

```
ics
& (depth < MAXDEPTH)
{
    if (inside & !inside)
    {
        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;
    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, R, Align );
    e.x + radiance.y + radiance.z );
    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;
```

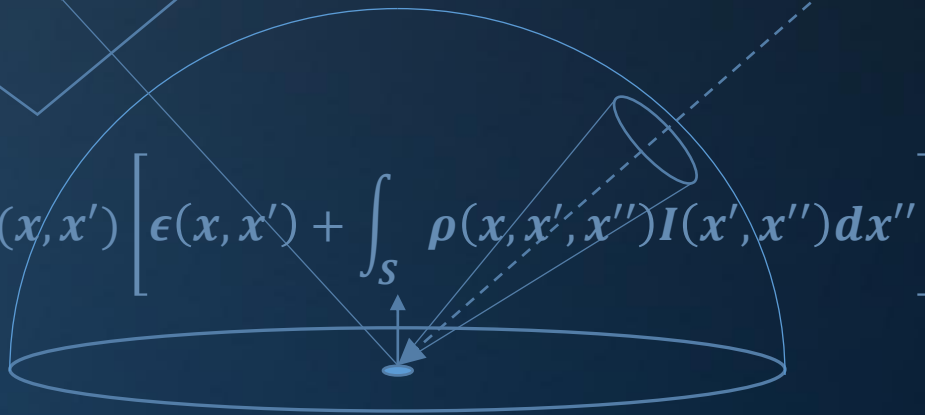


INFOMAGR – Advanced Graphics

Jacco Bikker - November 2022 - February 2023

Lecture 3 - “Light Transport”

Welcome!



Today's Agenda:

- Introduction
- The Rendering Equation
- Light Transport

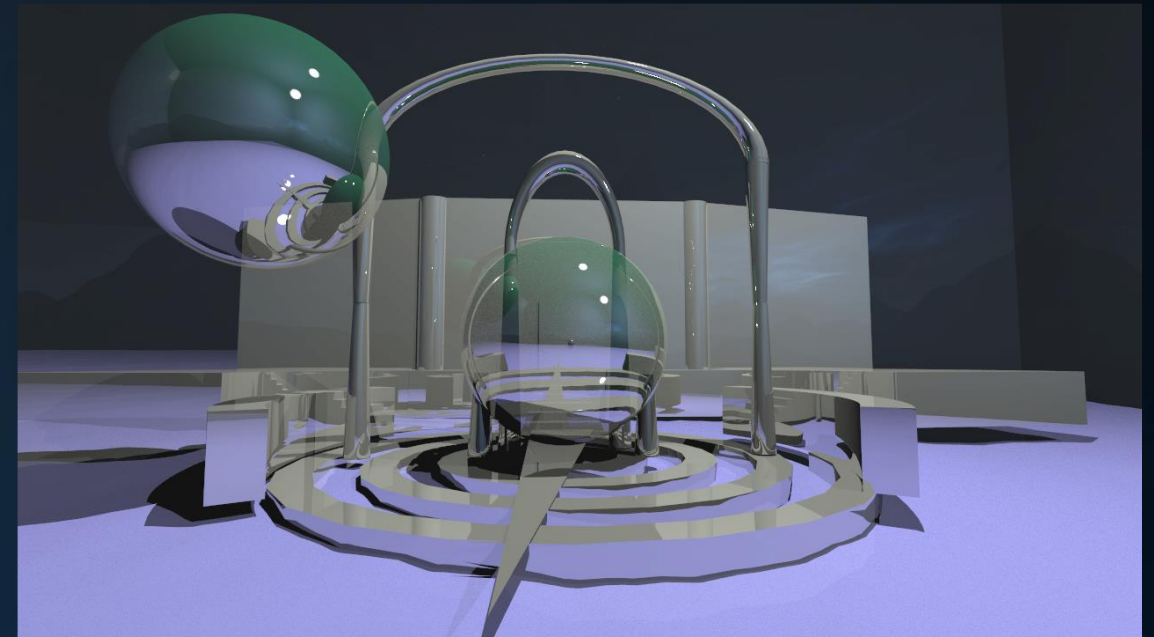


Introduction

Whitted

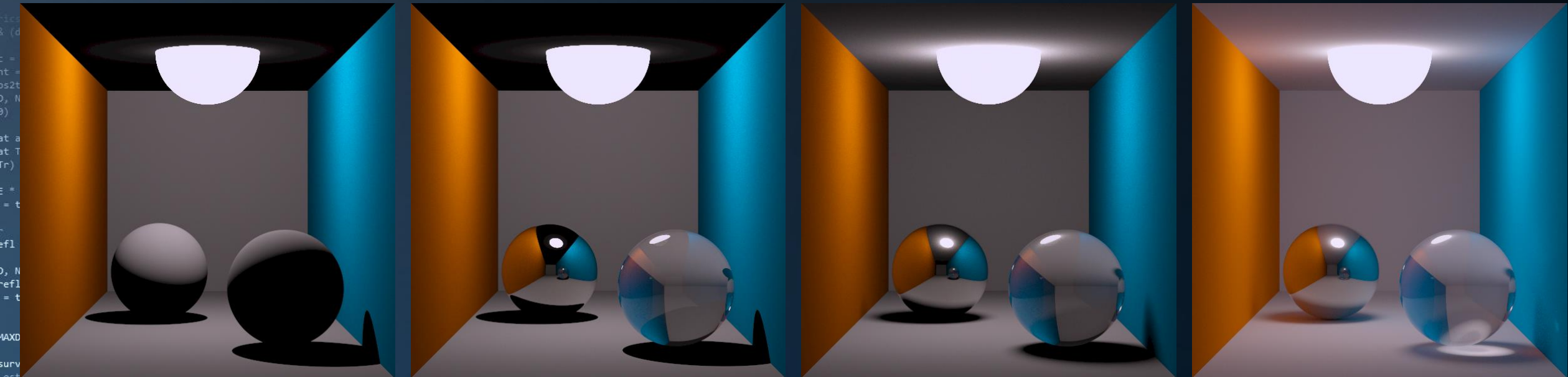


```
ics
& (depth < MAXDEPTH)
{
    if (inside & !
        nt = nt / nc;
        os2t = 1.0f -
        D, N );
    )
    at a = nt - nc;
    at Tr = 1 - (R
    Tr) R = (D * n
    E * diffuse;
    = true;
    efl + refr)) &
    D, N );
    refl * E * dif
    = true;
    MAXDEPTH)
    survive = Surv
    estimation -
    df;
    radiance = Sam
    e.x + radiance.y + radiance.z) > 0) & 0.5 *
    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 Russian
    vive)
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    sion = true;
```



