Ray Tracing for Games

Dr. Jacco Bikker - IGAD/BUAS, Breda, February 4

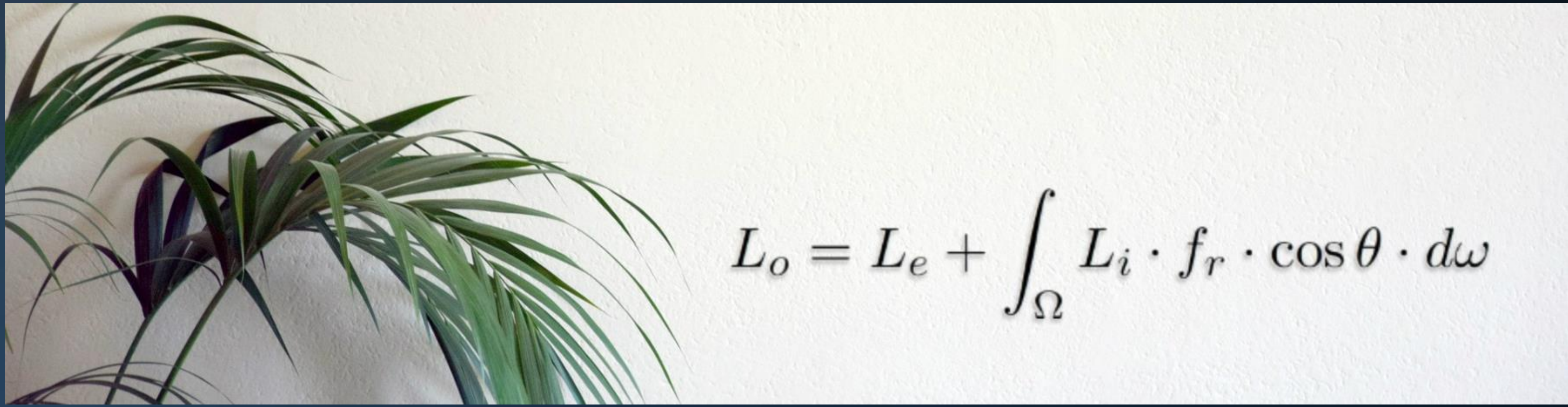
Welcome!

8

Agenda:

- Path Tracing

Previously in Ray Tracing for Games…



Previously in Ray Tracing for Games…

Monte Carlo integration:

Complex integrals can be approximated by replacing them by the expected value of a stochastic experiment.

- Soft shadows: randomly sample the area of a light source;
- Glossy reflections: randomly sample the directions in a cone;
- Depth of field: randomly sample the aperture;
- Motion blur: randomly sample frame time.

In the case of the rendering equation, we are dealing with a *recursive integral*.

Path tracing: evaluating this integral using a *random walk*.

Solving the Rendering Equation

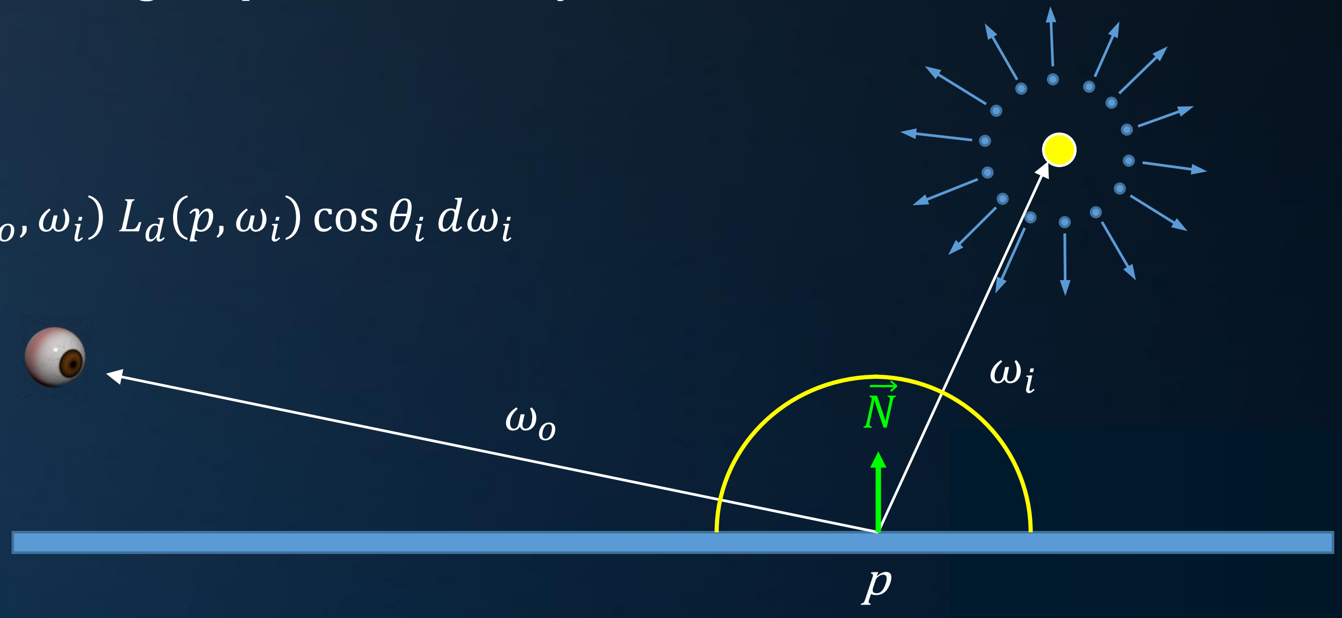
$$L_o(x,\omega_o) = L_E(x,\omega_o) + \int_\Omega f_r(x,\omega_o,\omega_i)\, L_i(x,\omega_i) \cos\theta_i \; d\omega_i$$

Let's start with direct illumination:

For a screen pixel, diffuse surface point p with normal $\vec{N}$ is directly visible.
What is the radiance travelling via p towards the eye?

Answer:

$$L_o(p,\omega_o) = \int_\Omega f_r(p,\omega_o,\omega_i)\, L_d(p,\omega_i) \cos\theta_i \, d\omega_i$$



