

```
ics
& (depth < MAXDEPTH))
{
    if (inside & !isrefl)
    {
        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 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
    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);
    ion = true;
```

7

Ray Tracing for Games

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

Welcome!



Ray Tracing for Games



GLOBAL GAME JAM

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 1

LAB 2



work @ home

End result day 2:

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

LAB 3

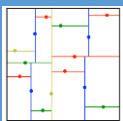


SIMD
applied SIMD
SIMD triangle
SIMD AABB

LAB 4

End result day 3:
A 5x faster tracer.

LAB 5



refitting
top-level BVH
threaded building

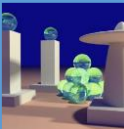
LAB 6

End result day 4:
A real-time tracer.

LAB 7



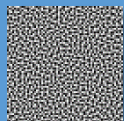
path tracing



LAB 8

End result day 5:
Cook or Kajiya.

LAB 9



importance
sampling
next event
estimation

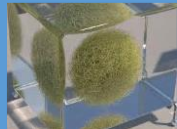
LAB 10

End result day 6:
Efficiency.

LAB 11



path guiding



LAB 10

End result day 6:
Great product.



```

ics
& (depth < MAXDEPTH)) {
    // Inside?
    if (inside ? 1 : 1.0f - 0.5f * nnt) {
        nt = nt / nc; ddn = ddn * nc;
        cos2t = 1.0f - nnt * ddn;
        D, N );
    }
    // Outside?
    if (a = nt - nc, b = nt + nc, c = 2 * nc) {
        at Tr = 1 - (R0 + (1 - R0) * ddn);
        Tr) R = (D * nnt - N * (ddn > 0) ? 1 : -1);
    }
    // Diffuse
    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, &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:

- Maths
- Monte-Carlo
- The Rendering Equation
- Applied



Math Symbols in Graphics Papers

$f(x) = x^2$: generic function, where ‘x’ is the parameter.

\sum (sigma): ‘for loop’.

E.g.:

$$v = \sum_{i=1}^4 f(x_i) = f(x_1) + f(x_2) + f(x_3) + f(x_4)$$

\prod (‘product’): also a for loop.

E.g.:

$$v = \prod_{i=1}^4 f(x_i) = f(x_1) f(x_2) f(x_3) f(x_4)$$

\int (‘integral’).

E.g.:

$$a = \int_0^{\pi} \sin x \, dx$$

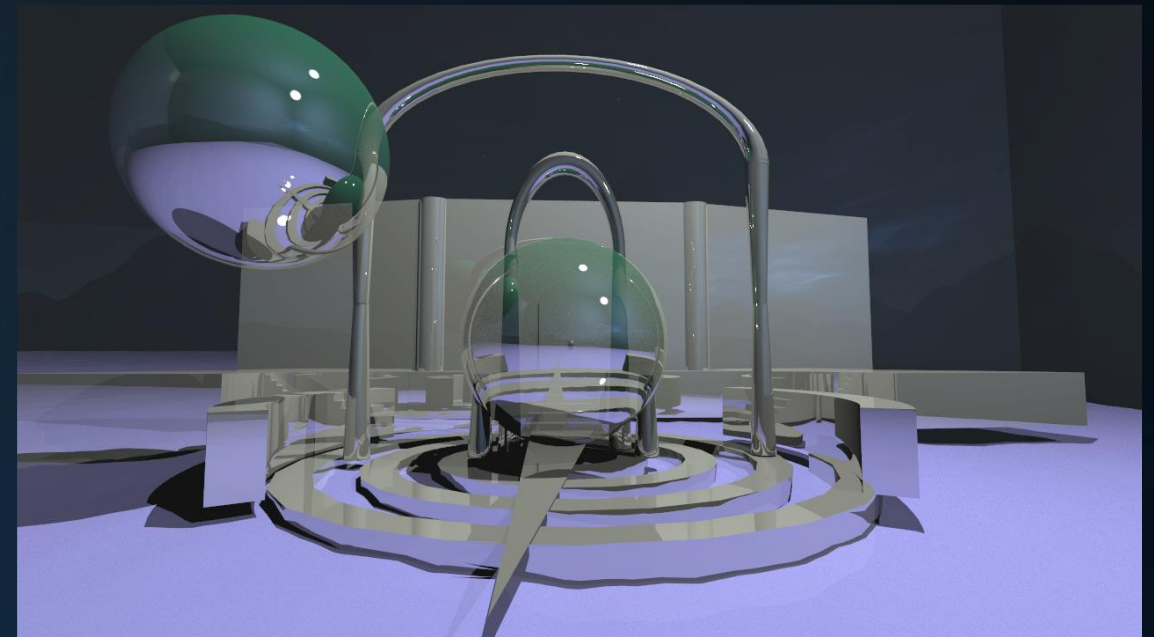
$\mathbb{R}^2, \mathbb{R}^3$: 2D and 3-dimensional space.

ω, Ω : ‘omega’.

Θ : ‘theta’, e.g. $\cos \theta = N \cdot L$



Whitted



```
xy: (1216, 682)
rgb: (0.476, 0.485, 0.505, 1.000)
before gamma correction

...
if (depth < MAXDEPTH)
{
    if (inside ? 1 : 0)
    {
        nt = nt / nc;
        cos2t = 1.0f - nt;
        D, N);
    }
    else
    {
        at a = nt - nc;
        at Tr = 1 - (R * R);
        Tr) R = (D * n
    }
    E * diffuse;
    = true;
    refl + refr)) &
    D, N);
    refl * E * dif
    = true;
    MAXDEPTH)
    survive = Surv
    estimation -
    df;
    radiance = Sam
    e.x + radiance.y + radiance.z) > 0) & R * R)
    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 3rd order
    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

Anti-aliasing

Adding anti-aliasing to a Whitted-style ray tracer:

Send multiple primary rays through each pixel and average their result.



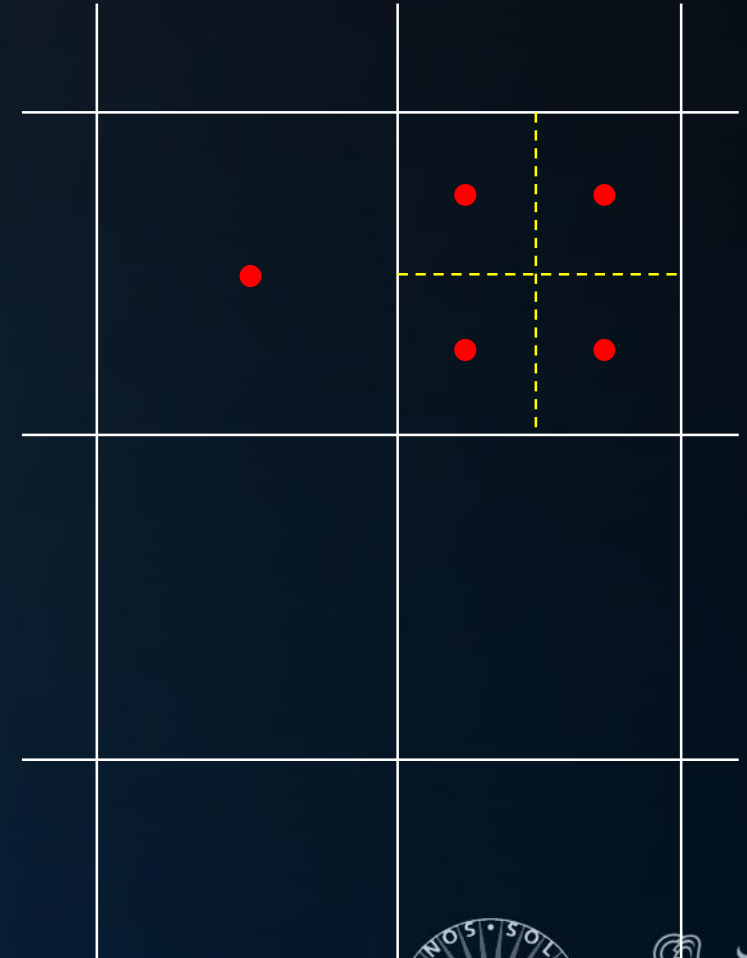
Anti-aliasing – Sampling Patterns

Adding anti-aliasing to a Whitted-style ray tracer:

