

Ray Tracing for Games

Dr. Jacco Bikker - IGAD/BUAS, Breda, January 29

Welcome!

```
rics
  & (depth < MAXDEPTH)
  c = inside ? 1 : 1.2f;
  nt = nt / ncy ddn = ddn * c;
  os2t = 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
if;
radiance = SampleLight( &rand, I, &L, &lightbuf,
e.x + radiance.y + radiance.z) > 0) && (dot( N
e = true;
at brdfPpdf = EvaluateDiffuse( L, N ) * Psurvive;
at t3 factor = diffuse * INVPI;
at weight = Mis2( directPpdf, brdfPpdf );
at cosThetaOut = dot( N, L );
E * (weight * cosThetaOut) / directPpdf) * (radiance
random walk - done properly, closely following Smits
alive)

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



Agenda:

- Introduction / Goals / Planning
- Template / LH2
- Whitted-style recap
- RT-centric Game Dev



Ray Tracing for Games

Thursday
09:00 – 14:00

Friday
09:00 – 17:00

Monday
09:00 – 17:00

Tuesday
09:00 – 17:00

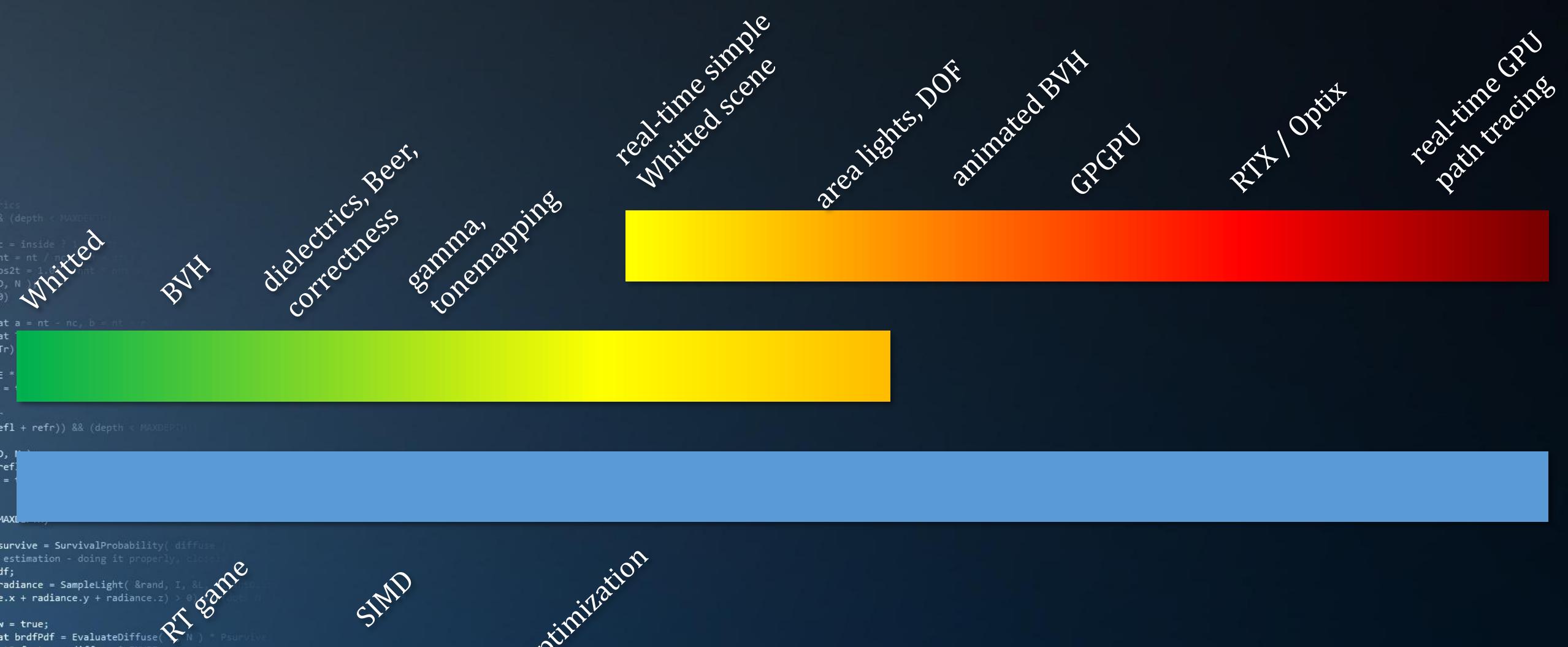
Thursday
09:00 – 17:00

Friday
09:00 – 17:00

Wednesday
13:00 – 17:00

```
rics
k (depth < MAXDEPTH)
c = inside ? 1 : 0;
nt = nt / ncy;
dn = dn / ncy;
es2t = 1.0f - imt;
o, N );
)
at a
at Tr
Tr) R
E * d
= tr
-
refl +
o, N
refl
= tr
MAXDE
survi
esti
df;
adia
e.x +
v = t
at br
at3 f
at we
at co
E *
random walk - done properly, closely following Sma
alive)
;
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf
alive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
ision = true;
```





```
rics  
3 & (depth < MAXDEPTH)  
    if = inside ? 1 : 0;  
    nt = nt / n;  
    os2t = 1.0 / (nt * os2t);  
    if (os2t > N )  
        N = 1;  
  
    if a = nt - nc, b = nt  
    at :  
        E *  
        =  
  
        refl + refr)) && (depth < MAXDEPTH)  
  
        , I, ref  
        = 1  
  
MAX
```

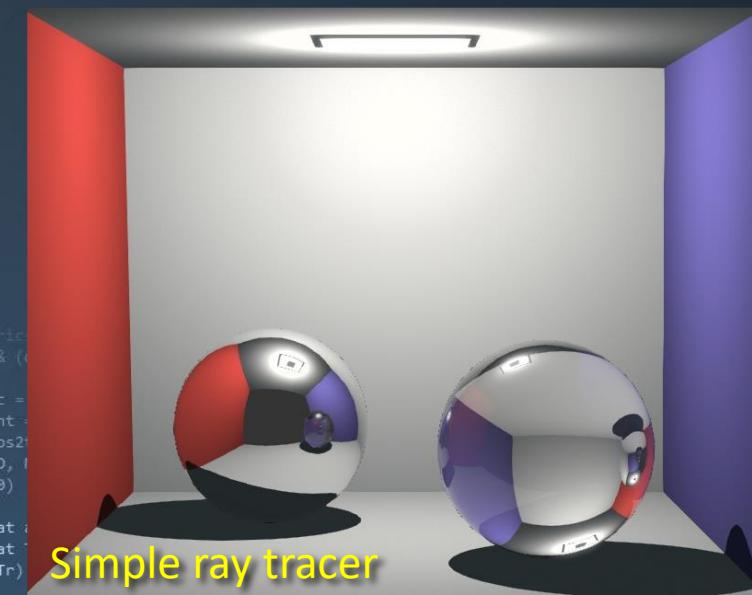
```
survive = SurvivalProbability( diffuse  
estimation - doing it properly, closed  
if;  
radiance = SampleLight( &rand, I, &L, &pdf  
e.x + radiance.y + radiance.z) > 0  
E * (weight * cosThetaOut) / directPdf) * (radiance  
  
random walk - done properly, closely following Smith  
alive)  
  
