

# INFOMAGR – Advanced Graphics

Jacco Bikker - November 2022 - February 2023

# Lecture 1 - "Introduction"

Welcome!

```
 ) = g(x,x') \left[ \epsilon(x,x') + \int_{S} \rho(x,x',x'') I(x',x'') dx'' \right]
```



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

ef1 + refr)) && (dept

refl \* E \* diffuse;

survive = SurvivalProbability( dif

at3 factor = diffuse \* INVPI; at weight = Mis2( directPdf, brdfPdf at cosThetaOut = dot( N, L );

E \* ((weight \* cosThetaOut) / directPdf)
andom walk - done properly, closely follo

), N );

(AXDEPTH)

### Today's Agenda:

- Advanced Graphics
- Recap: Ray Tracing
- Assignment 1



```
efl + refr)) && (depth < PAXDEPCH)

(2), N );
refl * E * diffuse;
= true;

MAXDEPTH)

Survive = SurvivalProbability( diffuse );
estimation - doing it properly, closes

dif;
radiance = SampleLight( &rand, I, &L, &lighton, e.x. + radiance.y + radiance.z) > 0) && (detains)

ex = true;
est brdfPdf = EvaluateDiffuse( L, N ) * Paurvive, est brdfPdf = EvaluateDiffuse( L, N ) * Paurvive, est weight = Mis2( directPdf, brdfPdf );
est weight = Mis2( directPdf, brdfPdf );
est weight = Mis2( directPdf, brdfPdf );
est (weight * cosThetaOut) / directPdf) * (radianse andom walk - done properly, closely following season vive)

est3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf privive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
ession = true:
```

), N );

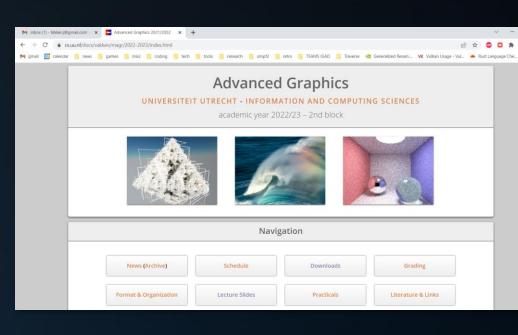
#### Website

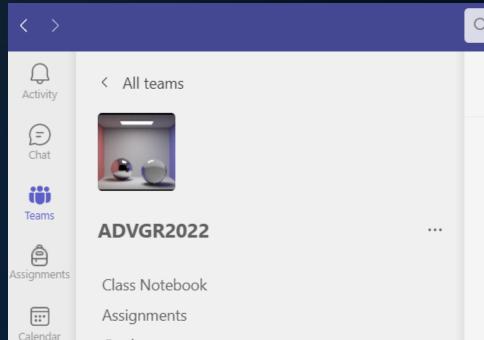
#### http://www.cs.uu.nl/docs/vakken/magr

- Downloads, news, slides, deadlines, links.
- Main source of information!
- Includes complex weekly room allocation.

Please check <u>Teams</u> for operational comms.







#### Abstract

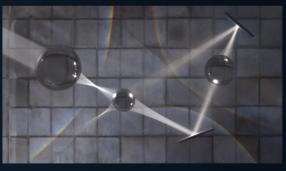
In this course, we explore *Physically Based Rendering* (PBR), with a focus on *interactivity*.

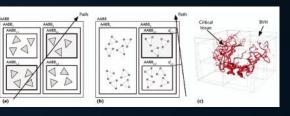
At the end of this course, you will have a solid theoretical understanding of efficient physically based light transport using ray tracing and stochastic evaluation of the Rendering Equation (so: *no rasterization, sorry*).

You will also have a good understanding of acceleration structures for fast ray/scene intersection for static and dynamic scenes.

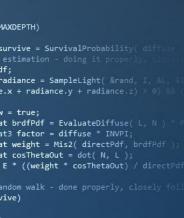
You will have hands-on experience with algorithms for efficient realistic rendering of static and dynamic scenes using ray tracing on CPU and GPU.











at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R

1 = E \* brdf \* (dot( N, R ) / pdf);

ef1 + refr)) && (dep

#### Abstract

In this course, we explore *physically based rendering*, with a focus on interactivity.

At the end of this course, you will have a solid theoretical understanding of efficient physically based light transport using ray tracing and stochastic evaluation of the Rendering Equation (so: *no rasterization, sorry*).

You will also have a good understanding of acceleration structures for fast ray/scene intersection for static and dynamic scenes.

You will have hands-on experience with algorithms for efficient realistic rendering of static and dynamic scenes using ray tracing on CPU and GPU.