Introduction

Whitted



```
if (rand() < 0.5) {  
    radiance = SampleLight( &rand, I, &L, &R, &pdf );  
    if (radiance.x + radiance.y + radiance.z > 0) && (rand() < 0.5) {  
        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 Monte Carlo  
        (survive)
```

```
if (rand() < 0.5) {  
    radiance = SampleLight( &rand, I, &L, &R, &pdf );  
    if (radiance.x + radiance.y + radiance.z > 0) && (rand() < 0.5) {  
        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 Monte Carlo  
        (survive)
```

```
if (rand() < 0.5) {  
    radiance = SampleLight( &rand, I, &L, &R, &pdf );  
    if (radiance.x + radiance.y + radiance.z > 0) && (rand() < 0.5) {  
        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 Monte Carlo  
        (survive)
```

```
if (rand() < 0.5) {  
    radiance = SampleLight( &rand, I, &L, &R, &pdf );  
    if (radiance.x + radiance.y + radiance.z > 0) && (rand() < 0.5) {  
        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 Monte Carlo  
        (survive)
```

```
if (rand() < 0.5) {  
    radiance = SampleLight( &rand, I, &L, &R, &pdf );  
    if (radiance.x + radiance.y + radiance.z > 0) && (rand() < 0.5) {  
        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 Monte Carlo  
        (survive)
```

```
if (rand() < 0.5) {  
    radiance = SampleLight( &rand, I, &L, &R, &pdf );  
    if (radiance.x + radiance.y + radiance.z > 0) && (rand() < 0.5) {  
        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 Monte Carlo  
        (survive)
```

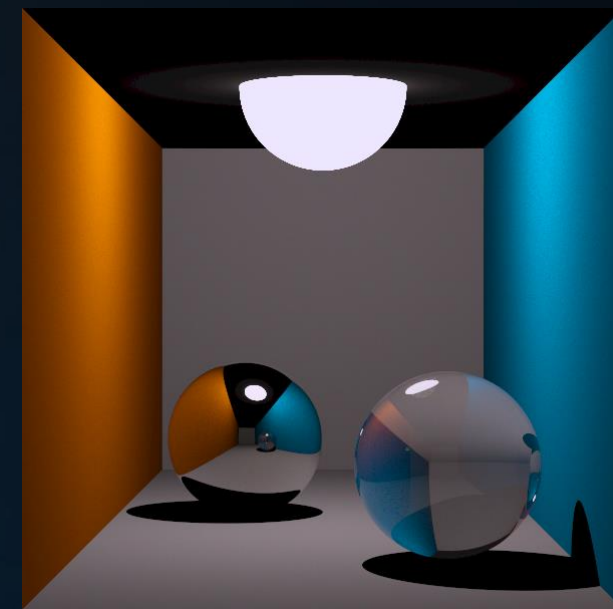


Introduction

Whitted

Missing:

- Area lights
- Glossy reflections
- Caustics
- Diffuse interreflections
- Diffraction
- Polarization
- Phosphorescence
- Temporal effects
- Motion blur
- Depth of field
- Anti-aliasing



Introduction

Anti-aliasing

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?



Introduction

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?



Introduction

Anti-aliasing – Sampling Patterns

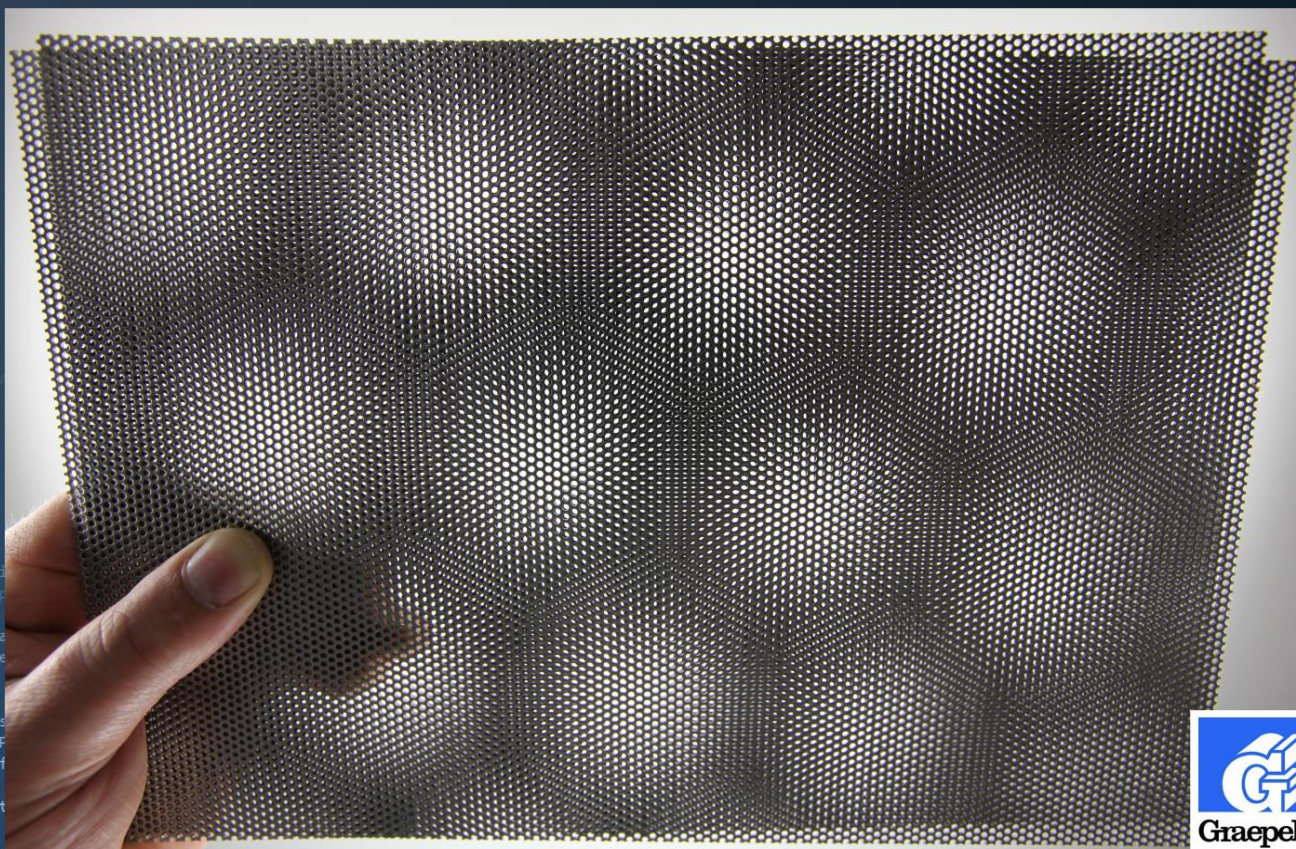
```

ics
& (depth < MAXDEPTH)
{
    if (inside ? 1 : 0)
    {
        nt = nt / nc; ddn = ddn * n;
        cos2t = 1.0f - nnt * nnt;
        D, N );
    }
    at a = nt - nc, b = nt * n;
    at Tr = 1 - (R0 + (1 - R0) *
    Tr) R = (D * nnt - N * (ddn
    E * diffuse;
    = true;
    refl + refr)) && (depth < M
    D, N );
    refl * E * diffuse;
    = true;
    MAXDEPTH)
    survive = SurvivalProbabil
    estimation - doing it prop
    df;
    radiance = SampleLight( &r
    e.x + radiance.y + radiance
    w = true;
    at brdfPdf = EvaluateDiffus
    at3 factor = diffuse * INV
    at weight = Mis2( directPdf
    at cosThetaOut = dot( N, L
    E * ((weight * cosThetaOut
    random walk - done properly
    vive)
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    sion = true;
  
