

```
ics
& (depth < MAXDEPTH)
{
    if (nt < 0)
        nt = inside ? 1 : -1;
    nt = nt / nc; ddn = dot(N, R);
    cos2t = 1.0f - nnt * nnt;
    D, N );
    )
    {
        at a = nt - nc, b = nt + nc;
        at Tr = 1 - (R0 + (1 - R0) * sqrt(r));
        Tr) R = (D * nnt - N * (ddn < 0 ? 1 : -1));
    }
    E * diffuse;
    = true;
    -
    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, &lightPdf );
        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 Small's
        vive)
        ;
        at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
        survive;
        pdf;
        n = E * brdf * (dot( N, R ) / pdf);
        sion = true;
    }
}
```

8

Ray Tracing for Games

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

Welcome!



```
ics
& (depth < MAXDEPTH))
{
    if (nt < nc)
    {
        nt = inside ? 1 + 1.2f * nc : 1.0f;
        nt = nt / nc; ddn = dot(N, N);
        r2s2t = 1.0f - nnt * nnt;
        D, N );
    }
    else
    {
        at a = nt - nc, b = nt + nc;
        at Tr = 1 - (R0 + (1 - R0) * r2s2t);
        Tr) R = (D * nnt - N * (ddn > 0) ? 1 : -1);

        E * diffuse;
        = true;

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

            MAXDEPTH)
            survive = SurvivalProbability( diffuse );
            estimation - doing it properly, closely following Small's
            if;
            radiance = SampleLight( &rand, I, &L, &lightPdf );
            e.x + radiance.y + radiance.z) > 0) && (dot(N, L) > 0)
            {
                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.x + radiance.y + radiance.z) > 0)
            }

            random walk - done properly, closely following Small's
            (survive)
            ;
            at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
            survive;
            pdf;
            n = E * brdf * (dot( N, R ) / pdf);
            sion = true;
        }
    }
}
```

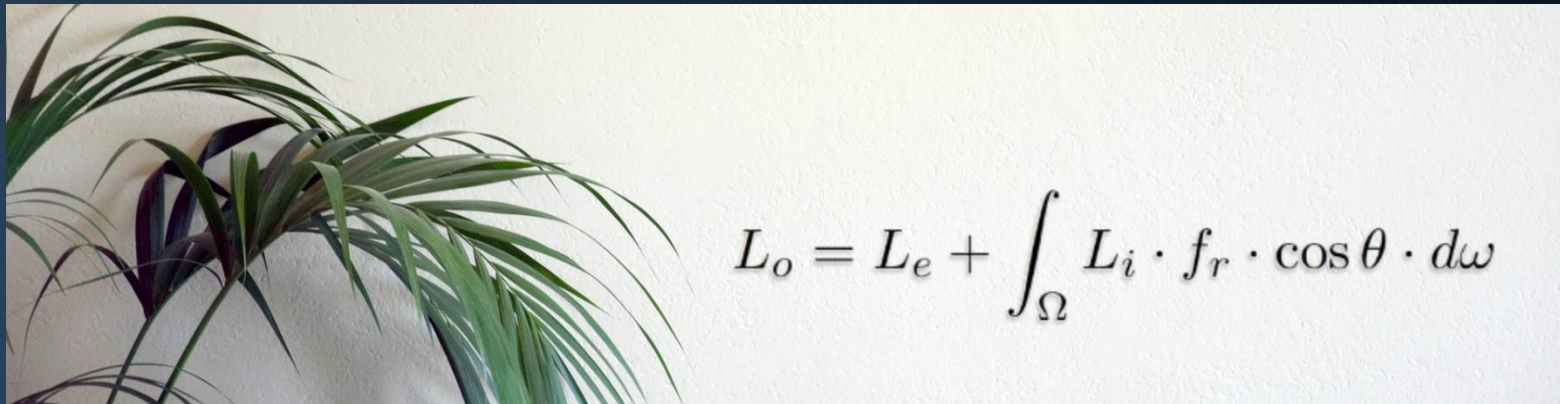
Agenda:

- Path Tracing



Previously in Ray Tracing for Games...

```
ics
& (depth < MAXDEPTH)
{
    if (nt < 0)
    {
        nt = inside ? 1.0f : 0.0f;
        nc = nt / nc; ddn = dot(N, D);
        if (ddn < 0) ddn = -ddn;
        cos2t = 1.0f - nnt * nnt;
        t = sqrt(cos2t);
        D = N * ddn + D * t;
        R = (D * nnt - N * (ddn < 0 ? 1 : -1));
        E * diffuse;
        = true;
        refrfl + refr)) && (depth < MAXDEPTH)
        {
            D, N );
            refrl * E * diffuse;
            = true;
        }
    }
    MAXDEPTH)
    survive = SurvivalProbability( diffuse );
    estimation - doing it properly, closely following
    if;
    radiance = SampleLight( &rand, I, &L, &lightPos );
    e.x + radiance.y + radiance.z) > 0) && (depth < MAXDEPTH)
    {
        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 Small's
    vive)
    ;
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    sion = true;
}
```



Ray Tracing for Games

Previously in Ray Tracing for Games...

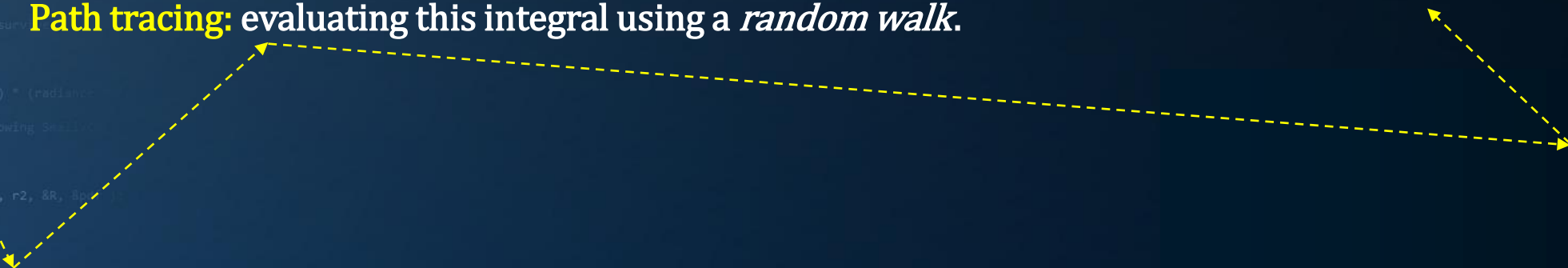
Monte Carlo integration:

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

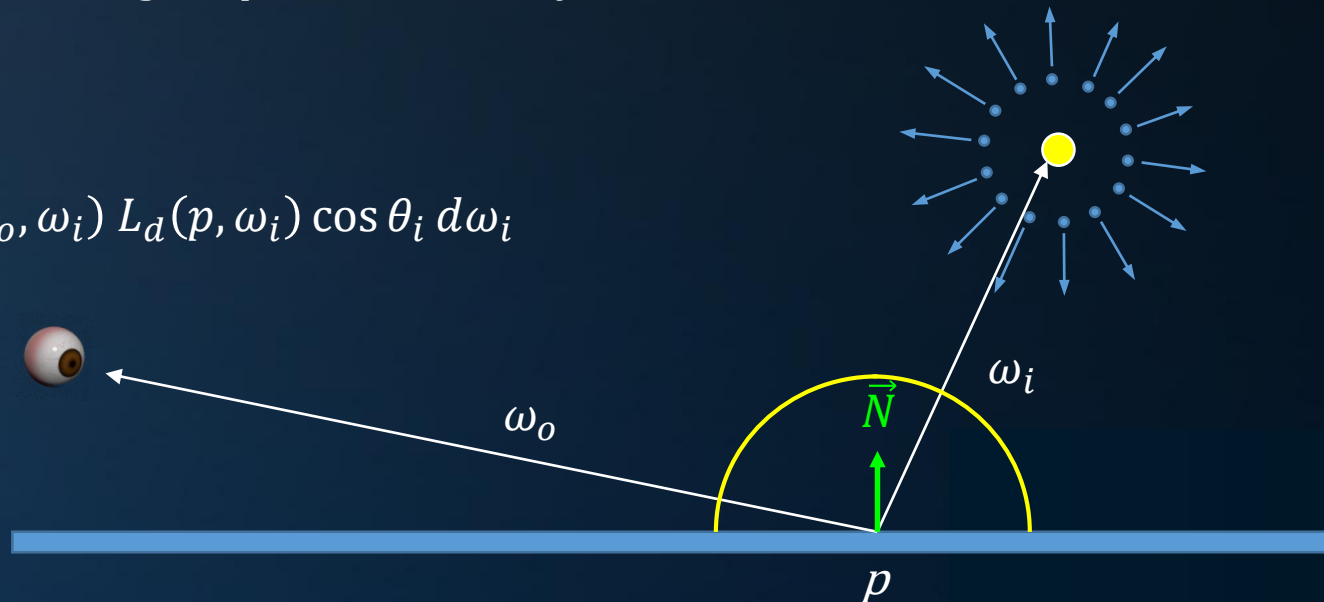
$$L_o(x, \omega_o) = L_E(x, \omega_o) + \int_{\Omega} f_r(x, \omega_o, \omega_i) L_i(x, \omega_i) \cos \theta_i d\omega_i$$

