INFOGR – Computer Graphics

Jacco Bikker - April-July 2016 - Lecture 3: "Ray Tracing (Introduction)"

Welcome!

efl + refr)) && (depth k H

survive = SurvivalProbability(diff

refl * E * diffuse; = true;

), N);

MAXDEPTH)



Today's Agenda:

- Primitives (contd.)
- Ray Tracing
- Intersections
- Assignment 2
- Textures



), N);

MAXDEPTH)

v = true;

efl * E * diffuse;

survive = SurvivalProbability(diff

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



```
efl + refr)) && (depth <
), N );
efl * E * diffuse;
MAXDEPTH)
survive = SurvivalProbability( diff
adiance = SampleLight( &rand, I
e.x + radiance.y + radiance.z) > 0) [[
v = true;
at brdfPdf = EvaluateDiffuse( L, N )
st3 factor = diffuse * INVPI;
st weight = Mis2( directPdf, brdfPdf ):
at cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf)
andom walk - done properly, closely fellow
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, R, lp:
rvive;
```

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

Previously in INFOGR



efl + refr)) && (depth

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

st brdfPdf = EvaluateDiffuse(L, N)
st3 factor = diffuse * INVPI;
st weight = Mis2(directPdf, brdfPdf)
st cosThetaOut = dot(N, L);

refl * E * diffuse; = true;

), N);

v = true;

```
Implicit curves: f(x, y) = 0
```

```
Circle: x^2 + y^2 - r^2 = 0
```

Line:
$$Ax + By + C = 0$$

Slope-intersect form of a line: y = ax + c

Normal of line
$$Ax + By + C = 0$$
: $\vec{N} = \begin{pmatrix} A \\ B \end{pmatrix}$

Distance of line Ax + By + C = 0 to the origin: |C| (if $||\vec{N}|| = 1$)



```
E * ((weight * cosThetaOut) / directPdf) * (radia-
andom walk - done properly, closely following
vive)
;
st3 brdf = SampleDiffuse( diffuse, N, r1, r2, iR, ico-
irvive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;
```

), N);

= true;

efl * E * diffuse;

st weight = Mis2(directPdf, brdfPdf) st cosThetaOut = dot(N, L);

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

ot3 brdf = SampleDiffuse(diffuse, N, r1, r2, RR,);

Parametric representation

Parametric curve:

$$\binom{x}{y} = \binom{g(t)}{h(t)}$$

Example: line

$$p_0 = (x_{p_0}, y_{p_0}), p_1 = (x_{p_1}, y_{p_1})$$

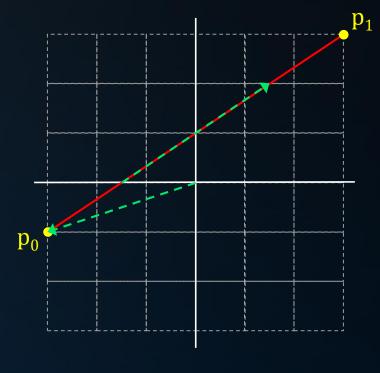
$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x_{p_0} \\ y_{p_0} \end{pmatrix} + t \begin{pmatrix} x_{p_1} - x_{p_0} \\ y_{p_1} - y_{p_0} \end{pmatrix}$$

Or

$$p(t) = p_0 + t(p_1 - p_0), t \in \mathbb{R}.$$

In this example:

 p_0 is the *support vector*; $p_1 - p_0$ is the *direction vector*.





), N);

MAXDEPTH)

v = true;

efl * E * diffuse;

survive = SurvivalProbability(dif

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

st brdfPdf = EvaluateDiffuse(L, N) *
st3 factor = diffuse * INVPI;
st weight = Mis2(directPdf, brdfPdf)
st cosThetaOut = dot(N, L);

Slope-intercept:

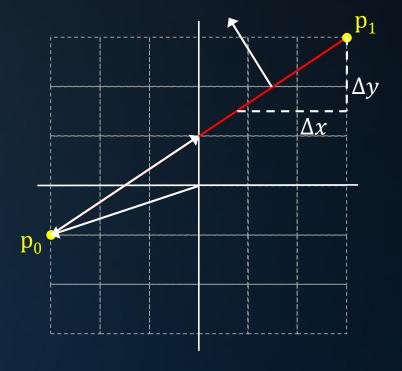
$$y = ax + c$$

Implicit representation:

$$-ax + y - c = 0$$
$$Ax + By + C = 0$$

Parametric representation:

$$p(t) = p_0 + t(p_1 - p_0)$$





```
E * ((weight * cosThetaOut) / directPdf) * (radio
andom walk - done properly, closely following a line
/ive)
;
st3 brdf = SampleDiffuse( diffuse, N, r1, r2, iR, ipo
urvive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;
```

efl + refr)) && (depth

adiance = SampleLight(&rand, I.

at cosThetaOut = dot(N, L);

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

ot3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &

efl * E * diffuse;

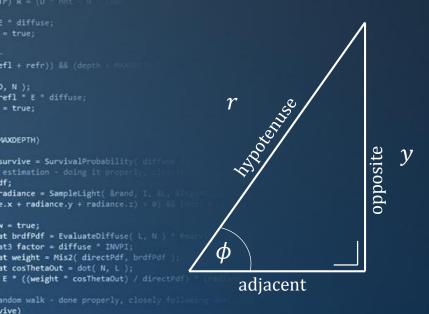
), N);

= true;

Circle - parametric

$$\binom{x}{y} = \binom{x_c + r\cos\phi}{y_c + r\sin\phi}$$

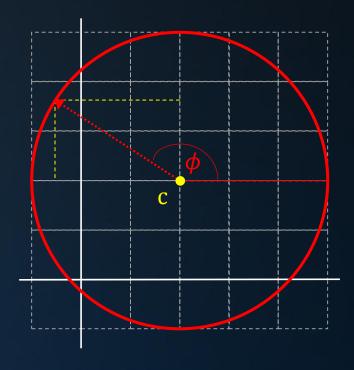
$$\phi$$
 = "phi"



$$\cos \phi = \frac{x}{r}$$

$$\sin \phi = \frac{y}{r}$$

$$\tan \phi = \frac{y}{x}$$



SOH CAH TOA



Circle - sphere (implicit)

Recall: the implicit representation for a circle with radius r and center c is:

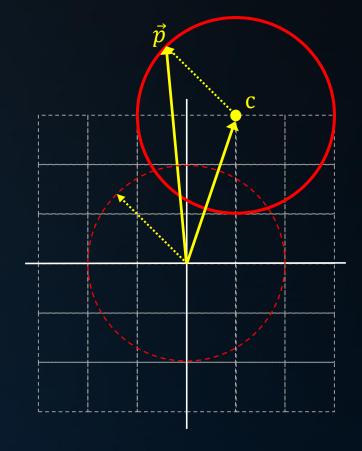
$$(x - c_x)^2 + (y - c_y)^2 - r^2 = 0$$

or:
$$\| p - c \|^2 - r^2 = 0 \implies \| p - c \| = r$$

In \mathbb{R}^3 , we get:

$$(x - c_x)^2 + (y - c_y)^2 + (z - c_z)^2 - r^2 = 0$$

or:
$$|| p - c || = r$$





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

), N);

= true;

MAXDEPTH)

efl * E * diffuse;

survive = SurvivalProbability(dif

adiance = SampleLight(&rand, I.