### Concrete / informal:

- 1. You'll know how a photorealistic image is produced
- 2. You know how to do this quickly / efficient
- 3. You have built such a renderer
- 4. You have built an interactive ray tracer
- 5. You know how to do this on the GPU
- 6. You got a great score
- 7. You had fun



efl + refr)) && (depth ), N ); refl \* E \* diffuse; = true; radiance = SampleLight( &rand, at weight = Mis2( directPdf, brdfPdf andom walk - done properly, closely

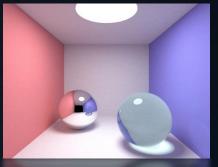
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R.

1 = E \* brdf \* (dot( N, R ) / pdf);

### **Topics**

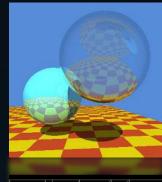
We will cover the following topics:

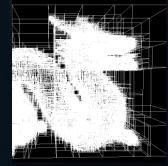
- Ray tracing fundamentals;
- Whitted-style ray tracing;
- Acceleration structure construction;
- Acceleration structure traversal;
- Data structures and algorithms for animation;
- Stochastic approaches to AA, DOF, soft shadows, ...;
- Path tracing;
- Variance reduction in path tracing algorithms;
- Filtering techniques;
- RTX / DXR (hardware options);
- State-of-the-art in ray tracing for games;
- Various forms of parallelism in ray tracing.

















refl \* E \* diffuse

radiance = SampleLight( &rand.

at cosThetaOut = dot( N, L );

at weight = Mis2( directPdf, brdfPdf

andom walk - done properly, closely fol

(AXDEPTH)

#### Lectures

16 lectures:

Tuesday <u>10:00 – 11:45</u>, Thursday 13:15 – 15:00

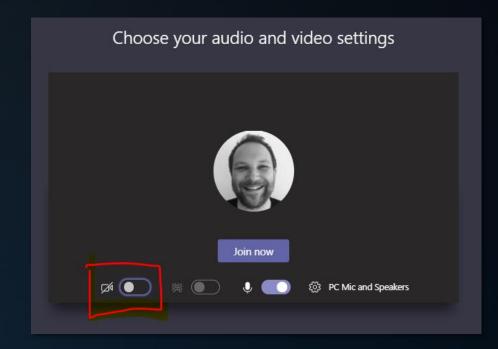
Working colleges:

Tuesdays 09:00 – 10:00 *(1 hour, before the lecture)*Thursdays 15:15 – 17:00 *(2 hours, after the lecture)* 

All lectures are ON CAMPUS and will be recorded.

Slides will be made available, along with recordings.

Attendance is not mandatory, but of course highly recommended. We move fast; missing a key lecture may be a serious problem.



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

andom walk - done properly, closely fol

survive = SurvivalProbability( dit

radiance = SampleLight( &rand,

efl + refr)) && (dept

refl \* E \* diffuse;

), N );

= true;

(AXDEPTH)



#### Literature

Papers and online resources will be supplied during the course.

Slides will be made available after each lecture.

Recommended literature:

Physically Based Rendering – From Theory to Implementation, Pharr & Humphreys. ISBN-10: 9780128006450.

The 3<sup>rd</sup> edition is available for free: www.pbr-book.org

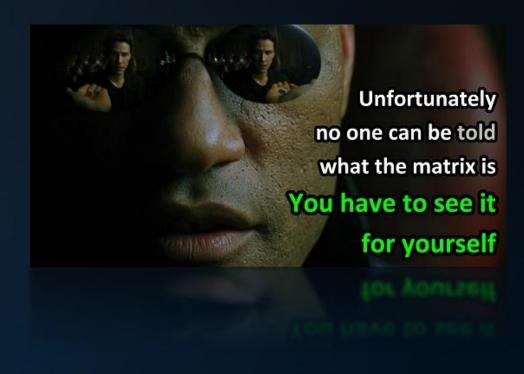


# Dependencies

It is assumed that you have basic knowledge of rendering (INFOGR) and associated mathematics.

You also should be a decent programmer; this is explicitly not a purely theoretical course. You are expected to verify the theory and experience the good and the bad.

