

```
ics
& (depth <
t = inside
nt = nt / nc
os2t = 1.0f
D, N );
0);
at a = nt - nc, b = nt + nc
at Tr = 1 - (R0 + (1 - R0)
Tr) R = (D * nnt - N * (dd
E * diffuse;
= true;
efl + refr)) && (depth < MAXDEPTH)
D, N );
refl * E * diffuse;
= true;
MAXDEPTH)
survive = SurvivalProbability( diffuse );
estimation - doing it properly, close
df;
radiance = SampleLight( &rand, I, &align
e.x + radiance.y + radiance.z );
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) * (radiance
random walk - done properly, closely following Small
ive)
;
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;
```

TL;DR version

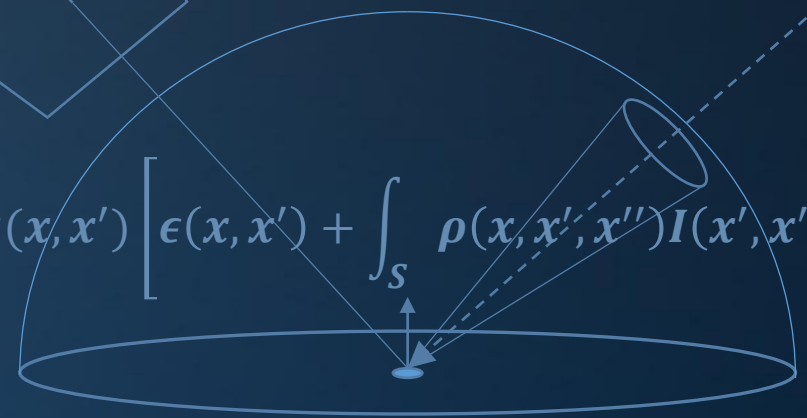


INFOMAGR – Advanced Graphics

Jacco Bikker - November 2022 - February 2023

Lecture 10 - “GPU Ray Tracing (2)”

Welcome!



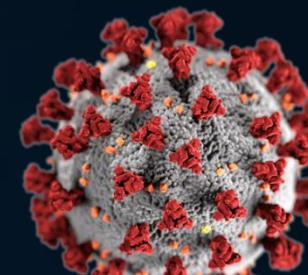
$$I(x, x') = g(x, x') \left[\epsilon(x, x') + \int_S \rho(x, x', x'') I(x', x'') dx'' \right]$$



Today's Agenda:

- State of the Art
- Wavefront Path Tracing
- Assignment 3 Kick-Off
- What's Next

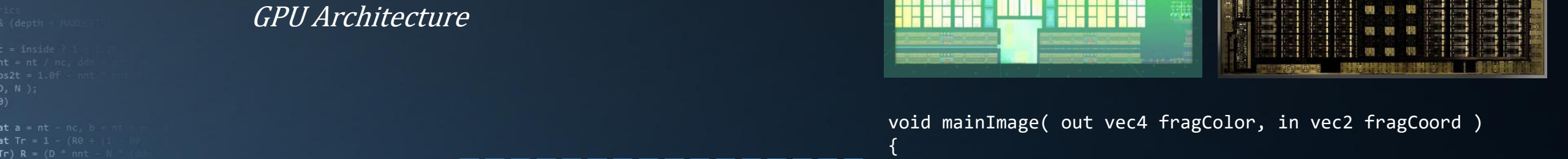
TL;DR version



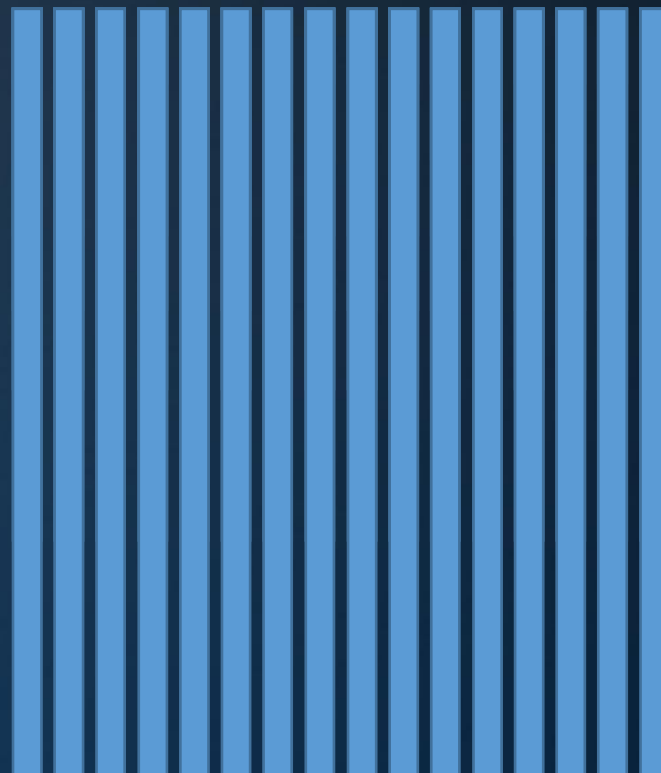
STAR

Previously in Advanced Graphics

GPU Architecture



<https://www.shadertoy.com/view/wdcBW2>



```
void mainImage( out vec4 fragColor, in vec2 fragCoord )
{
    vec2 uv = (fragCoord-.5*iResolution.xy)/iResolution.y;
    uv.y += .355;
    vec2 mouse = iMouse.xy/iResolution.xy;
    uv *= .29;
    vec3 col = vec3(0);
    uv.x = abs(uv.x);
    uv.y += tan(((5./6.)*3.1415))*0.68;
    vec2 n = N((5./6.)*3.1415);
    float d = dot(uv-vec2(.5, 0), n);
    uv -= n*max(0., d)*2.;
    n = N((2./3.)*3.1415);
    float scale = 1.;
    uv.x += .5;
    for(int i=0; i < 1; i++) {
        uv *= 3.;
        scale *= 3.;
        uv.x -= 1.5;
        uv.x = abs(uv.x);
        uv.x -= 2.1;
        uv -= n*min(0., dot(uv, n))*1.;
    }
}
```



STAR

Previously in Advanced Graphics

A Brief History of GPU Ray Tracing

- 2002: Purcell et al., multi-pass shaders with stencil, grid, low efficiency
- 2005: Foley & Sugerman, kD-tree, stack-less traversal with kdrestart
- 2007: Horn et al., kD-tree with short stack, single pass with flow control
- 2007: Popov et al., kD-tree with ropes
- 2007: Günther et al., BVH with packets.

- The use of BVHs allowed for complex scenes on the GPU (millions of triangles);
- CPU is now outperformed by the GPU;
- GPU compute potential is not realized;
- Aspects that affect efficiency are poorly understood.

Interactive k-D Tree GPU Raytracing

Daniel Reiter Horn Jeremy Sugerman Mike Houston Pat Hanrahan
Stanford University *

We add three major enhancements to their *kd-restart* algorithm: *push-down*, and *short-stack*. Packets are used to traverse the tree (Wald et al. 2001).

To appear in the IEEE/Eurographics Symposium on Interactive Ray Tracing 2007.

Realtime Ray Tracing on GPU with BVH-based Packet Traversal

Johannes Günther*
MPI Informatik

Stefan Popov†
Saarland University

Hans-Peter Seidel*
MPI Informatik

Philipp Slusallek‡
Saarland University



Figure 1: The Cornell Box rendered on the GPU.

STAR

Understanding the Efficiency of Ray Traversal on GPUs*

Observations on BVH traversal:

Ray/scene intersection consists of an unpredictable sequence of node traversal and primitive intersection operations. This is a major cause of inefficiency on the GPU.

Random access of the scene leads to high bandwidth requirement of ray tracing.

BVH packet traversal as proposed by Gunther et al. should alleviate bandwidth strain and yield near-optimal performance.