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:

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



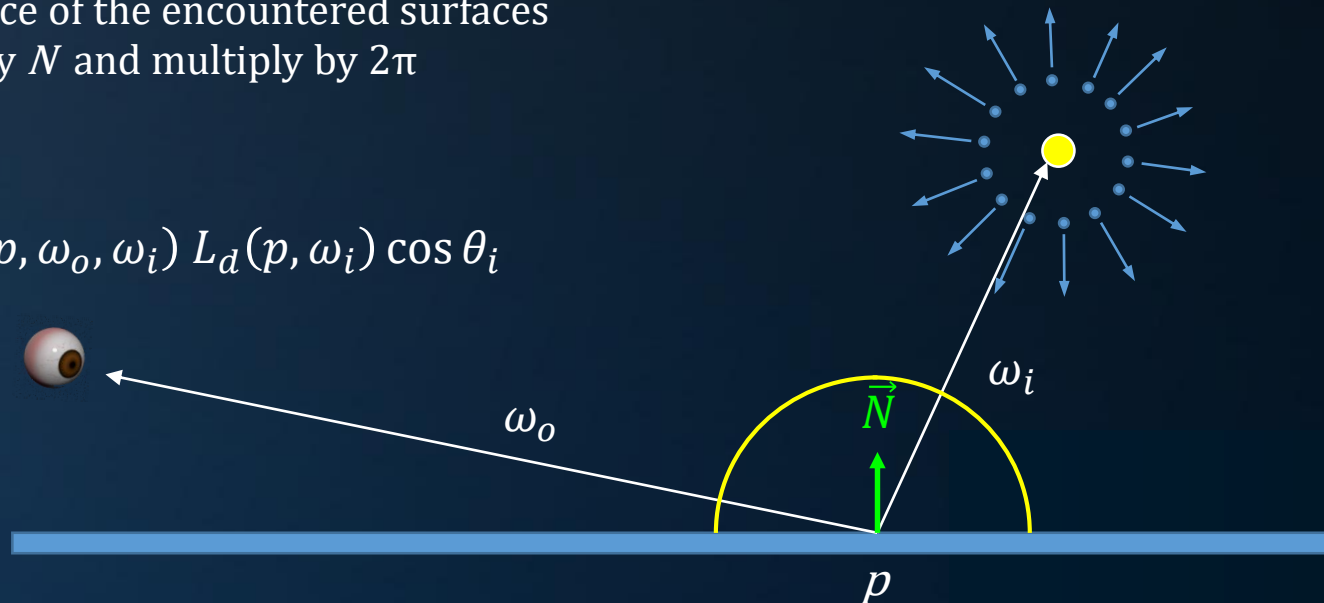
Direct Illumination

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

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

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



Ray Tracing for Games

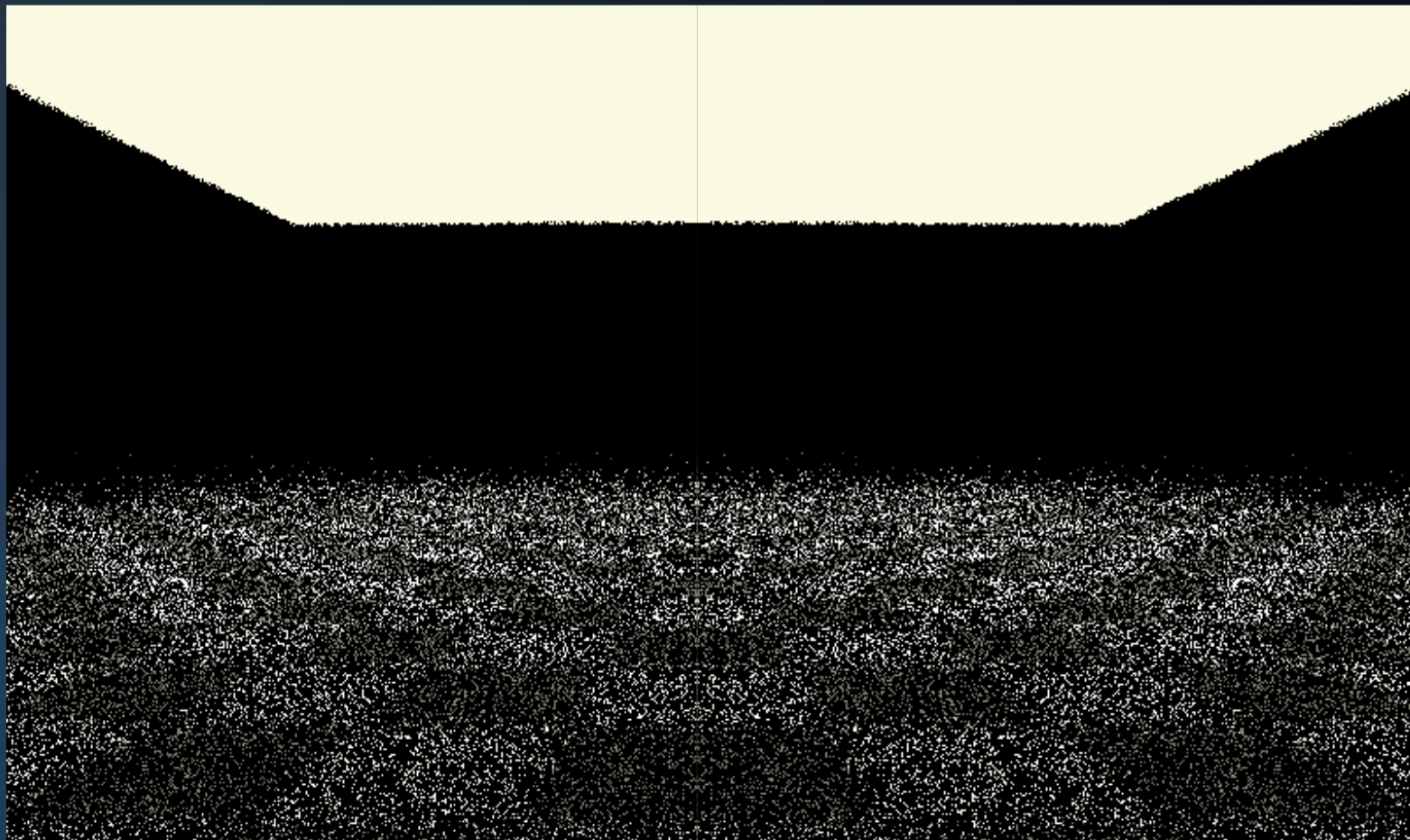
Practical

```
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 );
}
```

