

Ray Tracing for Games

Dr. Jacco Bikker - IGAD/BUAS, Breda, January 30

Welcome!

2



Agenda:

- Current State
- Advanced Whitted
- Generic Ray Queries
- Faster Whitted

```
ics
& (depth < MAXDEPTH))
{
    if (nt < 0)
        nt = inside ? 1 : -1;
    nt = nt / nc; ddn = dot(N, N);
    float r0 = 1 - nt; r1 = 0; r2 = 0;
    float cos2t = 1.0f - nnt * nnt;
    float r = sqrtf(cos2t);
    D, N );
    }

    float a = nt - nc, b = nt + nc;
    float Tr = 1 - (R0 + (1 - R0) * r);
    float R = (D * nnt - N * (ddn * r0 +
    ) * r);

    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 Smallcap
    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

YOU ARE HERE

Thursday
09:00 – 14:00

advanced Whitted
audio, AI & physics
faster Whitted
Heaven7

Wednesday
13:00 – 17:00

LAB 2



work @ home

End result day 2:

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

Friday
09:00 – 17:00

optimization
profiling, rules of
engagement
threading



LAB 3

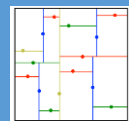
SIMD
applied SIMD
SIMD triangle
SIMD AABB

LAB 4

End result day 3:
A 5x faster tracer.

Monday
09:00 – 17:00

acceleration
grid, BVH, kD-tree
SAH
binning



LAB 5

refitting
top-level BVH
threaded building

LAB 6

End result day 4:
A real-time tracer.

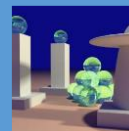
Tuesday
09:00 – 17:00

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



LAB 7

path tracing

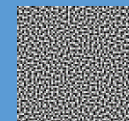


LAB 8

End result day 5:
Cook or Kajiya.

Thursday
09:00 – 17:00

random numbers
stratification
blue noise



LAB 9

importance
sampling
next event
estimation

LAB 10

End result day 6:
Efficiency.

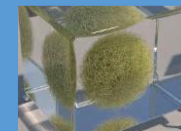
Friday
09:00 – 17:00

future work



LAB 11

path guiding



LAB 10

End result day 6:
Great product.

GLOBAL GAME JAM



End product:

1. Path tracer that produces pretty images in a few minutes.

You have something on the screen / you dusted off your old ray tracer.

2. Real-time ray tracer on the CPU or GPU.

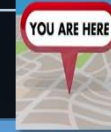
You explored the template and/or managed to build LH2.
You have something on the screen.

3. Raytraced game or demo.

You formed a team and assigned roles.
You have decided on a game concept.

4. RTX port of an existing game.

You formed a team.
You picked a game you want to port.



Thursday
09:00 – 14:00

advanced Whitted
audio, AI & physics
faster Whitted
Heaven7

Friday
09:00 – 17:00

optimization
profiling, rules of
engagement
threading



LAB 2

LAB 3



SIMD
applied SIMD
SIMD triangle
SIMD AABB

work @ home

End result day 2:

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

LAB 4

End result day 8:

A 5x faster tracer.

Wednesday
13:00 – 17:00

course intro
LH2
template
Whitted
refactoring
RT-centric games

LAB 1



Today: Consolidation

Ensure that you have something that renders something today, beat the 'First Image' obstacle.

Take some time to brush up your code so that it is convenient to work with and feels comfortable.

Take in some information about speeding things up, for a concrete short term goal.

When you go to sleep tonight you reached ALPHA.

(i.e., something that works – we'll improve it beyond recognition)

Going for the Game Jam? You will need some tricks to have a working engine before Friday evening.

YOU ARE HERE

Thursday

09:00 – 14:00

advanced Whitted
audio, AI & physics
faster Whitted
Heaven7

LAB 2



work @ home

End result day 2:

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

Friday

09:00 – 17:00

optimization
profiling, rules of
engagement
threading



LAB 3

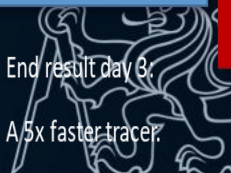
SIMD
applied SIMD
SIMD triangle
SIMD AABB

LAB 4

End result day 3:

A 5x faster tracer.

GLOBAL GAME JAM

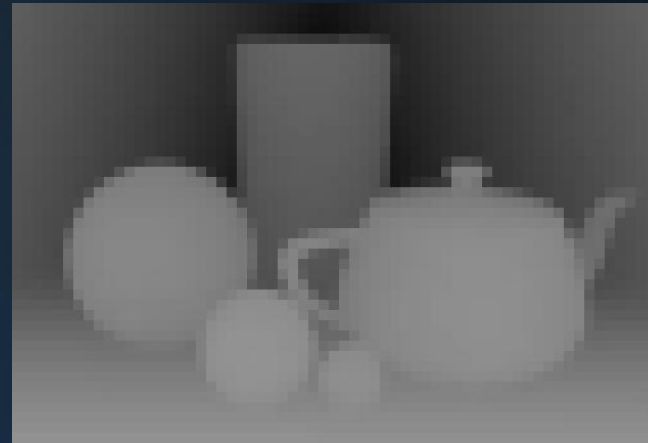
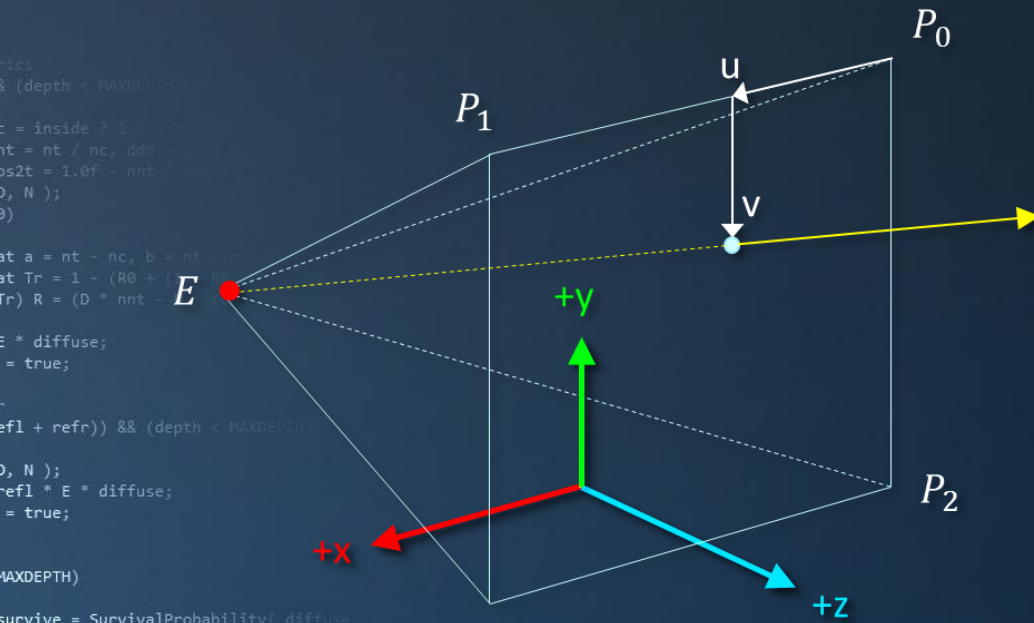


Agenda:

- Current State
- Advanced Whitted
- Generic Ray Queries
- Faster Whitted



Whitted – Bare Necessities



float3* screen = ...;

void NearestIntersection(...);

bool IsOccluded(...);

Plane: $P \cdot \vec{N} + d = 0$

Ray: $P(t) = O + t\vec{D}$

Substituting for $P(t)$, we get: $t = -(O \cdot \vec{N} + d) / (\vec{D} \cdot \vec{N})$

