INFOGR – Computer Graphics

Jacco Bikker & Debabrata Panja - April-July 2018

Lecture 13: "Visibility"

Welcome!



hics & (depth < MOXDEF

: = inside ? 1 ht = nt / nc, ddn bs2t = 1.0f - nnt ° n D, N); ≫)

at a = nt - nc, b = nt + r at Tr = 1 - (R0 + (1 - R0 Tr) R = (D = nnt - N = (000

= * diffuse = true;

efl + refr)) && (depth < MOXDEPT

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

AXDEPTH)

survive = SurvivalProbability(diffuse .estimation - doing it properly, if; radiance = SampleLight(&rand, I, &L, &II) 2.x + radiance.y + radiance.z) > 0) & _____

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

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

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

$T_{world} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ Do nothing, or: 1. $T_c = inv(T_{cam})$ $T_w = identity = I$ 2. Render 'world' with I, T_c world Render children 3. at a = nt T_{car2} Γ_{plane2} plane1 $\boldsymbol{T_{plane}} = \begin{bmatrix} 1 & 0 & 0 & x_{plane} \\ 0 & 1 & 0 & y_{plane} \\ 0 & 0 & 1 & z_{plane} \\ 0 & 0 & 0 & 1 \end{bmatrix}$ $T_w = T_{local} * T_{w_parent}$ 2. $T_w = T_{local} * T_{w_parent}$ plane plane car), N = true

AXDEPTH)

- survive = SurvivalProbability(diff lf;
- radiance = SampleLight(&rand, I, & .x + radiance.y + radiance.z)
- 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: /ive)

```
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, 8R,
urvive;
pdf;
1 = E * brdf * (dot( N, R ) / pdf);
sion = true:
```

void SGNode::Render(mat4 $T_{wparent}$, mat4 $T_{cparent}$) mat4 $T_w = T_{local} * T_{wparent}$; mat4 $T_c = T_{local} * T_{cparent}$; mesh.Draw(T_w , T_c); for each child: Draw(T_w , T_c);



- 3. Render 'plane' with T_w , T_c
- Render children 4.

To camera space:



To world space:

ics & (depth < ™xxxxx

: = inside ? 1 ht = nt / nc, ddn os2t = 1.0f - nnt 0, N); 0)

at a = nt - nc, b = nt = nc at Tr = 1 - (R0 + (1 - R0 Γ r) R = (D = nnt - N (300)

= * diffuse; = true;

efl + refr)) && (depth < MAXDEDID

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

AXDEPTH)

survive = SurvivalProbability(diffuse)
estimation - doing it properly.closed
if;
radiance = SampleLight(&rand, I, &L, &light
e.x + radiance.y + radiance.z) > 0) && doct

v = true;

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

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

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

Today's Agenda:

- Depth Sorting
- Clipping
- Visibility



tics & (depth < ⊅xxxxx

c = inside ? 1 ht = nt / nc, ddn bs2t = 1.0f - nnt D, N); B)

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

= * diffuse; = true;

efl + refr)) && (depth < MAXDEF

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

AXDEPTH)

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



Rendering – Functional overview

- 1. Transform: translating / rotating meshes
- 2. Project: calculating 2D screen positions
- 3. Rasterize: determining affected pixels
- 4. Shade: calculate color per affected pixel





Depth Sorting

-), N); refl * E * diffuse;
- AXDEPTH)
- survive = SurvivalProbability(diff lf: radiance = SampleLight(&rand, I, e.x + radiance.y + radiance.z) > 0



3. Rasterize:

affected pixel?

Questions:

determining affected pixels

Is that position actually on-screen?

Is the fragment the nearest fragment for the



Postprocessing



5

Part of the tree is off-screen

Too far away to draw

Tree requires little detail

City obscured by tree

Torso closer than ground

ZITU

Tree between ground & sun

B F

M ?