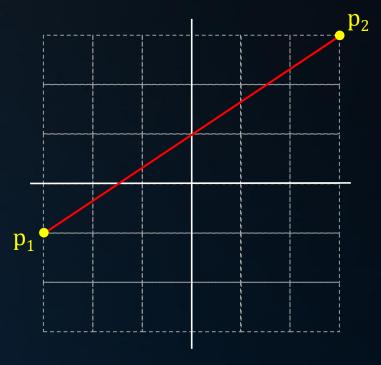
Line - plane (implicit)

Recall: the implicit representation for a line is:

$$Ax + By + C = 0$$

In \mathbb{R}^3 , we get a plane:

$$Ax + By + Cz + D = 0$$





```
st brdfPdf = EvaluateDiffuse( L, N ) * Paurolists
st3 factor = diffuse * INVPI;
st weight = Mis2( directPdf, brdfPdf );
st cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf) * (Paillise
sindom walk - done properly, closely felloward
vive)

st3 brdf = SampleDiffuse( diffuse, N, r1, r2, RR, Rp
urvive;
pdf;
i = E * brdf * (dot( N, R ) / pdf);
sion = true;
```

), N);

= true;

MAXDEPTH)

v = true;

efl * E * diffuse;

survive = SurvivalProbability(diff

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

Parametric surfaces

A parametric surface in \mathbb{R}^3 needs two parameters:

$$x = f(u, v),$$

$$y = g(u, v),$$

$$z = h(u, v).$$

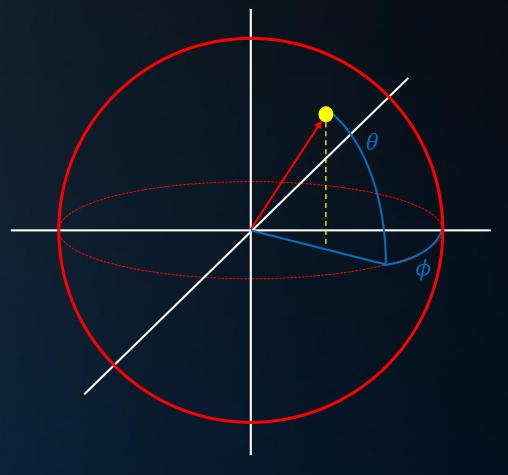
For example, a sphere:

```
x = r \cos \phi \sin \theta,

y = r \sin \phi \sin \theta,

z = r \cos \theta.
```

Doesn't look very convenient (compared to the implicit form), but it will prove useful for texture mapping.



sndom walk - done properly, closely follow III d
vive)

st3 brdf = SampleDiffuse(diffuse, N, r1, r2, UR, outvive;
pdf;
n = E * brdf * (dot(N, R) / pdf);

st weight = Mis2(directPdf, brdfPdf st cosThetaOut = dot(N, L);

adiance = SampleLight(&rand,)

), N);

efl * E * diffuse;

Parametric planes

Recall the parametric line definition:

$$p(t) = p_0 + t(p_1 - p_0)$$

For a plane, we need to parameters:

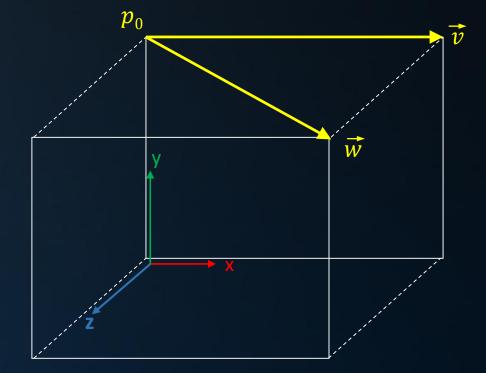
$$p(s,t) = p_0 + s(p_1 - p_0) + t(p_2 - p_0)$$

or:

$$p(s,t) = p_0 + s\vec{v} + t\vec{w}$$

where:

- p_0 is a point on the plane;
- \vec{v} and \vec{w} are two linearly independent vectors on the plane;
- $s, t \in \mathbb{R}$.



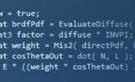




st a = nt - nc st Tr = 1 - (Re fr) R = (D * nr E * diffuse; = true; ef1 + refr)) &8







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



Today's Agenda:

- Primitives (contd.)
- Ray Tracing
- Intersections
- Assignment 2
- Textures



), N);

MAXDEPTH)

