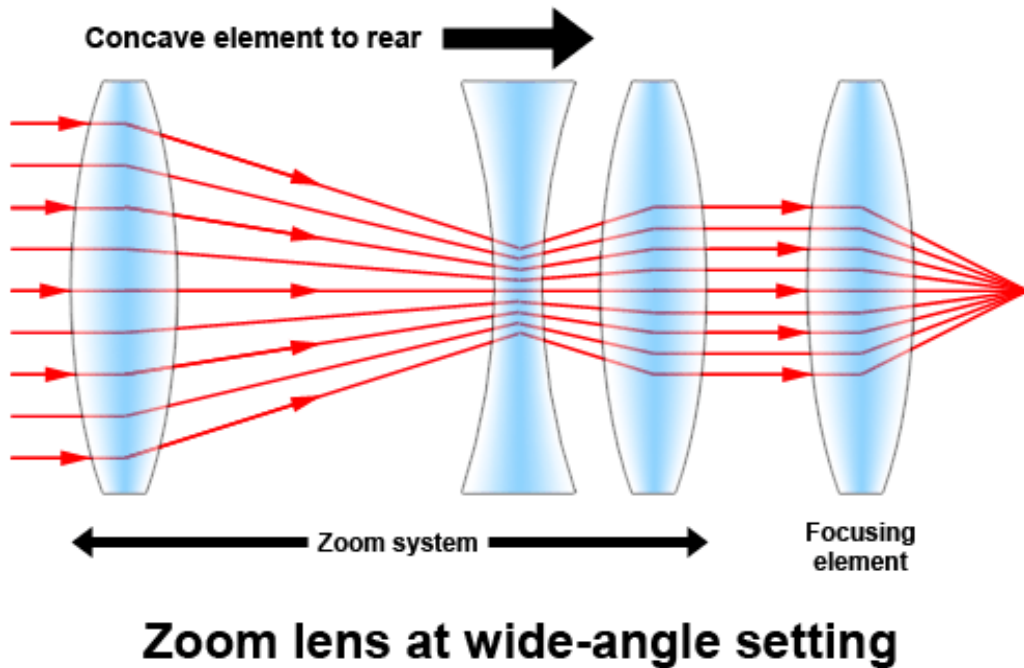
\* : Blinn, J. and Newell, M. 1976. Texture and Reflection in Computer Generated Images. Communications of the ACM 19:10 (1976), 542—547.

\*\* : Williams, L. 1978. Casting curved shadows on curved surfaces. In Computer Graphics (Proceedings of SIGGRAPH 78), vol. 12, 270–274.



# Whitted Recap

## Whitted-style Ray Tracing



```
void walk = done properly, closely follow the path of the ray (ive)
```

```
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, BR, &pdf );
```

```
survive;
```

```
pdf;
```

```
n = E * brdf * (dot( N, R ) / pdf);
```

```
tion = true;
```



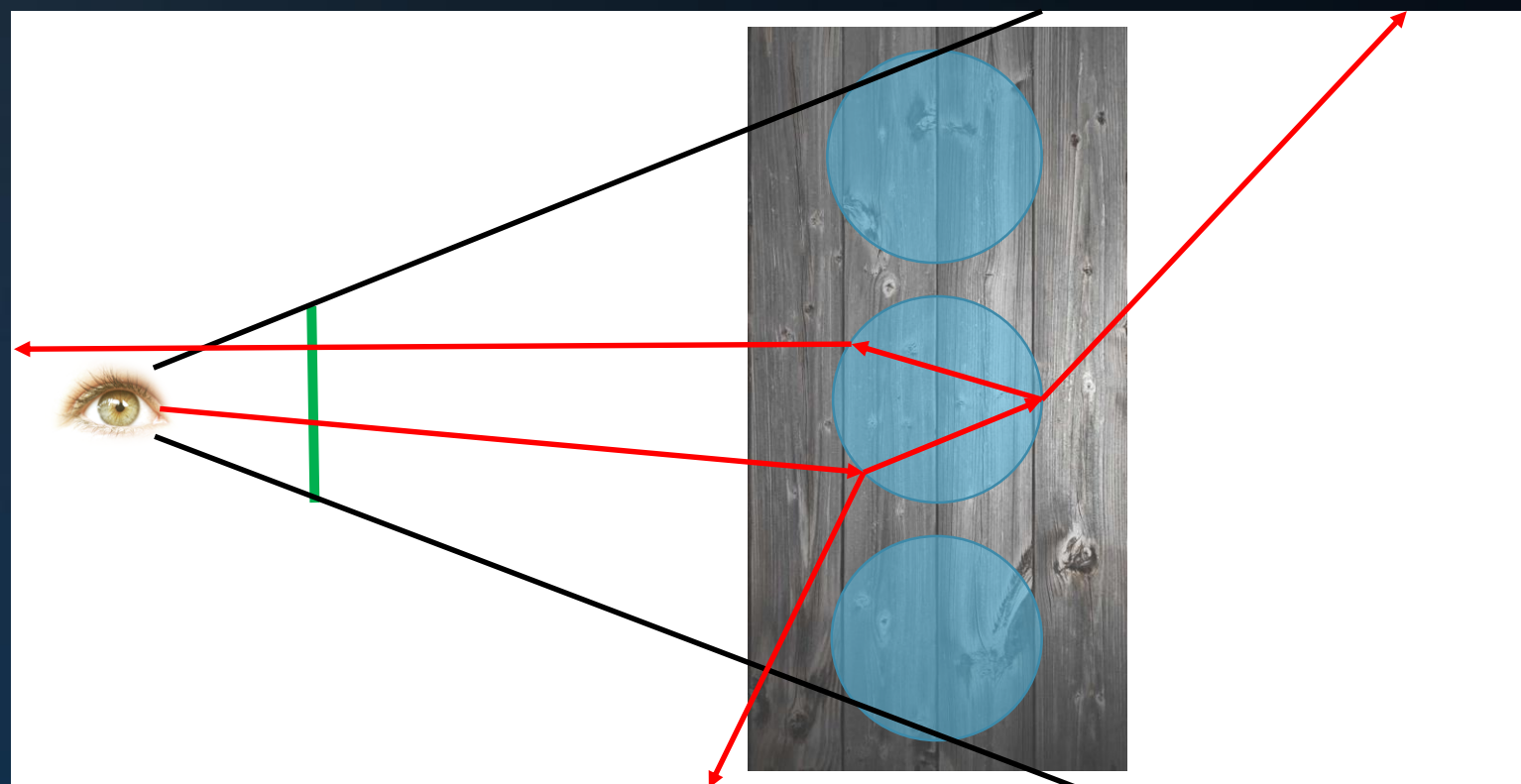
# Whitted Recap

## Whitted-style Ray Tracing

Color at pixel:

- sphere material color \*  
refracted ray
- + sphere material color \*  
reflected ray

This is a recursive process.



# Whitted Recap

## Whitted-style Ray Tracing

Color at pixel:

- sphere material color \* refracted ray
- + sphere material color \* reflected ray

*This is a recursive process.*

Fresnel equations

Snell's law

```
color Trace( O, D )
    I, N, mat = NearestIntersection( O, D )
    if (mat == DIFFUSE)
        return mat.color *
            DirectIllumination( I, N )
    if (mat == MIRROR)
        return mat.color *
            Trace( I, reflect( D, N ) )
    if (mat == GLASS)
        return mat.color *
            (X * Trace( I, reflect( D, N ) ) +
             (1-X) * Trace( I, refract( D, N ) ) )
```

angle of incidence = angle of reflection





# Whitted Recap

## Whitted-style Ray Tracing

Improved model:

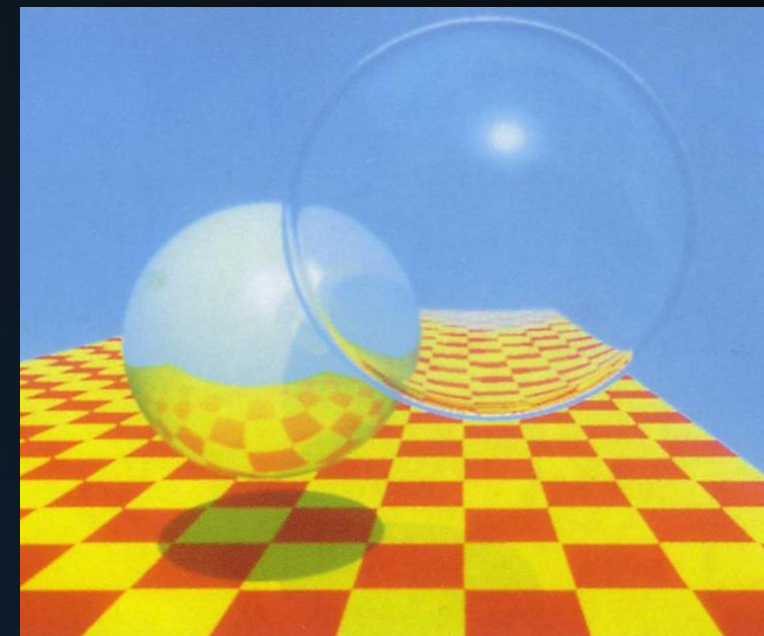
- Based on classical ray optics

Dust off your physics books.

