

```
ics
& (depth < MAXDEPTH)
{
    if (inside & !isInside)
    {
        nt = nt / nc; ddn = ddn * nc;
        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 following Small's
        if;
        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 Small's
        vive)

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

# Ray Tracing for Games

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

# Welcome!

# 10



```

ics
& (depth < MAXDEPTH))
{
    if (inside ? 1 : 1.01 - .0001 * depth)
    {
        nt = nt / nc, ddn = ddn * nc;
        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 Small's
        if;
        radiance = SampleLight( &rand, I, &L, &lightPdf );
        e.x + radiance.y + radiance.z) > 0) && (octa.n < 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
        (survive)
        ;
        at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
        survive;
        pdf;
        n = E * brdf * (dot( N, R ) / pdf);
        sion = true;
    }
}

```

# Agenda:

- Importance Sampling
- Russian Roulette



## Importance Sampling for Monte Carlo

Monte Carlo integration:

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

Example 1: rolling two dice  $D_1$  and  $D_2$ , the outcome is  $6D_1 + D_2$ . What is the expected value of this experiment?

(average die value is 3.5, so the answer is  $3.5 * 6 + 3.5 = 24.5$ )

Using Monte Carlo:

$$V = \frac{1}{N} \sum_{i=1}^N f(D_1) + g(D_2) \text{ where: } D_1, D_2 \in \{1, 2, 3, 4, 5, 6\}, f(x) = 6x, g(x) = x$$



## Importance Sampling for Monte Carlo

Changing the experiment slightly: each sample is one roll of one die.

Using Monte Carlo:

$$V = \frac{1}{N} \sum_{i=1}^N \frac{f(T, D)}{0.5} \text{ where: } D \in \{1, 2, 3, 4, 5, 6\}, T \in \{0, 1\}, f(t, d) = (5t + 1) d$$

*0.5: Probability of using die T.*

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

```
for( int i = 0; i < 1000; i++ )
{
    int D1 = IRand( 6 ) + 1;
    int D2 = IRand( 6 ) + 1;
    float f = (float)(6 * D1 + D2);
    total += f;
    rolls++;
}
```

```
for( int i = 0; i < 2000; i++ )
{
    int D = IRand( 6 ) + 1;
    int T = IRand( 2 );
    float f = (float)((5 * T + 1) * D) / 0.5f;
    total += f;
    rolls++;
}
```



## Importance Sampling for Monte Carlo

What happens when we don't pick each die with the same probability?

```
float D1_prob = 0.8f;
for( int i = 0; i < 1000; i++ )
{
    int D = IRand( 6 ) + 1;
    float r = Rand(); // uniform 0..1
    int T = (r < D1_prob) ? 0 : 1;
    float p = (T == 0) ? D1_prob : (1 - D1_prob);
    float f = (float)((5 * T + 1) * D) / p;
    total += f;
    rolls++;
}
```

- we get the correct answer;
- we get lower variance.



Sampling the large light with a greater probability yields a better estimate.



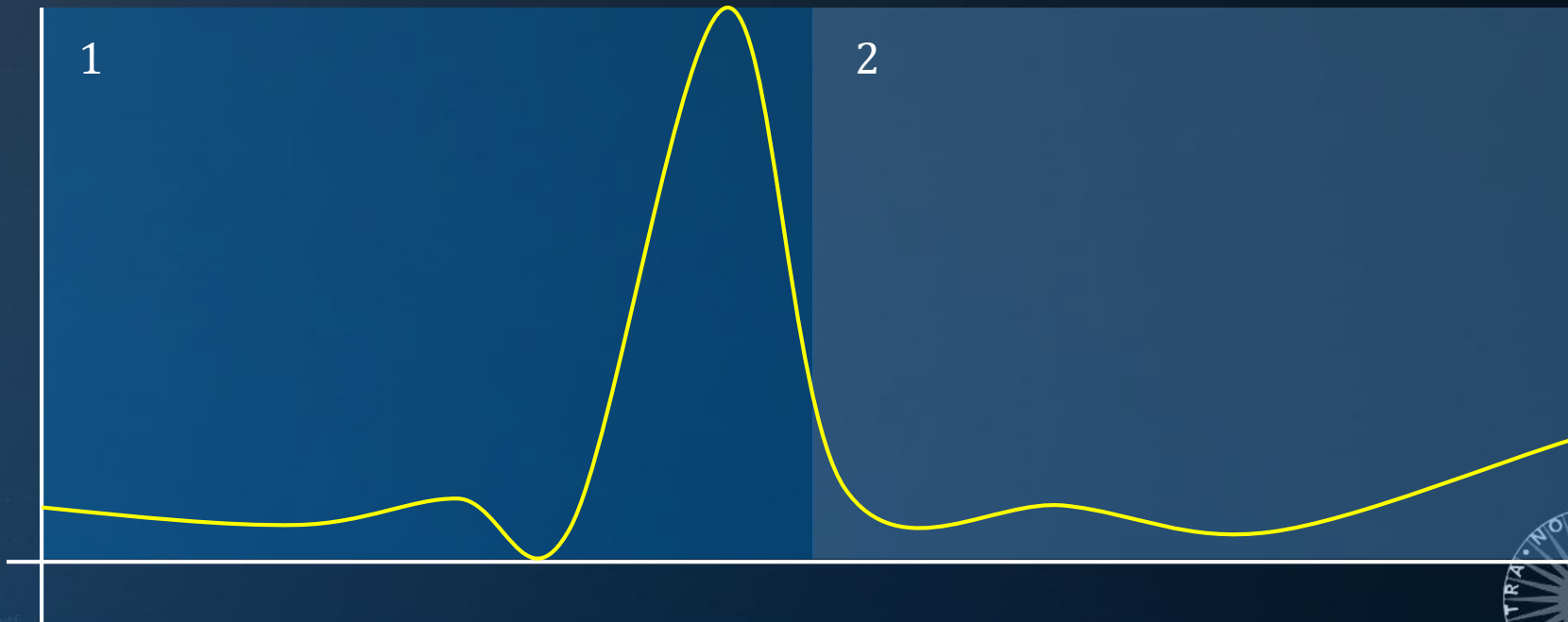


## Importance Sampling for Monte Carlo

### Example 3: sampling an integral.

Considering the previous experiments, which stratum should be sample more often?