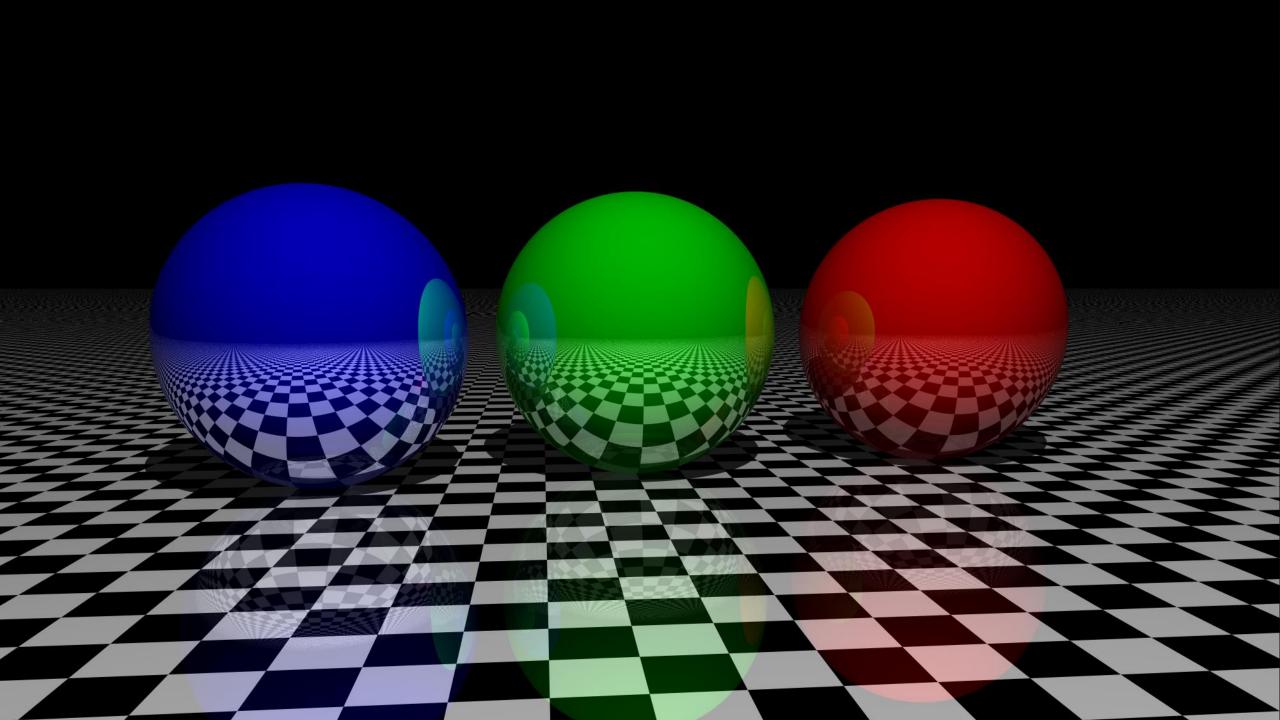
v = true;

efl * E * diffuse;

survive = SurvivalProbability(diff

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





PART 1: Introduction (today)

PART 2: Shading (May 10)

PART 3: Reflections, refraction, absorption (May 17)

PART 4: Path Tracing (June 21)

MAXDEPTH)

Survive = Surviv
estimation - do
if;
radiance = Sampl
e.x + radiance.y

v = true;
bt brdfPdf = Eva
st3 factor = dif
st weight = Mis2
st cosThetaOut =
E * ((weight *

at3 brdf = SampleDiffuse(diffuse, N, rl,

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

efl + refr)) && (depth

efl * E * diffuse;

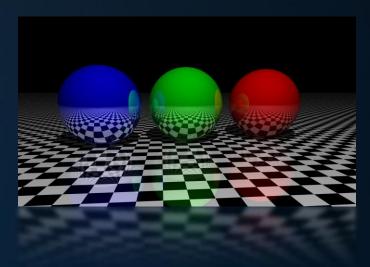
), N);

= true;

/ive)









Ray Tracing:

World space

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

Light transport e.x + radiance.y + radiance.z)

Extension rays

at weight = Mis2(directPdf, brdfPdf

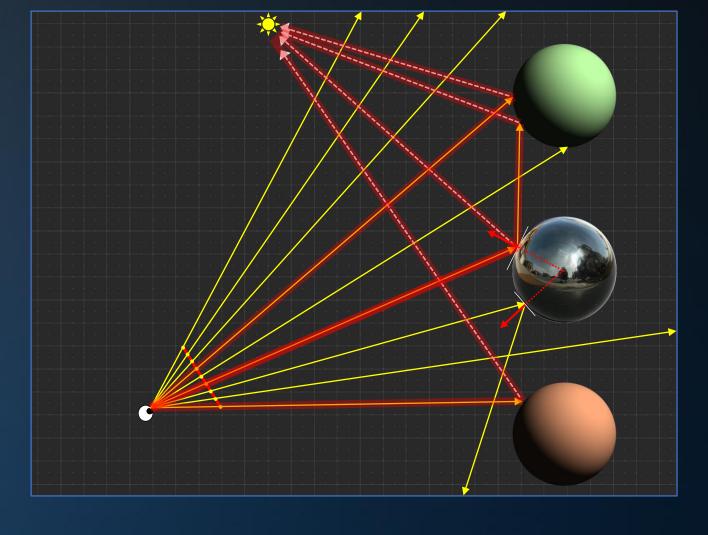
), N);

= true;

MAXDEPTH)

survive = SurvivalProbability

et cosThetaOut = dot(N, L) Light transport





ot3 brdf = SampleDiffuse(diffuse, N, r1, r2, RR,); 1 = E * brdf * (dot(N, R) / pdf);

andom walk - done properly, closely fell

Ray Tracing:

World space

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

Light transport e.x + radiance.y + radiance.z)

), N);

= true;

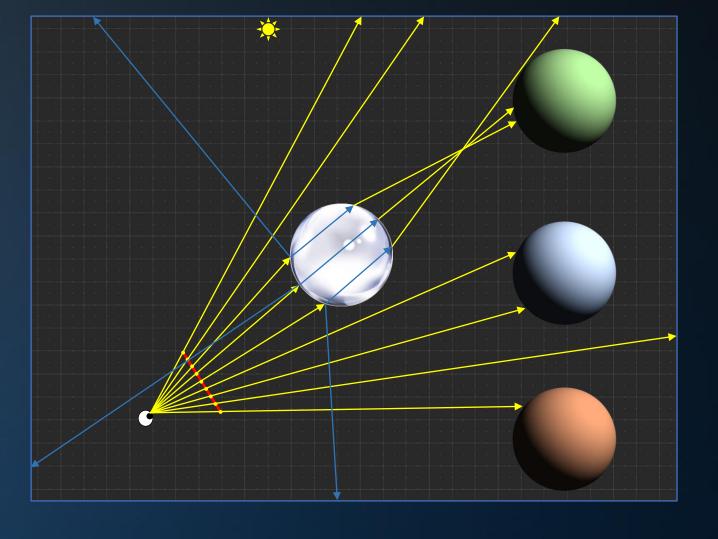
MAXDEPTH)

survive = SurvivalProbability

Extension rays st3 factor = diffuse * INVPI

at weight = Mis2(directPdf, brdfPdf

st cosThetaOut = dot(N, L)Light transport
E * ((weight * cosThetaOut)





ot3 brdf = SampleDiffuse(diffuse, N, r1, r2, RR,); 1 = E * brdf * (dot(N, R) / pdf);

andom walk - done properly, closely fell



Ray Tracing:

