INFOGR – Computer Graphics

J. Bikker - April-July 2016 - Lecture 7: "Accelerate"

Welcome!



t a = nt - nc.

), N);

= true;

MAXDEPTH)

efl + refr)) && (depth < HAX

survive = SurvivalProbability diff

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



Today's Agenda:

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



```
st weight = Mis2( directPdf, brdfPdf );
st cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf) * (rudling sindom walk - done properly, closely following sindom walk - done properly sindom walk - done pr
```

efl + refr)) && (depth x HA

survive = SurvivalProbability(diff.

radiance = SampleLight(&rand, I, 8.) e.x + radiance.y + radiance.z) > 0) 88

st brdfPdf = EvaluateDiffuse(L, N) = st3 factor = diffuse = INVPI:

), N);

= true;

MAXDEPTH)

v = true;

High-speed Ray Tracing

Ray Tracing – Needful things

Whitted-style ray tracing:

1 primary ray per pixel1 shadow ray per pixel per lightOptional: 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.



efl + refr)) && (depth efl * E * diffuse; E * ((weight * cosThetaOut) / directPdf andom walk - done properly, closely foll at3 brdf = SampleDiffuse(diffuse, N, r1, r2, R,

High-speed Ray Tracing

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?





```
efl + refr)) && (depth
efl * E * diffuse;
survive = SurvivalProbability diff
e.x + radiance.y + radiance.z) > 0
st weight = Mis2( directPdf, brdfPdf
E * ((weight * cosThetaOut) / directPdf)
andom walk - done properly, closely fell-
```

at3 brdf = SampleDiffuse(diffuse, N, r1, r2, iR.

High-speed Ray Tracing

Optimization

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

More in the master course Optimization & Vectorization.







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

Today's Agenda:

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



```
st weight = Mis2( directPdf, brdfPdf );
st cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf) * (rudling sindom walk - done properly, closely following sindom walk - done properly sindom walk - done pr
```

efl + refr)) && (depth x HA

survive = SurvivalProbability(diff.

radiance = SampleLight(&rand, I, 8.) e.x + radiance.y + radiance.z) > 0) 88

st brdfPdf = EvaluateDiffuse(L, N) = st3 factor = diffuse = INVPI:

), N);

= true;

MAXDEPTH)

v = true;

High-speed Ray Tracing

Optimization: reduce algorithmic complexity

Complexity:

number of ray/primitive intersections

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

= 1M * 1 * 2 * 1M

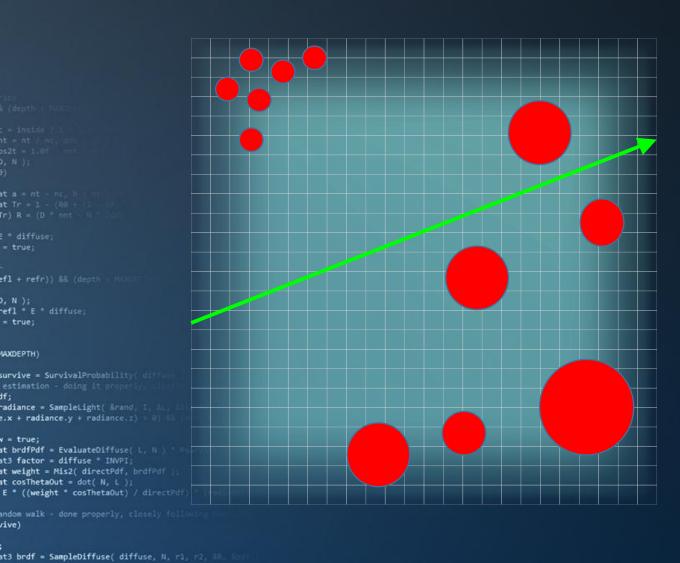




```
efl + refr)) && (depth c
), N );
= true;
MAXDEPTH)
survive = SurvivalProbability( diff.
adiance = SampleLight( &rand, I. M.
e.x + radiance.y + radiance.z) > 0) 6
v = true;
at brdfPdf = EvaluateDiffuse( L, N )
st3 factor = diffuse * INVPI:
st weight = Mis2( directPdf, brdfPdf )
at cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf)
andom walk - done properly, closely fello-
st3 brdf = SampleDiffuse( diffuse, N, r1, r2, iR, ic
1 = E * brdf * (dot( N, R ) / pdf);
```

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

Acceleration Structures



Option 1:

Use a grid.