```



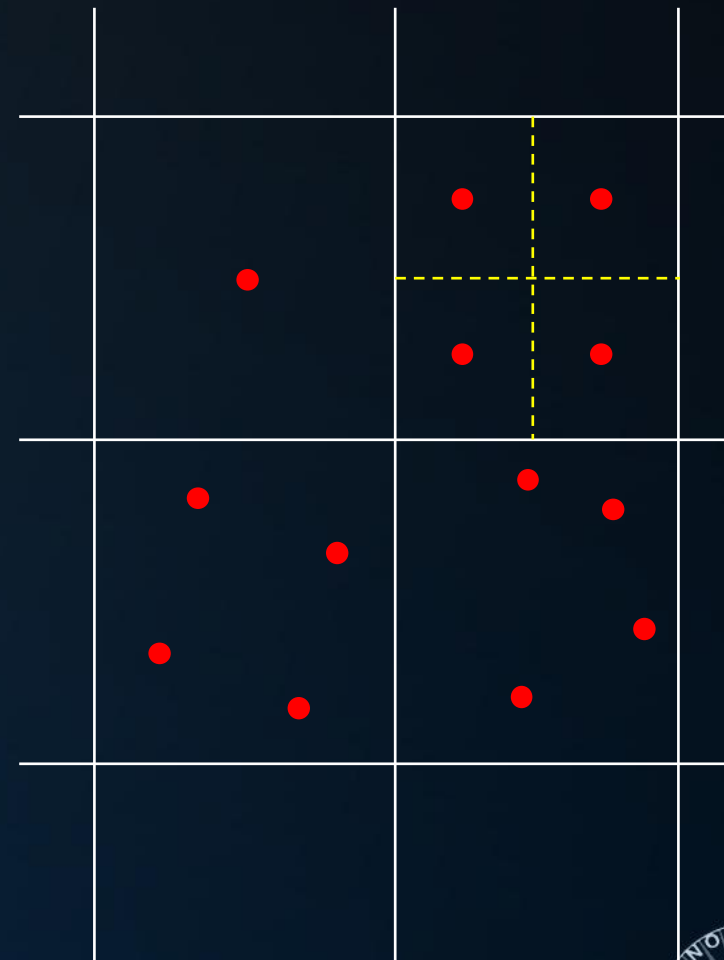
Introduction

Anti-aliasing – Sampling Patterns



Introduction

Anti-aliasing – Sampling Patterns



Introduction

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?



More info: <https://mynameismjp.wordpress.com/2012/10/24/msaa-overview>

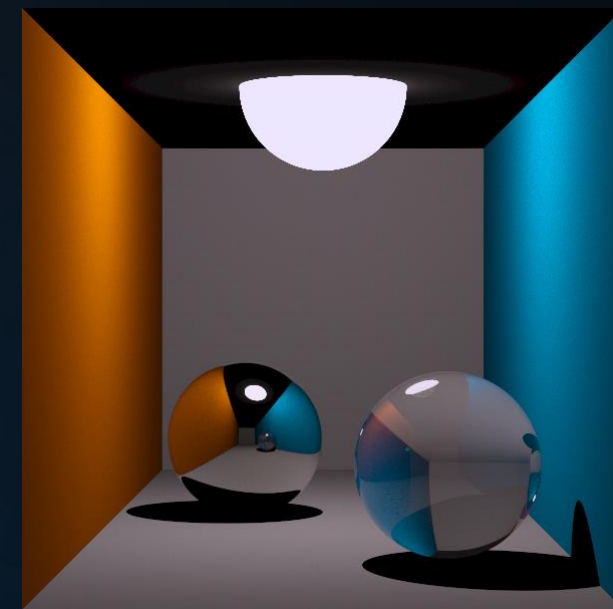


Introduction

Whitted

Missing:

- Area lights
- Glossy reflections
- Caustics
- Diffuse interreflections
- Diffraction
- Polarization
- Phosphorescence
- Temporal effects
- Motion blur
- Depth of field
- ✓ Anti-aliasing



Introduction

Distribution Ray Tracing*

```

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

```

```

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

```

```

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, &alignPdf );
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