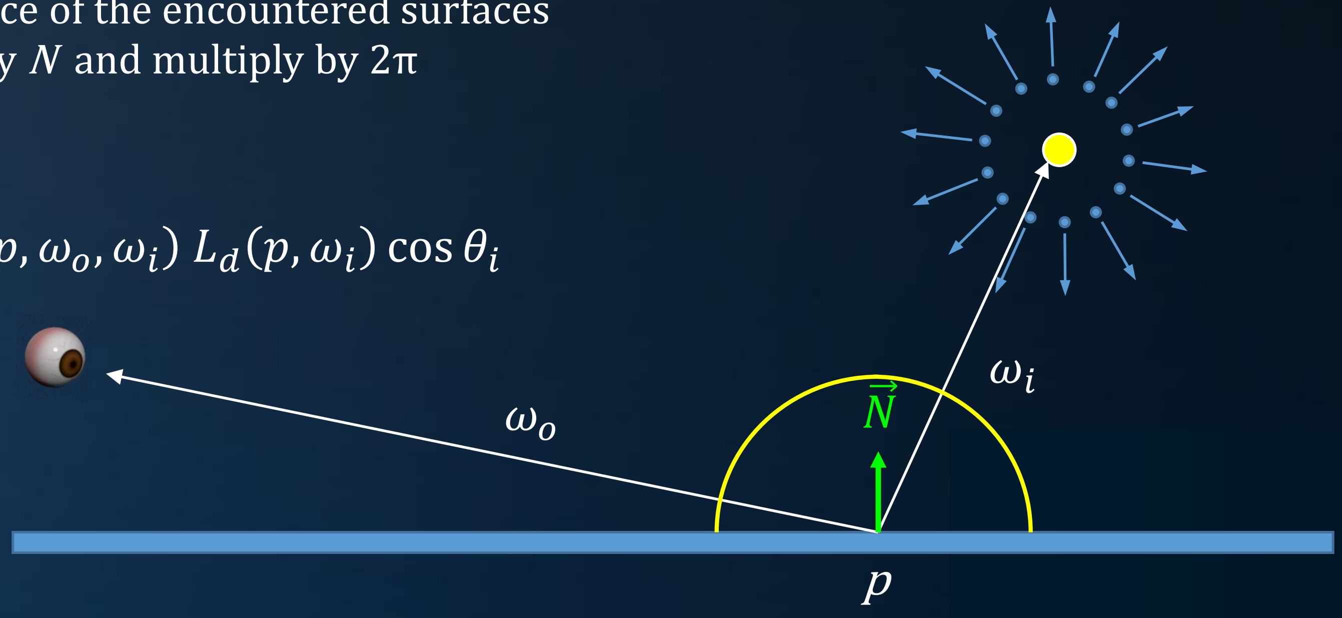
Direct Illumination

$$L_o(p,\omega_o) = \int_\Omega f_r(p,\omega_o,\omega_i)\, L_d(p,\omega_i) \cos\theta_i \, d\omega_i$$

We can solve this integral using Monte-Carlo integration:

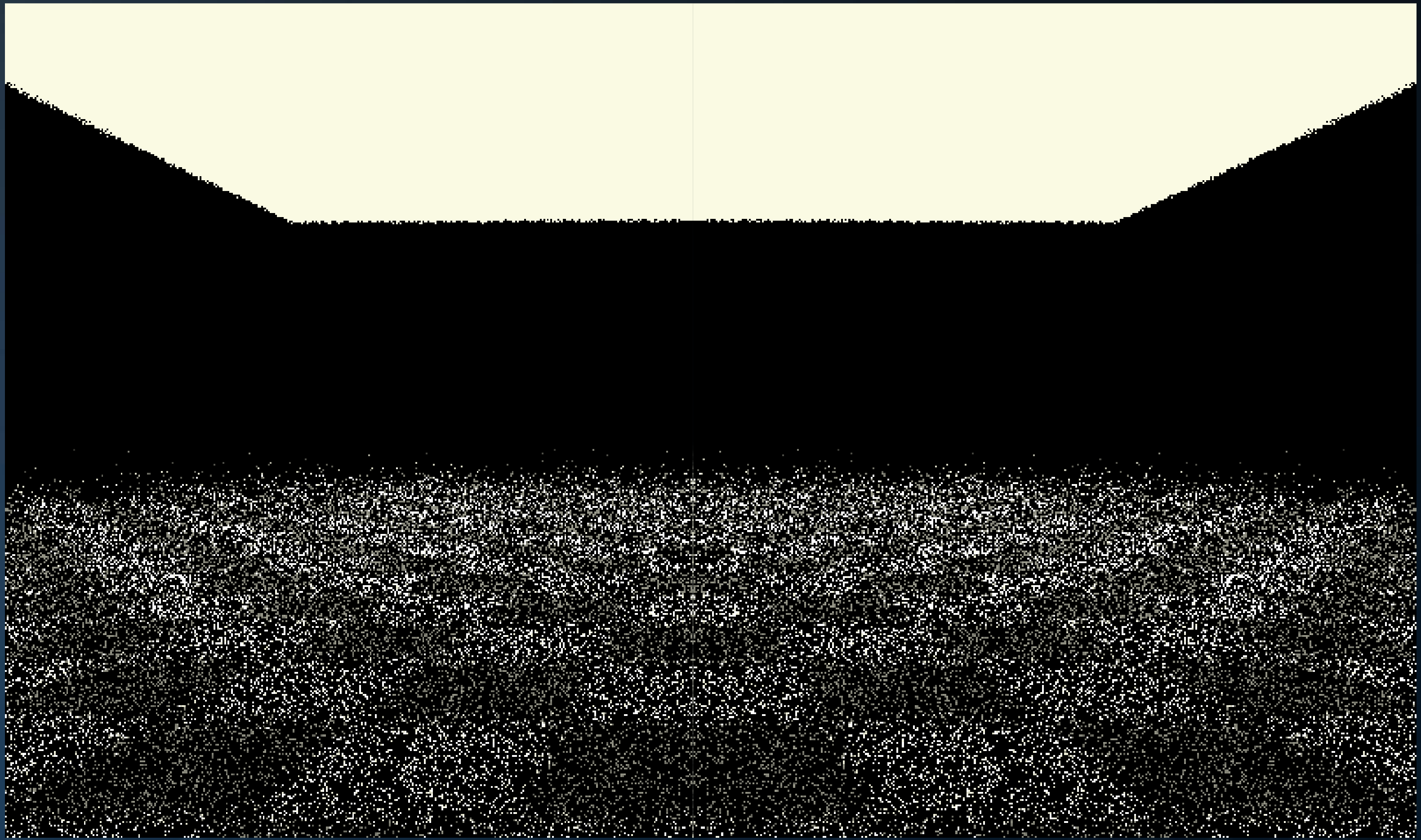
- Chose N random directions over the hemisphere for p
- Find the first surface in each direction by tracing a ray
- Sum the luminance of the encountered surfaces
- Divide the sum by N and multiply by 2π

$$L_o(p,\omega_o) \approx \frac{2\pi}{N} \sum_{i=1}^{N} f_r(p,\omega_o,\omega_i)\, L_d(p,\omega_i) \cos\theta_i$$