```
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 );
    estimation - doing it properly, closely following
    if;
    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
    ve)
    ;
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    ion = true;
}
```



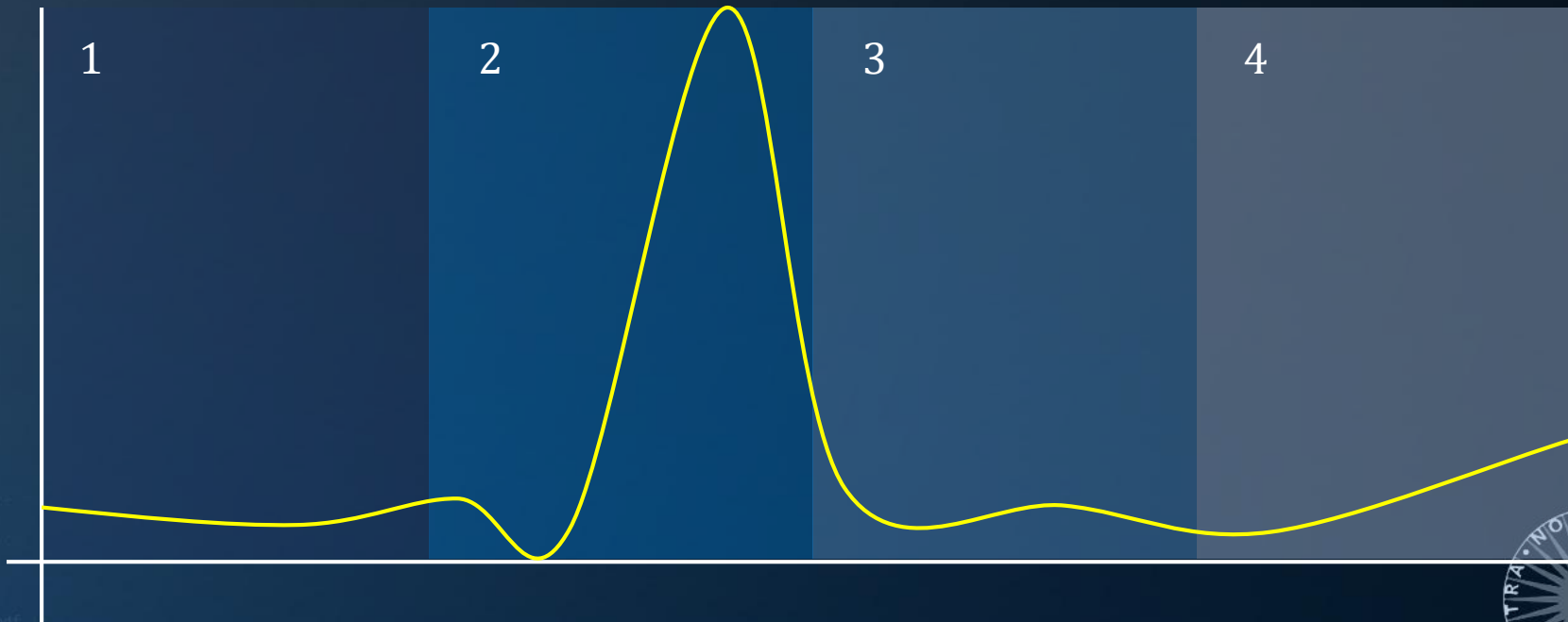
## Importance Sampling for Monte Carlo

Example 3: sampling an integral.

Considering the previous experiments, which stratum should be sample more often?