```

```

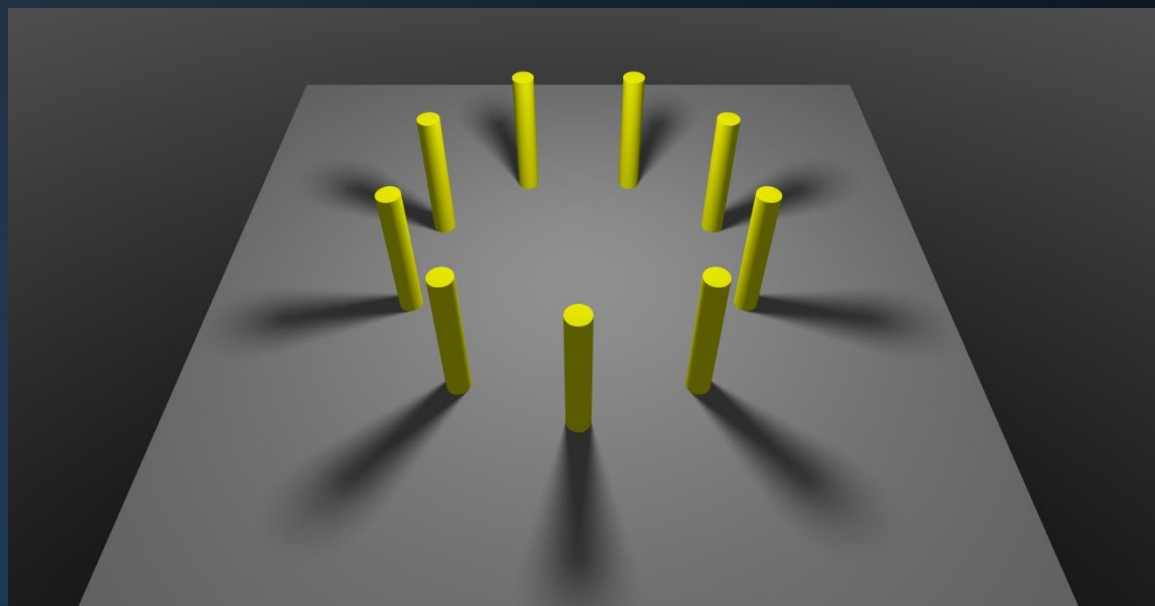
random walk - done properly, closely following
vive)

```

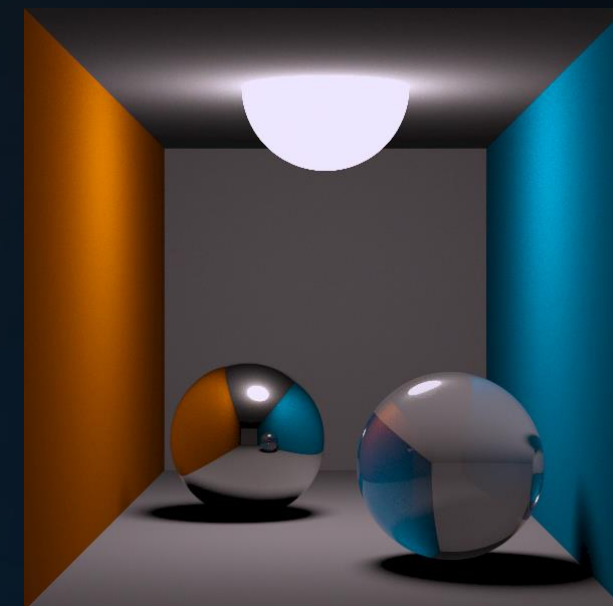
```

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



Introduction

Distribution Ray Tracing*

```

ics
& (depth < MAXDEPTH)

```

```

c = inside ? 1.0f : 0.0f;
nt = nt / nc; ddn = dot(N, L);
cos2t = 1.0f - nnt * ddn;
D, N );
0);

```

```

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;

```

```

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) && (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
vive)

```

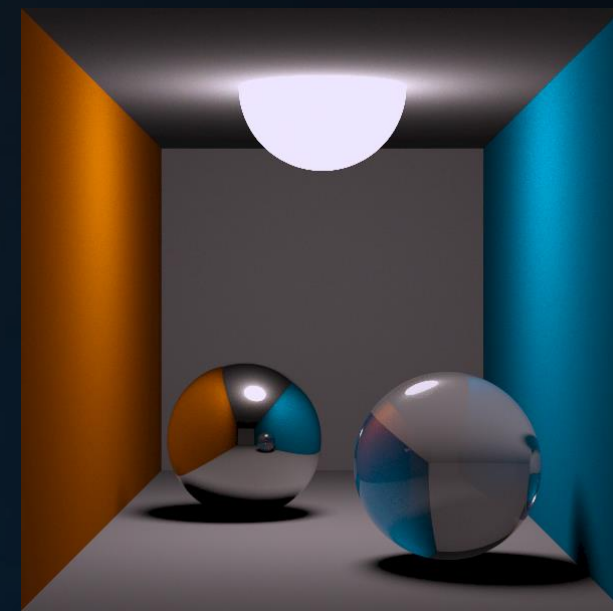
```

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

```



Glossy reflections



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



Introduction

Distribution Ray Tracing*

```

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

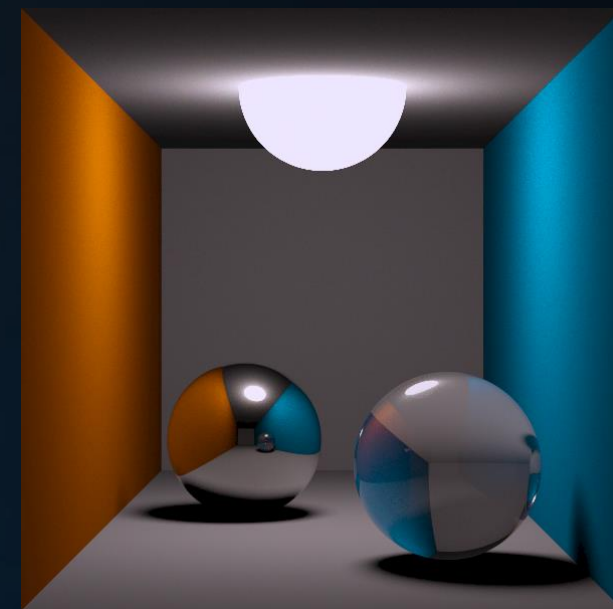
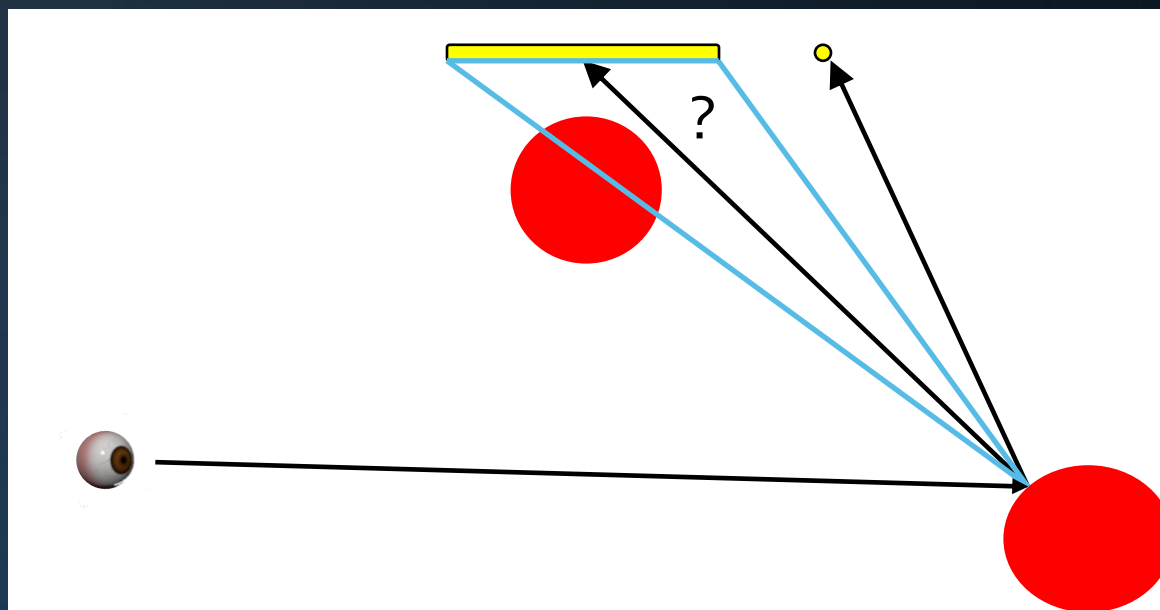
```

