tic: k (depth (⊂)s⊃

:= inside / 1 it = nt / nc, ddo os2t = 1.0f - not 0, N); 3)

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

= diffuse = true;

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

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

AXDEPTH)

v = true; t brdfPdf = EvaluateDiffuse(L, N) = Pour 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 o /ive)

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

INFOGR – Computer Graphics

J. Bikker - April-July 2015 - Lecture 11: "Accelerate"

Welcome!



tic: ⊾ (depth < 100:

:= inside / i it = nt / nc, dde os2t = 1.8f - nnt 0, N); 8)

at $a = nt - nc_{1}b - nt - nc_{2}b$ at Tr = 1 - (R0 + 1)Tr) R = (0 * nnt - N)

= diffuse; = true;

• •fl + refr)) 88 (death (1000)

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

AXDEPTH)

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

v = true; ot brdfPdf = EvaluateDiffuse(L, N,) Provide st3 factor = diffuse * INVPI; ot weight = Mis2(directPdf, brdfPdf); ot cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) * 1900

indom walk - done properly, closely following /ive)

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

Today's Agenda:

- High-speed Ray Tracing
- Acceleration Structures
- The Bounding Volume Hierarchy
- BVH Construction
- BVH Traversal
- Optimizing Construction
- High-speed Traversal



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

at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf

survive = SurvivalProbability(difference estimation - doing it proper) ff; radiance = SampleLight(&rand, I ... e.x + radiance.y + radiance.r) > > w = true; ot brdfPdf = EvaluateDiffuse(L, N) = P st brdfPdf = EvaluateDiffuse(L, N) = P st st weight = Mis2(directPdf, brdfPdf);

efl * E * diffuse; = true;

AXDEPTH)

D, N); refl * E * diffuse

E * diffuse; = true;

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

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

fic: ⊾ (depth () 10.





High-speed Ray Tracing

fici ⊾(depth (192)

: = inside / lo it = nt / nc, dde / ps2t = 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 < NADIII

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

AXDEPTH)

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

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

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

Ray Tracing – Needful things

Whitted-style ray tracing:

primary ray per pixel
 shadow ray per pixel per light
 Optional: rays for reflections & refraction

Estimate:

- 10 rays per pixel
- 1M pixels (~1280x800)
- 30 fps

→ 300Mrays/s

How does one intersect 300Mrays/s on a 3Ghz CPU? Easy: use no more than 10 cycles per ray.



ONE DOES NOT SIMPLY

CAST A RAY IN 10 CYCLES

High-speed Ray Tracing

tic: k (depth < 100⊂

: = 105108 / : it = nt / nc, dde 552t = 1.8f - not 5, N); 3)

st a = nt - nc, b - nt - n st Tr = 1 - (R0 + (1 - 1) fr) R = (D * nnt - N * 1.0

= * diffuse; = true;

efl + refr)) && (depth < HADDETT

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.z) > 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)

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

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

Actually...

- We have 8 cores (so 80 cycles)
- Executing AVX code (so 640 cycles)
- Plus 20% gains from hyperthreading (768 cycles).

But really...

Assuming we get a linear increase in performance for the number of cores and AVX, how do we intersect thousands of triangles in 768 cycles?

ONE DOES NOT SIMPLY

CAST A RAY IN 10 CYCLES



High-speed Ray Tracing

fice ⊾ (depth (⊂)000

: = inside / : it = nt / nc, dde os2t = 1.8f - nnt D, N); D)

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

= diffuse = true;

efl + refr)) && (depth is HANDIED)

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

AXDEPTH)

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

w = true; at brdfPdf = EvaluateDiffuse(L, N) Provident at3 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) pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Optimization

- 1. Measure: performance & scalability
- 2. High level optimizations: improve algorithmic complexity
- 3. Low level optimization: instruction level & threadlevel parallelism, caching
- 4. GPGPU

More in the master course Optimization & Vectorization.

