

```
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;
    }
}
```

# Ray Tracing for Games

Dr. Jacco Bikker - IGAD/BUAS, Breda, February 6

# Welcome!

# 9



```

ics
& (depth < MAXDEPTH))
{
    if (inside) {
        nt = nt / nc, ddn = ddn * nc;
        cos2t = 1.0f - nnt * ddn;
        D, N );
    }
    else {
        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
        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
        vive)
        ;
        at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
        survive;
        pdf;
        n = E * brdf * (dot( N, R ) / pdf);
        sion = true;

```

# Agenda:

- Introduction
- Stratification
- Blue Noise
- Next Event Estimation



# Ray Tracing for Games



Thursday  
09:00 – 14:00

advanced Whitted  
audio, AI & physics  
faster Whitted  
Heaven7

Friday  
09:00 – 17:00

optimization  
profiling, rules of  
engagement  
threading

Monday  
09:00 – 17:00

acceleration  
grid, BVH, kD-tree  
SAH  
binning

Tuesday  
09:00 – 17:00

Monte-Carlo  
Cook-style  
glossy, AA  
area lights, DOF

Thursday  
09:00 – 17:00

random numbers  
stratification  
blue noise

Friday  
09:00 – 17:00

future work

Wednesday  
13:00 – 17:00

course intro  
LH2  
template  
Whitted  
refactoring  
RT-centric games

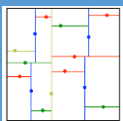
LAB 2



LAB 3



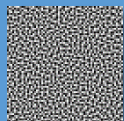
LAB 5



LAB 7



LAB 9



LAB 11



LAB 1

work @ home

End result day 2:

A solid Whitted-style  
ray tracer, as a basis  
for subsequent work.

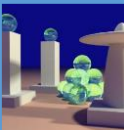
SIMD  
applied SIMD  
SIMD triangle  
SIMD AABB

LAB 4

refitting  
top-level BVH  
threaded building

LAB 6

path tracing



LAB 8

importance  
sampling  
next event  
estimation

LAB 10

path guiding



LAB 10



GLOBAL GAME JAM

End result day 3:

A 5x faster tracer.

End result day 4:

A real-time tracer.

End result day 5:

Cook or Kajiya.

End result day 6:

Efficiency.

End result day 6:

Great product.

## Path Tracing

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

The rendering equation allows us to accurately simulate light transport:

- Given a sensible scene description
- Given a sensible camera model
- Ignoring some complex effects.

We evaluate it using Monte Carlo integration:

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

➔ The result is always an estimate, with variance.



[www.mitsuba-renderer.org](http://www.mitsuba-renderer.org)





## Noise in the Real World

Noise in a digital photo can be the result of short exposure.



[www.photoreview.com.au/tips/shooting/how-to-control-image-noise](http://www.photoreview.com.au/tips/shooting/how-to-control-image-noise)



## Noise in the Real World

Noise in a digital photo can be the result of short exposure.

We solve this by lowering our ISO settings, so we gather more photons per pixel.

*(that obviously has consequences)*



[www.photoreview.com.au/tips/shooting/how-to-control-image-noise](http://www.photoreview.com.au/tips/shooting/how-to-control-image-noise)





Today:

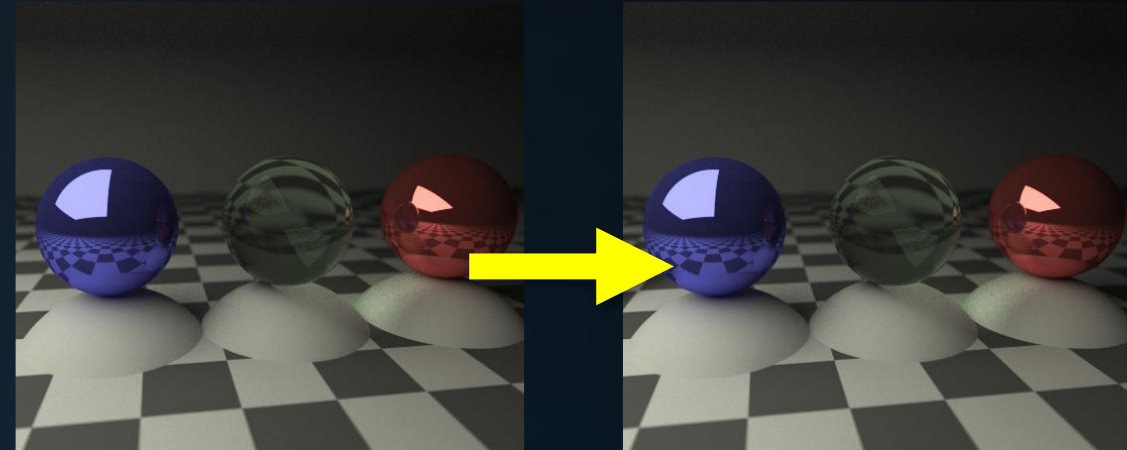
- Stratification
- Blue noise
- Next Event Estimation
- Importance Sampling

Aim:

- to get a better image with the same number of samples
- to increase the efficiency of a path tracer
- to reduce variance in the estimate

Requirement:

- *produce the correct image*



```

ics
& (depth < MAXDEPTH))
{
    if (inside) {
        nt = nt / nc, ddn = ddn * nc;
        cos2t = 1.0f - nnt * ddn;
        D, N );
    }
    else {
        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
        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 Smallwood
        vive)
        ;
        at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
        survive;
        pdf;
        n = E * brdf * (dot( N, R ) / pdf);
        sion = true;

```

# Agenda:

- Introduction
- Stratification
- Blue Noise
- Next Event Estimation



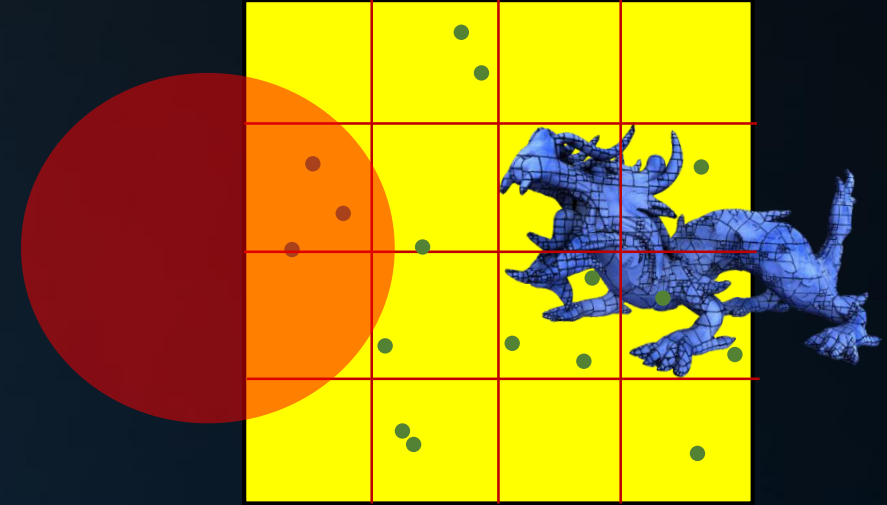


## Uniform Random Sampling

To sample a light source, we draw two random values in the range 0..1.

The resulting 2D positions are not uniformly distributed over the area.

We can improve uniformity using *stratification*: one sample is placed in each stratum.



```
ics
& (depth < MAXDEPTH)
{
    if (inside) {
        nt = nt / nc; ddn = ddn * ddn;
        cos2t = 1.0f - nnt * ddn;
        D, N );
    }
    at a = nt - nc, b = nt + nc;
    at Tr = 1 - (R0 + (1 - R0) * R);
    Tr) R = (D * nnt - N * (ddn *
    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, &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;
```



## Uniform Random Sampling

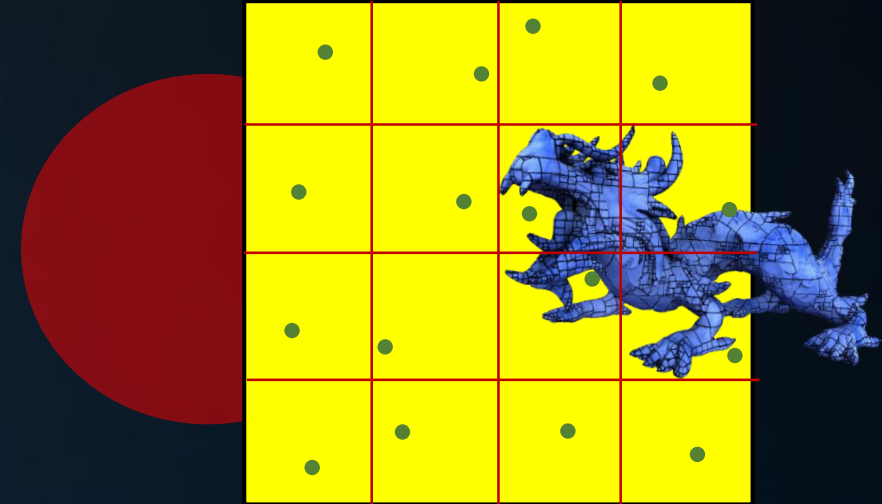
To sample a light source, we draw two random values in the range 0..1.