Ray Tracing for Games

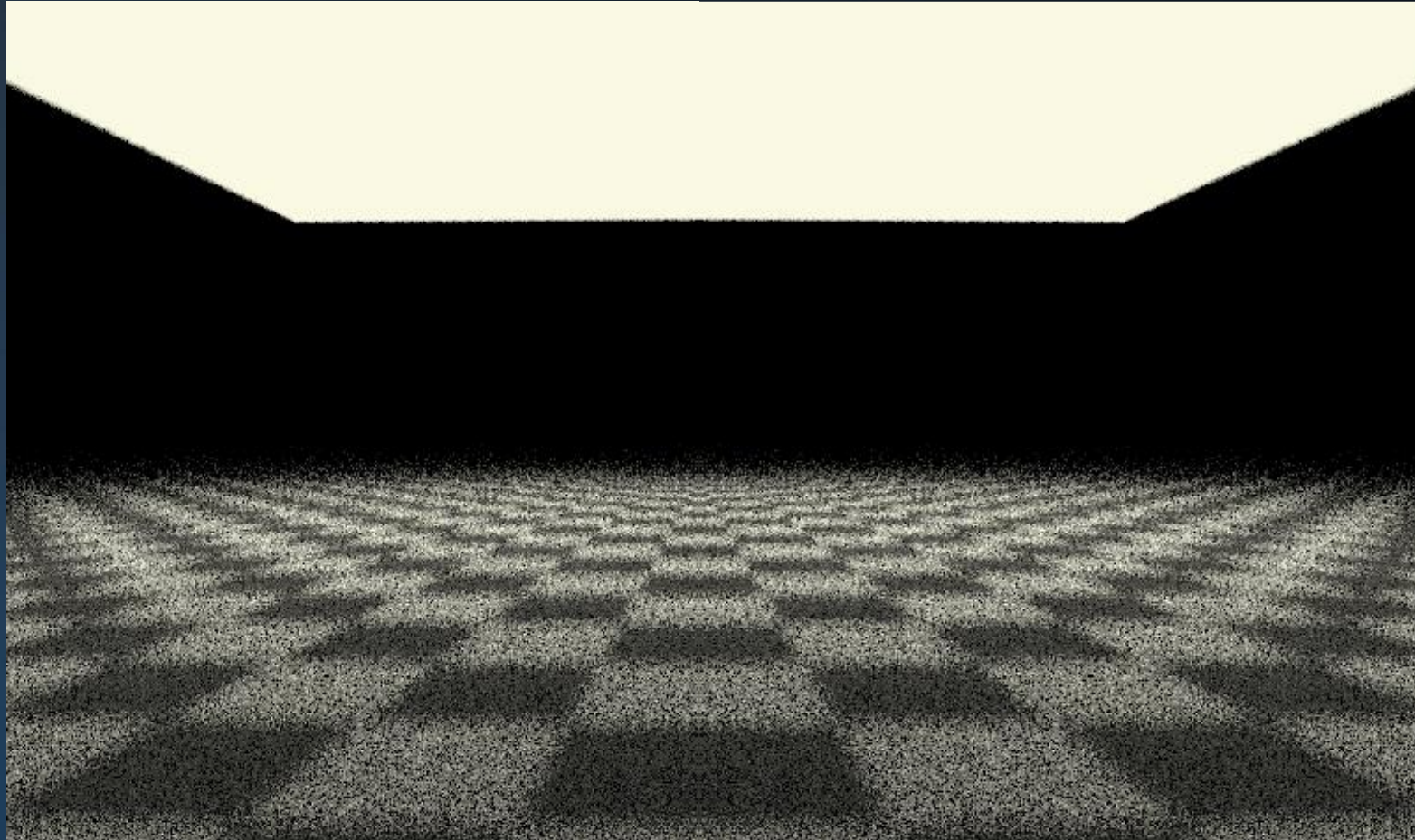
0.1s

```
ics
& (depth < MAXDEPTH)
{
    if ( ! inside ) {
        nt = nt / nc; ddn = ddn * ddn;
        float r = 1.0f - nnt * ddn;
        float r0 = 0.25f;
        float D, N );
    }
    float a = nt - nc, b = nt + nc;
    float Tr = 1 - (R0 + (1 - R0) * r);
    float R = (D * nnt - N * (ddn * r0));
    E * diffuse;
    = true;
    =
    refl + refr)) && (depth < MAXDEPTH)
    {
        D, N );
        refl * E * diffuse;
        = true;
    }
    MAXDEPTH)
    survive = SurvivalProbability( diffuse );
    estimation - doing it properly, closely following
    df;
    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;
}
```