World space

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

Light transport

Extension rays

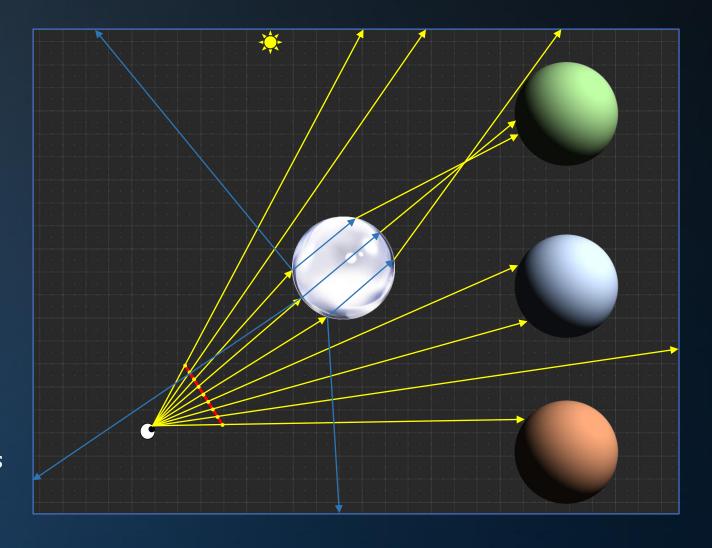
), N);

= true;

MAXDEPTH)

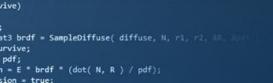
et cosThetaOut = dot(N, L) Light transport

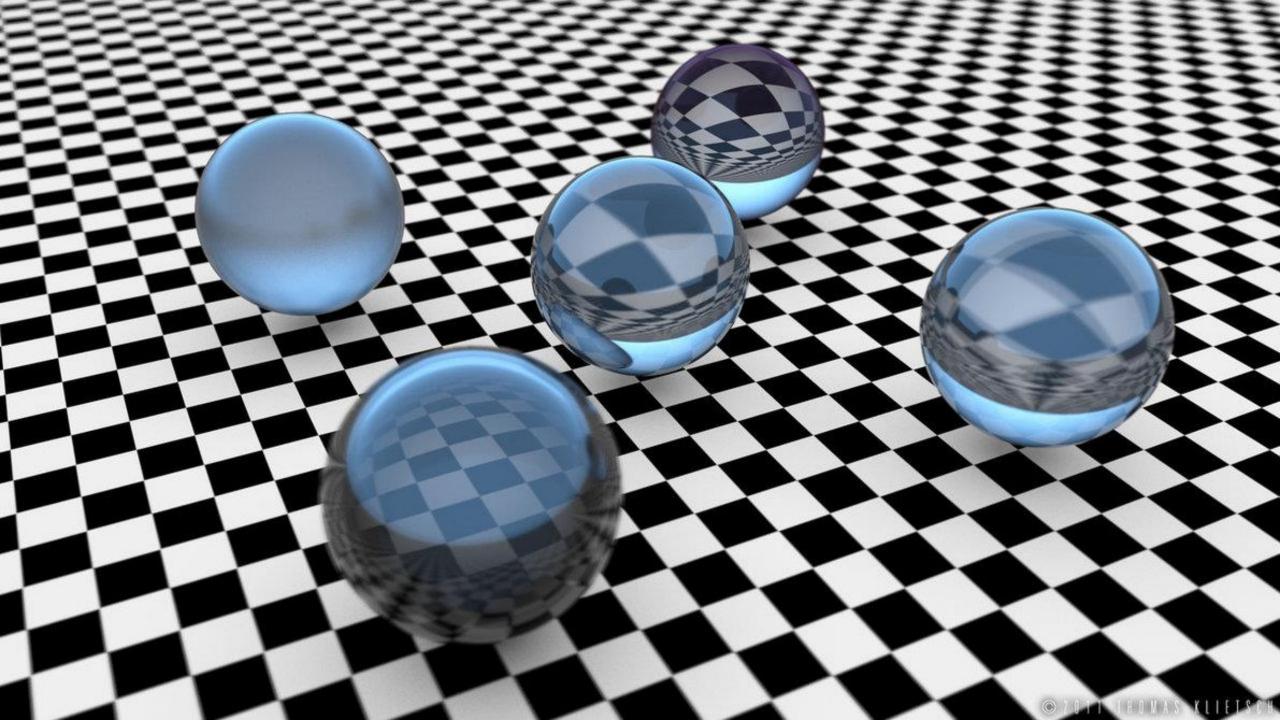
andom walk - done properly, closely fell



Note:

We are calculating light transport backwards.

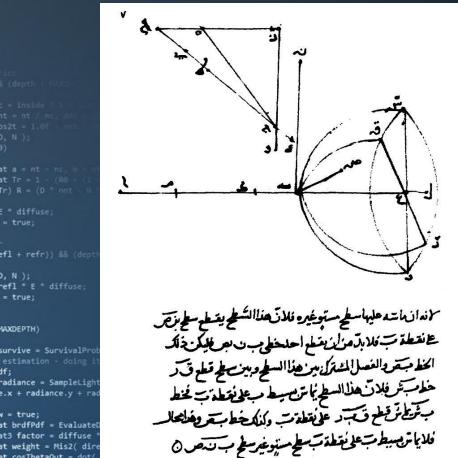


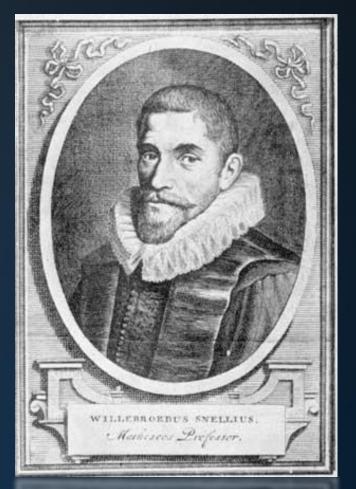


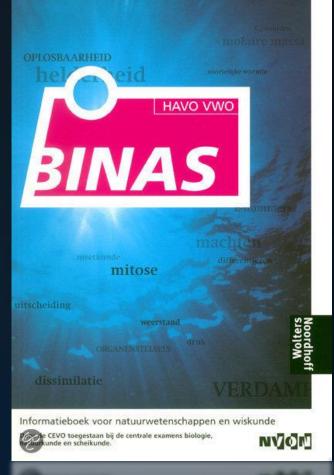
at cosThetaOut = dot

at3 brdf = SampleDiffuse(diffuse, N.