Depth Sorting

Old-skool depth sorting: Painter's Algorithm

- Sort polygons by depth
- Based on polygon center
- Render depth-first

Advantage:

Doesn't require z-buffer

Problems:

Cost of sorting Doesn't handle all cases Overdraw







at Tr = 1

AXDEPTH)

lf;

survive = SurvivalProbability(dif

radiance = SampleLight(&rand,

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

Depth Sorting

Overdraw:

ics (depth < Notice in

: = inside | | ht = nt / nc, ddn bs2t = 1.0f - nnt D, N); D)

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

= * diffuse = true;

efl + refr)) && (depth < MAXDEPTH

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

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly if; radiance = SampleLight(&rand, I, &L 2.x + radiance.y + radiance.z) > 0) &&



Inefficiency caused by drawing multiple times to the same pixel.



Depth Sorting

Overdraw:

at a = nt

AXDEPTH)

survive = SurvivalProbability(diffu radiance = SampleLight(&rand, I, &L,) e.x + radiance.y + radiance.z) > 0) 8a



Inefficiency caused by drawing multiple times to the same pixel.



9



Depth Sorting

Overdraw:

at a = nt

AXDEPTH)

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



Inefficiency caused by drawing multiple times to the same pixel.





Depth Sorting

Overdraw:

Inefficiency caused by drawing multiple times to the same pixel.

), N); refl * E * diffus

AXDEPTH)

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





Correct order: BSP

tics & (depth < Motocr

z = inside / 1 ht = nt / nc, ddn bs2t = 1.0f - nnt - n D, N); D)

at a = nt - nc, b = nt - nc at Tr = 1 - (R0 + (1 - R0 Fr) R = (D - nnt - N - (300

= * diffuse; = true;

efl + refr)) && (depth < MAXDEPTI

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

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly, if; radiance = SampleLight(&rand, I, &L, &L() 2.x + radiance.y + radiance.z) > 0) && ()



root



Correct order: BSP



AXDEPTH)

survive = SurvivalProbability(diffu radiance = SampleLight(&rand, I, &L, &l e.x + radiance.y + radiance.z) > 0) 88





13

Correct order: BSP



-•fl + refr)) && (depth < MAA

9, N); efl * E * diffuse = true;

AXDEPTH)

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





Depth Sorting Correct order: BSP root back front efl * E * diffuse; AXDEPTH) survive = SurvivalProbability(diffu radiance = SampleLight(&rand, I, &L, &l

15

on = true:

Depth Sorting Correct order: BSP root back front efl * E * diffuse/ AXDEPTH) survive = SurvivalProbability(dif radiance = SampleLight(&rand, I, &L, &

16



17

Depth Sorting Correct order: BSP root back front efl * E * diffuse AXDEPTH) survive = SurvivalProbability radiance = SampleLight(&rand, I, & .x + radiance.y + radiance.z Sorting by BSP traversal: Recursively Render far side of plane 1.

2. Render near side of plane



Depth Sorting

Draw order using a BSP:

- Guaranteed to be correct (hard cases result in polygon splits)
- No sorting required, just a tree traversal

But:

- Requires construction of BSP: not suitable for dynamic objects
- Does not eliminate overdraw

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

AXDEPTH)

at a = nt

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



19



on = true:

nics & (depth < MOXDOFT

: = inside ? 1 ht = nt / nc, ddn ss2t = 1.0f - nnt " n 2, N); ≫)

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

= * diffuse; = true;

efl + refr)) && (depth < MAXDEPTH

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

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly closed H; radiance = SampleLight(&rand, I, &L, &Light) 2.x + radiance.y + radiance.z) > 0) &&



Z-buffer

A z-buffer stores, per screen pixel, a depth value. The depth of each fragment is checked against this value:

If the fragment is further away, it is discarded

• Otherwise, it is drawn, and the z-buffer is updated.

The z-buffer requires:

- An additional buffer
- Initialization of the buffer to *z_{max}*
- Interpolation of *z* over the triangle
- A z-buffer read and compare, and possibly a write.





Z-buffer

What is the best representation for depth in a z-buffer?

- 1. Interpolated z (convenient, intuitive);
- 2. 1/z (or: $n + f \frac{fn}{z}$) (more accurate nearby);
- 3. $(int)((2^31-1)/z);$
- 4. (uint)((2^32-1)/-z);
- 5. $(uint)((2^32-1)/(-z+1)).$

Note: we use $z_{int} = \frac{(2^{32}-1)}{-z+1}$: this way, any z < 0 will be in the range $z_{adjusted} = -z_{original} + 1 = 1..\infty$, therefore $1/z_{adjusted}$ will be in the range 0..1, and thus the integer value we will store uses the full range of $0..2^{32} - 1$. Here, $z_{int} = 0$ represents $z_{original} = 0$, and $z_{int} = 2^{32} - 1$ represents $z_{original} = -\infty$.

