tic: ⊾ (depth < 1955

: = inside / l nt = nt / nc, dd os2t = 1.0f - on 0, N); 0)

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

E ⁼ diffuse = true;

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

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

AXDEPTH)

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

v = true; at brdfPdf = EvaluateDiffuse(L, N) * Puncture 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 & Debabrata Panja - April-July 2017

Lecture 12: "Visibility"

Welcome!





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

AXDEPTH)

adiance = SampleLight e.x + radiance.y + rad

v = true; at brdfPdf = EvaluateD at weight = Mis2(dire at cosThetaOut = dot(E * ((weight * cosThe

t3 brdf = SampleDiffuse(diffuse, N, −1, −2, NR, prvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:



















and a second





Smallest Ray Tracers:

,c=c*255,P(c.x),P(c.y),P(c.z);}

#import<cmath> #import<cstdio> #define R return #define 0 operator #define P putchar #define I int #define f float I l=2e9;struct v{f x,y,z;v(){}v(f a, f b, f c){x=a;y=b;z=c;}v 0+(v o){R{x+o.x,y+o.y,z+o.z};}

fl + refr)) 88 (de), N);

AXDEPTH)

st weight = Mis2(di at cosThetaOut = dot(E * ((weight * cosTh

andom walk - done pr /ive)

968

R t*(1/sqrt(t%t));}};struct b{v p,c;f r;};f T(v o,v d,b*W,v*C,I M){if(M>9)R l;f Z=1,t,h; v c,p,q;for(I i=3;i--;)c=W[i].p+-o,t=c%d,c=c+d*-t,(h=W[i].r-c%c)>0&(t-=sqrt(h))>.01&Z>t?

 $5-c.y, -c.z), W, C, M+1) < 1?v(0,0,0):v(t=I(c.x)+I(c.z)&1,t,t)*(q%p); Z==1&d.y>0?*C={.5,1,1}:q;$ R Z;}main(){printf("P6 512 512 255 ");v c;b W[]={{{2,3,6},{1,0,0},3},{{5,1,4},{0,1,0},1},{{ 9,3,7},{0,0,1},4}};for(f y=512;y--;)for(f x=512;x--;)T({5,2,0},!v(1-x/256,y/256-1,1),W,&c,0)

v O-(){R*this*-1;}v O*(f s){R{x*s,y*s,z*s};}f O%(v o){R x*o.x+y*o.y+z*o.z;}v O!(){v t=*this; (*C=W[i].c,Z=t,c=d*Z+o,p=!(c+-W[i].p),p=d+p*-2*(p%d),T(c,p,W,&c,M+1),*C=v(C->x*c.x,C->y*c.y, $C \rightarrow z^*c.z$, 1):1;if(d.y<0){q={0,1,0};t=-0%q/(d%q);if(t>.01&Z>t)Z=t,c=d*Z+o,*C=T(c,p=!v(5-c.x,

L=:9e9e=:1e_4h=:%:@:(+/@:*:)n=:%hd=:+/@:*Y=: :0r=:4 Yt=.(-&0.5)@:%&s(3\$0);(n(t x),(t s-y),1);L)V=:3 Y(;0{y)+(0{;2{y}*;1{y}i=:4 Y c=.(;0{0{y}-;0{xt=.c d;1{xq=.t-%:(*:;1{0{y}-+/*:c-t*;1{xe=.(<;2{x},<yif.0<*:q-tdo.if.(q<;2{x})*q>0do.e=.(<q),<yend.end.e)k=:4 Yif.</pre> (;0{x)<;0{ydo.xelse.yend.)G=:3 Yk/1(y&i)\4 5\$((_3,0,8);1;(0,0,1);0.1;0),((0,0,8);1;(0,1,0);0;0),((3,0,8);1;(1,0,0);0.4;0), (0, 10001,8);1e4;(3\$e);0.2;1)K=:4 Y0.5>(x{y)-<.x{y}c=:3 YD=:D+1F=.G ye=.0{;0{FS=:0{;1{Fm=.0{;3{Sr=.(2{.y},<ei=.V rN=.n i-;0{Su=. (e<L)*(1-m)*((;2{S)*3\$-.(;4{S)*(0 K i)=2 K i)*r A SM=.0if.(e<L)*(m>0)*D<9do.M=.m*c i;(n(;1{r)-N*((;1{r)d N)*2};Lend.u+M)T=:4 Y p=.V;0{xs=.(;0{0{y}-pD=.n st=.h s(3\$(;0{G(p+D*e);D;t)>=t)*(;1{0{y}*(1%*:t)*(;1{x)d D)F=:4 Y(0>.;x)+0>.;y)A=:4 YF/1(<@:((x;-(;0{y}-V x)&T))\2 2\$(0,9,0);3\$99)P=:3 YD=:0":255<.<.255*c(s|y)r<.y%s)s=:512stdout'P3 512 512 255',1 P\i.*:sexit''