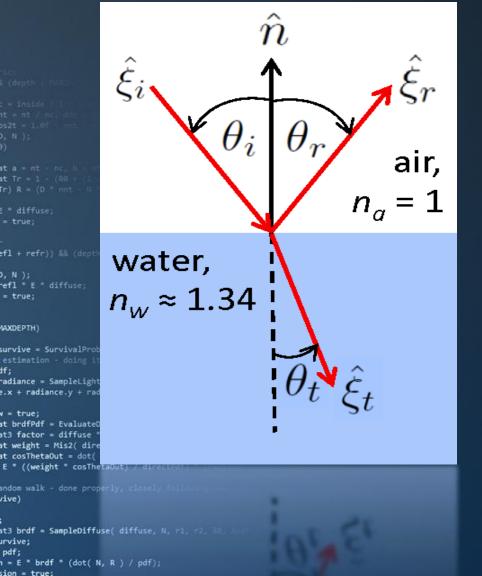
= E * brdf * (dot(N, R) / pdf);

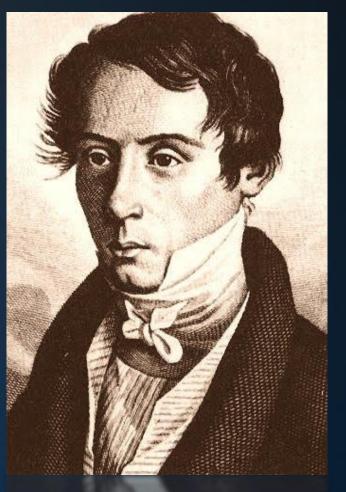


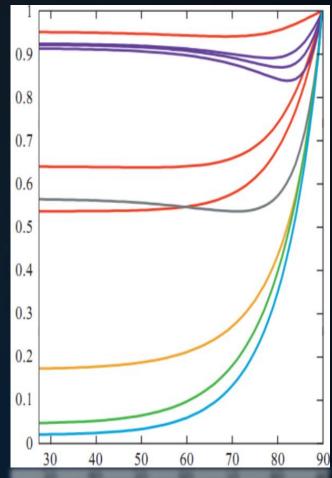














MAXDEPTH)

survive = SurvivalProbability(dif

adiance = SampleLight(&rand, I

st weight = Mis2(directPdf, brdfPdf st cosThetaOut = dot(N, L);

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

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

ot3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &

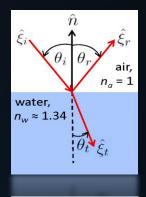
Physical basis

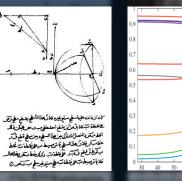
Ray tracing uses *ray optics* to simulate the behavior of light in a virtual environment.

It does so by finding light transport paths:

- From the 'eye'
- Through a pixel
- Via scene surfaces
- To one or more light sources.

At each surface, the light is modulated. The final value is deposited at the pixel (simulating reception by a sensor).











Today's Agenda:

- Primitives (contd.)
- Ray Tracing
- Intersections
- Assignment 2
- Textures



), N);

MAXDEPTH)

v = true;

efl * E * diffuse;

survive = SurvivalProbability(diff

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



Intersections

), N);

= true;

efl * E * diffuse;

Ray definition

A ray is an infinite line with a start point:

```
p(t) = 0 + t\vec{D}, where t > 0.
struct Ray
    float3 0; // ray origin
    float3 D; // ray direction
    float t; // distance
};
```

The ray direction is generally *normalized*.

```
MAXDEPTH)
survive = SurvivalProbability( diff
adiance = SampleLight( &rand, I.
e.x + radiance.y + radiance.z) > 0)
v = true;
at brdfPdf = EvaluateDiffuse( L, N )
st3 factor = diffuse * INVPI
st weight = Mis2( directPdf, brdfPdf
at cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf)
andom walk - done properly, closely fello
ot3 brdf = SampleDiffuse( diffuse, N, r1, r2, R, );
1 = E * brdf * (dot( N, R ) / pdf);
```





efl + refr)) && (depth

survive = SurvivalProbability(dif

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

st3 factor = diffuse * INVPI; st weight = Mis2(directPdf, brdfPdf st cosThetaOut = dot(N, L);

efl * E * diffuse;

= true;

v = true;

Ray setup

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

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

View direction: $ec{V}$

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 the camera matrix.



```
E * ((weight * cosThetaOut) / directPdf) * (radia-
andom walk - done properly, closely following a
/ive)
;
st3 brdf = SampleDiffuse( diffuse, N, r1, r2, 4R, 4ps
urvive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;
```

Ray setup

Point on the screen:

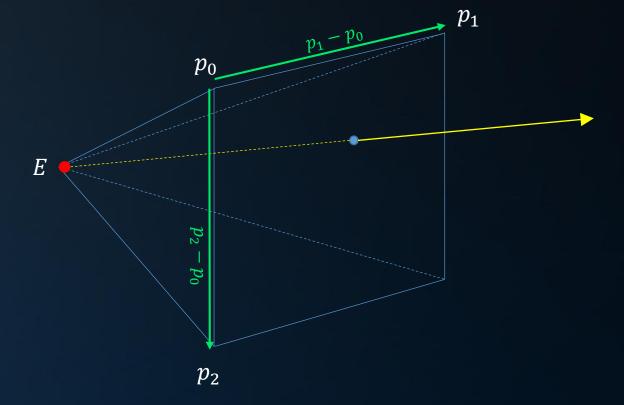
$$p(u,v) = p_0 + u(p_1 - p_0) + v(p_2 - p_0)$$

Ray direction (before normalization):

$$\vec{D} = p(u, v) - E$$

Ray origin:

$$O = E$$





```
), N );
efl * E * diffuse;
= true;
MAXDEPTH)
survive = SurvivalProbability diff
adiance = SampleLight( &rand, I.
e.x + radiance.y + radiance.z) > 0)
v = true;
at brdfPdf = EvaluateDiffuse( L, N )
st3 factor = diffuse * INVPI;
st 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, R)
= E * brdf * (dot( N, R ) / pdf);
```

efl * E * diffuse;

st3 brdf = SampleDiffuse(diffuse, N, r1, r2, iR.