Ray Tracing for Games

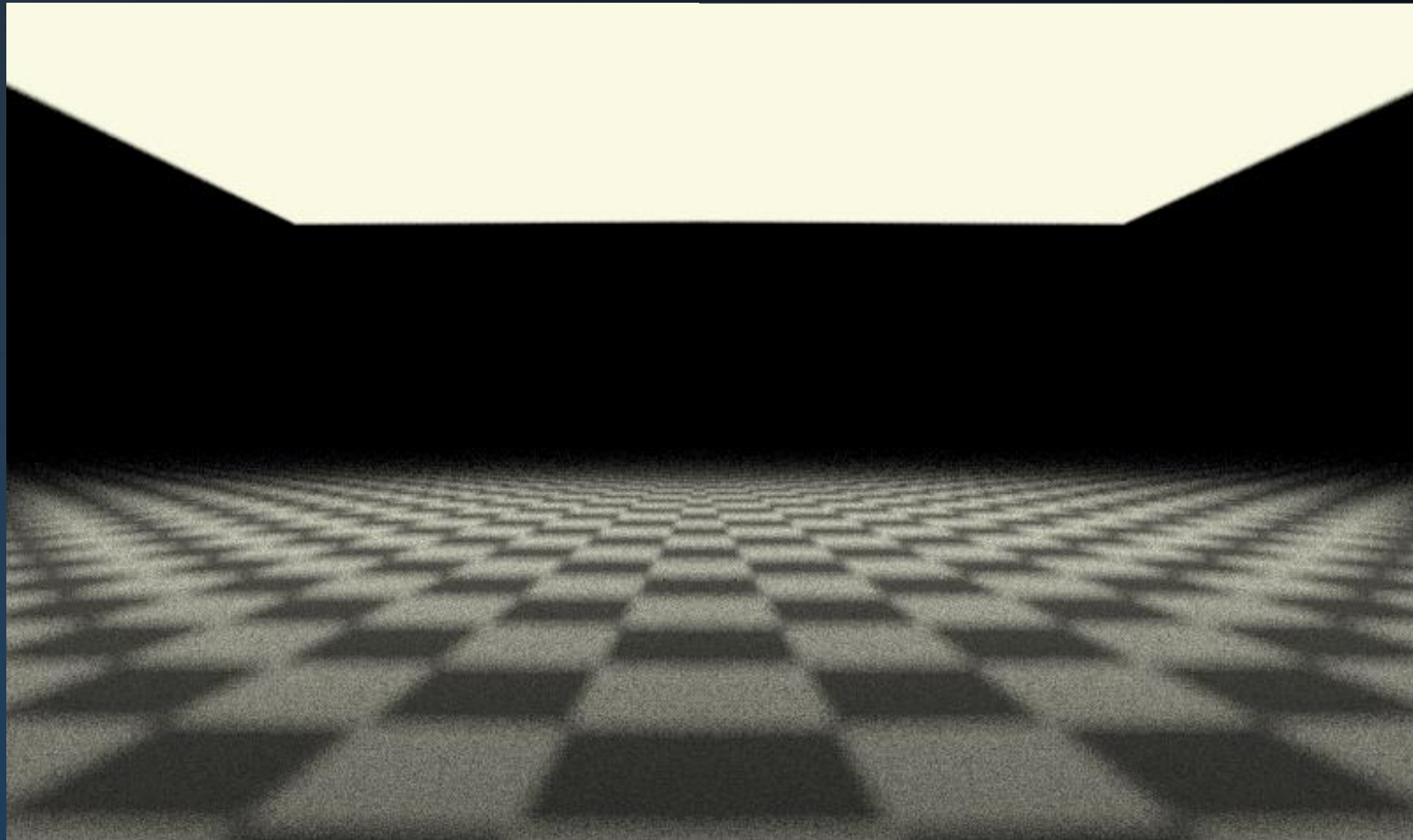
0.5s



Ray Tracing for Games

2.0s

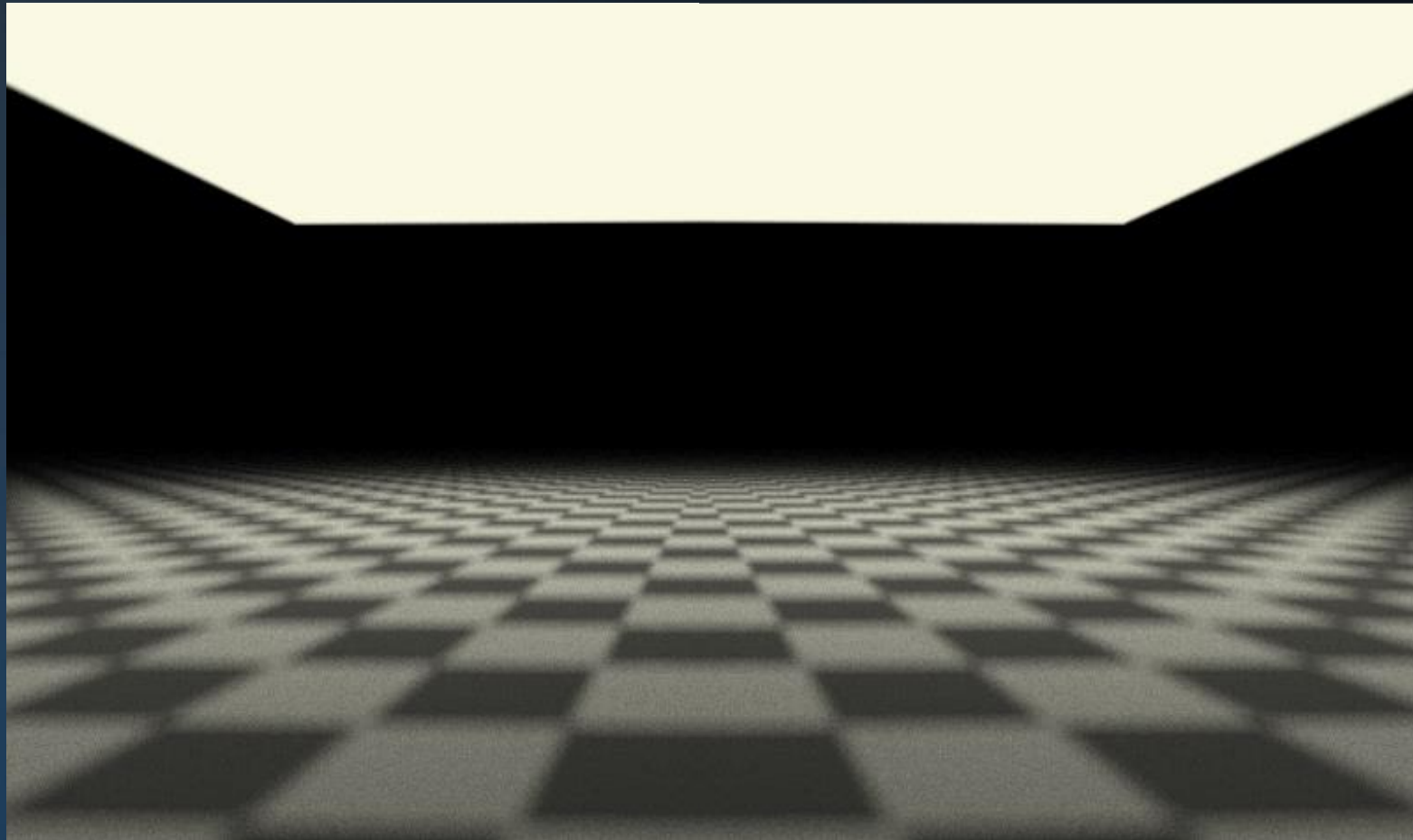
```
ics
& (depth < MAXDEPTH)
{
    if (nt < 0)
        nt = inside ? 1 : -1;
    nc = nt / nc; ddn = dot(N, D);
    if (ddn < 0)
        nnt = 1.0f - nnt * nnt;
    else
        nnt = 1.0f - nnt * nnt;
    D, N );
    (0);
    {
        at a = nt - nc, b = nt + nc;
        at Tr = 1 - (R0 + (1 - R0) * nnt);
        (Tr) R = (D * nnt - N * (ddn < 0 ? 1 : -1));
        E * diffuse;
        = true;
        -
        refl + refr)) && (depth < MAXDEPTH)
        {
            D, N );
            refl * E * diffuse;
            = true;
        }
    }
    MAXDEPTH)
    survive = SurvivalProbability( diffuse );
    estimation - doing it properly, closely following
    df;
    radiance = SampleLight( &rand, I, &L, &lightPos );
    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 Small's
    vive)
    ;
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    sion = true;
}
```



Ray Tracing for Games

30.0s

```
ics
& (depth < MAXDEPTH)
{
    if (nt < 0)
        nt = inside ? 1 : 0.25;
    nc = nt / nt; ddn = 1 - nc;
    r2 = rand(0, 1); r1 = rand(0, 1);
    cos2t = 1.0f - nnt * r1;
    t = sqrt(cos2t);
    D, N );
    )
    {
        at a = nt - nc, b = nt + nc;
        at Tr = 1 - (R0 + (1 - R0) * r1);
        Tr) R = (D * nnt - N * (ddn * r1 +
    E * diffuse;
    = true;
    -
    refl + refr)) && (depth < MAXDEPTH)
    D, N );
    refl * E * diffuse;
    = true;
    MAXDEPTH)
    survive = SurvivalProbability( diffuse );
    estimation - doing it properly, closely following
    df;
    radiance = SampleLight( &rand, I, &L, &lightDir,
    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 Small's
    ve)
    ;
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    ion = true;
}
```



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:

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

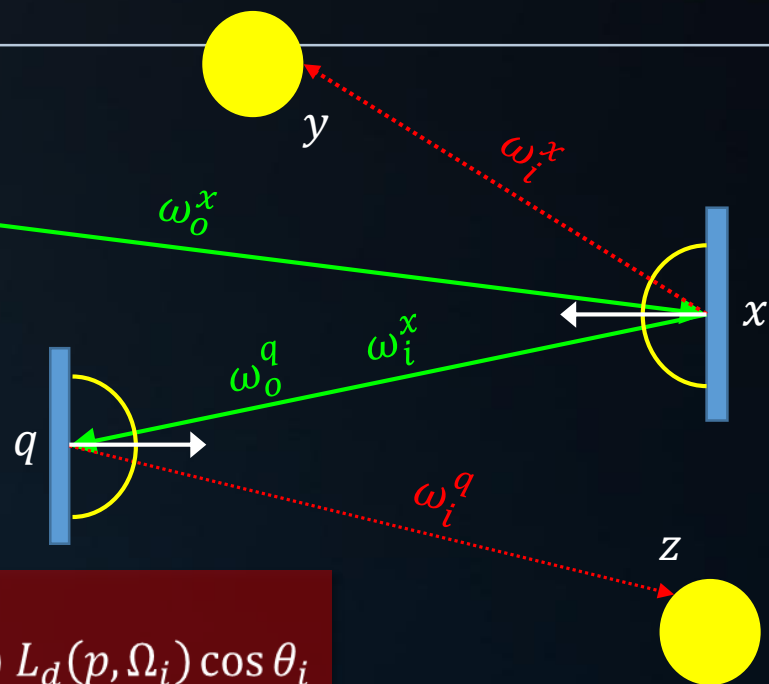
$$L_o(p, \omega_o) \approx \frac{2\pi}{N} \sum_{i=1}^N f_r(p, \omega_o, \Omega_i) L_d(p, \Omega_i) \cos \theta_i$$