at3 brdf = SampleDiffuse(diffuse, N, r1, r2, IR rvive; pdf;

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

Fastest Ray Tracer:

tic: k (depth < 19

= inside / : nt = nt / nc. : ps2t = 1.0f p, N); 3)

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

= diffuse = true;

-:fl + refn)) && (depth is MARDITIN

D, N); ref1 * E * diffu = true;

AXDEPTH)

survive = SurvivalProbability(difference estimation - doing it properly if; adiance = SampleLight(&rand, I, I) 2.x + radiance.y + radiance.r) 00 55

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) * Poulse

ndom walk - done properly, closely following a ive)

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





tic: € (depth < 1555

:= inside / i 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 - N

= diffuse; = true;

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

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.r) = 0.000

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) (Public)

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

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

Today's Agenda:

- Depth Sorting
- Clipping
- Visibility





Perspective





Perspective

tic: k (depth < 100

= inside / 1 it = nt / nc. dde ss2t = 1.8f - no), N);))

at a = nt - nc, b = ntat Tr = 1 - (R0 + (1 - R))Tr) $R = (D^{+} nnt - R^{-})$

= diffuse = true;

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

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

AXDEPTH)

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

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

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

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



The world according to the camera: *Camera space*



7

Perspective

Transformation Pipeline



i = E * brdf * (dot(N, R) / pdf);
sion = true:

tic: € (depth < 1555

:= inside / i 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 - N

= diffuse; = true;

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

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.r) = 0.000

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) (Public)

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

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

Today's Agenda:

- Depth Sorting
- Clipping
- Visibility





Depth Sorting

fice (depth of Ross

- - AD2400 - . it = nt / nc. ddo 552t = 1.0f - ... 5, N); 3)

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

= diffuse; = true;

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

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

AXDEPTH)



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



pixels

Postprocessing



Depth Sorting

- nics ⊾ (depth < 1000
- - inside / . it = nt / nc, dde -552t = 1.8f - nnt -5, N); 8)
- at a = nt nc, b = nt at Tr = 1 - (R0 + (1 - 0) Tr) R = (D * nnt - N *
- = diffuse; = true;
- -: :fl + refr)) && (depth & HAODE
- D, N); refl * E * diffuse; = true;
- AXDEPTH)
- survive = SurvivalProbability(difference estimation - doing it property if; radiance = SampleLight(&rand, I, ... e.x + radiance.y + radiance.z) > 0)



3. Rasterize: *determining affected pixels*

Questions:

- What is the screen space position of the fragment?
- Is that position actually on-screen?
- Is the fragment the nearest fragment for the affected pixel?

How do we efficiently determine *visibility* of a pixel?



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









13

ion = true:

st a = nt

), N);

= true;

AXDEPTH)

if:

-efl * E * diffuse;

survive = SurvivalProbability dif

-adiance = SampleLight(&rand.

.x + radiance.y + radiance.z)

Depth Sorting

Overdraw:

Inefficiency caused by drawing multiple times to the same pixel.

= diffuse = true;

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

), N); ~efl * E * diffu: = true;

WXDEPTH)



ion = true:



Depth Sorting

Overdraw:

Inefficiency caused by drawing multiple times to the same pixel.