The resulting 2D positions are not uniformly distributed over the area.

We can improve uniformity using *stratification*: one sample is placed in each stratum.

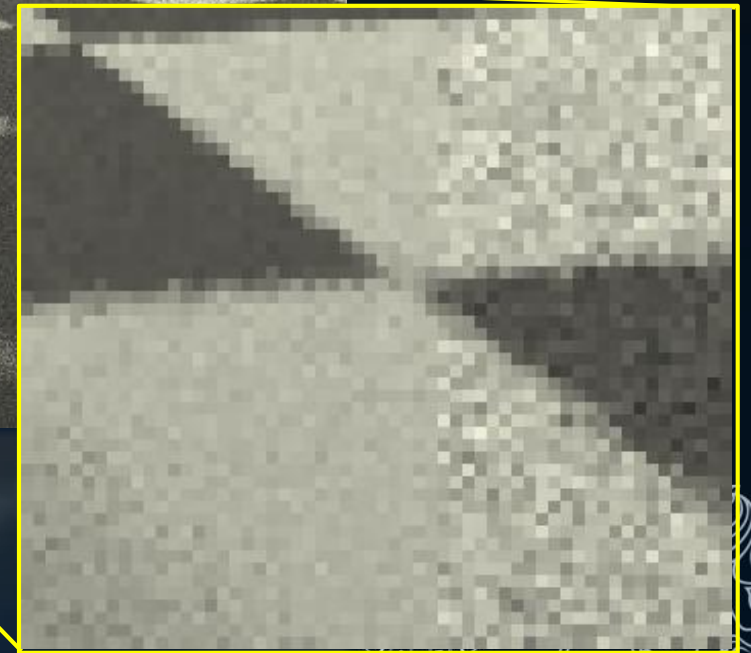
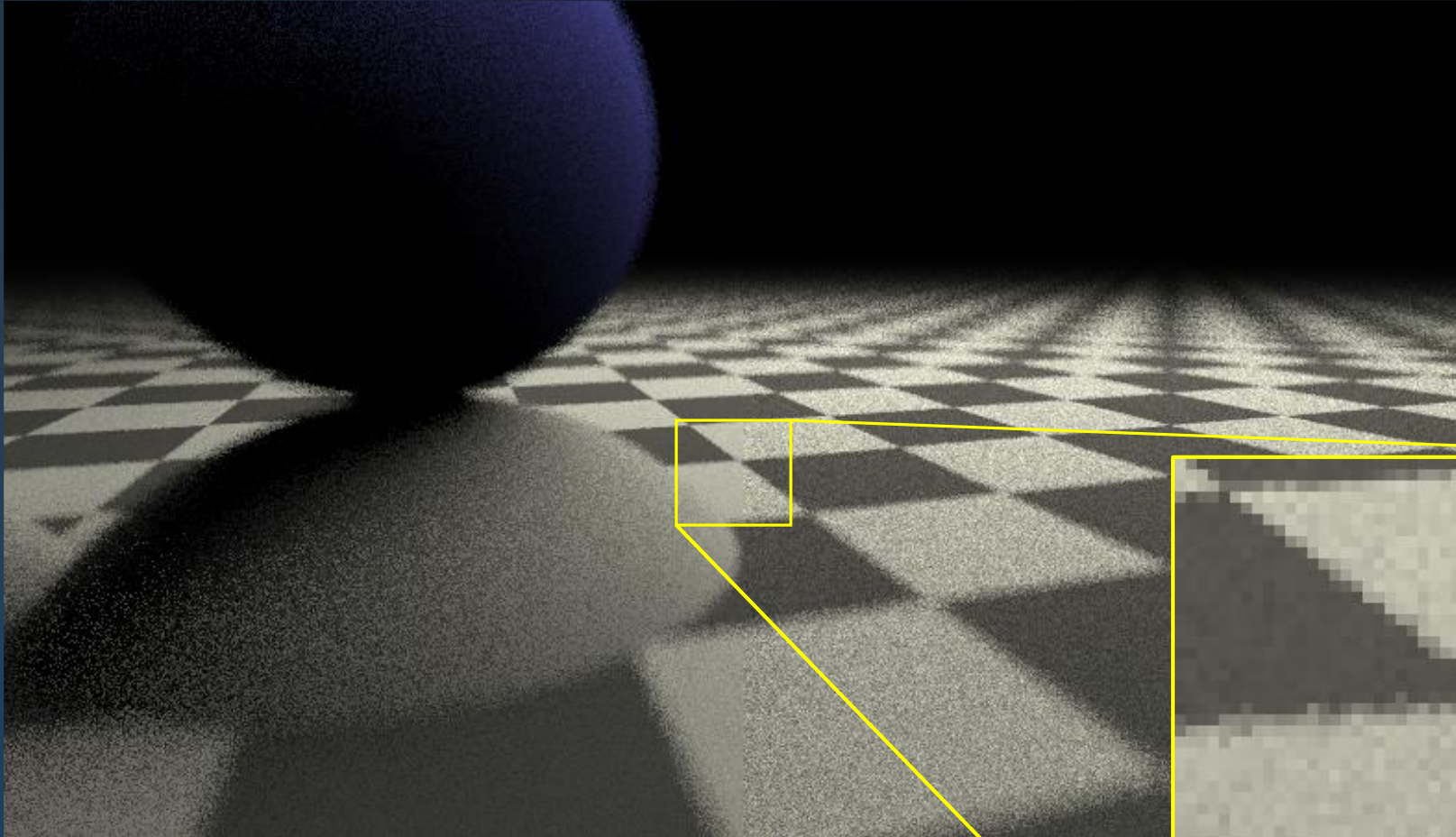
For 4x4 strata:

```
stratum_x = (idx % 4) * 0.25 // idx = 0..15
stratum_y = (idx / 4) * 0.25
r0 = Rand() * 0.25
r1 = Rand() * 0.25
P = vec2( stratum_x + r0, stratum_y + r1 )
```



# Ray Tracing for Games

```
ics
& (depth < MAXDEPTH)
{
    if (nt < nc)
    {
        nt = inside ? 1.0f - nc : nc;
        nnt = nt / nc; ddn = dot(N, D);
        cos2t = 1.0f - nnt * nnt;
        D = D * cos2t + N * sin2t;
        D = D * D;
        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
    df;
    radiance = SampleLight( &rand, I, &L, &align, &pdf );
    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 Smith's
    (survive)
    {
        at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
        survive;
        pdf;
        n = E * brdf * (dot( N, R ) / pdf);
        sion = true;
    }
}
```





## Troubleshooting Path Tracing Experiments

When experimenting with stratification and other variance reduction methods you will frequently produce incorrect images.

Tip:

Keep a simple reference path tracer without any tricks.  
Compare your output to this reference solution frequently.



```

ics
& (depth < MAXDEPTH))
{
    if (nt < 1)
        nt = inside ? 1 : 1.25;
    nt = nt / nc; ddn = ddn * nt;
    r2s2t = 1.0f - nnt * nnt;
    D, N );
    )
    {
        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
        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
        vive)
        ;
        at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
        survive;
        pdf;
        n = E * brdf * (dot( N, R ) / pdf);
        sion = true;