Packet traversal doesn't yield near-optimal performance. Why not?

*: Understanding the Efficiency of Ray Tracing on GPUs, Aila & Laine, 2009.

and: Understanding the Efficiency of Ray Tracing on GPUs – Kepler & Fermi addendum, 2012.



STAR

Understanding the Efficiency of Ray Traversal on GPUs*

```

ics
& (depth < MAXDEPTH)
{
    if (nt < 0)
        nt = inside ? 1 : -1;
    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) * f);
        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, &light );
    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;
}

```

What follows is a wonderful exposition of engineering details, featuring an ‘ideal GPU’ simulator and a series of simple traversal schemes – which leads to an unlikely culprit. Spoiler: bandwidth is *not* the problem.

Now go an read the paper. 😊

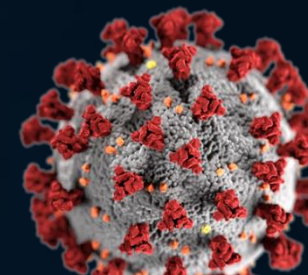
Why not?



Today's Agenda:

- State of the Art
- Wavefront Path Tracing
- Assignment 3 Kick-Off
- What's Next

TL;DR version



Wavefront

Mapping Path Tracing to the GPU

The path tracing loop from lecture 8 is straight-forward to implement on the GPU.

However:

- Terminated paths become idling threads;
- A significant number of paths will not trace a shadow ray.

```
Color Sample( Ray ray )
{
    T = ( 1, 1, 1 ), E = ( 0, 0, 0 );
    while (1)
    {
        I, N, material = Trace( ray );
        BRDF = material.albedo / PI;
        if (ray.NOHIT) break;
        if (material.isLight) break;
        // 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;
            lightPDF = 1 / solidAngle;
            E += T * (N·L / lightPDF) * BRDF * lightColor;
        }
        // continue random walk
        R = DiffuseReflection( N );
        hemiPDF = 1 / (PI * 2.0f);
        ray = Ray( I, R );
        T *= ((N·R) / hemiPDF) * BRDF;
    }
    return E;
}
```



Wavefront



```

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

```

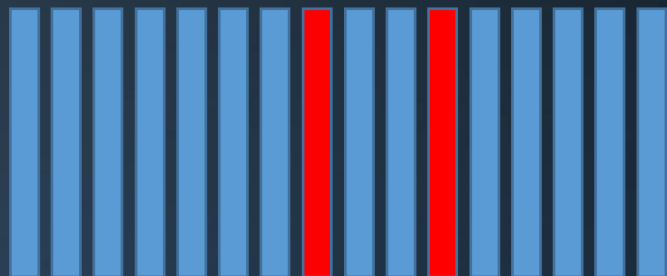
```

Color Sample( Ray ray )
{
    T = ( 1, 1, 1 ), E = ( 0, 0, 0 );
    while (1)
    {
        I, N, material = Trace( ray );
        BRDF = material.albedo / PI;
        if (ray.NOHIT) break;
        if (material.isLight) break;
        // 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;
            lightPDF = 1 / solidAngle;
            E += T * (N·L / lightPDF) * BRDF * lightColor;
        }
        // continue random walk
        R = DiffuseReflection( N );
        hemiPDF = 1 / (PI * 2.0f);
        ray = Ray( I, R );
        T *= ((N·R) / hemiPDF) * BRDF;
    }
    return E;
}

```



Wavefront



```

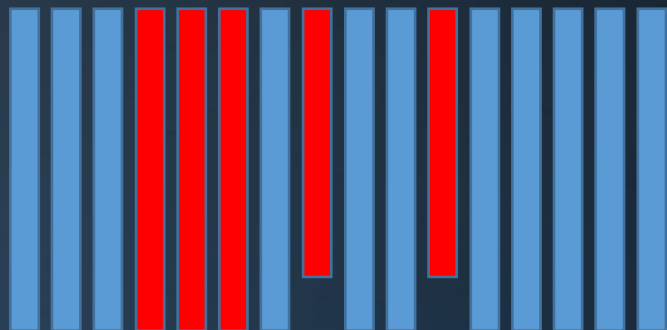
ics
& (depth < MAXDEPTH)
{
    if (inside ? 1 : 0)
    {
        nt = nt / nc; ddn = ddn * nc;
        cos2t = 1.0f - nnt * ddn;
        D, N );
    }
    at a = nt - nc, b = nt + nc;
    at Tr = 1 - (R0 + (1 - R0) *
    Tr) R = (D * nnt - N * (ddn
    E * diffuse;
    = true;
    -
    efl + refr)) && (depth < MAXDEPTH)
    D, N );
    refl * E * diffuse;
    = true;
    MAXDEPTH)
    survive = SurvivalProbability( diffuse );
    estimation - doing it properly, closely following
    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;
    
```

```

Color Sample( Ray ray )
{
    T = ( 1, 1, 1 ), E = ( 0, 0, 0 );
    while (1)
    {
        I, N, material = Trace( ray );
        BRDF = material.albedo / PI;
        if (ray.NOHIT) break;
        if (material.isLight) break;
        // 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;
            lightPDF = 1 / solidAngle;
            E += T * (N·L / lightPDF) * BRDF * lightColor;
        }
        // continue random walk
        R = DiffuseReflection( N );
        hemiPDF = 1 / (PI * 2.0f);
        ray = Ray( I, R );
        T *= ((N·R) / hemiPDF) * BRDF;
    }
    return E;
}
    
```



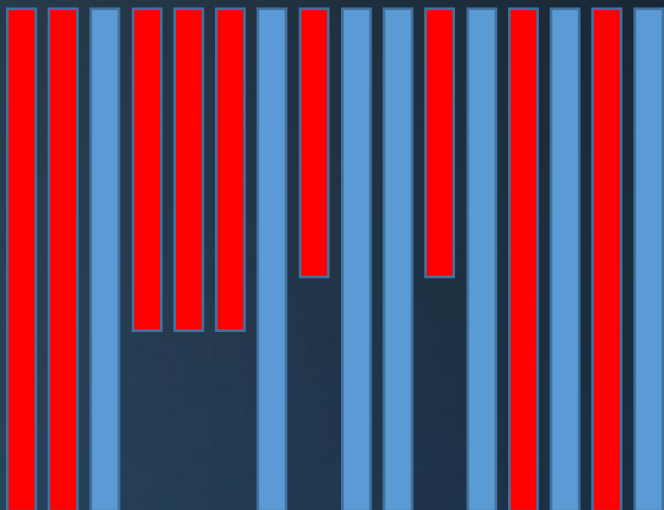
Wavefront



```
Color Sample( Ray ray )
{
    T = ( 1, 1, 1 ), E = ( 0, 0, 0 );
    while (1)
    {
        I, N, material = Trace( ray );
        BRDF = material.albedo / PI;
        if (ray.NOHIT) break;
        if (material.isLight) break;
        // 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;
            lightPDF = 1 / solidAngle;
            E += T * (N·L / lightPDF) * BRDF * lightColor;
        }
        // continue random walk
        R = DiffuseReflection( N );
        hemiPDF = 1 / (PI * 2.0f);
        ray = Ray( I, R );
        T *= ((N·R) / hemiPDF) * BRDF;
    }
    return E;
}
```



Wavefront



```

Color Sample( Ray ray )
{
    T = ( 1, 1, 1 ), E = ( 0, 0, 0 );
    while (1)
    {
        I, N, material = Trace( ray );
        BRDF = material.albedo / PI;
        if (ray.NOHIT) break;
        if (material.isLight) break;
        // 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;
            lightPDF = 1 / solidAngle;
            E += T * (N·L / lightPDF) * BRDF * lightColor;
        }
        // continue random walk
        R = DiffuseReflection( N );
        hemiPDF = 1 / (PI * 2.0f);
        ray = Ray( I, R );
        T *= ((N·R) / hemiPDF) * BRDF;
    }
    return E;
}