```

survive = SurvivalProbability( diffuse );
estimation - doing it properly, closely following the
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 the
ive)
;
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;

```

```

bool RayIntersectsTriangle(vec3 rayOrigin, vec3 rayVector, Triangle* tri, vec3* intersection)
{
    const float EPSILON = 0.000001;
    vec3 v0 = tri->vertices[0];
    vec3 v1 = tri->vertices[1];
    vec3 v2 = tri->vertices[2];
    vec3 e0 = v1 - v0;
    vec3 e1 = v2 - v0;
    float f0 = dot(e0, e0);
    float f1 = dot(e1, e1);
    float f2 = dot(e0, e1);
    float s = dot(rayVector, e0);
    float t = dot(rayVector, e1);
    float D = f0 * f1 - f2 * f2;
    if (D < 0) return false;
    float invD = 1 / D;
    float u = (f1 * s - f2 * t) * invD;
    if (u < 0 || u > 1) return false;
    float v = (f2 * s + f0 * t) * invD;
    if (v < 0 || v > 1) return false;
    float tval = dot(rayOrigin, e0) * s + dot(rayOrigin, e1) * t + dot(rayOrigin, e0 * e1 - e1 * e0) * f2 * invD;
    if (tval < 0 || tval > 1) return false;
    *intersection = rayOrigin + rayVector * tval;
    return true;
}

```