```

survive = SurvivalProbability( diffuse );
estimation - doing it properly, closely following
df;
radiance = SampleLight( &rand, I, &L, &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
    random walk - done properly, closely following
    vive)
}

```

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



Introduction

Distribution Ray Tracing*

```

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

```

```

at a = nt - nc, b = nt + nc;
at Tr = 1 - (R0 + (1 - R0) * rnt);
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, &alignPdf );
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

```

```

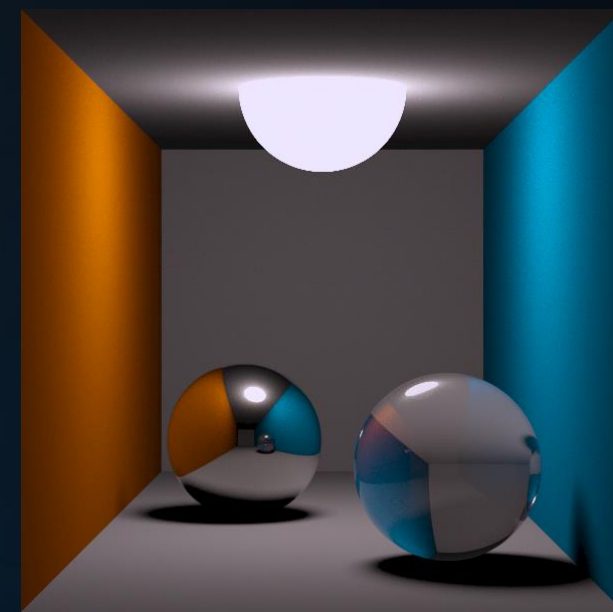
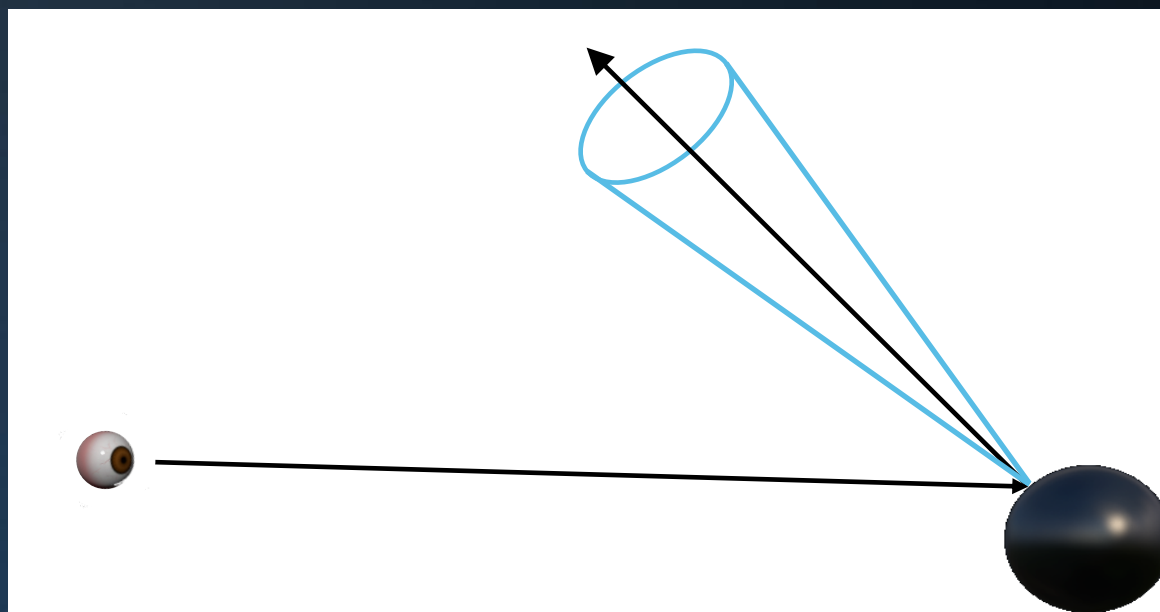
random walk - done properly, closely following
vive)

```