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

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

$$+ \int_{\Omega} \int_{\Omega} \int_{\Omega} \dots$$

$\approx \dots$



direct light

1st bounce

2nd bounce



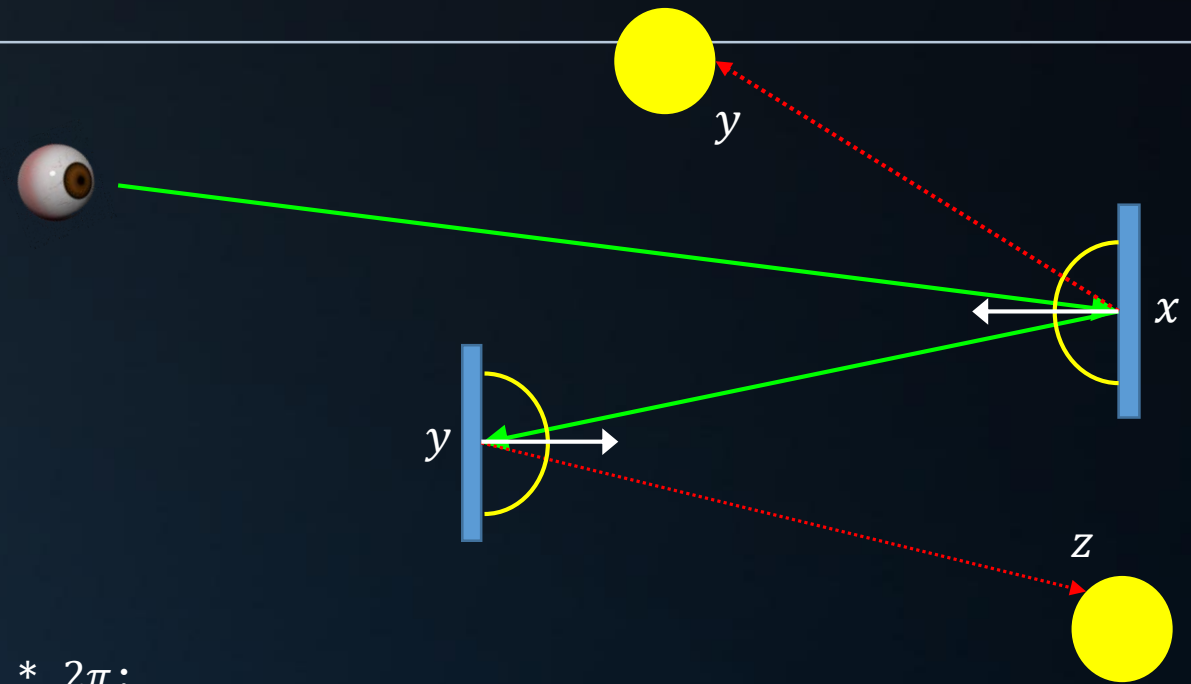
Indirect Light

One particle finding the light via a surface:

```
I, N = Trace( ray );
R = DiffuseReflection( N );
lightColor = Trace( new Ray( I, R ) );
return dot( R, N ) *  $\frac{\text{albedo}}{\pi}$  * lightColor *  $2\pi$ ;
```

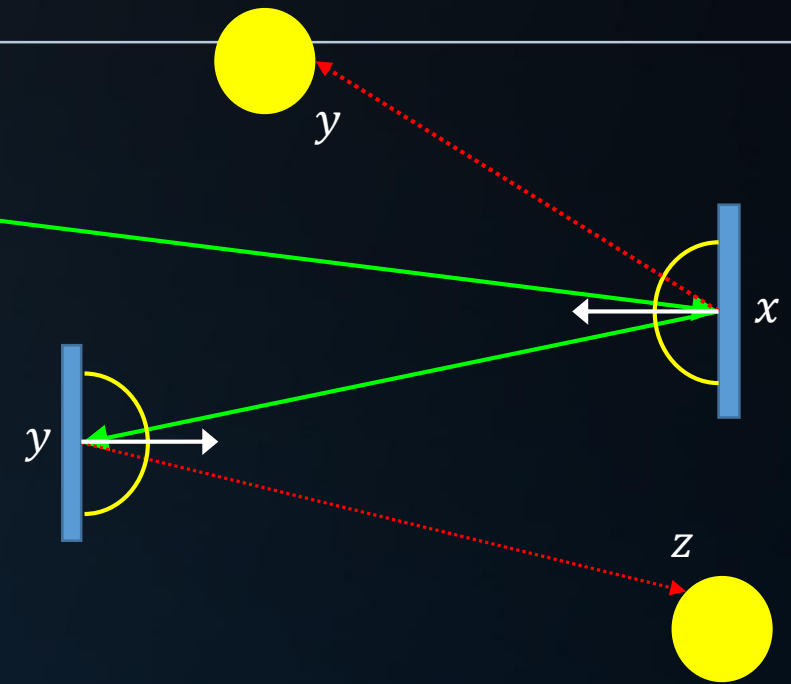
One particle finding the light via two surfaces:

```
I1, N1 = Trace( ray );
R1 = DiffuseReflection( N1 );
I2, N2 = Trace( new Ray( I1, R1 ) );
R2 = DiffuseReflection( N2 );
lightColor = Trace( new Ray( I2, R2 ) );
return dot( R1, N1 ) *  $\frac{\text{albedo}}{\pi}$  *  $2\pi$  * dot( R2, N2 ) *  $\frac{\text{albedo}}{\pi}$  *  $2\pi$  * lightColor;
```

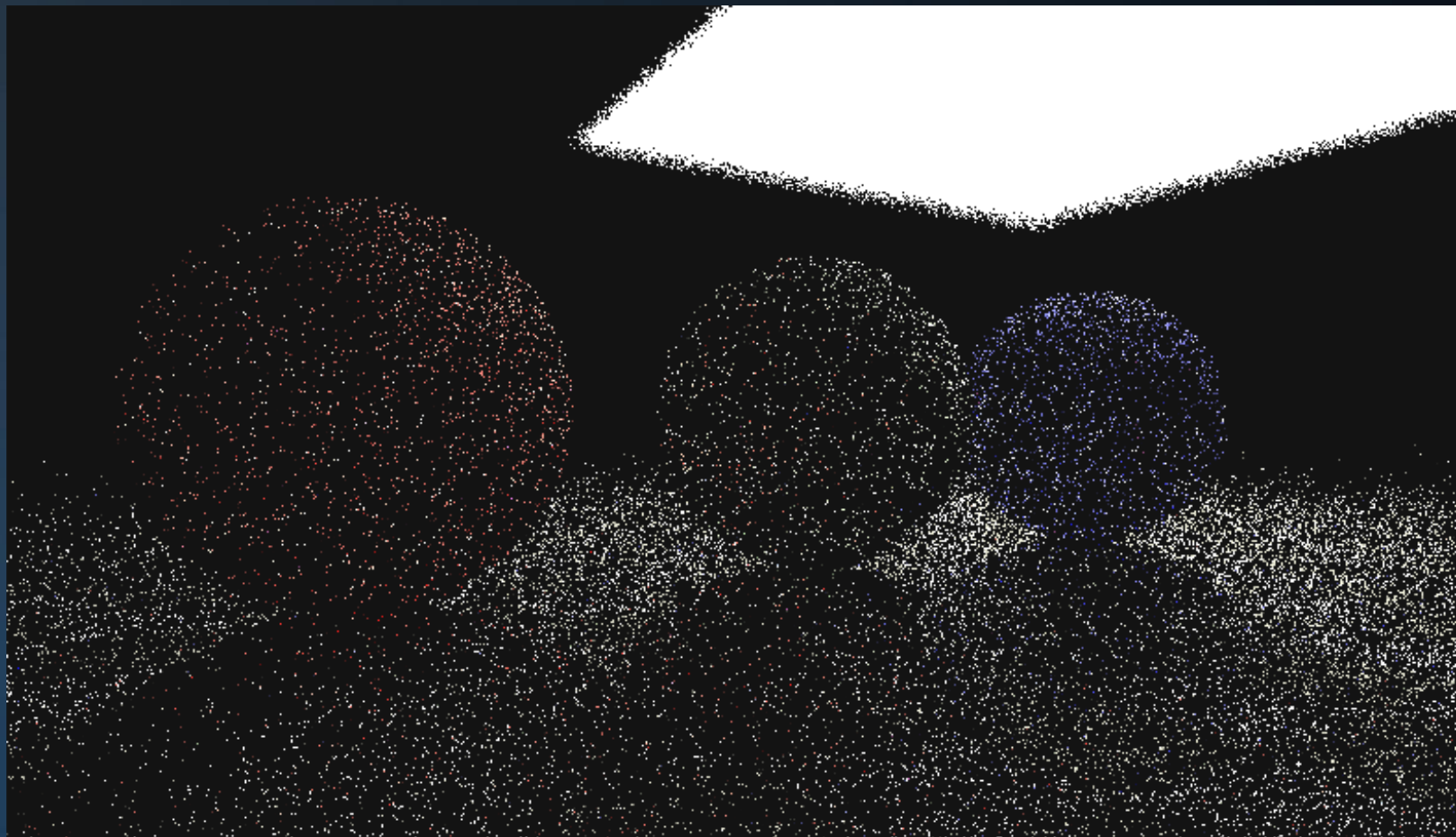


Path Tracing Algorithm

```
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;
}
```

