tic: **(depth** ⊂ Pas

: = inside / L it = nt / nc, dde os2t = 1.0f 0, N); 3)

st a = nt - nc, b - nt st Tr = 1 - (80 + (1 Tr) R = (0 * nnt - N

= diffuse = true;

-:fl + refr)) && (depth < HANDIN

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

AXDEPTH)

survive = SurvivalProbability difference estimation - doing it properion if; adiance = SampleLight(@rand I = 1) =x + radiance.y + radiance.z) = 0

v = true; at brdfPdf = EvaluateDiffuse(L, N) * Pourse st3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf)

andom walk - done properly, closely following a /ive)

; pt3 brdf = SampleDiffuse(diffuse, N, r1, r2, NR, brd pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

INFOGR – Computer Graphics

Jacco Bikker - April-July 2016 - Lecture 6: "Boxes"

Welcome!



tice (depth is two

= inside / 1 it = nt / nc, ddo os2t = 1.0f - nnt 0; N(); 3)

at a = nt - nc, b - nt - at Tr = 1 - (R0 + 1 fr) R = (D * nnt - N

= diffuse; = true;

-:fl + refr)) && (depth k HAND)

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

AXDEPTH)

survive = SurvivalProbability difference estimation - doing it property if; adiance = SampleLight(&rand, I 2.x + radiance.y + radiance.z)

v = true; at brdfPdf = EvaluateDiffuse(L, N.) * Pauro st3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) * radii

indom walk - done properly, closely following -/ive)

; pt3 brdf = SampleDiffuse(diffuse, N, r1, r2, UR, bod urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Today's Agenda:

- Introduction
- Boxes
- AABBs
- Groupings
- Efficiency
- To Rasterization





Introduction

tic: K (depth ⊂ 1925

: = inside / 1 it = nt / nc, dde os2t = 1.0f = nnt '), N); 3)

at a = nt - nc, b - nt - at Tr = 1 - (R0 - - - -Fr) R = (D * nnt - N

= diffuse; = true;

-:fl + refr)) && (depth k HANDIII

D, N); ~efl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability difference estimation - doing it properly if; radiance = SampleLight(@rand I = 1) e.x + radiance.y + radiance.r) = 0.000

v = true; at brdfPdf = EvaluateDiffuse(L, N) * Pourch st3 factor = diffuse * INVPI; st weight = Mis2(directPdf, brdfPdf); st cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) * 000

andom walk - done properly, closely following -/ive)

; pt3 brdf = SampleDiffuse(diffuse, N, r1, r2, NR, Dof urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Finalizing the Ray Tracer

... and slowly moving to rasterization:

- Generic scenes: intersecting triangles
- More speed
- Application responsiveness
- Boxes



Introduction

tic: ⊾(depth < 10.

: = inside / l = it = nt / nc, dde os2t = 1.0f = nnt /), N); 3)

at a = nt - nc, b - nt at Tr = 1 - (R0 + (1 Tr) R = (D * nnt - N *

= diffuse; = true;

efl + refr)) && (depth is HANDING

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

AXDEPTH)

survive = SurvivalProbability difference estimation - doing it properly if; radiance = SampleLight(%rand, I = .x + radiance.y + radiance.z) > 0) %

v = true; t brdfPdf = EvaluateDiffuse(L, N) * Pu st3 factor = diffuse * INVPI; st weight = Mis2(directPdf, brdfPdf); st cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf)

andom walk - done properly, closely fo /ive)

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

Intersecting a Triangle

Many ways to intersect a triangle...

Start with the plane:

$$\vec{N} = normalize((v_2 - v_1) \times (v_3 - v_1))$$
$$d = -(\vec{N} \cdot v_1)$$

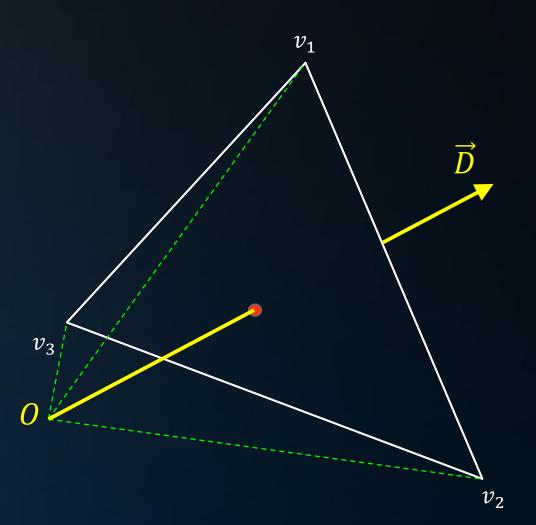
Calculate the intersection of the ray and the plane:

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

And finally, see if point P is on the same side of the three planes between the edges and the origin.

For a more efficient algorithm, see:

Fast, Minimum Storage Ray/Triangle Intersection, Möller & Trumbore. Journal of Graphics Tools, 1997.





Introduction

tice k (depth < 10.5

: = inside / 1 it = nt / nc, dde os2t = 1.0f - nn: 0, N); 0)

at a = nt - nc, b - nt at Tr = 1 - (R0 + 1 Tr) R = (D * nnt - N

= diffuse; = true;

-:fl + refr)) && (depth k HANDIII

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

AXDEPTH)

survive = SurvivalProbability(difference estimation - doing it properly if; radiance = SampleLight(%rand, I, M) e.x + radiance.y + radiance.z) = 0.000

v = true; at brdfPdf = EvaluateDiffuse(L, M) at3 factor = diffuse = INVPI;

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

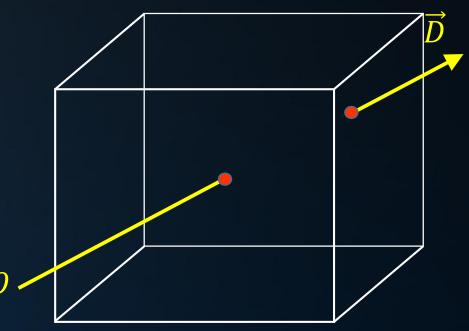
andom walk - done properly, closely following a /ive)

; pt3 brdf = SampleDiffuse(diffuse, N, r1, r2, NR, source; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Intersecting a Box

Basic ray/box intersection:

- 1. Intersect the ray with each of the 6 planes;
- 2. Keep the intersections that are on the same side of the remaining planes;
- 3. Determine the closest intersection point.





tice (depth is two

= inside / 1 it = nt / nc, ddo os2t = 1.0f - nnt 0; N(); 3)

at a = nt - nc, b - nt - at Tr = 1 - (R0 + 1 fr) R = (D * nnt - N

= diffuse; = true;

-:fl + refr)) && (depth k HAND)