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

Ray intersection

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

Ray / plane intersection:

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

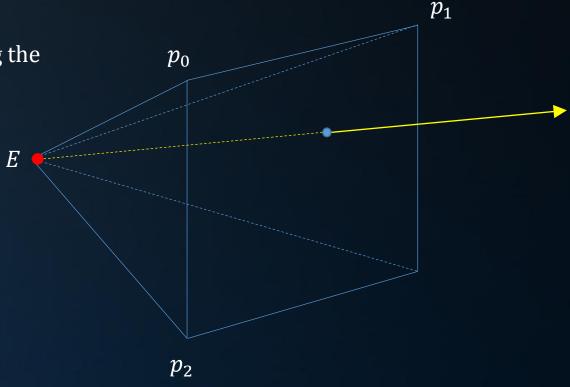
Ray:
$$p(t) = O + t\vec{D}$$

Substituting for p(t), we get

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

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

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





efl * E * diffuse;

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

Ray intersection

Ray / sphere intersection:

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

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

Substituting for p(t), we get

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

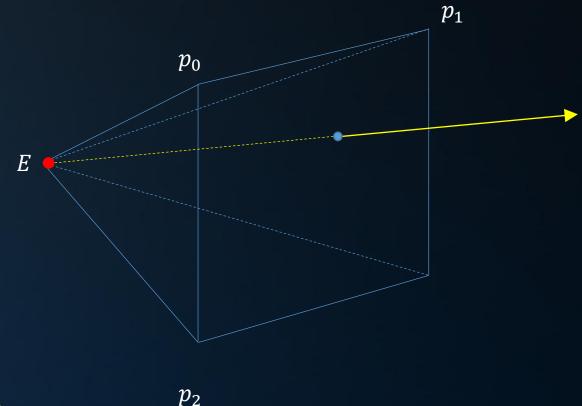
$$\overrightarrow{D}\cdot\overrightarrow{D}\ t^2+2\overrightarrow{D}\cdot(O-C)\ t+(O-C)^2-r^2=0$$

$$ax^{2} + bx + c = 0 \rightarrow x = \frac{-b \pm \sqrt{b^{2} - 4ac}}{2a}$$

$$a=\overrightarrow{D}\cdot\overrightarrow{D}$$
 $b=2\overrightarrow{D}\cdot(O-C)$

leDiffuse(diffuse, N, rl, rl, $c=(O-C)\cdot(O-C)-r^2$

Negative: no intersections

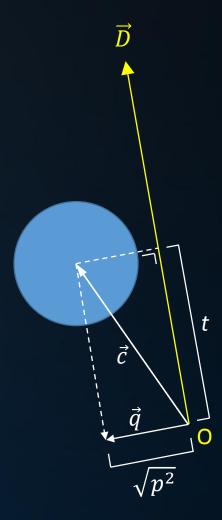


ot3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &

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

```
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;
efl * E * diffuse;
                             t -= sqrt( sphere.r2 - p2 );
                             if ((t < ray.t) && (t > 0)) ray.t = t;
                             // or: ray.t = min( ray.t, max( 0, t ) );
adiance = SampleLight( &rand, I
e.x + radiance.y + radiance.z)
v = true;
t brdfPdf = EvaluateDiffuse(
st3 factor = diffuse * INVPI:
                         Note:
st weight = Mis2( directPdf, brdfPdf
at cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf
andom walk - done properly, closely fell-
```

This only works for rays that start outside the sphere.





Today's Agenda:

- Primitives (contd.)
- Ray Tracing
- Intersections
- Assignment 2
- Textures



), N);

MAXDEPTH)

v = true;

efl * E * diffuse;

survive = SurvivalProbability(diff

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

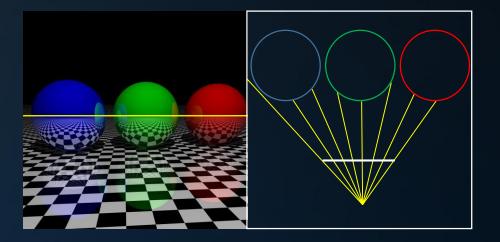


Deadline assignment 1:

Wednesday May 11, 23.59

Assignment 2: "Write a basic ray tracer."

- Using the template
- In a 1024x512 window
- Two views, each 512x512
- Left view: 3D
- Right view: 2D slice





```
sndom walk - done properly, closely following
vive)

st3 brdf = SampleDiffuse( diffuse, N, r1, r2, 4R, 1pd
urvive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sign = ***
```

), N);

= true;

MAXDEPTH)

v = true;

efl * E * diffuse;

survive = SurvivalProbability(dif

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

st brdfPdf = EvaluateDiffuse(L, N) *
st3 factor = diffuse * INVPI;
st weight = Mis2(directPdf, brdfPdf)
st cosThetaOut = dot(N, L);

E * ((weight * cosThetaOut) / directPdf)

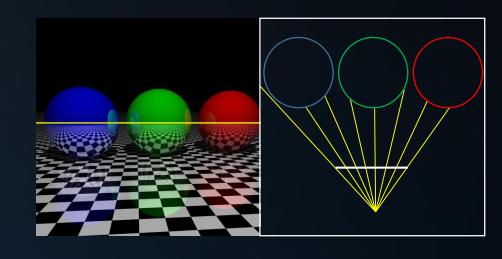
at3 brdf = SampleDiffuse(diffuse, N, rl

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

Assignment 2: "Write a basic ray tracer."

Steps:

- 1. Create a Camera class; default: position (0,0,0), looking at (0,0,-1).
- 2. Create a Ray class
- 3. Create a Primitive class and derive from it a Sphere and a Plane class
- 4. Add code to the Camera class to create a primary ray for each pixel
- 5. Implement Intersect methods for the primitives
- 6. Per pixel, find the nearest intersection and plot a pixel
- 7. Add controls to move and rotate the camera
- 8. Add a checkerboard pattern to the floor plane.
- 9. Add reflections and shadow rays (next lecture).



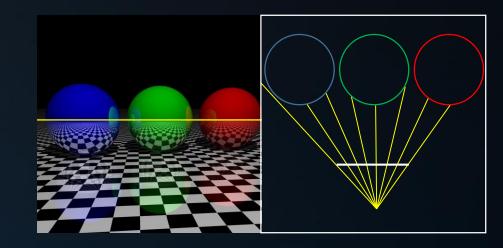
For y = 0, visualize every 10^{th} ray

Visualize the intersection points



Extra points:

- Add additional primitives, e.g.:
 - Triangle, quad, box
 - Torus, cylinder
 - Fractal
- Add textures to all primitives
- Add a sky dome
- Add refraction and absorption (next lecture)
- One extra point for the fastest ray tracer
- One extra point for the smallest ray tracer meeting the minimum requirements.