You can code in C/C++ or C# or Rust or basically any other Turing-complete language.





```
at3 factor = diffuse * INVPI;
at weight = Mis2( directPdf, brdfPdf );
at cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf) * (radium
andom walk - done properly, closely following securive)

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

efl + refr)) && (depth

radiance = SampleLight( &rand, I, e.x + radiance.y + radian<u>ce.z) > (</u>

at brdfPdf = EvaluateDiffuse( L, I

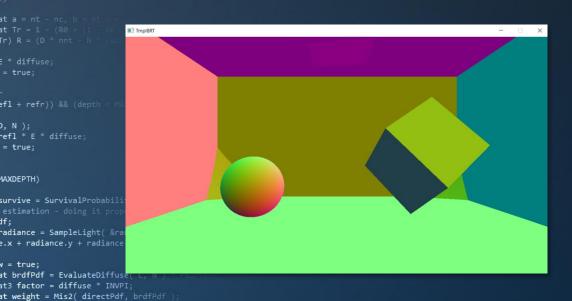
refl \* E \* diffuse;

), N );

(AXDEPTH)

#### Resources

You will develop a ray tracing testbed for assignment 1. As a starting point, a 'template' is available.





However: feel free to use your own framework.



at cosThetaOut = dot( N, L );

E \* ((weight \* cosThetaOut) / directPdf

refl \* E \* diffuse;

andom walk - done properly, closely foll

1 = E \* brdf \* (dot( N, R ) / pdf);

at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R,

#### Assignments

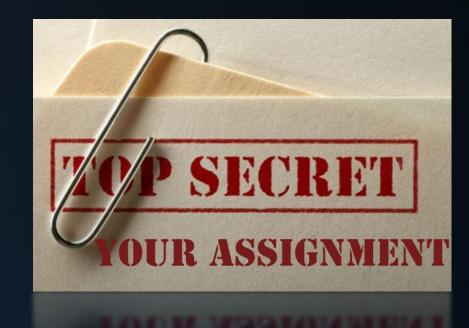
1. (weight: 1):
Light transport framework

For this assignment, you prepare a testbed for subsequent assignments.

2. (weight: 1):
Acceleration structures

In this assignment, you expand your testbed with efficient acceleration structure construction and traversal. This enables you to run ray tracing in real-time (well...)

In this assignment, you either implement an interactive path tracer, or a rendering algorithm you chose, using CPU and/or GPU rendering.





#### Exam

One final exam at the end of the block.

Materials to study:

- Slides
- Notes taken during the lectures
- Provided literature
- Assignments



```
efl + refr)) && (depth < M
D, N );
(AXDEPTH)
survive = SurvivalProbability( diff)
radiance = SampleLight( &rand, I, &L.
e.x + radiance.y + radiance.z) > 0) 8
v = true;
at brdfPdf = EvaluateDiffuse( L, N )
at3 factor = diffuse * INVPI;
at weight = Mis2( directPdf, brdfPdf );
at cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf)
andom walk - done properly, closely following
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pd
```

1 = E \* brdf \* (dot( N, R ) / pdf);

efl + refr)) && (depth < N

survive = SurvivalProbability( diff

radiance = SampleLight( &rand, I, &L

e.x + radiance.y + radiance.z) > 0) &

ot brdfPdf = EvaluateDiffuse( L, N ) ot3 factor = diffuse \* INVPI; ot weight = Mis2( directPdf, brdfPdf );

E \* ((weight \* cosThetaOut) / directPdf)

at cosThetaOut = dot( N, L );

refl \* E \* diffuse;

), N );

= true;

v = true;

### Grading & Retake

```
Final grade for assignments P = (P1 + P2 + 2 * P3) / 4
Final grade for INFOMAGR G = (2P + E) / 3
```

#### Passing criteria:

- $P \ge 4.50$
- $E \ge 4.50$
- *G* ≥ 5.50

Repairing your grade using the retake exam or retake assignment:

- only if  $5.50 > G \ge 4.00$
- you redo P1 or P2 or P3 or E
- this replaces the original P1, P2, P3 or E grade.



andom walk - done properly, closely following sections/vive)

ista brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdfurvive;
pdf;
n = E \* brdf \* (dot( N, R ) / pdf);

### Today's Agenda:

- Advanced Graphics
- Recap: Ray Tracing
- Assignment 1



```
efl + refr)) && (depth < PAXDEPCH)

(2), N );
refl * E * diffuse;
= true;

MAXDEPTH)

Survive = SurvivalProbability( diffuse );
estimation - doing it properly, closes

dif;
radiance = SampleLight( &rand, I, &L, &lighton, e.x. + radiance.y + radiance.z) > 0) && (detains)

ex = true;
est brdfPdf = EvaluateDiffuse( L, N ) * Paurvive, est brdfPdf = EvaluateDiffuse( L, N ) * Paurvive, est weight = Mis2( directPdf, brdfPdf );
est weight = Mis2( directPdf, brdfPdf );
est weight = Mis2( directPdf, brdfPdf );
est (weight * cosThetaOut) / directPdf) * (radianse andom walk - done properly, closely following season vive)

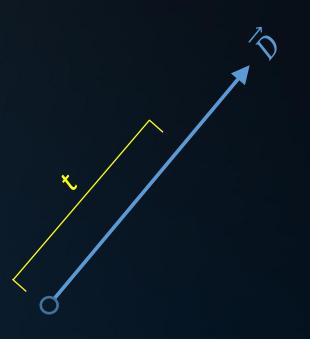
est3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf privive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
ession = true:
```

### Ray

A ray is an infinite line with a start point:

$$P(t) = O + t\overrightarrow{D}$$
, where  $t \ge 0$ .

The ray direction  $\vec{D}$  is usually normalized: this way, t becomes a distance along the ray.