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

AXDEPTH)

survive = SurvivalProbability difference estimation - doing it property if; adiance = SampleLight(&rand, I 2.x + radiance.y + radiance.z)

v = true; at brdfPdf = EvaluateDiffuse(L, N.) * Pauro st3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) * read

indom walk - done properly, closely following -/ive)

; pt3 brdf = SampleDiffuse(diffuse, N, r1, r2, UR, bod urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Today's Agenda:

- Introduction
- Boxes
- AABBs
- Groupings
- Efficiency
- To Rasterization





Boxes

tic: k (depth < 100

= inside / 1 it = nt / nc, ddo os2t = 1.8f - ont 0, N); 3)

st a = nt - nc, b = nt = -ncst Tr = 1 - (R0 + (1 - 1))Tr) R = (0 + nnt - 1)

= diffuse; = true;

= efl + refr)) && (depth < HANDIT

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

AXDEPTH)

survive = SurvivalProbability(difference estimation - doing it property if; radiance = SampleLight(&rand, I, A), e.x + radiance.y + radiance.z) > 0) %

v = true; at brdfPdf = EvaluateDiffuse(L, N) * F at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf

andom walk - done properly, closely following vive)

; st3 brdf = SampleDiffuse(diffuse, N, r1, r2, NR, NA pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Special Case: AABB

AABB: Axis Aligned Bounding Box.

Slab test:

Intersect the ray against pairs of planes;

 $t_{min} = +\infty, t_{max} = -\infty$ $t_{min} = \max(t_{min}, \min(t1, t2))$ $t_{max} = \min(t_{max}, \max(t1, t2))$ intersection if: $t_{min} < t_{max}$

Since the box is axis aligned, calculating t is cheap:

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

 $=(x_{plane}-O_x)/\vec{D}_x$

 $d = -(\vec{N} \cdot P)$, where P is a point on the plane. In this case, for $\vec{N} = (1,0,0)$: $d = -P_x = -x_{plane}$, and thus: $t = -(O_x \cdot \vec{N}_x + d) / (\vec{D}_x \cdot \vec{N}_x)$ $= -(O_x - x_{plane})/\overline{D}_x$ t_2 $=(x_{plane}-O_x)/\vec{D}_x$ Note: during college, the last equation was t_{min} erroneously written as $(O_x - x_{plane})/\vec{D}_x$. t_2 t_{min} t_1 t_{max} t_1

Boxes

tic: k (depth < 100

: = inside / 1 it = nt / nc, ddo ss2t = 1.0f = ont 5, N); 3)

st a = nt - nc, b - nt st Tr = 1 - (R0 + (1 - 1 Tr) R = (D * nnt - N *

= diffuse; = true;

-:fl + refr)) && (depth & HAODET

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

AXDEPTH)

v = true; at brdfPdf = EvaluateDiffuse(L, N) Probt3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) *

andom walk - done properly, closely following /ive)

, t3 brdf = SampleDiffuse(diffuse, N, r1, r2, HR, hor urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Special Case: AABB

In pseudo-code:

bool intersection(box b, ray r)

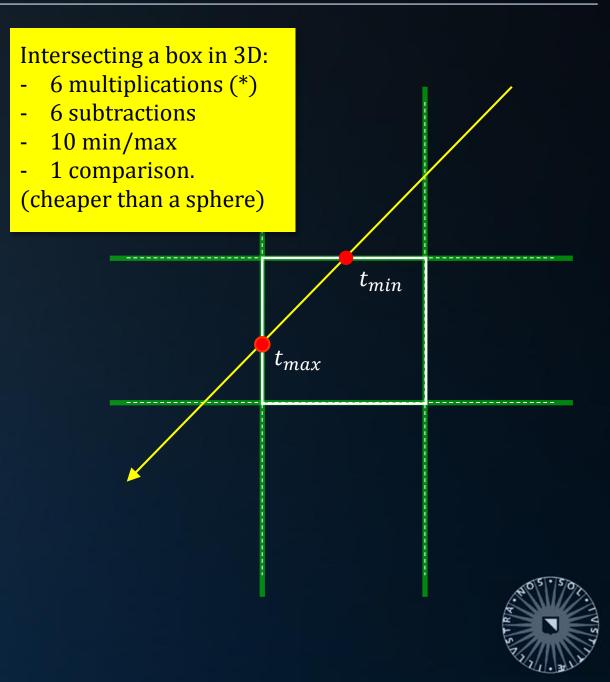
float tx1 = (b.min.x - r.0.x) / r.D.x;
float tx2 = (b.max.x - r.0.x) / r.D.x;

float tmin = min(tx1, tx2);
float tmax = max(tx1, tx2);

float ty1 = (b.min.y - r.0.y) / r.D.y;
float ty2 = (b.max.y - r.0.y) / r.D.y;

tmin = max(tmin, min(ty1, ty2)); tmax = min(tmax, max(ty1, ty2));

return tmax >= tmin;



tice (depth is two

= inside / 1 it = nt / nc, ddo os2t = 1.0f - nnt 0; N(); 3)

at a = nt - nc, b - nt - at Tr = 1 - (R0 + 1 fr) R = (D * nnt - N

= diffuse; = true;

-:fl + refr)) && (depth k HAND)

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

AXDEPTH)

survive = SurvivalProbability difference estimation - doing it property if; adiance = SampleLight(&rand, I 2.x + radiance.y + radiance.z)

v = true; at brdfPdf = EvaluateDiffuse(L, N.) * Pauro st3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) * read

indom walk - done properly, closely following -/ive)

; pt3 brdf = SampleDiffuse(diffuse, N, r1, r2, UR, bod urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Today's Agenda:

- Introduction
- Boxes
- AABBs
- Groupings
- Efficiency
- To Rasterization





AABBs

tice ≰ (depth (⊂1933)

= inside / 1 it = nt / nc, dda 552t = 1.0f - nn 5, N); 3)

st a = nt - nc, b - nt + st Tr = 1 - (R0 + (1 Tr) R = (D * nnt - N *

= diffuse = true;

: :fl + refr)) && (depth (MACO)

D, N); ~efl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(difference estimation - doing it properly if; adiance = SampleLight(%rand, I, Market e.x + radiance.y + radiance.r) > 0)

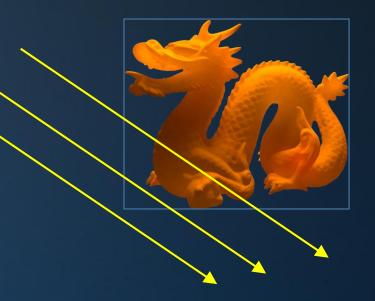
v = true; t brdfPdf = EvaluateDiffuse(L, N.) Provident st3 factor = diffuse * INVPI; st weight = Mis2(directPdf, brdfPdf); st cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) * Factore

andom walk - done properly, closely following -/ive)

; pt3 brdf = SampleDiffuse(diffuse, N, r1, r2, UR, D) pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Why Do We Care

We can use an AABB to quickly discard objects.