ONE DOES NOT SIMPLY

CAST A RAY IN 10 CYCLES



tic: ⊾ (depth < 100:

:= inside / i it = nt / nc, dde os2t = 1.8f - nnt 0, N); 8)

at $a = nt - nc_{1}b - nt - nc_{2}b$ at Tr = 1 - (R0 + 1)Tr) R = (0 * nnt - N)

= diffuse; = true;

• •fl + refr)) 88 (death (1000)

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

AXDEPTH)

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

v = true; ot brdfPdf = EvaluateDiffuse(L, N,) Provide st3 factor = diffuse * INVPI; ot weight = Mis2(directPdf, brdfPdf); ot cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) * 1900

indom walk - done properly, closely following /ive)

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

Today's Agenda:

- High-speed Ray Tracing
- Acceleration Structures
- The Bounding Volume Hierarchy
- BVH Construction
- BVH Traversal
- Optimizing Construction
- High-speed Traversal



High-speed Ray Tracing

tica k (depth (⊂104)

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

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

= diffuse = true;

efl + refr)) && (depth < NAODITIE

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

AXDEPTH)

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

v = true; at brdfPdf = EvaluateDiffuse(L, N) * Promote st3 factor = diffuse * INVPI; st weight = Mis2(directPdf, brdfPdf); st cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) * 1000

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

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

Optimization: reduce algorithmic complexity

Complexity:

number of ray/primitive intersections

= pixels * paths per pixel * average path length * primitives







8

tica ⊾ (depth (c))⇔

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

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

= diffuse = true;

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

), N); refl * E * diff: = 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,) at3 factor = diffuse " INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPd

indom walk - done properly, closely following
vive)

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

Option 1:

Use a grid.

- Each grid cell has a list of primitives that overlap it.
- The ray traverses the grid, and intersects only primitives in the grid cells it visits.

Problems:

- Many primitives will be checked more than once.
- It costs to traverse the grid.
- How do we chose grid resolution?
- What if scene detail is not uniform?



Pr

tic: ⊾(depth k 1955

: = inside / L it = nt / nc, ddo os2t = 1.0f - on: " D, N); B)

= diffuse = true;

efl + refr)) && (depth < HAODETH

D, N); ref1 * E * diff(= true;

AXDEPTH)

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

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

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

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



Option 2:

Use a nested grid.

- We use fewer cells. Each grid cell that overlaps multiple primitives has a smaller grid in it.
- The ray rapidly traverses empty space, and checks the nested grids when needed.

Problems:

- How do we chose grid resolutions?
- Is this the optimal way to traverse space?



tica ⊾ (depth (c))⇔

: = inside / L it = nt / nc, ddo os2t = 1.0f - ont T D, N); B)

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

= diffuse; = true;

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

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

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



Option 3:

Use an octree.

- We start with a bounding box of the scene;
- The box is recursively subdivided in 8 equal boxes as long as it contains more than X primitives.

Problems:

- What if all the detail is exactly in the centre of the scene?
- Splitting in 8 boxes: is that the optimal subdivision?



tic: ⊾(depth c no

: = inside / 1 it = nt / nc, dde os2t = 1.0f - nnt 7 0, N); 3)

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

= diffuse = true;

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

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

AXDEPTH)

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

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

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



Option 4:

Use an kD-tree.

- We start with a bounding box of the scene;
- Using arbitrary axis-aligned planes, we recursively cut it in two halves as long as it contains more than X primitives.

Problems:

- Primitives may end up in multiple leaf nodes.
- How hard is it to build such a tree?





tic: ⊾ (depth < 100:

:= inside / i it = nt / nc, dde os2t = 1.8f - nnt 0, N); 8)

at $a = nt - nc_{1}b - nt - nc_{2}b$ at Tr = 1 - (R0 + 1)Tr) R = (0 * nnt - N)

= diffuse; = true;

• •fl + refr)) 88 (death (1000)

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

AXDEPTH)

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

v = true; ot brdfPdf = EvaluateDiffuse(L, N,) Provide st3 factor = diffuse * INVPI; ot weight = Mis2(directPdf, brdfPdf); ot cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) * 1900

indom walk - done properly, closely following /ive)

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

Today's Agenda:

- High-speed Ray Tracing
- Acceleration Structures
- The Bounding Volume Hierarchy
- BVH Construction
- BVH Traversal
- Optimizing Construction
- High-speed Traversal



BVH

tic: ⊾ (depth ic iv.)