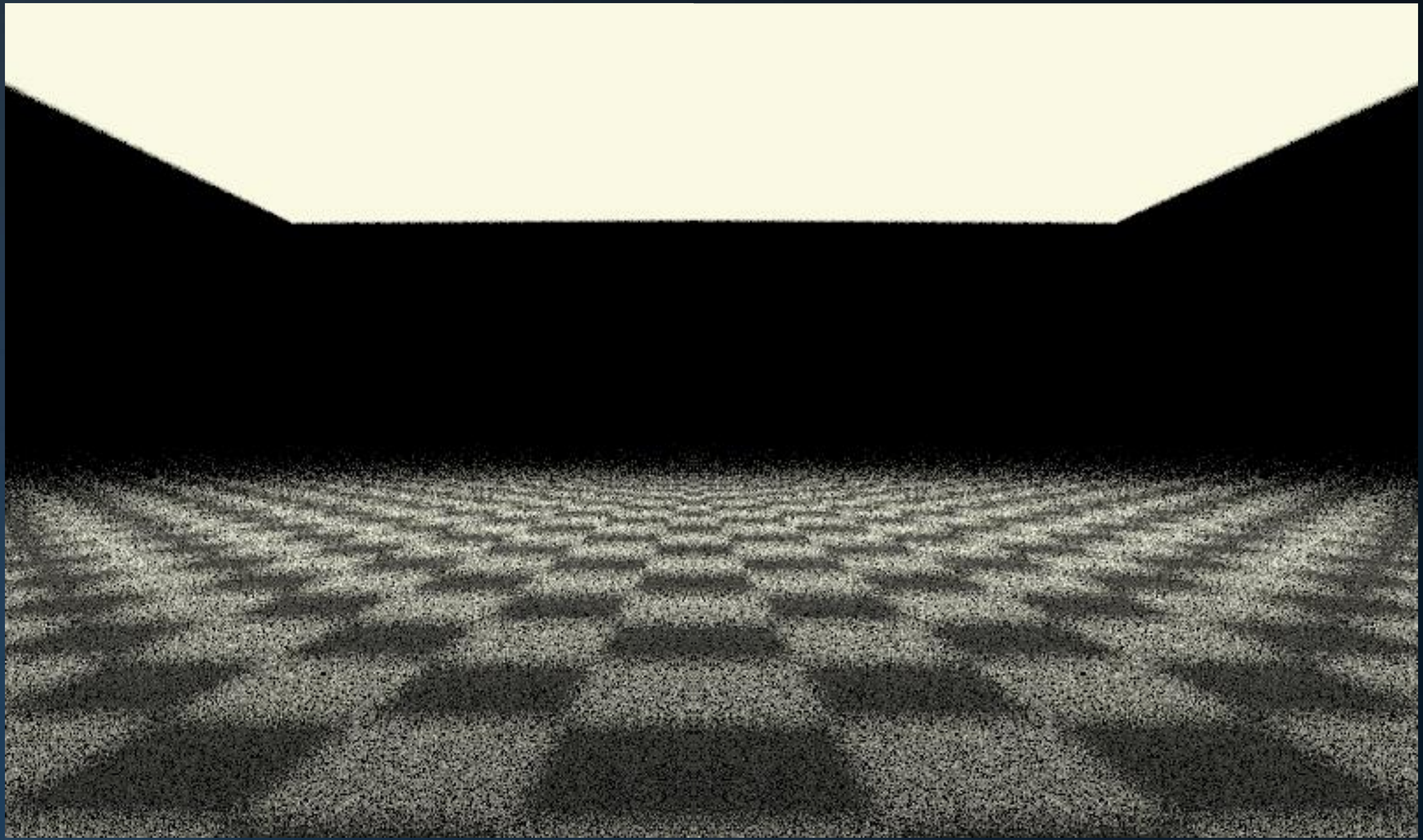


Practical

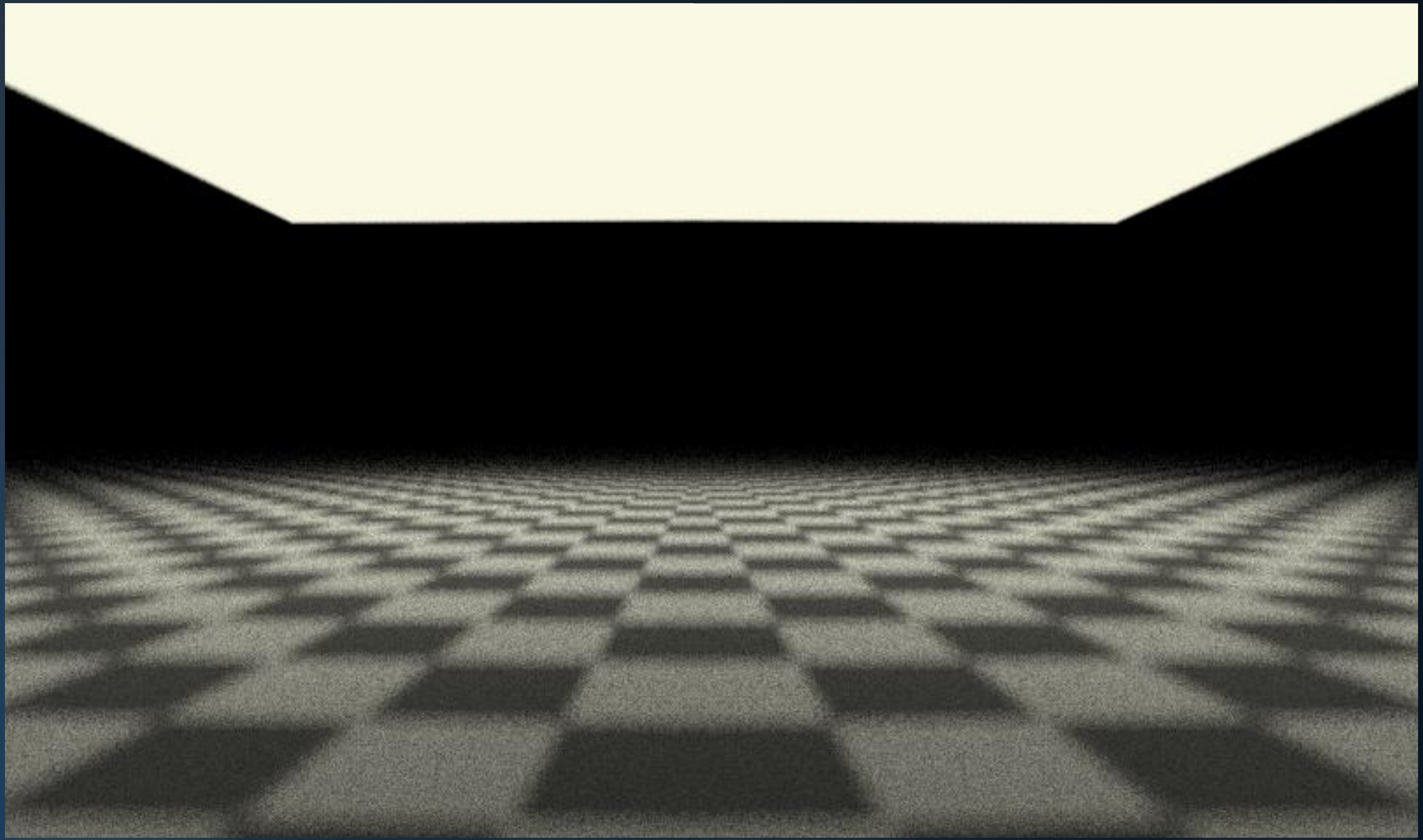
```
float3 R = DiffuseReflection( ray.N );
Ray rayToHemisphere = new Ray( I + R * EPSILON, R, 1e34f );
Scene.Intersect( rayToHemisphere );
if (rayToHemisphere.objIdx == LIGHT)
{
   float3 BRDF = material.color * INVPI;
   return 2.0f * PI * BRDF * lightColor * dot( R, ray.N );
}
```