AABBs

tic: (depth cliss:

= inside / 1 it = nt / nc, dde os2t = 1.0f - nnt -D, N); B)

at a = nt - nc, b - nt at Tr = 1 - (R0 + (1 - 7) Tr) R = (D * nnt - 8

= diffuse; = true;

efl + refr)) && (depth k HANDIIII

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

AXDEPTH)

survive = SurvivalProbability difference estimation - doing it property ff; radiance = SampleLight(&rand, I, L, .x + radiance.y + radiance.r) = 0.000

v = true; t brdfPdf = EvaluateDiffuse(L, N) Promote st3 factor = diffuse * INVPI; st weight = Mis2(directPdf, brdfPdf); st cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) * Definition

andom walk - done properly, closely following : /ive)

; t33 brdf = SampleDiffuse(diffuse, N, r1, r2, RR, R, r rrvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

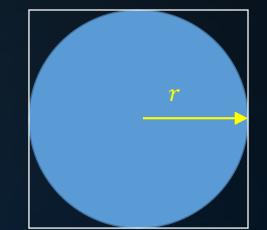
Calculating the AABB

Definition:

struct AABB
{
 vec3 bmin, bmax;
};

For a sphere:

AABB box; box.bmin = centre - vec3(r, r, r); box.bmax = centre + vec3(r, r, r);





AABBs

efl + refr)) && (depth

survive = SurvivalProbability(dif

radiance = SampleLight(&rand, I, 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

-efl * E * diffuse;

), N);

AXDEPTH)

v = true;

Calculating the AABB

For a triangle:

```
AABB box;
```

box.bmin = vec3(+INF, +INF, +INF); box.bmax = vec3(-INF, -INF, -INF); for(int i = 0; i < 3; i++)</pre>

```
for( int a = 0; a < 3; a++ )</pre>
```

box.bmin[a] = min(vert[i][a], box.bmin[a]); box.bmax[a] = max(vert[i][a], box.bmax[a]);



For multiple triangles, the algorithm is the same.

andom walk - done properly, closely followin /ive)

, t3 brdf = SampleDiffuse(diffuse, N, F1, F2, UR, D) prvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:



AABBs

ic: (depth < NAS

: = inside / 1 it = nt / nc, dde os2t = 1.0f - nnt ' D, N); D)

at a = nt - nc, b = nt - ncat Tr = 1 - (R0 + (1 - 0))Tr) R = (D * nnt - N - 0)

= diffuse; = true;

: :fl + refr)) && (depth < HANDE

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

AXDEPTH)

v = true;

at brdfPdf = EvaluateDiffuse(L, N) = Pauro st3 factor = diffuse = INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) = Faul

andom walk - done properly, closely following -/ive)

; pt3 brdf = SampleDiffuse(diffuse, N, r1, r2, R pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Calculating the AABB

For multiple AABBs (union):

box.bmin.x	=	min(A.bmin.x,	B.bmin.x);
<pre>box.bmax.x</pre>	=	max(A.bmax.x,	B.bmax.x);
<pre>box.bmin.y</pre>	=	min(A.bmin.y,	B.bmin.x);
<pre>box.bmax.y</pre>	=	max(A.bmax.y,	B.bmax.y);
box.bmin.z	=	min(A.bmin.z,	B.bmin.z);
<pre>box.bmax.z</pre>	=	max(A.bmax.z,	B.bmax.z);





AABBs

tic: ⊾ (depth ⊂ 1925)

: = inside / 1 it = nt / nc, dde ss2t = 1.0f - nnt), N); 3)

at a = nt - nc, b - nt at Tr = 1 - (R0 + () fr) R = (D * nnt - N *)

= diffuse; = true;

-:fl + refr)) && (depth < NACOOO

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

AXDEPTH)

survive = SurvivalProbability difference estimation - doing it properly if; radiance = SampleLight(@rand I down e.x + radiance.y + radiance.r) = 0.000

v = true; at brdfPdf = EvaluateDiffuse(L, N.) Provident st3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) * read

andom walk - done properly, closely following a /ive)

; pt3 brdf = SampleDiffuse(diffuse, N, r1, r2, NR, brd pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Calculating the AABB

Checking AABB intersection:

	(A.bmin.x	<	B.bmax.x)
&&	(A.bmin.y	<	B.bmax.y)
&&	(A.bmin.z	<	B.bmax.z)





tice (depth is two

= inside / 1 it = nt / nc, ddo os2t = 1.0f - nnt 0; N(); 3)

at a = nt - nc, b - nt - at Tr = 1 - (R0 + 1 fr) R = (D * nnt - N

= diffuse; = true;

-:fl + refr)) && (depth k HAND)

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

AXDEPTH)

survive = SurvivalProbability difference estimation - doing it property if; adiance = SampleLight(&rand, I 2.x + radiance.y + radiance.z)

v = true; at brdfPdf = EvaluateDiffuse(L, N.) * Pauro st3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) * read

indom walk - done properly, closely following -/ive)

; pt3 brdf = SampleDiffuse(diffuse, N, r1, r2, UR, bod urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Today's Agenda:

- Introduction
- Boxes
- AABBs
- Groupings
- Efficiency
- To Rasterization





Groupings

tic: K (depth < 10

= inside / 1 it = nt / nc, ddo 552t = 1.8f = nnt 3, N); 3)

at a = nt - nc, b - nt - at Tr = 1 - (R0 + 11 - - - -Tr) R = (D * nnt - N * - -

= diffuse; = true;

-:fl + refr)) && (depth is HANDIII

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

AXDEPTH)

v = true; at brdfPdf = EvaluateDiffuse(L, N) at3 factor = diffuse = INVPI;

st weight = Mis2(directPdf, brdfPdf); st cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) * (P)

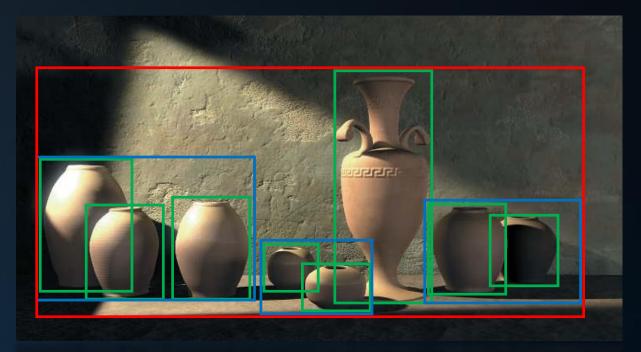
indom walk - done properly, closely following -/ive)

; t3 brdf = SampleDiffuse(diffuse, N, r1, r2, R, spin urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true;

Hierarchical Grouping

Using AABBs, we can recursively group objects.

- A ray that misses a green box will not check the triangles inside it;
- A ray that misses a blue box will skip the two green boxes inside it;
- A ray that misses the red box doesn't hit anything at all.