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

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

= diffuse = true;

-:**fl + refr))** && (depth < MAQUE

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

AXDEPTH)

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

v = true; at brdfPdf = EvaluateDiffuse(L, N) * Pr 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, bpd prvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true;



Option 5:

Use a bounding volume hierarchy.





BVH

fice k (depth < nace

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

at a = nt - nc, b - nt - r at Tr = 1 - (R8 + (1 - 7 fr) R = (D * nnt - 8

= diffuse; = true;

-:fl + refr)) && (dept)

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

AXDEPTH)

survive = SurvivalProbability(d.f estimation - doing it properly if; radiance = SampleLight(\$rand, I. 4 e.x + radiance.y + radiance.z) > 0)

v = true; at brdfPdf = EvaluateDiffuse(L, N) * Provide st3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) * 1

indom walk - done properly, closely following /ive)

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

The Bounding Volume Hierarchy

BSPs, grids, octrees and kD-trees are examples of *spatial subdivisions*. The BVH is of a different category: it is an *object partitioning scheme*:

Rather than recursively splitting space, it splits collections of objects.

primitive array

+ splitplane =

primitives in 'left' child

primitives in 'right' child



BVH

fice k (depth < nace

:= inside / L it = nt / nc, dde 552t = 1.8f - nor 5, N); 8)

st a = nt - nc, b - nt s st Tr = 1 - (RB - (1 - 1) Tr) R = (D - nnt - N

= diffuse; = true;

efl + refr)) && (depth

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

AXDEPTH)

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

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) * 1

indom walk - done properly, closely following /ive)

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

The Bounding Volume Hierarchy

Sorting an array of elements based on a value: BVH is very similar to *QuickSort*.

In the BVH construction algorithm, the split plane position is the pivot.

primitive array

+ splitplane =

primitives in 'left' child

primitives in 'right' child



BVH

fice ⊾ (depth-c nose

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

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

= diffuse; = true;

: efl + refr)) && (depth < HANDLIND

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

AXDEPTH)

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

w = true; t brdfPdf = EvaluateDiffuse(L, N) * P st3 factor = diffuse * INVPI; st 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, M, so pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Bounding Volume Hierarchy: data structure

struct BVHNode

};

BVHNode* left; BVHNode* right; aabb bounds; bool isLeaf; vector<Primitive*> primitives; // 4 or 8 bytes
// 4 or 8 bytes
// 2 * 3 * 4 = 24 bytes
// ?
// ?



BVH

tic: ⊾ (depth (⊂)u.s:

= inside / 1 it = nt / nc, ddo ps2t = 1.0f - ont D, N); D)

st a = nt - nc, b - nt - st st Tr = 1 - (R0 + 11 - 7 fr) R = (D * nnt - N *

= diffuse = true;

= :fl + refr)) && (depth < HANDEFIN

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

AXDEPTH)

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

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

andom walk - done properly, closely followir /ive)

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

Bounding Volume Hierarchy: construction

void ConstructBVH(Primitive* primitives)

```
BVHNode* root = new BVHNode();
root->primitives = primitives;
root->bounds = CalculateBounds( primitives );
root->isLeaf = true;
root->Subdivide();
```

void BVHNode::Subdivide()

```
if (primitives.size() < 3) return;
this.left = new BVHNode(), this.right = new BVHNode();
...split 'bounds' in two halves, assign primitives to each half...
this.left->Subdivide();
this.right->Subdivide();
this.isLeaf = false;
```



BVH

tic: ⊾ (depth k)uus

: = inside / : it = nt / nc, ddo ss2t = 1.8f - not 5, N); 3)

st a = nt - nc, b - nt - nc st Tr = 1 - (88 + (1 Tr) R = (D * nnt - N

= diffuse = true;

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

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

AXDEPTH)

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

v = true; at brdfPdf = EvaluateDiffuse(L, N,) * Pus 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, UR, pdf; n = E * brdf * (dot(N, R) / pdf); sion = true;

Bounding Volume Hierarchy: construction



Construction consequences:

- Construction happens *in place:* primitive array is constant, index array is changed
- Very similar to Quicksort (split plane = pivot)

Data consequences:

- 'Primitive list' for node becomes offset + count
- No pointers!
- No pointers? (what about left / right?)



S

0 1 2...