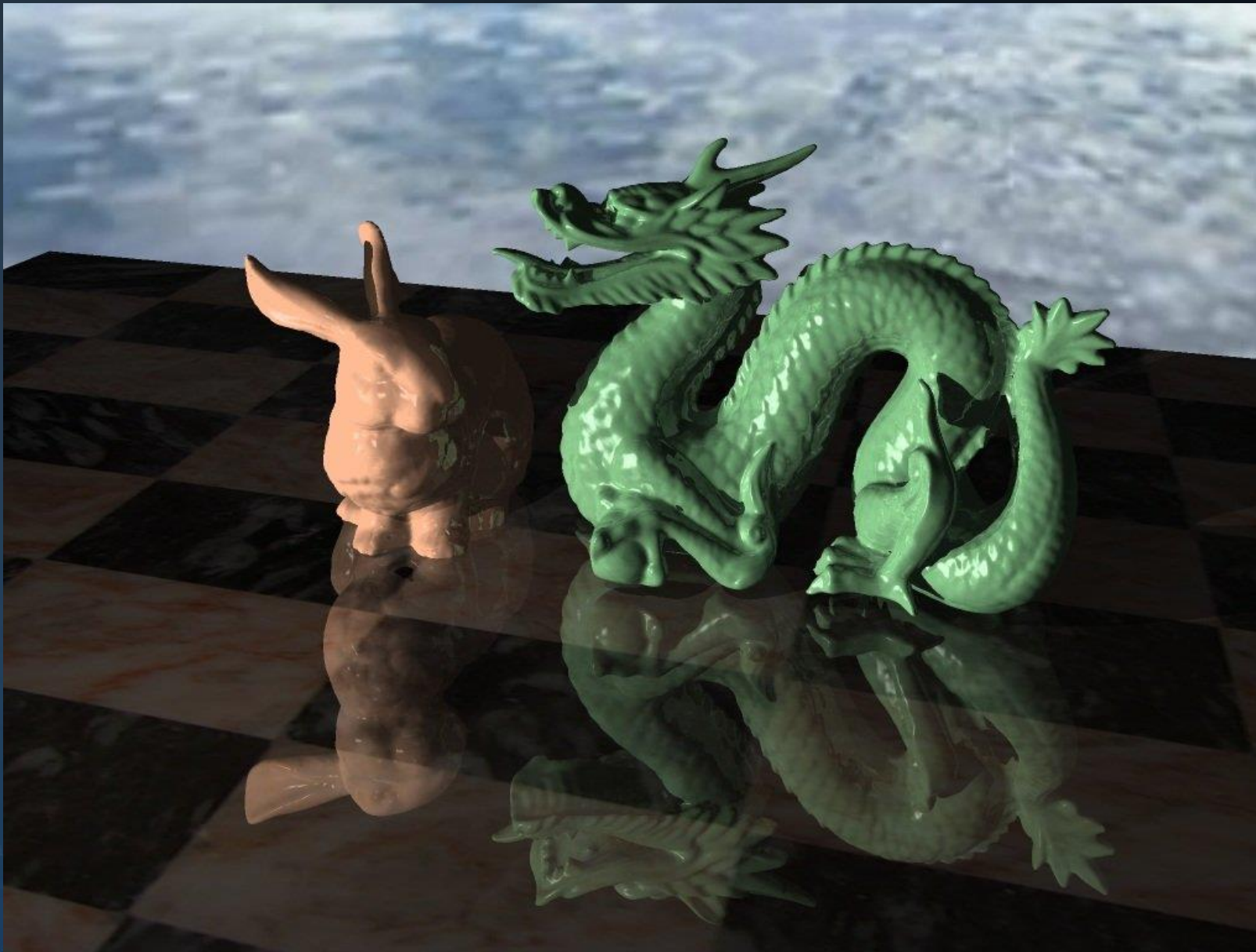
Physical basis of Whitted-style ray tracing:

Light paths are generated (backwards) from the camera to the light sources, using rays to simulate optics.

Whitted-style ray tracing is deterministic: it cannot simulate area lights, glossy reflections, and diffuse reflections.



```
...ics
& (depth < MAXDEPTH)
{
    // Inside / Outside
    int nt = nc; nct = nc; nct2 = nc;
    float cos2t = 1.0f - nnt;
    float D, N;
    // ...
    float a = nt - nc, b = nt - nc;
    float Tr = 1 - (R0 + (1 - R0) * a);
    float R = (D * nnt - N * (a * a));
    // ...
    E * diffuse;
    // ...
    refl + refr)) && (depth < MAXDEPTH)
    {
        D, N;
        refl * E * diffuse;
        // ...
        MAXDEPTH)
    survive = SurvivalProbability( diffuse );
    estimation - doing it properly, closely following
    if;
    radiance = SampleLight( &rand, I, &L, &light );
    radiance.x + radiance.y + radiance.z > 0) && (depth <
    w = true;
    float brdfPdf = EvaluateDiffuse( L, N ) * Psurvive;
    float3 factor = diffuse * INVPI;
    float weight = Mis2( directPdf, brdfPdf );
    float cosThetaOut = dot( N, L );
    E * ((weight * cosThetaOut) / directPdf) * (radi
    random walk - done properly, closely following the
    survive)
    ;
    float3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    sion = true;
}
```







 NVIDIA

# Today's Agenda:

- Limitations of Whitted-style Ray Tracing
- Monte Carlo
- Path Tracing



# Monte-Carlo

## Distributed Ray Tracing\*

### Problem:

Ray tracing is currently limited to sharp shadows, sharp reflections, and sharp refraction.

### Goal:

- Augment Whitted-style ray tracing with glossy reflections and refractions, as well as soft shadows.



\*: “Distributed Ray Tracing”, Cook et al., 1984.

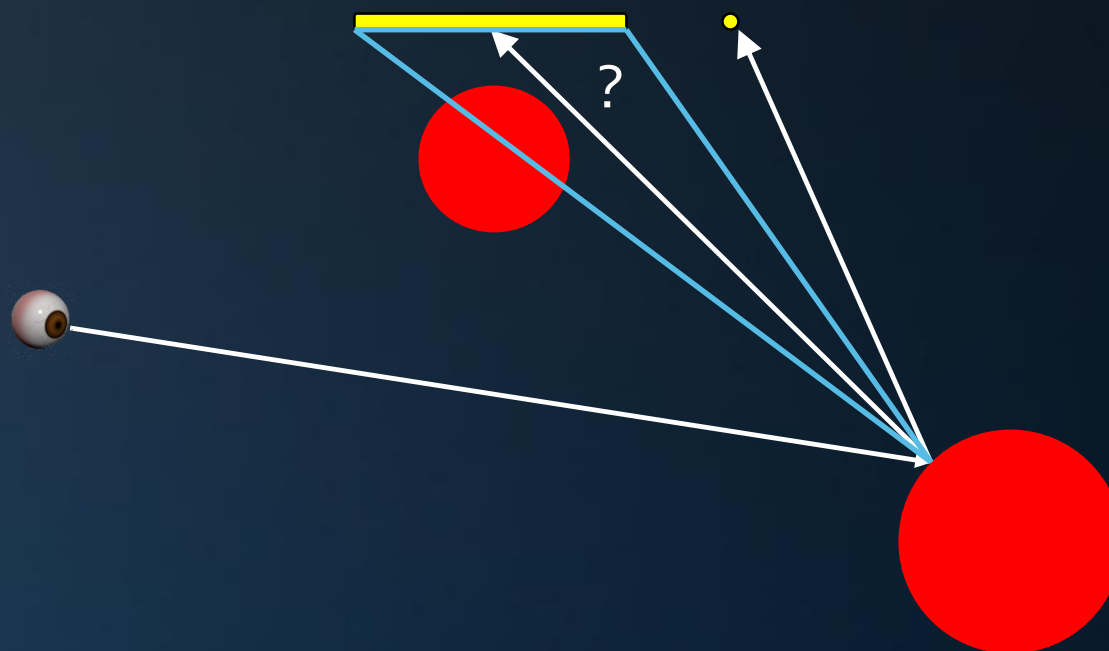














# Monte Carlo

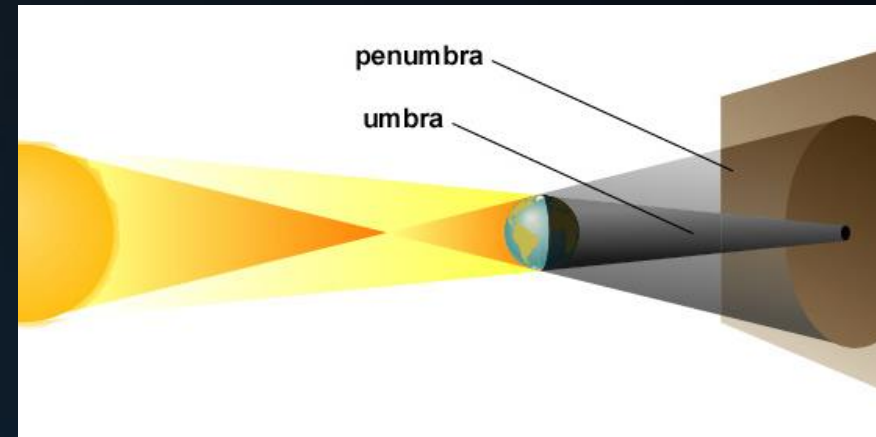
## Analytic Soft Shadows

### Anatomy of a shadow – regions

- Fully occluded area: *umbra*
- Partially occluded area: *penumbra*

*A soft shadow requires an area light source.*

In nature, all light sources are area lights (although some approximate point lights).





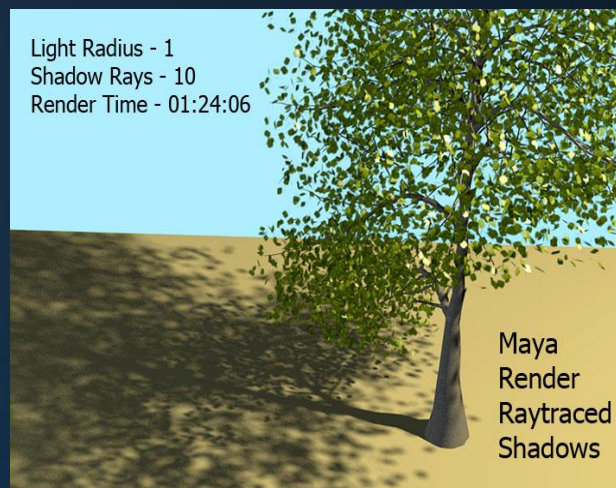
# Monte Carlo

## Analytic Soft Shadows

Surface points in the penumbra are lit by a part of the light source.

Rendering soft shadows requires that we determine the visible portion of the light source.

In most cases, this is a very hard problem.



# Monte Carlo

## Approximate Soft Shadows

When using shadow mapping, we can simulate soft shadows by blurring the shadow map.



In this example, filter kernel radius is adjusted based on the distance from the occluder.



# Monte Carlo

## Calculating Accurate Soft Shadows

*“Rendering soft shadows requires that we determine the visible portion of the light source.”*

In other words:

The amount of light cast on a surface point P by area light L is determined by the integral of the visibility between P and L over the surface of the light source:

$$I_{L \rightarrow P} = \int_{A_L} V(P, L)$$

## Monte-Carlo Integration

To solve this integral for the generic case, we will use Monte-Carlo integration.

Using Monte-Carlo, we replace the integral by the expected value of a stochastic experiment.