at t3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf  
survive;  
pdf;  
n = E * brdf * (dot( N, R ) / pdf);  
ision = true;
```

SIMD

optimization



Ray Tracing for Games



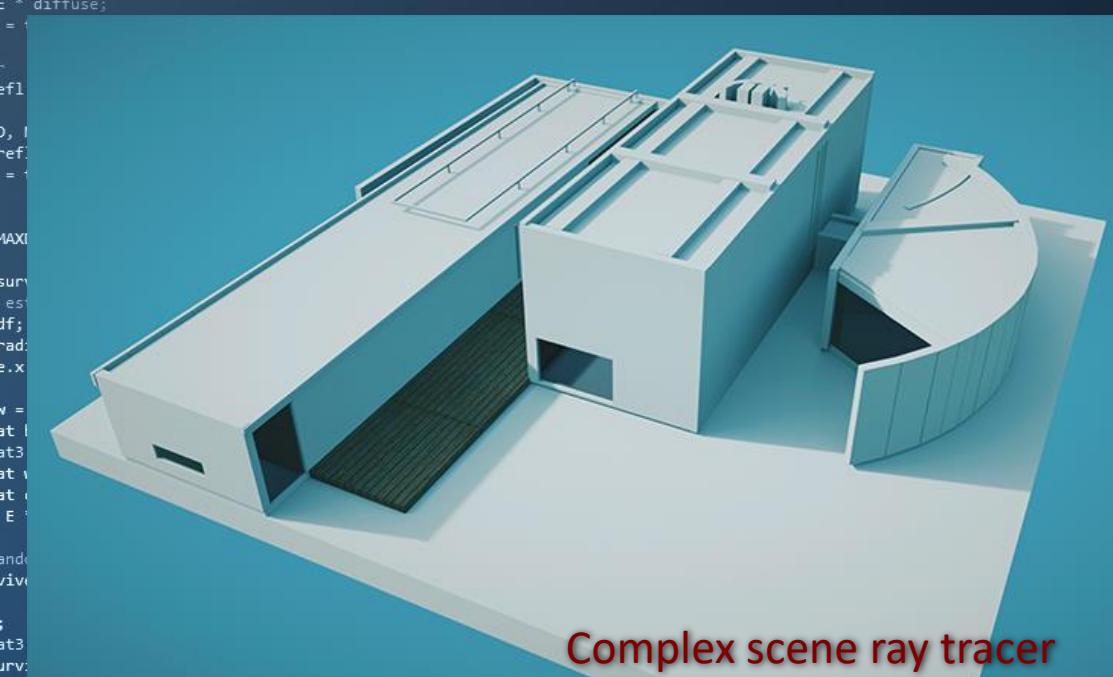
Simple ray tracer



Pretty fast ray tracer



Maxed-out ray tracer

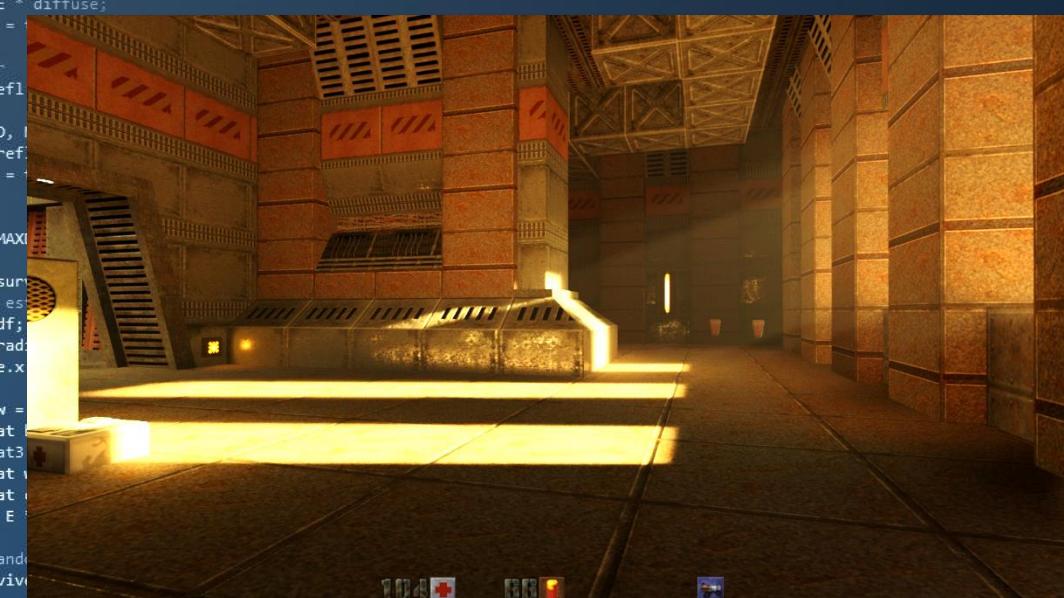
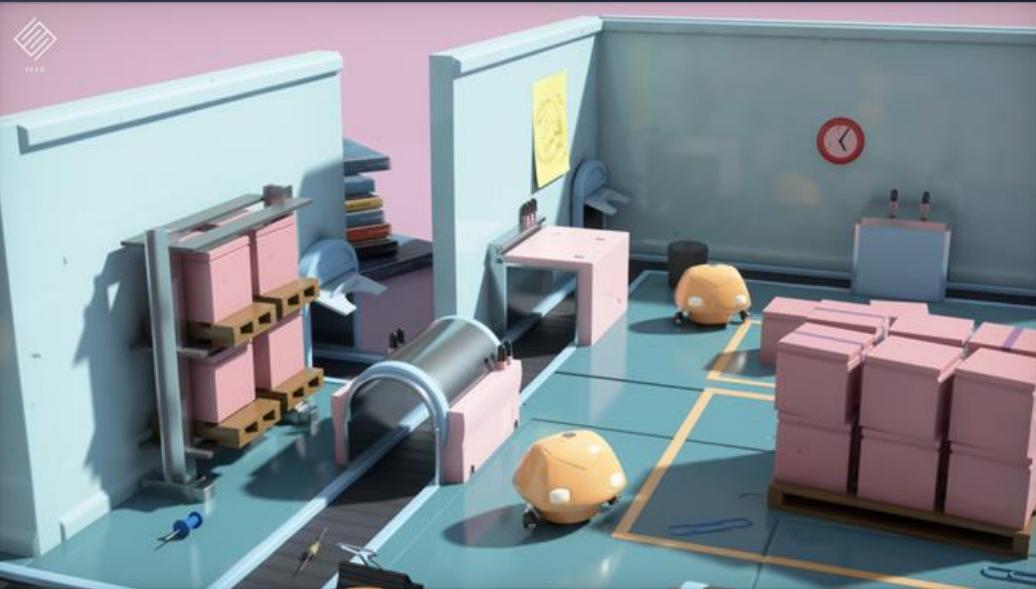


Complex scene ray tracer



Game-ready ray tracer

Ray Tracing for Games



```
E * diffuse;
= 
efl
O,
ref
= 
MAXI
sur
est
df;
ad
ad
e.x
v =
at
at
at
at
E
;
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
survive;
pdf;
in = E * brdf * (dot( N, R ) / pdf);
ision = true;
```

Google famous games that are now open source

All Images News Videos Shopping More Settings Tools

About 336.000.000 results (0,74 seconds)

According to makeuseof.com View 10+ more

0 A.D. Alien Arena Armagetron Advanced The Battle for Wesnoth Dungeon Crawl Stone Soup FreeCiv Freedom

Games are listed in alphabetical order.

- 0 A.D. Available for Windows, Mac, and Linux. ...
- Alien Arena. Available for Windows and Linux. ...
- Armagetron Advanced. Available for Windows, Mac, and Linux. ...
- Battle for Wesnoth. Available for Windows, Mac, Linux, and Android. ...
- Dungeon Crawl Stone Soup. ...
- FreeCiv. ...
- Freedom. ...
- Hedgewars. ...

More items... • Aug 14, 2017

20 Best Open Source Video Games - MakeUseOf
<https://www.makeuseof.com/tag/open-source-video-games>

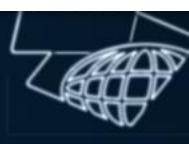
<https://www.slant.co/topics/1933/~best-open-source-games>

People also ask

Are any video games public domain?

Is Doom public domain?

Where can I download Linux games?



Ray Tracing for Games





Wednesday
13:00 – 17:00

course intro
LH2
template
Whitted
refactoring
RT-centric games

LAB 1

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

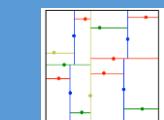
End result day 3:
A 5x faster tracer.

GLOBAL GAME JAM

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



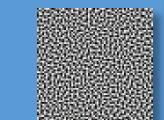
LAB 8

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.

Agenda:

- Introduction / Goals / Planning
- Template / LH2
- Whitted-style recap
- RT-centric Game Dev



1. No Foundation



2. Template



3. Lighthouse 2



Lighthouse 2

Application

- Opens an OpenGL window
 - Prepares the scene
 - Handles user input
 - Calls 'Render'
- (see provided samples)

RenderSystem

- Loads 3D scene data
- Owns & maintains the scene
- Handles animation
- Synchronizes scene data with a 'Core'
- Invokes a 'Core'

API overview in:
API → [render_api.h](#)

You should not have to change anything in the RenderSystem.

Core

- Receives raw triangle data from RenderSystem
 - Performs actual rendering
- Several are available:
- Optix7, OptixPrime & Vulkan ray tracing cores;
 - A software rasterizer.



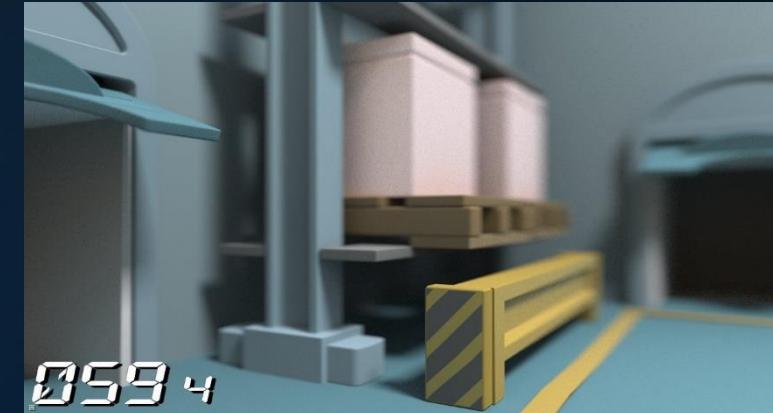
Lighthouse 2

Lighthouse 2

The Lighthouse 2 system has been designed to facilitate the development of rendering software based on ray tracing.

The RenderSystem takes care of the boring things, such as model loading and scene management. You get to concentrate on the actual ray tracing.

The current system needs a high-end NVIDIA device for pretty images. Without a recent GPU, only a software rasterizer fallback is available. This is where you come in.



Application

- Opens an OpenGL window
 - Prepares the scene
 - Handles user input
 - Calls 'Render'
- (several samples provided)

RenderSystem

- Loads 3D scene data
- Owns & maintains the scene
- Handles animation
- Synchronizes scene data with a 'Core'
- Invokes a 'Core'

API overview in:
[API → render_api.h](#)

You should not have to change anything in the RenderSystem.

Core

- Receives raw triangle data from RenderSystem
 - Performs actual rendering
- Several are available:
- Optix7, OptixPrime & Vulkan ray tracing cores;
 - A software rasterizer.

Lighthouse 2

RenderCore

A core implements the low-level rendering functionality.

To build your own: start by copying an existing one.
(project folder → docs → cloning a core.docx)

A core receives raw geometry from the RenderSystem:

```
void RenderCore::SetGeometry(
    const int meshIdx,
    const float4* vertexData,
    const int vertexCount,
    const int triangleCount,
    const CoreTri* triangles,
    const uint* alphaFlags
```

Note:

The core is expected to copy the data it receives. There is no guarantee at all that the RenderSystem leaves the provided data intact after SetGeometry returns.

float4: x, y, z, padding

**also contain vertices. raw
verts are for intersection;
tris are for shading.**

**RenderSystem makes
CoreTris from HostTris,
CoreMaterials from HostMaterials
and so on.**

Lighthouse 2

RenderCore

A core implements the low-level rendering functionality.

A core receives instances from the RenderSystem:

```
void RenderCore::SetInstance(
    const int instanceIdx,
    const int meshIdx,
    const mat4& matrix
)
```

Note:

- A mesh that is not instanced is not visible.
- SetInstance always receives a flattened list of instances; there is no hierarchy.



Lighthouse 2

RenderCore

A core implements the low-level rendering functionality.

A core receives a Render command from the RenderSystem:

```
void RenderCore::Render(
    const ViewPyramid& view,
    const Convergence converge,
    const bool async
```

}
{
}

Note:

- Converge will be true when the camera is stationary. This will be useful later for path tracing.
- Async can be ignored for now.



Lighthouse 2

```

void Init();
CoreStats GetCoreStats();
void SetProbePos( int2 pos );
void SetTarget( GLTexture* target, uint spp );
void Setting( char* name, float value );
void Render( ViewPyramid& view, Convergence converge, float brightness, float contrast );
void Shutdown();
void SetTextures( CoreTexDesc* tex, int textureCount );
void SetMaterials( CoreMaterial* mat, CoreMaterialEx* matEx, int materialCount );
void SetLights( CoreLightTri* areaLights, int areaLightCount,
               CorePointLight* pointLights, int pointLightCount,
               CoreSpotLight* spotLights, int spotLightCount,
               CoreDirectionalLight* directionalLights, int directionalLightCount );
void SetSkyData( float3* pixels, uint width, uint height );
void SetGeometry( int meshIdx, float4* vertexData, int vertexCount,
                  int triangleCount, CoreTri* triangles, uint* alphaFlags = 0 );
void SetInstance( int instanceIdx, int modelIdx, mat4& transform );
void UpdateToplevel();

random walk - done properly, closely following Smith's
survive);

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

```



Agenda:

- Introduction / Goals / Planning
- Template / LH2
- Whitted-style recap
- RT-centric Game Dev



Ray Tracing

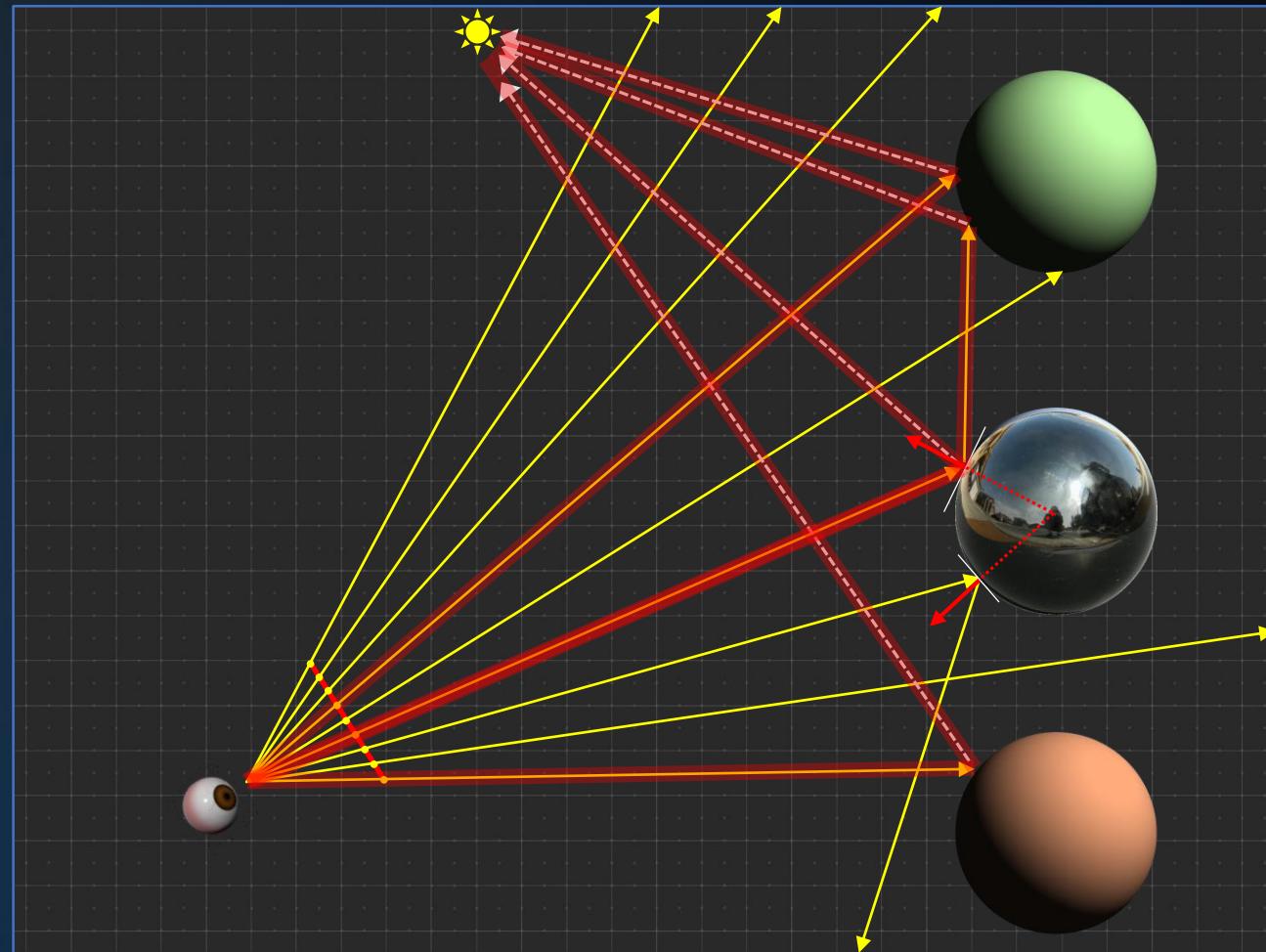
World space

- Geometry
- Eye
- Screen plane
- Screen pixels
- Primary rays
- Intersections
- Point light
- Shadow rays

Light transport

- Extension rays

Light transport



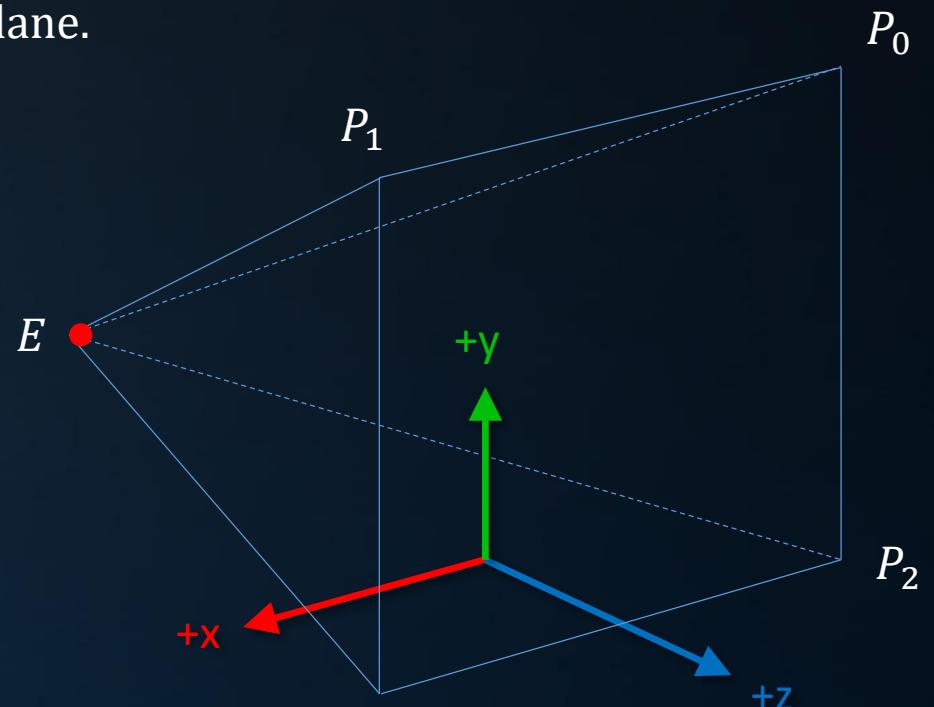
Ray setup

A ray is initially shot through a pixel on the screen plane.
The screen plane is defined in *world space*, e.g.:

$$\begin{aligned} \text{Camera position: } E &= (0,0,0) \\ \text{View direction: } \vec{V} &= (0,0,1) \\ \text{Screen center: } C &= E + d\vec{V} \\ \text{Screen corners: } P_0 &= C + (-1,1,0), \\ &P_1 = C + (1,1,0), \\ &P_2 = C + (-1,-1,0) \end{aligned}$$

From here:

- Change FOV by altering d ;
- Transform camera by multiplying E, P_0, P_1, P_2 with the camera matrix.



Ray setup

do not normalize!

Point on the screen:

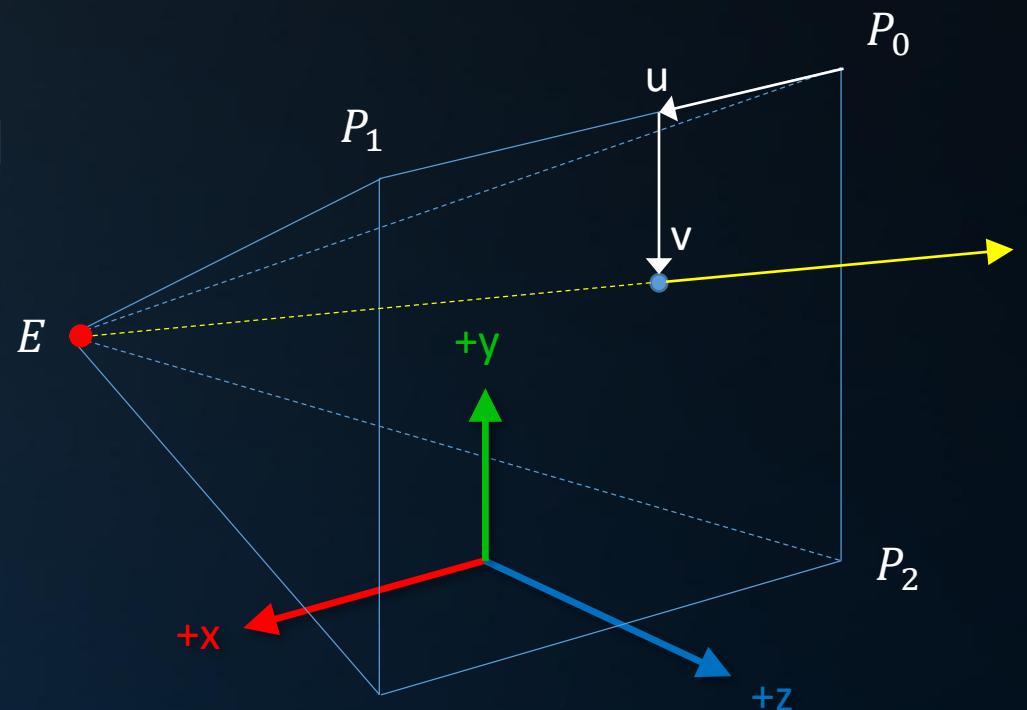
$$P(u, v) = P_0 + u(P_1 - P_0) + v(P_2 - P_0), u, v \in [0,1]$$

Ray direction (normalized):

$$\vec{D} = \frac{P(u, v) - E}{\| P(u, v) - E \|}$$

Ray origin:

$$O = E$$



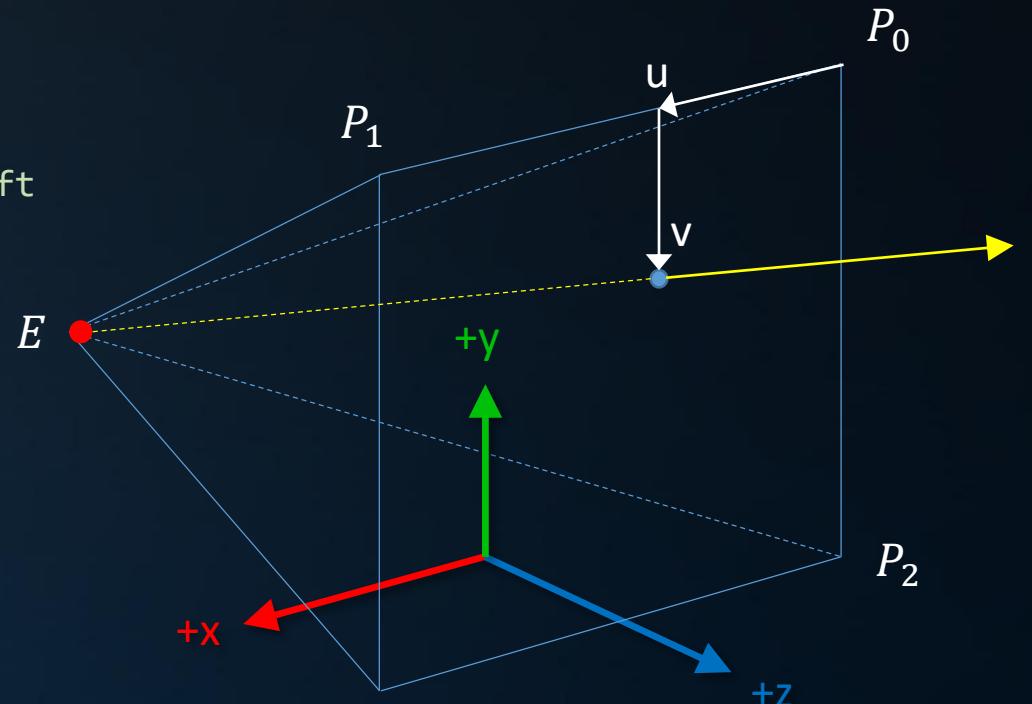
Ray setup