```

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

```



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



Introduction

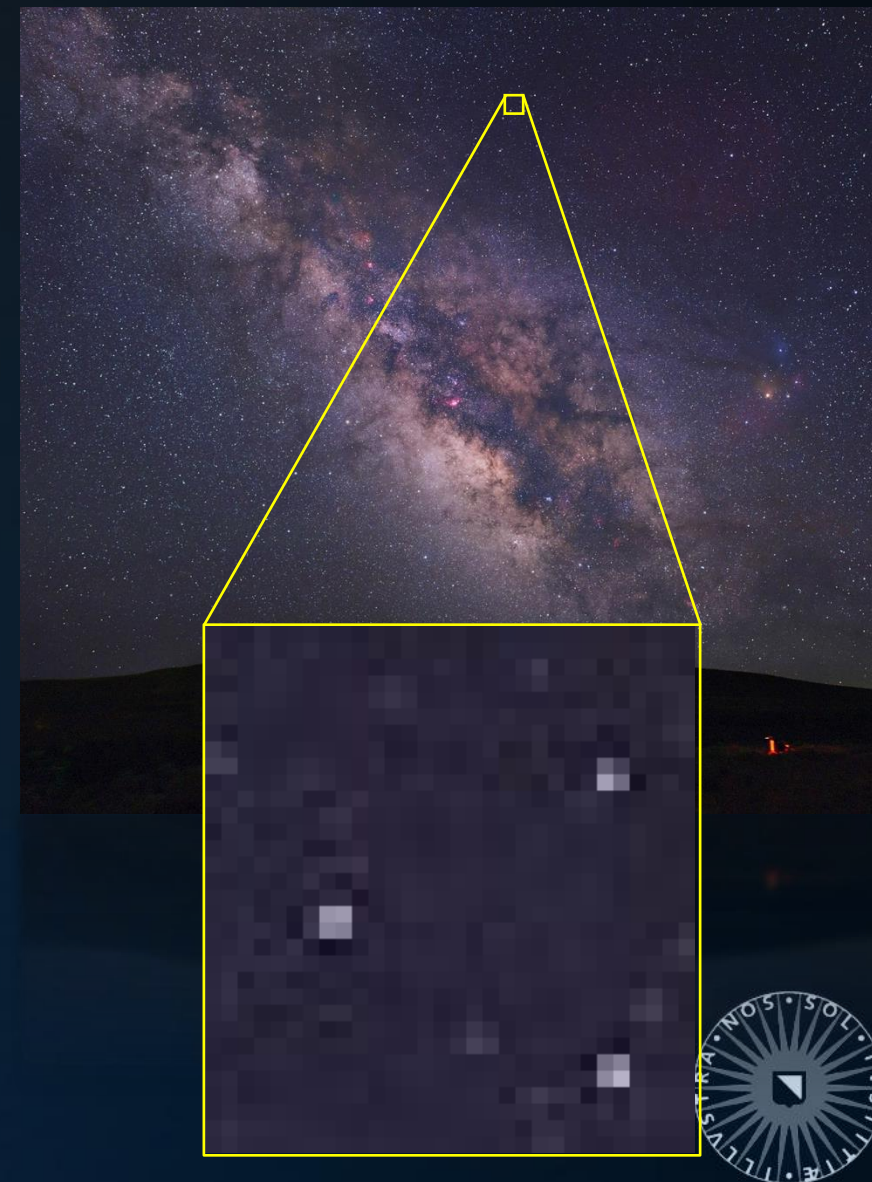
Distribution Ray Tracing

Whitted-style ray tracing is a *point sampling* algorithm:

- We may miss small features
- *We cannot sample areas*

Area sampling:

- Anti-aliasing: one pixel
- Soft shadows: one area light source
- Glossy reflection: directions in a cone
- Diffuse reflection: directions on the hemisphere



Introduction

Area Lights

Visibility of an area light source:

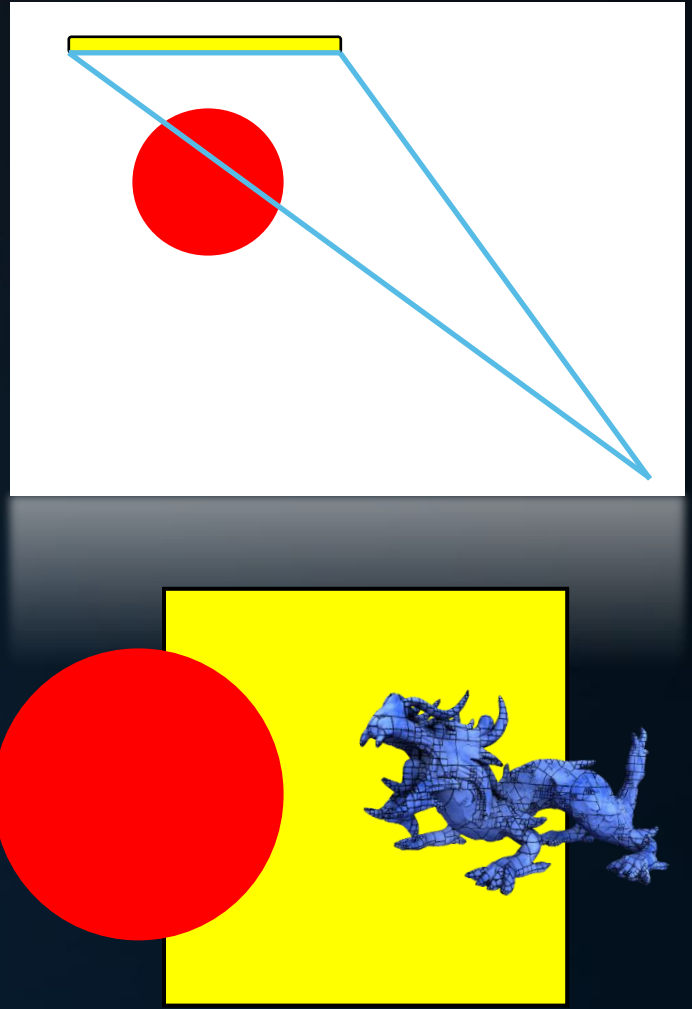
$$V_A = \int_A V(x, \omega_i) d\omega_i$$

Analytical solution case 1:

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

Analytical solution case 2:

$$V_A = ?$$



Introduction

Approximating Integrals

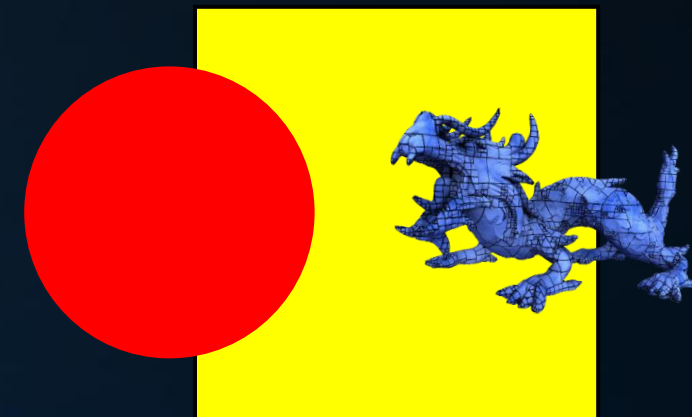
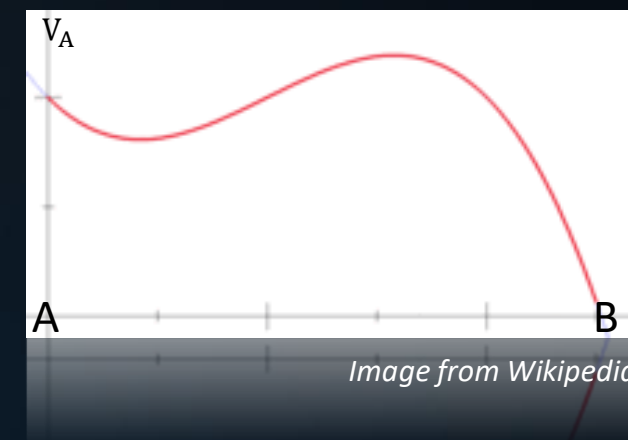
An integral can be approximated as a Riemann sum:

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

Note that the intervals do not need to be uniform, as long as we sample the full interval. If the intervals are uniform, then