```
efl + refr)) && (depth
D, N );
refl * E * diffuse;
(AXDEPTH)
survive = SurvivalProbability( diff.
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) / directPdf)
andom walk - done properly, closely follow
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &p
1 = E * brdf * (dot( N, R ) / pdf);
```

#### Scene

The scene consists of a number of primitives:

- Spheres
- Planes
- Triangles

...or anything for which we can calculate the intersection with a ray.

We also need:

- A camera (position, direction, FOV, focal distance, aperture size)
- Light sources





```
efl + refr)) && (depth
), N );
refl * E * diffuse;
= true;
(AXDEPTH)
survive = SurvivalProbability( diff
radiance = SampleLight( &rand, I, 8
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) / directPdf)
```

andom walk - done properly, closely follow

1 = E \* brdf \* (dot( N, R ) / pdf);

at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &p

```
D, N );
= true;
(AXDEPTH)
survive = SurvivalProbability( diff
radiance = SampleLight( &rand, I,
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) / directPdf
andom walk - done properly, closely foll

1 = E \* brdf \* (dot( N, R ) / pdf);

at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &p

Ray Tracing

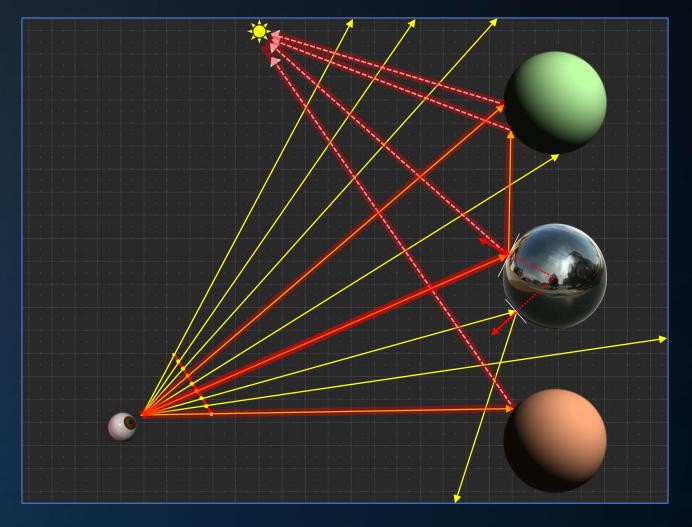
World space

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

Light transport

Extension rays

Light transport





efl + refr)) && (depth

survive = SurvivalProbability( dif

radiance = SampleLight( &rand, I, & e.x + radiance.y + radiance.z) > 0)

at brdfPdf = EvaluateDiffuse( L, N at3 factor = diffuse \* INVPI;

refl \* E \* diffuse;

), N );

= true;

(AXDEPTH)

v = true;

#### Ray setup

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

```
Camera position: E = (0,0,0)
```

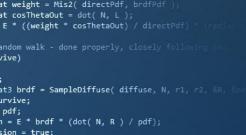
View direction:  $\vec{V} = (0,0,1)$ 

Screen center:  $C = E + d\vec{V}$ 

Screen corners:  $P_0 = C + (-1, -1, 0), P_1 = C + (1, -1, 0), P_2 = C + (-1, 1, 0)$ 

#### From here:

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





), N );

(AXDEPTH)

v = true;

refl \* E \* diffuse;

survive = SurvivalProbability( diff

radiance = SampleLight( &rand, I, & e.x + radiance.y + radiance.z) > 0)

at brdfPdf = EvaluateDiffuse( L, N ) at3 factor = diffuse \* INVPI;

### Ray setup

Point on the screen:

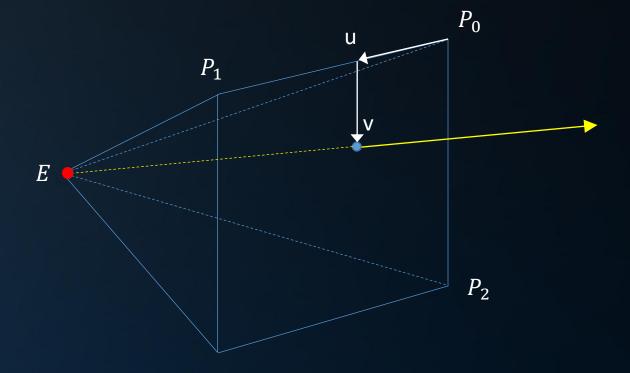
$$P(u,v) = P_0 + u(P_1 - P_0) + v(P_2 - P_0)$$
  