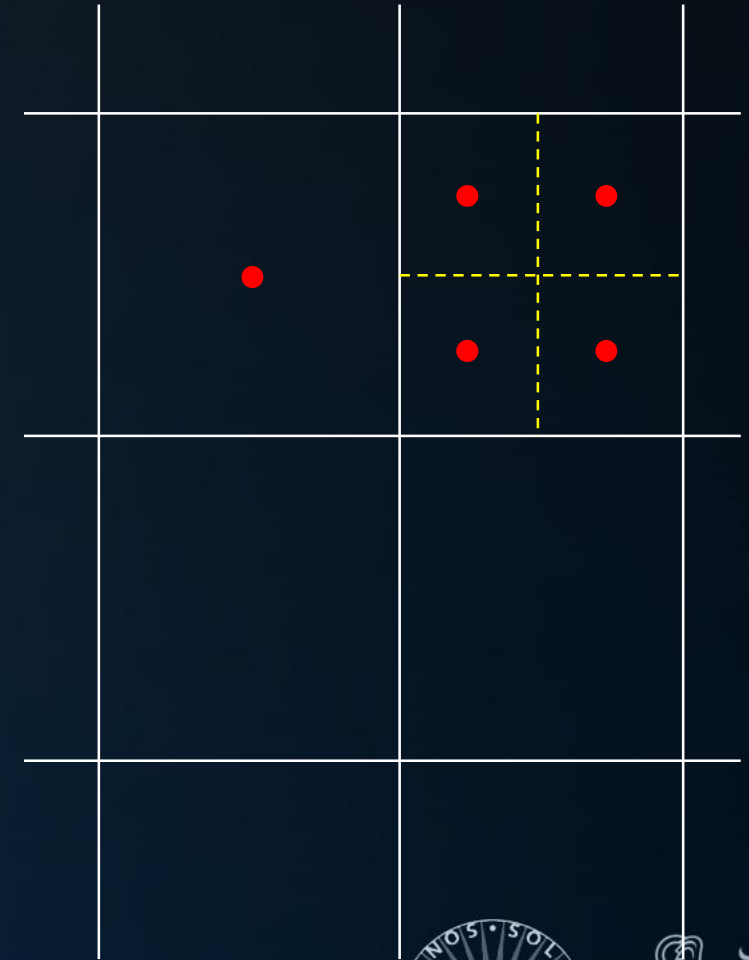
Send multiple primary rays through each pixel, and average their result.

Problem:

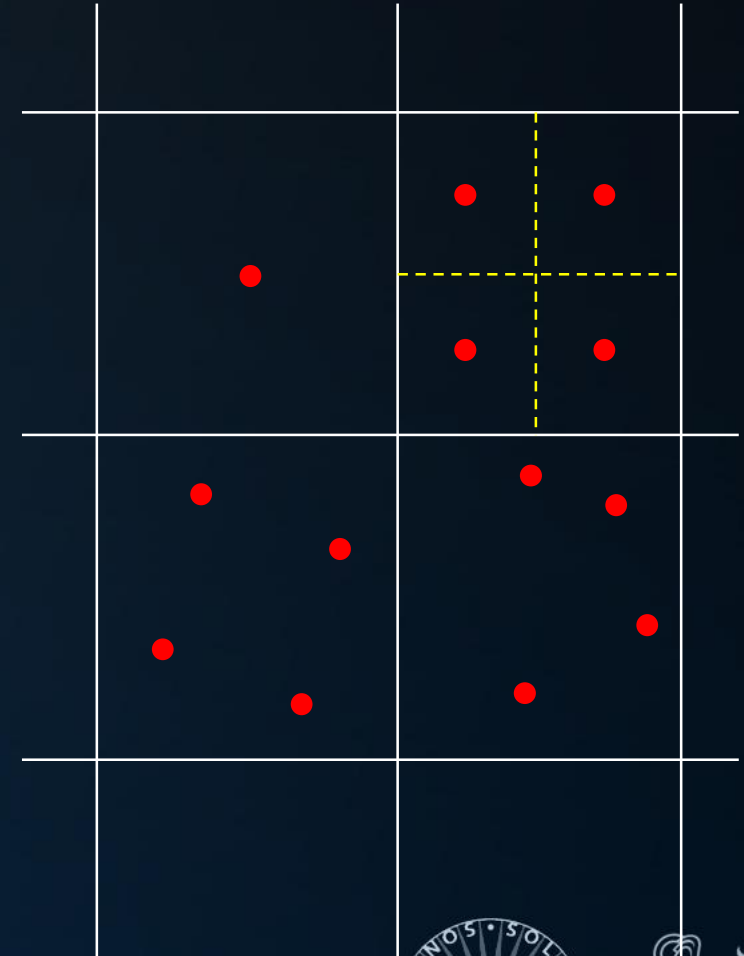
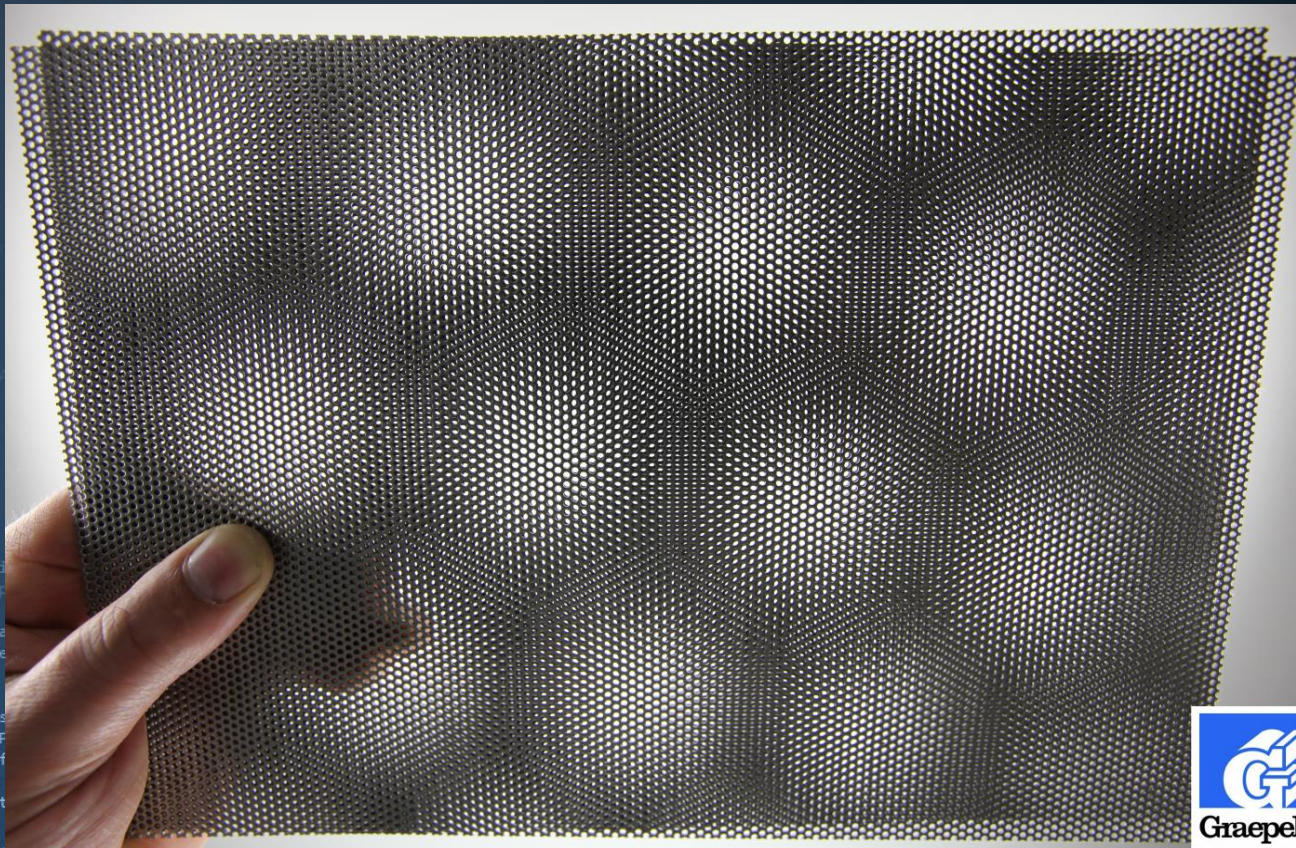
- How do we aim those rays?
- What if all rays return the same color?