. efl + refr)

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

AXDEPTH)

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



Depth Sorting

Overdraw:

Inefficiency caused by drawing multiple times to the same pixel.





), N); ~efl * E ' = true;

st a = nt

WXDEPTH)



Correct order: BSP

tic: ⊾(depth < 10.

= = inside / : it = nt / nc, dd os2t = 1.8f - nn O, N); D)

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

= diffuse; = true;

efl + refr)) && (depth is HADDII

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

AXDEPTH)

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



root



Correct order: BSP



AXDEPTH)

survive = SurvivalProbability(differ -adiance = SampleLight(&rand, I, LL, L e.x + radiance.y + radiance.z) > 0) ##





Correct order: BSP



-:fl + refr)) 88 (depth (H

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

AXDEPTH)





Depth Sorting Correct order: BSP root back front efl * E * diffuse; AXDEPTH) -adiance = SampleLight(&rand, I, LL, e.x + radiance.y + radiance.z) > 0) []



20





on = true:





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

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

AXDEPTH)

st a = nt

survive = SurvivalProbability diff if; adiance = SampleLight(&rand, I. LL e.x + radiance.y + radiance.z) > 0)



Does not eliminate overdraw

Depth Sorting

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

AXDEPTH)

survive = SurvivalProbability(dif if: adiance = SampleLight(&rand, I. e.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?

- 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 $z_{int} = \frac{(2^{32}-1)}{-7+1}$: 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





f1 + refr)) && (dept

-efl * E * diffuse;

= true:

), N);

= true;

AXDEPTH)

adiance = SampleLight(&rand, I .x + radiance.y + radiance.z) > 1





Depth Sorting

Z-buffer optimization

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'.

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

efl + refr)) && (depth

AXDEPTH)

st Tr = 1

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





Depth Sorting

tic: ↓ (depth < 10

= * inside / : it = nt / nc, dde ss2t = 1.8f 2, N); 3)

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

= diffuse; = true;

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

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

AXDEPTH)

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



'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

tic: € (depth < 1555

:= inside / i 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 - N

= diffuse; = true;

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

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.r) = 0.000

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) (Public)

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

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

Today's Agenda:

- Depth Sorting
- Clipping
- Visibility





Clipping

tice ≰ (depth < RAS

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

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

= diffuse = true;

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

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) * Indi

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

; pt3 brdf = SampleDiffuse(diffuse, N, r1, r2, NR, bp3 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).





Clipping

st a = nt

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

AXDEPTH)

survive = SurvivalProbability(di if: -adiance = SampleLight(&rand. e.x + radiance.y + radiance.z)

v = true; st 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 follo /ive)

at3 brdf = SampleDiffuse(diffuse, N, r1, r2, UR, U rvive; pdf; i = 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 v_0 is 'out', but v_1 is 'in', emit C and v_1

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

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

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

out





in





Clipping

tica ⊾ (depth k 19

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

st $a = nt - nc_{1}b - nt$ st Tr = 1 - (R0 + (1 - 1))(r) R = (0 + nrt - 1)

= diffuse; = true;

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

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

AXDEPTH)

v = true; at brdfPdf

at brdfPdf = EvaluateDiffuse(L, N) Process 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)

; pt3 brdf = SampleDiffuse(diffuse, N, r1, r2, UR, bp4 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:

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



Clipping

fic: ⊾ (depth ⊂ NASC

= inside / 1 it = nt / nc, dde 552t = 1.0f = nnt 3, N); 3)

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

= diffuse; = true;

efl + refr)) && (depth k HANDII

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; tbrdfPdf = EvaluateDiffuse(L, N) * F st3 factor = diffuse * INVPI; st 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, r1, r2, R, lpd prvive; 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

tic: k (depth < 10

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

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

= diffus = true;

efl + refr)) && (depth k MANDI)

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

AXDEPTH)

v = true;

bt brdfPdf = EvaluateDiffuse(L, N,) * Pun st3 factor = diffuse * INVPI; st 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, r1, r2, NR, s); pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Sutherland-Hodgeman