  $u,v \in [0,1]$ 

Ray direction (normalized):

$$\overrightarrow{D} = \frac{P(u,v) - E}{\parallel P(u,v) - E \parallel}$$

Ray origin:

$$O = E$$





```
at weight = Mis2( directPdf, brdfPdf );
at cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf) * (redisorder)
andom walk - done properly, closely following Section (rive)
;
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pd urvive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true:
```

D, N );

(AXDEPTH)

v = true;

refl \* E \* diffuse;

survive = SurvivalProbability( diff)

radiance = SampleLight( &rand, I, &L, & e.x + radiance.y + radiance.z) > 0) &&

at brdfPdf = EvaluateDiffuse( L, N ) \*
at3 factor = diffuse \* INVPI;
at weight = Mis2( directPdf, brdfPdf );
at cosThetaOut = dot( N, L );

E \* ((weight \* cosThetaOut) / directPdf)

andom walk - done properly, closely followi

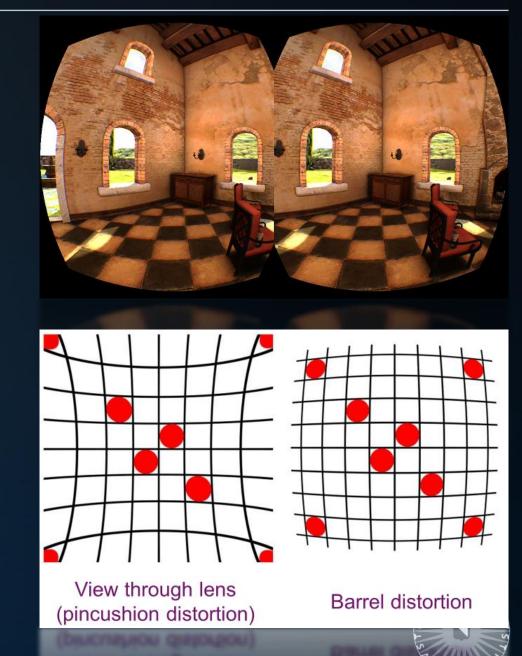
1 = E \* brdf \* (dot( N, R ) / pdf);

at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pd

### Ray setup

#### Alternatives:

Taking into account HMD lens distortion



### Ray setup

### Alternatives:

Taking into account HMD lens distortion

```
Fisheye lens
D, N );
refl * E * diffuse;
(AXDEPTH)
survive = SurvivalProbability( diffu
radiance = SampleLight( &rand, I, &L, &
e.x + radiance.y + radiance.z) > 0) &a
v = true;
at brdfPdf = EvaluateDiffuse( L, N ) | |
at3 factor = diffuse * INVPI;
at weight = Mis2( directPdf, brdfPdf );
at cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf)
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pd
= E * brdf * (dot( N, R ) / pdf);
```



### efl + refr)) && (depth < MAX D, N ); refl \* E \* diffuse; (AXDEPTH) survive = SurvivalProbability( diff) radiance = SampleLight( &rand, I, &L, I e.x + radiance.y + radiance.z) > 0) &a v = true; at brdfPdf = EvaluateDiffuse( L, N ) at3 factor = diffuse \* INVPI; at weight = Mis2( directPdf, brdfPdf ); at cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf) andom walk - done properly, closely followi

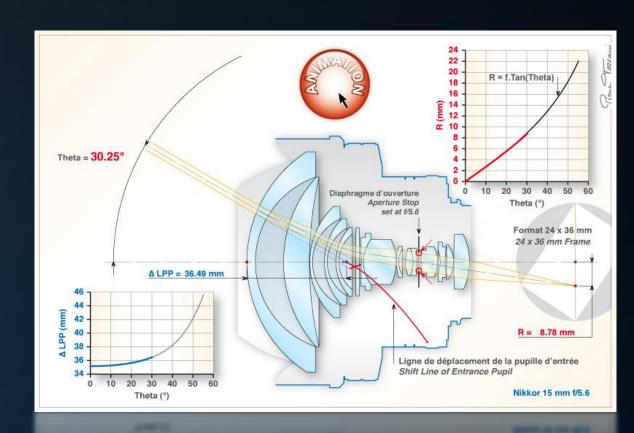
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pd

1 = E \* brdf \* (dot( N, R ) / pdf);

### Ray setup

#### Alternatives:

- Taking into account HMD lens distortion
- Fisheye lens
- Complex lens system



efl + refr)) && (dept

refl \* E \* diffuse;

survive = SurvivalProbability(

### Ray Intersection

Given a ray  $P(t) = O + t\vec{D}$ , we determine the closest intersection distance t by intersecting the ray with each of the primitives in the scene.