```

void Sphere::IntersectSphere( Ray ray )
{
    vec3 C = this->pos - ray.O;
    float t = dot( C, ray.D );
    vec3 Q = C - t * ray.D;
    float p2 = dot( Q, Q );
    if (p2 > sphere.r2) return; // r2 = r * r
    t -= sqrt( sphere.r2 - p2 );
    if ((t < ray.t) && (t > 0)) ray.t = t;
}

```



Whitted – Intermediate

Shadows

Direct illumination: $\min d \frac{1}{dist^2}$ and $N \cdot L$

Prevent shadow acne

Reflections

Normals, reflection vector, recursion, recursion cap

No shadow rays for reflection

Prevent $t = 0$ reflections

Materials

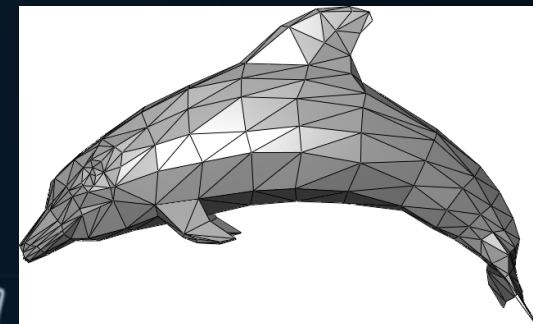
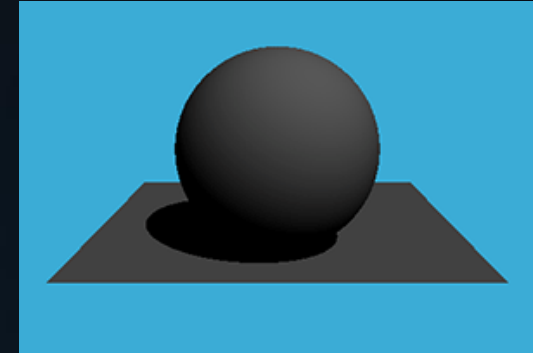
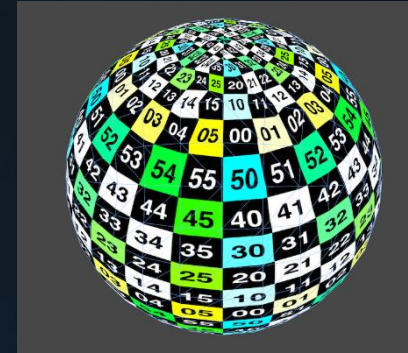
Blending reflection & ‘Lambert’

Textures, texture coordinates (detail textures, specular map, ...)

Primitives

You may want to limit yourself to triangles –

Supporting multiple primitives is a pain.



Whitted – Advanced

Transmission

Snell's law, Fresnel equations, Beer's law

Soft shadows

'Cook-style', stochastic, stratification, blue noise, ...

Glossy, anti-aliasing

```
...
& (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);
    (R) R = (D * nnt - N * (ddn
    E * diffuse;
    = true;
    refl + refr)) && (depth < MAXDEPTH)
    ...
    D, N );
    refl * E * diffuse;
    = true;
    MAXDEPTH)
    survive = SurvivalProbability(
    estimation - doing it properly
    if;
    radiance = SampleLight( &rand,
    e.x + radiance.y + radiance.z)
    w = true;
    at brdfPdf = EvaluateDiffuse(
    at3 factor = diffuse * INVPI;
    at weight = Mis2( directPdf, b
    at cosThetaOut = dot( N, L );
    E * ((weight * cosThetaOut) /
    random walk - done properly, closely following the
    survive)
    ;
    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 += (light.color * dot(N,L))/dist2
return sum
```