```

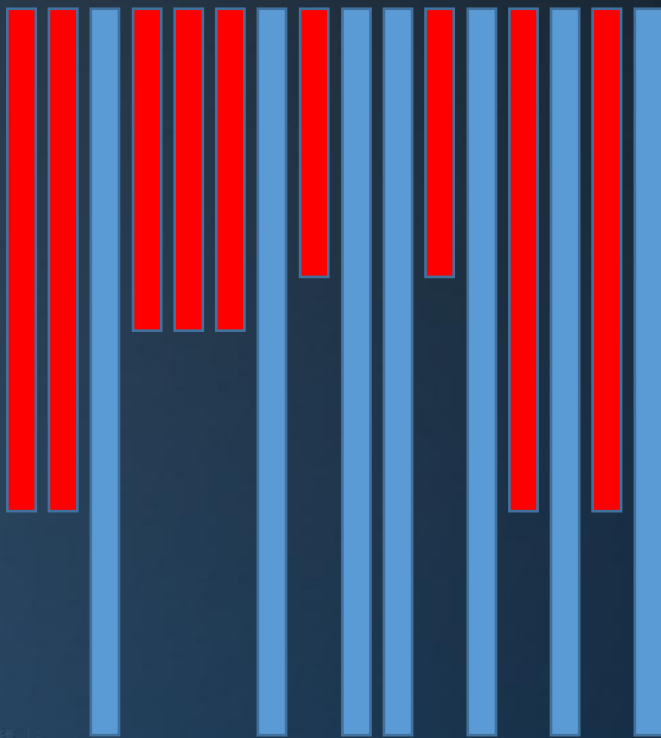


Wavefront

```

ics
& (depth < MAXDEPTH)
{
    if (nt < nc) {
        nt = nt / nc;
        ddn = ddn * nt;
        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
    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);
    sion = true;

```



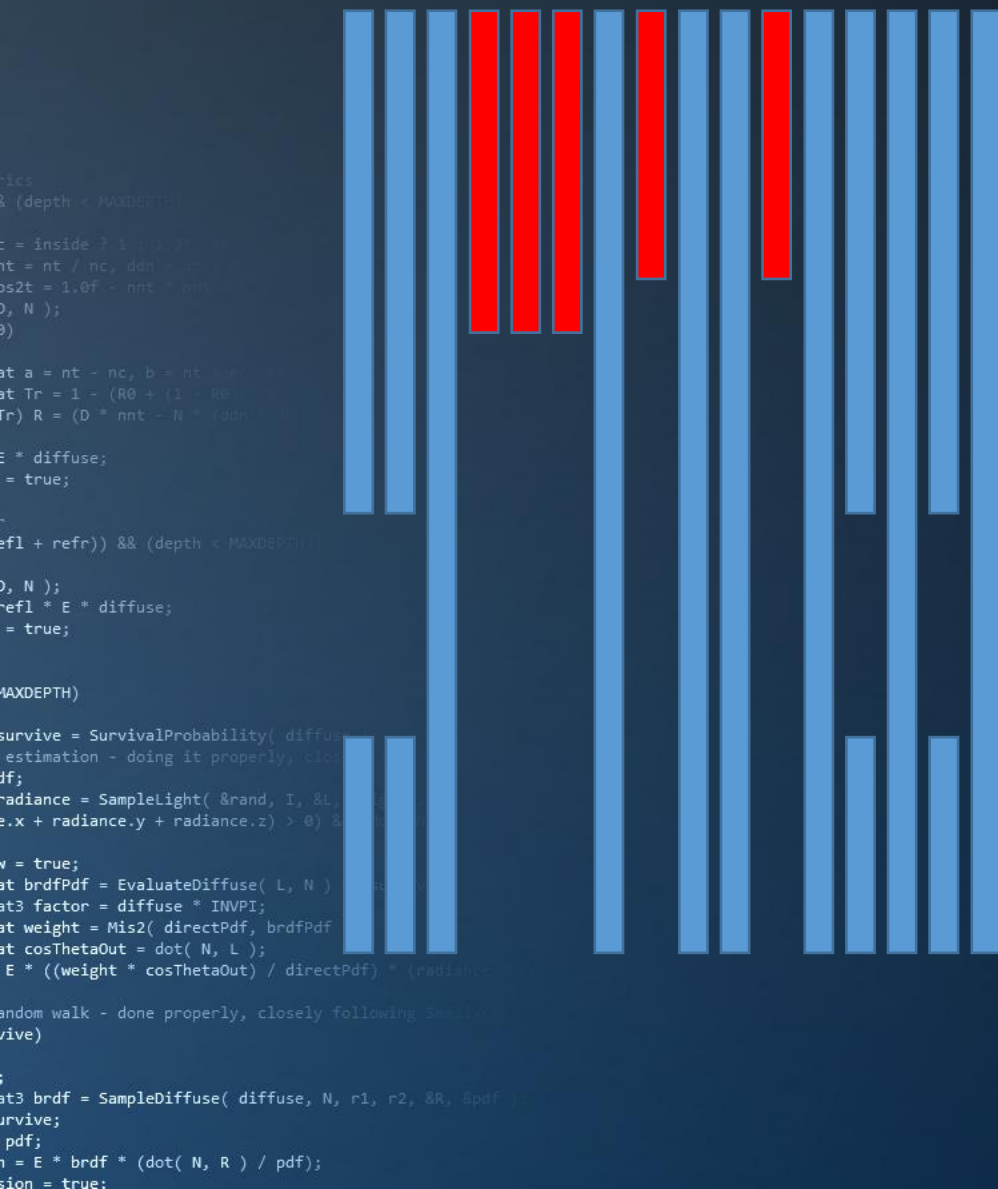
```

Color Sample( Ray ray )
{
    T = ( 1, 1, 1 ), E = ( 0, 0, 0 );
    while (1)
    {
        I, N, material = Trace( ray );
        BRDF = material.albedo / PI;
        if (ray.NOHIT) break;
        if (material.isLight) break;
        // 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;
            lightPDF = 1 / solidAngle;
            E += T * (N·L / lightPDF) * BRDF * lightColor;
        }
        // continue random walk
        R = DiffuseReflection( N );
        hemiPDF = 1 / (PI * 2.0f);
        ray = Ray( I, R );
        T *= ((N·R) / hemiPDF) * BRDF;
    }
    return E;
}

```



Wavefront



```

Color Sample( Ray ray )
{
    T = ( 1, 1, 1 ), E = ( 0, 0, 0 );
    while (1)
    {
        I, N, material = Trace( ray );
        BRDF = material.albedo / PI;
        if (ray.NOHIT) break;
        if (material.isLight) break;
        // 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;
            lightPDF = 1 / solidAngle;
            E += T * (N·L / lightPDF) * BRDF * lightColor;
        }
        // continue random walk
        R = DiffuseReflection( N );
        hemiPDF = 1 / (PI * 2.0f);
        ray = Ray( I, R );
        T *= ((N·R) / hemiPDF) * BRDF;
    }
    return E;
}

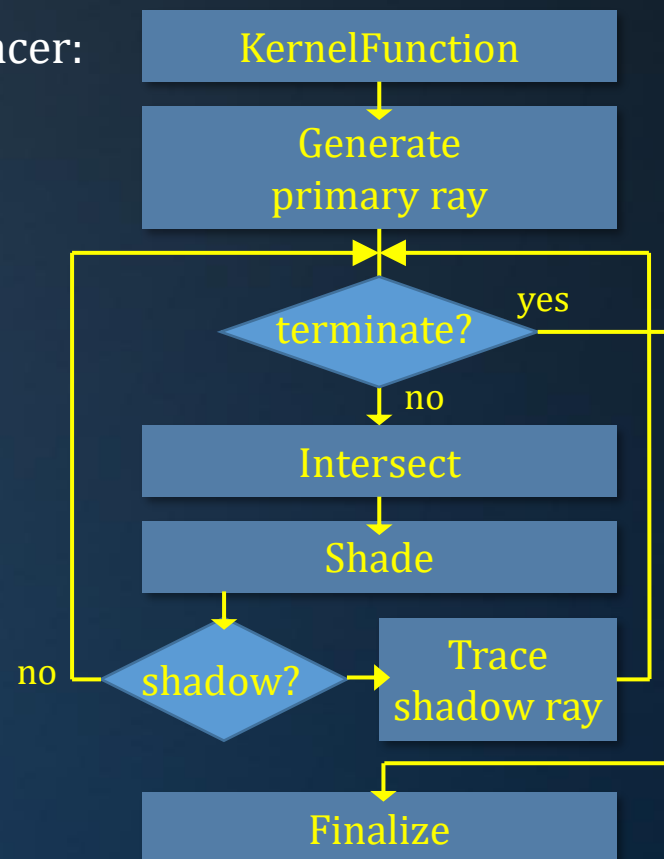
```