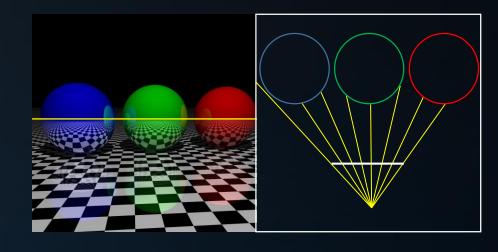


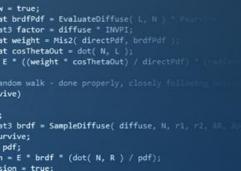


1 = E * brdf * (dot(N, R) / pdf);

Official:

- Full details in the official assignment 2 document, available today from the website.
- Deadline: <u>May 31st 2016, 23:59</u>.
- Small exhibition of noteworthy entries on Thursday June 2nd and on the website.





), N);

= true;

MAXDEPTH)

efl * E * diffuse;

survive = SurvivalProbability(diff

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



Today's Agenda:

- Primitives (contd.)
- Ray Tracing
- Intersections
- Assignment 2
- Textures



), N);

MAXDEPTH)

v = true;

efl * E * diffuse;

survive = SurvivalProbability(diff

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



Textures

efl + refr)) && (depth <

st3 factor = diffuse * INVPI

at cosThetaOut = dot(N, L);

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

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

ot3 brdf = SampleDiffuse(diffuse, N, r1, r2, NR, No

efl * E * diffuse;

), N);

= true;

MAXDEPTH)

Texturing a Plane

Given a plane: y = 0 (i.e., with a normal vector (0,1,0)).

Two vectors on the plane define a basis:

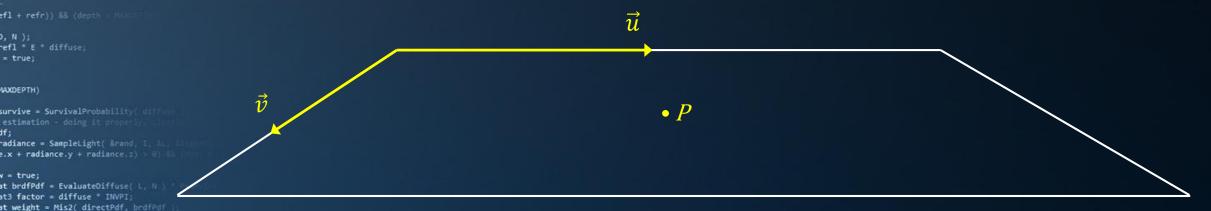
Using these vectors, any point on the plane can be reached:

We can now use λ_1 , λ_2 to define a color at P:

$$\vec{u} = (1,0,0)$$
 and $\vec{v} = (0,0,1)$.

$$P = \lambda_1 \vec{u} + \lambda_2 \vec{v}.$$

$$F(\lambda_1, \lambda_2) = \cdots$$





), N);

MAXDEPTH)

v = true;

efl * E * diffuse;

survive = SurvivalProbability(diff.

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

at brdfPdf = EvaluateDiffuse(L, N)

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

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

at3 brdf = SampleDiffuse(diffuse, N, r1, r2, R, lp:

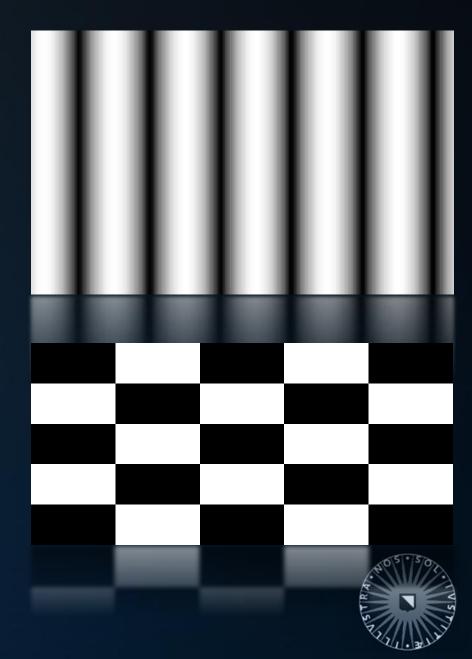
st3 factor = diffuse * INVPI; st weight = Mis2(directPdf, brdfPdf); st cosThetaOut = dot(N, L);

Example:

$$F(\lambda_1, \lambda_2) = \sin(\lambda_1)$$

Another example:

$$F(\lambda_1, \lambda_2) = ((int)(2 * \lambda_1) + (int)\lambda_2) \& 1$$



), N);

MAXDEPTH)

efl * E * diffuse; = true;

survive = SurvivalProbability diff

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

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

st3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf Other examples (not explained here):

Perlin noise

Details: http://www.noisemachine.com/talk1

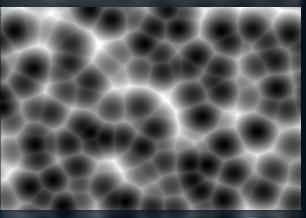
Voronoi / Worley noise

Details: "A cellular texture basis function", S. Worley, 1996.











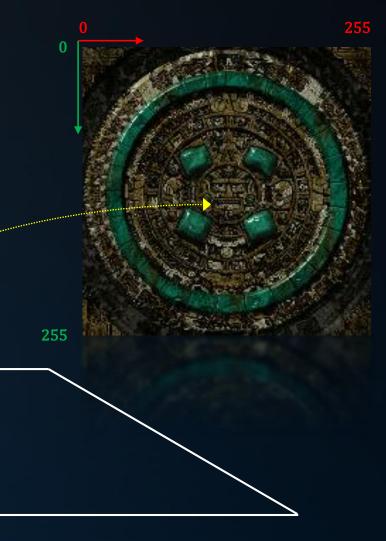


Obviously, not all textures can be generated procedurally.