```
Color Trace( ray r )
I,  $\vec{N}$ , mat = NearestIntersection( scene, r )
if (mat == DIFFUSE)
    return mat.color * DirectIllumination( I,  $\vec{N}$  )
if (mat == MIRROR)
    return mat.color * Trace( I, reflect(  $r.\vec{D}$ ,  $\vec{N}$  ) )
```



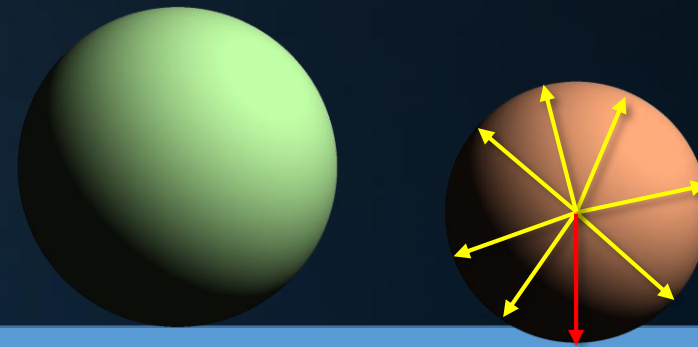
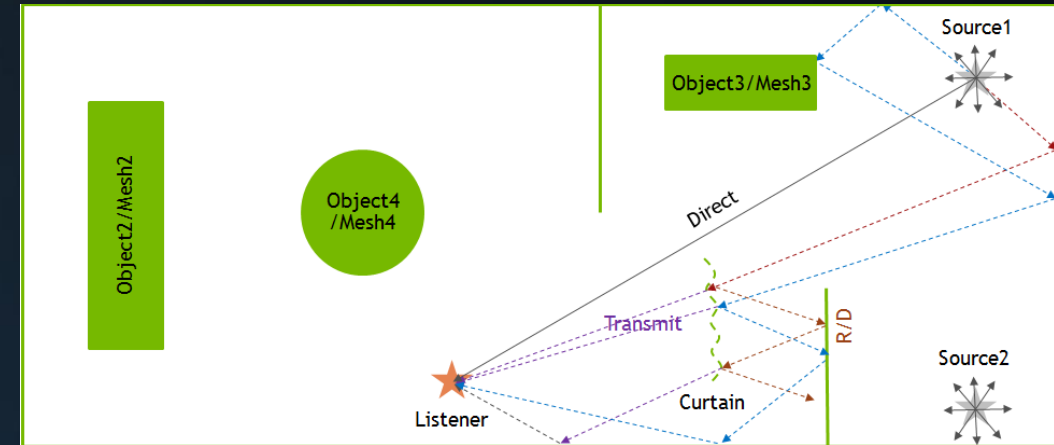
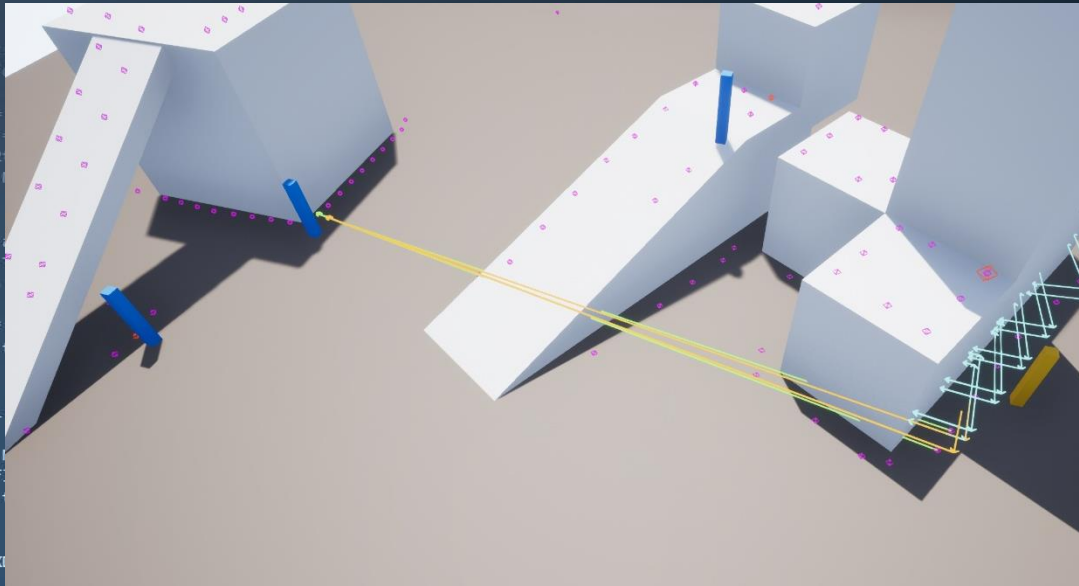
Agenda:

- Current State
- Advanced Whitted
- Generic Ray Queries
- Faster Whitted