BVH

nic) ⊾ (depth ⊂ Nor:

: = inside / L it = nt / nc, dda os2t = 1.8f - ant 7 O, N); B)

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

= diffuse; = true;

efl + refr)) && (depth < HANDLIN

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

AXDEPTH)

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

v = true; at 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 b /ive)

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

Bounding Volume Hierarchy: data structure

BVHNode* left;// 4 byteBVHNode* right;// 4 byteaabb bounds;// 4 byteaabb bounds;// 24 bytebool isLeaf;bool isLeaf;vector <primitive*> primitives;uint first;uint count;// 4 byte</primitive*>	BVHNode	truct BVHNode	
}; // // 44 byt	Node* left; Node* right; b bounds; l isLeaf; tor <primitive*> primitives;</primitive*>	<pre>uint left;</pre>	bytes bytes bytes bytes bytes bytes

BVH node pool



BVH

tice (depth (1945)

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

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

= diffuse; = true;

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

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)

v = true; st brdfPdf = EvaluateDiffuse(L, N) * | st3 factor = diffuse * INVPI; st weight = Mis2(directPdf, brdfPdf);

at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) * (no

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

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

Bounding Volume Hierarchy: data structure





// 4 bytes
// 4 bytes
// 24 bytes
// 4 bytes
// 4 bytes
// 4 bytes
// 4 bytes
// -----// 44 bytes



BVH

tica ⊾ (depth- ⊂ 1950)

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

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

= diffuse; = true;

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

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

AXDEPTH)

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

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

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

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

Bounding Volume Hierarchy: data structure

struct BVHNode

};

float3 bmin;
uint leftFirst;
float3 bmax;
uint count;

// bounds: minima
// or a union
// bounds: maxima
// leaf if 0
// -----// 32 bytes



tic: ⊾ (depth < 100:

:= inside / i it = nt / nc, dde os2t = 1.8f - nnt 0, N); 8)

at $a = nt - nc_{1}b - nt - nc_{2}b$ at Tr = 1 - (R0 + 1)Tr) R = (0 * nnt - N)

= diffuse; = true;

• •fl + refr)) 88 (death (1000)

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

AXDEPTH)

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

v = true; ot brdfPdf = EvaluateDiffuse(L, N,) Provide st3 factor = diffuse * INVPI; ot weight = Mis2(directPdf, brdfPdf); ot cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) * 1900

indom walk - done properly, closely following /ive)

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

Today's Agenda:

- High-speed Ray Tracing
- Acceleration Structures
- The Bounding Volume Hierarchy
- BVH Construction
- BVH Traversal
- Optimizing Construction
- High-speed Traversal



BVH Traversal

fice ⊾ (depth ≥ 100

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

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

= diffuse = true;

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

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

AXDEPTH)

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

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

indom walk - done properly, closely following
/ive)

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



BVH Traversal Algorithm:

Starting with the root:

If the node is a leaf node: Intersect triangles. Else:

> If the ray intersects the left child AABB: Traverse left child If the ray intersects the right child AABB: Traverse right child

How efficient is this?

In this case: we check every AABB, but we only try to intersect one red sphere. (total: 8 tests)



BVH Traversal

tic: ⊾ (depth is tus:

= = inside / 1 it = nt / nc, dde os2t = 1.8f - nnt 1 D, N); B)

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

E * diffuse = true;

efl + refr)) && (depth is HANDEFT)

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

AXDEPTH)

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

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

BVH Efficiency

The number of nodes in a BVH is at most 2N - 1. Example:

16	
8 + 8	
(4+4) + (4+4)	4
((2+2) + (2+2)) + ((2+2) + (2+2))	
1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 +	1

- In this case, we get from the root to a leaf in 5 steps, or: $\log_2 N + 1$.
- For 1024 primitives, we get to a leaf in 11 steps.
- For 1M primitives, we get to a leaf in 21 steps.