For the generic case, we lookup the color value in a pixel buffer.

$$\binom{x}{y} = \binom{P \cdot \vec{u}}{P \cdot \vec{v}} * \binom{T_{width}}{T_{height}}$$

Note that we find the pixel to read based on *P*; we don't find a '*P*' for every pixel of the texture.





```
sndom walk - done properly, closely following
sive)

st3 brdf = SampleDiffuse( diffuse, N, r1, r2, RR, ipper
urvive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;
```

E * ((weight * cosThetaOut) / directPdf

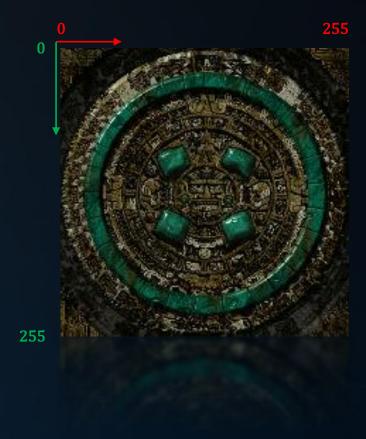
Text<u>ures</u>

Retrieving a pixel from a texture:

$$\binom{x}{y} = \binom{P \cdot \vec{u}}{P \cdot \vec{v}} * \binom{T_{width}}{T_{height}}$$

We don't want to read outside the texture. To prevent this, we have two options:

Tiling is efficiently achieved using a bitmask. This requires texture dimensions that are a power of 2.





ot3 brdf = SampleDiffuse(diffuse, N, r1, r2, NR, N

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

Texture mapping: oversampling

```
), N );
efl * E * diffuse;
MAXDEPTH)
survive = SurvivalProbability( diff
radiance = SampleLight( &rand, I.
e.x + radiance.y + radiance.z) > 0
v = true;
at brdfPdf = EvaluateDiffuse( L, N
st3 factor = diffuse * INVPI;
st weight = Mis2( directPdf, brdfPdf )
st cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf
```



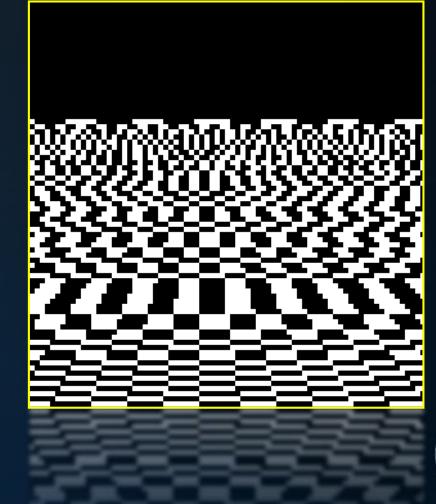


```
vive)

st3 brdf = SampleDiffuse( diffuse, N, r1, r2, IR, pp. pp. pp. pp. pp. pdf;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
```

Texture mapping: undersampling

```
), N );
efl * E * diffuse;
MAXDEPTH)
survive = SurvivalProbability( dif
radiance = SampleLight( &rand, I.
e.x + radiance.y + radiance.z) > 0
v = true;
at brdfPdf = EvaluateDiffuse( L, N )
st3 factor = diffuse * INVPI;
st weight = Mis2( directPdf, brdfPdf );
at cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf)
andom walk - done properly, closely foll
at3 brdf = SampleDiffuse( diffuse, N, rl,
= E * brdf * (dot( N, R ) / pdf);
```





Text<u>ures</u>

efl * E * diffuse;

E * ((weight * cosThetaOut) / directPdf)

andom walk - done properly, closely fello

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

ot3 brdf = SampleDiffuse(diffuse, N, r1, r2, ER,

Fixing oversampling

Oversampling: reading the same pixel from a texture multiple times. Symptoms: blocky textures.

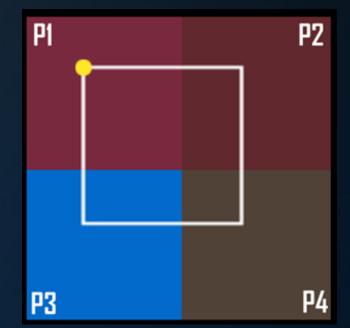
Remedy: bilinear interpolation: Instead of clamping the pixel location to the nearest pixel, we read from four pixels.

```
w_{p1}: (1 - frac(x)) * (1 - frac(y))

w_{p2}: frac(x) * (1 - frac(y))

w_{p3}: (1 - frac(x)) * frac(y)

w_{p4}: 1 - (wP_{1_{+}} wP_{2_{+}} wP_{3_{+}})
```







st cosThetaOut = dot(N, L);

E * ((weight * cosThetaOut) / directPdf)

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

ot3 brdf = SampleDiffuse(diffuse, N, r1, r2, NR,

Fixing oversampling

```
), N );
efl * E * diffuse;
MAXDEPTH)
survive = SurvivalProbability( diffe
adiance = SampleLight( &rand, I. &
e.x + radiance.y + radiance.z) > 0
v = true;
at brdfPdf = EvaluateDiffuse( L, N )
st3 factor = diffuse * INVPI;
at weight = Mis2( directPdf, brdfPdf )
```





), N);

= true;

efl * E * diffuse;

adiance = SampleLight(&rand,

st weight = Mis2(directPdf, brdfPdf

Fixing undersampling

Undersampling: skipping pixels while reading from a texture.

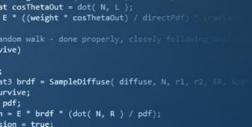
Symptoms: Moiré, flickering, noise.

Remedy: MIP-mapping.

The texture is reduced to 25% by averaging 2x2 pixels. This is repeated until a 1x1 image remains.

When undersampling occurs, we switch to the next MIP level.











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

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

Trilinear interpolation: blending between MIP levels.

```
hardware
guide
), N );
MAXDEPTH)
survive = SurvivalProbability diff
adiance = SampleLight( &rand, I.
                                          Bilinear
                                                                                                                                             Trilinear
e.x + radiance.y + radiance.z) > 0
v = true;
at brdfPdf = EvaluateDiffuse( L, N
st3 factor = diffuse * INVPI;
st weight = Mis2( directPdf, brdfPdf )
st cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf)
```

Today's Agenda:

- Primitives (contd.)
- Ray Tracing
- Intersections
- Assignment 2
- Textures



), N);

MAXDEPTH)

v = true;

efl * E * diffuse;

survive = SurvivalProbability(diff

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



INFOGR – Computer Graphics

Jacco Bikker - April-July 2016 - Lecture 3: "Ray Tracing (Introduction)"

END of "Ray Tracing (Introduction)"

next lecture: "Ray Tracing (Part 2)"

```
st weight = Mis2( directPdf, brdfPdf );
st cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf) * (rudio-
sndom walk - done properly, closely following a
//ve)
;
st3 brdf = SampleDiffuse( diffuse, N, r1, r2, 4R, b
//vive;
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)

at brdfPdf = EvaluateDiffuse(L. N)

(AXDEPTH)

v = true;