```
ics
& (depth < MAXDEPTH))
{
    if (nt < 0)
        nt = inside ? 1 : -1;
    nt = nt / nc; ddn = dot(N, N);
    float r0 = 0.25, r1 = 0.5, r2 = 0.75, r3 = 0.9, r4 = 0.98, r5 = 0.99, r6 = 0.995, r7 = 0.999, r8 = 0.9995, r9 = 0.9999;
    float r = 1.0f - nnt * nnt;
    if (r < r0) return D, N );
    if (r < r1) return D, N );
    if (r < r2) return D, N );
    if (r < r3) return D, N );
    if (r < r4) return D, N );
    if (r < r5) return D, N );
    if (r < r6) return D, N );
    if (r < r7) return D, N );
    if (r < r8) return D, N );
    if (r < r9) return D, N );
    float a = nt - nc, b = nt + nc;
    float Tr = 1 - (R0 + (1 - R0) * r);
    float R = (D * nnt - N * (ddn < 0 ? 1 : -1));
    if (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);
    sion = true;
```



Ray Queries



Make your ray queries accessible outside your engine:

NearestIntersection(...)

IsOccluded(...)

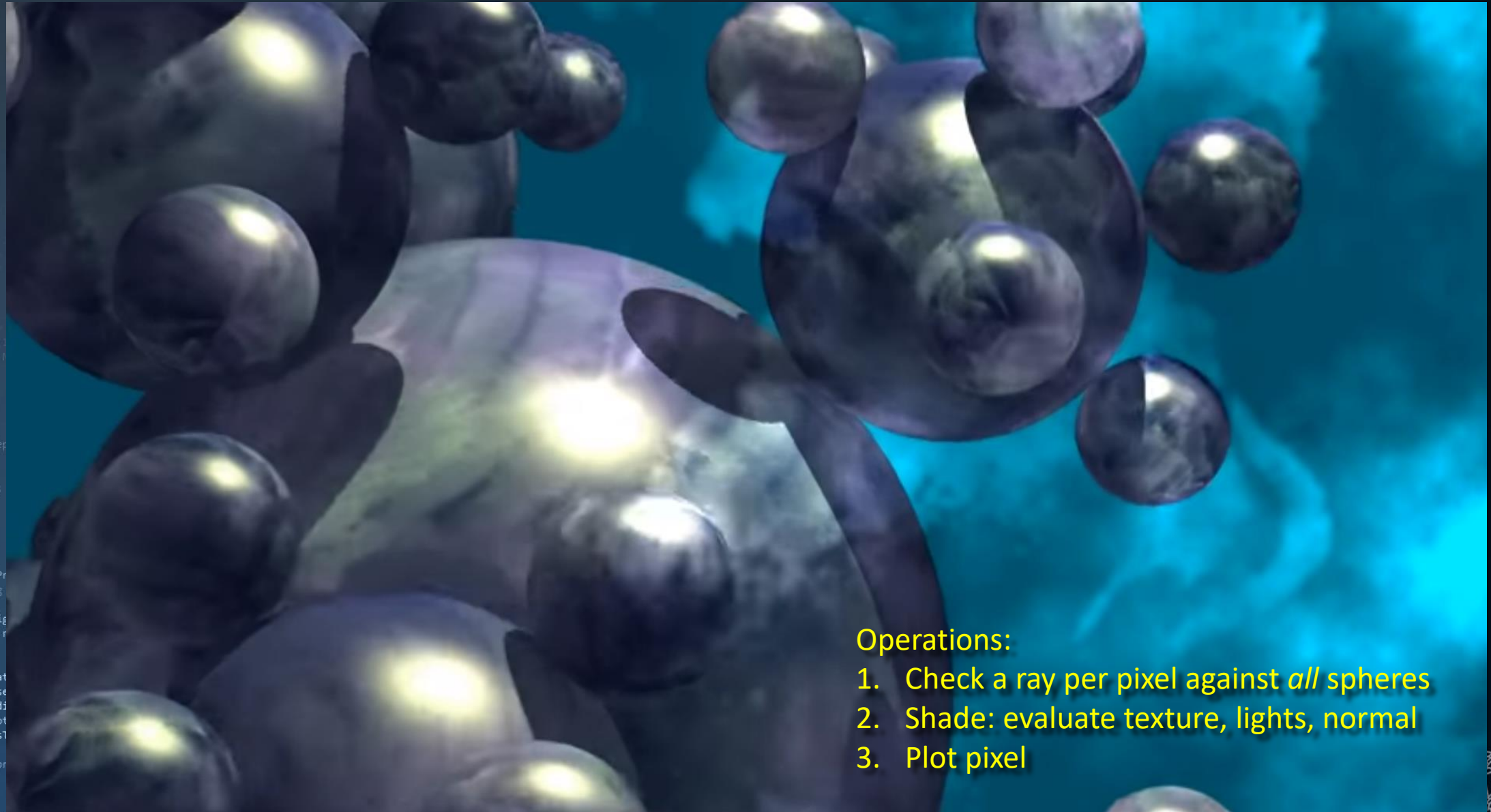


Agenda:

- Current State
- Advanced Whitted
- Generic Ray Queries
- Faster Whitted