Ray / plane intersection:

Plane: 
$$P \cdot \vec{N} + d = 0$$
  
Ray:  $P(t) = 0 + t\vec{D}$ 

Substituting for P(t), we get

$$(O + t\overrightarrow{D}) \cdot \overrightarrow{N} + d = 0$$
  

$$t = -(O \cdot \overrightarrow{N} + d) / (\overrightarrow{D} \cdot \overrightarrow{N})$$
  

$$P = O + t\overrightarrow{D}$$

Math reminder, dot product:

E

$$A \cdot B = A_x B_x + A_y B_y + A_z B_z$$
  
 $A \cdot B = \cos \theta$   
 $A \cdot B$  is: the *length* of the *projection* of A on B.  
 $A \cdot B$  is a *scalar*.

Math notation:

 $P_1$ 

P is a point,  $\overrightarrow{D}$  is a vector t is a scalar.

 $P_2$ 

 $P_{\mathbf{0}}$ 



andom walk - done properly, closely fo vive) ; at3 brdf = SampleDiffuse( diffuse, N, r

at weight = Mis2( directPdf, brdfPdf

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

refl \* E \* diffuse;

survive = SurvivalProbability( dif

at weight = Mis2( directPdf, brdfPdf

n = E \* brdf \* (dot( N, R ) / pdf);

at3 brdf = SampleDiffuse( diffuse, N, r1, r2,

at cosThetaOut = dot( N, L );

#### Ray Intersection

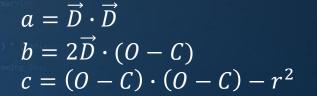
Ray / sphere intersection:

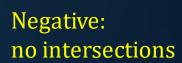
Sphere: 
$$(P-C) \cdot (P-C) - r^2 = 0$$

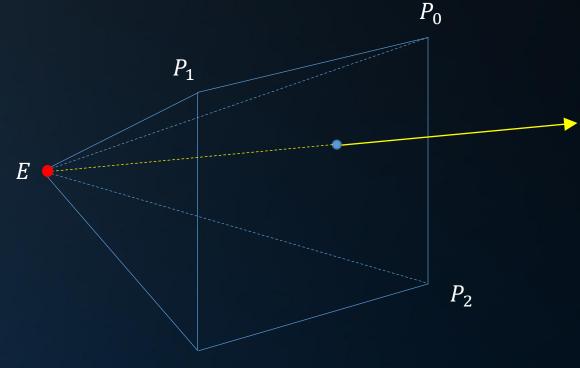
Substituting for P(t), we get

$$(O+t\overrightarrow{D}-C)\cdot(O+t\overrightarrow{D}-C)-r^2=0$$
  
$$\overrightarrow{D}\cdot\overrightarrow{D}\ t^2+2\overrightarrow{D}\cdot(O-C)\ t+(O-C)^2-r^2=0$$

$$at^{2} + bt + c = 0 \rightarrow t = \frac{-b \pm \sqrt{b^{2} - 4ac}}{2a}$$







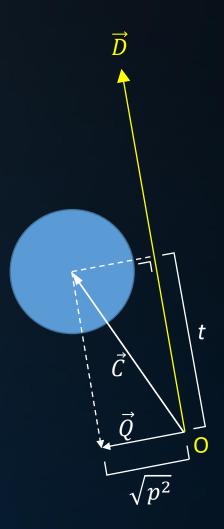


andom walk - done properly, closely follo

1 = E \* brdf \* (dot( N, R ) / pdf);

at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &p

```
Ray Intersection
                         Efficient ray / sphere intersection:
                         void Sphere::IntersectSphere( Ray ray )
                             vec3 C = this.pos - ray.0;
                             float t = dot( C, ray.D );
                             vec3 Q = C - t * ray.D;
efl + refr)) && (depth
                             float p2 = dot( Q, Q );
), N );
                             if (p2 > sphere.r2) return; // r2 = r * r
refl * E * diffuse;
                             t -= sqrt( sphere.r2 - p2 );
                             if ((t < ray.t) && (t > 0)) ray.t = t;
survive = SurvivalProbability( di
radiance = SampleLight( &rand, I,
e.x + radiance.y + radiance.z) > 0
v = true;
                         Note:
at brdfPdf = EvaluateDiffuse( L, N
at3 factor = diffuse * INVPI
at weight = Mis2( directPdf, brdfPdf
                         This only works for rays that start outside the sphere.
at cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf
```





efl + refr)) && (depth < 1

survive = SurvivalProbability( diff.

refl \* E \* diffuse;

), N );

(AXDEPTH)

#### Observations

Ray tracing is a *point sampling process*:

- we may miss small details;
- aliasing will occur.