Anti-aliasing – Sampling Patterns



Anti-aliasing – Sampling Patterns



Distribution Ray Tracing*

```
ics
& (depth < MAXDEPTH)
{
    if (nt < inside ? 1.0f : 0.5f)
    {
        nt = nt / nc; ddn = ddn * ddn;
        float r = 1.0f - nnt * ddn;
        float D, N );
        D, N );
    }
    else
    {
        float a = nt - nc, b = nt + nc;
        float Tr = 1 - (R0 + (1 - R0) * r);
        float R = (D * nnt - N * (ddn * r));
        E * diffuse;
        = true;
    }
    if (refl + refr) && (depth < MAXDEPTH)
    {
        D, N );
        refl * E * diffuse;
        = true;
    }
    MAXDEPTH)

```

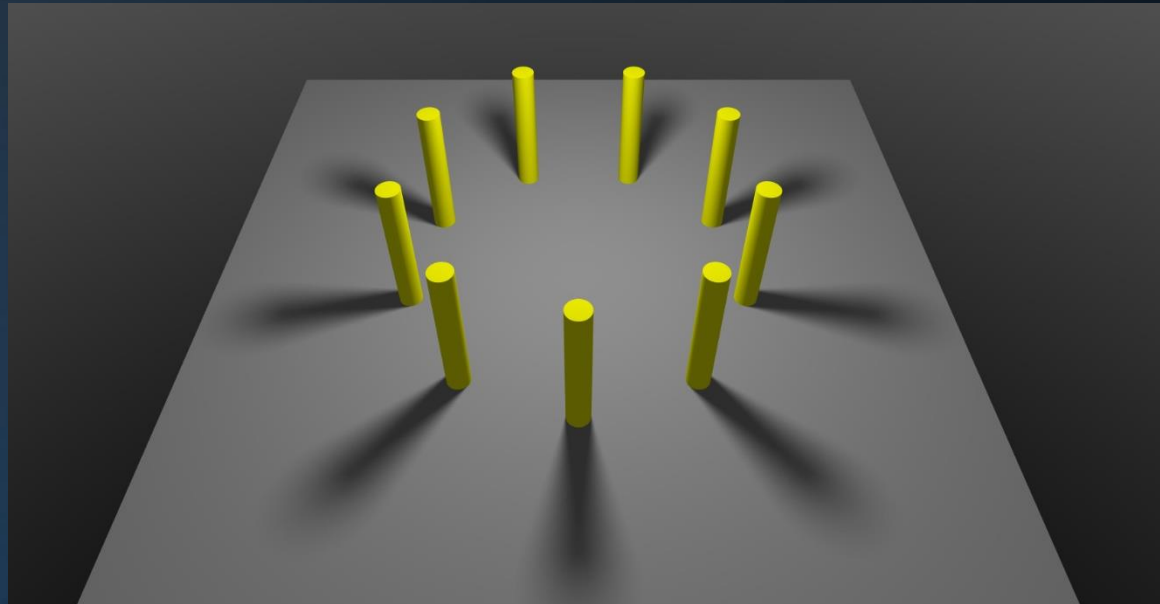
```
survive = SurvivalProbability( diffuse );
estimation - doing it properly, closely following
df;
radiance = SampleLight( &rand, I, &L, &light );
e.x + radiance.y + radiance.z) > 0) && (depth <
w = true;
at brdfPdf = EvaluateDiffuse( L, N ) * Psurvive;
at3 factor = diffuse * INVPI;
at weight = Mis2(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

```

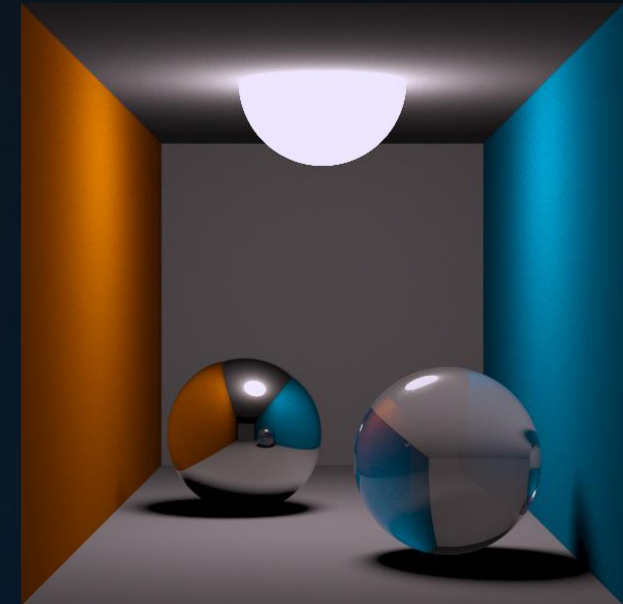
```

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

```



Soft shadows



*: Distributed Ray Tracing, Cook et al., 1984

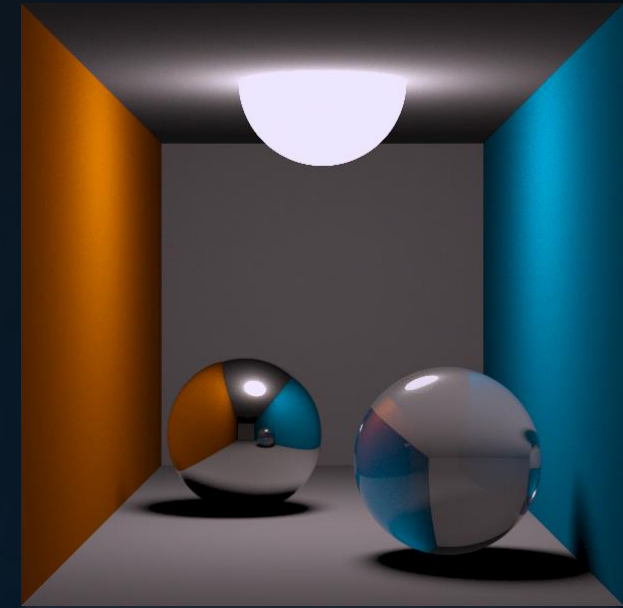


Distribution Ray Tracing*

```
ics
& (depth < MAXDEPTH)
{
    if (nt < inside ? 1.0f : 0.5f)
    {
        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) * 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, &lightPdf );
    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, &R, &pdf );
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    ion = true;
}
```