16 -----31



tic: ⊾ (depth < 100:

:= inside / i it = nt / nc, dde os2t = 1.8f - nnt 0, N); 8)

at $a = nt - nc_{1}b - nt - nc_{2}b$ at Tr = 1 - (R0 + 1)Tr) R = (0 * nnt - N)

= diffuse; = true;

• •fl + refr)) 88 (death (1000)

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

AXDEPTH)

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

v = true; ot brdfPdf = EvaluateDiffuse(L, N,) Provide st3 factor = diffuse * INVPI; ot weight = Mis2(directPdf, brdfPdf); ot cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) * 1900

indom walk - done properly, closely following /ive)

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

Today's Agenda:

- High-speed Ray Tracing
- Acceleration Structures
- The Bounding Volume Hierarchy
- BVH Construction
- BVH Traversal
- Optimizing Construction
- High-speed Traversal



Optimizing Construction

tic: € (depth < 100

= inside / 1 nt = nt / nc, dde os2t = 1.8f - nnt O, N); B)

at a = nt - nc, b - nt at Tr = 1 - (R8 + 1 - 1 Tr) R = (D * nnt - N *

= diffuse; = true;

efl + refr)) && (depth is HANDET

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

AXDEPTH)

survive = SurvivalProbabilit; estimation - doing it prope if; radiance = SampleLight(&ranc 2.x + radiance;y + radiance;;

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

andom walk - done properly, a **/ive)**

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



How do we construct a 'good' BVH?

What is a good BVH?

→ One that minimizes the number of ray/primitive intersections, and the number of ray/AABB intersections.



Optimizing Construction

fice K (depth <)sa

: = inside / 1 it = nt / nc, dde os2t = 1.8f - nnt 1 D, N); B)

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

= diffuse = true;

efl + refr)) && (depth < HADDIN

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

AXDEPTH)

survive = SurvivalProbability(difference estimation - doing it properly if; radiance = SampleLight(%rand, I, L, 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 follow /ive)

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

BVH Quality

A good BVH minimizes the number of intersections.

Concrete:

$$Q_{bvh} = \sum_{1}^{N} P_{AABB} (C_{AABB} + N_{tri} C_{tri})$$

Where:

N is the number of BVH nodes; P_{AABB} is the probability of a ray hitting the AABB; C_{AABB} is the cost of a ray intersecting the AABB; N_{tri} is the number of triangles in the node; C_{tri} is the cost of intersecting a triangle.

Probability of hitting an AABB with an arbitrary ray:

Proportional to the surface area of the AABB.



Binned BVH Construction

tic: ⊾(depth (10

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

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

= * diffuse = true:

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

D, N); refl * E * diff = 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

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

Surface Area Heuristic (Or: what is the best way to slice a bunny?)





Binned BVH Construction

tic: ⊾ (depth ≥ noo

= inside / 1 it = nt / nc, ddo 552t = 1.0f - not 5, N); 3)

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

= diffuse = true;

-**:fl + refr))** && (depth < NA)

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

AXDEPTH)

survive = SurvivalProbability(difference estimation - doing it property if; 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)

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

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



Cost:

 $N_{left} * A_{left} + N_{right} * A_{right}$

Select the split with the lowest cost.



Optimizing Construction

fica k (depth (⊂)use

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

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

= diffuse; = true;

efl + refr)) && (depth < MADICI

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

AXDEPTH)

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

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

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

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

Surface Area Heuristic

We construct a BVH by minimizing the cost after each split, i.e. we use the split plane position and orientation that minimizes the cost function:

 $C_{split} = N_{left} A_{left} + N_{right} Arig_{ht}$

The split is not made at all if the best option is more expensive than *not* splitting, i.e.

 $C_{nosplit} = N A$

This provides a natural termination criterion for BVH construction.



Optimizing Construction

Efficiency of the Surface Area Heuristic

fic: K (depth < 1955

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

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

= diffuse = true;

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

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

AXDEPTH)

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

v = true; at 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, UR, pdf) pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

A BVH constructed with the Surface Area Heuristic is typically *twice as efficient* as a tree constructed with naïve midpoint subdivision.



tic: ⊾ (depth < 100:

:= inside / i it = nt / nc, dde os2t = 1.8f - nnt 0, N); 8)

at $a = nt - nc_{1}b - nt - nc_{2}b$ at Tr = 1 - (R0 + 1)Tr) R = (0 * nnt - N)

= diffuse; = true;

• •fl + refr)) 88 (death (1000)

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

AXDEPTH)