Groupings

tice ≰ (depth < 100

: = inside / 1 ht = nt / nc, dde ps2t = 1.0f - nnt / p, N); 3)

st a = nt - nc, b - nt + + st Tr = 1 - (R0 + (1 Tr) R = (D * nnt - N

= diffuse = true;

-:fl + refr)) && (depth & HADDET

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

AXDEPTH)

survive = SurvivalProbability(difference estimation - doing it property if; radiance = SampleLight(&rand, I e.x + radiance.y + radiance.r) > 0)

v = true;

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

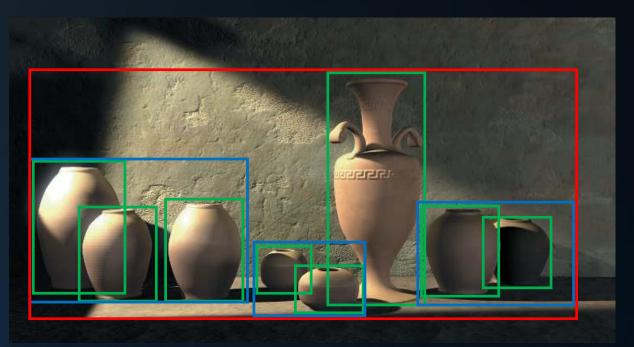
andom walk - done properly, closely following : /ive)

; pt3 brdf = SampleDiffuse(diffuse, N, r1, r2, UR, pp4 pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Hierarchical Grouping

In a rasterization-based world:

- If a green box is outside the view frustum, we don't have to render the triangles inside it;
- If a blue box is outside the view frustum, we don't have to test the green boxes inside it;
- If the red box is outside the view frustum, we don't see anything.





Groupings

tic: (depth (1995

: = inside / 1 ht = nt / nc, ddo os2t = 1.0f = nnt 0, N); 3)

at a = nt - nc, b + nt + + at Tr = 1 - (R0 + (1 - 1) Tr) R = (D * nnt - 1 -

= diffus = true;

-:fl + refr)) && (depth is HANDI-

D, N); -efl * E * diffuse; = true;

AXDEPTH)

v = true; at brdfPdf = EvaluateDiffuse(L, N,) * Psu st3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf)

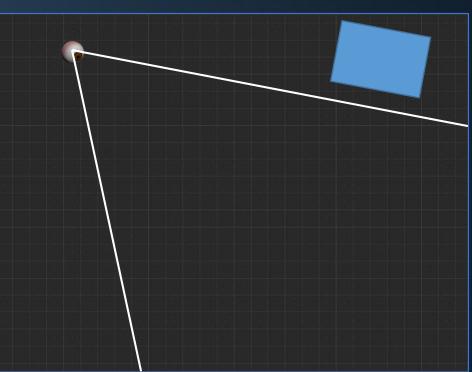
andom walk - done properly, closely following : /ive)

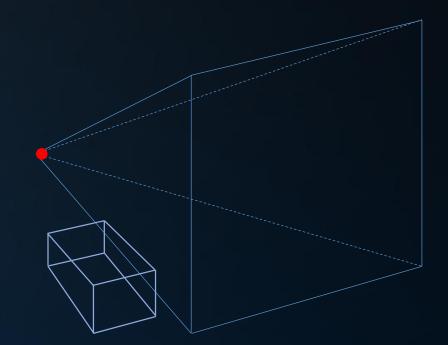
, H33 brdf = SampleDiffuse(diffuse, N, F1, F2, RR, S, H prvive; pdf; h = E * brdf * (dot(N, R) / pdf); sion = true:

Culling a Bounding Box

A bounding box is outside the view frustum when:

• All it's vertices on the backside of a single plane.







Groupings

tice ≰ (depth (⊂1935

: = inside / 1 it = nt / nc, ddo os2t = 1.0f - ont 0, N); 3)

at a = nt - nc, b + nt + + at Tr = 1 - (R0 + (1 - 1) Tr) R = (D * nnt - 1 -

E * diffus: = true;

efl + refn)) && (depth is MARDIT

D, N); -efl * E * diffuse; = true;

AXDEPTH)

v = true; at brdfPdf = EvaluateDiffuse(L, N,) * Psu st3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf)

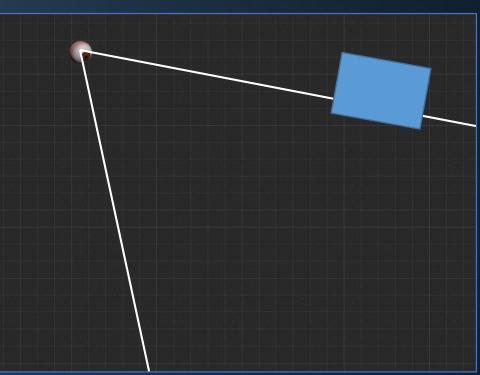
andom walk - done properly, closely following : /ive)

, H33 brdf = SampleDiffuse(diffuse, N, F1, F2, RR, F2, prvive; pdf; h = E * brdf * (dot(N, R) / pdf); sion = true:

Culling a Bounding Box

A bounding box is outside the view frustum when:

• All it's vertices on the backside of a single plane.







Groupings

tic: k (depth < 100

= inside / L it = nt / nc, dde os2t = 1.0f - nnt D, N); B)

at a = nt - nc, b + nt + + at Tr = 1 - (R0 + (1 - - - -Tr) R = (D * nnt - N * - - -

= diffuse; = true;

: :fl + refr)) && (depth k HADIII

D, N); ~efl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(difference estimation - doing it property ff; radiance = SampleLight(%rand, I e.x + radiance.y + radiance.z) > 0)

v = true; at brdfPdf = EvaluateDiffuse(L, N,) * Provident st3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) * 0

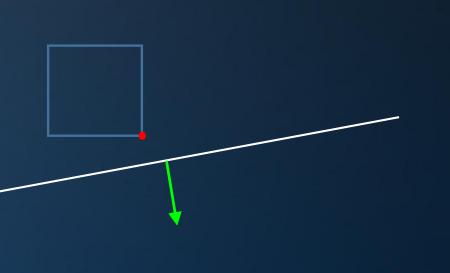
andom walk - done properly, closely following /ive)

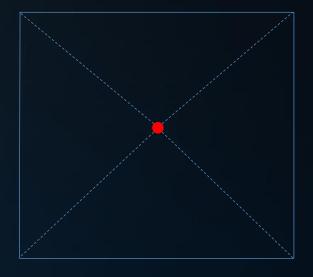
; pt3 brdf = SampleDiffuse(diffuse, N, r1, r2, NR, Soci pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Culling a Bounding Box

Instead of checking all eight vertices, we can limit the test to a single vertex.

- If $N_x > 0$, we use bmax.x, else bmin.x;
- If $N_y > 0$, we use bmax.y, else bmin.y;
- If $N_z > 0$, we use bmax.z, else bmin.z.