Wavefront

Megakernels Considered Harmful*

Naïve path tracer:



Translating this to CUDA or OpenCL code yields a single kernel: individual functions are still compiled to one monolithic chunk of code.

Resource requirements (registers) - and thus parallel slack - are determined by 'weakest link', i.e. the functional block that requires most registers.

Conditional code leads to idling threads that wait until others are done.

*Megakernels Considered Harmful: Wavefront Path Tracing on GPUs, Laine et al., 2013



Wavefront

Megakernels Considered Harmful

Solution: *split the kernel*.

Example:

Kernel 1: Generate primary rays.

Kernel 2: Trace paths.

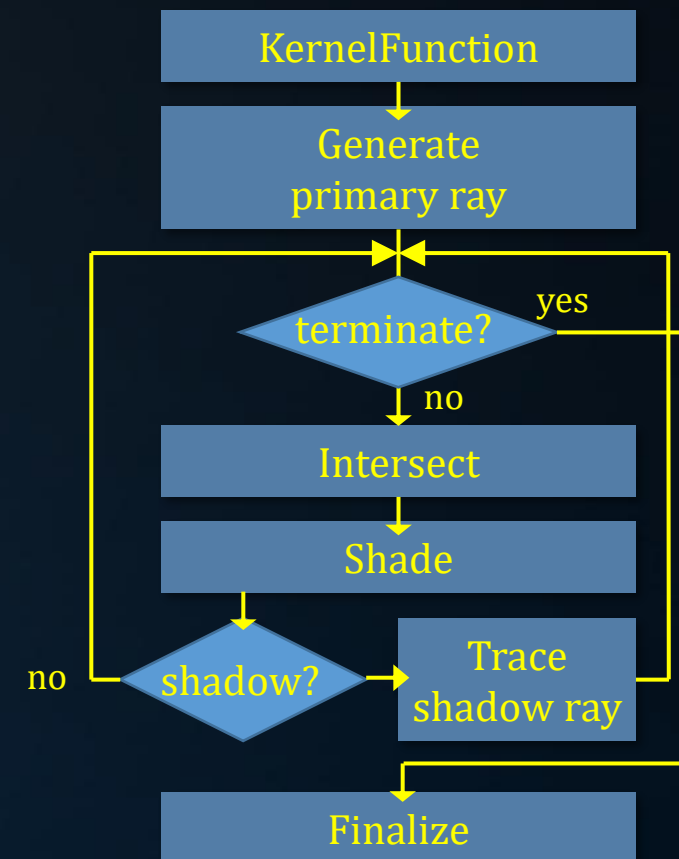
Kernel 3: Accumulate, gamma correct, convert to ARGB32.

Consequence:

Kernel 1 generates *all* primary rays, and stores the result.

Kernel 2 takes this buffer and operates on it.

➔ *Massive memory I/O.*



Wavefront

Megakernels Considered Harmful

Taking this further: streaming path tracing*.

Kernel 1: generate primary rays.

Kernel 2: extend.

Kernel 3: shade.

Kernel 4: connect.

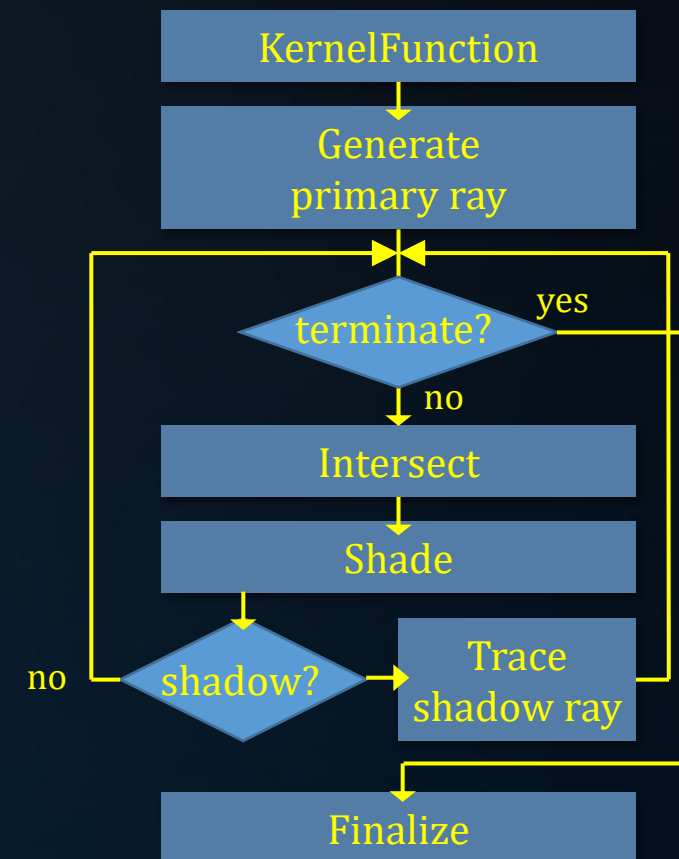
Kernel 5: finalize.

Here, kernel 2 traces a set of rays to find the next path vertex (the random walk).

Kernel 3 processes the results and generates new path segments and shadow rays (2 separate buffers).

Kernel 4 traces the shadow ray buffer.

Kernel 1, 2, 3 and 4 are executed in a loop until no rays remain.



```

ics
& (depth < MAXDEPTH)
{
    if (inside ? 1 : 0)
    {
        nt = nt / nc; ddn = ddn * nc;
        cos2t = 1.0f - nnt * nnt;
        D, N );
    }
    at a = nt - nc, b = nt + nc;
    at Tr = 1 - (R0 + (1 - R0) *
    (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
    df;
    radiance = SampleLight( &rand, I, &L, Align
    e.x + radiance.y + radiance.z) > 0) && (ac
    w = true;
    at brdfPdf = EvaluateDiffuse( L, N ) * Psum
    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 S
    vive)
    ;
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    sion = true;
  
```

*Improving SIMD Efficiency for Parallel Monte Carlo Light Transport on the GPU, van Antwerpen, 2011



Wavefront

Megakernels Considered Harmful

Zooming in:

The **generate** kernel produces N primary rays:

0, 1,, $N-1$

Buffer 1: path segments (N times $O,D,t,primIdx$)

The **extend** kernel traces extension rays and produces intersections*.