Glossy reflections



*: Distributed Ray Tracing, Cook et al., 1984



Distribution Ray Tracing*

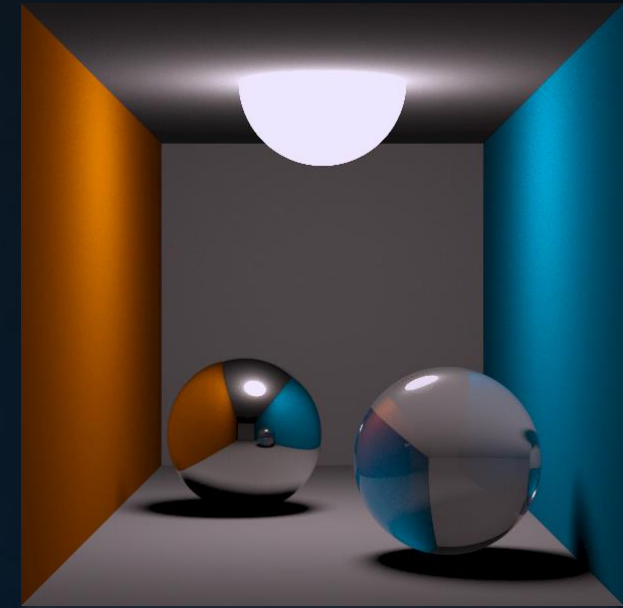
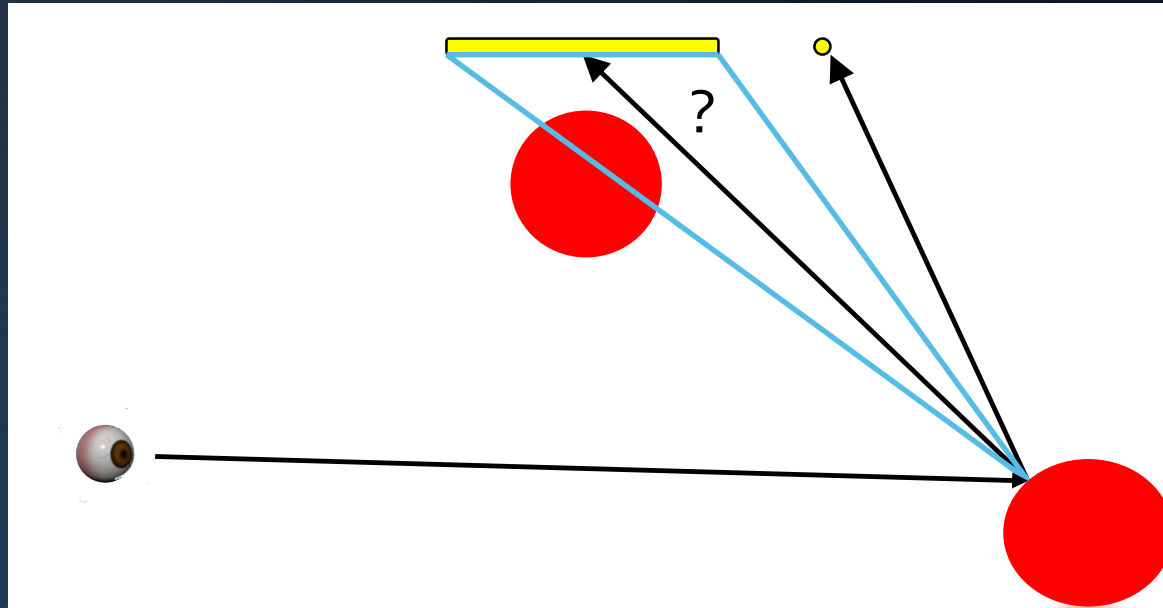
```
ics
& (depth < MAXDEPTH)
{
    if (nt < nc)
    {
        nt = inside ? 1.0f : 0.0f;
        nt = nt / nc; ddn = 1.0f - nt;
        float r2 = 1.0f - nnt * nnt;
        float theta = 2 * M_PI * r2;
        float phi = sqrt(1 - r2);
        float D, N;
        D = (float) rand() * 255;
        N = (float) rand() * 255;
        float a = nt - nc, b = nt + nc;
        float Tr = 1 - (R0 + (1 - R0) * phi);
        float R = (D * nnt - N * (ddn * phi));
        if (R < 0)
        {
            R = 0;
            N = 0;
        }
        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, &align, &pdf );
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
vive)

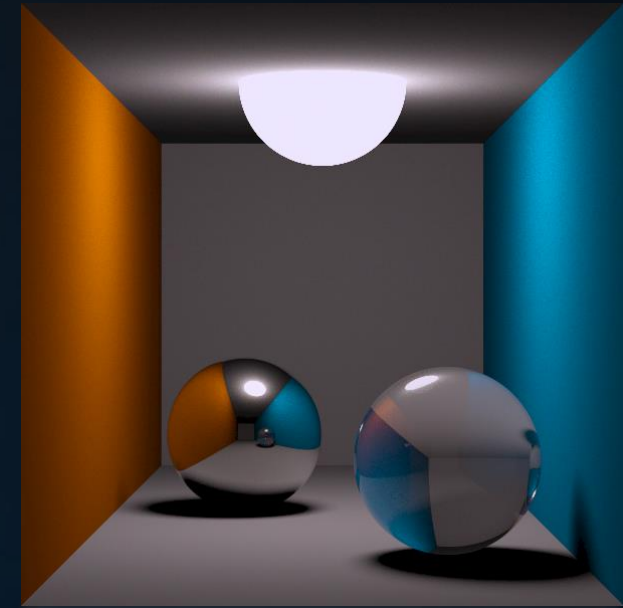
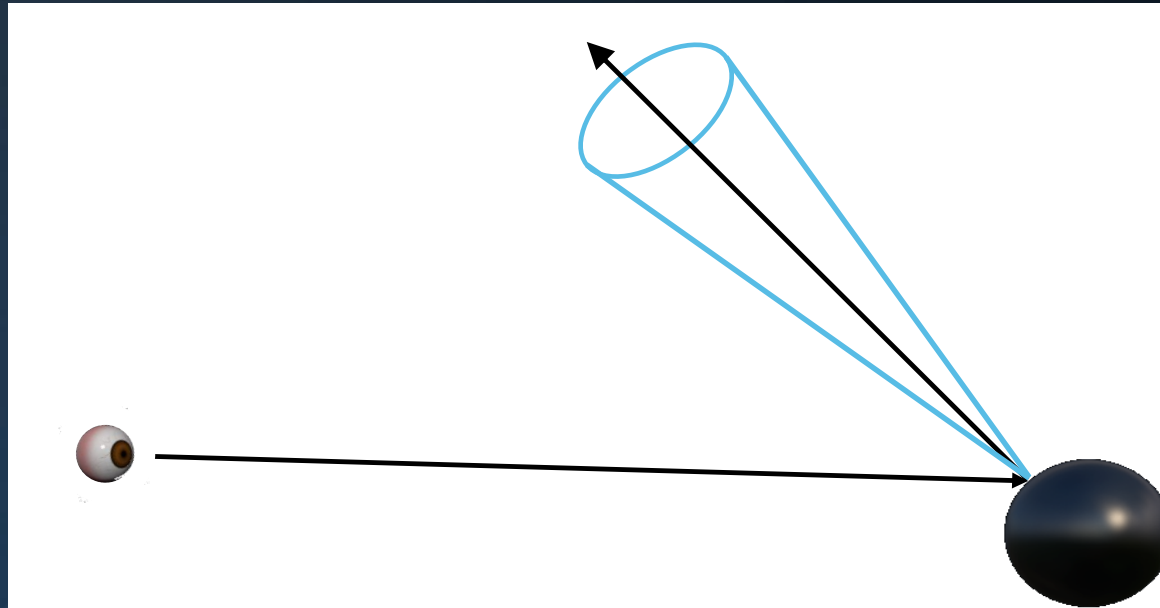
```

*: Distributed Ray Tracing, Cook et al., 1984



Distribution Ray Tracing*

```
ics
& (depth < MAXDEPTH)
{
    if (inside ? 1 : 0)
    {
        nt = nt / nc; ddn = ddn * ddn;
        r = 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 * r));
    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
    vive)
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    ion = true;
```



*: Distributed Ray Tracing, Cook et al., 1984



Area Lights

Visibility of an area light source:

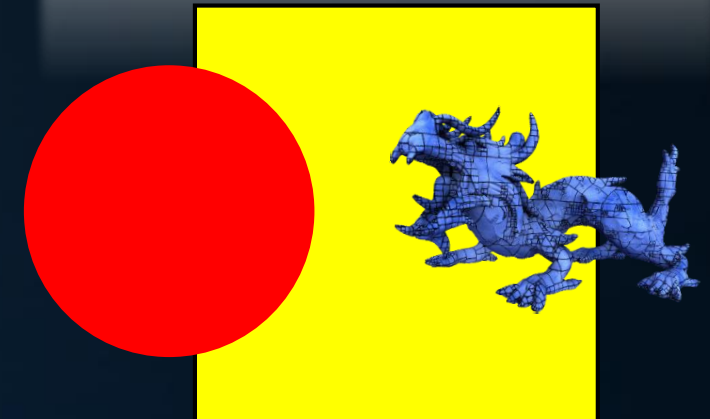
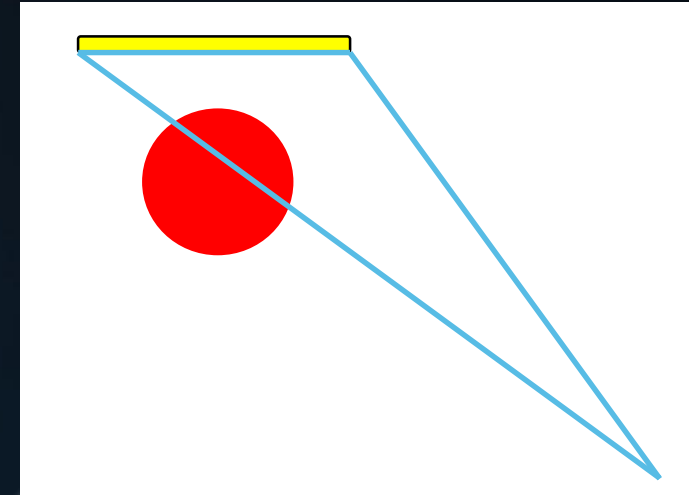
$$V_A = \int_A V(x) dx$$

Perfect ('analytical') solution case 1:

$$V_A = A_{light} - A_{light \cap sphere}$$

Analytical solution case 2:

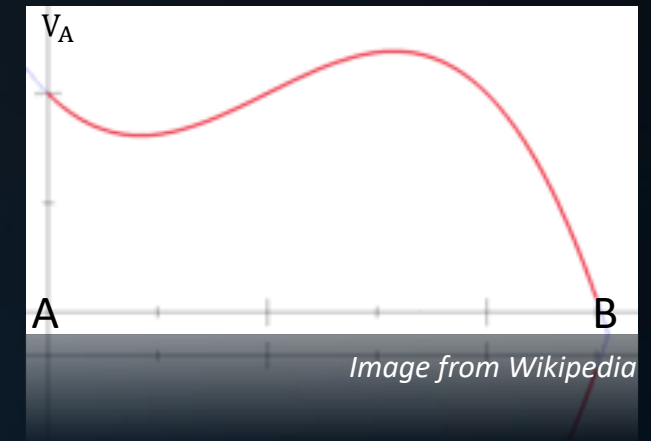
$$V_A = ?$$



Approximating Integrals

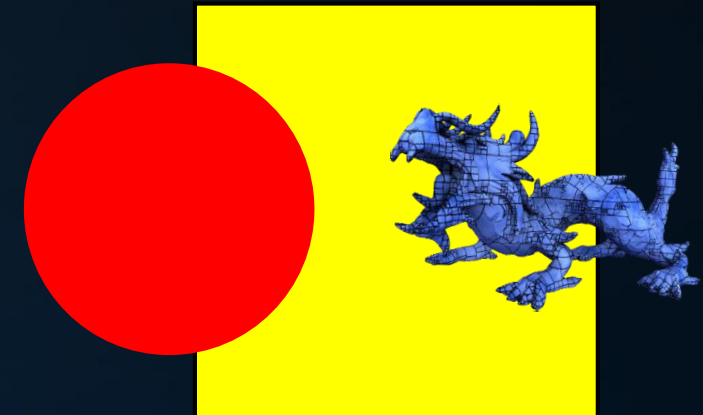
An integral can be approximated as a Riemann sum:

$$V_A = \int_A^B f(x) dx \approx \sum_{i=1}^N f(t_i) \Delta_i, \text{ where } \sum_{i=1}^N \Delta_i = B - A$$



Alternatively, we can approximate an integral by taking *random* samples:

$$V_A = \int_A^B f(x) dx \approx \frac{B - A}{N} \sum_{i=1}^N f(X_i)$$



Monte Carlo Integration of Area Light Visibility

To estimate the visibility of an area light source, we take N random point samples.

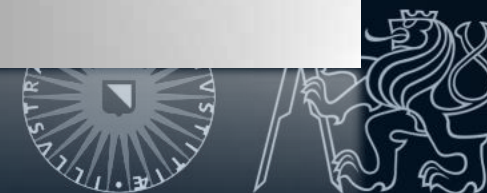
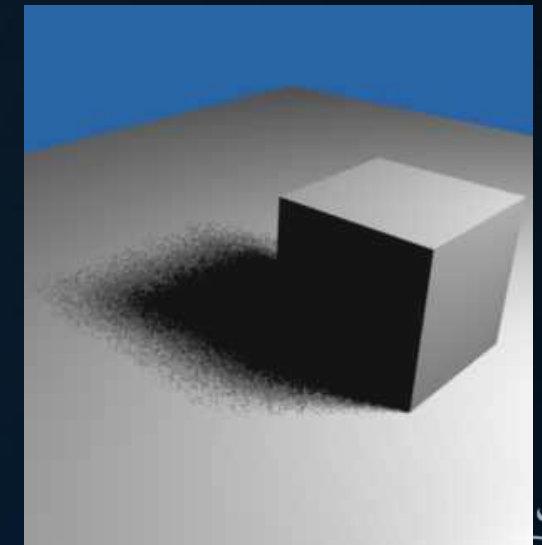
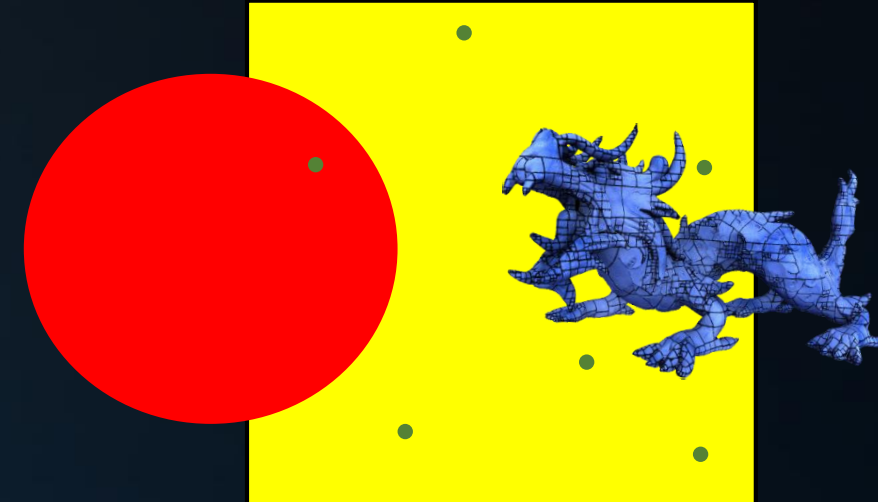
In this case, 5 out of 6 samples are unoccluded:

$$V \approx \frac{1}{6}(1 + 1 + 1 + 0 + 1 + 1) = \frac{5}{6}$$

Properly formulated using a MC integrator:

$$V = \int_{S^2} V(p) dp \approx \frac{1}{N} \sum_{i=1}^N V(P)$$

With a small number of samples, the *variance* in the estimate shows up as noise in the image.