Groupings

tic: ⊾ (depth < 100

= inside / 1 it = nt / nc, dde ss2t = 1.0f - nnt), N); 3)

st a = nt - nc, b - nt st Tr = 1 - (80 + (1 Tr) R = (0 * nnt - N *

= diffuse = true;

-:fl + refr)) && (depth & HANDITT

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

AXDEPTH)

survive = SurvivalProbability difference estimation - doing it property if; adiance = SampleLight(%rand, I = x + radiance.y + radiance.z) > 0) %

v = true; at brdfPdf = EvaluateDiffuse(L, N) * Punn st3 factor = diffuse * INVPI; st weight = Mis2(directPdf, brdfPdf); st cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) *

andom walk - done properly, closely f /ive)

; ot3 brdf = SampleDiffuse(diffuse, N, r1, r2, UR, Dot prvive; pdf; a = E * brdf * (dot(N, R) / pdf); sion = true;

Culling a Bounding Box

What about the problematic case?

- 1. Our test is a conservative test; i.e. it will produce *false negatives*, but no false positives.
- 2. We can improve accuracy (at the cost of extra calculations) by reversing roles: use the planes of the AABB to cull the frustum.

Note that this still leaves certain tricky cases. For a perfect solution, check: <a href="http://gamedev.stackexchange.com/questions/44500/how-many-and-which-axes-to-use-for-3d-obb-collision-with-sates-to-use-for-



tice (depth is two

= inside / 1 it = nt / nc, ddo os2t = 1.0f - nnt 0; N(); 3)

at a = nt - nc, b - nt - at Tr = 1 - (R0 + 1 fr) R = (D * nnt - N

= diffuse; = true;

-:fl + refr)) && (depth k HAND)

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

AXDEPTH)

survive = SurvivalProbability difference estimation - doing it property if; adiance = SampleLight(&rand, I 2.x + radiance.y + radiance.z)

v = true; at brdfPdf = EvaluateDiffuse(L, N.) * Pauro st3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) * read

indom walk - done properly, closely following -/ive)

; pt3 brdf = SampleDiffuse(diffuse, N, r1, r2, UR, bod urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Today's Agenda:

- Introduction
- Boxes
- AABBs
- Groupings
- Efficiency
- To Rasterization





Efficiency

tice ⊾ (depth k 1935

= inside / 1 it = nt / nc, dde ss2t = 1.0f - nnt -5, N); 3)

st a = nt - nc, b - nt st Tr = 1 - (R0 + (1 fr) R = (D * nnt - N

= * diffuse; = true;

= efl + refr)) && (depth k HANDIIII

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

AXDEPTH)

survive = SurvivalProbability(difference estimation - doing it properly if; radiance = SampleLight(&rand, 1, 8, 8)

e.x + radiance.y + radiance.r) > 0) 68 (... w = true;

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

indom walk - done properly, closely following :
/ive)

; pt3 brdf = SampleDiffuse(diffuse, N, r1, r2, NR, SS pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Measuring Performance

Stopwatch class:

Using System.Diagnostics.Stopwatch;

Useful property:

long ElapsedMilliseconds { get; }

Methods:

- Reset
- Start
- Stop

Note:

Accuracy may vary. Measure lots of work, not a single line of code. Aim for tens of milliseconds, not nanoseconds.

Note:

Multithreading affects measurements. Profile single-threaded code; tune your multi-threading independently.

Note:

Use a profiler for more accuracy and detail. Try e.g. SlimTune, or Prof-It: <u>http://prof-it.sourceforge.net</u>



23

Efficiency

tic: k (depth < 100

: = inside / l ht = nt / nc, ddo os2t = 1.0f - nnt -D, N); B)

at a = nt - nc, b - nt - - at Tr = 1 - (R0 + (1 - - - -Tr) R = (0 * nnt - N - - - -

= diffuse; = true;

efl + refr)) && (depth (MAX)

D, N); ~efl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability difference estimation - doing it property ff; radiance = SampleLight(%rand, I e.x + radiance.y + radiance.z) > 0) %

v = true;

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

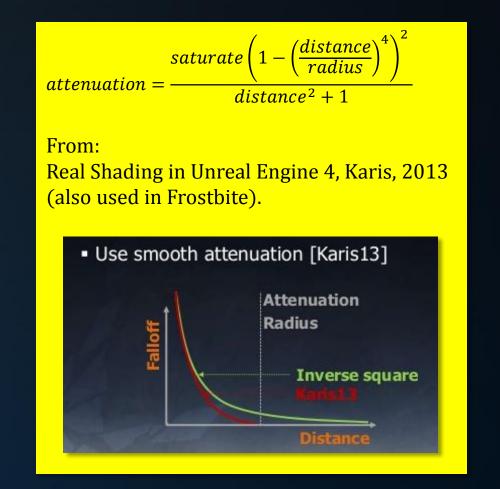
indom walk - done properly, closely following -/ive)

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

Optimization Primer

Some things to keep in mind:

- Float or double
- Don't do work you don't need to do
 - Early out
 - Reduce precision
 - Lights with finite radius
 - Things that can't occlude light





Efficiency

tice ≰ (depth < 10.5

= inside / 1 it = nt / nc, dde ss2t = 1.0f - nnt -), N); 3)

at a = nt - nc, b = n1 at Tr = 1 - (R0 + 15 Tr) R = (D * nnt - N

= diffuse; = true;

≕ efl + refr)) 88 (depth k HANDIII

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

AXDEPTH)

survive = SurvivalProbability difference estimation - doing it properly if; radiance = SampleLight(%rand, I & .x + radiance.y + radiance.z) > 0) %

v = true;

st brdfPdf = EvaluateDiffuse(L, N.) Pauro bit st3 factor = diffuse " INVPI; st weight = Mis2(directPdf, brdfPdf); st cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) * (Pauro E * (weight * cosThetaOut) / directPdf) * (Pauro E * (weight * cosThetaOut) / directPdf) * (Pauro E * (weight * cosThetaOut) * (weight * cosThet

andom walk - done properly, closely following -/ive)

; pt3 brdf = SampleDiffuse(diffuse, N, F1, F2, R, F3, pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Optimization Primer

Some things to keep in mind:

- Float or double
- Don't do work you don't need to do
- Precalculate
 - Loop hoisting
 - Vertex shaders

 \rightarrow



Efficiency

tice € (depth < 100

: = inside / 1 it = nt / nc, dde os2t = 1.0f - nnt -), N); 3)

at a = nt - nc, b - nt at Tr = 1 - (R0 + (1 Tr) R = (D * nnt - n -

= diffuse; = true;

-: efl + refr)) && (depth k HAADII

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

AXDEPTH)

v = true; at brdfPdf = EvaluateDiffuse(L, N) * Pours) at 3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf)

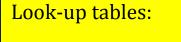
andom walk - done properly, closely following -/ive)

; pt3 brdf = SampleDiffuse(diffuse, N, r1, r2, UR, source) pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Optimization Primer

Some things to keep in mind:

- Float or double
- Don't do work you don't need to do
- Precalculate
- Expensive operations
 - sin, cos
 - sqrt
 - /
 - *
 - +,-
- * + -
- +, -



...