The **shade** kernel processes intersections, and produces new extension paths as well as shadow rays:

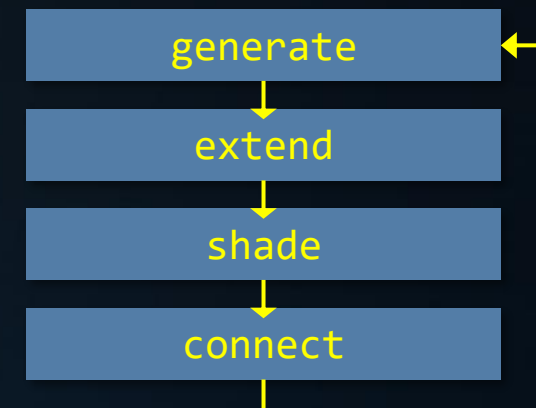
0, 1,, $N-1$

Buffer 2: generated path segments (N times $O,D,t,primIdx$)

0, 1,, $N-1$

Buffer 3: generated shadow rays (N times $O,D,t, E, pixelIdx$)

Finally, the **connect** kernel traces shadow rays.



Note: here, the loop is implemented on the host. Each block is a separate kernel invocation.

*: An intersection is at least the t value, plus a primitive identifier.



Wavefront

Megakernels Considered Harmful

Generate:

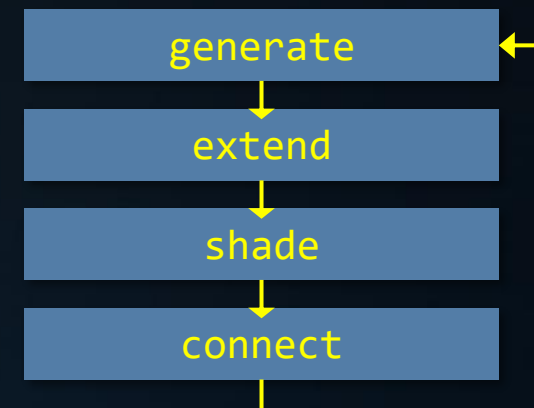
for each screen pixel i

{

0, D = GenerateRayDirection(i)

rayBuffer[i] = Ray(0, D, infinity, -1)

}



Wavefront

Megakernels Considered Harmful

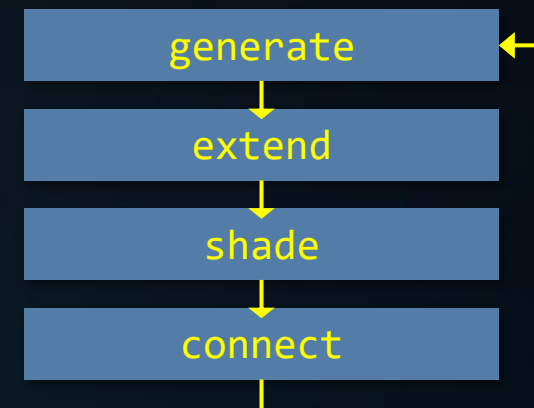
Extend:

for each buffered ray r

{

```
O,D,dist = rayBuffer[i]
dist, primIdx = FindNearestIntersection( 0, D, dist )
rayBuffer[i].dist = dist
rayBuffer[i].primIdx = primIdx
```

}



Wavefront

Megakernels Considered Harmful

Shade:

for each buffered ray r

{

O,D,dist,primIdx = rayBuffer[i]

I = IntersectionPoint(O, D, dist)

N = PrimNormal(primIdx, I)

if (NEE) {

si = atomicInc(shadowRayIdx)

shadowBuffer[si] = ShadowRay(...)

}

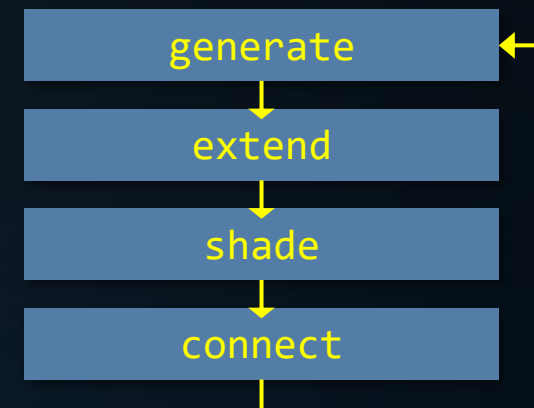
if (bounce) {

ei = atomicInc(extensionRayIdx)

newRayBuffer[ei] = ExtensionRay(...)

}

}



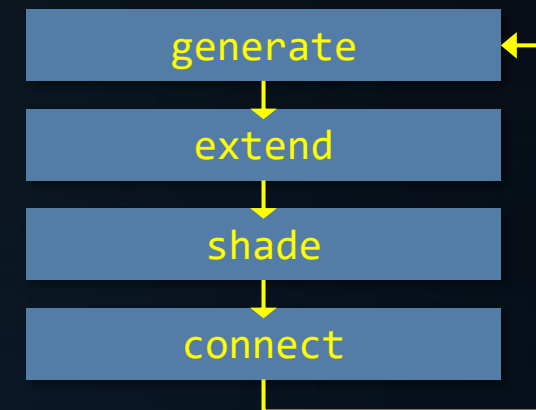
Wavefront

Megakernels Considered Harmful

Connect:

for each buffered shadowRay r

```
{
    O,D,dist,E, pixelIdx = shadowBuffer[i]
    if (!Occluded( O, D, dist ))
    {
        accumulator[pixelIdx] += E;
    }
}
```



Wavefront

Megakernels Considered Harmful

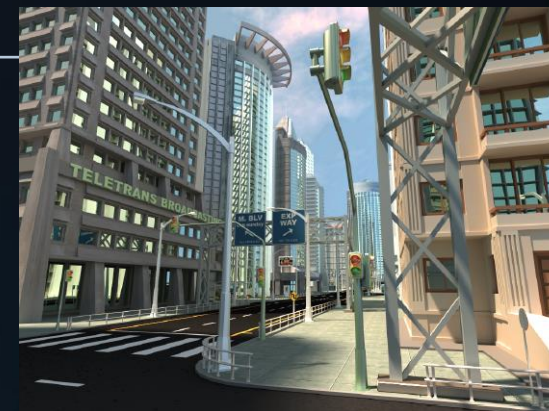
Digest:

Streaming path tracing introduces seemingly costly operations:

- Repeated I/O to/from large buffers;
- A significant number of kernel invocations per frame;
- Communication with the host.

The Wavefront paper claims that this is beneficial for complex shaders. In practice, this also works for (very) simple shaders.

Also note that the megakernel paper (2013) presents an idea already presented by Dietger van Antwerpen (2011).



Today's Agenda:

- State of the Art

TL;DR version

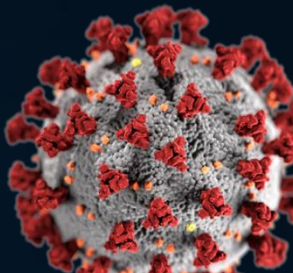
- Wavefront Path Tracing
- Assignment 3 Kick-Off
- What's Next



Today's Agenda:

- State of the Art
- Wavefront Path Tracing
- 2023 Update: SER
- Assignment 3 Kick-Off
- What's Next

TL;DR version



Shader Execution Reordering (SER) is a new scheduling technology introduced with the Ada Lovelace generation of NVIDIA GPUs. It is highly effective at simultaneously reducing both execution divergence and data divergence. SER achieves this by on-the-fly reordering threads across the GPU such that groups of threads perform similar work and therefore use GPU resources more efficiently. This happens with minimal overhead: the Ada hardware architecture was designed with SER in mind and includes optimizations to the SM and memory system specifically targeted at efficient thread reordering. Using SER, we observe speedups of up to 2x in raytracing regimes of real-world applications, achieved with only a small amount of developer effort. To applications, SER is exposed through a small API that gives developers new flexibility and full control over where in their shaders reordering will happen. This API is detailed in the following sections.