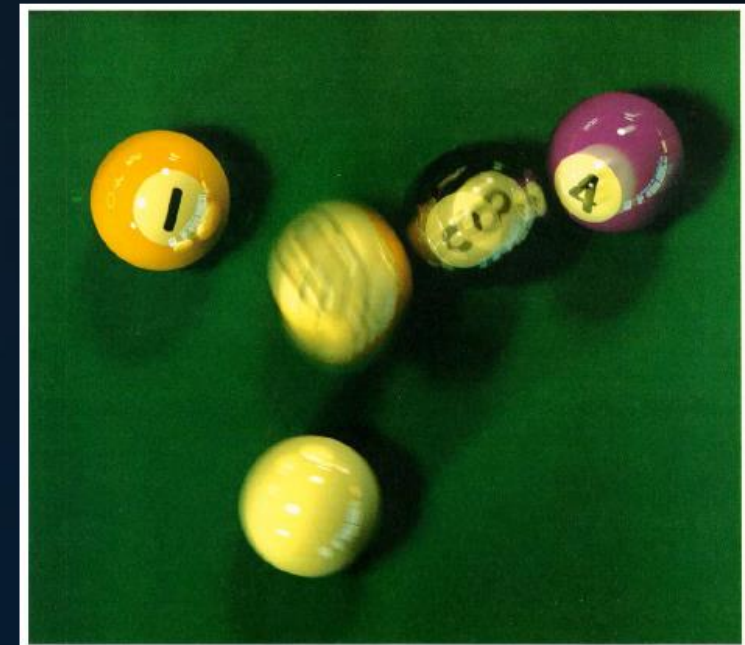
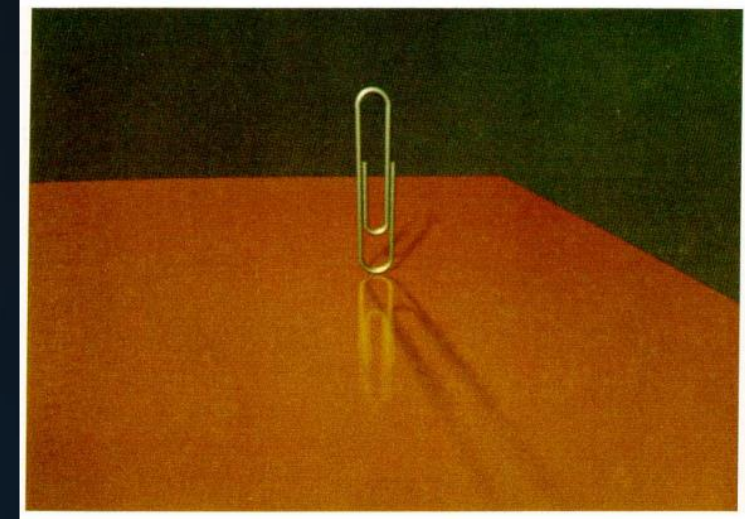
Distribution Ray Tracing

Key concept of distribution ray tracing:

We estimate integrals using Monte Carlo integration.

Integrals in rendering:

- Area of a pixel
- Lens area (aperture)
- Frame time
- Light source area
- Cones for glossy reflections
- ...



Agenda:

- Maths
- Monte-Carlo
- The Rendering Equation
- Applied



God's Algorithm

1 room

1 bulb

100 watts

10^{20} photons per second

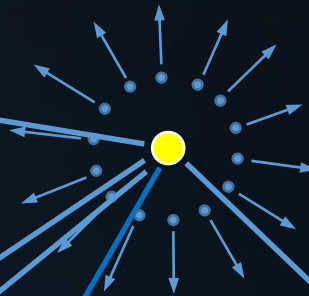
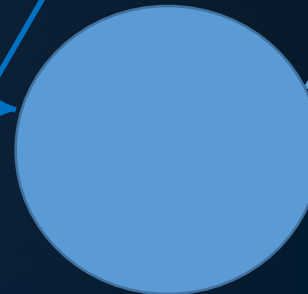
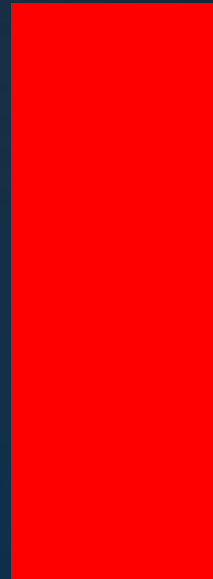
Photon behavior:

- Travel in straight lines
- Get absorbed, or change direction:
 - Bounce (random / deterministic)
 - Get transmitted
- Leave into the void
- Get detected



Ray Tracing for Games

```
ics
& (depth < MAXDEPTH)
{
    if (inside) {
        nt = nt / nc; ddn = ddn * ddn;
        cos2t = 1.0f - nnt * ddn;
        D, N );
    }
    if (a = nt - nc, b = nt + nc, c = 0.0f,
    at Tr = 1 - (R0 + (1 - R0) * rand(0, 1));
    if (Tr) R = (D * nnt - N * (ddn * nnt));
    else {
        E * diffuse;
        = true;
    }
    if (refl + refr) && (depth < MAXDEPTH)
    {
        D, N );
        refl * E * diffuse;
        = true;
    }
    if (MAXDEPTH)
    {
        survive = SurvivalProbability( diffuse );
        estimation - doing it properly, closely following Small's
        if;
        radiance = SampleLight( &rand, I, &L, &light, &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
            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;
    }
}
```



God's Algorithm - Mathematically

A photon may arrive at a sensor after travelling in a straight line from a light source to the sensor:

$$L(s \leftarrow x) = L_E(s \leftarrow x)$$

Or, it may be reflected by a surface towards the sensor:

$$L(s \leftarrow x) = \int_{\Omega} f_r(s \leftarrow x \leftarrow y) L(x \leftarrow y) G(x \leftrightarrow y) dy$$

Those are the options.

Adding direct and indirect illumination together:

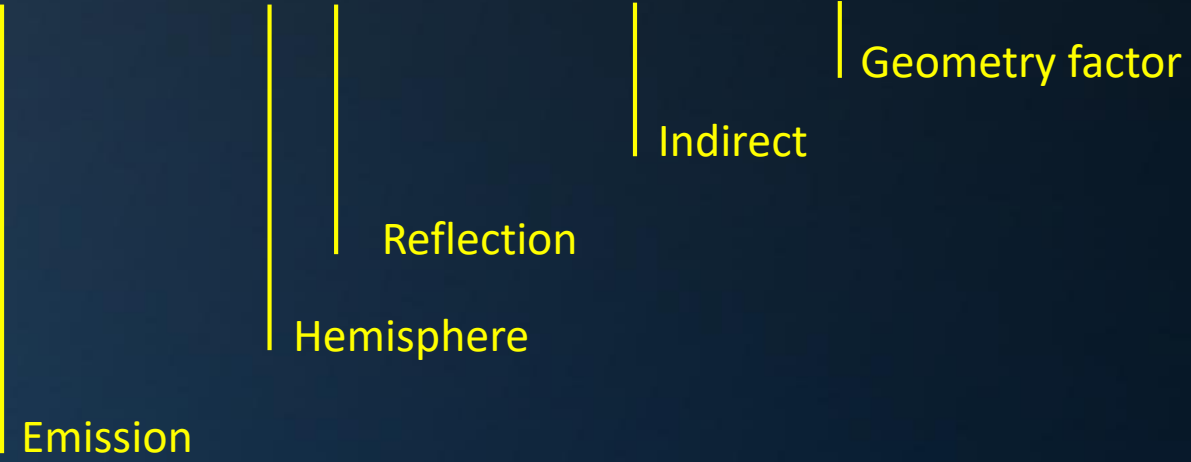
$$L(s \leftarrow x) = L_E(s \leftarrow x) + \int_A f_r(s \leftarrow x \leftarrow y) L(x \leftarrow y) G(x \leftrightarrow y) dA(y)$$