```
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 - (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
    f;
    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 Small
    ve)
    ;
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    ion = true;

```

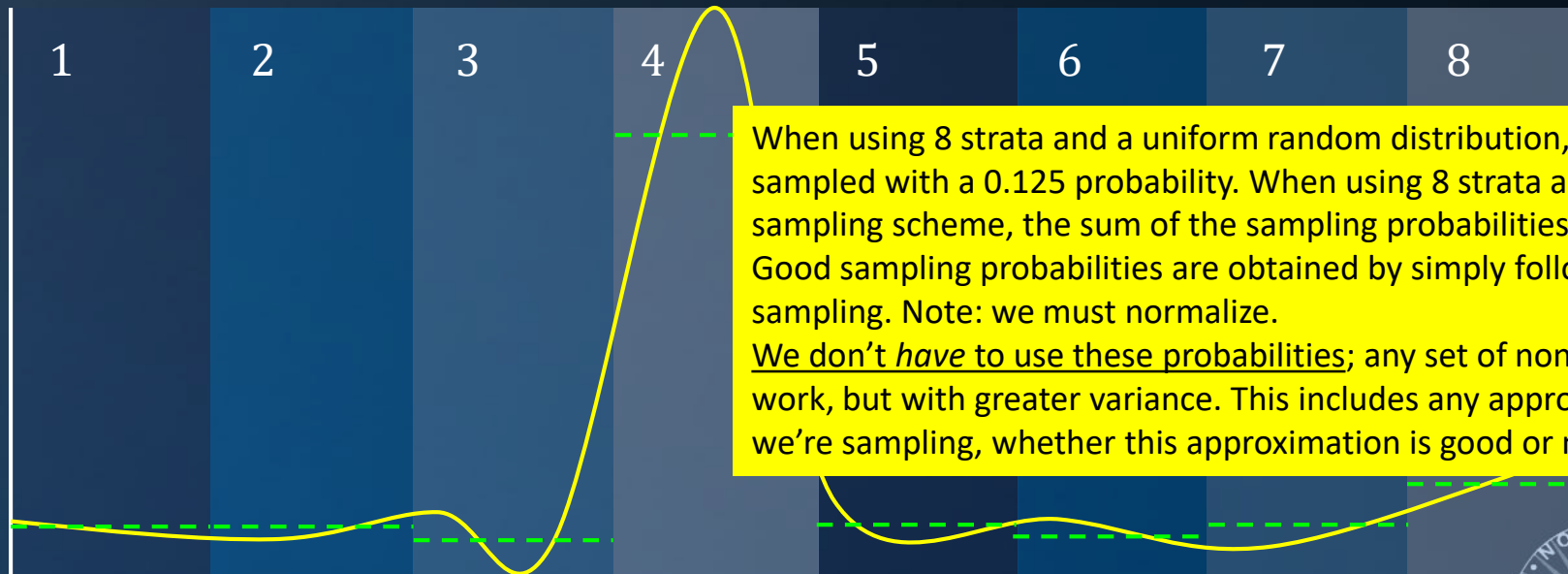




## Importance Sampling for Monte Carlo

### Example 3: sampling an integral.

Considering the previous experiments, which stratum should be sample more often?



When using 8 strata and a uniform random distribution, each stratum will be sampled with a 0.125 probability. When using 8 strata and a non-uniform sampling scheme, the sum of the sampling probabilities must be 1. Good sampling probabilities are obtained by simply following the function we're sampling. Note: we must normalize.

We don't *have* to use these probabilities; any set of non-zero probabilities will work, but with greater variance. This includes any approximation of the function we're sampling, whether this approximation is good or not.

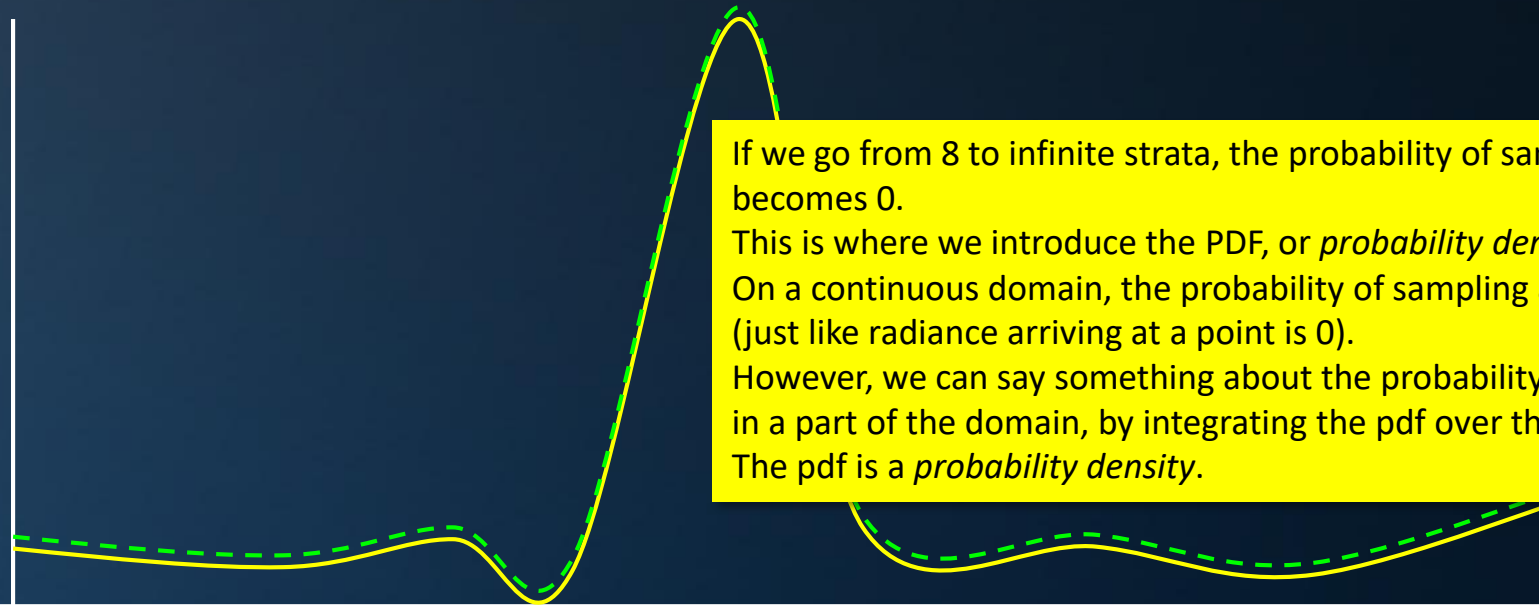


## Importance Sampling for Monte Carlo

### Example 3: sampling an integral.

Considering the previous experiments, which stratum should be sample more often?