Even more details:

https://developer.nvidia.com/content/depth-precision-visualized http://outerra.blogspot.nl/2012/11/maximizing-depth-buffer-range-and.html



= true:

), N);

= true;

AXDEPTH)

fl + refr)) && (dept)

survive = SurvivalProbability(dif

radiance = SampleLight(&rand, I

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

22

Depth Sorting

Z-buffer optimization

hics & (depth < Monor

: = inside ? 1 1 1 1 ht = nt / nc, ddn - 1 ps2t = 1.0f - nmt - 1 D, N); 3)

at a = nt - nc, b = nt - n at Tr = 1 - (R0 + (1 - R0) Fr) R = (D - nnt - N - (100)

= * diffuse = true;

efl + refr)) && (depth < MAXDEPID

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

AXDEPTH)

survive = SurvivalProbability(diffuse .estimation - doing it properly if; radiance = SampleLight(&rand, I, &L, &light) 2.x + radiance.y + radiance.z) > 0) && (doing)



In the ideal case, the nearest fragment for a pixel is drawn first:

- This causes all subsequent fragments for the pixel to be discarded;
- This minimizes the number of writes to the frame buffer and z-buffer.

The ideal case can be approached by using Painter's to 'pre-sort'.



Depth Sorting

nics & (depth < Mooss

c = inside ? 1 1 1 1 ht = nt / nc, ddn ss2t = 1.0f - nnt " n D, N); B)

at a = nt - nc, b = nt at Tr = 1 - (R0 + (1 - R0 Fr) R = (D - nnt - N - (10)

= * diffuse; = true;

efl + refr)) && (depth < MODEPTH

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

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly if; radiance = SampleLight(&rand, I, &L, &light) 2.x + radiance.y + radiance.z) > 0) && (doing)



'Z-fighting':

Occurs when two polygons have almost identical z-values.

Floating point inaccuracies during interpolation will cause unpleasant patterns in the image.





Part of the tree is off-screen

Stuff that is too far to draw

Tree requires little detail

$\sqrt{}$ City obscured by tree

35

Tree between ground & sun

B F

八

M

5

ics & (depth < ™xxxxx

: = inside ? 1 ht = nt / nc, ddn os2t = 1.0f - nnt 0, N); 0)

at a = nt - nc, b = nt = nc at Tr = 1 - (R0 + (1 - R0 Γ r) R = (D = nnt - N (300)

= * diffuse; = true;

efl + refr)) && (depth < MAXDEDID

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

AXDEPTH)

survive = SurvivalProbability(diffuse)
estimation - doing it properly.closed
if;
radiance = SampleLight(&rand, I, &L, &light
e.x + radiance.y + radiance.z) > 0) && doct

v = true;

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

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

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

Today's Agenda:

- Depth Sorting
- Clipping
- Visibility



nics & (depth < Moonant

: = inside ? 1 = 1 00 ht = nt / nc, ddn = 0 s2t = 1.0f - nnt = n 2, N); ≫)

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

= * diffuse; = true;

efl + refr)) && (depth < MOXDEP

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

AXDEPTH)

survive = SurvivalProbability(diffuse)
estimation - doing it properly.closed
if;
radiance = SampleLight(&rand, I, &L, &light(&rand, &rand,

v = true;

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

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

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

Clipping

Many triangles are partially off-screen. This is handled by *clipping* them.

Sutherland-Hodgeman clipping:

Clip triangle against 1 plane at a time; Emit *n*-gon (0, 3 or 4 vertices).





nics & (depth < Mocoo

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

= * diffuse; = true;

• efl + refr)) && (depth < MA

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

AXDEPTH)

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

v = true;

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

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

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

Sutherland-Hodgeman

Input: list of vertices

Algorithm:

Per edge with vertices v_0 and v_1 :

- If v_0 and v_1 are 'in', emit v_1
- If v₀ is 'in', but v₁ is 'out', emit C
- If v₀ is 'out', but v₁ is 'in', emit C and v₁

where C is the intersection point of the edge and the plane.