```
ics
& (depth < MAXDEPTH))
{
    if (nt < 0)
        nt = inside ? 1 : -1;
    nt = nt / nc; ddn = dot(N, N);
    float r0 = 0.25, r1 = 0.5, r2 = 0.75, r3 = 0.9, r4 = 0.98, r5 = 0.99, r6 = 0.995, r7 = 0.999, r8 = 0.9995, r9 = 0.9999;
    float r = 1.0f - nnt * nnt;
    if (r < r0) return D, N );
    if (r < r1) return D, N );
    if (r < r2) return D, N );
    if (r < r3) return D, N );
    if (r < r4) return D, N );
    if (r < r5) return D, N );
    if (r < r6) return D, N );
    if (r < r7) return D, N );
    if (r < r8) return D, N );
    if (r < r9) return D, N );
    float a = nt - nc, b = nt + nc;
    float Tr = 1 - (R0 + (1 - R0) * r);
    float R = (D * nnt - N * (ddn < 0 ? 1 : -1));
    if (E * diffuse;
    = true;
    -
    refl + refr)) && (depth < MAXDEPTH))
    {
        D, N );
        refl * E * diffuse;
        = true;
    }
    MAXDEPTH)
    survive = SurvivalProbability( diffuse );
    estimation - doing it properly, closely following Small's
    if;
    radiance = SampleLight( &rand, I, &L, &lightDir, &lightPos, &lightType );
    e.x + radiance.y + radiance.z) > 0) && (depth < MAXDEPTH))
    {
        w = true;
        at brdfPdf = EvaluateDiffuse( L, N ) * Psurvive;
        at3 factor = diffuse * INVPI;
        at weight = Mis2( directPdf, brdfPdf );
        at cosThetaOut = dot( N, L );
        E * ((weight * cosThetaOut) / directPdf) * (radiance.x + radiance.y + radiance.z);
    }
    random walk - done properly, closely following Small's
    vive)
    ;
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    sion = true;
}
```