```
ics
& (depth < MAXDEPTH)
{
    if (inside) {
        nt = inside ? 1.0 : 0.0;
        nt = nt / nc; ddn = dot(N, D);
        cos2t = 1.0f - nnt * nnt;
        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
    if;
    radiance = SampleLight( &rand, I, &L, &align,
    e.x + radiance.y + radiance.z > 0) && (depth <
    w = true;
    at brdfPdf = EvaluateDiffuse( L, N ) * Psurvive;
    at3 factor = diffuse * INVPI;
    at weight = Mis2( directPdf, brdfPdf );
    at cosThetaOut = dot( N, L );
    E * ((weight * cosThetaOut) / directPdf) * (radiance
    random walk - done properly, closely following
    ve)
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    sion = true;
```



If we go from 8 to infinite strata, the probability of sampling a stratum becomes 0.

This is where we introduce the PDF, or *probability density function*.

On a continuous domain, the probability of sampling a specific  $X$  is 0 (just like radiance arriving at a point is 0).

However, we can say something about the probability of choosing  $X$  in a part of the domain, by integrating the pdf over the subdomain.

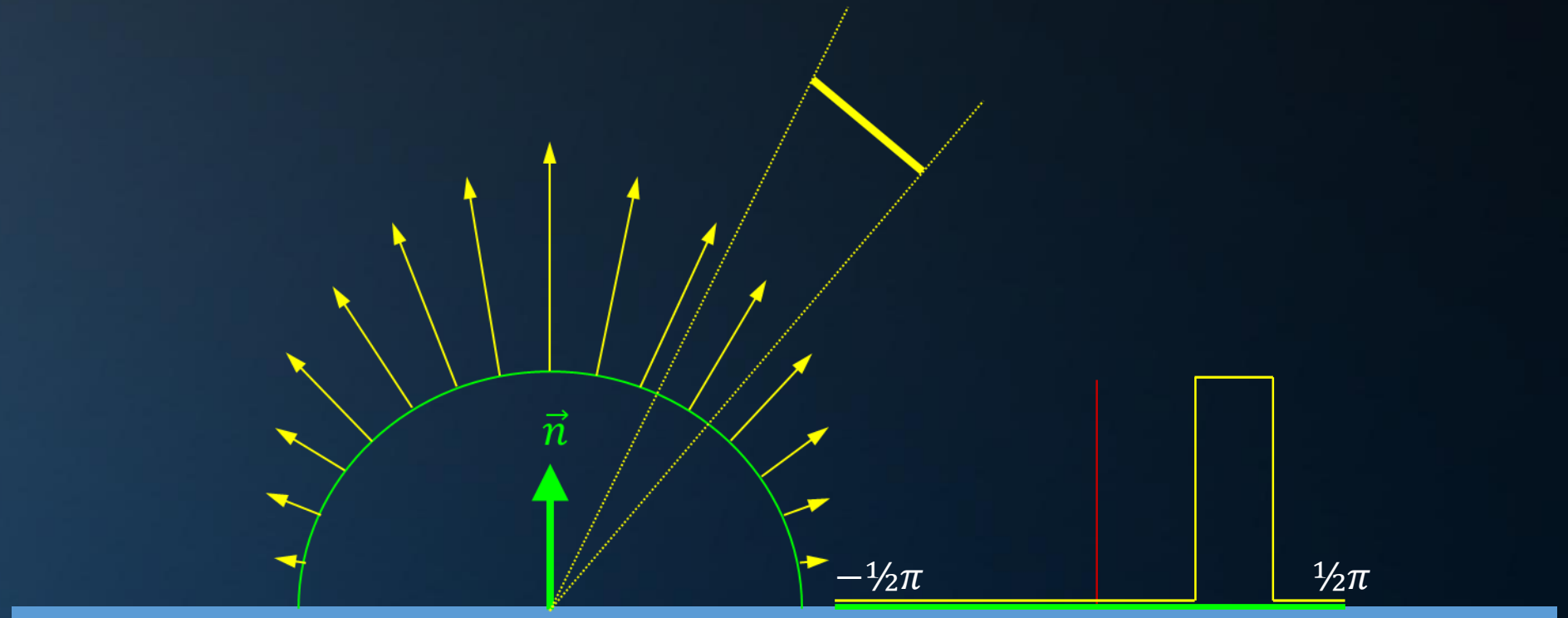
The pdf is a *probability density*.



## Importance Sampling for Monte Carlo

### Example 4: sampling the hemisphere.

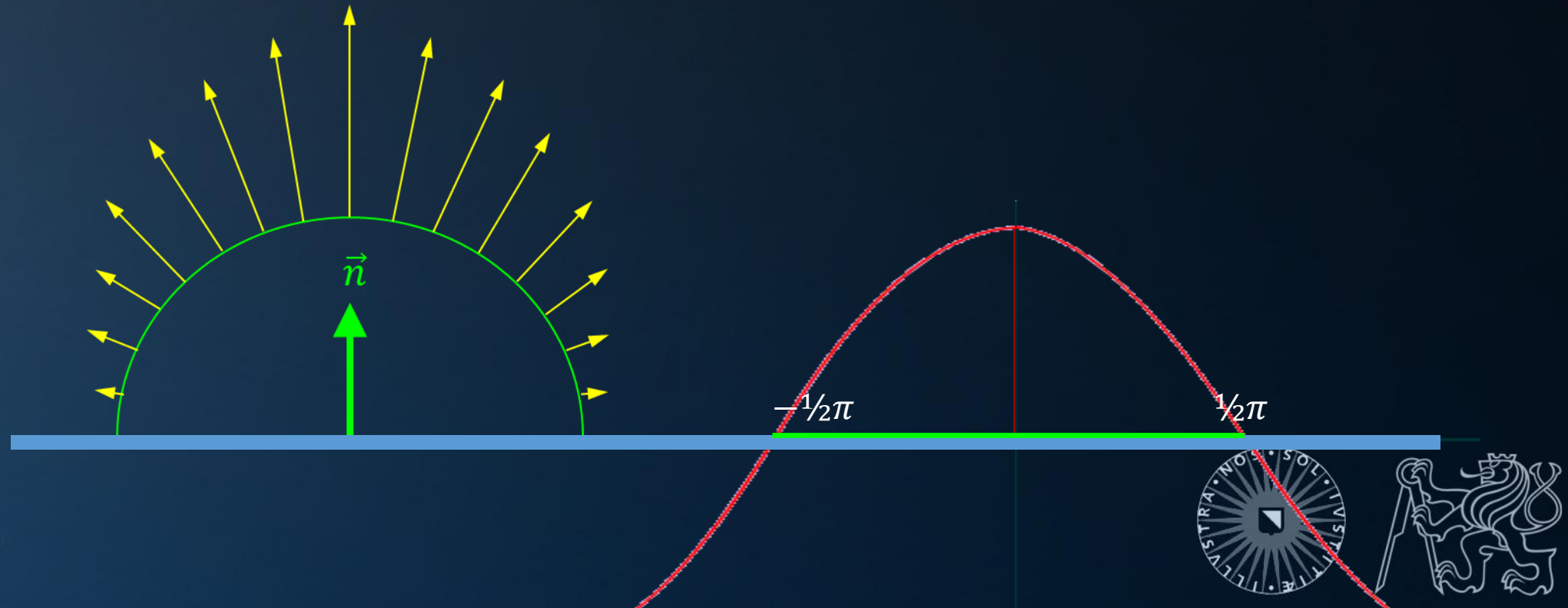
```
ics
& (depth < MAXDEPTH)
{
    if (inside) {
        nt = inside ? 1.0 : 0.0;
        nc = nt / nc; ddn = dot(N, N);
        cos2t = 1.0f - nnt * nnt;
        D, N );
    }
    else {
        at a = nt - nc, b = nt + nc;
        at Tr = 1 - (R0 + (1 - R0) * sqrt(r));
        Tr) R = (D * nnt - N * (ddn * cos2t));
    }
    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, &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
    (survive)
    ;
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    sion = true;
}
```



## Importance Sampling for Monte Carlo

### Example 4: sampling the hemisphere.