```

rics
    & (depth < MAXDEPTH)
    = inside ? 1 : 1.2;
    nt = nt / nc; ddn = ddn / nc;
    os2t = 1.0f - nt * nc;
    D = (D * nnt + N * fdd);
    R = (R0 + (1 - R0) * Tr) * R = (D * nnt + N * fdd);
    E = T * vec3( 0, 0, 0 );
    right = p1 - p0;
    down = p2 - p0;
    for( int y = 0; y < 512; y++ )
    {
        for( int x = 0; x < 512; x++ )
        {
            float u = x / 512.f;
            float v = y / 512.f;
            vec3 P = p0 + u * right + v * down;
            vec3 D = normalize( P - E );
            vec3 c = Trace( Ray( E, D ) );
            // TODO: visualize c somehow
        }
    }
}

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

```



Ray setup

```

rics
    & (depth < MAXDEPTH)
    c = inside ? 1 : 1.2;
    nt = nc / ncy; ddn = nc * nt;
    os2t = 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;

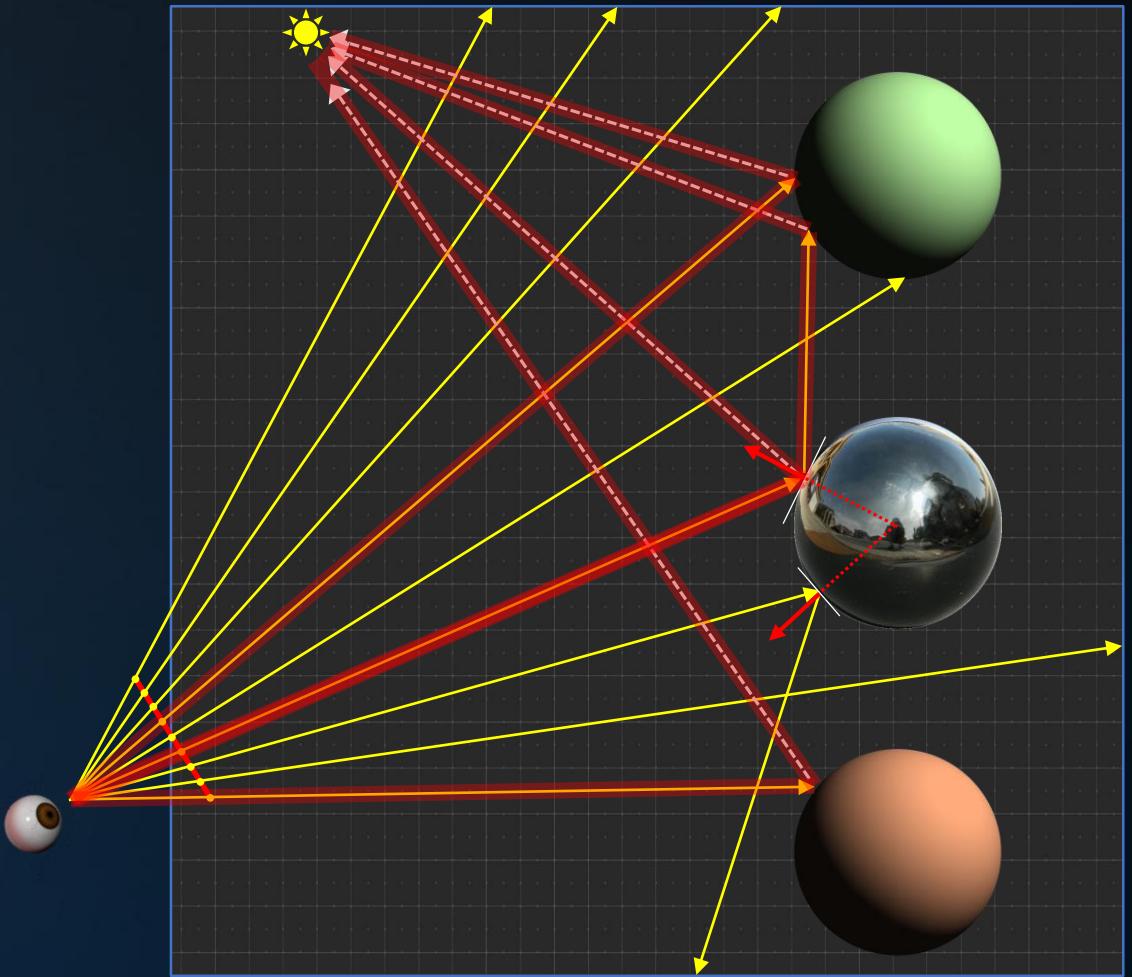
    refl + refr)) && (depth < MAXDEPTH);
    D, N );
    refl * E * diffuse;
    = true;