```

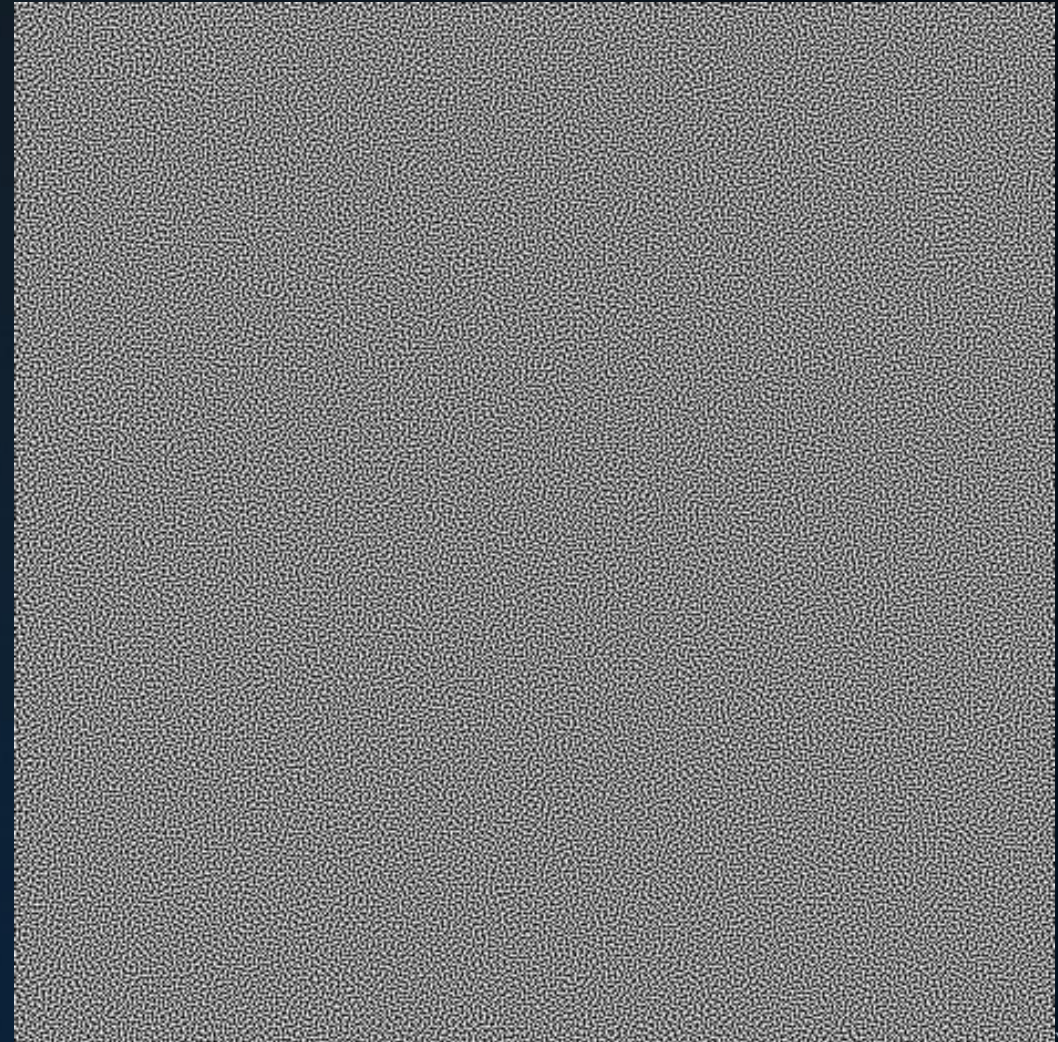
# Agenda:

- Introduction
- Stratification
- Blue Noise
- Next Event Estimation



## Not All Noise is Equal

Our eyes expect noise. ‘Good noise’ is uniform, high-frequent, and... blue\*.

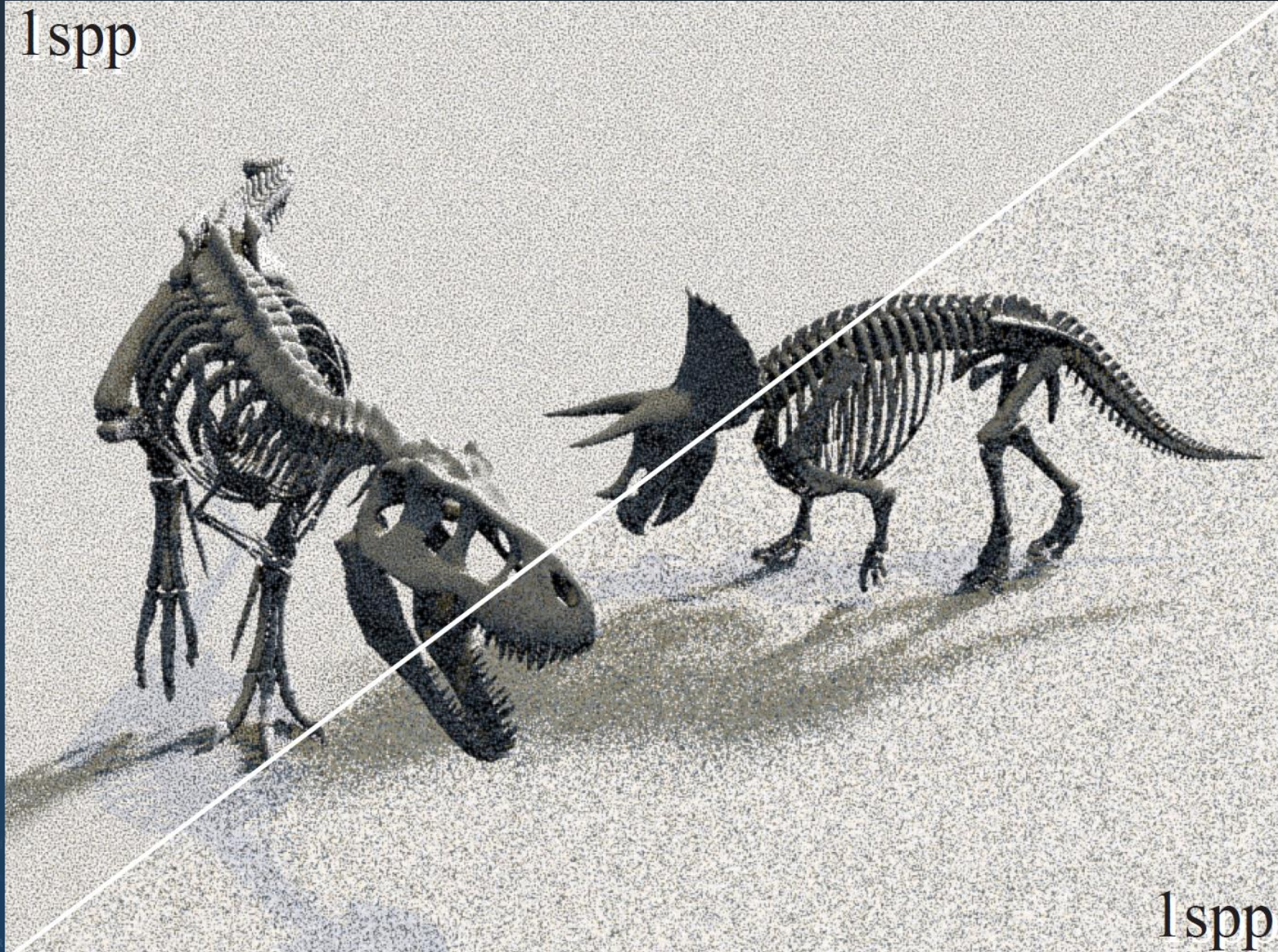


```
ics
& (depth < MAXDEPTH)
{
    if (inside & !isRefr)
    {
        nt = nt / nc; ddn = ddn * ddn;
        cos2t = 1.0f - nnt * ddn;
        D, N );
    }
    else
    {
        at a = nt - nc, b = nt + nc;
        at Tr = 1 - (R0 + (1 - R0) * rand());
        Tr) R = (D * nnt - N * (ddn * cos2t));
    }
    E * diffuse;
    = true;
}
-
refl + refr)) && (depth < MAXDEPTH)
{
    D, N );
    refl * E * diffuse;
    = true;
}
MAXDEPTH)
survive = SurvivalProbability( diffuse, I );
estimation - doing it properly, closely following
if;
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
    random walk - done properly, closely following
    survive)
};
at3 brdf = SampleDiffuse( diffuse, N, r1, r2 );
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;
```