Clipping can be done against arbitrary planes.









tic: € (depth < 1555

:= inside / i 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 - N

= diffuse; = true;

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

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.r) = 0.000

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) (Public)

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

; pt3 brdf = SampleDiffuse(diffuse, N, F1, F2, UR, Sov 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

 $\sqrt{1}$ Torso closer than ground

Tree between ground & sun

B F

八

M

5





Visibility

at a = nt

), N);

= true;

AXDEPTH)

if:

-efl * E * diffuse;

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
- e.x + radiance.y + radiance.z) > 0) ## feature w = true; ot brdfPdf = EvaluateDiffuse(L, N) * Pauro st3 factor = diffuse * INVPI; st weight = Mis2(directPdf, brdfPdf); st cosThetaOut = dot(N, L);

survive = SurvivalProbability(dia

adiance = SampleLight(&rand, I

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

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

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



tice € (depth < 10.5

= inside / 1 it = nt / nc, dde -552t = 1.0f - nnt -3, N); 3)

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

= * diffuse; = true;

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

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

AXDEPTH)

survive = SurvivalProbability(difference)
estimation - doing it properly
ff;
radiance = SampleLight(&rand, I.)
e.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 /ive)

; pt3 brdf = SampleDiffuse(diffuse, N, r1, r2, NR, Dpd prvive; 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.







tic: k (depth < 100

= inside / :
it = nt / nc, ddn
ss2t = 1.8f = nt
), N);
3)

E = diffuse; = true;

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

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

AXDEPTH)

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

v = true; at brdfPdf = EvaluateDiffuse(L, N) * Pourst3 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, NR, bpd prvive; 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.





ic: (depth < NAS

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

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

E * diffuse; = true;

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

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

AXDEPTH)

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

v = true; at brdfPdf = EvaluateDiffuse(L, N) = P 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)

; pt3 brdf = SampleDiffuse(diffuse, N, r1, r2, UR, Upd) prvive; 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.





tice k (depth < 155

= inside / 1 nt = nt / nc, dde 552t = 1.8f - nnt 5, N); 3)

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

= diffuse; = true;

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

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

AXDEPTH)

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 a /ive)

; pt3 brdf = SampleDiffuse(diffuse, N, F1, F2, UR, body 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.





Visibility

tic: ⊾ (depth < N4

= 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

= diffus: = true;

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

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

AXDEPTH)

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

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)

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

; pt3 brdf = SampleDiffuse(diffuse, N, r1, r2, UR, Dpd prvive; 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

tic: k (depth < 10.55

- : = inside / l it = nt / nc, ddo os2t = 1.0f - nn: 0, N); 3)
- st a = nt nc, b nt --st Tr = 1 - (R0 + (1 - --fr) R = (D * nnt - N -
- = diffuse; = true;
- D, N); refl * E * diffuse; = true;

MXDEPTH)

- survive = SurvivalProbability(difference estimation - doing it property ff; radiance = SampleLight(%rand, I do to e.x + radiance.y + radiance.z) = 0
- v = true; at brdfPdf = EvaluateDiffuse(L, N) at3 factor = diffuse = INVPI; at weight = Mis2(directPdf, brdfPdf)
- st weight = Mis2(directPdf, bronpdf); 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:

Visibility determination

Coarse:

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

Finer:

- Frustum culling
 - Occlusion culling

Finest:

- Backface culling
- Clipping
- Z-buffer

tic: € (depth < 1555

:= inside / i 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 - N

= diffuse; = true;

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

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.r) = 0.000

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) (Public)

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

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

Today's Agenda:

- Depth Sorting
- Clipping
- Visibility

tica ⊾ (depth⊂c NAS

z = inside / L it = nt / nc, dd os2t = 1.0f o, N); 8)

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

E ⁼ 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 2.x + radiance.y + radiance.z) = 0.000

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) * Ond

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

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

INFOGR – Computer Graphics

Jacco Bikker & Debabrata Panja - April-July 2017

END OF lecture 12: "Visibility"

Next lecture: "Postprocessing"