```
ics
& (depth < MAXDEPTH)
{
    if (inside) {
        nt = inside ? 1.0 : 0.0;
        nt = nt / nc; ddn = dot(N, D);
        cos2t = 1.0f - nnt * ddn;
        D, N );
    }
    else {
        at a = nt - nc, b = nt + nc;
        at Tr = 1 - (R0 + (1 - R0) * sqrt(a * b));
        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, &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 Small
    vive)
    ;
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    sion = true;
}
```



# Importance Sampling for Monte Carlo

## Monte Carlo without importance sampling:

$$E(f(X)) \approx \frac{1}{N} \sum_{i=1}^N f(X)$$

## With importance sampling:

$$E(f(X)) \approx \frac{1}{N} \sum_{i=1}^N \frac{f(X_i)}{p(X_i)}$$

Here,  $p(x)$  is the *probability density function* (PDF).



## Cosine-weighted Random Direction

Without deriving this in detail:

A cosine-weighted random distribution is obtained by generating points on the unit disc, and projecting the disc on the unit hemisphere. In code:

```
float3 CosineWeightedDiffuseReflection()
{
    float r0 = Rand(), r1 = Rand();
    float r = sqrt( r0 );
    float theta = 2 * PI * r1;
    float x = r * cosf( theta );
    float y = r * sinf( theta );
    return float3( x, y, sqrt( 1 - r0 ) );
}
```

Note: you still have to transform this to tangent space.





# Ray Tracing for Games

```
Color Sample( Ray ray )
{
    // trace ray
    I, N, material = Trace( ray );
    // terminate if ray left the scene
    if (ray.NOHIT) return BLACK;
    // terminate if we hit a light source
    if (material.isLight) return emittance;
    // continue in random direction
    R = DiffuseReflection( N );
    Ray r( I, R );
    // update throughput
    BRDF = material.albedo / PI;
    PDF = 1 / (2 * PI);
    Ei = Sample( r ) * (N·R) / PDF;
    return BRDF * Ei;
}
```

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