# Monte Carlo

## Stochastic shadows

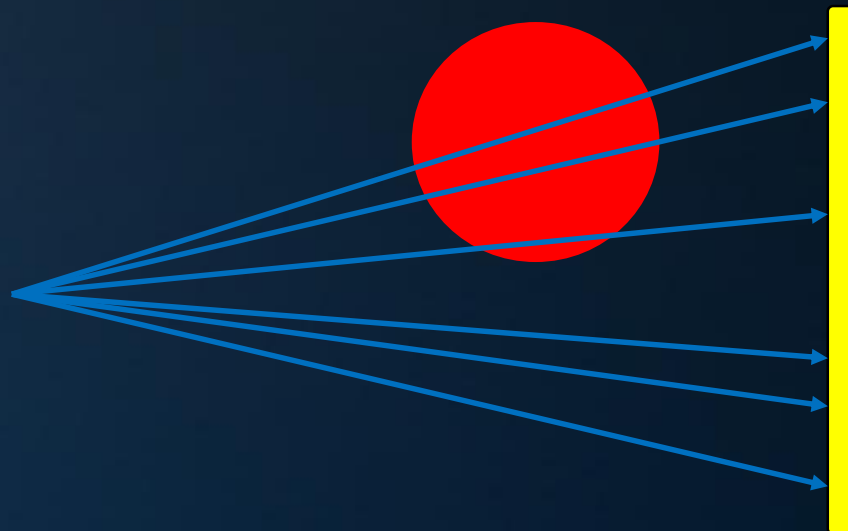
For soft shadows, we want to know the visible area of a light source, which can be 0..100%.

The light source could be (partially) obscured by any number of objects.

*We can approximate the visibility of the light source using a number of random rays.*

Using 6 rays:

$$V \approx \frac{1}{6} \sum_{i=1}^6 V_i$$





# Monte Carlo

## Stochastic shadows

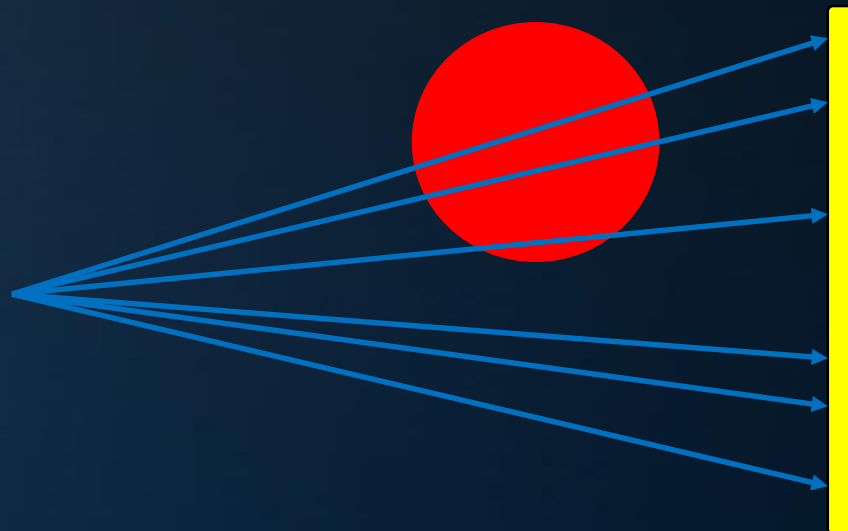
For soft shadows, we want to know the visible area of a light source, which can be 0..100%.

The light source could be (partially) obscured by any number of objects.

*We can approximate the visibility of the light source using a number of random rays.*

Using N rays:

$$V \approx \frac{1}{N} \sum_{i=1}^N V_i$$



# Monte Carlo

## Stochastic shadows

$$V \approx \frac{1}{N} \sum_{i=1}^N V_i$$

As  $N$  approaches infinity, the result becomes equal to the expected value, which is the integral we were looking for.

Before that, the result will exhibit *variance*.  
In the case of soft shadows, this shows up as noise.

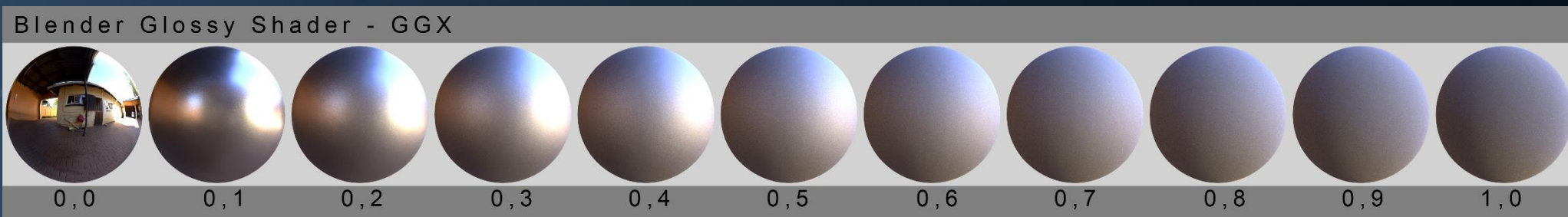


# Monte Carlo

## Approximate Diffuse Reflections

When rendering diffuse reflections, we face a similar problem:

A glossy surface reflects light arriving from a range of directions.



In rasterization, we can achieve this by blurring the environment map.





# Monte Carlo

Note that a correct glossy reflection requires a filter kernel size based on distance to the reflected object.

```
...
    & (depth < MAXDEPTH)
    {
        // Inside / Outside
        int nt = nt / nc;
        double r2t = 1.0f - nt;
        double r2b = 1.0f - r2t;
        double R = (D * nnt - N * (r2b - r2t));

        // Diffuse
        E * diffuse;
        = true;

        // Refractive
        refl + refr) && (depth < MAXDEPTH)
        {
            D, N);
            refl * E * diffuse;
            = true;

            MAXDEPTH)
            survive = SurvivalProbability( diffuse );
            estimation - doing it properly, closely
            if;
            radiance = SampleLight( &rand, I, &I, &light );
            e.x + radiance.y + radiance.z > 0) && (depth < MAXDEPTH)
            {
                v = true;
                brdfPdf = EvaluateDiffuse( L, N ) * Psurvive;
                factor = diffuse * INVPI;
                weight = Mis2( directPdf, brdfPdf );
                cosThetaOut = dot( N, L );
                E * ((weight * cosThetaOut) / directPdf) * (radiance);
            }
            // Random walk - done properly, closely following walls
            survive)
            {
                brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
                survive;
                pdf;
                n = E * brdf * (dot( N, R ) / pdf);
                sion = true;
            }
        }
    }
}
```

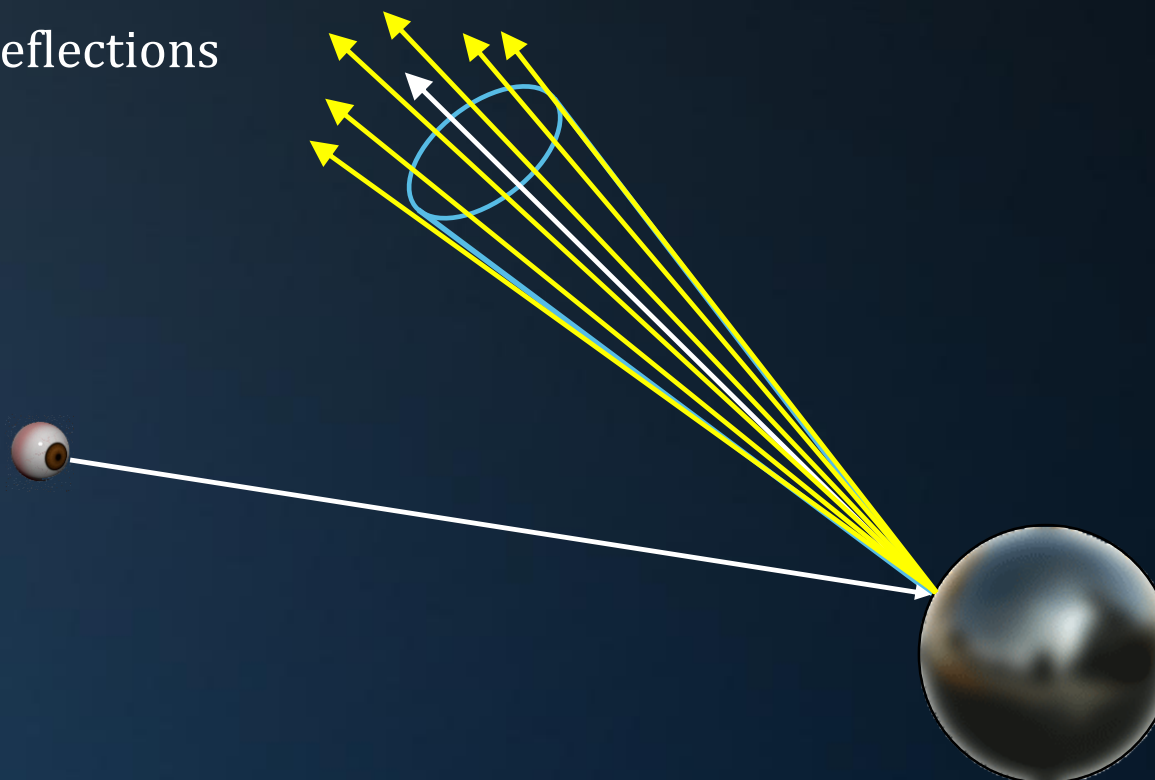




```

100
101 (depth < MAXD)
102 {
103     nc = inside / 1.0f * 1.0f;
104     nt = nt / nc; ddo = 1.0f - nt;
105     cos2t = 1.0f - nt * nt;
106     D, N );
107 }
108
109 at a = nt - nc, b = nt * nc;
110 at Tr = 1 - (RB + (1 - RB) * a);
111 Tr) R = (D * nnt - N * (ddo
112
113 E * diffuse;
114 = true;
115
116
117 refl + refr)) && (depth < MAXDEPTH)
118 {
119     D, N );
120     refl * E * diffuse;
121     = true;
122
123
124 MAXDEPTH)
125
126 survive = SurvivalProbability( diffuse,
127 estimation - doing it properly, clearly
128 if;
129 radiance = SampleLight( &rand, I, &L, &align,
130 e.x + radiance.y + radiance.z) > 0) && (maxN <
131
132 w = true;
133 at brdfPdf = EvaluateDiffuse( L, N); * Psurvive;
134 at3 factor = diffuse * INVPI;
135 at weight = Mix2( directPdf, brdfPdf );
136 at cosThetaOut = dot( N, L );
137 E * ((weight * cosThetaOut) / directPdf) * (radiance
138
139 random walk - done properly, closely following
140 (survive)
141
142
143 at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf
144 survive;
145 pdf;
146 n = E * brdf * (dot( N, R ) / pdf);
147 sion = true;

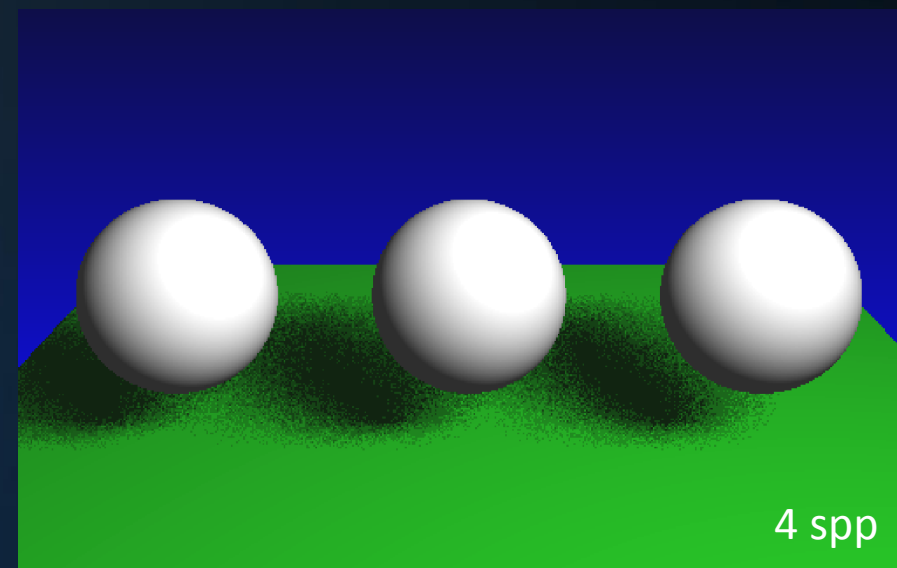
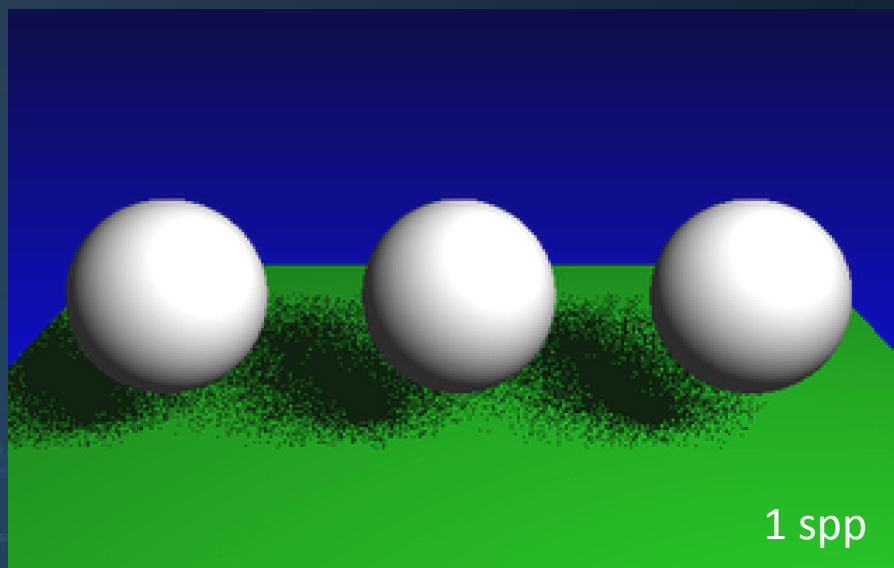
```