1 spp



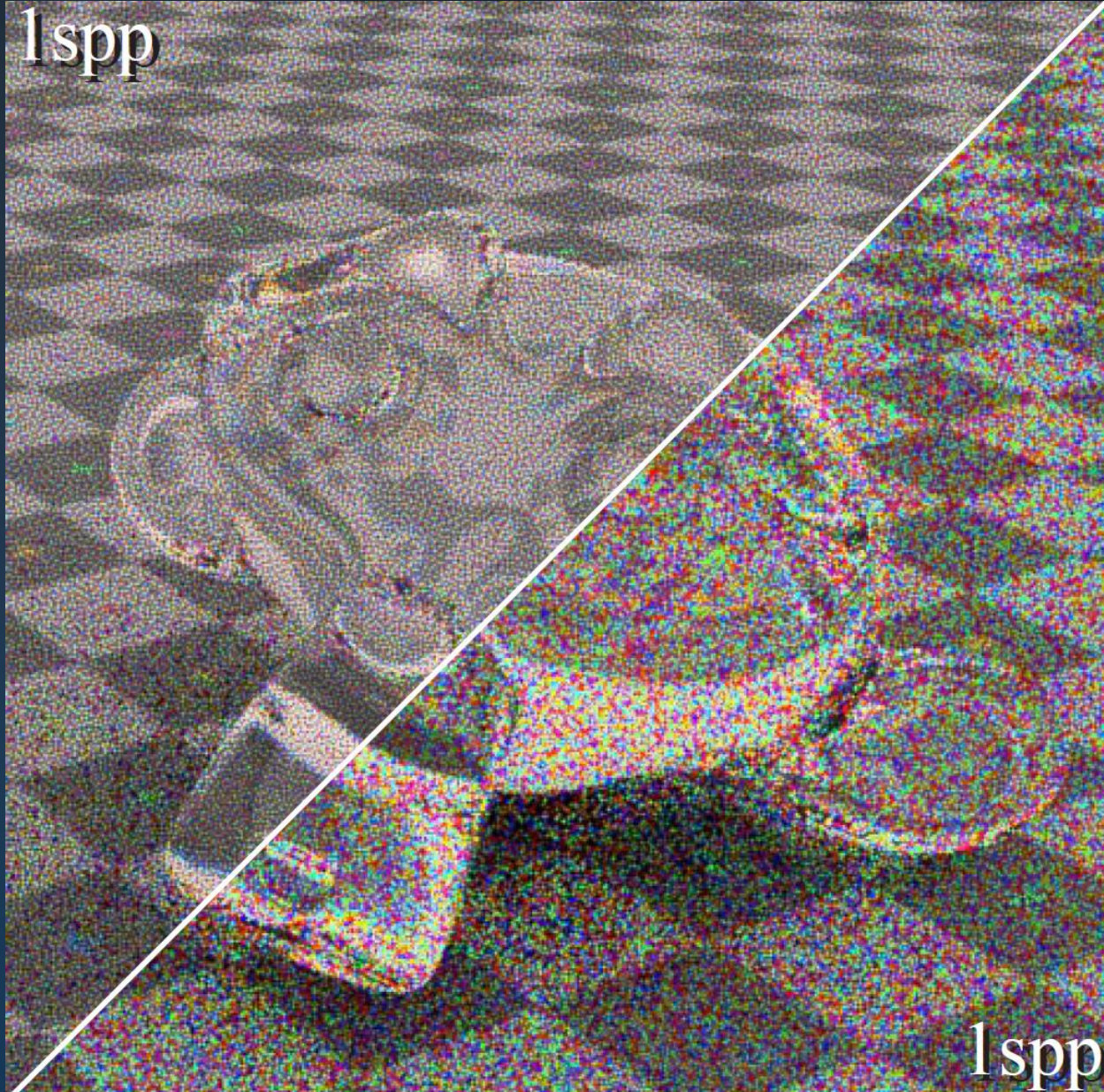
1 spp





# Ray Tracing for Games

```
ics
& (depth < MAXDEPTH)
{
    if (nt < 0)
        nt = inside ? 1.0f - nt : nt;
    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 R0 = 1 - (R0 + (1 - R0) * r1);
    float R = (D * nnt - N * (ddn < 0 ? 1 : -1));
    if (R < 0)
    {
        R = 0;
        E * diffuse;
        = true;
    }
    else
    {
        refl + refr)) && (depth < MAXDEPTH)
    {
        D, N );
        refl * E * diffuse;
        = true;
    }
    if (depth < MAXDEPTH)
    {
        survive = SurvivalProbability( diffuse, L, N );
        estimation - doing it properly, closely following Small's
        if;
        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
        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;
}
```













```

ics
& (depth < MAXDEPTH)) {
    if (inside) {
        nt = nt / nc, ddn = ddn * nc;
        cos2t = 1.0f - nnt * ddn;
        D, N );
    }
    if (a = nt - nc, b = nt + nc, c = 0) {
        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
    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 Smallwood
    vive)
    ;
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    sion = true;

```

# Agenda:

- Introduction
- Stratification
- Blue Noise
- Next Event Estimation



# Ray Tracing for Games

## Incoming direct light

$$= \int_{\Omega} L_d(x, \omega_i) \cos \theta_i d\omega_i$$

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

$$-\frac{1}{2}\pi$$
 $+1/2\pi$  $x$ 



Incoming direct light

$-\frac{1}{2}\pi$

A

B

C

D

$+\frac{1}{2}\pi$

$$\begin{aligned} &= \int_{\Omega} L_d(x, \omega_i) \cos \theta_i d\omega_i \\ &\approx \frac{2\pi}{N} \sum_{i=1}^N L_d(p, \Omega_i) \cos \theta_i \\ &= \int_{A..B} L_d(x, \omega_i) \cos \theta_i d\omega_i + \int_{C..D} L_d(x, \omega_i) \cos \theta_i d\omega_i \end{aligned}$$



$\chi$



# Ray Tracing for Games

Incoming direct + indirect light

$-\frac{1}{2}\pi$

$+\frac{1}{2}\pi$

*A*

*B*

*C*

*D*

$\chi$

```
ics
& (depth < MAXDEPTH)
{
    if (inside) {
        nt = nt / nc; ddn = ddn * ddn;
        cos2t = 1.0f - nnt * ddn;
        t = sqrt(cos2t);
        D, N );
    }
    at a = nt - nc, b = nt + nc;
    at Tr = 1 - (R0 + (1 - R0) * t);
    Tr) R = (D * nnt - N * (ddn * 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, &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) * (radiant
    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);
    ion = true;
```



Incoming direct + indirect light

$-\frac{1}{2}\pi$

$+\frac{1}{2}\pi$

$-\frac{1}{2}\pi$

$+\frac{1}{2}\pi$

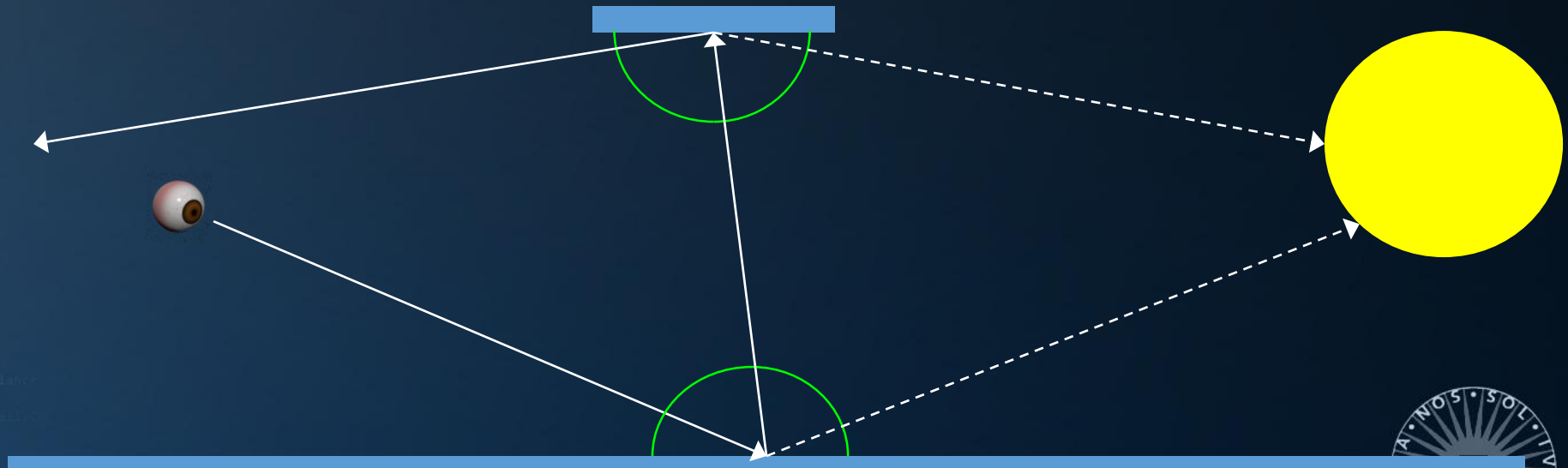




## Next Event Estimation

Observation: light travelling via any vertex on the path consists of indirect light and direct light *for that vertex*.