API overview

ReorderThread

The main functionality of SER is encapsulated in a single function:

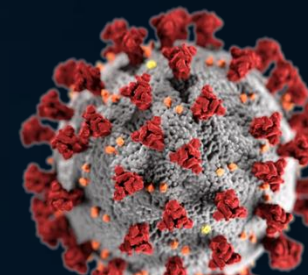
```
void ReorderThread( key )
```

Today's Agenda:

- State of the Art

TL;DR version

- Wavefront Path Tracing
- 2023 Update: SER
- Assignment 3 Kick-Off
- What's Next



Assignment 1: *Build a framework*

Assignment 2: *Add an acceleration structure*

Assignment 3: Freestyle.

```

    % (depth < MAXDEPTH)
    nc = inside ? 1 : 1.01 * nt;
    nt = nt / nc; ddn = ddn * nc;
    cos2t = 1.0f - nnt * nnt;
    D, N );
    )
    at a = nt - nc, b = nt + nc;
    at Tr = 1 - (R0 + (1 - R0) * ddn);
    Fr) 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, close
    ff;
    radiance = SampleLight( &rand, I, &L
    .x + radiance.y + radiance.z) > 0)
    w = true;
    at brdfPdf = EvaluateDiffuse( L, N )
    at3 factor = diffuse * INVPI;
    at weight = Mis2( directPdf, brdfPdf
    at cosThetaOut = dot( N, L );
    E * (weight * cosThetaOut) / direc
    random walk - done properly, closely
    vive)
    ;
    at3 brdf = SampleDiffuse( diffuse, N
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    sion = true;

```

ion structure

Advanced Graphics 2022/2023 – Assignment 3 (FINAL)

DRAFT (until Jan 8)

Assignment 1 was about light transport

Assignment 2 dealt with efficient

Assignment 3 you

Introduction

Introduction

For this assignment, you have considerable freedom. **Assignment 1** was about light transport fundamentals and setting up the framework for basic ray tracing. **Assignment 2** dealt with efficient ray/scene intersection: the low-level core of any ray tracing based renderer. For **Assignment 3** you either build on this by implementing recent research.

theory presented in the lectures

Assignment

Broadly speaking, the main types of organizational structures are:

- badly speaking, there are four areas to work on, linked to the theory presented in the lectures:
1. Acceleration structures:

Assignment 3

Freestyle

General idea: *implement a somewhat recent paper (or part of it) from the field of computer graphics.*

Scoring:

- 6 for a correct implementation of an ‘easy’ paper
- 7.5 for a ‘medium’ one
- 9 for a ‘hard’ paper.

NOTE: if you tackle multiple papers, the grade will be based on the hardest one (no stacking), so feel free to focus on one problem.

The final point is for excellence, presentation and analysis and is strongly subjective (but open for discussion afterwards).



Assignment 3

Freestyle

Topics – part 1 – Acceleration Structures

- Expanding on assignment 2, implement the paper “Spatial Splits in Bounding Volume Hierarchies” by Stich et al. This yields a high-quality BVH, at the expense of construction time. Note: Proof that the resulting tree is of a high quality is required. **Difficulty: medium/hard.**
- Or, go the opposite route: implement the paper “Bonsai: Rapid Bounding Volume Hierarchy Generation using Mini Trees” by Ganestam et al., which maximizes build performance. Note: build performance must be roughly in line with the numbers in the paper. **Difficulty: medium/hard.**
- Starting from a TLAS/BLAS builder from the tutorial series, add ‘partial rebraiding’, based on the paper “Improved two-level BVHs using partial re-braiding” by Benthin et al. Prove that the resulting TLAS/BLAS performs in line with the findings in the paper. **Difficulty: easy.**
- Other options exist. There is a paper by NVIDIA that explains how to efficiently traverse a (CPU-built) 8-wide BVH on the GPU for state-of-the-art #RTXoff performance (**hard**). There exist other heuristics than SAH; an implementation and an in-depth analysis may provide you with an interesting project (**easy**).
- Feel free to propose something interesting!



Assignment 3

Freestyle

Topics – part 2 – Physically-based Rendering

- Implement a basic but correct bidirectional path tracer, as described by Veach in his Ph.D. thesis (as well as in many other sources). Proof the correctness of your implementation by comparing energy levels. **Warning: hard.**
- Implement photon mapping, as described by Henrik Wann Jensen (“Global Illumination using Photon Maps”, 1996). Your implementation should correctly handle caustics. Verify your result against ground truth produced by a path tracer. **Warning: must produce correct energy levels. Difficulty: easy – medium.**
- Implement a basic but correct volumetric path tracer for non-homogeneous media, e.g. a cloud. Provide a test scene that clearly demonstrates the functionality. **Difficulty: medium.**
- Other options: feel free to propose something interesting.



Assignment 3

Freestyle

Topics – part 3 – Filtering & Reprojection

- Implement a filter that combines reprojection and spatial filtering to improve image quality. Suggestion: use a simple scene to make reprojection worthwhile. **Difficulty: easy.**
- Implement Adaptive Sampling (see e.g. “A Survey of Adaptive Sampling in Realistic Image Synthesis”, M. Šik. **Difficulty: easy.**
- Add the “Open Path Guiding Library” to your existing renderer and report on the integration process. **Difficulty: easy, I think.**
- Implement a recent paper on the topic of filtering (see 4).



Assignment 3

Freestyle

Topics – part 4 – Other Options

- “Path Guiding in Production”, by Vorba et al., 2019. Path guiding is a class of ‘learning algorithms’, which estimate importance of directions over the hemisphere based on earlier transport results. **Difficulty: medium.**
- “Direct Ray Tracing of Smoothed and Displacement Mapped Triangles”, Smits et al., 2000. The paper describes a method for directly rendering displacement maps in a ray tracer. The method is computationally expensive, which made it impractical in 2000. Now, in 2020, things have changed. **Difficulty: I think hard. (2023 update: yes, hard).**
- “Spatiotemporal reservoir resampling for real-time ray tracing with dynamic direct lighting”, Bitterli et al., 2020. As discussed in the slides: a method for importance sampling thousands of lights. **Difficulty: medium, but potentially a lot of work.**
- Implement Wavefront Path Tracing on the GPU. **Difficulty: medium, but rewarding.**

```

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

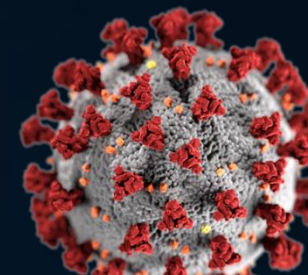
```



Today's Agenda:

- State of the Art
- Wavefront Path Tracing
- 2023 Update: SER
- Assignment 3 Kick-Off
- What's Next