# Monte Carlo

## Variance

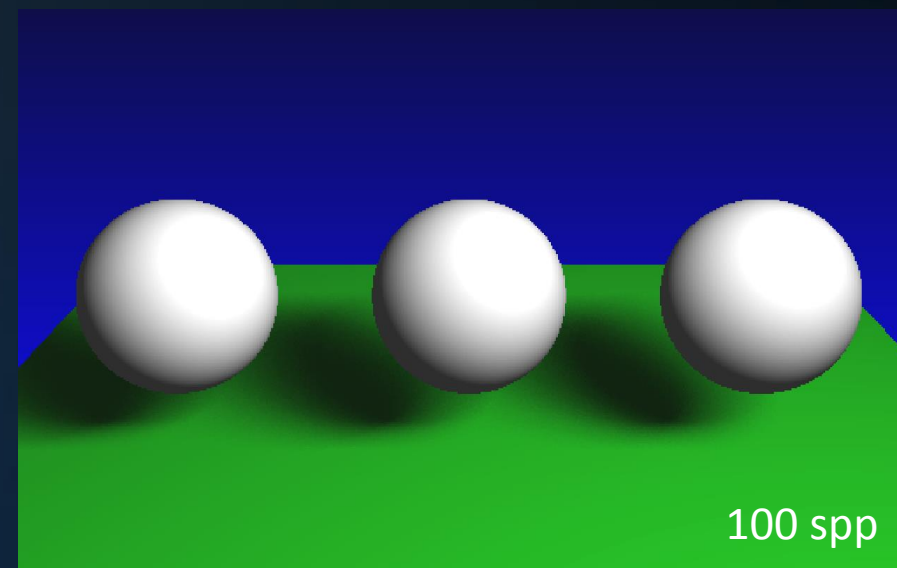
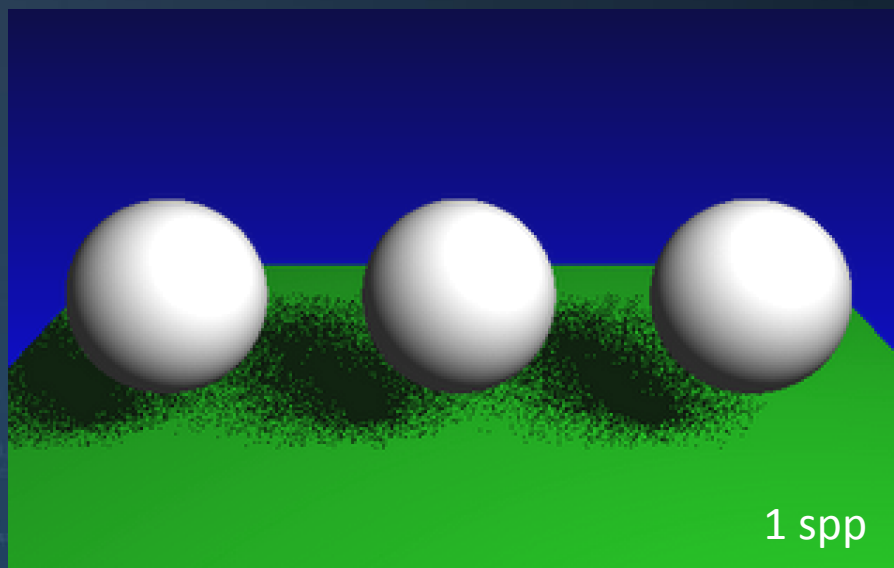
As long as we don't take an infinite amount of samples, the result of the stochastic process exhibits variance.



# Monte Carlo

## Variance

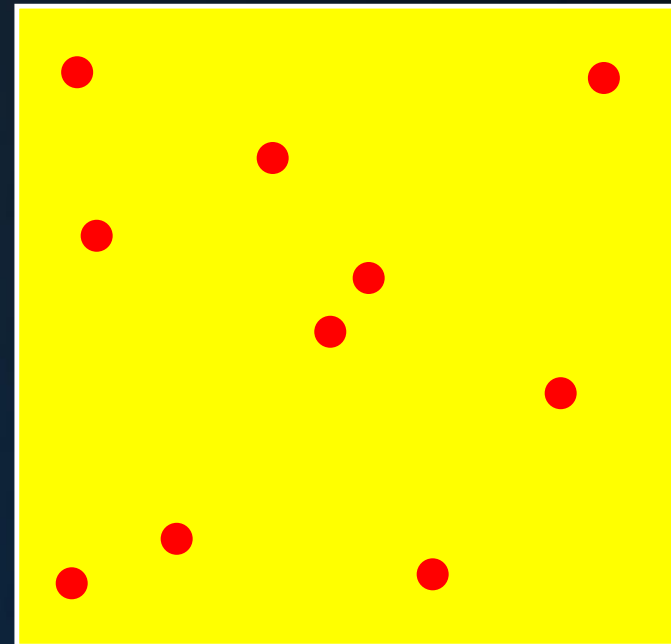
As long as we don't take an infinite amount of samples, the result of the stochastic process exhibits variance.



# Monte Carlo

## Variance reduction: stratification

The variance in random sampling can be reduced using *stratification*.



$N=16$

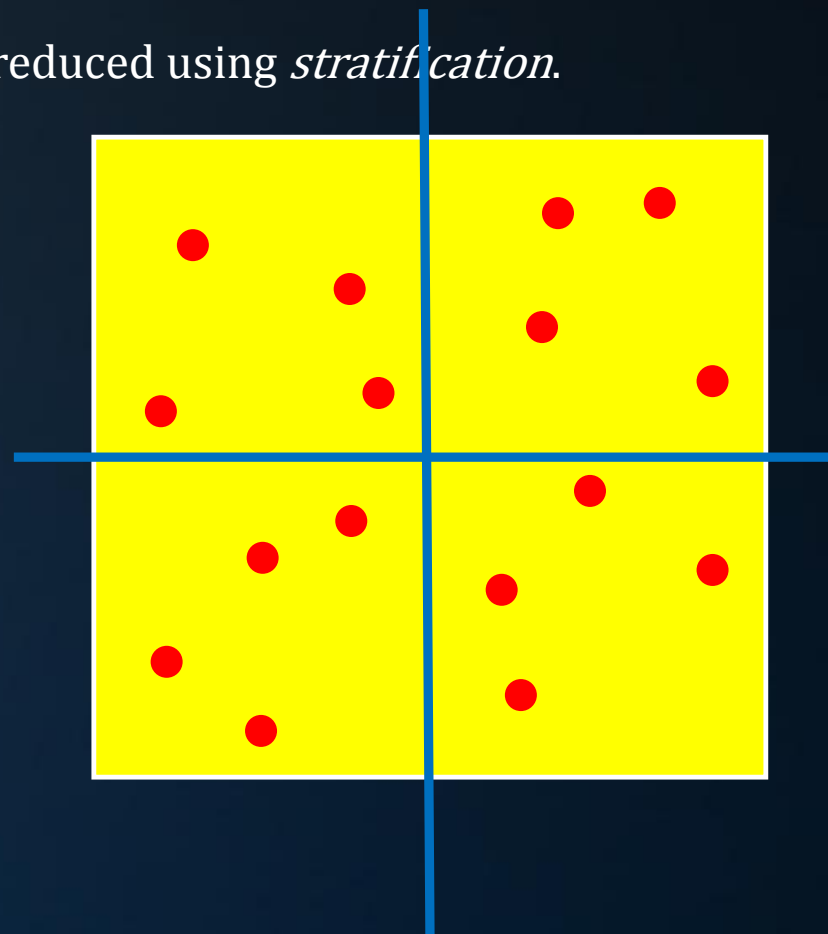




# Monte Carlo

## Variance reduction: stratification

The variance in random sampling can be reduced using *stratification*.