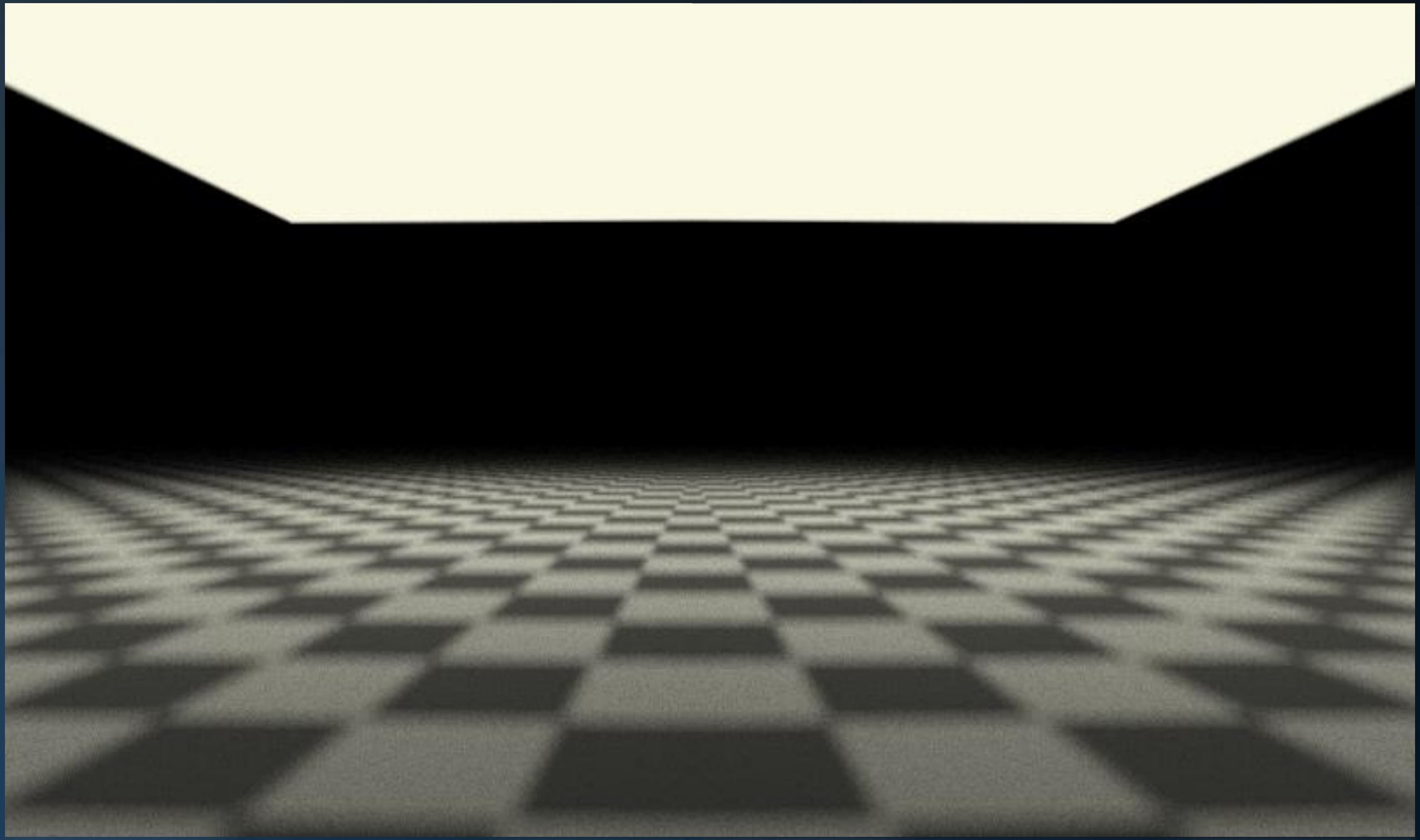
0.1s



0.5s



2.0s



30.0s

Indirect Light

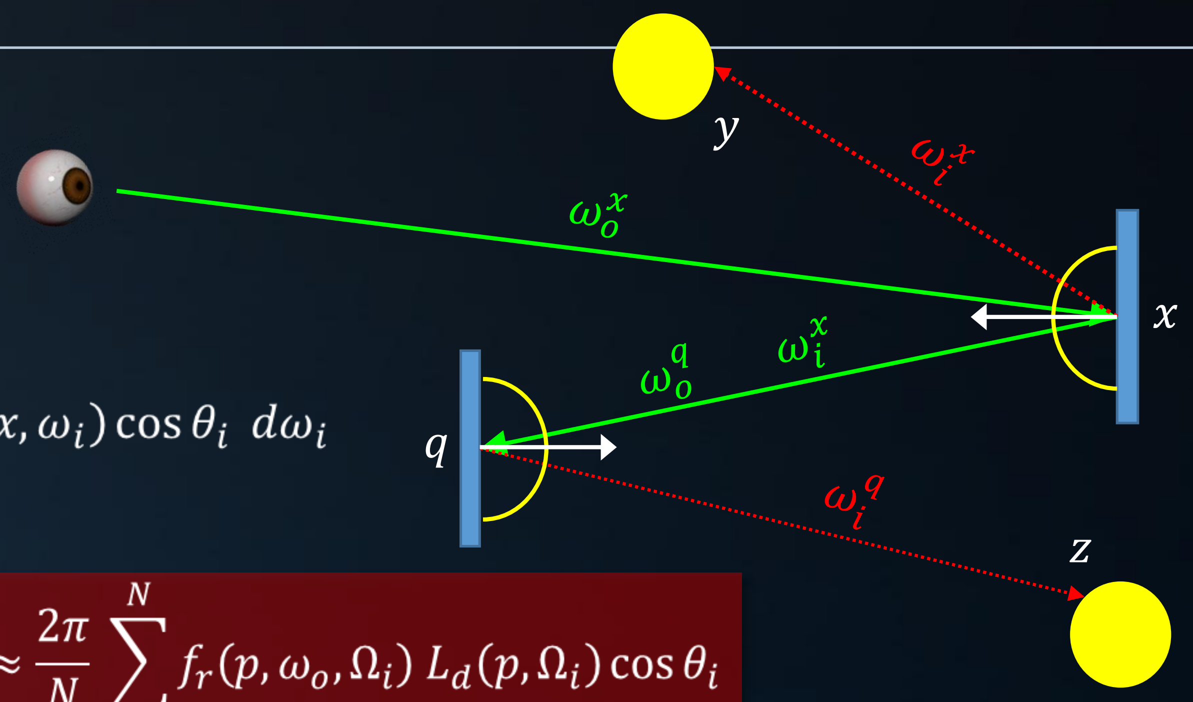
Returning to the full rendering equation:

$$L_o(x,\omega_o) = L_E(x,\omega_o) + \int_\Omega f_r(x,\omega_o,\omega_i)\, L_i(x,\omega_i)\cos\theta_i \; d\omega_i$$

We know how to evaluate direct lighting:

$$L_o(p,\omega_o) = \int_\Omega f_r(p,\omega_o,\omega_i)\, L_d(p,\omega_i)\cos\theta_i \, d\omega_i$$

What remains is indirect light.
This is the light that is not emitted by the surface in direction ω_i, but *reflected*.



Indirect Light

$$L_o(x,\omega_o) = L_E(x,\omega_o) + \int_\Omega f_r(x,\omega_o,\omega_i)\, L_i(x,\omega_i) \cos\theta_i \; d\omega_i$$

Let's expand / reorganize this:

$$L_o(p,\omega_o) \approx \frac{2\pi}{N}\sum_{i=1}^{N} f_r(p,\omega_o,\Omega_i)\, L_d(p,\Omega_i) \cos\theta_i$$

$$L_o(x,\omega_o^x) = L_E(x,\omega_o^x)$$

$$+ \int_\Omega L_E(y,\omega_o^y)\, f_r(x,\omega_o^x,\omega_i^x)\; \cos\theta_i^x \; d\omega_i^x$$

direct light

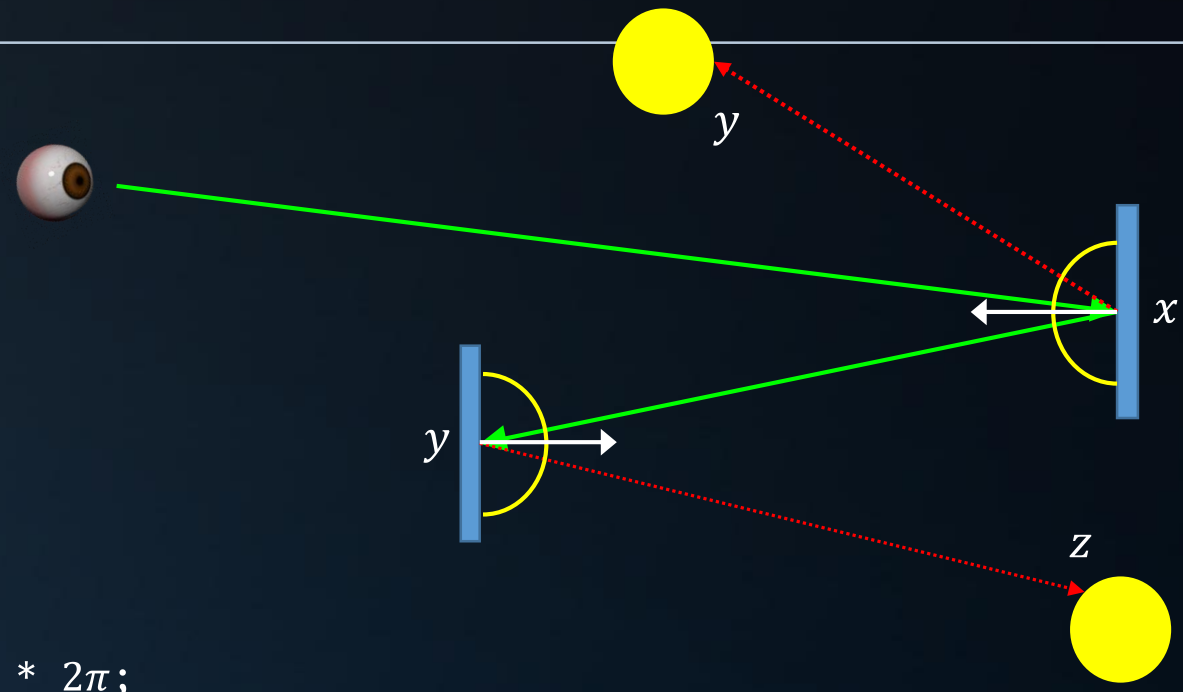
$$+ \int_\Omega \int_\Omega L_E(z,\omega_o^q)\, f_r(y,\omega_o^q,\omega_i^q)\; \cos\theta_i^q \;\; f_r(x,\omega_o^x,\omega_i^x)\; \cos\theta_i^x \; d\omega_i^x \, d\omega_i^q$$