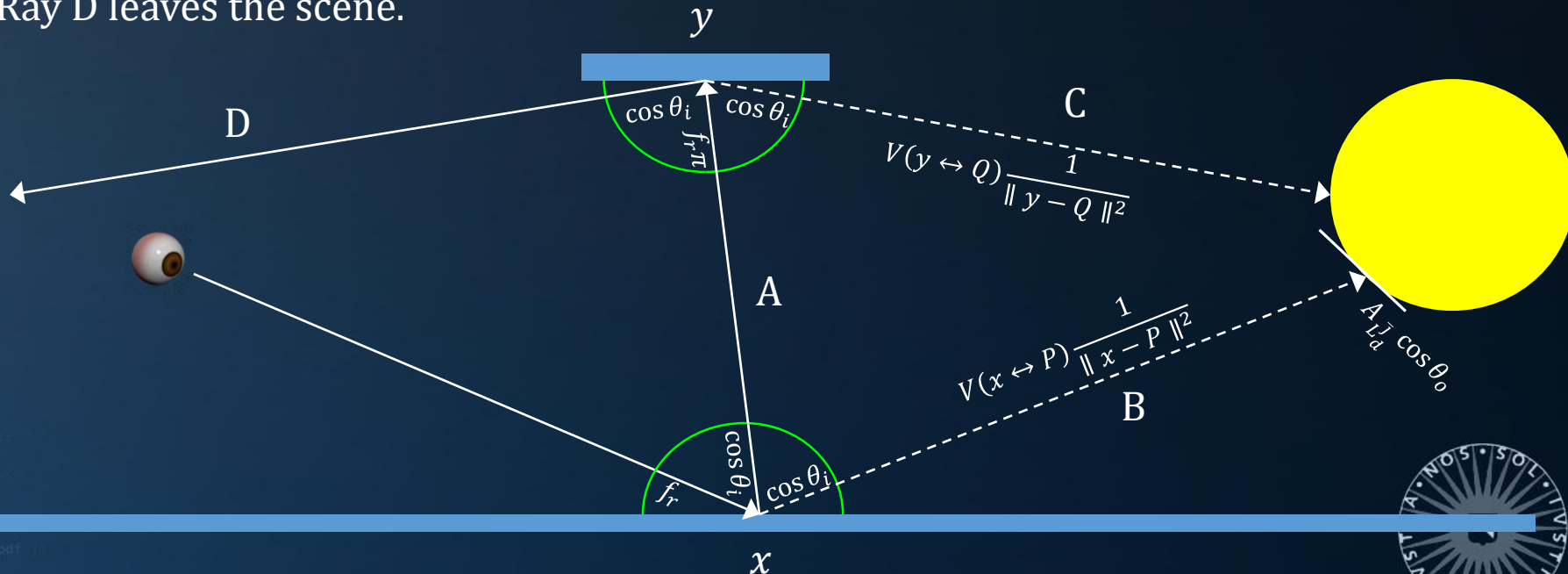
*Next Event Estimation*: sampling direct and indirect separately.



## Next Event Estimation

Per surface interaction, we trace *two* random rays.

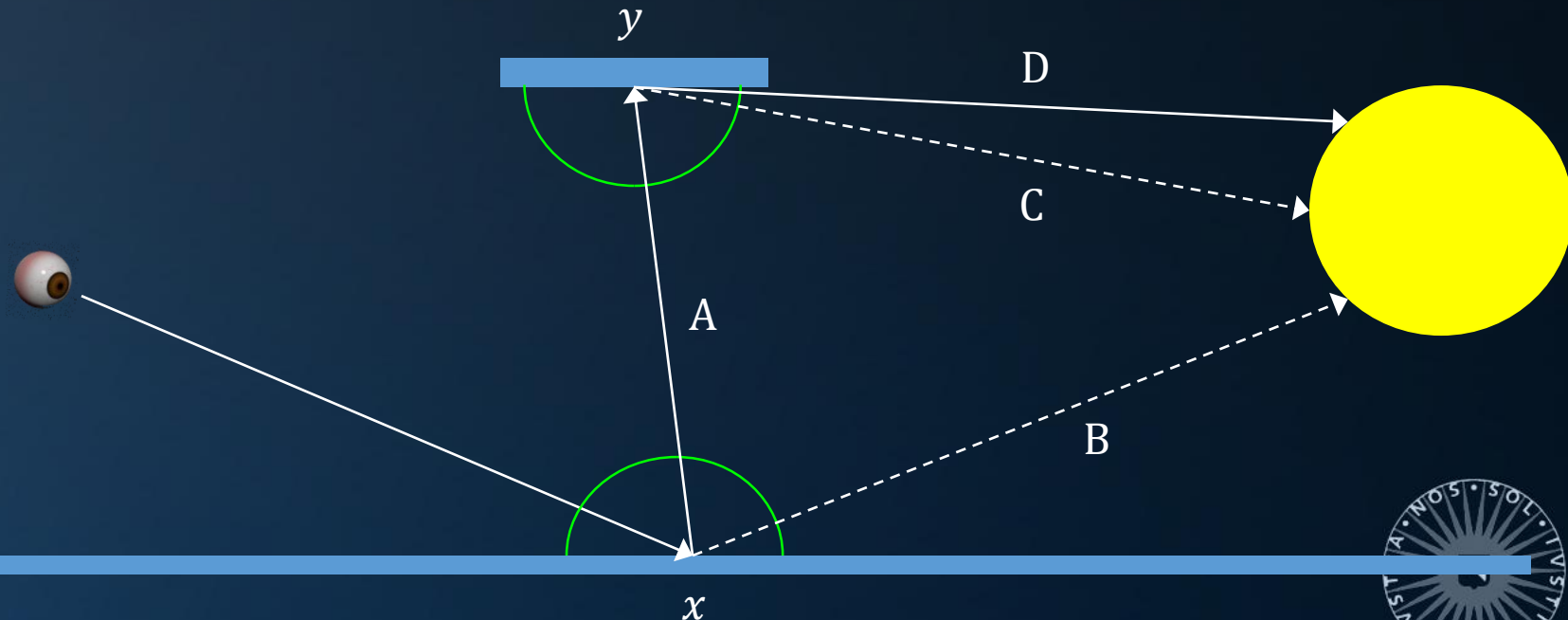
- Ray A returns (via point  $x$ ) the energy reflected by  $y$  (estimates indirect light for  $x$ ).
- Ray B returns the direct illumination on point  $x$  (estimates direct light on  $x$ ).
- Ray C returns the direct illumination on point  $y$ , which will reach the sensor via ray A.
- Ray D leaves the scene.



## Next Event Estimation

When a ray for indirect illumination stumbles upon a light, the path is terminated and no energy is transported via ray D:

This way, we prevent accounting for direct illumination on point  $y$  twice.





## Next Event Estimation

We thus split the hemisphere into two distinct areas:

1. The area that has the projection of the light source on it;
2. The area that is not covered by this projection.

We can now safely send a ray to each of these areas and sum whatever we find there.

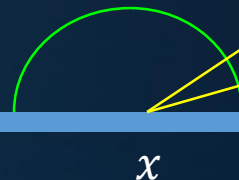
(or: we integrate over these non-overlapping areas and sum the energy we receive via both to determine the energy we receive over the entire hemisphere)

Area 1:

Send a ray directly to a random light source. Reject it if it hits anything else than the targeted light.

Area 2:

Send a ray in a random direction on the hemisphere. Reject it if it hits a light source.



# Ray Tracing for Games

```

    &(depth < MAXDEPTH))
{
    if (inside ? 1 : 0.25)
    {
        nt = nt / nc, ddn = ddn / nc;
        ps2t = 1.0f - nnt * nnt * nnt;
        D, N );
    }
    at a = nt - nc, b = nt - nc;
    at Tr = 1 - 1/2π;
    (Tr) R = (D * nnt - N *
    E * diffuse;
    = true;
    -
    refl + refr)) && (depth < MAXDEPTH))
{
    D, N );
    refl * E * diffuse;
    = true;
    MAXDEPTH)
}