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

v = true; ot brdfPdf = EvaluateDiffuse(L, N,) Provide st3 factor = diffuse * INVPI; ot weight = Mis2(directPdf, brdfPdf); ot cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) * 1900

indom walk - done properly, closely following /ive)

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

Today's Agenda:

- High-speed Ray Tracing
- Acceleration Structures
- The Bounding Volume Hierarchy
- BVH Construction
- BVH Traversal
- Optimizing Construction
- High-speed Traversal



Fast Traversal

tic: ⊾ (depth < 100

= inside / 1 nt = nt / nc, dda ps2t = 1.0f - nn D, N); B)

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

= diffuse = true;

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

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

AXDEPTH)

survive = SurvivalProbability different estimation - doing it property if; adiance = SampleLight(&rand, I. 2.x + radiance.y + radiance.r) = 0 & &

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

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

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







Fast Traversal

tica k (depth < 100

= inside / L it = nt / nc, doe 552t = 1.0f - not 5, N); 3)

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

= diffuse = true;

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

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

AXDEPTH)

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

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

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

Ray Packet Traversal

Primary rays for a small square of pixels tend to travel the same BVH nodes.

We can exploit this by explicitly traversing *ray packets*.



Fast Traversal

tic: (depth < 14

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

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

= diffuse = true;

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

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

AXDEPTH)

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

indom walk - done properly, closely following
/ive)

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



Packet Traversal Algorithm:

Starting with the root:

If the node is a leaf node: Intersect triangles. Else: If any ray intersects the left child AABB: Traverse left child If any ray intersects the right child AABB: Traverse right child



Fast Traversal

tica k (depth ≥ 1000

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

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

= diffuse; = true;

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

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

AXDEPTH)

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

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

indom walk - done properly, closely following /ive)

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

Ray Packet Traversal

Quickly determining if *any* ray intersects a node:

• Test the first one.

If it intersects, we're done. Else:

• Test if the AABB is outside the frustum encapsulating the packet.

If it misses, we're done. Else:

 Brute force test all rays. The first one that hits the AABB will be the ray we check first while processing the child nodes.



Fast Traversal

fic: ⊾(depth (192)

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

= diffuse; = true;

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

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

AXDEPTH)

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

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

andom walk - done properly, closely followin /ive)

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

Ray Packet Traversal Efficiency

Using the packet traversal approach, we can very efficiently traverse large packets of rays that travel roughly in the same direction. For primary rays, this can be 32x faster than single ray traversal.

Note that this requires the rays in the packet to traverse a similar set of BVH nodes. The ray packet must be *coherent* (as opposed to *divergent*). Ray coherence can be expressed as the extend to which rays in a packet travel the same nodes, or:

 $coherence = \frac{\#rays \ in \ packet}{average \ \#rays \ intersecting \ a \ node}$

Combined with an efficient BVH, we now have the performance needed for real-time ray tracing.





tic: ⊾ (depth < 100:

:= inside / i it = nt / nc, dde os2t = 1.8f - nnt 0, N); 8)

at $a = nt - nc_{1}b - nt - nc_{2}b$ at Tr = 1 - (R0 + 1)Tr) R = (0 * nnt - N)

= diffuse; = true;

• •fl + refr)) 88 (death (1000)

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

AXDEPTH)

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

v = true; ot brdfPdf = EvaluateDiffuse(L, N,) Provide st3 factor = diffuse * INVPI; ot weight = Mis2(directPdf, brdfPdf); ot cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) * 1900

indom walk - done properly, closely following /ive)

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

Today's Agenda:

- High-speed Ray Tracing
- Acceleration Structures
- The Bounding Volume Hierarchy
- BVH Construction
- BVH Traversal
- Optimizing Construction
- High-speed Traversal



tic: k (depth (⊂)s⊃

:= inside / 1 ht = nt / nc, ddo bs2t = 1.0f = nnt D; N); B)

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

= diffuse = true;

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

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

AXDEPTH)

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

v = true; at brdfPdf = EvaluateDiffuse(L, N.) * Promote 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 o /ive)

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

INFOGR – Computer Graphics

J. Bikker - April-July 2015 - Lecture 11: "Accelerate"

END of "Accelerate"

next lecture: "Shading Models"