1st bounce

$$+ \int_\Omega \int_\Omega \int_\Omega \ldots$$

2nd bounce

$$\approx \ldots$$



Indirect Light

One particle finding the light via a surface:

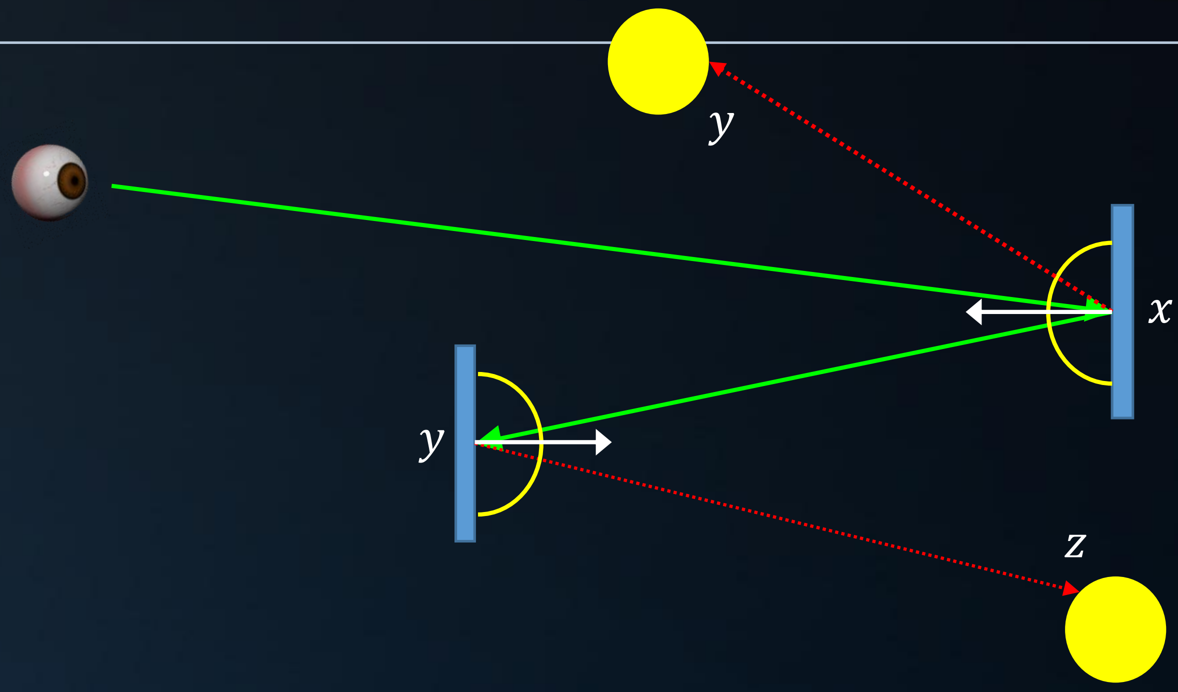
```
I, N = Trace( ray );
R = DiffuseReflection( N );
lightColor = Trace( new Ray( I, R ) );
```

$$\texttt{return dot(R, N)} * \frac{albedo}{\pi} * \texttt{lightColor} * 2\pi;$$

One particle finding the light via two surfaces:

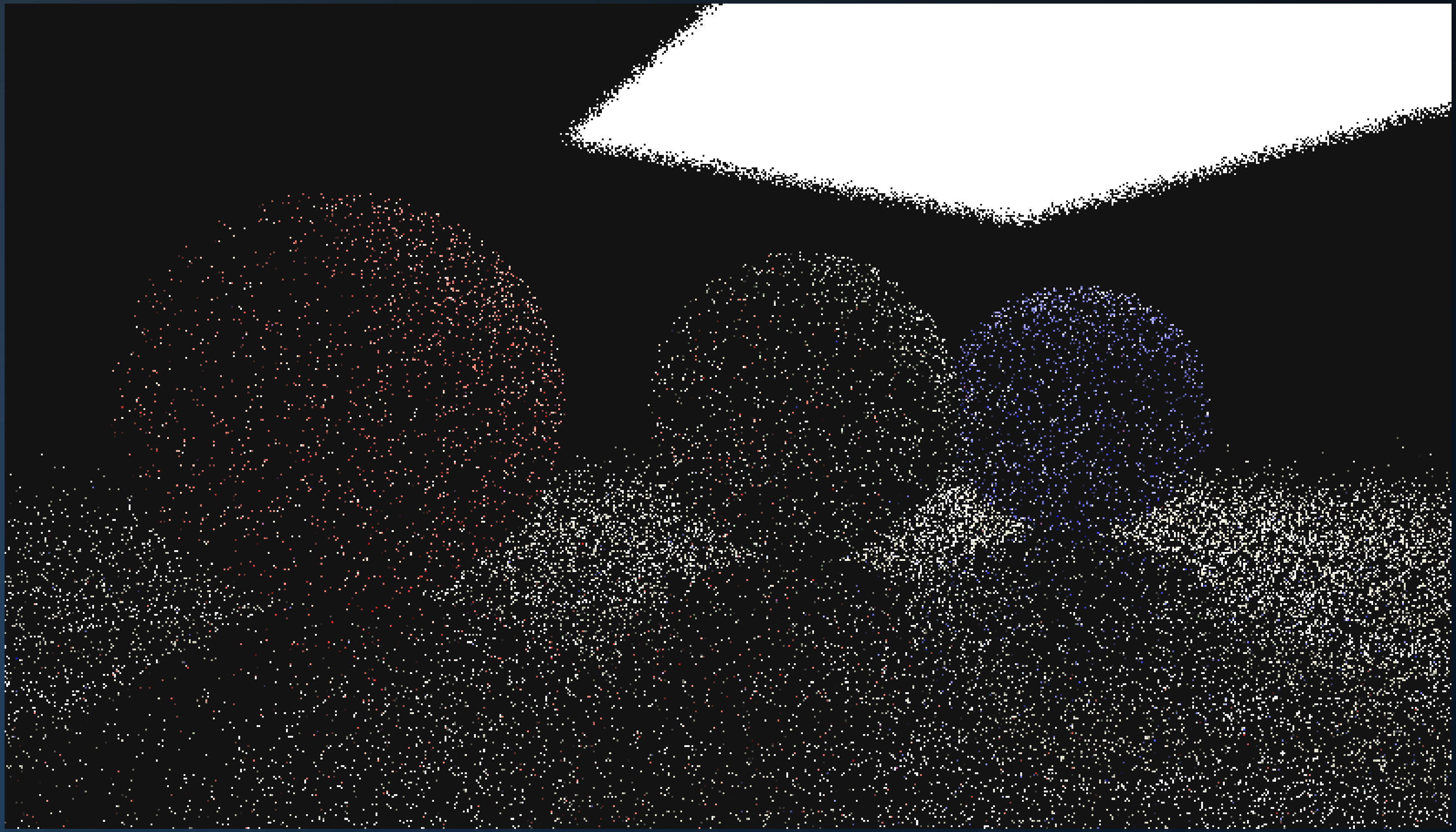
```
I1, N1 = Trace( ray );
R1 = DiffuseReflection( N1 );
I2, N2 = Trace( new Ray( I1, R1 ) );
R2 = DiffuseReflection( N2 );
lightColor = Trace( new Ray( I2, R2 ) );
```