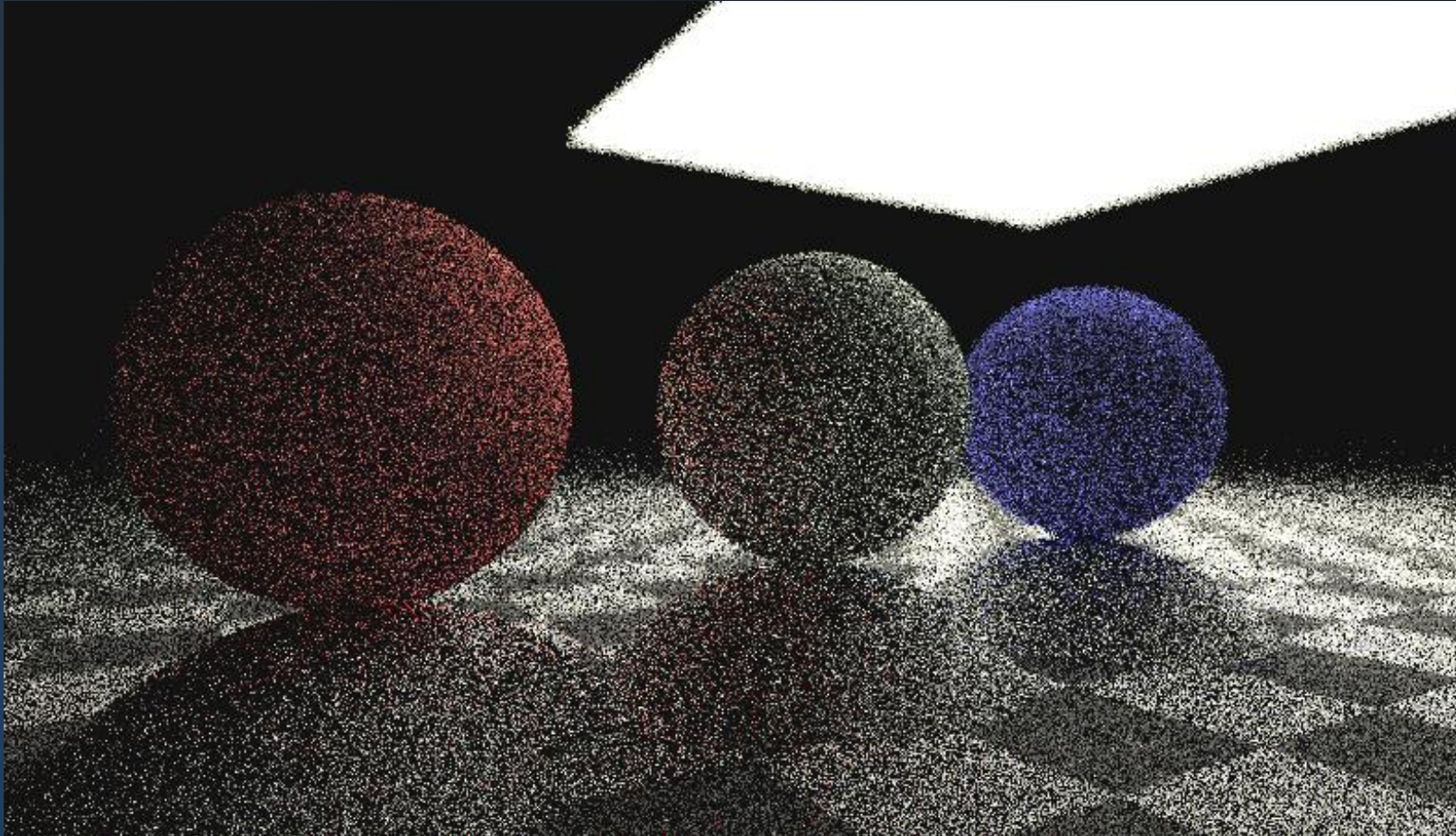


Ray Tracing for Games



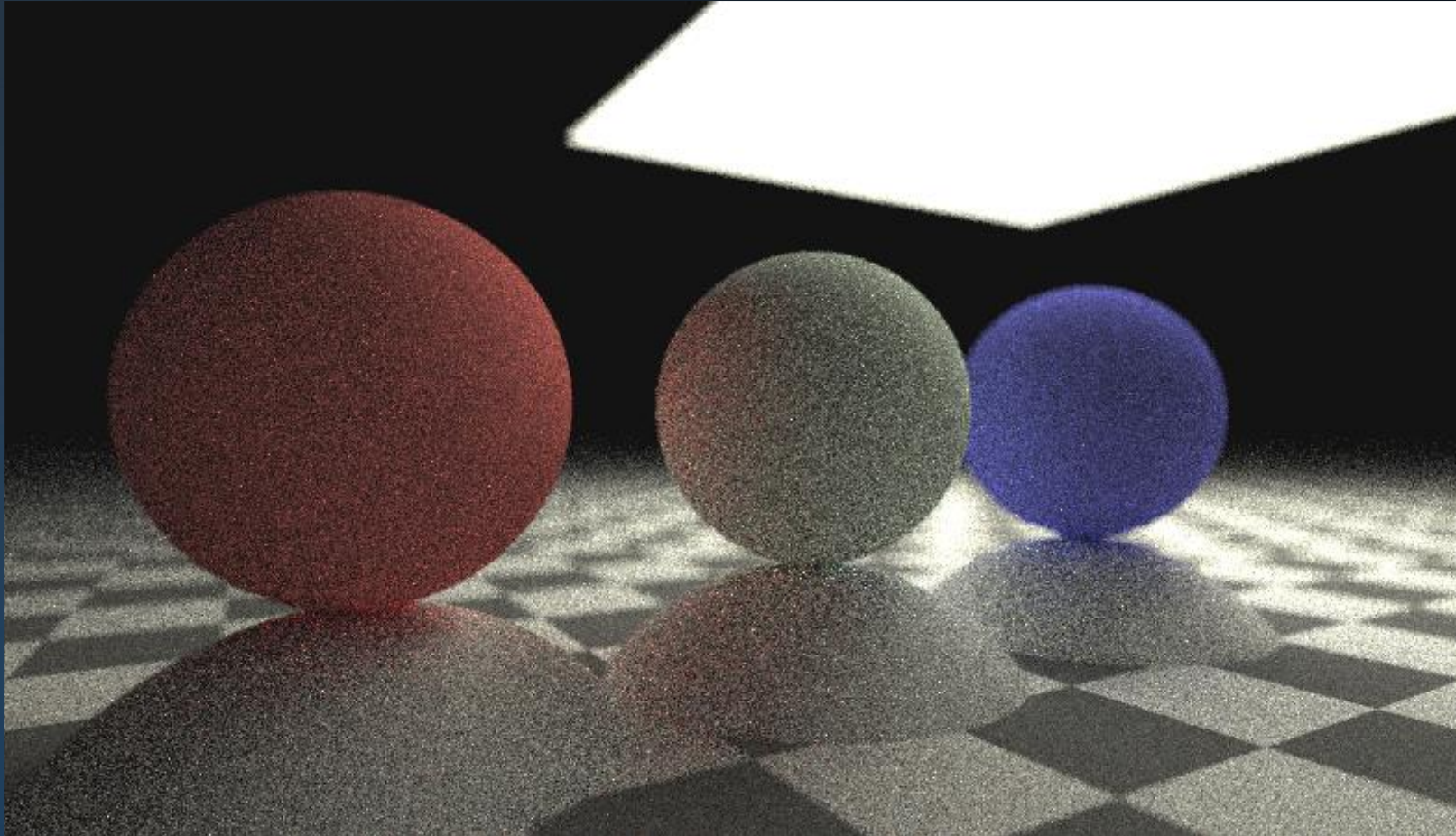
Ray Tracing for Games

```
ics
& (depth < MAXDEPTH)
{
    if (nt < 0)
        nt = inside ? 1.0f : 0.0f;
    nt = nt / nc; ddn = ddn * nt;
    float r0 = 1.0f - nnt * nnt;
    float r1 = 0, r2 = 0;
    do
    {
        r1 = D, N );
        r2 = 0;
    }
    while (r1 < 0);
    float a = nt - nc, b = nt + nc;
    float Tr = 1 - (R0 + (1 - R0) * r1);
    float R = (D * nnt - N * (ddn * r1 + refr)) && (depth < MAXDEPTH)
    {
        D, N );
        refr * E * diffuse;
        = true;
    }
    else
    {
        refr * E * diffuse;
        = true;
    }
    MAXDEPTH)
    survive = SurvivalProbability( diffuse, L, N );
    estimation - doing it properly, closely following
    df;
    radiance = SampleLight( &rand, I, &L, &light, &N );
    e.x + radiance.y + radiance.z > 0) && (depth < MAXDEPTH)
    {
        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.x + radiance.y + radiance.z > 0) && (depth < MAXDEPTH)
    }
    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);
        ion = true;
    }
}
```