2

in

Output: list of vertices, defining a convex n-gon.

Vertex 0Vertex 1Vertex 1Intersection 1Vertex 2Intersection 2Vertex 0Vertex 0

out





ics & (depth < ≫occo

: = inside ? 1 ; 1 ; ht = nt / nc, ddn os2t = 1.0f - nmt * nm 0, N); ∂)

at a = nt - nc, b = nt + n at Tr = 1 - (R0 + (1 - R0 Fr) R = (D ⁺ nnt - N ⁻ (dd

= * diffuse; = true;

efl + refr)) && (depth < MAXDED)

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

AXDEPTH)

survive = SurvivalProbability(diffus estimation - doing it properly, clo H;

v = true;

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

andom walk - done properly, closely following Sa

/ive)

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

Sutherland-Hodgeman

Calculating the intersections with plane ax + by + cz + d = 0:

$$dist_{v} = v \cdot \begin{pmatrix} a \\ b \\ c \end{pmatrix} + d$$

$$= \frac{|dist_{v0}|}{|dist_{v0}| + |dist_{v1}|}$$

$$= v_0 + f(v_1 - v_0)$$



After clipping, the input n-gon may have at most 1 extra vertex. We may have to triangulate it:

 $0,1,2,3,4 \rightarrow 0, 1, 2 + 0, 2, 3 + 0, 3, 4.$



ics & (depth < Modern

: = inside ? 1 ()) ht = nt / nc, ddn ss2t = 1.0f - n⊓t *) 2, N); ≫)

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

= * diffuse; = true;

efl + refr)) && (depth < MAXDE

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

AXDEPTH)

survive = SurvivalProbability(diffus
estimation - doing it properly, class
if;
radiance = SampleLight(&rand, I, &L,

e.x + radiance.y + radiance.z) > 0) 85 (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 Sec /ive)

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

Guard bands

To reduce the number of polygons that need clipping, some hardware uses *guard bands* : an invisible band of pixels outside the screen.

- Polygons outside the screen are discarded, even if they touch the guard band;
- Polygons partially inside, partially in the guard band are drawn without clipping;
 - Polygons partially inside the screen, partially outside the guard band are clipped.





Clipping

ics & (depth < MODE)

: = inside ? l | ... ht = nt / nc, ddn - ... bs2t = 1.0f - nnt - ... D, N); D)

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

= * diffus = true;

. efl + refr)) && (depth < MAXDEPT

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

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly, closed ff; radiance = SampleLight(&rand, I, &L, &L) e.x + radiance.y + radiance.z) > 0) &&

v = true;

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

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

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

Sutherland-Hodgeman

Clipping can be done against arbitrary planes.









ics & (depth < ™xxxxx

: = inside ? 1 ht = nt / nc, ddn os2t = 1.0f - nnt 0, N); 0)

at a = nt - nc, b = nt = nc at Tr = 1 - (R0 + (1 - R0 Γ r) R = (D = nnt - N (300)

= * diffuse; = true;

efl + refr)) && (depth < MAXDEDID

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

AXDEPTH)

survive = SurvivalProbability(diffuse)
estimation - doing it properly.closed
if;
radiance = SampleLight(&rand, I, &L, &light
e.x + radiance.y + radiance.z) > 0) && doct

v = true;

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

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

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

Today's Agenda:

- Depth Sorting
- Clipping
- Visibility



/ Part of the tree is off-screen

Stuff that is too far to draw

Tree requires little detail

$\sqrt{}$ City obscured by tree

Tree between ground & sun

BF

八

M

5





Visibility

at Tr = 1

= true:

), N);

AXDEPTH)

survive = SurvivalProbability(dif lf: radiance = SampleLight(&rand, I .x + radiance.y + radiance.z)

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: /ive)

at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, A urvive; pdf; 1 = E * brdf * (dot(N, R) / pdf); sion = true

Only rendering what's visible:

"Performance should be determined by visible geometry, not overall world size."