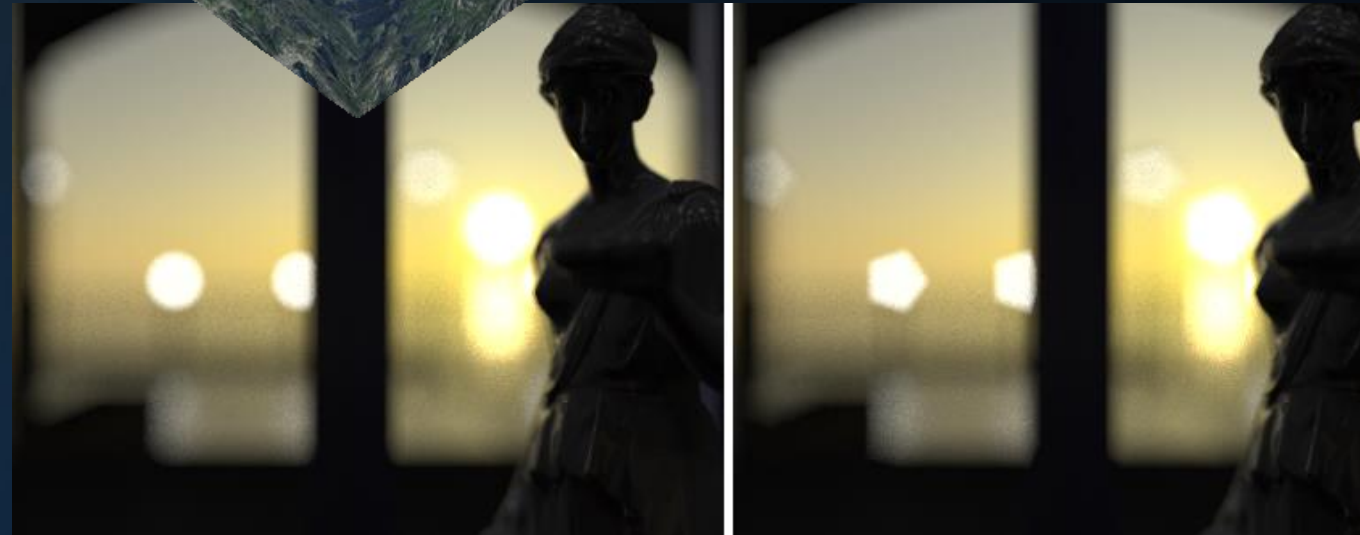
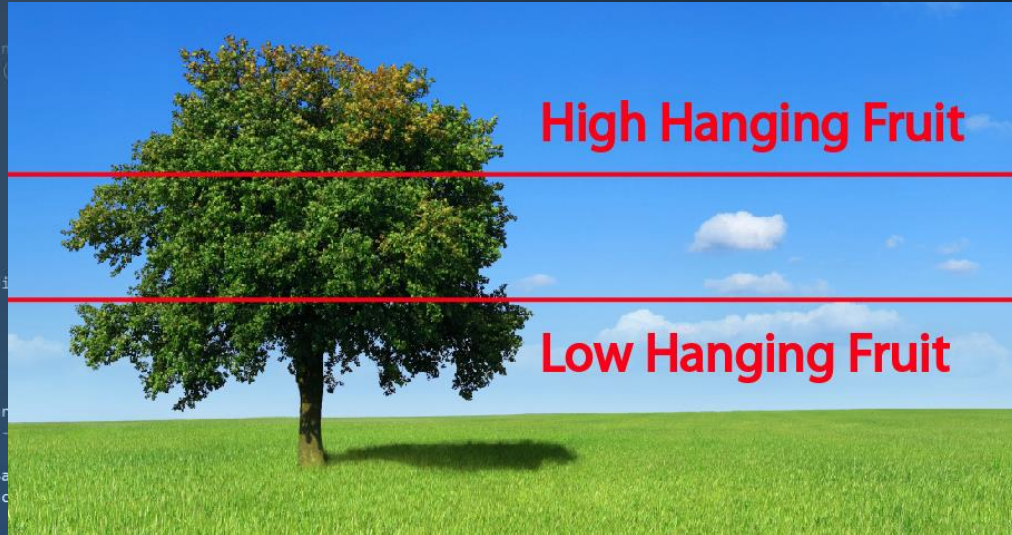
TL;DR version



MAGR – What's Next

Upcoming Topics

Thursday: “Various”



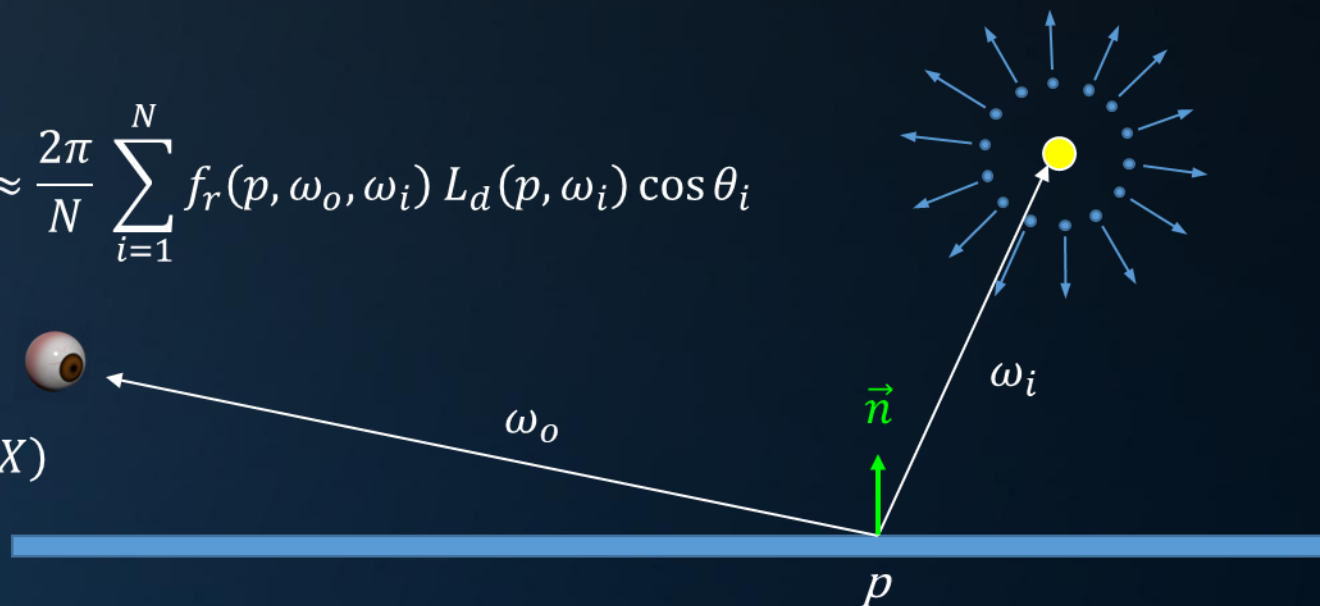
Upcoming Topics

```

1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
183
```

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

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



Upcoming Topics

Next

```

MAXDEPTH)

survive = SurvivalProbability( diffuse, N, L );
// estimation - doing it properly, close to Monte Carlo
pdf;
radiance = SampleLight( &rand, I, &L, &lightN );
if ( diffuse.x + radiance.y + radiance.z ) > 0 ) && ( dots < 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);
}
// random walk - done properly, closely following Monte Carlo
survive)

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