```

```

survive = SurvivalProbability( diffuse, L, N );
estimation - doing it properly, closely following Survival
if;
radiance = SampleLight( &rand, I, &L, &light, &pdf );
r.x + radiance.y + radiance.z) > 0) && (data < 0.0001)
v = true;
at3 brdfPdf = EvaluateDiffuse( L, N ) * Psurvive;
at3 factor = diffuse * INVPI;
at weight = Min( directPdf, brdfPdf );
at cosThetaOut = sqrt( 1 - 2 * PI );
E * ((weight * cosThetaOut) / directPdf) * (radiance
random walk - done properly, closely following Survival
ive)
;
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;

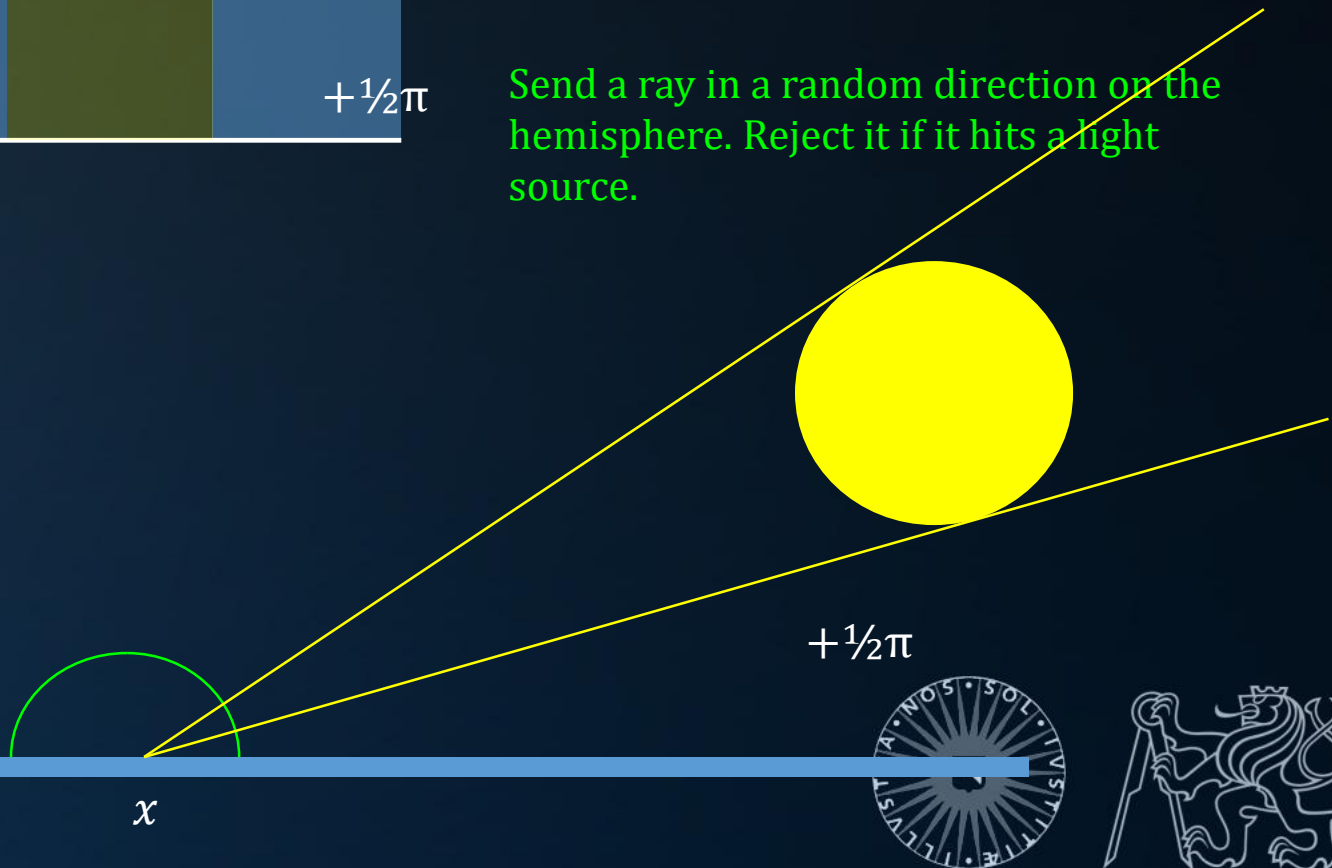
```

Area 1:

Send a ray directly to a random light source. Reject it if it hits anything else than the targeted light.

## Area 2:

Send a ray in a random direction on the hemisphere. Reject it if it hits a light source.





# Ray Tracing for Games

```
ics
& (depth < MAXDEPTH)
{
    if (inside ? 1 : 0)
    {
        nt = nt / nc; ddn = ddn * ddn;
        cos2t = 1.0f - nnt * ddn;
        D, N );
    }
    at a = nt - nc; b = nt + nc;
    at Tr = 1 - nnt * ddn;
    (Tr) R = (D * a + N * b);

    E * diffuse;
    = true;

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

    MAXDEPTH)
    {
        survive = SurvivalProbability( diffuse );
        estimation - doing it properly, closely following Smallwood
        if;
        radiance = SampleLight( &rand, I, &L, &lightPdf );
        e.x + radiance.y + radiance.z > 0) && (depth < MAXDEPTH)
        {
            w = true;
            at brdfPdf = EvaluateDiffuse( L, N ) * Survive;
            at3 factor = brdfPdf * InvPr;
            at weight = Mis2( directPdf, brdfPdf );
            at cosThetaOut = dot( N, L );
            E * ((weight * cosThetaOut) / directPdf) * (radiance
            random walk - done properly, closely following Smallwood
            (survive)
            ;
            at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
            survive;
            pdf;
            n = E * brdf * (dot( N, R ) / pdf);
            sion = true;
        }
    }
}
```

$-\frac{1}{2}\pi$

$+\frac{1}{2}\pi$

$-\frac{1}{2}\pi$

$+\frac{1}{2}\pi$

$+\frac{1}{2}\pi$

$\chi$

Area 1:

Send a ray directly to a random light source. Reject it if it hits anything else than the targeted light.

Area 2:

Send a ray in a random direction on the hemisphere. Reject it if it hits a light source.



## Next Event Estimation

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

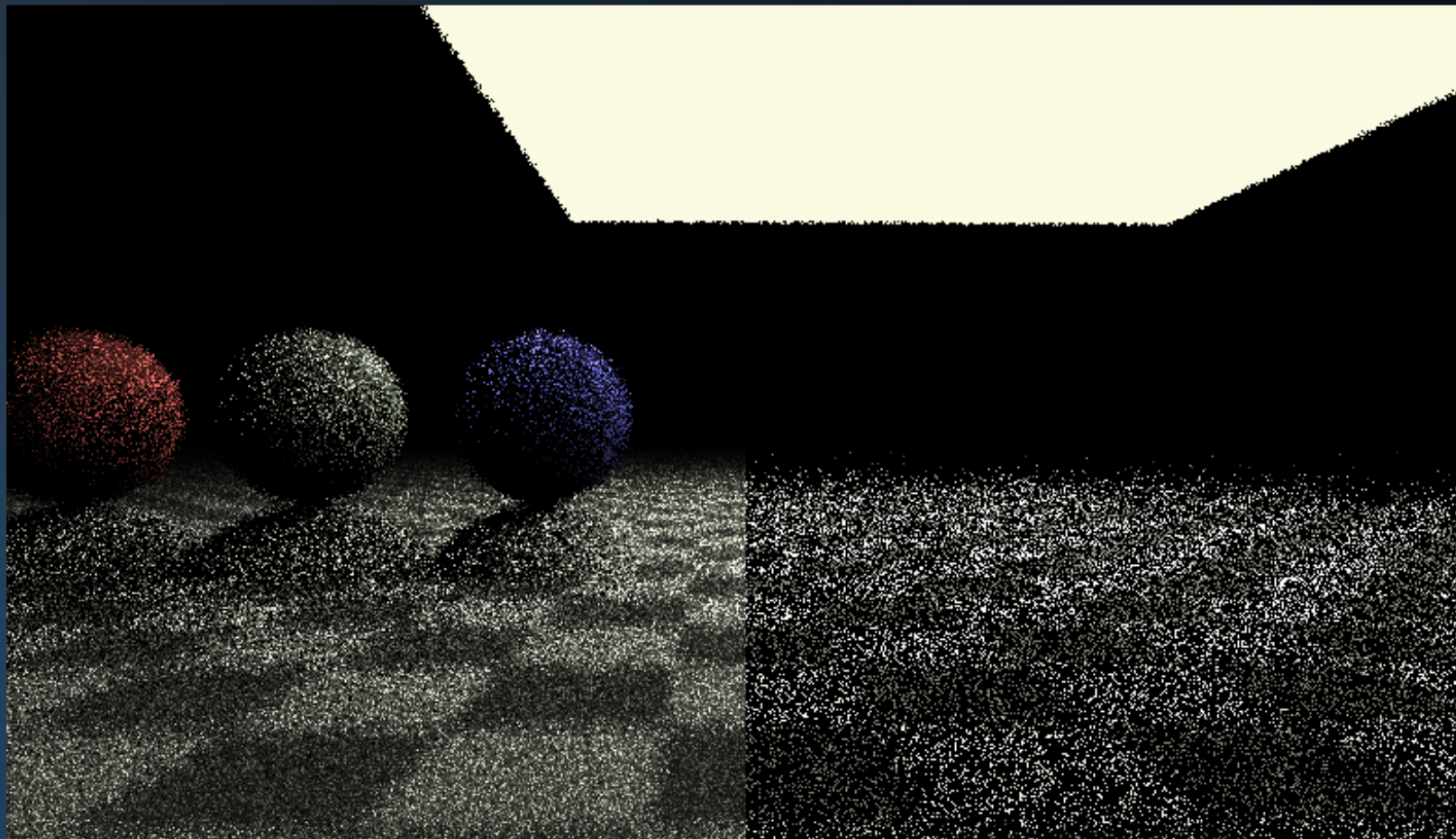
```
Color Sample( Ray ray )
{
    // trace ray
    I, N, material = Trace( ray );
    BRDF = material.albedo / PI;
    // terminate if ray left the scene
    if (ray.NOHIT) return BLACK;
    // terminate if we hit a light source
    if (material.isLight) return BLACK;
    // sample a random light source
    L, Nl, dist, A = RandomPointOnLight();
    Ray lr( I, L, dist );
    if (N·L > 0 && Nl·-L > 0) if (!Trace( lr ))
    {
        solidAngle = ((Nl·-L) * A) / dist2;
        Ld = lightColor * solidAngle * BRDF * N·L;
    }
    // continue random walk
    R = DiffuseReflection( N );
    Ray r( I, R );
    Ei = Sample( r ) * (N·R);
    return PI * 2.0f * BRDF * Ei + Ld;
}
```