```

ics
& (depth < MAXDEPTH))
{
    if (inside ? 1 : 1.01 - .0001 * depth)
    {
        nt = nt / nc, ddn = ddn * nc;
        r2t = 1.0f - nnt * nnt;
        r2b = 1.0f - nnt * nnt;
        D, N );
    }
    {
        at a = nt - nc, b = nt + nc;
        at Tr = 1 - (R0 + (1 - R0) * r2t);
        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 Smallwood
        if;
        radiance = SampleLight( &rand, I, &L, &lightPdf );
        e.x + radiance.y + radiance.z) > 0) && (octa.n < 1))
        {
            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
        survive)
        {
            ;
            at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
            survive;
            pdf;
            n = E * brdf * (dot( N, R ) / pdf);
            sion = true;
        }
    }
}

```

# Agenda:

- Importance Sampling
- Russian Roulette



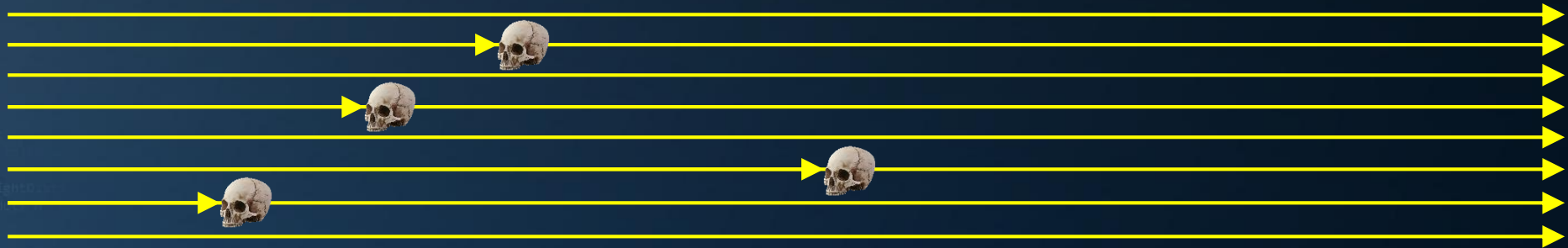
## Russian Roulette

Core idea:

The longer a path becomes, the less energy it transports.

Killing half of 16 rays is easy; what do we do with a single path?

→ Kill it with a probability of 50%.



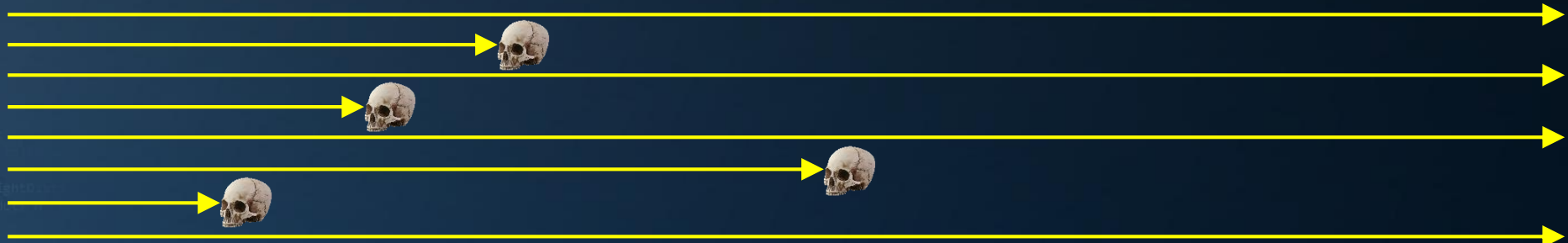
8 rays, returning 16 Watts of radiance each, 128 Watts in total.  
= 4 rays, returning 32 Watts of radiance each, 128 Watts in total.



## Russian Roulette

Russian roulette is applied to the random walk.

Most basic implementation: just before you start calculating the next random direction, you decide if the path lives or dies.



8 rays, returning 16 Watts of radiance each, 128 Watts in total.  