If you need sin/cos, it's often much faster to use a look-up table.

```
float sintab[3600], costab[3600];
for( int i = 0; i < 3600; i++ )
{</pre>
```

```
sintab[i] = Math.Sin( i / 10 );
costab[i] = Math.Cos( i / 10 );
```

```
float s = sintab[(int)(a * 10)];
float c = costab[(int)(a * 10)];
```



Efficiency

tice ≰ (depth < RAS

= inside / l it = nt / nc, dde os2t = 1.0f - nnt -D, N); B)

st a = nt - nc, b - nt st Tr = 1 - (R0 + (1 Tr) R = (D * nnt - N

= diffuse; = true;

: :fl + refr)) && (depth & MANDIIII

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

AXDEPTH)

survive = SurvivalProbability(difference estimation - doing it properly if; radiance = SampleLight(%rand, I, e.x + radiance.y + radiance.z) > 0) %%

v = true; at brdfPdf = EvaluateDiffuse(L, N) * Pour bast st3 factor = diffuse * INVPI; st weight = Mis2(directPdf, brdfPdf); st cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) * Context E * ((weight * cosThetaOut) / directPdf) * Context E * ((weight * cosThetaOut) / directPdf) * Context E * ((weight * cosThetaOut) / directPdf) * Context E * ((weight * cosThetaOut) / directPdf) * Context E * ((weight * cosThetaOut) / directPdf) * Context E * ((weight * cosThetaOut) / directPdf) * Context E * ((weight * cosThetaOut) / directPdf) * Context E * ((weight * cosThetaOut) / directPdf) * Context E * ((weight * cosThetaOut) / directPdf) * Context E * ((weight * cosThetaOut) * Context E * (weight * cosThetaOut) * (

andom walk - done properly, closely following -/ive)

; pt3 brdf = SampleDiffuse(diffuse, N, r1, r2, NR, Sch pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Optimization Primer

Some things to keep in mind:

- Float or double
- Don't do work you don't need to do
- Precalculate
- Expensive operations
- Programming Language
 - C#/C++
 - C++/Asm



Efficiency

tice € (depth < 1000

= inside / 1 ht = nt / nc, dde os2t = 1.0f - nnt 0, N); 3)

at a = nt - nc, b = nt = at Tr = 1 - (R0 + (1 Tr) R = (D * nnt - N

= diffuse; = true;

efl + refr)) && (depth & NADIII

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

AXDEPTH)

survive = SurvivalProbability difference estimation - doing it property ff; radiance = SampleLight(%rand, I, %) e.x + radiance.y + radiance.z) > 0)

w = true; at brdfPdf = EvaluateDiffuse(L, N) P at3 factor = diffuse = INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf

andom walk - done properly, closely following : /ive)

; ot3 brdf = SampleDiffuse(diffuse, N, r1, r2, NR, Npd) prvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true;

Perceived Performance

Incremental Rendering

- 1. Real-time preview:
- Depth map
- Depth map plus materials
- Render without recursive reflections
- Render with very limited recursion

Still not real-time?

- Render half-res
- Adaptive resolution
 - Optimize the application a bit



Efficiency

tic: k (depth < 100

: = inside / l it = nt / nc, dde os2t = 1.0f - nnt -D, N); B)

at a = nt - nc, b + nt + + at Tr = 1 - (80 + (1 - - - - fr) R = (0 * nnt - N * - - -

= diffuse; = true;

≕ efl + refr)) && (depth < HANDII

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

AXDEPTH)

survive = SurvivalProbability(difference estimation - doing it property ff; radiance = SampleLight(%rand, I e.x + radiance.y + radiance.z) > 0) %

w = true; t brdfPdf = EvaluateDiffuse(L, N) Promote st3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf)

andom walk - done properly, closely following -/ive)

; st3 brdf = SampleDiffuse(diffuse, N, r1, r2, R, staturvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Perceived Performance

Incremental Rendering

- 2. Stationary camera:
- Render with normal recursion

Keep the application responsive:

Render lines of pixels until a certain number of milliseconds has passed; continue in the next frame.



Efficiency

tice ⊾ (depth < 1000

: = inside / 1 it = nt / nc, ddo os2t = 1.0f - nnt -O, N); 3)

st a = nt - nc, b = nt = st Tr = 1 - (80 + (1 Tr) R = (0 * nnt - N

= diffuse; = true;

-:fl + refr)) && (depth k HANDII

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

AXDEPTH)

survive = SurvivalProbability(different estimation - doing it properly if; radiance = SampleLight(&rand, I, I, I) e.x + radiance.y + radiance.z) > 0) 55

v = true; at brdfPdf = EvaluateDiffuse(L, N) Prost3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf)

indom walk - done properly, closely following : vive)

; st3 brdf = SampleDiffuse(diffuse, N, r1, r2, UR, sr urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Perceived Performance

Incremental Rendering

- 3. 'Photograph mode':
- Invoked with a key
- Render with extreme recursion
- Use anti-aliasing
- Add screenshot feature

Keep the application responsive!



tice (depth is two

= inside / 1 it = nt / nc, ddo os2t = 1.0f - nnt 0; N(); 3)

at a = nt - nc, b - nt - at Tr = 1 - (R0 + 1 fr) R = (D * nnt - N

= diffuse; = true;

-:fl + refr)) && (depth k HAND)

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

AXDEPTH)

survive = SurvivalProbability difference estimation - doing it property if; adiance = SampleLight(&rand, I 2.x + radiance.y + radiance.z)

v = true; at brdfPdf = EvaluateDiffuse(L, N.) * Pauro st3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) * read

indom walk - done properly, closely following -/ive)

; pt3 brdf = SampleDiffuse(diffuse, N, r1, r2, UR, bod urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Today's Agenda:

- Introduction
- Boxes
- AABBs
- Groupings
- Efficiency
- To Rasterization





Rasterization

tice (depth (1935

= inside / 1 tt = nt / nc, dde 552t = 1.0f = nnt -5, N); 8)

= diffuse; = true;

-:fl + refr)) && (depth K MADI

), N); refl * E * diffi = true;