```

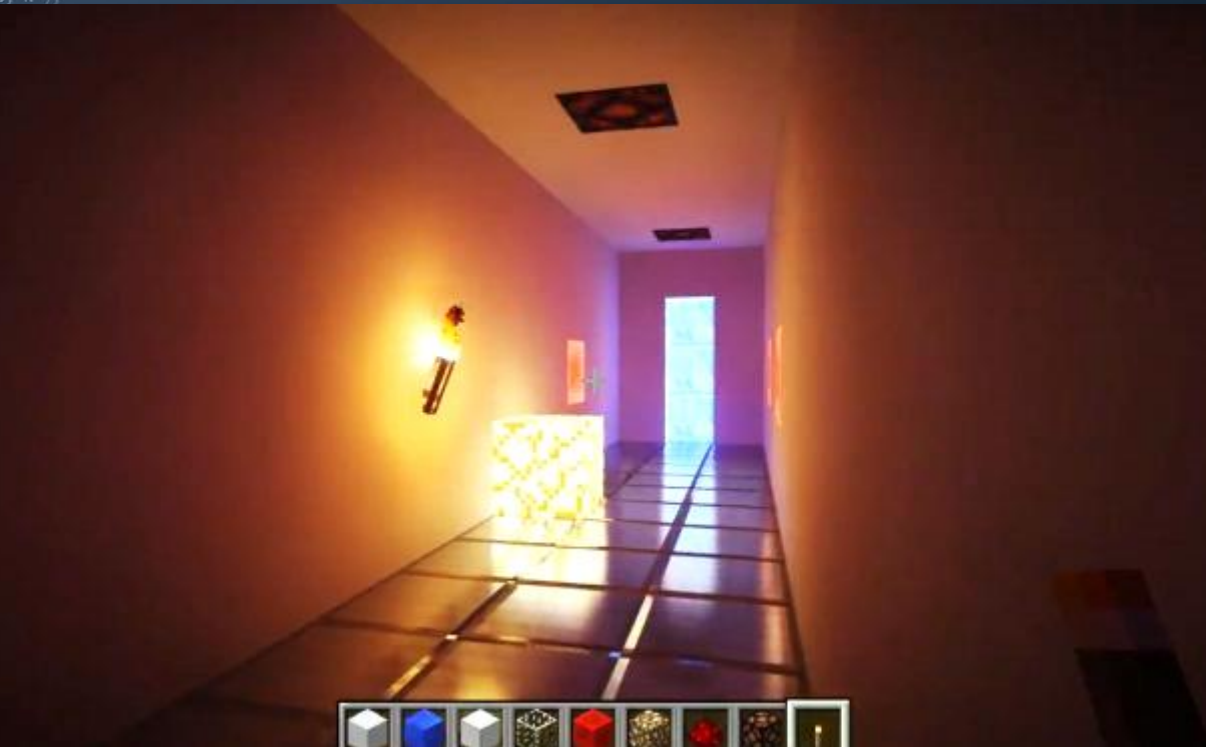


MAGR – What's Next

Upcoming Topics

Later: “ReSTIR”

```
ics  
& (depth < MAXDEPTH)  
  
t = inside ? 1.0f : 0.0f;  
nt = nt / nc, ddn = ddn / nc;  
pos2t = 1.0f - nnt * ddn;  
D, N );
```



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



MAGR – What's Next

Upcoming Topics

Later: “Filtering”

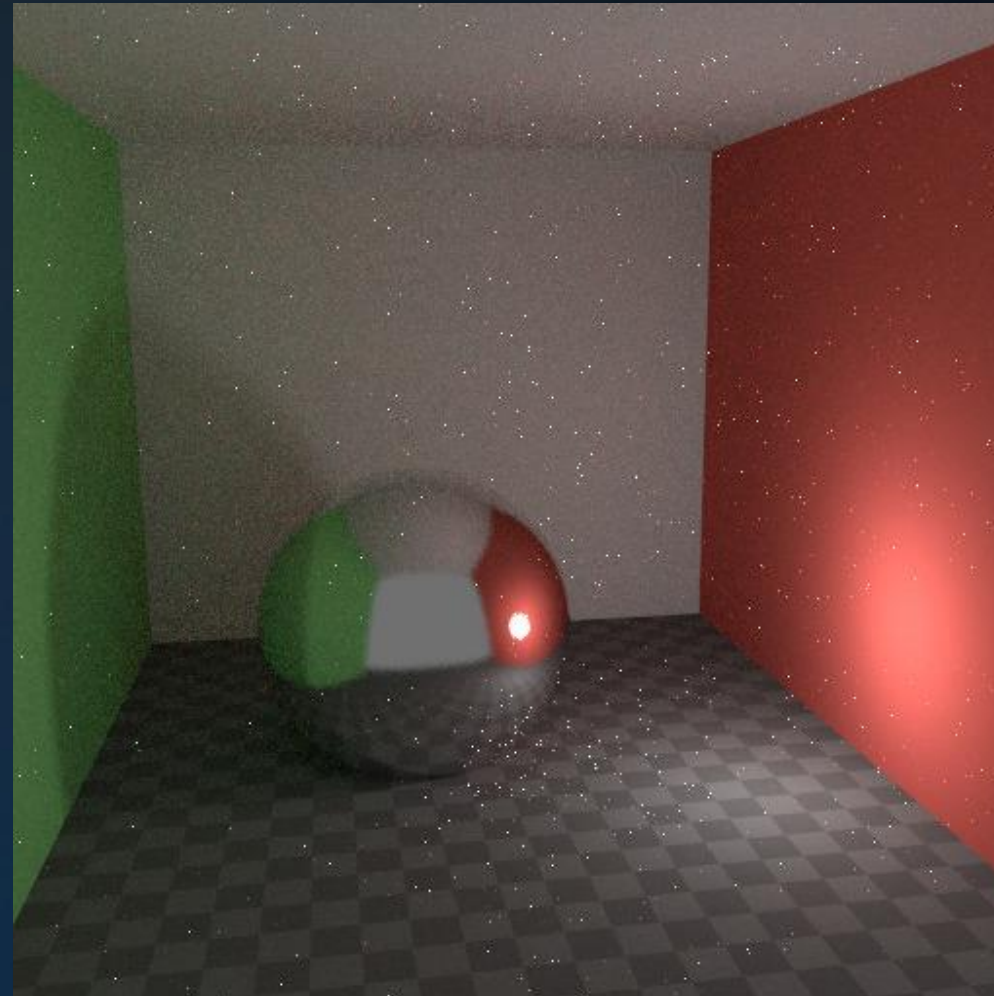
```

ics
& (depth < MAXDEPTH)
{
    if (inside ? 1 : 0)
    {
        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) * f);
        Tr) R = (D * nnt - N * (ddn < 0 ? 1 : -1));
        E * diffuse;
        = true;
    }
    else
    {
        refl + refr)) && (depth < MAXDEPTH)
        D, N );
        refl * E * diffuse;
        = true;
    }
}

MAXDEPTH)

survive = SurvivalProbability( diffuse );
estimation - doing it properly, closely following Small's
df;
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;
}

```



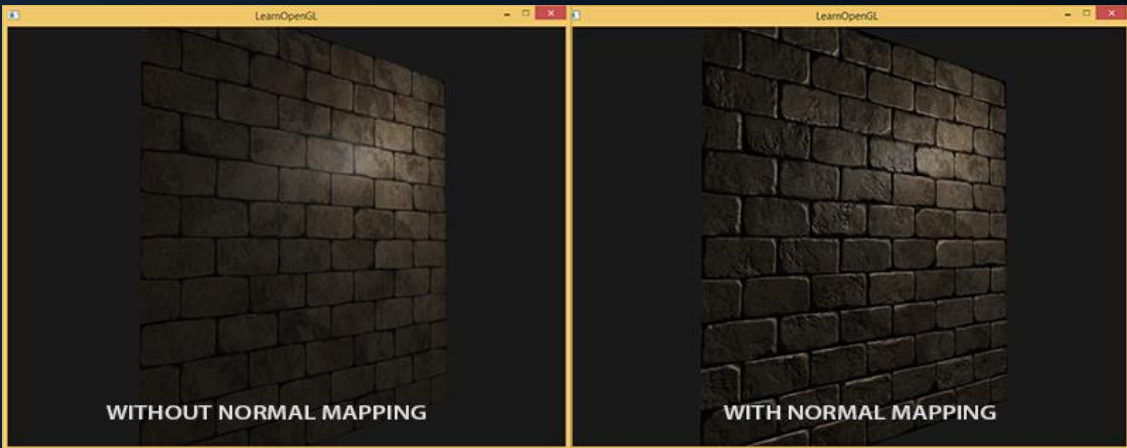
MAGR – What’s Next

Upcoming Topics

Later: “Bit’s & Pieces”



EXAM PRACTICE

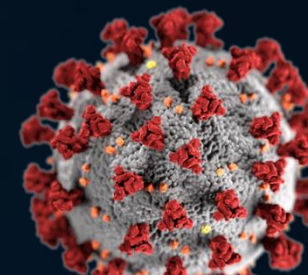


Today's Agenda:

- State of the Art

TL;DR version

- Wavefront Path Tracing
- 2023 Update: SER
- Assignment 3 Kick-Off
- What's Next



INFOMAGR – Advanced Graphics

Jacco Bikker - November 2022 - February 2023

END of “GPU Ray Tracing (2)”

next lecture: “Variance Reduction (2)”