= 4 rays, returning 32 Watts of radiance each, 128 Watts in total.



## Better Russian Roulette

The termination probability of 50% is arbitrary.

*Any probability is statistically correct.*

However: for 50% survival rate, survivors scale up by 2  $\left(= \frac{1}{50\%}\right)$ .

→ In general, for a survival probability  $\rho$ , survivors scale up by  $\frac{1}{\rho}$ .

We can choose the survival probability *per path*. It is typically linked to albedo: the color of the last vertex. A good survival probability is:

$$\rho_{\text{survive}} = \text{clamp}\left(\frac{\text{red} + \text{green} + \text{blue}}{3}, 0.1, 0.9\right)$$

Note that  $\rho_{\text{survive}} > 0$  to prevent bias. Also note that  $\rho = 1$  is never a good idea.

Better:

$$\rho_{\text{survive}} = \text{clamp}(\max(\text{red}, \text{green}, \text{blue}), 0, 1)$$

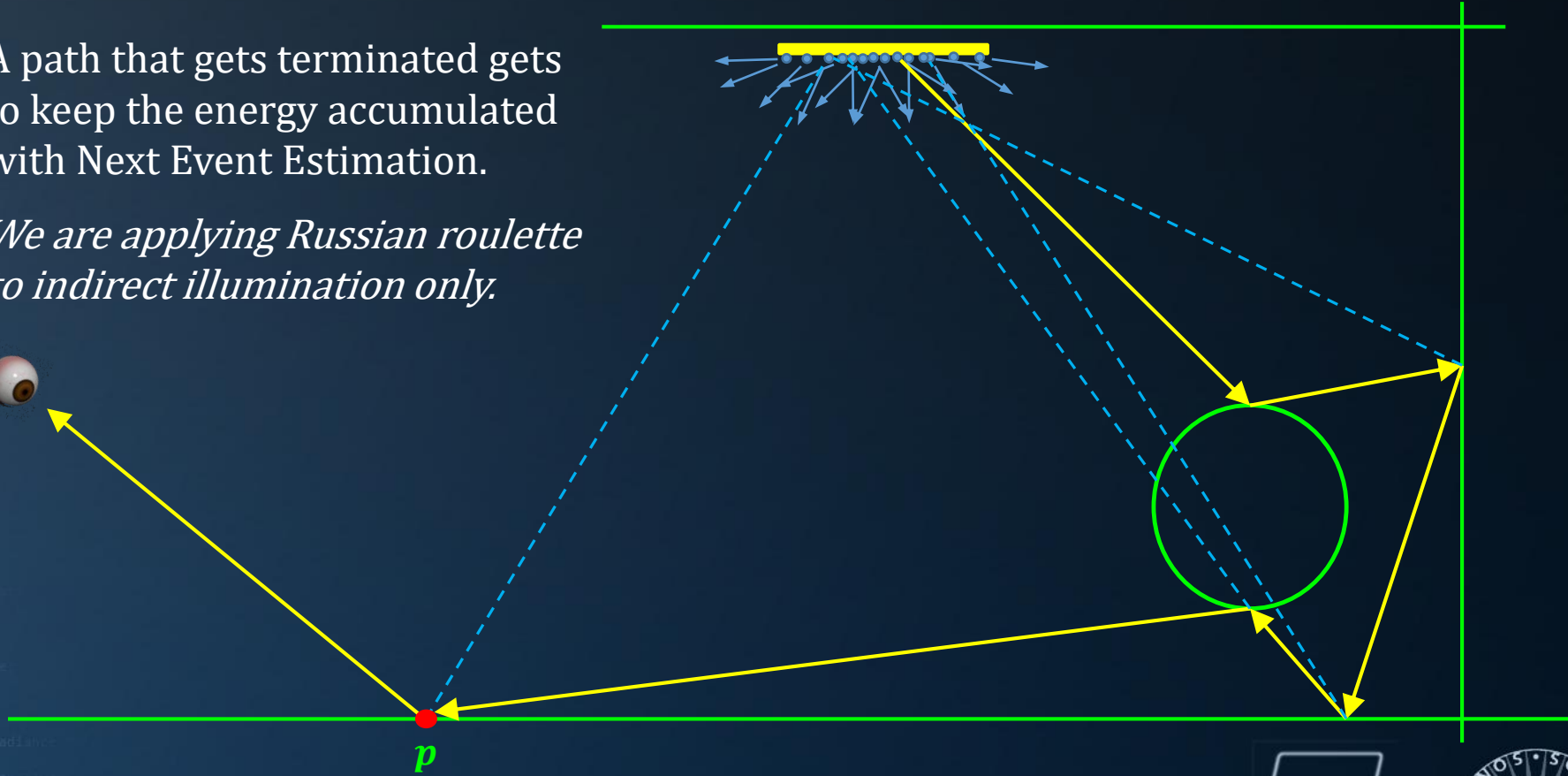




## RR and Next Event Estimation

A path that gets terminated gets to keep the energy accumulated with Next Event Estimation.

*We are applying Russian roulette to indirect illumination only.*





```

R = (depth < MAXDEPTH) {
    if (inside ? 1 : 0.125) {
        nc = nt / 2; ddn = ddn * 0.5;
        cos2t = 1.0f - nnt * 0.5;
        D, N );
    }
    if (a = nt - nc, b = nt * nc, R) {
        Tr = 1 - (R0 + (1 - R0) * ddn);
        R = (D * nnt - N * (ddn * 0.5));
    }
    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 the
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 = Mix2( 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;

```