```
ics  
& (depth < MAXDEPTH)  
t = inside ? 1 : -1;  
nt = nt / nc; ddn =  
cos2t = 1.0f - nnt *  
(D, N );  
)  
at a = nt - nc, b =  
at Tr = 1 - (R0 + (1 -  
Tr) R = (D * nnt - t  
E * diffuse;  
= true;  
efl + refr)) && (deg  
(D, N );  
refl * E * diffuse;  
= true;  
MAXDEPTH)  
survive = SurvivalPr  
estimation - doing  
df;  
radiance = SampleLig  
e.x + radiance.y + r  
w = true;  
at brdfPdf = Evaluat  
at3 factor = diffuse  
at weight = Mis2( di  
at cosThetaOut = dot  
E * ((weight * cosT  
andom walk - done pr  
ive)  
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );  
survive;  
pdf;  
n = E * brdf * (dot( N, R ) / pdf);  
ision = true;
```

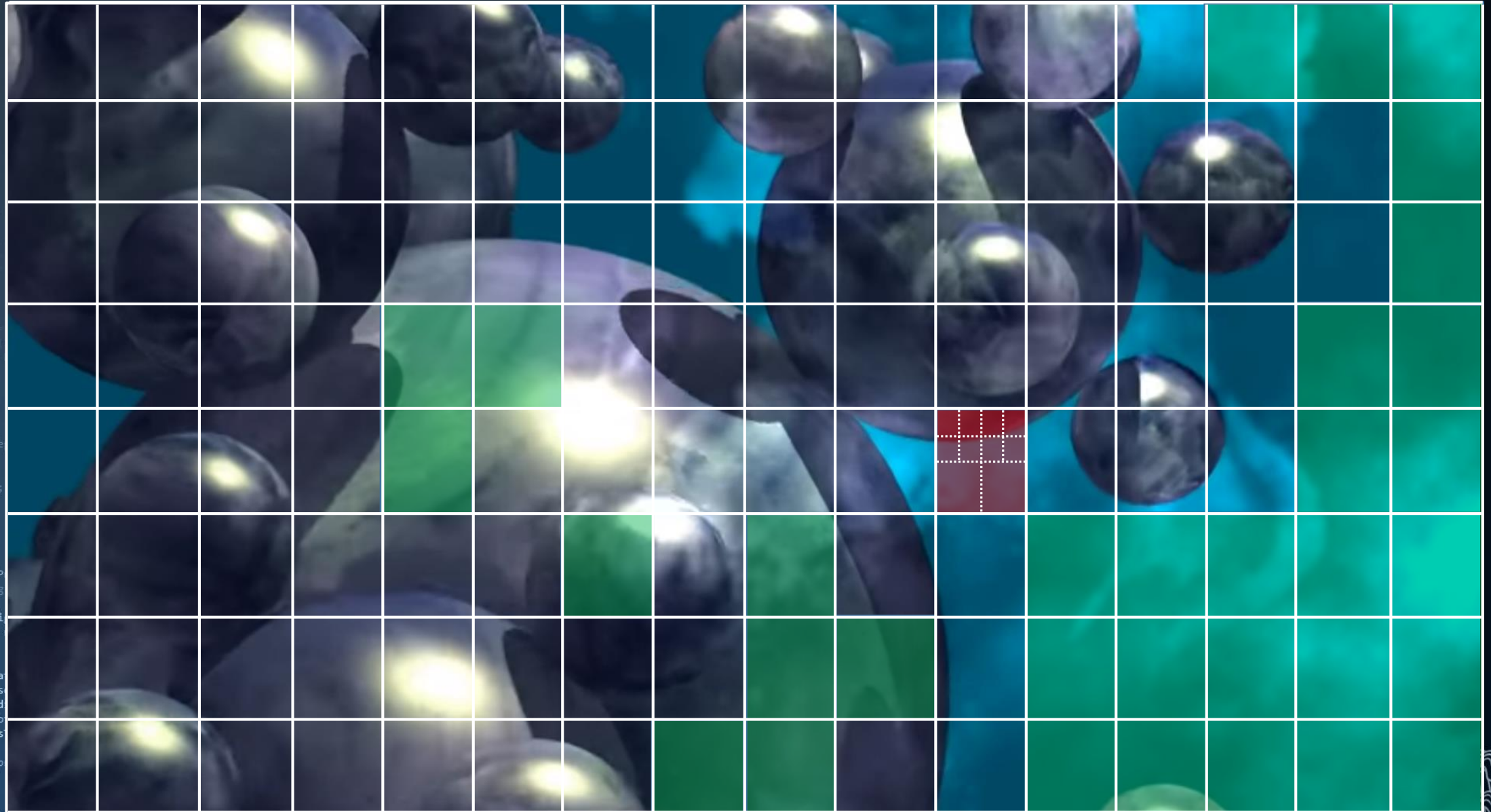
Operations:

1. Check a ray per pixel against *all* spheres
2. Shade: evaluate texture, lights, normal
3. Plot pixel



Ray Tracing for Games

```
ics
& (depth < MAXDEPTH)
{
    if (nt < inside ? 1 : 0)
    {
        nt = nt / nc; ddn =
        cos2t = 1.0f - nnt;
        D, N );
        0);
        at a = nt - nc, b =
        at Tr = 1 - (R0 + 1
        Tr) R = (D * nnt -
        E * diffuse;
        = true;
        -
        refl + refr)) && (de
        D, N );
        refl * E * diffuse;
        = true;
        MAXDEPTH)
        survive = SurvivalP
        estimation - doing
        df;
        radiance = SampleLi
        e.x + radiance.y +
        w = true;
        at brdfPdf = Evalua
        at3 factor = diffus
        at weight = Mis2( d
        at cosThetaOut = do
        E * ((weight * cos
        random walk - done p
        ivate)
    }
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, sppd
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    sion = true;
}
```



Screen-space Subdivision

Quickly bring down the average amount of work per pixel.

Limitations:

- Smallest detail should be larger than a block
- Spheres could be flattened even if much larger than a block



Could we also use this for shadow rays?

```
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(
        estimation - doing it properly, close);
        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 Small's
        (survive)
    }
    {
        at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
        survive;
        pdf;
        n = E * brdf * (dot( N, R ) / pdf);
        sion = true;
    }
}
```



Ray Tracing for Games

```
ics
& (depth < MAXDEPTH)
{
    if (inside) {
        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));