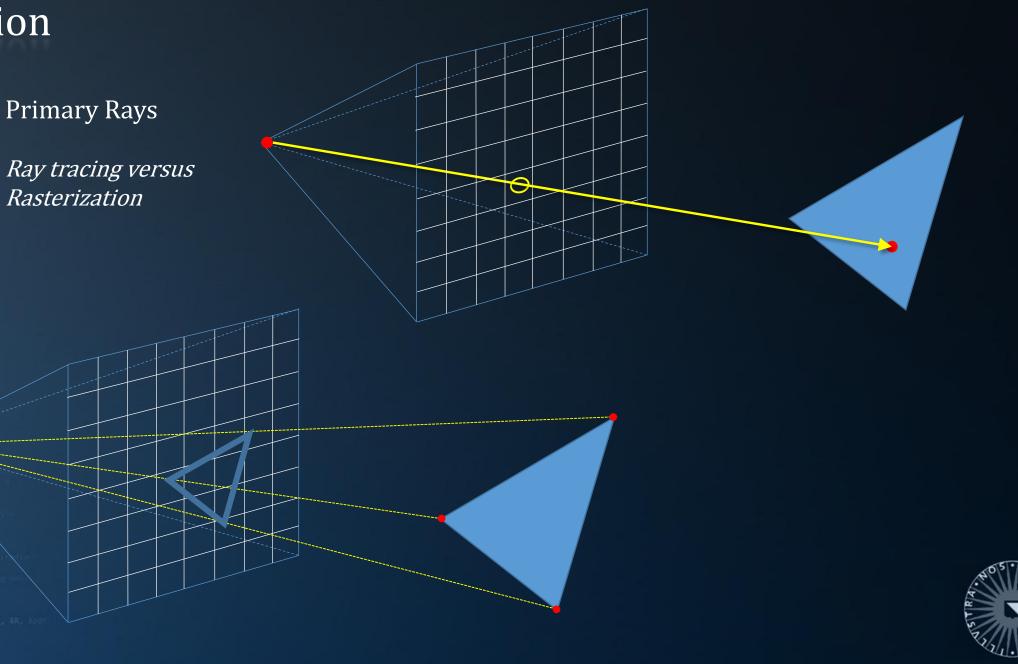
AXDEPTH)

survive = SurvivalProbabi estimation - doing it pr f; radiance = SampleLight(%rand e.x + radiance.y + radiance.z)

w = true; ot brdfPdf = EvaluateDiffuse(L, N) st3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf)

andom walk - done properly, closely follow: /ive)

; t3 brdf = SampleDiffuse(diffuse, N, r1, r2, 48, 4; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:



Rasterization

tica k (depth < 1825

= inside / 1 tt = nt / nc. dde -552t = 1.0f = nnt -5, N); 8)

st a = nt - nc, b = mt + s st Tr = 1 - (R0 + (1 Tr) R = (D * nnt - N *

= diffuse; = true;

-:fl + refr)) && (depth & HADIII

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

AXDEPTH)

survive = SurvivalProbabile estimation - doing it proif; radiance = SampleLight(&rand, e.x + radiance.y + radiance.z)

w = 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: /ive)

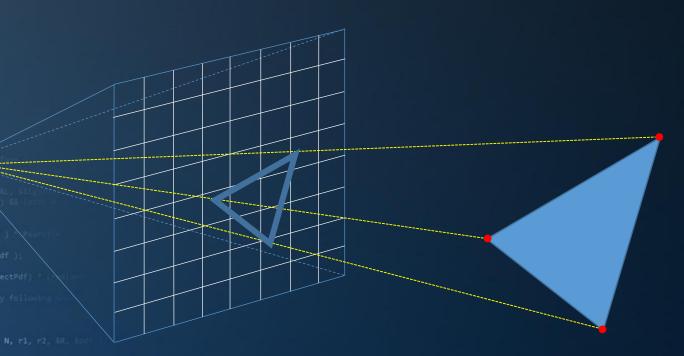
; tt3 brdf = SampleDiffuse(diffuse, N, F1, F2, SR. prvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

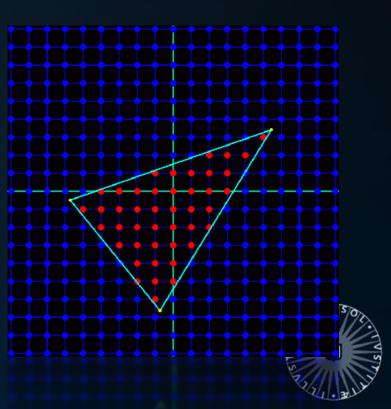
Primary Rays

Ray tracing versus Rasterization

Rasterization:

- 1. Transform primitive into camera space
- 2. Project vertices into 2D screen space
- 3. Determine which pixels are affected
- 4. Use z-buffer to sort (pixels of) primitives
- 5. Clip against screen boundaries





Rasterization

tice ⊾ (depth ∈ NASS

= inside / 1 it = nt / nc, dde ss2t = 1.0f - nnt 3, N); 3)

st a = nt - nc, b - nt - st Tr = 1 - (R0 + 1 Tr) R = (0 * nnt - N

= diffuse; = true;

efl + refr)) && (depth & MANDIAN

D, N); ~efl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability difference estimation - doing it properly if; adiance = SampleLight(&rand I = 1) e.x + radiance.y + radiance.r) = 0.000

v = true; at brdfPdf = EvaluateDiffuse(L, N) * Purple st3 factor = diffuse * INVPI; st weight = Mis2(directPdf, brdfPdf); st cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) * Purple

andom walk - done properly, closely following a /ive)

; pt3 brdf = SampleDiffuse(diffuse, N, r1, r2, NR, brd pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Shadow Rays

The rasterization pipeline renders triangles one at a time.

- Shading calculations remain the same
- But determining light visibility is non-trivial.

Rasterization does not have access to *global data*.



34

Rasterization

tica ⊾ (depth ic 1925

= inside / L it = nt / nc, dde -552t = 1.0f - nnt -5, N); 3)

st a = nt - nc, b - nt - ... st Tr = 1 - (R0 + (1 - ... fr) R = (D * nnt - N *

= diffuse; = true;

-:fl + refr)) && (depth k HAODIII

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

AXDEPTH)

survive = SurvivalProbability(difference estimation - doing it properly if; radiance = SampleLight(&rand, I, I, I, e.x + radiance.y + radiance.z) = 0)

v = true; ot brdfPdf = EvaluateDiffuse(L, N.) Pours st3 factor = diffuse * INVPI; ot weight = Mis2(directPdf, brdfPdf); st cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf)

andom walk - done properly, closely following: /ive)

; pt3 brdf = SampleDiffuse(diffuse, N, F1, F2, UR, S pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Spaces

Ray tracing typically happens in a single 3D coordinate system.

In rasterization, we use many coordinate systems:

- Camera space
- Clip space

- 2D screen space
 - Model space
- Tangent space

We need efficient tools to get from one space to another. We will make extensive use of matrices to do this.