Ray tracing is a *visibility algorithm*:

• For each pixel, we find the nearest object (which occludes objects farther away).

```
if;
radiance = SampleLight( &rand, I, &L, &IIsmon,
e.x + radiance.y + radiance.z) > 0) && (down)

v = true;
ot brdfPdf = EvaluateDiffuse( L, N ) * Psurvive
at3 factor = diffuse * INVPI;
ot cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf) * (radiance)
andom walk - done properly, closely following season
vive)

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



efl + refr)) && (depth

radiance = SampleLight( &rand, I, & e.x + radiance.y + radiance.z) > 0)

at brdfPdf = EvaluateDiffuse( L, N ) \* at3 factor = diffuse \* INVPI; at weight = Mis2( directPdf, brdfPdf ) at cosThetaOut = dot( N, L );

refl \* E \* diffuse;

), N );

= true;

(AXDEPTH)

v = true;

#### Observations

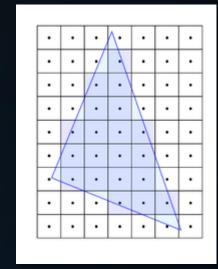
Note: rasterization (Painter's or z-buffer) is also a visibility algorithm.

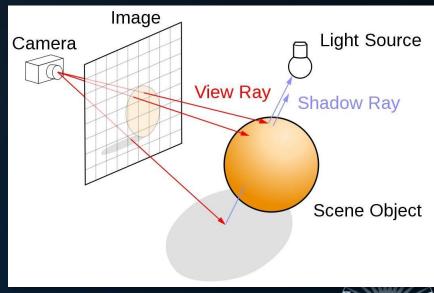
#### Rasterization:

- loop over objects / primitives;
- per primitive: loop over pixels.

#### Ray tracing:

- loop over pixels;
- per pixel: loop over objects / primitives.







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

E \* ((weight \* cosThetaOut) / directPdf)
andom walk - done properly, closely follow

### Today's Agenda:

- Advanced Graphics
- Recap: Ray Tracing
- Assignment 1



```
efl + refr)) && (depth < PAXDEPCH)

(2), N );
refl * E * diffuse;
= true;

MAXDEPTH)

Survive = SurvivalProbability( diffuse );
estimation - doing it properly, closes

dif;
radiance = SampleLight( &rand, I, &L, &lighton, e.x. + radiance.y + radiance.z) > 0) && (detains)

ex = true;
est brdfPdf = EvaluateDiffuse( L, N ) * Paurvive, est brdfPdf = EvaluateDiffuse( L, N ) * Paurvive, est weight = Mis2( directPdf, brdfPdf );
est weight = Mis2( directPdf, brdfPdf );
est weight = Mis2( directPdf, brdfPdf );
est (weight * cosThetaOut) / directPdf) * (radianse andom walk - done properly, closely following season vive)

est3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf privive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
ession = true:
```

efl + refr)) && (depth

survive = SurvivalProbability( diff

radiance = SampleLight( &rand, I, &

e.x + radiance.y + radiance.z) > 0

at brdfPdf = EvaluateDiffuse( L, N

at cosThetaOut = dot( N, L );

at weight = Mis2( directPdf, brdfPdf

1 = E \* brdf \* (dot( N, R ) / pdf);

E \* ((weight \* cosThetaOut) / directPdf)

andom walk - done properly, closely follow

at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &

refl \* E \* diffuse

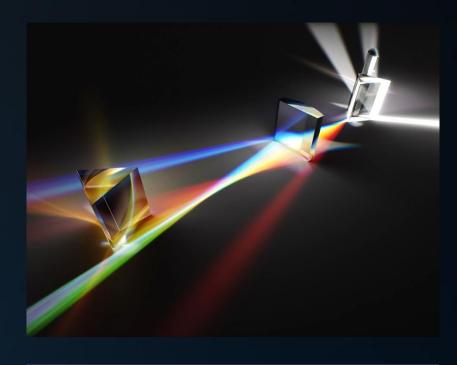
), N );

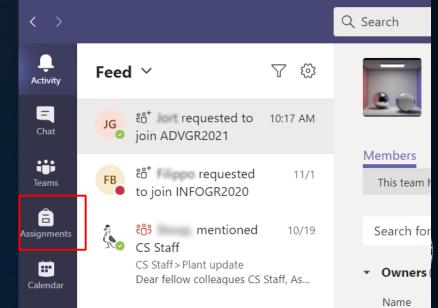
(AXDEPTH)

Ray Tracing Testbed

Basic functionality of a ray tracer:

- 1. Calculate a color for each pixel on the screen
- 2. Do so using rays:
  - Start a ray at the camera location
  - Figure out where the pixel is in world space
  - Extend the (normalized) ray through the pixel
  - Find the nearest intersection (try them all)
  - Do fancy things at the nearest intersection.





1 = E \* brdf \* (dot( N, R ) / pdf);