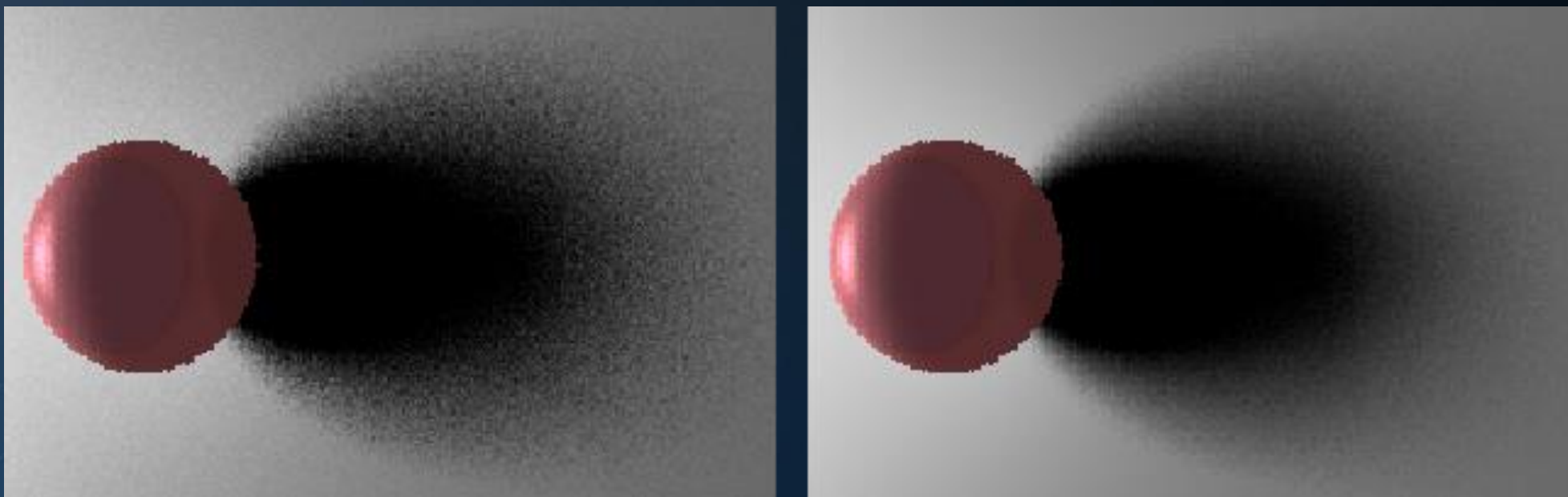
$N=16$



# Monte Carlo

## Variance reduction: stratification

The variance in random sampling can be reduced using *stratification*.



*Uniform vs stratified, 36 samples, 6x6 strata*



# Monte-Carlo

## Distributed Ray Tracing

Integrating over area of light sources: soft shadows

Integrating over reflection cone: glossy reflections

Integrating over pixel: anti-aliasing

Integrating over time: motion blur

Integrating over lens: depth of field

Integrating over wavelength: dispersion



# Monte Carlo

## Distributed Ray Tracing

### Improved model:

- Still based on classical ray optics
- Combined with probability theory to solve integrals

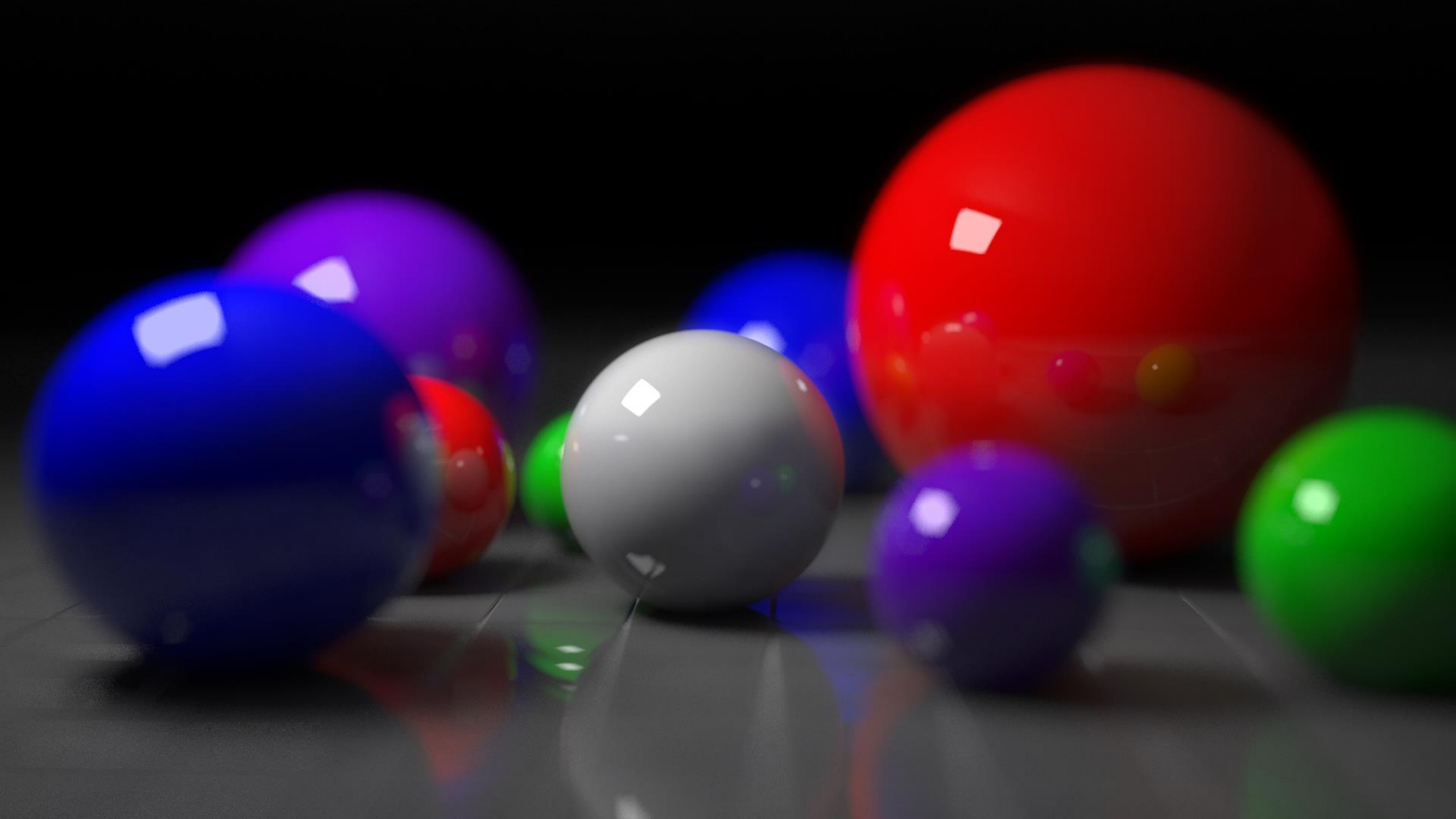
### Physical basis of distributed ray tracing:

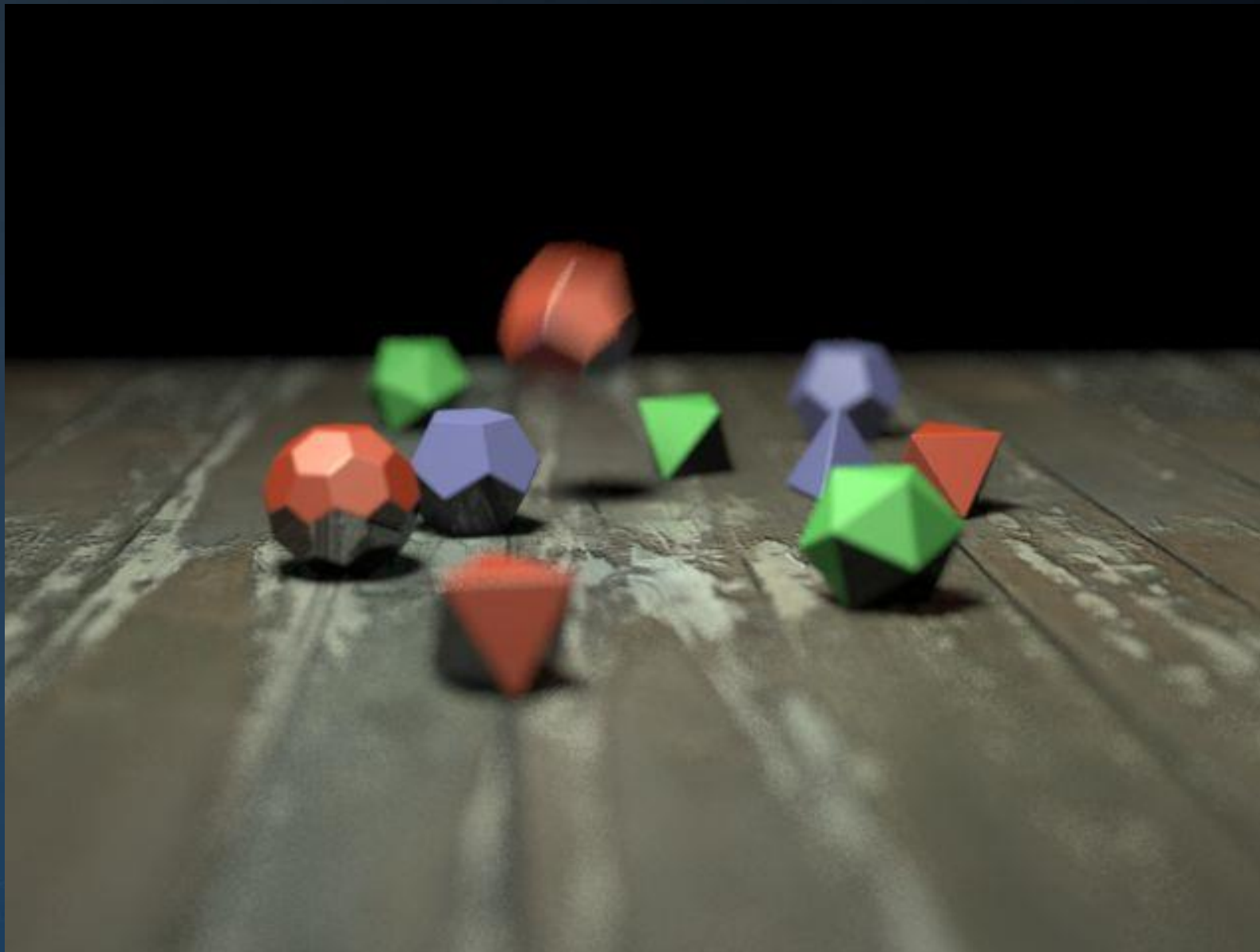
Light paths are generated (backwards) from the camera to the light sources, using rays to simulate optics.