# Ray Tracing for Games

0.1s



MAXDEPTH)

```
survive = SurvivalProbability( diffuse );
estimation - doing it properly, closely
df;
```

```
radiance = SampleLight( &rand, I, &L, &lightPos, &normVec);  
if (radiance.x + radiance.y + radiance.z > 0) {&& cout<<" "
```

```
v = 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 ) * (rho

```

```
float3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R);
float3 survive;
```

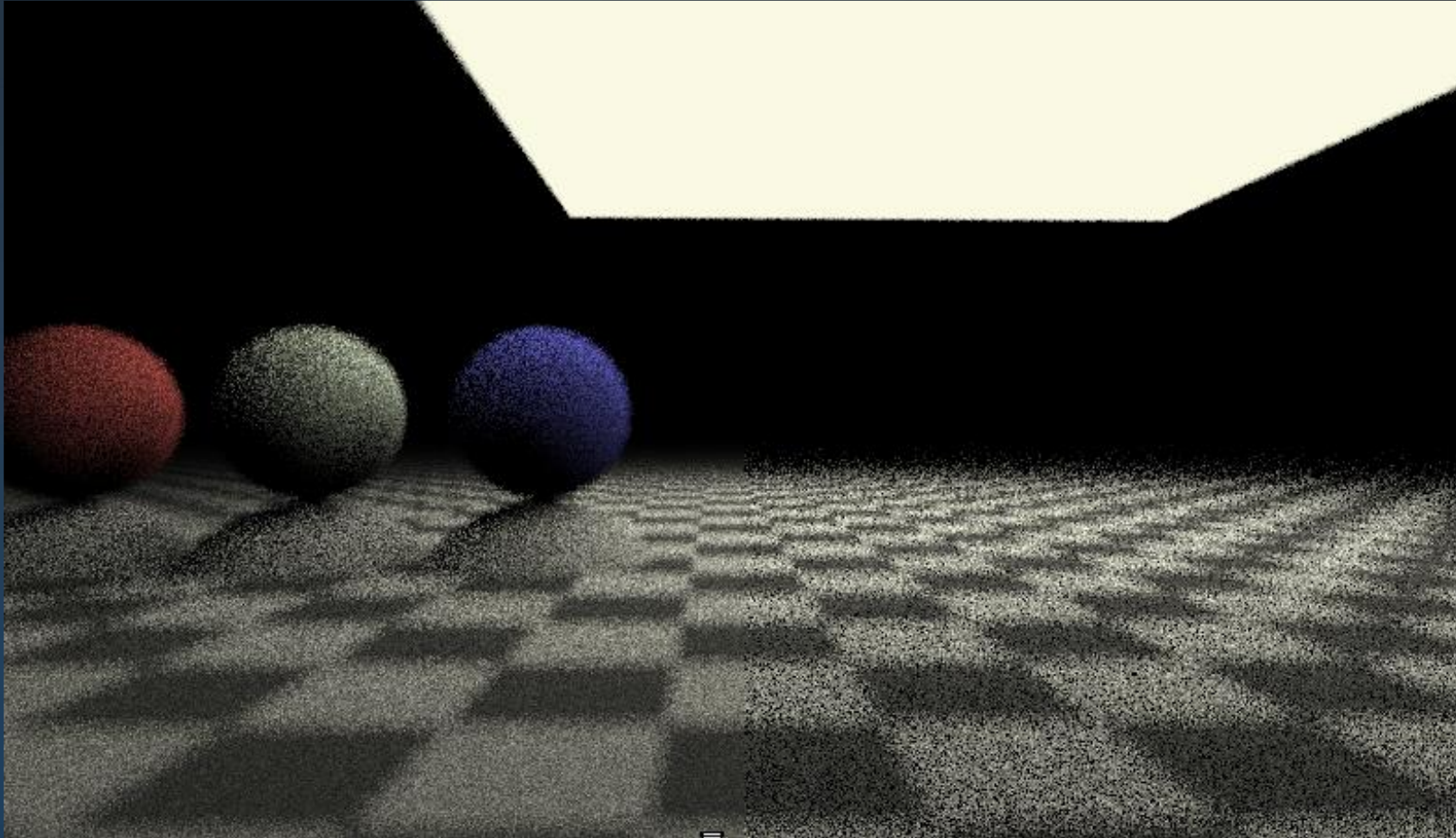
```
pdf;
n = E * brdf * (dot( N, R ) / pdf);
return n;
}
```



# Ray Tracing for Games

```
ics
& (depth < MAXDEPTH)
{
    if (nt < inside ? 1.0f : 0.5f)
    {
        nt = nt / nc; ddn = ddn * ddn;
        float r = sqrt(1.0f - nnt * ddn);
        Vec D, N );
        Vec O(0);
        Vec a = nt - nc, b = nt + nc;
        float Tr = 1 - (R0 + (1 - R0) * r);
        Vec R = (D * nnt - N * (ddn > 0 ? 1 : -1));
        Vec E * diffuse;
        bool = true;
        Vec refl + refr)) && (depth < MAXDEPTH)
        Vec D, N );
        Vec refl * E * diffuse;
        bool = true;
        Vec MAXDEPTH)
        Vec survive = SurvivalProbability( diffuse );
        Vec estimation - doing it properly, closely following
        Vec df;
        Vec radiance = SampleLight( &rand, I, &L, &align, &align,
        Vec e.x + radiance.y + radiance.z) > 0) && (depth <
        Vec w = true;
        Vec at brdfPdf = EvaluateDiffuse( L, N ) * Psurvive;
        Vec at3 factor = diffuse * INVPI;
        Vec at weight = Mis2( directPdf, brdfPdf );
        Vec at cosThetaOut = dot( N, L );
        Vec E * ((weight * cosThetaOut) / directPdf) * (radiance
        Vec random walk - done properly, closely following
        Vec survive)
        Vec ;
        Vec at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
        Vec survive;
        Vec pdf;
        Vec n = E * brdf * (dot( N, R ) / pdf);
        Vec sion = true;