```
Ray Tracing Testbed
                               Implement an experimentation framework for ray tracing. Ingredients:
                               Scene
                                       Primitives: spheres, planes, triangles
                                       I/O (e.g., obj loader)
                                       Intersection
                                       Materials: diffuse color, diffuse / specular / dielectric, absorption
efl + refr)) && (depth
                               Camera
), N );
refl * E * diffuse;
                                       Position, target, FOV
                                       Ray generation
(AXDEPTH)
survive = SurvivalProbability( dif
                               Ray
radiance = SampleLight( &rand, I, &
                               Renderer
e.x + radiance.y + radiance.z) > 0) 8
v = true;
                                       Whitted-style
at brdfPdf = EvaluateDiffuse( L, N
at3 factor = diffuse * INVPI;
at weight = Mis2( directPdf, brdfPdf
                               User interface
at cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf)
                                       Input handling
andom walk - done properly, closely follo
                                       Presentation
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R,
```



#### Ray Tracing Testbed

Regarding the file loading requirement:

- You may want to start with the included hardcoded scene
- Don't build your own OBJ loader, that's a waste of time
- Use assimp or tinyobjloader (C++) or find a lib if you're using C#

```
http://www.stefangordon.com/parsing-wavefront-obj-files-in-c/
http://www.rexcardan.com/2014/10/read-obj-file-in-c-in-just-10-lines-of-code/
https://github.com/ChrisJansson/ObjLoader
```

Start with small scenes; minimize your development cycle.



```
at weight = Mis2( directPdf, brdfPdf );
at cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf) * (radiance
andom walk - done properly, closely following Section
arive)
;
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pd
urvive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true:
```

efl + refr)) && (depth < P

survive = SurvivalProbability( diff

radiance = SampleLight( &rand, I, &L e.x + radiance.y + radiance.z) > 0)

at brdfPdf = EvaluateDiffuse( L, N ) at3 factor = diffuse \* INVPI;

r**efl \*** E \* diffuse; <u>= tr</u>ue;

), N );

(AXDEPTH)

v = true;

```
Ray Tracing Testbed
```

Intersecting triangles:

An easy to implement and quite efficient algorithm is:

Fast, Minimum Storage Ray/Triangle Intersection, Möller & Trumbore, 1997.

...which is explained in elaborate detail by scratchapixel.com:

http://www.scratchapixel.com/lessons/3d-basic-rendering/ray-tracing-rendering-a-triangle/moller-trumbore-ray-triangle-intersection

```
v = true;
st brdfPdf = EvaluateDiffuse( L, N ) * Psurvive
st3 factor = diffuse * INVPI;
st weight = Mis2( directPdf, brdfPdf );
st cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf) * (radio
andom walk - done properly, closely following seaso
vive)
;
st3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &
urvive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;
```

efl + refr)) && (depth :

survive = SurvivalProbability( diff

radiance = SampleLight( &rand, I, &L e.x + radiance.y + radiance.z) > 0)

refl \* E \* diffuse; = true;

), N );

(AXDEPTH)



### Today's Agenda:

- Advanced Graphics
- Recap: Ray Tracing
- Assignment 1



```
efl + refr)) && (depth < PAXDEPCH)

(2), N );
refl * E * diffuse;
= true;

MAXDEPTH)

Survive = SurvivalProbability( diffuse );
estimation - doing it properly, closes

dif;
radiance = SampleLight( &rand, I, &L, &lighton, e.x. + radiance.y + radiance.z) > 0) && (detains)

ex = true;
est brdfPdf = EvaluateDiffuse( L, N ) * Paurvive, est brdfPdf = EvaluateDiffuse( L, N ) * Paurvive, est weight = Mis2( directPdf, brdfPdf );
est weight = Mis2( directPdf, brdfPdf );
est weight = Mis2( directPdf, brdfPdf );
est (weight * cosThetaOut) / directPdf) * (radianse andom walk - done properly, closely following season vive)

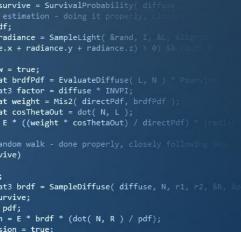
est3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf privive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
ession = true:
```

# INFOMAGR – Advanced Graphics

Jacco Bikker - November 2022 – February 2023

# END of "Introduction"

next lecture: "Whitted-style Ray Tracing"



efl + refr)) && (depth < M

refl \* E \* diffuse; = true;

), N );

(AXDEPTH)