God's Algorithm - Mathematically

```
...ics
& (depth < MAXDEPTH)
{
    if (inside ? 1 : 1.25)
    {
        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) *
    (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 following
    df;
    radiance = SampleLight( &rand, I, &L, &lightPdf,
    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) * (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;
}
```

$$L(s \leftarrow x) = L_E(s \leftarrow x) + \int_A f_r(s \leftarrow x \leftarrow y) L(x \leftarrow y) G(x \leftrightarrow y) dA(y)$$



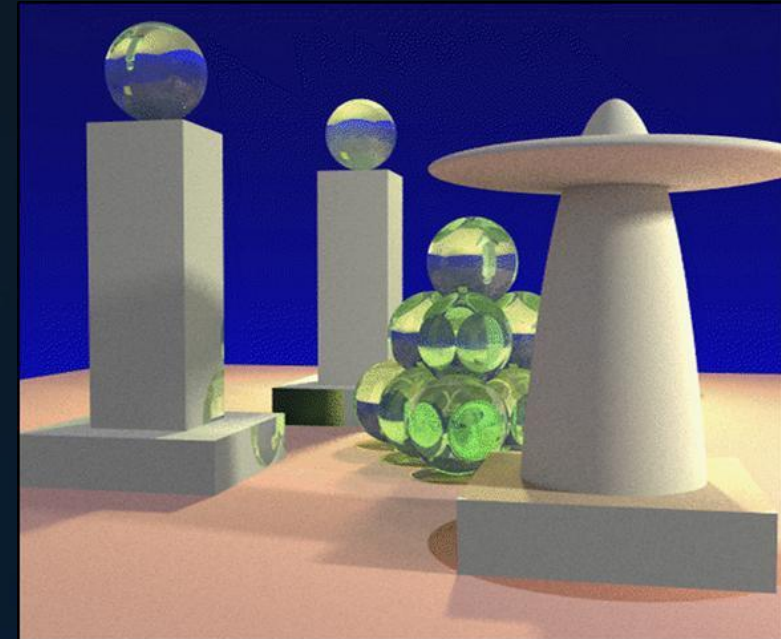
$$L(s \leftarrow x) = L_E(s \leftarrow x) + \int_A f_r(s \leftarrow x \leftarrow y) L(x \leftarrow y) G(x \leftrightarrow y) dA(y)$$

The Rendering Equation*:

- Light transport from lights to sensor
- Recursive
- Physically based

The equation allows us to determine to which extend rendering algorithms approximate real-world light transport.

*: The Rendering Equation, Kajiya, 1986




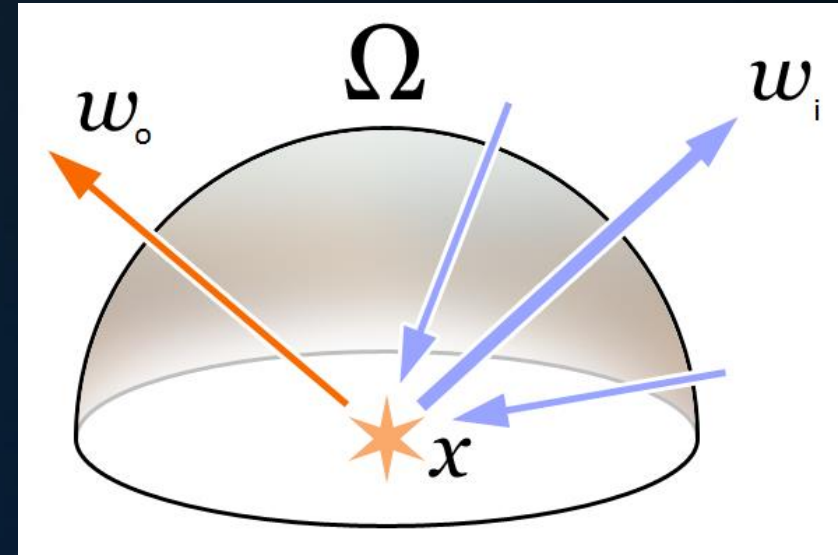
Light Transport

$$L(s \leftarrow x) = L_E(s \leftarrow x) + \int_A f_r(s \leftarrow x \leftarrow y) L(x \leftarrow y) G(x \leftrightarrow y) dA(y)$$

The above formulation integrates over all the points in the scene.
An alternative formulation:

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


$$L_o = L_e + \int_{\Omega} L_i \cdot f_r \cdot \cos \theta \cdot d\omega$$

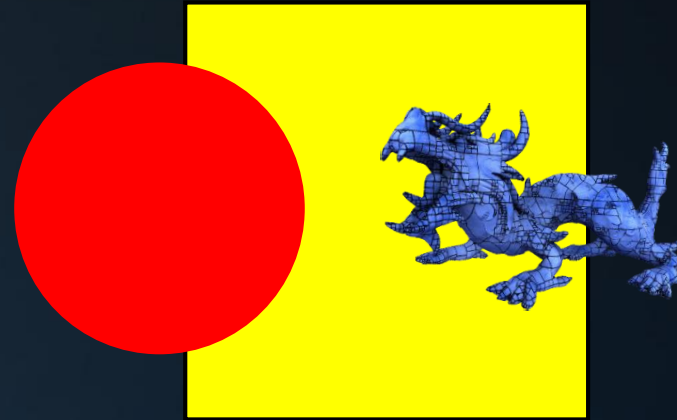


Agenda:

- Maths
- Monte-Carlo
- The Rendering Equation
- Applied



$$V_A = \int_A^B f(x) dx \approx \frac{B-A}{N} \sum_{i=1}^N f(X_i)$$



```

ics
& (depth < MAXDEPTH)
{
    if (inside ? 1 : 0)
    {
        nt = nt / nc; ddn = ddn * ddn;
        r2t = 1.0f - nnt * ddn;
        D, N );
    }
}

```

```

at a = nt - nc; b = nt + nc;
at Tr = 1 - (R0 + (1 - R0) * r2t);
Tr) R = (D * nnt - N * (ddn * r2t));

```

```

E * diffuse;
= true;

```

```

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

```

```

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

```

```

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 ) *
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;

```

Color DirectIllumination(I, N)

Color sum = BLACK

for each light l

L = light.pos - I

dist2 = L.squareLength()

if (!IsOccluded(I, L, sqrt(dist2)))

normalize(L)

sum += (l.color * dot(N,L))/dist2)

return sum

Color DirectIllumination(I, N)

Color sum = BLACK

for each light l

P = l.RandomPointOnLight()

L = P - I

dist2 = L.squareLength()

if (!IsOccluded(I, L, sqrt(dist2)))

normalize(L)

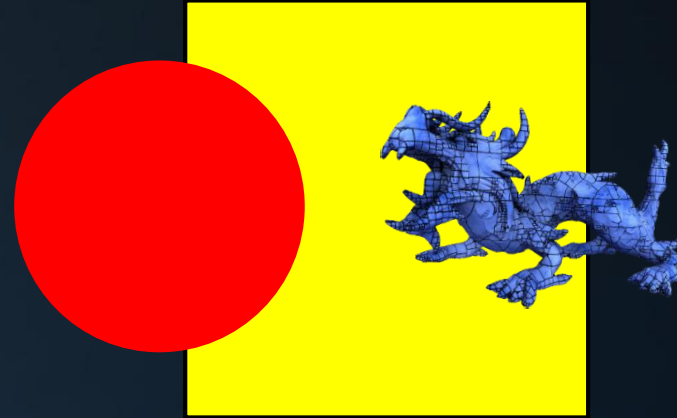
sum += (l.color * dot(N,L))/dist2)

return sum



Ray Tracing for Games

$$V_A = \int_A^B f(x) dx \approx \frac{B-A}{N} \sum_{i=1}^N f(X_i)$$



```

ics
& (depth < MAXDEPTH)
{
    if (inside ? 1 : 0)
    {
        nt = nt / nc; ddn = ddn * ddn;
        pos2t = 1.0f - nnt * ddn;
        D, N );
    }
}