Rasterization

tica ≰ (depth < 10.5

= inside / 1 it = nt / nc, dde os2t = 1.0f - nnt -D, N); B)

at a = nt - nc, b - nt - at Tr = 1 - (R0 - - - -Fr) R = (D * nnt - N

= diffuse; = true;

-:fl + refr)) && (depth & HADDIN

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

AXDEPTH)

survive = SurvivalProbability(difference estimation - doing it properly if; adiance = SampleLight(%rand, I, Market e.x + radiance.y + radiance.z) > 0) %

v = true;

st brdfPdf = EvaluateDiffuse(L, N.) Pauro bit st3 factor = diffuse * INVPI; st weight = Mis2(directPdf, brdfPdf); st cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) * (Paulon);

andom walk - done properly, closely following -/ive)

; t33 brdf = SampleDiffuse(diffuse, N, r1, r2, NR, hpt urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Common Concepts

Many things remain the same:

- Normal interpolation
- Shading
- Texture mapping
- The camera
- Boxes.



tice (depth is two

= inside / 1 it = nt / nc, ddo os2t = 1.0f - nnt 0; N(); 3)

at a = nt - nc, b - nt - at Tr = 1 - (R0 + 1 fr) R = (D * nnt - N

= diffuse; = true;

-:fl + refr)) && (depth k HAND)

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

AXDEPTH)

survive = SurvivalProbability difference estimation - doing it property if; adiance = SampleLight(&rand, I 2.x + radiance.y + radiance.z)

v = true; at brdfPdf = EvaluateDiffuse(L, N.) * Pauro st3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) * radii

indom walk - done properly, closely following -/ive)

; pt3 brdf = SampleDiffuse(diffuse, N, r1, r2, UR, bod urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Today's Agenda:

- Introduction
- Boxes
- AABBs
- Groupings
- Efficiency
- To Rasterization





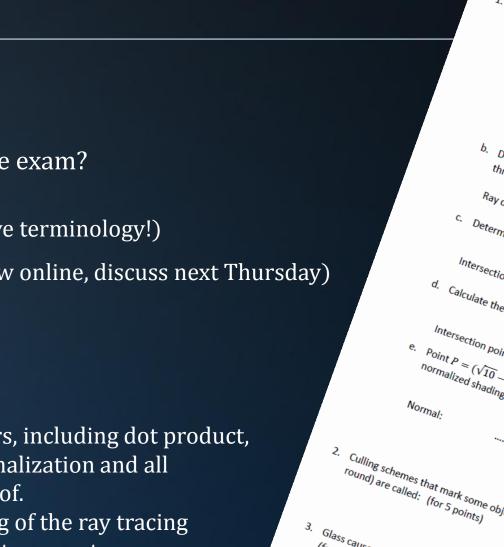
Mid-term Exam

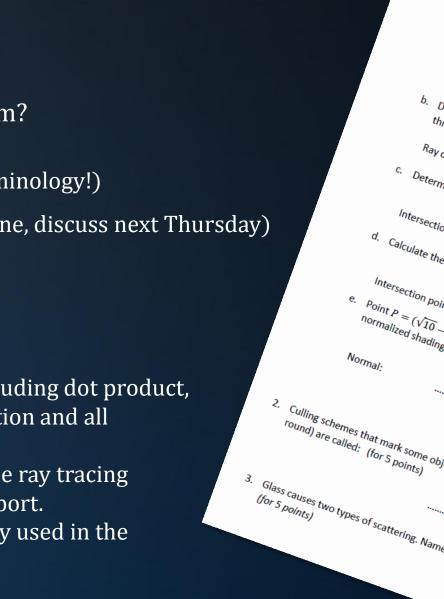
What to study for the exam?

- 1. Slides (mind cursive terminology!)
 - Example exam (now online, discuss next Thursday) 2.
 - 3. Tutorial sheets

Expectations:

- Fluency with vectors, including dot product, cross product, normalization and all combinations thereof.
- Good understanding of the ray tracing algorithm and light transport.
- Knowledge of terminology used in the lectures.





Given: an eve position E = (1,1,1), a view vector $\vec{V} = (1,0,1)$, an up vector $\vec{v} = (1,0,1)$, an up vector $\vec{v} = (1,0,1)$, an up vector $\vec{v} = (1,0,1)$, $\vec{v} = (1$ Given: an eye position $\mathcal{E} = (1,1,1)$, a view vector $\mathcal{V} = (1,0,1)$, an up vector $\mathcal{I} = (1,0,1)$, an up vector $\mathcal{I} = (1,0,1)$, an up vector $\mathcal{I} = (1,0,1)$, $\mathcal{I} =$ a. For the purpose of ray tracing, calculate the four corners of a virtual screen of the second state the four corners of a virtual screen of the second state the four corners of a virtual screen of the second state the second For the purpose of ray tracing, canculate the root control of the purpose of ray tracing, canculate the root control of the perpendicular to \vec{V} and \vec{kp} , taking into account the spectrum. b. Determine the normalized direction of ray r, which originates from E and extend : Determine the 3D coordinate of the intersection of r with the plane: $-X - Z \approx 10$. Intersection point: d. Calculate the intersection of r with the sphere $||p - C|| = \sqrt{20}$, where C = (-1, 1, -1). e. Point $P = (\sqrt{10} - 1, 1, \sqrt{10} - 1)$ is a point on the sphere $||_{P} - C|| = \sqrt{20}$. Calculate the inside of the sphere. 2. Culling schemes that mark some objects as visible when they are not (but not the other way · Glass causes two types of scattering. Name both. First type: Second type:

st a = nt

), N); -efl * E * diffuse;

AXDEPTH)

survive = SurvivalProbability(d) if: adiance = SampleLight(&rand x + radiance.v + radiance.z

v = true: at brdfPdf = EvaluateDiffuse

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

andom walk - done properly, closely fol /ive)

at3 brdf = SampleDiffuse(diffuse, N, r1, r2, NR) rvive; pdf; i = E * brdf * (dot(N, R) / pdf); sion = true:

tics ⊾ (depth ⊂ NA

s = inside / 1 it = nt / nc, dd os2t = 1.0f 0, N); 3)

st $a = nt - nc_{1} b - nt$ st Tr = 1 - (80 + (1))Tr) R = (0 = nnt - n)

= diffuse = true;

-: :fl + refr)) && (depth k HANDIII

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

AXDEPTH)

survive = SurvivalProbability difference estimation - doing it properly if; adiance = SampleLight(@rand, I =.x + radiance.y + radiance.z) > 0)

v = true; t brdfPdf = EvaluateDiffuse(L, N,) Process st3 factor = diffuse * INVPI; st weight = Mis2(directPdf, brdfPdf); st cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf)

andom walk - done properly, closely following a /ive)

; pt3 brdf = SampleDiffuse(diffuse, N, r1, r2, NR pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

INFOGR – Computer Graphics

Jacco Bikker - April-July 2016 - Lecture 6: "Boxes"

END of "Boxes"

next lecture: "Acceleration Structures"