Do not render geometry outside the view frustum

- Better: do not *process* geometry outside frustum
- Do not render occluded geometry
 - Do not render anything more detailed than strictly necessary



nics & (depth < Mos⊡a

t = inside ? 1 ht = nt / nc, ddn = 0 os2t = 1.0f - nnt " n 2, N); ≥)

at a = nt - nc, b = nt at Tr = 1 - (R0 + (1 - R0 Γ r) R = (D = nnt - N - (3.5)

= * diffuse; = true;

. efl + refr)) && (depth < MAXDEPTH

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

AXDEPTH)

survive = SurvivalProbability(diffus estimation - doing it properly if; adiance = SampleLight(&rand, I, &L, 2.x + radiance.y + radiance.z) > 0) &

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

andom walk - done properly, closely following 30 /ive)

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

Culling

Observation: 50% of the faces of a cube are not visible.

On average, this is true for all meshes.

Culling 'backfaces':

Triangle: ax + by + cz + d = 0Camera: (x, y, z)Visible: fill in camera position in plane equation.

ax + by + cz + d > 0: visible.

Cost: 1 dot product per triangle.







tics & (depth < NACCO

c = inside ? 1 ()) ht = nt / nc, ddn ss2t = 1.0f - nnt *), N); »)

at a = nt - nc, b = nt at Tr = 1 - (R0 + (1 - R0) Fr) R = (D = nnt - N - (30)

= * diffuse; = true;

. efl + refr)) && (depth < MAXDEPIN

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

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly if; radiance = SampleLight(&rand, I, &L 2.x + radiance.y + radiance.z) > 0) &

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

andom walk - done properly, closely following 300. /ive)

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

Culling

Observation:

If the *bounding sphere* of a mesh is outside the view frustum, the mesh is not visible.

But also:

If the *bounding sphere* of a mesh intersects the view frustum, the mesh may be not visible.

View frustum culling is typically a *conservative test:* we sacrifice accuracy for efficiency.

Cost: 1 dot product per mesh.





ics & (depth < PVXCC

: = inside ? 1 : 1 : 2 ht = nt / nc, ddn os2t = 1.0f - nnt * 2, N); 3)

at a = nt - nc, b = nt - n at Tr = 1 - (R0 + (1 - R0) Fr) R = (D = nnt - N = (00)

= * diffuse; = true;

. :fl + refr)) && (depth < MAX

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

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly if; radiance = SampleLight(&rand, I, &L, & 2,x + radiance.y + radiance.z) > 0)

v = 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 30 /ive)

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

Culling

Observation:

If the *bounding sphere* over a group of bounding spheres is outside the view frustum, a group of meshes is invisible.

We can store a bounding volume hierarchy in the scene graph:

- Leaf nodes store the bounds of the meshes they represent;
- Interior nodes store the bounds over their child nodes.

Cost: 1 dot product per scene graph subtree.





nics & (depth < 200000

at a = nt - nc, b = nt - r at Tr = 1 - (R0 + (1 - R0 Tr) R = (D = nnt - N = (000

= * diffuse; = true;

. efl + refr)) && (depth < MAXDEPTH

D, N); refl * E * diffu: = true;

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly, closed if; radiance = SampleLight(&rand, I, &L, &light) 2.x + radiance.y + radiance.z) > 0_____

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

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

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

Culling

Observation:

If a grid cell is outside the view frustum, the contents of that grid cell are not visible.

Cost: 0 for out-of-range grid cells.





tics & (depth < Motos

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

= * diffuse; = true;

. efl + refr)) && (depth < NOCCEPTIO

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

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly, closed ff; radiance = SampleLight(&rand, I, &L, &L) e.x + radiance.y + radiance.z) > 0 = 20



Occlusion Culling

Not rendering things that are guaranteed to be behind something else.

Hierarchical z-buffer:

a set of MIP-maps of the z-buffer.

Use: with a small amount of tests, we can check the bounds of a mesh against this buffer.