$$\sum_{i=1}^N f(t_i) \Delta_i = \Delta_i \sum_{i=1}^N f(t_i) = \frac{B-A}{N} \sum_{i=1}^N f(t_i).$$

Regardless of uniformity, restrictions apply to N when sampling multi-dimensional functions (ideally, $N = M^d$, $M \in \mathbb{N}$). Also note that aliasing may occur if the intervals are uniform.



Introduction

Monte Carlo Integration

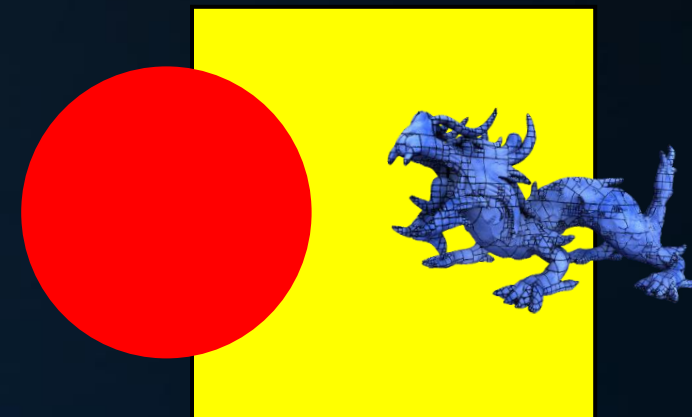
Alternatively, we can approximate an integral by taking random samples:

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

Here, $X_1..X_N \in [A, B]$.

As N approaches infinity, V_A approaches the *expected value* of $f(X)$.

Unlike in Riemann sums, we can use arbitrary N for Monte Carlo integration, regardless of dimension.



Introduction

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:

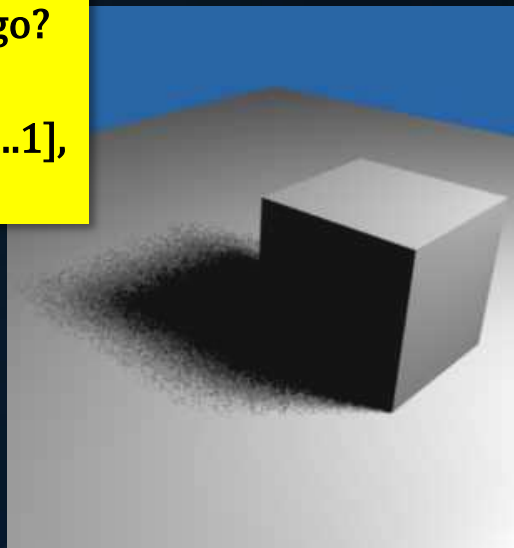
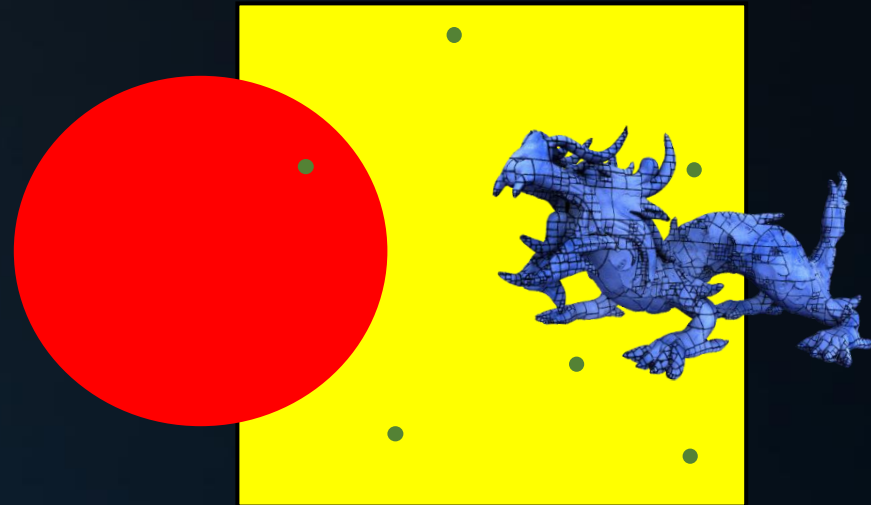
$$V \approx \frac{1}{6}(1 + 1 + 1 + 0 + 1 + 1) = \frac{5}{6}$$

Properly formulated using a MC integrator:

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

Q: Where did the $\frac{B-A}{N}$ go?
A: the domain of the visibility function is $[0..1]$, so $B - A = 1$.

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