```



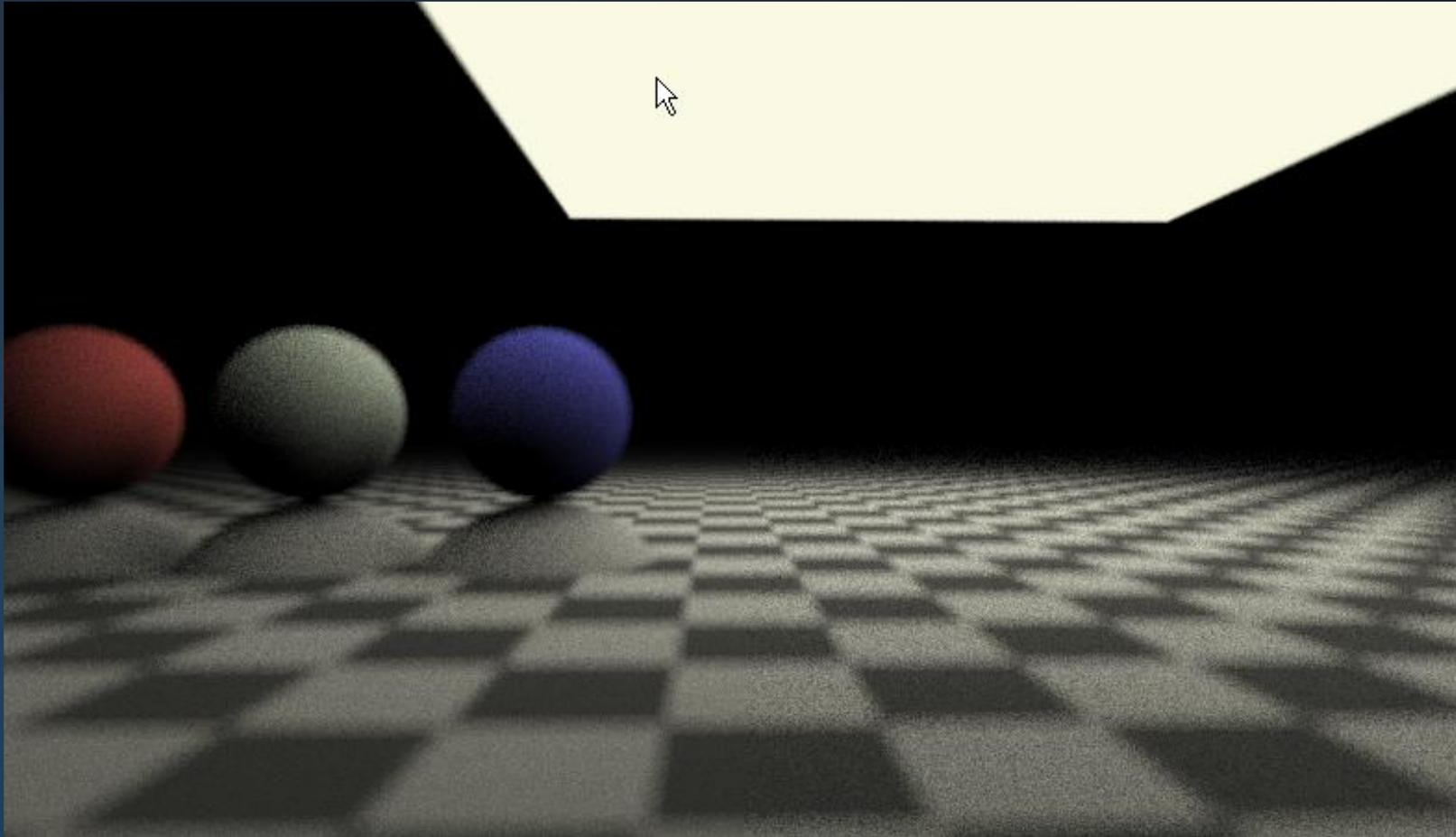
0.5s





# Ray Tracing for Games

2.0s



```
ics
& (depth < MAXDEPTH)
{
    if (inside) {
        nt = inside ? 1.0f : 0.0f;
        nt = nt / nc; ddn = dot(N, D);
        if (ddn < 0) ddn = -ddn;
        cos2t = 1.0f - nnt * nnt;
        t = sqrt(cos2t);
        D, N );
    }
    else {
        at a = nt - nc, b = nt + nc;
        at Tr = 1 - (R0 + (1 - R0) * t);
        Tr) R = (D * nnt - N * (ddn < 0 ? 1 : -1));
    }
    E * diffuse;
    = true;
    = 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;
}
```



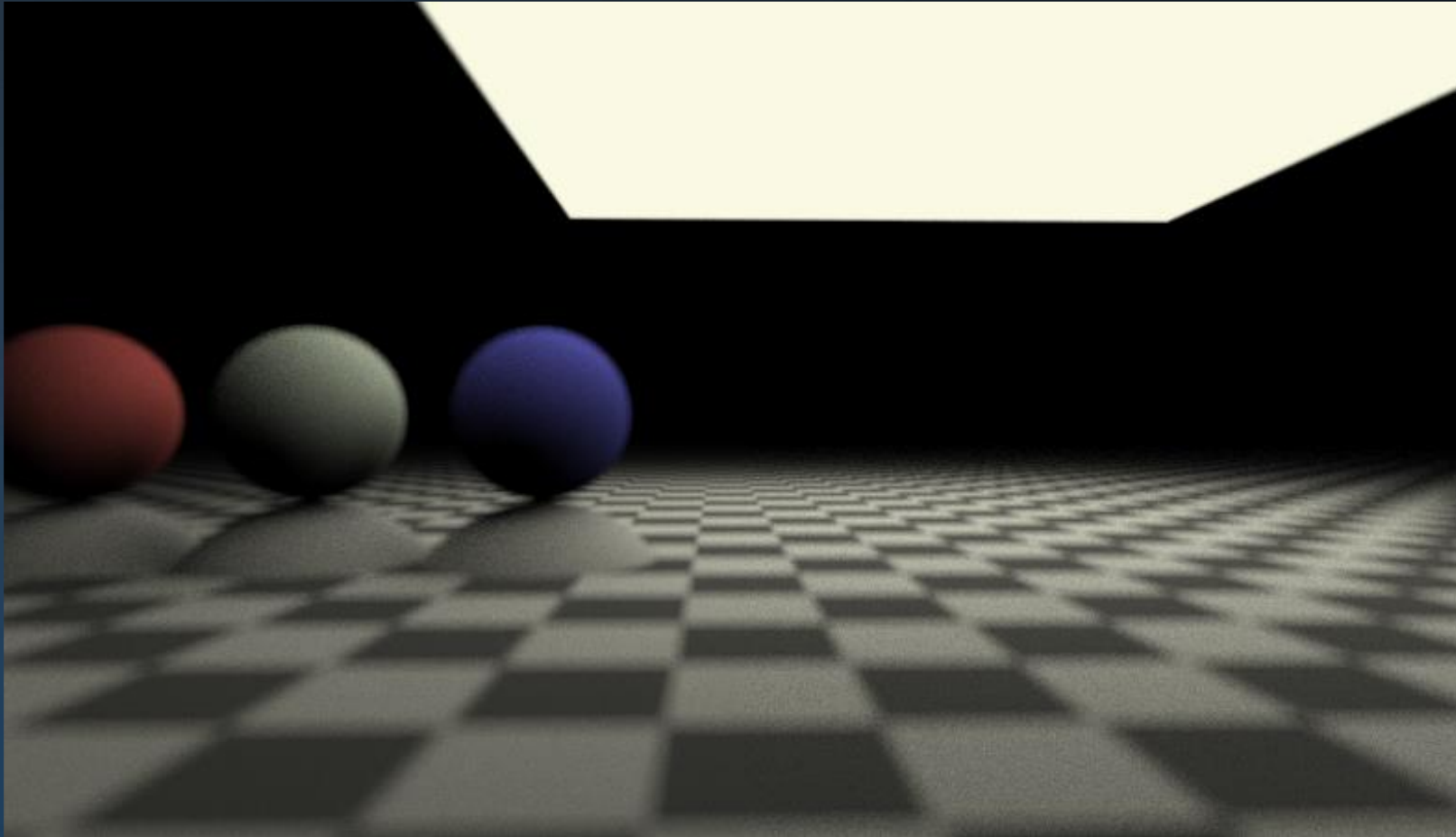
# Ray Tracing for Games

30.0s

```
ics
& (depth < MAXDEPTH)
{
    if (nt < 0)
        nt = inside ? 1 : 0;
    nt = nt / nc; ddn = dot(N, D);
    float r1 = 0, r2 = 0, r3 = 0;
    float cos2t = 1.0f - nnt * nnt;
    float t = sqrt(cos2t);
    D, N );
    D = D * t + N * (ddn * t);
    float a = nt - nc, b = nt + nc;
    float Tr = 1 - (R0 + (1 - R0) * r1 * r1 * r2 * r2);
    float R = (D * nnt - N * (ddn * t));
    if (R < Tr)
    {
        E * diffuse;
        = true;
    }
    else
    {
        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
ive)
;
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;
```





## Next Event Estimation

Some vertices require special attention:

- If the first vertex after the camera is emissive, its energy can't be reflected to the camera.
- For specular surfaces, the BRDF to a light is always 0.

Since a light ray doesn't make sense for specular vertices, we will include emission from a vertex directly following a specular vertex.

The same goes for the first vertex after the camera: if this is emissive, we will also include this.

*This means we need to keep track of the type of the previous vertex during the random walk.*



# Ray Tracing for Games

```
Color Sample( Ray ray, bool lastSpecular )
{
    // trace ray
    I, N, material = Trace( ray );
    BRDF = material.albedo / PI;
    // terminate if ray left the scene
    if (ray.NOHIT) return BLACK;
    // terminate if we hit a light source
    if (material.isLight)
        if (lastSpecular) return material.emissive;
        else return BLACK;
    // sample a random light source
    L, Nl, dist, A = RandomPointOnLight();
    Ray lr( I, L, dist );
    if (N·L > 0 && Nl·-L > 0) if (!Trace( lr ))
    {
        solidAngle = ((Nl·-L) * A) / dist2;
        Ld = lightColor * solidAngle * BRDF * N·L;
    }
    // continue random walk
    R = DiffuseReflection( N );
    Ray r( I, R );
    Ei = Sample( r, false ) * (N·R);
    return PI * 2.0f * BRDF * Ei + Ld;
}
```



# End of PART 9.

```
ics
& (depth < MAXDEPTH)
{
    if (inside ? 1 : 1.25 * nnt)
    {
        nt = nt / nc, ddn = ddn * nc;
        rps2t = 1.0f - nnt * nnt;
        D, N );
    }
    {
        at a = nt - nc, b = nt + nc;
        at Tr = 1 - (R0 + (1 - R0) * rps2t);
        Tr) R = (D * nnt - N * (ddn < 0));
    }
    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;
}
```