Ray Tracing for Games

```
ics
& (depth < MAXDEPTH)
{
    if (nt < 0)
        nt = inside ? 1 : -1;
    nc = nt / nc; ddn = dot(N, D);
    nnt = nnt * nnt;
    cos2t = 1.0f - nnt * nnt;
    D, N );
    if (cos2t < 0)
        cos2t = 0;
    cos t = sqrt(cos2t);
    at a = nt - nc, b = nt + nc;
    at Tr = 1 - (R0 + (1 - R0) * cos t);
    (Tr) R = (D * nnt - N * (ddn * cos t));
    E * diffuse;
    = true;
    refl + refr)) && (depth < MAXDEPTH)
    D, N );
    refl * E * diffuse;
    = true;
    MAXDEPTH)
    survive = SurvivalProbability( diffuse );
    estimation - doing it properly, closely following
    df;
    radiance = SampleLight( &rand, I, &L, &lightPos );
    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;
```



Ray Tracing for Games

```
ics
& (depth < MAXDEPTH)
{
    if (nt < 0)
        nt = inside ? 1.0f : 0.0f;
    nt = nt / nc, ddn = ddn * nt;
    float r2 = 1.0f - ddn;
    float r1 = 0, r2 = 1.0f - r2;
    float r = sqrt(r2);
    float D, N );
    return 0;
}

float a = nt - nc, b = nt + nc;
float Tr = 1 - (R0 + (1 - R0) * r);
float R = (D * nnt - N * (ddn < 0 ? 1 : -1));

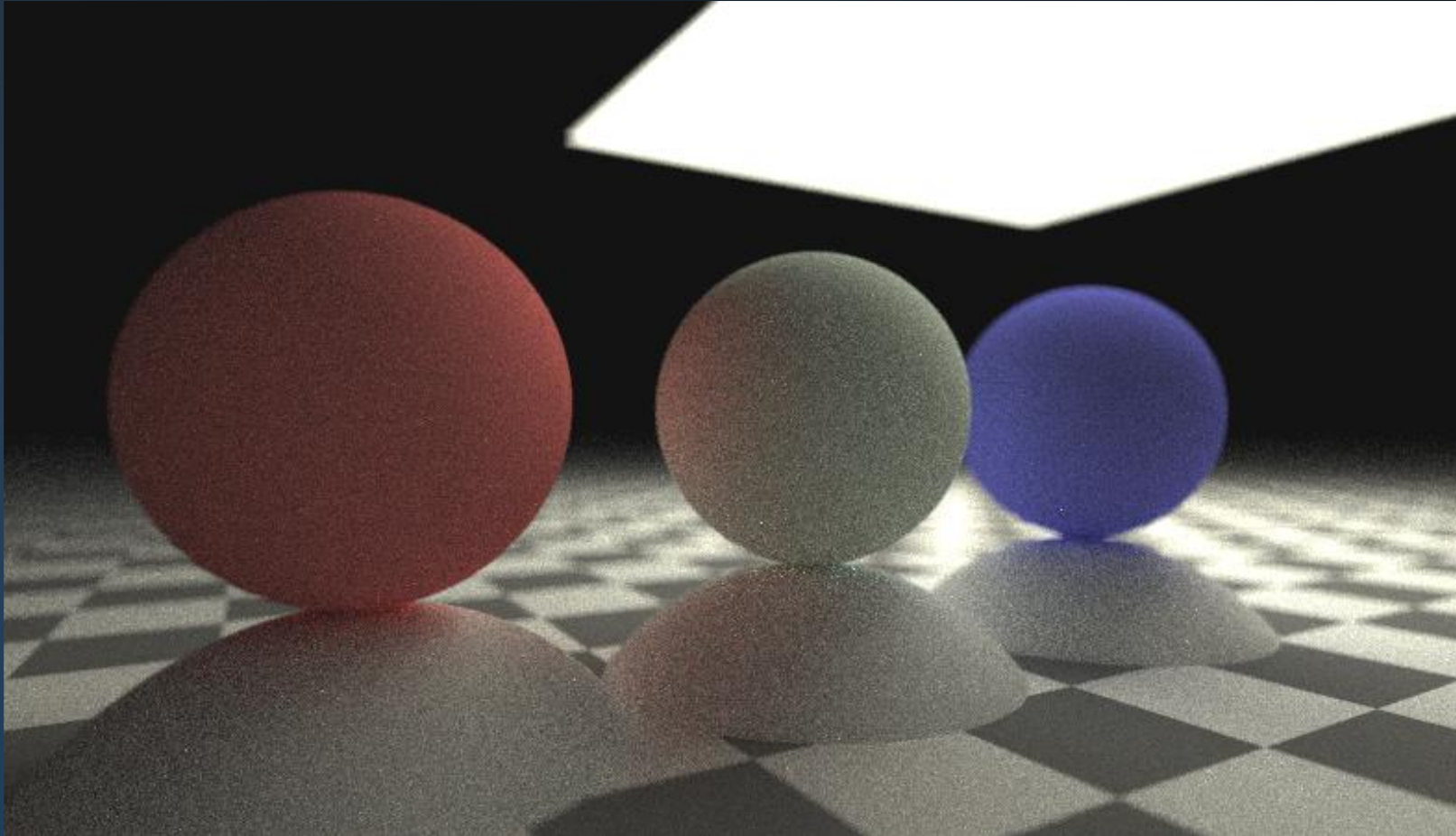
E * diffuse;
= true;

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

MAXDEPTH)

survive = SurvivalProbability( diffuse );
estimation - doing it properly, closely following
df;
radiance = SampleLight( &rand, I, &L, &lightPos );
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
ive)

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



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.

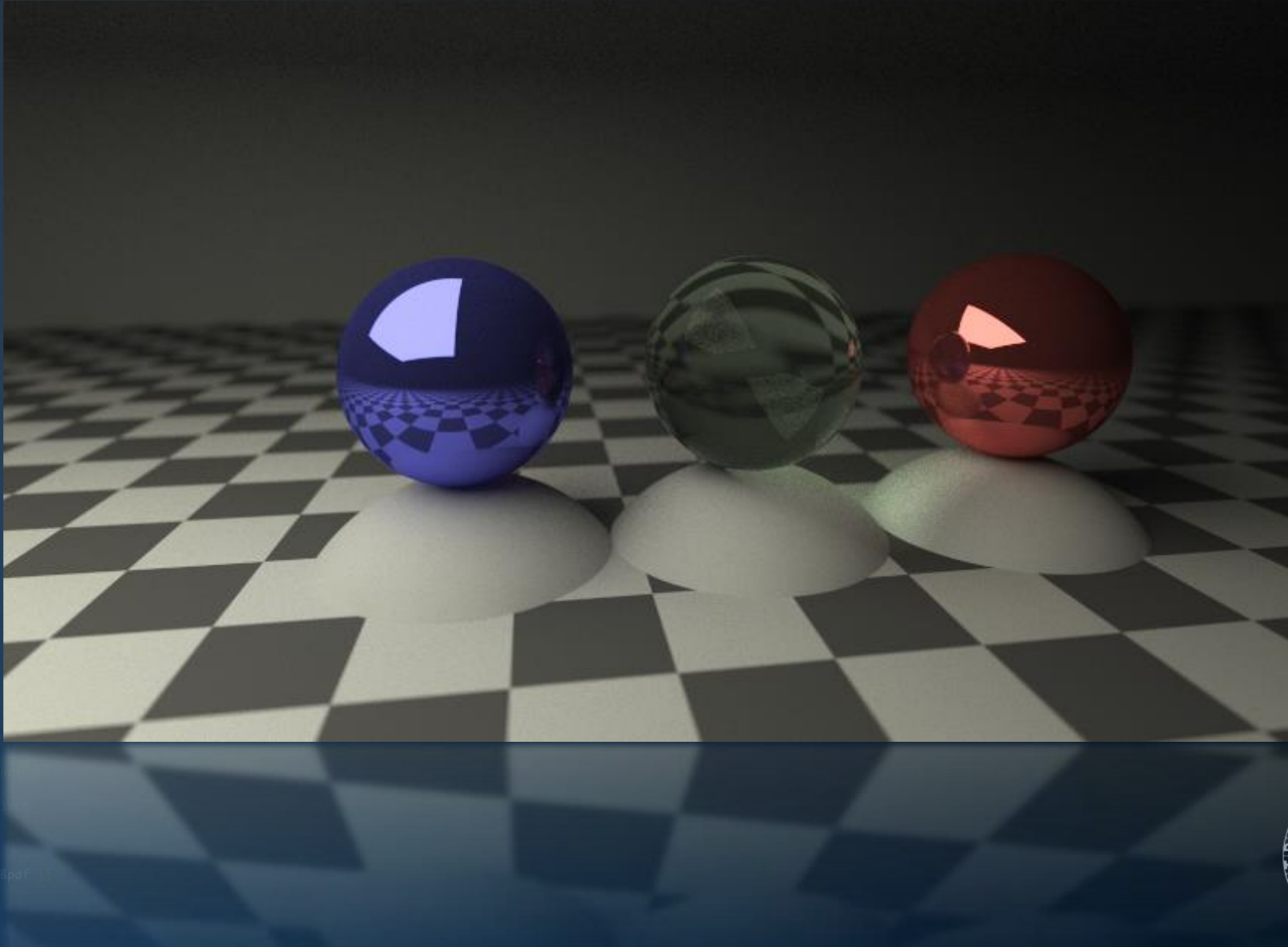
The probability of each choice is calculated using the Fresnel equations.