MAXDEPTH)

survive = SurvivalProbability( diffuse );
estimation = doing it properly, closely
if;
radiance = SampleLight( &rand, I, &L, &lightbase );
e.x + radiance.y + radiance.z ) > 0 && (dot( N
    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 ) * (radi
        alive);
    }

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

```



Trace

Needful things:

Ray/plane intersection:

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

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

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

Ray/sphere intersection:

```
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;
}
```

Ray/triangle intersection:

Adapt from [Moller-Trumbore](#), via [Wikipedia](#):

```
bool RayIntersectsTriangle(vec3 rayOrigin, vec3 rayVector, Triangle* tri, vec3& intersection)
{
    const float EPSILON = 0.000001;
    vec3 vertex0 = inTriangle->vertex0;
    vec3 vertex1 = inTriangle->vertex1;
    vec3 vertex2 = inTriangle->vertex2;
    vec3 edge1, edge2, h, s, q;
    float a,f,u,v;
    edge1 = vertex1 - vertex0, edge2 = vertex2 - vertex0;
    h = rayVector.crossProduct(edge2);
    a = edge1.dotProduct(h);
    if (a > -EPSILON && a < EPSILON) return false; // ray is parallel to this triangle
    f = 1.0/a, s = rayOrigin - vertex0, u = f * s.dotProduct(h);
    if (u < 0.0 || u > 1.0) return false; // miss
    q = s.crossProduct(edge1), v = f * rayVector.dotProduct(q);
    if (v < 0.0 || u + v > 1.0) return false; // miss
    float t = f * edge2.dotProduct(q);
    if (t > EPSILON && t < 1/EPSILON) // ray intersection
    {
        intersection = rayOrigin + rayVector * t;
        return true;
    }
    else return false; // miss
}
```



Ray setup

```

rics
    & (depth < MAXDEPTH)
    c = inside ? 1 : 1.2;
    nt = nc / ncy;
    os2t = 1.0f - nnt * os2;
    D, N );
}

at a = nt - nc, b = nt + nc;
at Tr = 1 - (R0 + (1 - R0) * Tr);
Tr) R = (D * nnt - N * (dd
E * diffuse;
= true;

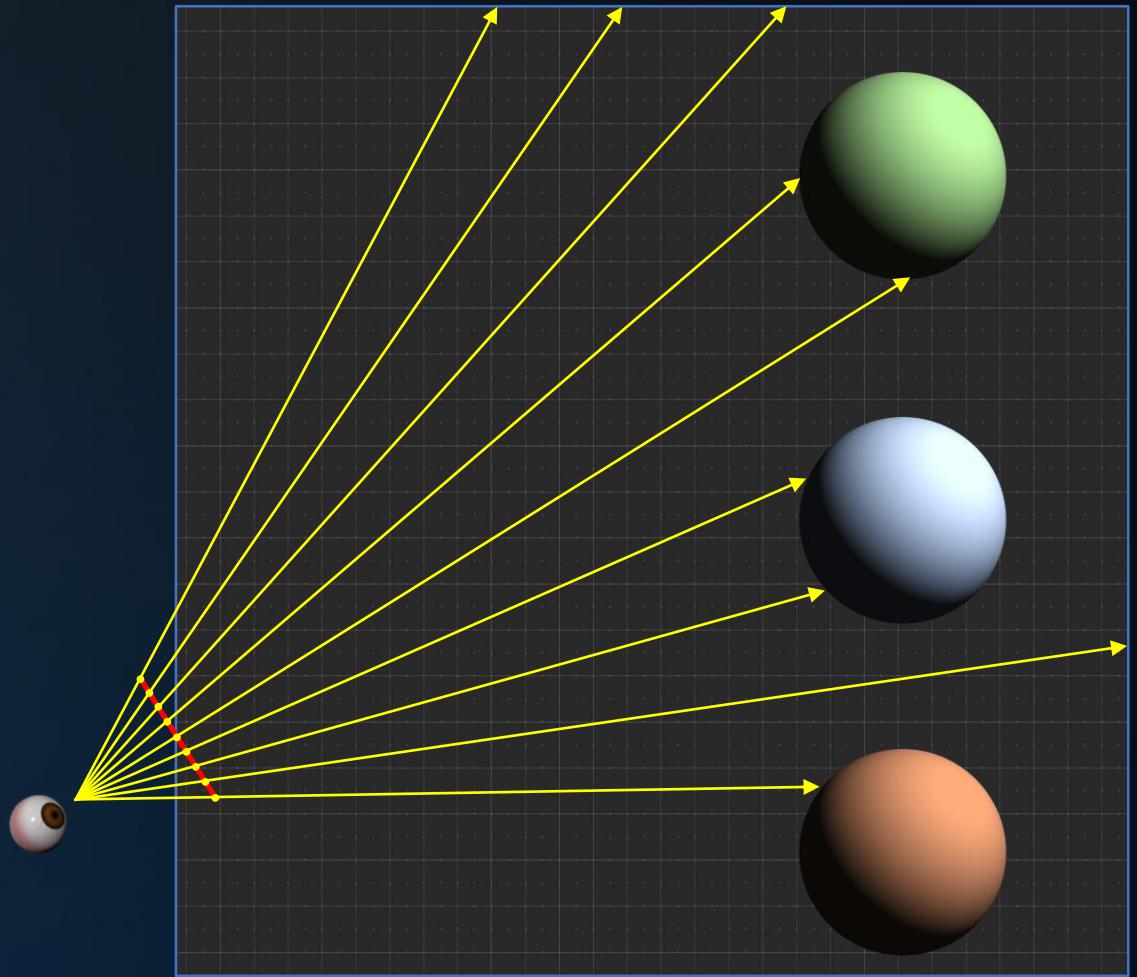
- refl + refr)) && (depth < MAXDEPTH);
D, N );
refl * E * diffuse;
= true;

MAXDEPTH)

survive = SurvivalProbability( diffuse,
estimation - doing it properly, closely
if;
radiance = SampleLight( &rand, I_s, &L, &lightbase,
e.x + radiance.y + radiance.z) > 0) && (dot( N
N * true;
at brdfPpdf = EvaluateDiffuse( L, N ) * Psurvive;
at3 factor = diffuse * INVPI;
at weight = Mis2( directPpdf, brdfPpdf );
at cosThetaOut = dot( N, L );
E * (weight * cosThetaOut) / directPpdf) * (radi
survive)
};

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

```



Whitted-style Ray Tracing

```
vec3 Trace( const Ray& r )
{
    I, N, material = NearestIntersection( scene, r );
    return material.color * DirectIllumination( I, N );
}
```

Direct illumination:

Summed contribution of *unoccluded* point light sources, taking into account:

- Distance to I
- Angle between \vec{L} and \vec{N}
- Intensity of light source

Note that this requires a ray per light source.



Whitted-style Ray Tracing