Distributed ray tracing requires many rays to bring down variance to acceptable levels.









```
rice  
& (depth  
t = insid  
nt = nt /  
os2t = 1.  
D, N );  
0)  
at a = nt  
at Tr = 1  
Tr) R = (  
E * diffu  
= true;  
-  
efl + ref  
D, N );  
-refl * E  
= true;  
MAXDEPTH)  
survive =  
estimat  
if;  
-radiance  
e.x + rad  
v = true;  
at brdfPd  
at3 facto  
at weight  
at cosThe  
E * ((we  
random wal  
vive)
```

```
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, BR, kpt  
survive;  
pdf;  
n = E * brdf * (dot( N, R ) / pdf);  
sion = true;
```





# Monte Carlo in Rasterization

Robert Toth &amp; Erik Lindler, 2008





# Monte Carlo

## Monte Carlo in Rasterization

Screen Space Ambient Occlusion,  
CryEngine 2, 2007.

```

    if (depth < MAXDEPTH)
    {
        // Inside / Outside
        int nt = nc;
        float cos2t = 1.0f - nnt;
        Vec t = D * N;
        Vec a = nt - nc;
        Vec b = nt + nc;
        float Tr = 1 - (R0 + (1 - R0) * cos2t);
        Vec R = (D * nnt - N * (1 - Tr));
        Vec E * diffuse;
        bool = true;
        Vec refl + refr;
        Vec D, N;
        Vec refl * E * diffuse;
        bool = true;
        Vec MAXDEPTH);
        Vec survive = SurvivalProbability( diffuse );
        Vec estimation - doing it properly, closely following wall (survive);
        Vec radience = SampleLight( &rand, I, &L, &light );
        Vec e.x + radience.y + radience.z > 0) && (e.x + radience.y + radience.z > 0);
        Vec w = true;
        Vec brdfPdf = EvaluateDiffuse( L, N ) * Psurvive;
        Vec3 factor = diffuse * INVPI;
        Vec weight = Mix2( directPdf, brdfPdf );
        Vec cosThetaOut = dot( N, L );
        Vec E * ((weight * cosThetaOut) / directPdf) * (radiance);
        Vec random walk - done properly, closely following wall (survive);
        Vec3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
        Vec survive;
        Vec pdf;
        Vec n = E * brdf * (dot( N, R ) / pdf);
        Vec ion = true;
    }

```



**HORDE 3D**  
NEXT-GENERATION GRAPHICS ENGINE



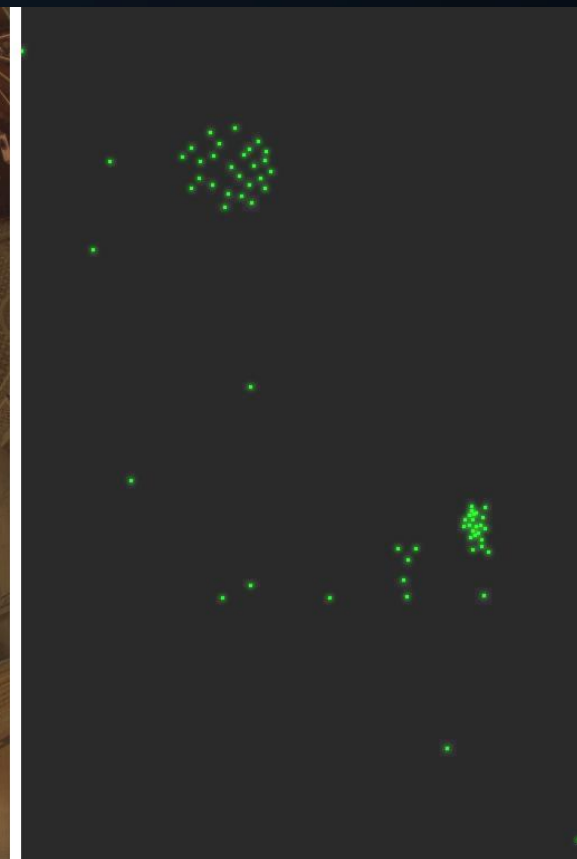
# Monte Carlo

## Monte Carlo in Rasterization

Light from an environment map,  
from:

“Wavelet Importance Sampling: Efficiently  
Evaluating Products of Complex  
Functions”,

Clarberg et al., 2005.



# Monte Carlo

## Cost of Distributed Ray Tracing

Distributed Ray Tracing is an expensive process:

- Per primary hit point, we need  $\sim 64$  shadow rays *per light*
- Per primary hit point on a glossy surface, we need  $\sim 64$  reflection rays,
  - ...and, for each reflection ray hit point, we need  $\sim 64$  shadow rays per light.

If we use 4x4 anti-aliasing per pixel, multiply the above by 16.

Now imagine a glossy surface reflects another glossy surface...



# Monte Carlo

```
...
    & (depth < MAXDEPTH)
{
    // Inside / Outside
    int nt = nt / nc, nde = nde / nc;
    double r2t = 1.0f - nnt * nnt;
    double D, N );
}

// Russian roulette
int a = nt - nc, b = nt / nc;
// Russian roulette
int Tr = 1 - (R0 + (1 - R0) * r2t);
// Russian roulette
Tr) R = (D * nnt - N * (1 - Tr));

// Russian roulette
E * diffuse;
= true;

// Russian roulette
refl + refr)) && (depth < MAXDEPTH)
{
    D, N );
    refl * E * diffuse;
    = true;

    MAXDEPTH)

survive = SurvivalProbability( diffuse );
estimation - doing it properly, closely following
if;
radiance = SampleLight( &rand, I, &L, &light,
e.x + radiance.y + radiance.z ) > 0) && (depth <
w = true;
at brdfPdf = EvaluateDiffuse( L, N ) * Psurvive;
at3 factor = diffuse * INVPI;
at weight = Mis2( directPdf, brdfPdf );
at cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf) * (radiance
Random walk - done properly, closely following
survive)

;
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;
```





# Today's Agenda:

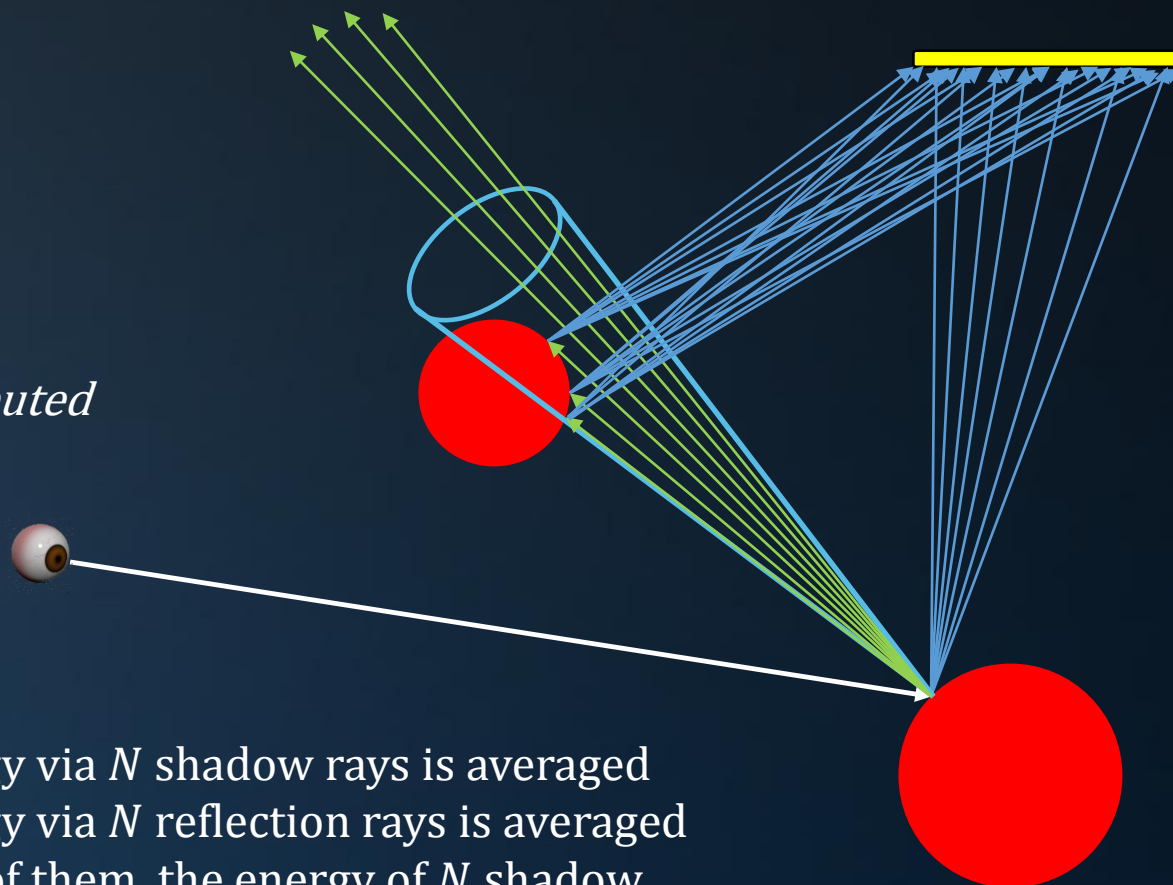
- Limitations of Whitted-style Ray Tracing
- Monte Carlo
- Path Tracing



# Physically Based

## Ray Tree

*Using distributed ray tracing:*



- The energy via  $N$  shadow rays is averaged
- The energy via  $N$  reflection rays is averaged
- For each of them, the energy of  $N$  shadow rays is averaged.