```
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;
}
```

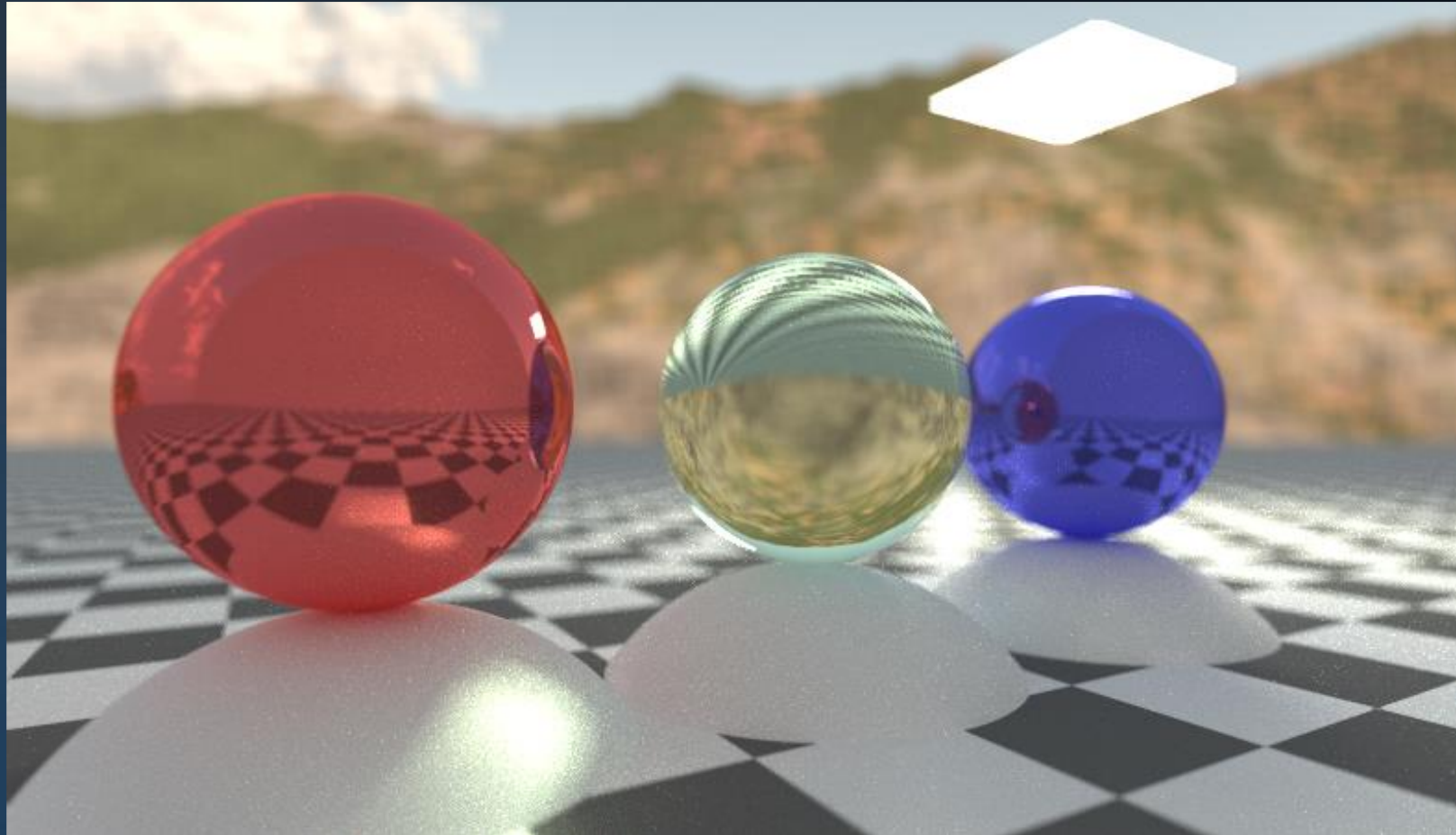


Ray Tracing for Games

```
ics
& (depth < MAXDEPTH)
{
    nt = inside ? 1.0f : 0.0f;
    nt = nt / nc, ddn = dot(N, D);
    cos2t = 1.0f - nnt * ddn;
    D, N );
    )
    at a = nt - nc, b = nt + nc;
    at Tr = 1 - (R0 + (1 - R0) * sqrt(cos2t));
    Tr) R = (D * nnt - N * (ddn < 0 ? 1 : -1));
    E * diffuse;
    = true;
    -
    efl + refr)) && (depth < MAXDEPTH))
    D, N );
    refl * E * diffuse;
    = true;
    MAXDEPTH)
    survive = SurvivalProbability( diffuse );
    estimation - doing it properly, closely following
    df;
    radiance = SampleLight( &rand, I, &L, &lightPos,
    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
    ve)
    ;
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    sion = true;
}
```



Ray Tracing for Games



End of PART 8.

```
ics
& (depth < MAXDEPTH)
{
    if (inside ? 1 : 1.25 * nnt)
    {
        nt = nt / nc; ddn = ddn * nc;
        cos2t = 1.0f - nnt * nnt;
        D, N );
    }
    {
        at a = nt - nc, b = nt + nc;
        at Tr = 1 - (R0 + (1 - R0) * ddn);
        Tr) R = (D * nnt - N * (ddn > 0 ? 1 : -1));
    }
    E * diffuse;
    = true;
    -
    refl + refr)) && (depth < MAXDEPTH)
    {
        D, N );
        refl * E * diffuse;
        = true;
    }
    MAXDEPTH)
    {
        survive = SurvivalProbability( diffuse );
        estimation - doing it properly, closely following Small's
        if;
        radiance = SampleLight( &rand, I, &L, &lightDir );
        e.x + radiance.y + radiance.z) > 0) && (depth < MAXDEPTH)
        {
            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 Small's
        (survive)
    }
    {
        at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
        survive;
        pdf;
        n = E * brdf * (dot( N, R ) / pdf);
        sion = true;
    }
}
```