```

vec3 Trace( const Ray& r )
{
    I, N, material = NearestIntersection( scene, r );
    if (material.IsDiffuse())
        return material.color * DirectIllumination( I, N );
    else
        return material.color * Trace( ... );
}

```

Reflection:

$$\vec{R} = \vec{D} - 2(\vec{D} \cdot \vec{N})\vec{N}$$

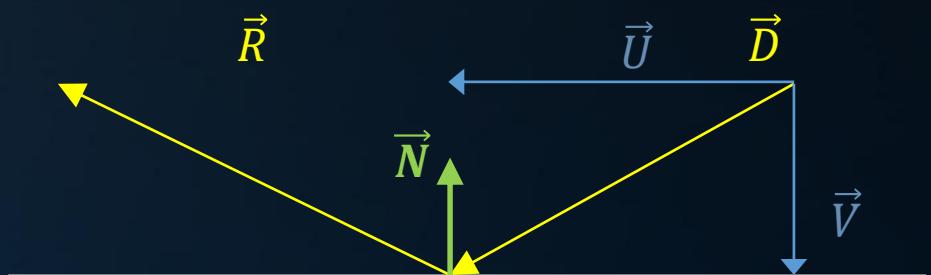
$$\vec{V} = \vec{N}(\vec{D} \cdot \vec{N})$$

$$\vec{U} = \vec{D} - \vec{V}$$

$$\vec{R} = \vec{U} + (-\vec{V})$$

$$\vec{R} = \vec{D} - \vec{N}(\vec{D} \cdot \vec{N}) - \vec{N}(\vec{D} \cdot \vec{N})$$

$$\vec{R} = \vec{D} - 2(\vec{D} \cdot \vec{N})\vec{N}$$



Dielectrics

The direction of the transmitted vector \vec{T} depends on the refraction indices n_1, n_2 of the media separated by the surface. According to Snell's Law:

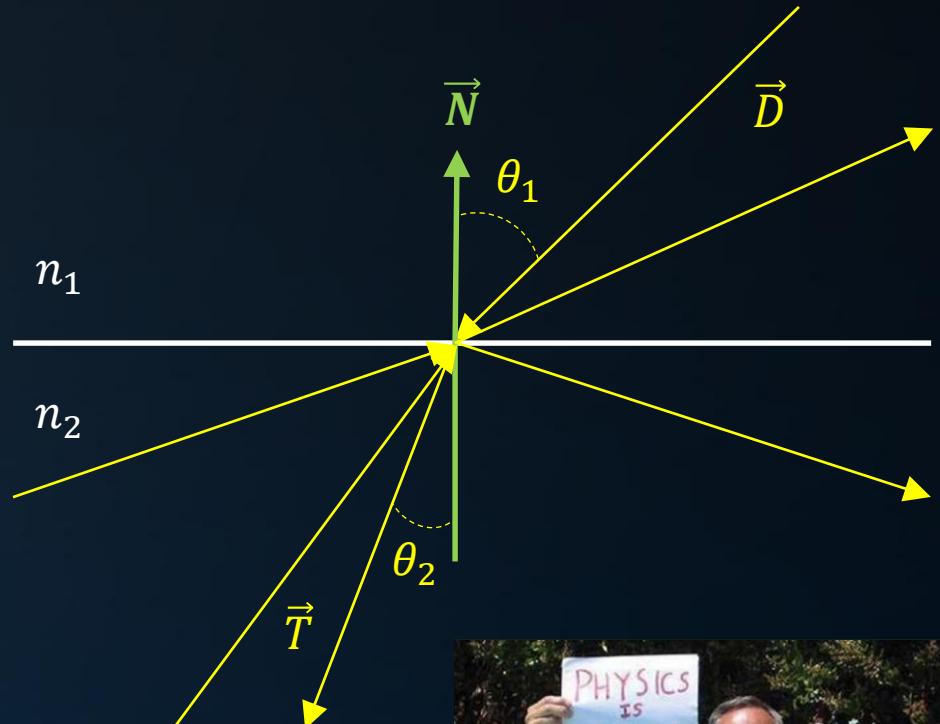
$$n_1 \sin \theta_1 = n_2 \sin \theta_2$$

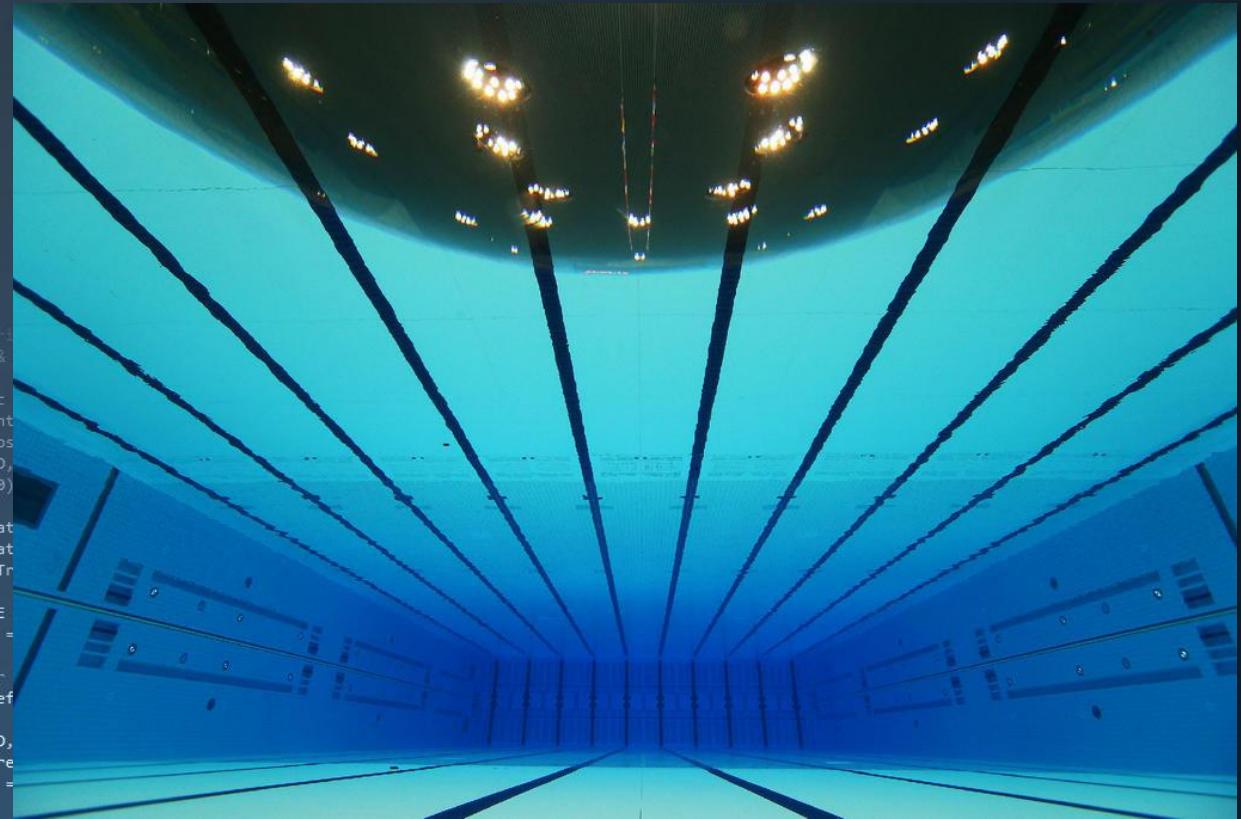
or

$$\frac{n_1}{n_2} \sin \theta_1 = \sin \theta_2$$

Note: left term may exceed 1, in which case θ_2 cannot be computed. Therefore:

$$\frac{n_1}{n_2} \sin \theta_1 = \sin \theta_2 \Leftrightarrow \sin \theta_1 \leq \frac{n_2}{n_1} \rightarrow \theta_{critical} = \arcsin \left(\frac{n_2}{n_1} \sin \theta_2 \right)$$

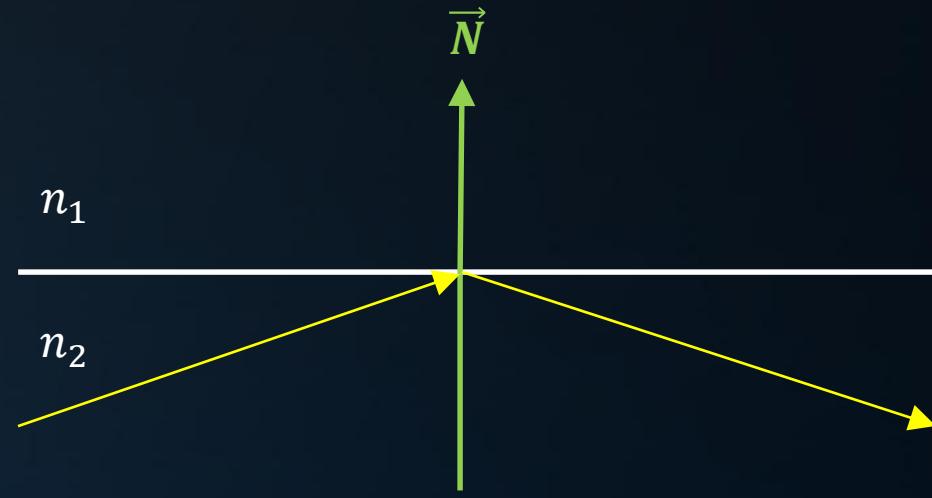




```
MAXDEPTH)
survive = SurvivalProbability( diffuse );
estimation = doing it properly, closely
df;
radiance = SampleLight( &rand, I, &L, &light );
if( e.x + radiance.y + radiance.z ) > 0 && (dot( N, L ) >
```

Note: left term may exceed 1, in which case θ_2 cannot be computed. Therefore:

$$\frac{n_1}{n_2} \sin \theta_1 = \sin \theta_2 \Leftrightarrow \sin \theta_1 \leq \frac{n_2}{n_1} \rightarrow \theta_{critical} = \arcsin \left(\frac{n_2}{n_1} \sin \theta_2 \right)$$



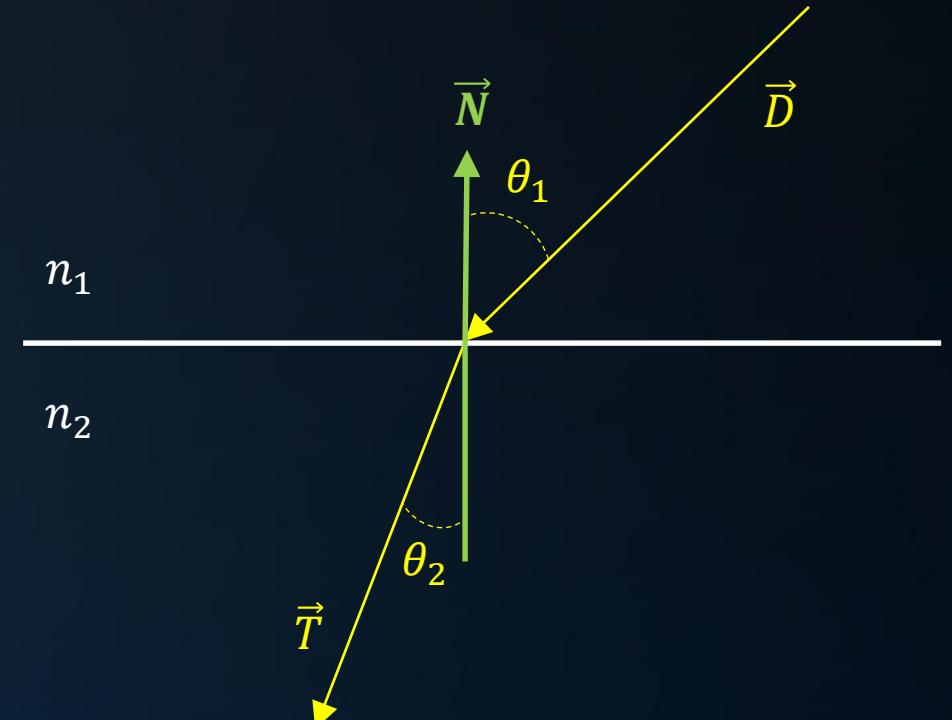
Dielectrics

$$\frac{n_1}{n_2} \sin\theta_1 = \sin\theta_2 \Leftrightarrow \sin\theta_1 \leq \frac{n^2}{n_1}$$

$$k = 1 - \left(\frac{n_1}{n_2} \right)^2 (1 - \cos\theta_1^2)$$

$$\vec{T} = \begin{cases} TIR, & \text{for } k < 0 \\ \frac{n_1}{n_2} \vec{D} + \vec{N} \left(\frac{n_1}{n_2} \cos\theta_1 - \sqrt{k} \right), & \text{for } k \geq 0 \end{cases}$$

Note: $\cos\theta_1 = \vec{N} \cdot -\vec{D}$, and $\frac{n_1}{n_2}$ should be calculated only once.



* For a full derivation, see http://www.flipcode.com/archives/reflection_transmission.pdf



Dielectrics

A typical dielectric transmits *and* reflects light.