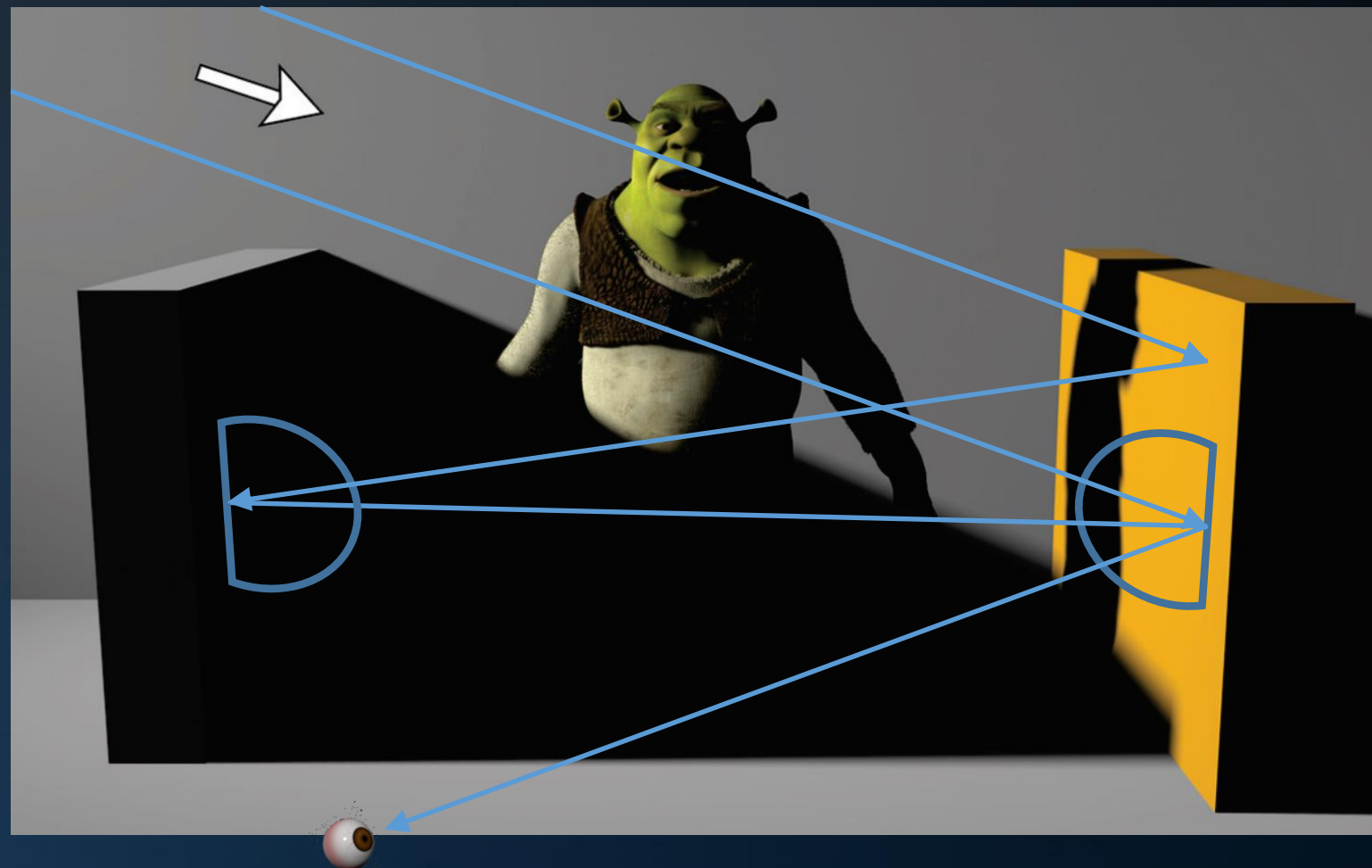
→ *The energy via each shadow ray is very low.*



# Physically Based

## Diffuse reflections

Apart from specular and glossy materials, diffuse materials also reflect light.

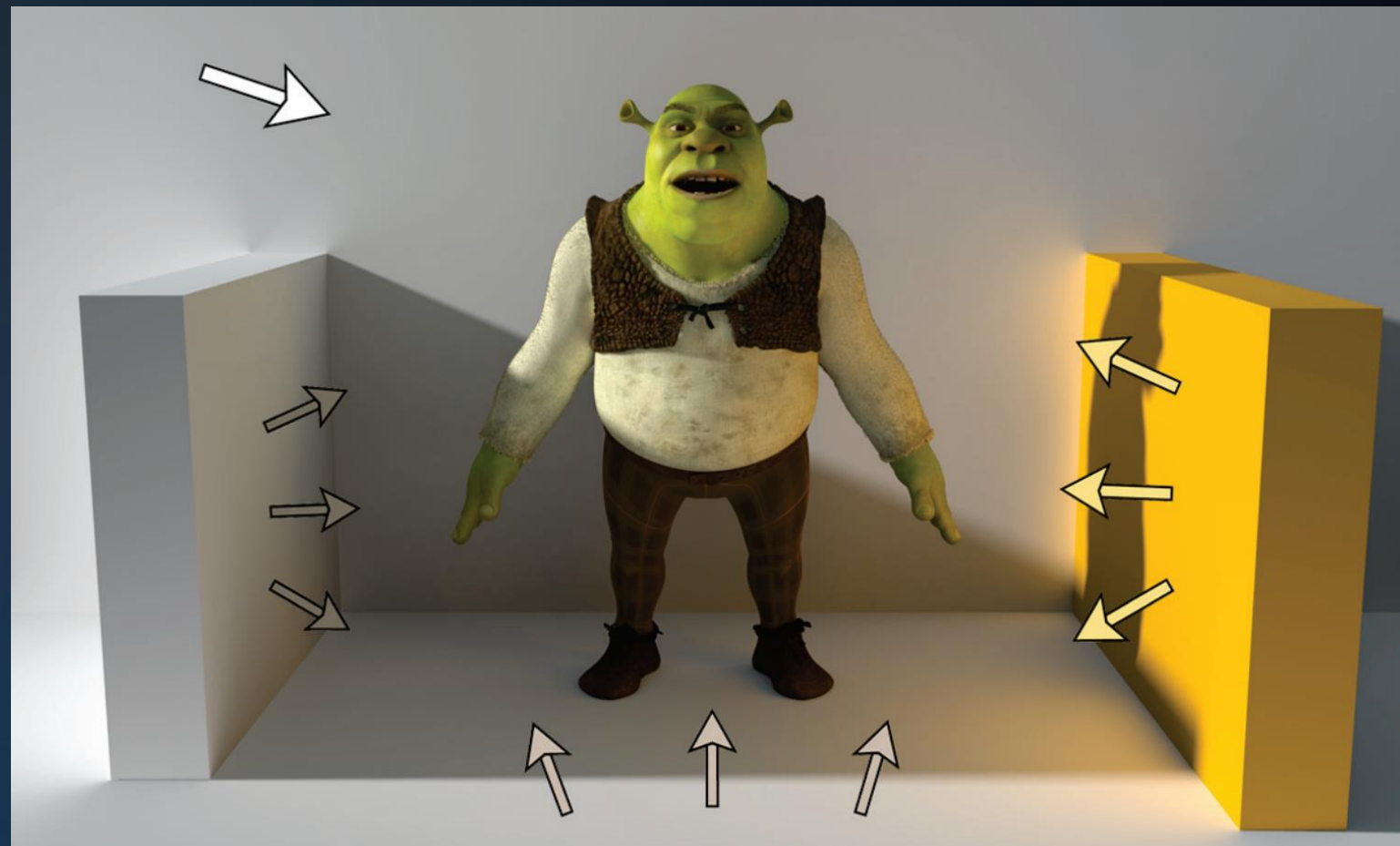


# Physically Based

## Diffuse reflections

Apart from specular and glossy materials, diffuse materials also reflect light.

This is why a shadow is seldom black.



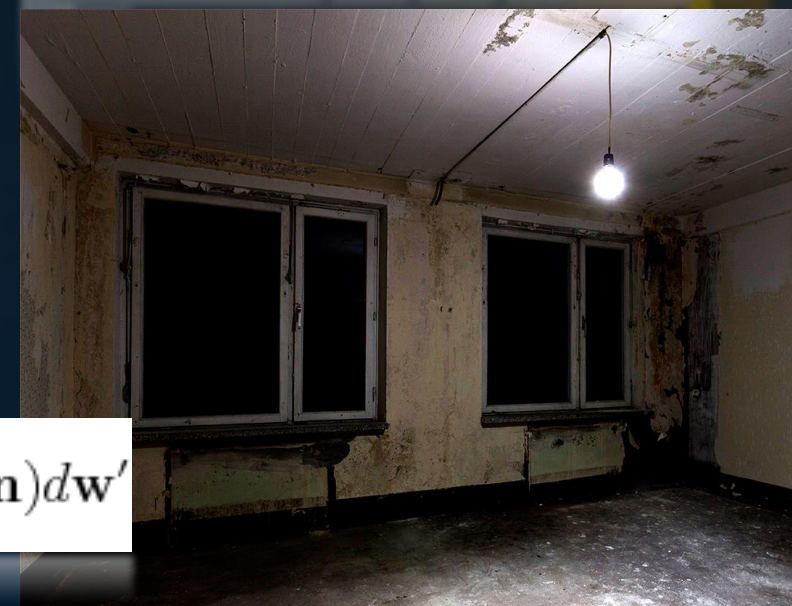
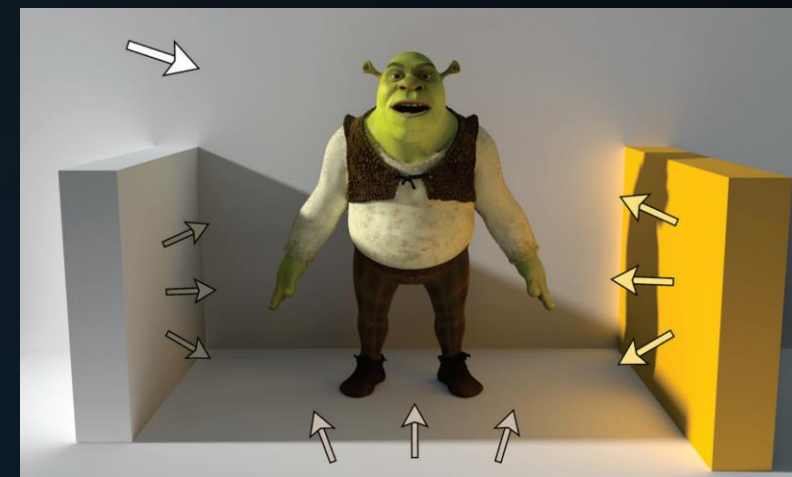


# Physically Based

## Physically based rendering

Calculating all light transport from the light sources to the camera, directly or via scene surfaces.