```
ics
& (depth < MAXDEPTH)
{
    if (inside > 1)
    {
        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 followi
    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) * (radi
    random walk - done properly, closely followi
    vive)
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    sion = true;
```



Introduction

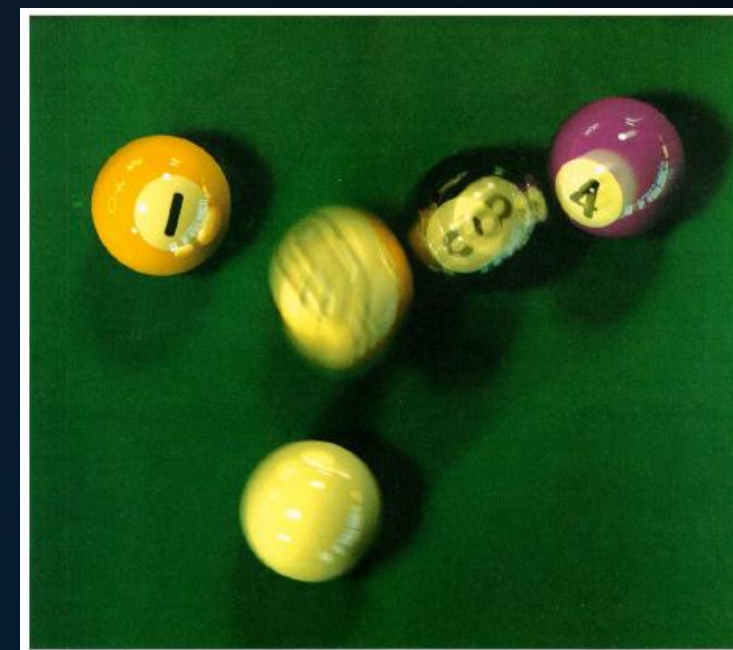
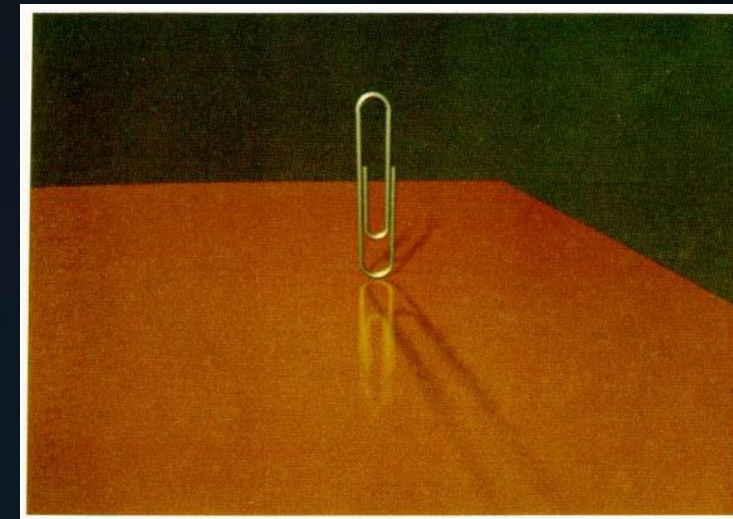
Distribution Ray Tracing

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
- Wavelengths
- Many lights
- ...



Today's Agenda:

- Introduction
- The Rendering Equation
- Light Transport



Rendering Equation

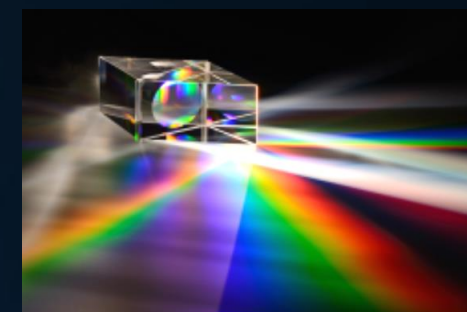
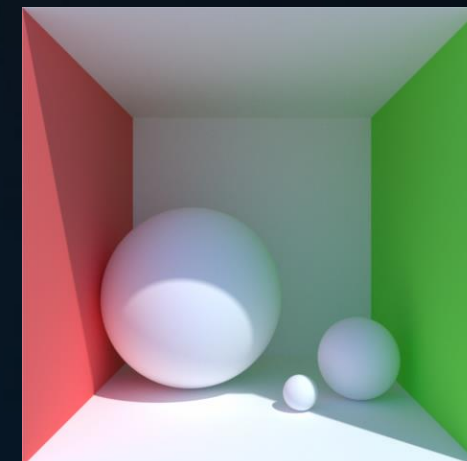
Whitted, Cook & Beyond

Missing in Whitted:

- Area lights
- Glossy reflections
- Caustics
- Diffuse interreflections
- Diffraction
- Polarization
- Phosphorescence
- Temporal effects
- Motion blur
- Depth of field
- Anti-aliasing

Cook:

- ✓ Area lights
- ✓ Glossy reflections
- ✗ Caustics
- ✗ Diffuse interreflections
- ✗ Diffraction
- ✗ Polarization
- ✗ Phosphorescence
- ✗ Temporal effects
- ✓ Motion blur
- ✓ Depth of field
- ✓ Anti-aliasing



Rendering Equation

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



Rendering Equation



Rendering Equation

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:

$$L(s \leftarrow x) = \int_A f_r(s \leftarrow x \leftarrow x') L(x \leftarrow x') G(x \leftrightarrow x') dA(x')$$

Those are the options.

Adding direct and indirect illumination together:

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



Rendering Equation

God's Algorithm - Mathematically

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

Emission

Hemisphere

Reflection

Indirect

Geometry factor



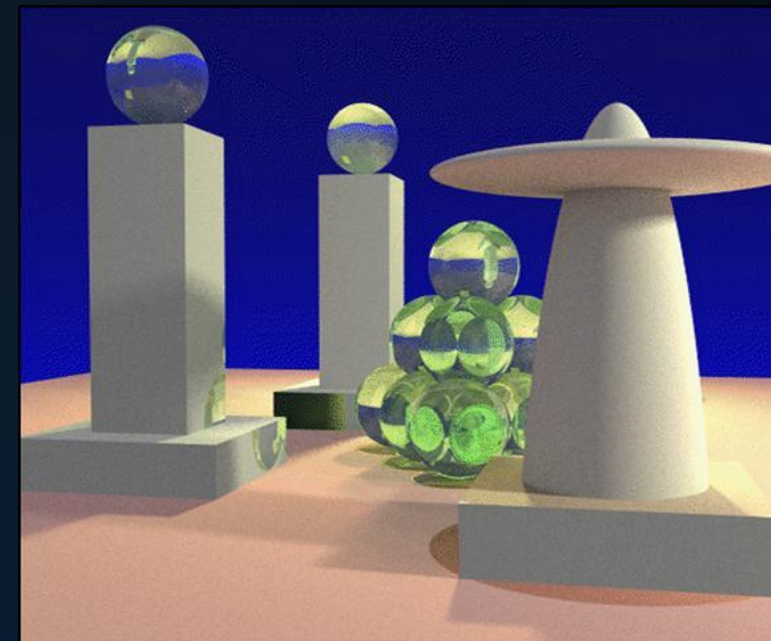
Rendering Equation

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

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.



```

ics
& (depth < MAXDEPTH)
{
    if (inside ? 1 : 0)
    {
        nt = nt / nc; ddn = dot(N, L);
        cos2t = 1.0f - nnt * nnt;
        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 followi
    if;
    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
    }
    random walk - done properly, closely followi
    vive)
    {
        at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
        survive;
        pdf;
        n = E * brdf * (dot( N, R ) / pdf);
        sion = true;
    }
}

```

*: The Rendering Equation, Kajiya, 1986



Today's Agenda:

- Introduction
- The Rendering Equation
- Light Transport



Today's Agenda:

- Introduction
- The Rendering Equation
- Light Transport



INFOMAGR – Advanced Graphics

Jacco Bikker - November 2022 - February 2023

END of “Light Transport”

next lecture: “Path Tracing”