```
rics
    & (depth < MAXDEPTH)
    c = inside ? 1 : 1.25;
    nt = nc / ncy; ddn = ddc * c;
    os2t = 1.0f - nnt * nnt;
    D, N );
}
}

at a = nt - nc, b = nt + nc;
at Tr = 1 - (R0 + (1 - R0) *
Tr) R = (D * nnt - N * fdd);
E * diffuse;
= true;

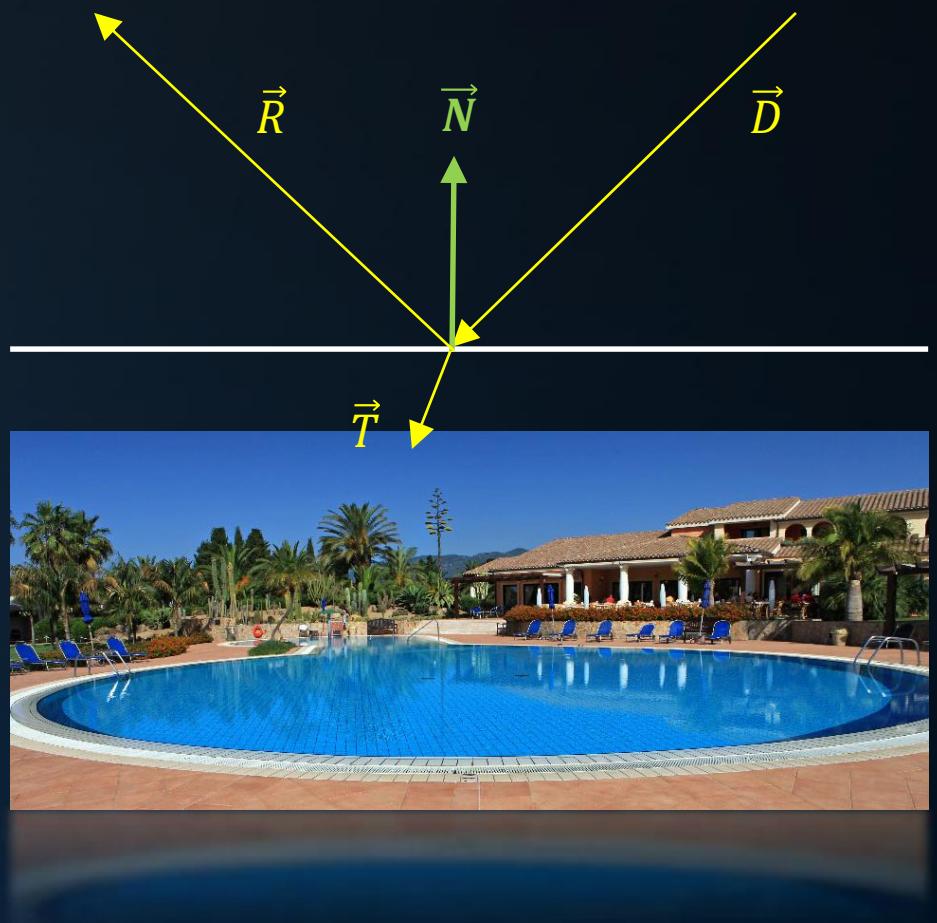
    refl + refr)) && (depth < MAXDEPTH)

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

MAXDEPTH)

survive = SurvivalProbability( diffuse |
estimation - doing it properly, closely
if;
radiance = SampleLight( &rand, I_s, &L, &lightbuf,
e.x + radiance.y + radiance.z) > 0) && (dot( N
e = true;
at brdfPpdf = EvaluateDiffuse( L, N ) * Psurvive;
at3 factor = diffuse * INVPI;
at weight = Mis2( directPpdf, brdfPpdf );
at cosThetaOut = dot( N, L );
E * (weight * cosThetaOut) / directPpdf) * (radiance
random walk - done properly, closely following Saini
ive)

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



Dielectrics

A typical dielectric transmits *and* reflects light.

Based on the Fresnel equations, the reflectivity of the surface for non-polarized light is formulated as:

$$F_r = \frac{1}{2} \left(\left(\frac{n_1 \cos \theta_i - n_2 \cos \theta_t}{n_1 \cos \theta_i + n_2 \cos \theta_t} \right)^2 + \left(\frac{n_1 \cos \theta_t - n_2 \cos \theta_i}{n_1 \cos \theta_t + n_2 \cos \theta_i} \right)^2 \right)$$

where: $\cos \theta_t = \sqrt{1 - \left(\frac{n_1}{n_2} \sin \theta_i \right)^2}$

And: $F_t = 1 - F_r$



Agenda:

- Introduction / Goals / Planning
- Template / LH2
- Whitted-style recap
- RT-centric Game Dev





Games are often built to showcase technology.

Wolfenstein: indoor, no detail on the floors, sprites for enemies.





Games are often built to showcase technology.
Doom: indoor, height levels, horizontal floors.



```
rics  
    & (depth < MAXDEPTH)  
  
    c = inside ? 1 : 1.2f;  
    nt = nt / nc; ddn = ddn / nc;  
    os2t = 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;  
  
at  
    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, &lightbuf  
e.x + radiance.y + radiance.z) > 0) && (dot( N  
v = true;  
at brdfPdf = EvaluateDiffuse( L, N ) * Psurvive  
at t3 factor = diffuse * INVPI;  
at weight = Mis2( directPdf, brdfPdf );  
at cosThetaOut = dot( N, L );  
E * ((weight * cosThetaOut) / directPdf) * (radianc  
random walk - done properly, closely following Smilin  
ive)  
;  
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf  
urvive;  
pdf;  
    E * brdf * (dot( N, R ) / pdf);  
    = true;
```



Games are often built to showcase technology.
Quake: textures everywhere, polygon enemies, skybox. Still indoor.





```
rics  
& (depth < MAXDEPTH)  
  
    c = inside ? 1 : 1.2f;  
    nt = nt / nc, ddn = ddn / nc;  
    os2t = 1.0f - nnt * nnt;  
    D, N );  
}  
  
at a = nt - nc, b = nt + nc, c = nc;  
at Tr = 1 - (R0 + (1 - R0) / (1 - a));  
Tr) R = (D * nnt - N * (ddn -  
E * diffuse;  
= true;  
  
(refl + refr)) && (depth < MAXDEPTH)  
D, N );  
refl * E * diffuse;  
= true;  
  
MAXDEPTH)  
  
survive = SurvivalProbability( da  
estimation - doing it properly,  
if;  
radiance = SampleLight( &rand, I,  
e.x + radiance.y + radiance.z ) >  
e = true;  
at brdfPdf = EvaluateDiffuse( L,  
at3 factor = diffuse * INVPI;  
at weight = Mis2( directPdf, brdf  
at cosThetaOut = dot( N, L );  
E * ((weight * cosThetaOut) / di  
  
random walk - done properly, closest following -  
alive)  
;  
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );  
survive;  
pdf;  
E * brdf * (dot( N, R ) / pdf);  
ision = true;
```

Games are often built to showcase technology.
Quake 2: colored lights, more detail, some outdoor parts.



```
rics  
    & (depth < MAXDEPTH)  
  
    c = inside ? 1 : 1.2;  
    nt = nt / ncy ddn = ddn / ncy  
    os2t = 1.0f - nnt * os2t  
    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_s, &L, &lightbox  
e.x + radiance.y + radiance.z ) > 0) && (dot( N  
v = true;  
at brdfPpdf = EvaluateDiffuse( L, N ) * Psurvive  
at3 factor = diffuse * INVPI;  
at weight = Mis2( directPpdf, brdfPpdf );  
at cosThetaOut = dot( N, L );  
E * ((weight * cosThetaOut) / directPpdf) * (radiance  
random walk - done properly, closely following Smiley  
ive)  
;  
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf  
survive;  
pdf;  
E * brdf * (dot( N, R ) / pdf);  
ision = true;
```



Games are often built to showcase technology.
Unreal 1: much more outdoor gameplay. Particles. Shiny floors.



```
rics  
3 (depth < MAXDEPTH)  
c = inside ? 1 : 1.25;  
nt = nt / nc; ddn = ddn / nc;  
os2t = 1.0f - nnt * os2t;  
(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, &lightbox  
e.x + radiance.y + radiance.z ) > 0) && (dot( N  
v = true;  
at brdfPdf = EvaluateDiffuse( L, N ) * Psurvive  
at t3 factor = diffuse * INVPI;  
at weight = Mis2( directPdf, brdfPdf );  
at cosThetaOut = dot( N, L );  
E * ((weight * cosThetaOut) / directPdf) * (radiance  
random walk - done properly, closely following Smiley  
alive);  
;  
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf  
survive;  
pdf;  
E * brdf * (dot( N, R ) / pdf);  
ision = true;
```