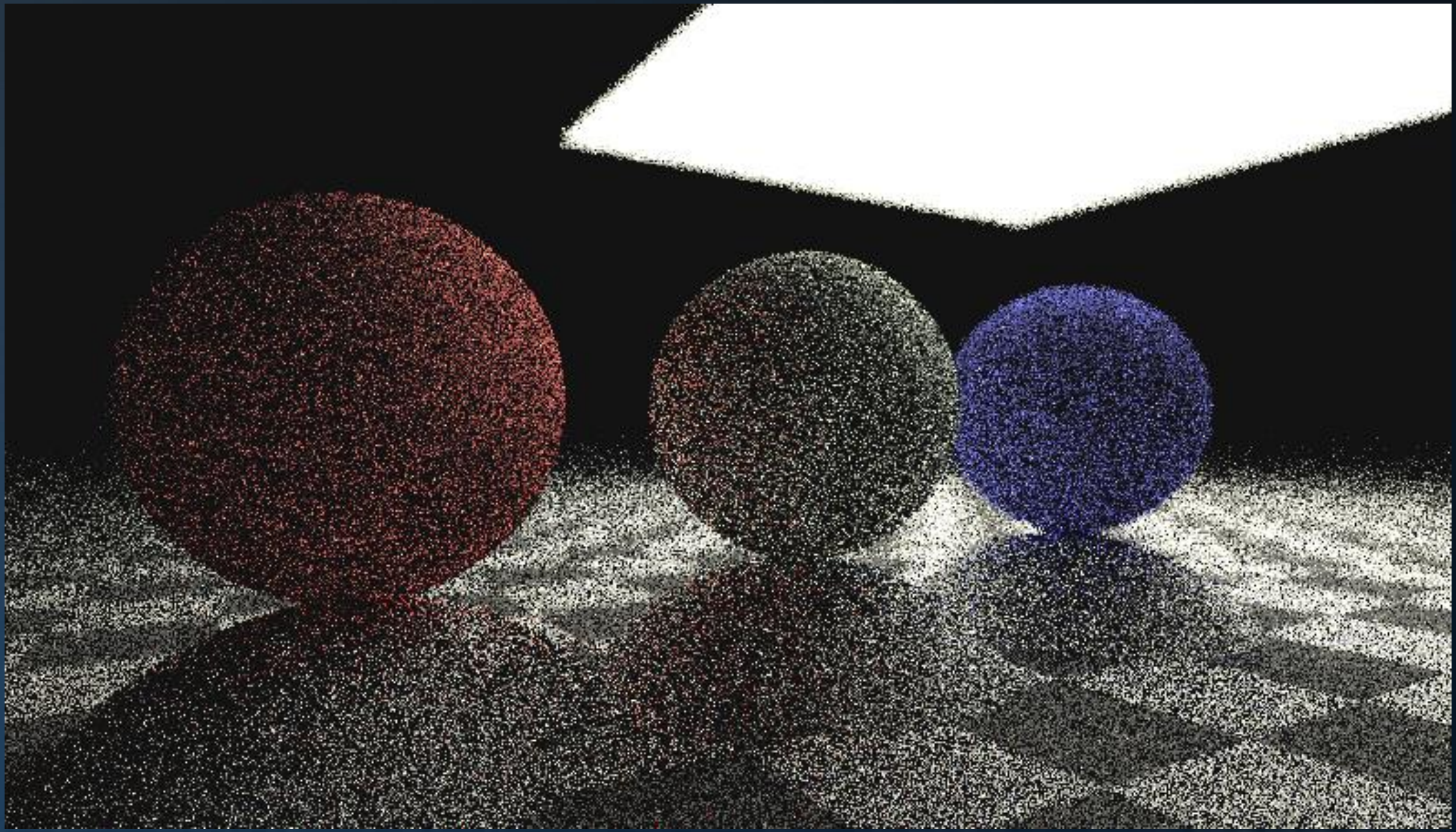
$$\texttt{return dot(R1, N1)} * \frac{albedo}{\pi} * 2\pi * \texttt{dot(R2, N2)} * \frac{albedo}{\pi} * 2\pi * \texttt{lightColor;}$$