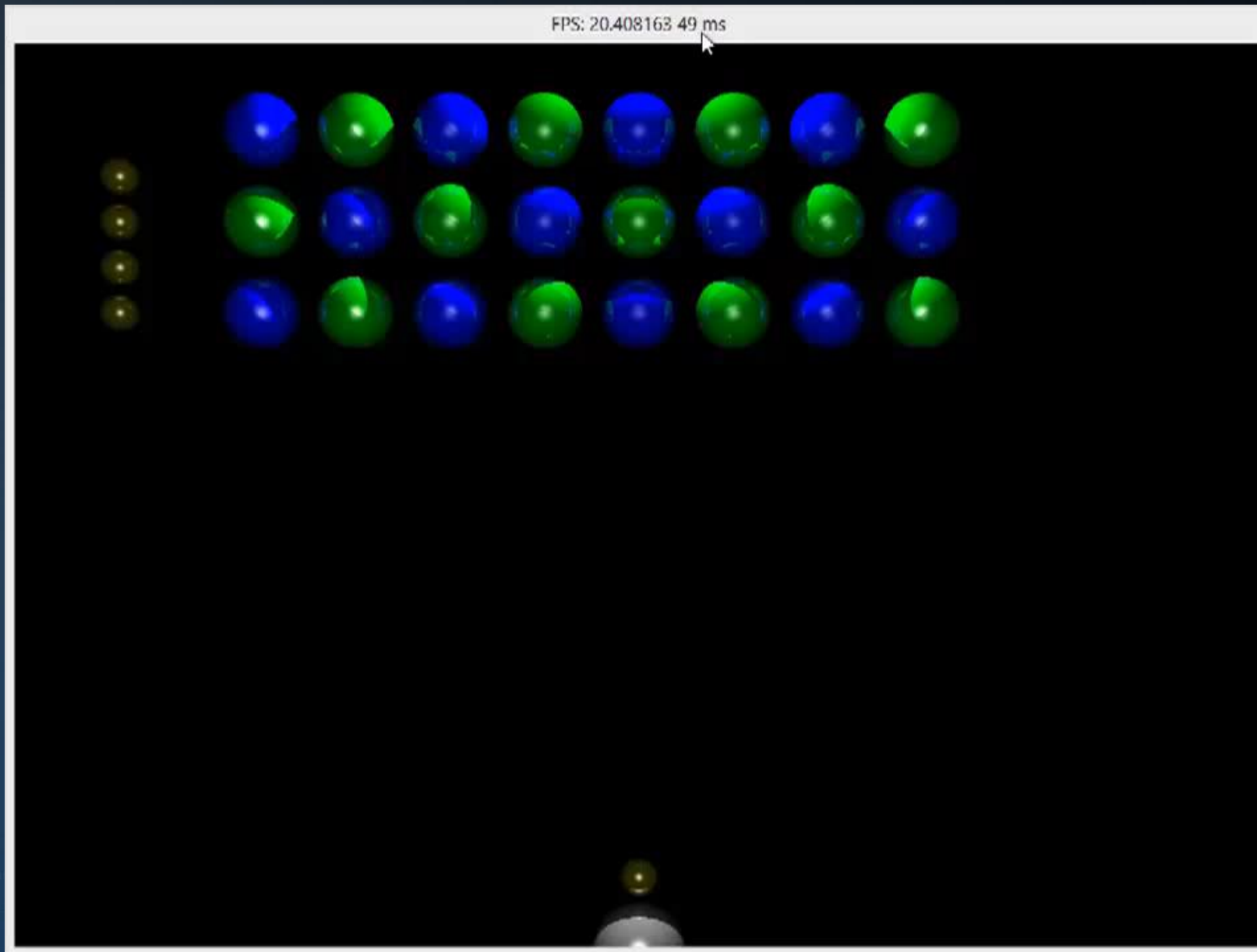
        E * diffuse;
        = true;

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

    MAXDEPTH)

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



Making it Fast before we get to the Cool Stuff™

Rendering at interactive frame rates is important.

Methods:

- Render pixel blocks to speed up rendering; refine when the camera is stationary.
- Multithread. Use OpenMP, tiles of pixels and the dynamic scheduler for now.
- Keep your scene simple.

Ask yourself what you need at the bare minimum to test your game code;
build this functionality first – refine later.



Great product.

Agenda:

- Current State
- Advanced Whitted
- Generic Ray Queries
- Faster Whitted



End of PART 2.

```
ics
& (depth < MAXDEPTH)
{
    if (inside ? 1 : 1.25 * nnt)
    {
        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, &lightDir );
    e.x + radiance.y + radiance.z) > 0) && (depth < MAXDEPTH)
    {
        w = true;
        at brdfPdf = EvaluateDiffuse( L, N ) * Psurvive;
        at3 factor = diffuse * INVPI;
        at weight = Mis2( directPdf, brdfPdf );
        at cosThetaOut = dot( N, L );
        E * ((weight * cosThetaOut) / directPdf) * (radiance
    }
    random walk - done properly, closely following Small's
    (survive)
    ;
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    sion = true;
}
```