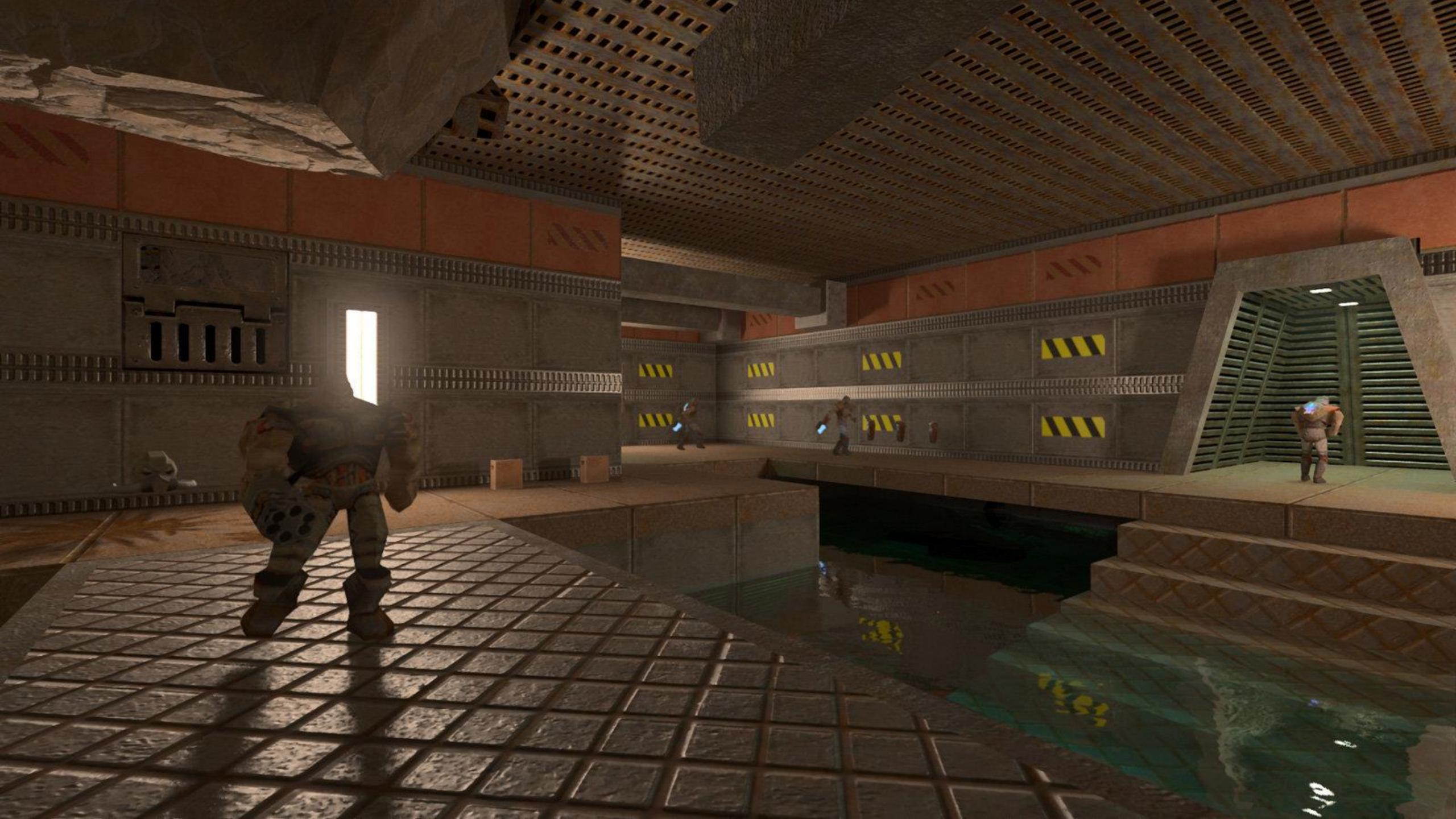
Games are often built to showcase technology.
Duke Nukem: camera can't tilt up and down (well, not really).





Games are often built to showcase technology.
FarCry: instancing (trees). Almost everything outside.







AKER

GACAZIJN

P





A Ray Traced Game

Strengths of ray tracing:

- Correct shadows
- Correct reflection
- Correct refraction
- ...

Not so strong:

- Lots of animation
- High resolution
- High frame rate

```
rics
3 & (depth < MAXDEPTH)
    c = inside ? 1 : 1.2f;
    nt = nc / ncy;
    nnt = nc / ncy;
    os2t = 1.0f - nnt * nnt;
    D, N );
}
)

at a = nt - nc, b = nt + nc;
at Tr = 1 - (R0 + (1 - R0) *
Tr) R = (D * nnt - N * (dd
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, &lighting,
e.x + radiance.y + radiance.z) > 0) && (dot( N
e, L ) > 0);
E * (weight * cosThetaOut) / directPdf) * (radiance
random walk - done properly, closely following Smith's
survive)

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



1997 3Dfx

```

rics
3 < depth < MAXDEPTH
    c = inside ? 1 : 1.2;
    nt = nt / ncy;
    pos2t = 1.0f - nnt * ncy;
    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);
)
N, N );
    refl * E * diffuse;
    = true;
)
MAXDEPTH)
survive = SurvivalProbability( diffuse
estimation - doing it properly, closely
df;
radiance = SampleLight( &rand, I, &L, &lightbase
e.x + radiance.y + radiance.z) > 0) && (dot( N
)
v = true;
at brdfPdf = EvaluateDiffuse( L, N ) * Psurvive;
at t3 factor = diffuse * INVPI;
at weight = Mis2( directPdf, brdfPdf );
at cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf) * (radiance
random walk - done properly, closely following Smiley
alive)
)
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
ision = true;
)

```



```
rics  
    & (depth < MAXDEPTH)  
    c = inside ? 1 : 1.25;  
    nt = nc / ncy; ddn = ddn * c;  
    pos2t = 1.0f - nnt * nnt;  
    D, N );  
)  
  
at a = nt - nc, b = nt + c;  
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, close  
if;  
radiance = SampleLight( &rand, I, &L  
e.x + radiance.y + radiance.z ) > 0 )  
= true;  
at brdfPdf = EvaluateDiffuse( L, N )  
at3 factor = diffuse * INVPI;  
at weight = Mis2( directPdf, brdfPdf  
at cosThetaOut = dot( N, L );  
E * (weight * cosThetaOut) / direct  
random walk - done properly, closely  
survive)  
;  
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf  
survive;  
pdf;  
E * brdf * (dot( N, R ) / pdf);  
ision = true;
```



1999
Ragdoll
physics



```
rics  
    & (depth < MAXDEPTH)  
    c = inside ? 1 : 1.25;  
    nt = nc / ncy; ddn = ddn * c;  
    pos2t = 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)  
  
, N );  
refl * E * diffuse;  
= true;  
  
MAXDEPTH)  
  
survive = SurvivalProbability( diffuse  
estimation - doing it properly, close  
df;  
radiance = SampleLight( &rand, I, &L, &  
e.x + radiance.y + radiance.z) > 0) &&  
v = true;  
at brdfPdf = EvaluateDiffuse( L, N ) *  
at3 factor = diffuse * INVPI;  
at weight = Mis2( directPdf, brdfPdf );  
at cosThetaOut = dot( N, L );  
E * ((weight * cosThetaOut) / directP  
random walk - done properly, closely fol  
survive)  
;  
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf  
survive;  
pdf;  
E * brdf * (dot( N, R ) / pdf);  
ision = true;
```



2005
lens
flares



Ray Tracing for Games

```

    & (depth < MAXDEPTH)
    if (inside) {
        nt = nt / nc;
        os2t = 1.0f - (1.0f - os2) * (1.0f - nt);
        D, N );
    }

    if (a == nt - nc) {
        Tr = 1.0f - (1.0f - Tr) * R;
        D, N );
    }

    E * diffuse;
    isRefractive = true;
}

if (isRefractive) {
    D, N );
    refl * E * diffuse;
    isRefractive = true;
}

MAXDEPTH)

survive = Survive( L, N, directPdf, estimation );
estimation -= (1.0f - survive) * directPdf;
radiance = Survive( L, N, directPdf, estimation );
e.x + radiance);

if (isRefractive) {
    E * ((weight * cosThetaOut) / directPdf) * (radiance -
        random walk - done properly, closely following Spherical
        survive);

    dot3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
    survive;
    pdf;
    E * brdf * (dot( N, R ) / pdf);
    isRefractive = true;
}

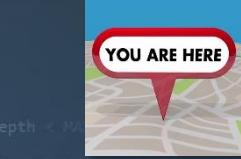
```



Agenda:

- Introduction / Goals / Planning
- Template / LH2
- Whitted-style recap
- RT-centric Game Dev





Wednesday
13:00 – 17:00

course intro
LH2
template
Whitted
refactoring
RT-centric games

LAB 1

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

End product:

1. Path tracer that produces pretty images in a few minutes.
2. Real-time ray tracer on the CPU or GPU.
3. Raytraced game or demo.
4. RTX port of an existing game.



End of PART 1.