Visibility

tics & (depth < No000

c = inside ? 1 ht = nt / nc, ddn os2t = 1.0f - nnt = 2, N); 3)

at a = nt - nc, b = nt at Tr = 1 - (R0 + (1 - R0 Tr) R = (D = nnt - N - (300

= * diffuse; = true;

efl + refr)) && (depth < MAXDEPTH

D, N); refl * E * diffu: = true;

AXDEPTH)

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



Occlusion Culling

Not rendering things that are guaranteed to be behind something else.

Coverage buffer:

A low-resolution version of (a simplified version of) the scene, rendered on the CPU, which we can use for visibility tests.



Flipcode IOTD, 2000: https://www.flipcode.com/archives/10-18-2000.shtml



Visibility

tics & (depth < ™0005

: = inside ? | : : : ht = nt / nc, ddn ss2t = 1.0f - nmt ? 2, N); ≫)

at a = nt - nc, b = nt at Tr = 1 - (R0 + (1 - R0 Tr) R = (D = nnt - N - (0.0

= * diffuse; = true;

efl + refr)) && (depth < MAXDEPTH

D, N); ∵efl * E * diffu = true;

AXDEPTH)

survive = SurvivalProbability(diffuse
estimation - doing it properly
f;
radiance = SampleLight(&rand, I, &L, &Llg)
e.x + radiance.y + radiance.z) > 0) &&



Occlusion Culling

Not rendering things that are guaranteed to be behind something else.

Coverage buffer:

A low-resolution version of (a simplified version of) the scene, rendered on the CPU, which we can use for visibility tests.







Visibility

rics & (depth < ™0000

: = inside ? 1 : ... ht = nt / nc, ddn os2t = 1.0f - n⊓t ° 0, N); ⊅)

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

= * diffuse; = true;

. efl + refr)) && (depth < MAXDED⊺

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

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly, closed if; radiance = SampleLight(&rand, I, &L, elignu e.x + radiance.y + radiance.z) > 0) && closed closed = close



Occlusion Culling

Not rendering things that are guaranteed to be behind something else.

Potential Visibility Set:

a table that tells us which areas are mutually visible.



PVS-Table



Visibility

tics ≰(depth < NOCC

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

= * diffuse = true;

efl + refr)) && (depth < MANDEPTH

D, N); refl * E * diffu: = true;

AXDEPTH)

survive = SurvivalProbability(diffuse .estimation - doing it properly, closed if; radiance = SampleLight(&rand, I, &L, &light) 2.x + radiance.y + radiance.z) > 0

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

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

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

Indoor visibility: Portals

Observation: if a window is invisible, the room it links to is invisible.

























Visibility

nics & (depth < ⊅000001

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

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

= * diffuse; = true;

efl + refr)) && (depth < MAXDEE

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

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly, close if; radiance = SampleLight(&rand, I, &L, 2.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) = (r

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

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

Visibility determination

Coarse:

- Grid-based (typically outdoor)
- Portals (typically indoor)

Finer:

- Frustum culling
 - Occlusion culling

Finest:

- Backface culling
- Clipping
- Z-buffer





ics & (depth < ™xxxxx

: = inside ? 1 ht = nt / nc, ddn os2t = 1.0f - nnt 0, N); 0)

at a = nt - nc, b = nt = nc at Tr = 1 - (R0 + (1 - R0 Γ r) R = (D = nnt - N (300)

= * diffuse; = true;

efl + refr)) && (depth < MAXDEDID

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

AXDEPTH)

survive = SurvivalProbability(diffuse)
estimation - doing it properly.closed
if;
radiance = SampleLight(&rand, I, &L, &light
e.x + radiance.y + radiance.z) > 0) && doct

v = true;

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

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

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

Today's Agenda:

- Depth Sorting
- Clipping
- Visibility



tics & (depth < MODEL)

c = inside ? 1 ht = nt / nc, ddn bs2t = 1.0f - nnt 2, N); 2)

at a = nt - nc, b = nt at Tr = 1 - (R0 + (1 - R0) Γ r) R = (D = nnt - N - (10)

= * diffuse; = true;

efl + refr)) && (depth < MAXDERII

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

AXDEPTH)

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

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

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

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

INFOGR – Computer Graphics

Jacco Bikker & Debabrata Panja - April-July 2018

END OF lecture 13: "Visibility"

Next lecture: "Postprocessing"