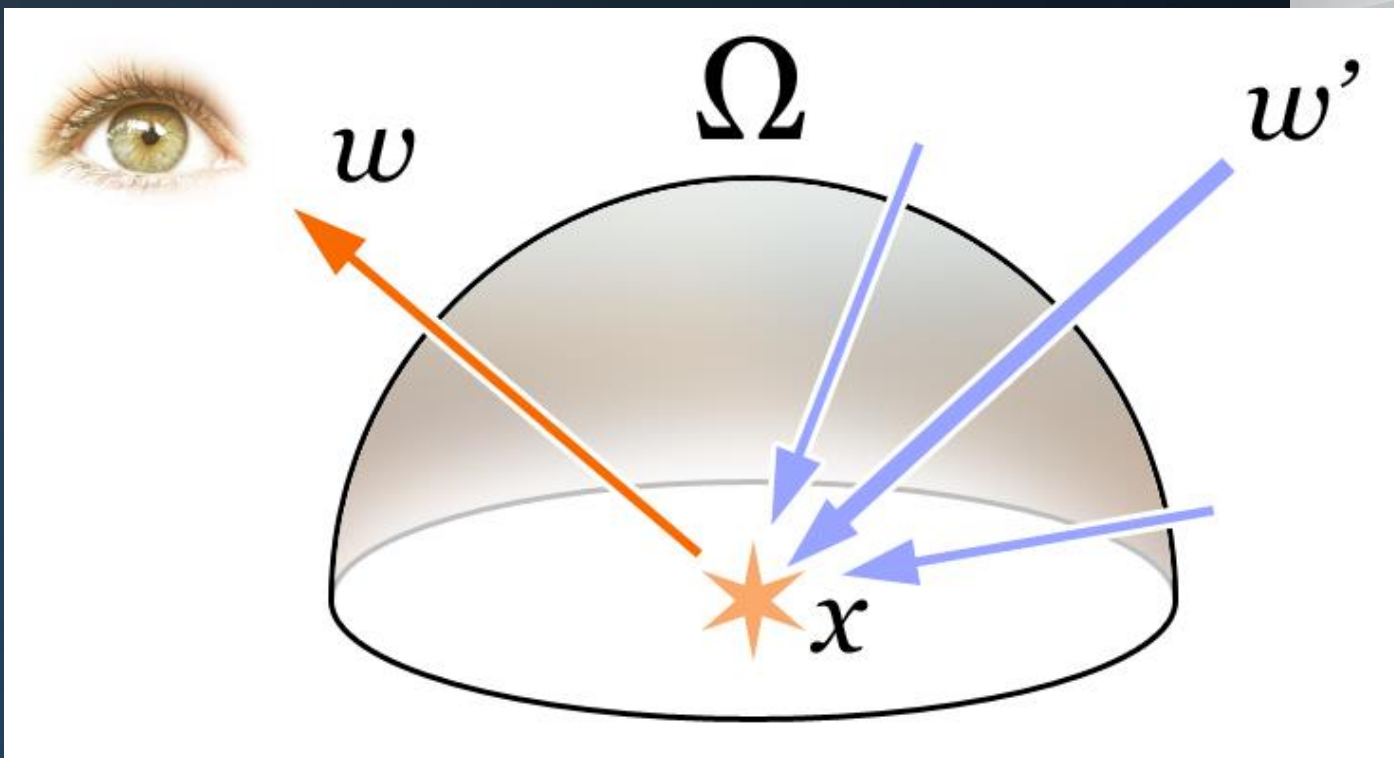
Nature solves this using a “random walk”: a large number of photons travelling through space from lights to sensors.



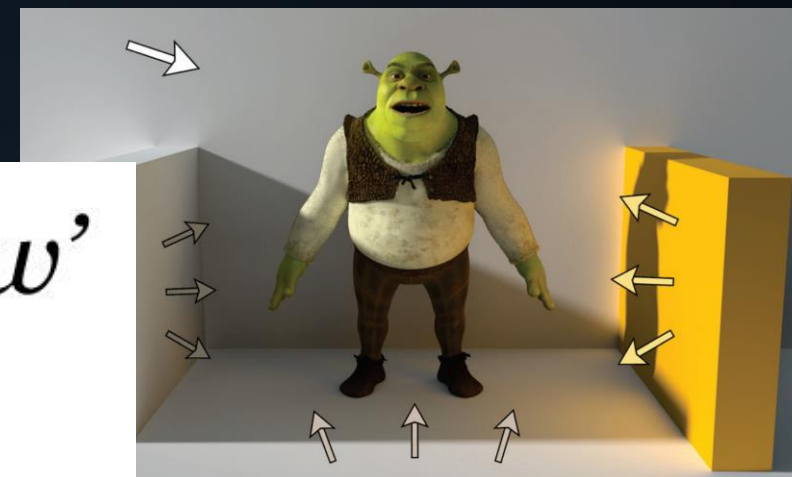
$$L_o(\mathbf{x}, \mathbf{w}) = L_e(\mathbf{x}, \mathbf{w}) + \int_{\Omega} f_r(\mathbf{x}, \mathbf{w}', \mathbf{w}) L_i(\mathbf{x}, \mathbf{w}') (-\mathbf{w}' \cdot \mathbf{n}) d\mathbf{w}'$$



# Physically Based



$$L_o(\mathbf{x}, \mathbf{w}) = L_e(\mathbf{x}, \mathbf{w}) + \int_{\Omega} f_r(\mathbf{x}, \mathbf{w}', \mathbf{w}) L_i(\mathbf{x}, \mathbf{w}') (-\mathbf{w}' \cdot \mathbf{n}) d\mathbf{w}'$$



```

100
(depth < MAXD);

nc = inside / 1.0f; // normal cosine
nt = nt / nc; ddo = 1.0f - nt * nt;
cos2t = 1.0f - nt * nt; // not needed
(D, N));
)

at a = nt - nc; b = nt * nc;
at Tr = 1 - (RB + (1 - RB) * cos2t);
Tr) R = (D * nnt - N * (ddo

E * diffuse;
= true;

refl
    && (depth < MAXDEPTH)
D, N
refl * E * diffuse;
= true;

MAXDEPTH)

survive = SurvivalProbability( diffuse );
estimation - doing it properly, clearly
if;
radiance = SampleLight( &rand, I, &L, &align,
N, radiance.y + radiance.z) > 0) && (maxDepth <
w = true;
at brdfPdf = EvaluateDiffuse( L, N ) * Psurvive;
at3 factor = diffuse * INVPI;
at weight = Mix2( directPdf, brdfPdf );
at cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf) * (radiance

random walk - done properly, closely following Monte Carlo
vive)

;
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;

```



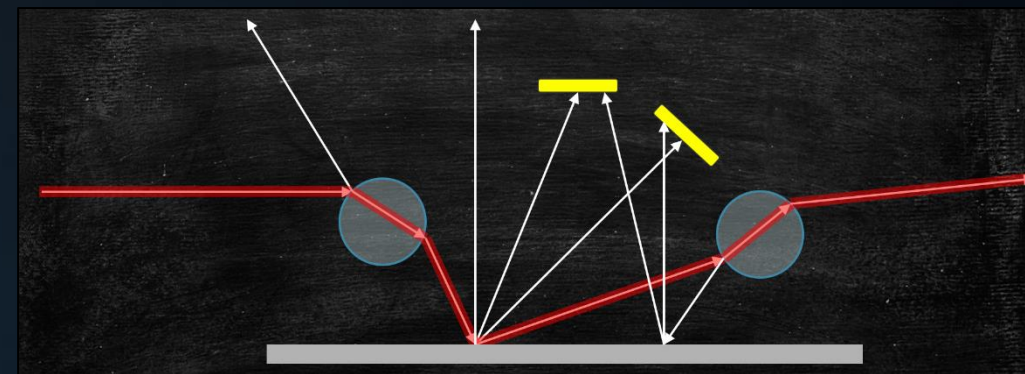


# Physically Based

## Path Tracing

Color Trace( vec3 O, vec3 D )

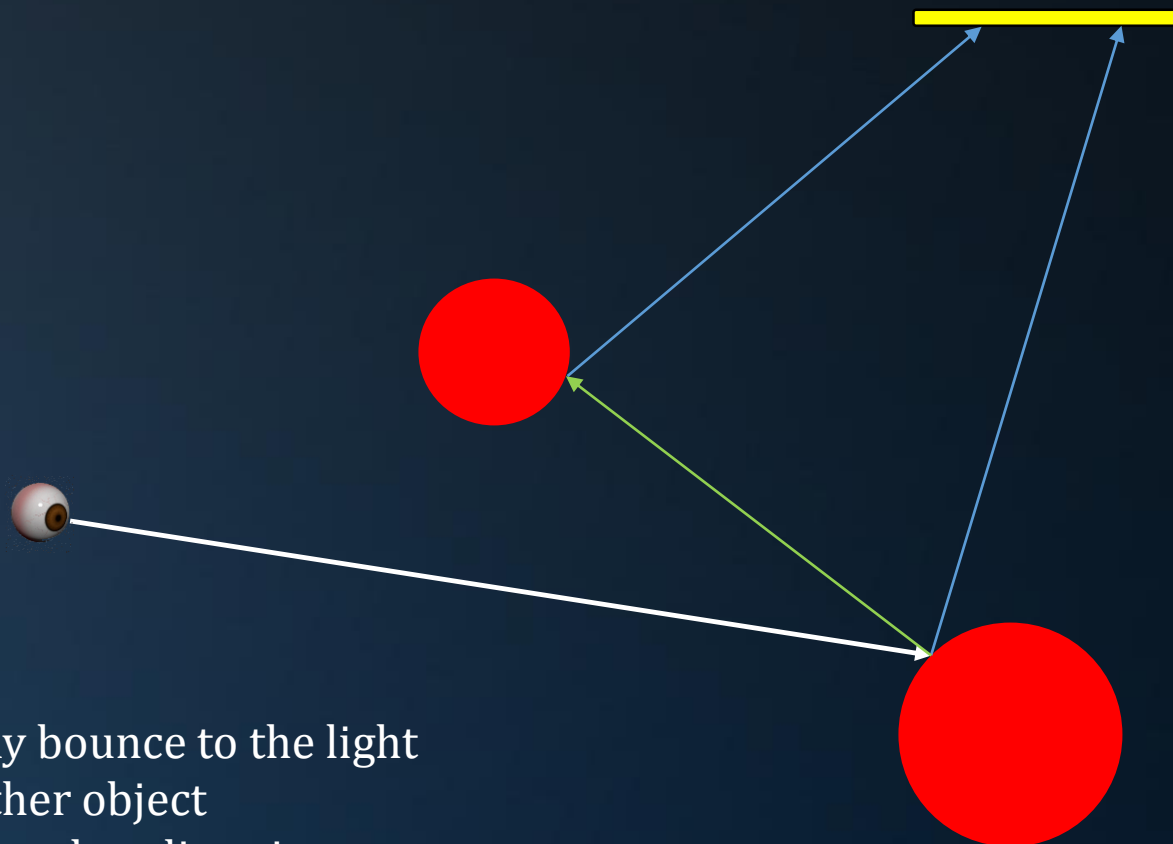
```
{
    I,N,mat = Intersect( O, D );
    if (mat.IsLight()) return mat.emissive;
    vec3 R = RandomReflection( N );
    BRDF = mat.color;
    return BRDF * dot( N, R ) * Trace( I, R );
}
```





# Physically Based

## Ray Tree



- A path may bounce to the light
- Or to another object
- Or in some other direction



1. **PROBLEM STATEMENT**

100





# Today's Agenda:

- Limitations of Whitted-style Ray Tracing
- Monte Carlo
- Path Tracing





# INFOGR – Computer Graphics

J. Bikker - April-July 2015 - Lecture 10: “Ground Truth”

## END of “Ground Truth”

next lecture: “Accelerate”