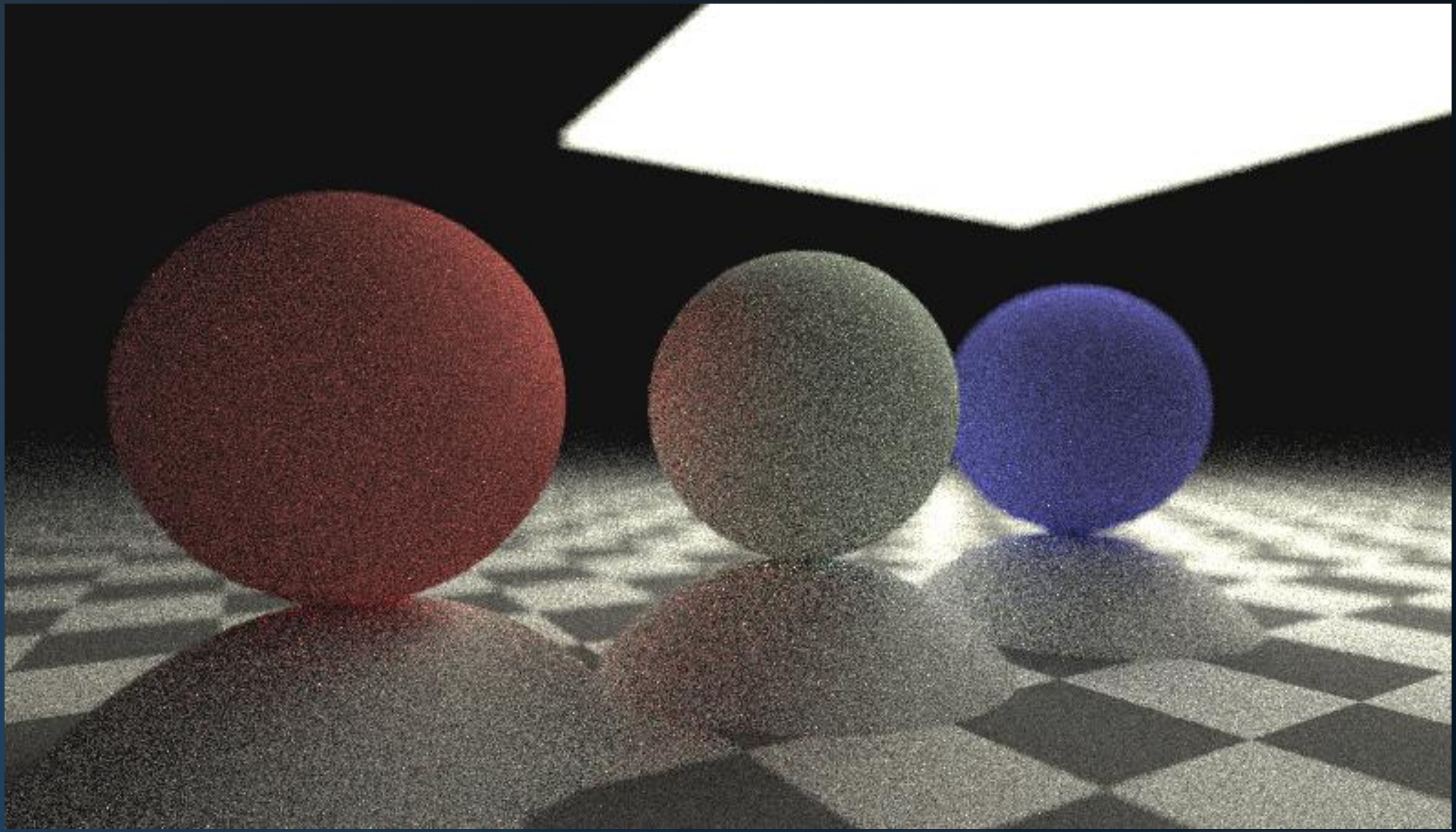
Path Tracing Algorithm

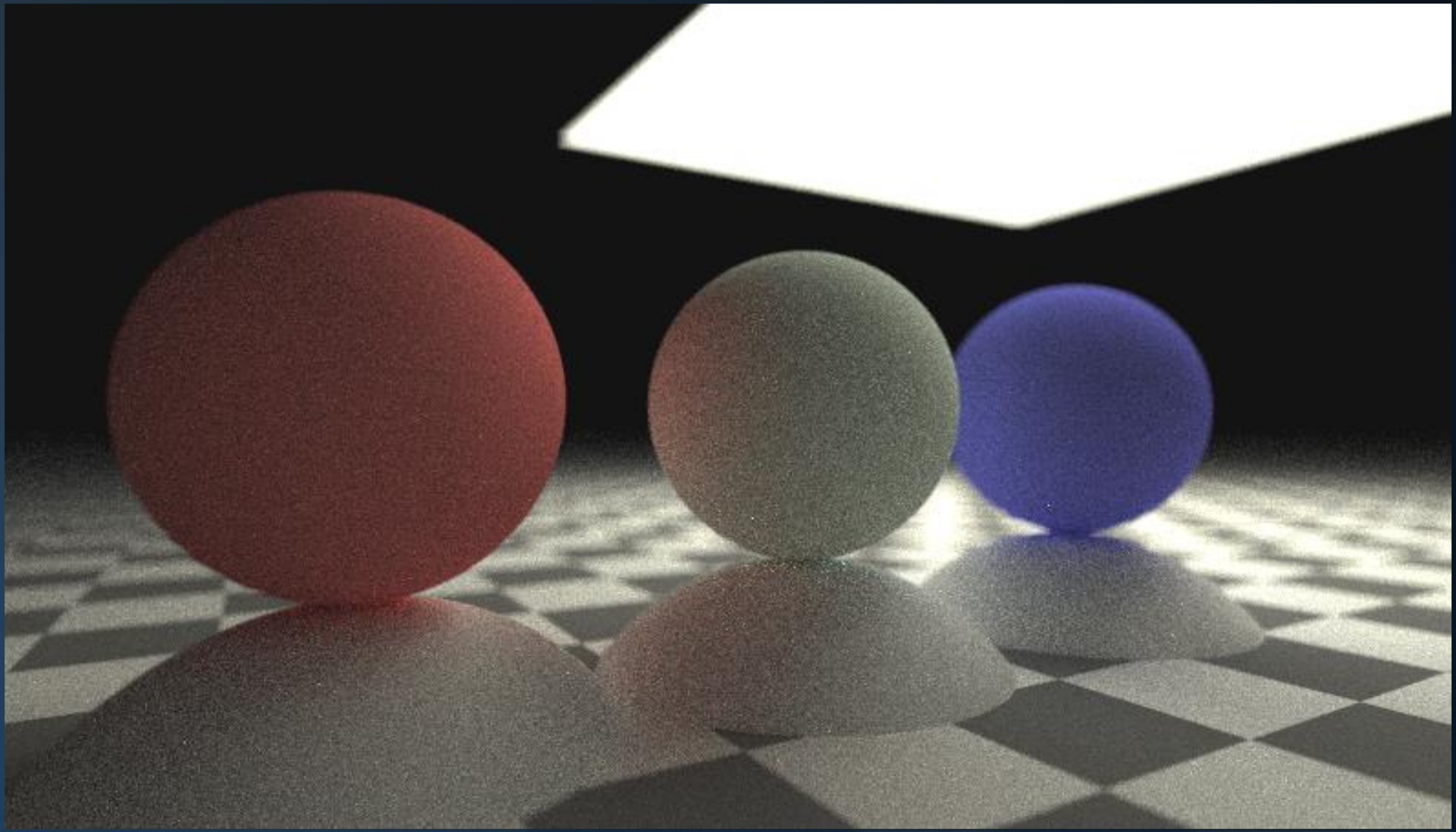
```
Color Sample( Ray ray )
{
    // trace ray
    I, N, material = Trace( ray );
    // terminate if ray left the scene
    if (ray.NOHIT) return BLACK;
    // terminate if we hit a light source
    if (material.isLight) return material.emittance;
    // continue in random direction
    R = DiffuseReflection( N );
    Ray newRay( I, R );
    // update throughput
    BRDF = material.albedo / PI;
    Ei = Sample( newRay ) * dot( N, R ); // irradiance
    return PI * 2.0f * BRDF * Ei;
}
```





Ray Tracing for Games





Particle Transport

The random walk is analogous to particle transport:

- a particle leaves the camera
- at each surface, energy is absorbed proportional to 1-albedo ('surface color')
- at each surface, the particle picks a new direction
- at a light, the path transfers energy to the camera.

```
Color Sample( Ray ray )
{
    // trace ray
    I, N, material = Trace( ray );
    // terminate if ray left the scene
    if (ray.NOHIT) return BLACK;
    // terminate if we hit a light source
    if (material.isLight) return emittance;
    // continue in random direction
    R = DiffuseReflection( N );
    Ray r( I, R );
    // update throughput
    BRDF = material.albedo / PI;
    Ei = Sample( r ) * (N·R);
    return PI * 2.0f * BRDF * Ei;
}
```