```

```

at a = nt - nc; b = nt + nc;
at Tr = 1 - (R0 + (1 - R0) * pos2t);
Tr) R = (D * nnt - N * (ddn * pos2t));

```

```

E * diffuse;
= true;

```

```

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

```

```

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

```

```

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

```

```

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

```

```

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

```

```

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

```

```

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

```

```

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

```

```

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

```

```

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

```

```

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

```

Color DirectIllumination(I, N)

Color sum = BLACK

for each light l

L = light.pos - I

dist2 = L.squareLength()

if (!IsOccluded(I, L, sqrt(dist2)))

normalize(L)

sum += (l.color * dot(N,L))/dist2)

return sum

Color DirectIllumination(I, N)

l = SelectRandomLight()

P = l.RandomPointOnLight()

L = P - I

dist2 = L.squareLength()

if (!IsOccluded(I, L, sqrt(dist2)))

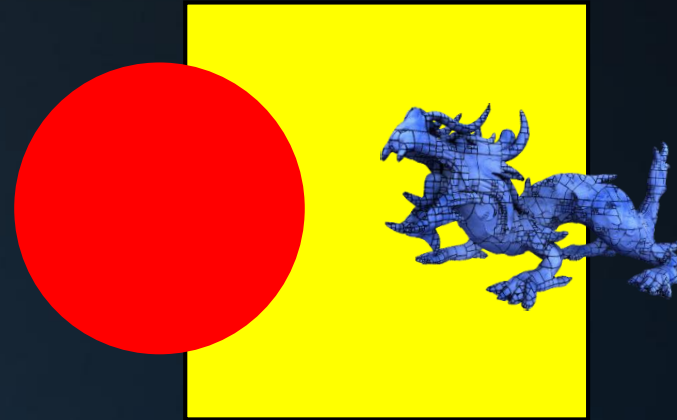
normalize(L)

return l.color * lightCount * dot(N,L) / dist2

return BLACK



$$V_A = \int_A^B f(x) dx \approx \frac{B-A}{N} \sum_{i=1}^N f(X_i)$$



```

ics
& (depth < MAXDEPTH)
{
    if (inside ? 1 : 0)
    {
        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) * ddn);
        (Tr) R = (D * nnt - N * (ddn * nnt));
    }
    E * diffuse;
    = true;
    if (refl + refr) && (depth < MAXDEPTH)
    {
        D, N );
        refl * E * diffuse;
        = true;
    }
    MAXDEPTH)
    survive = SurvivalProbability(
    estimation - doing it properly, close
    if;
    radiance = SampleLight( &rand, I, &L, &N );
    e.x + radiance.y + radiance.z) > 0) && (depth <
    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 a
    vive)
    ;
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    sion = true;
}

```

float3 accumulator =
 new float3[512 * 512];
 memset(accumulator, 0, 512 * 512 * 4);

scale = 1.0f / spp;
 for(int y = 0; y < 512; y++)
 for(int x = 0; x < 512; x++)
 float3 p = accumulator[x + y * 512];
 p *= scale;
 int red = min(1.f, p.x) * 255;
 int green = min(1.f, p.y) * 255;
 int blue = min(1.f, p.z) * 255;

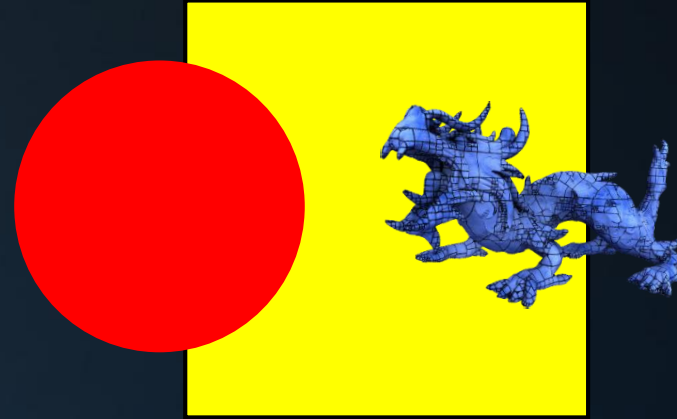
```

Color DirectIllumination( I, N )
{
    l = SelectRandomLight()
    P = l.RandomPointOnLight()
    L = P - I
    dist2 = L.squareLength()
    if (!IsOccluded( I, L, sqrt(dist2) ))
        normalize(L)
        return l.color * lightCount * dot(N,L) / dist2
    return BLACK
}

```



$$V_A = \int_A^B f(x) dx \approx \frac{B-A}{N} \sum_{i=1}^N f(X_i)$$



```

ics
& (depth < MAXDEPTH)
{
    if (inside ? 1 : 0)
    {
        nt = nt / nc; ddn = ddn * ddn;
        r2 = 1.0f - nnt * ddn;
        D, N );
    }
}

```

```

at a = nt - nc; b = nt + nc;
at Tr = 1 - (R0 + (1 - R0) * r2);
Tr) R = (D * nnt - N * (ddn * r2)

```

```

E * diffuse;
= true;

```

```

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

```

```

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

```

```

MAXDEPTH)

```

```

survive = SurvivalProbability(
estimation - doing it properly, close

```

```

if;
radiance = SampleLight( &rand, I, &L, N );
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 a
vive)

```

```

;
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
survive;
pdf;

```

```

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

```

```

float3 accumulator =
    new float3[512 * 512];
memset( accumulator, 0, 512 * 512 * 4 );

scale = 1.0f / spp;
for( int y = 0; y < 512; y++ )
    for( int x = 0; x < 512; x++ )
        float3 p = accumulator[x + y * 512];
        p *= scale;
        int red    = sqrtf( min( 1.f, p.x ) ) * 255;
        int green  = sqrtf( min( 1.f, p.y ) ) * 255;
        int blue   = sqrtf( min( 1.f, p.z ) ) * 255;

```

Also take a look at tonemapping, e.g. ACES filmic tonemapping.



Other Uses for Monte-Carlo:

- Dielectrics: sample reflection *or* refraction.
- Glossy reflections: jitter the reflected ray.
- Diffuse interreflections: ...
- Full Rendering Equation: ...

That noise tho...

```
ics
& (depth < MAXDEPTH))
{
    if (nt < 0)
        nt = inside ? 1 - r : r;
    nt = nt / nc; ddn = dot(N, L);
    cos2t = 1.0f - nnt * nnt;
    D, N );
    )
    {
        at a = nt - nc, b = nt + nc;
        at Tr = 1 - (R0 + (1 - R0) * 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 Small's
            if;
            radiance = SampleLight( &rand, I, &L, &lightPos );
            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)
            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;
        }
    }
}
```



End of PART 7.

```
ics
& (depth < MAXDEPTH)
{
    if (inside ? 1 : 1.0f - nt)
    {
        nt = nt / nc; ddn = ddn * nc;
        r2 = 1.0f - nnt * nnt;
        r = sqrt(r2);
        D, N );
    }
    {
        a = nt - nc; b = nt + nc;
        Tr = 1 - (R0 + (1 - R0) * r);
        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, &light );
        e.x + radiance.y + radiance.z) > 0) && (dot(N, L) > 0)
        w = true;
        {
            brdfPdf = EvaluateDiffuse( L, N ) * Psurvive;
            {
                3 factor = diffuse * INVPI;
                {
                    weight = Mis2( directPdf, brdfPdf );
                    {
                        cosThetaOut = dot( N, L );
                        E * ((weight * cosThetaOut) / directPdf) * (radiance
        random walk - done properly, closely following Small's
        (survive)
    };
    {
        {
            3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
            survive;
            pdf;
            n = E * brdf * (dot( N, R ) / pdf);
            sion = true;
        }
    }
}
```