Particle Transport - Glass

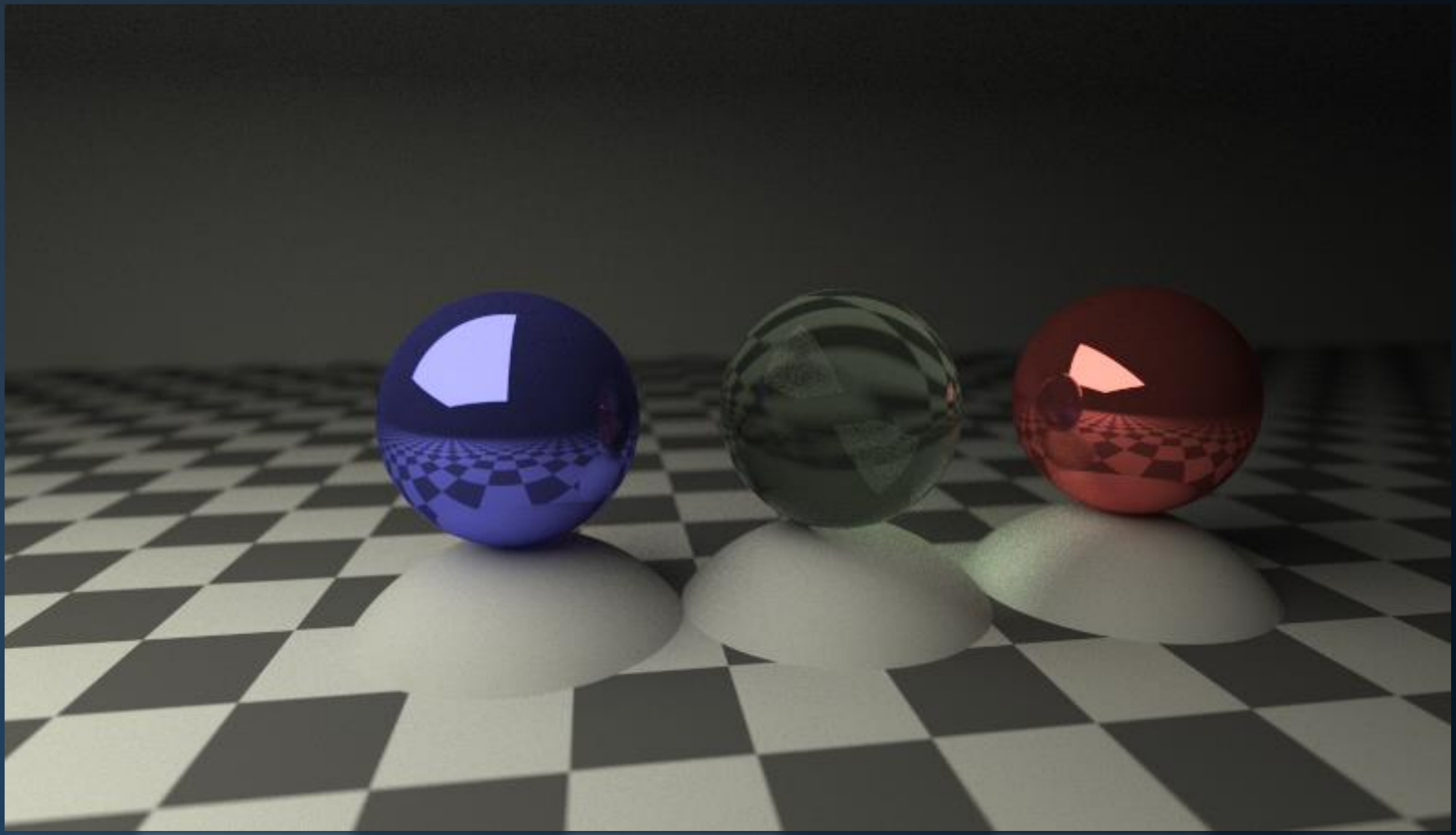
Handling dielectrics:

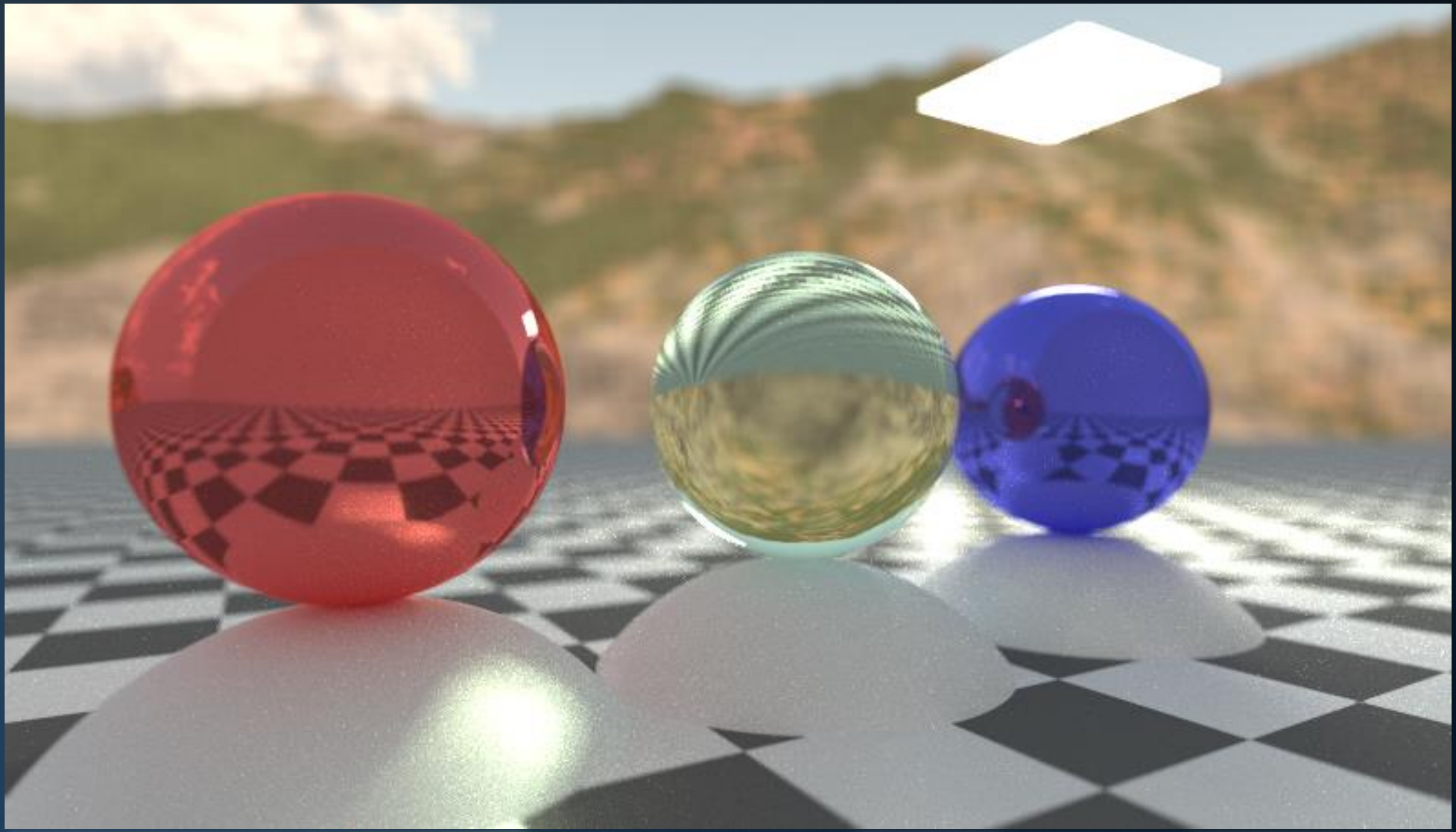
Dielectrics reflect *and* transmit light.
In the ray tracer, we handled this using two rays.

A particle must chose.

The probability of each choice is calculated using the Fresnel equations.

```
Color Sample( Ray ray )
{
    // trace ray
    I, N, material = Trace( ray );
    // terminate if ray left the scene
    if (ray.NOHIT) return BLACK;
    // terminate if we hit a light source
    if (material.isLight) return emittance;
    // surface interaction
    if (material.isMirror)
    {
        // continue in fixed direction
        Ray r( I, Reflect( N ) );
        return material.albedo * Sample( r );
    }
    // continue in random direction
    R = DiffuseReflection( N );
    BRDF = material.albedo / PI;
    Ray r( I, R );
    // update throughput
    Ei = Sample( r ) * (N·R);
    return PI * 2.0f * BRDF * Ei;
}
```





```
rics
& (depth < MAXDEPTH)

c = inside ? 1 : 1.2
nt = nt / nc, ddn = 
os2t = 1.0f - nnt * nn
D, N );
0)

at a = nt - nc, b = nt + n
at Tr = 1 - (R0 + (1 - R0
Tr) R = (D * nnt - N * (ddn

E * diffuse;
= true;

efl + refr)) && (depth < MAXDEPTH)

D, N );
refl * E * diffuse;
= true;

MAXDEPTH)

survive = SurvivalProbability( diffuse
estimation - doing it properly, closely
df;
radiance = SampleLight( &rand, I, &L, &lightD
e.x + radiance.y + radiance.z) > 0) && (dot( N

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) * (radia

andom walk - done properly, closely following Sm
vive)

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

End of PART 8.