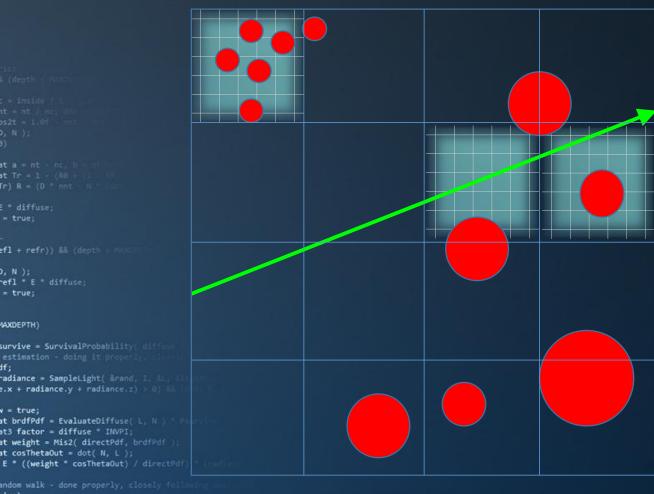
- Each grid cell has a list of primitives that overlap it.
- The ray traverses the grid, and intersects only primitives in the grid cells it visits.

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?



Acceleration Structures



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.

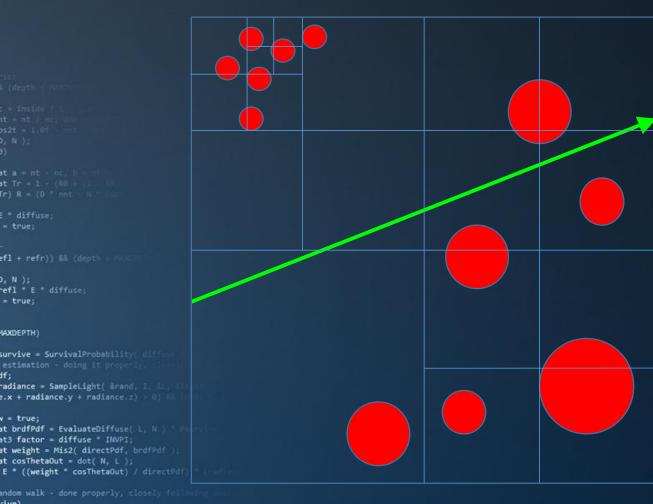
Problems:

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



//ve)
;
st3 brdf = SampleDiffuse(diffuse, N, r1, r2, 4R, 4pd)
srvive;
pdf;
n = E * brdf * (dot(N, R) / pdf);

Acceleration Structures



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.

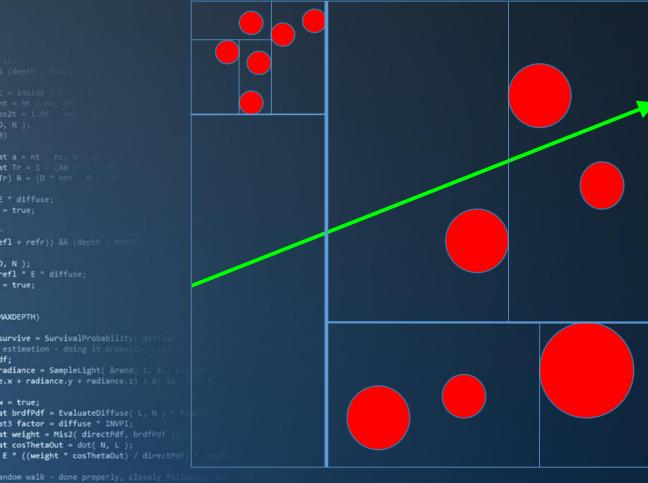
Problems:

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



;
st3 brdf = SampleDiffuse(diffuse, N, r1, r2, 48, 1900)
srvive;
pdf;
n = E * brdf * (dot(N, R) / pdf);

Acceleration Structures



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?



```
/ive)
;
st3 brdf = SampleDiffuse( diffuse, N, r1, r2, 88, 8500
urvive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
```

```
), N );
(AXDEPTH)
survive = SurvivalProbability( diff...
adiance = SampleLight( &rand, I, M., M.
e.x + radiance.y + radiance.z) > 0) [[]
v = true;
at brdfPdf = EvaluateDiffuse( L, N ) * Poo
st3 factor = diffuse * INVPI;
st weight = Mis2( directPdf, brdfPdf );
st cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf) * (Full-
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, NR, National
rvive;
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



```
st weight = Mis2( directPdf, brdfPdf );
st cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf) * (rudling sindom walk - done properly, closely following sindom walk - done properly sindom walk - done pr
```

efl + refr)) && (depth x HA

survive = SurvivalProbability(diff.

radiance = SampleLight(&rand, I, 8.) e.x + radiance.y + radiance.z) > 0) 88

st brdfPdf = EvaluateDiffuse(L, N) = st3 factor = diffuse = INVPI:

), N);

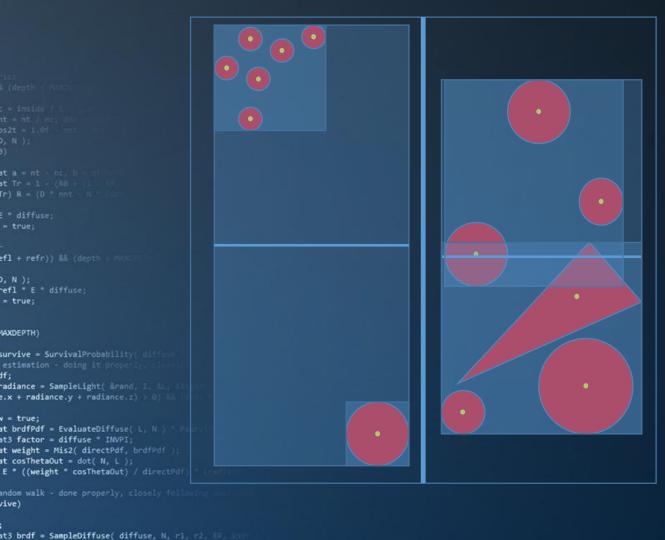
= true;

MAXDEPTH)

v = true;

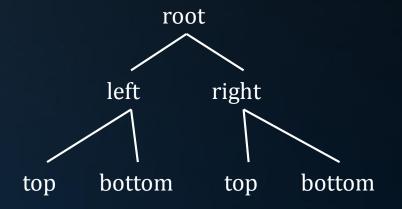
BVH

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



Option 5:

Use a bounding volume hierarchy.





BVH

efl + refr)) && (depth

survive = SurvivalProbability(d)

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

radiance = SampleLight(&rand e.x + radiance.y + radiance.z

), N); refl * E = true;

MAXDEPTH)

v = 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

```
st weight = Mis2( directPdf, brdfPdf );
st cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf) * (rudling);
sndom walk - done properly, closely following vive)
;
st3 brdf = SampleDiffuse( diffuse, N, r1, r2, IR, ippurvive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;
```



BVH

fl + refr)) && (depth

survive = SurvivalProbability(d)

at brdfPdf = EvaluateDiffuse(L, N at3 factor = diffuse = INVPI:

radiance = SampleLight(&rand e.x + radiance.y + radiance.z

), N); refl * E = true;

MAXDEPTH)

v = 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

```
st weight = Mis2( directPdf, brdfPdf );
st cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf) * (radio
sndom walk - done properly, closely following
vive)
;
st3 brdf = SampleDiffuse( diffuse, N, r1, r2, RR, radurvive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;
```



Acceleration Structures

};

struct BVHNode

BVHNode* left; BVHNode* right;

aabb bounds;

bool isLeaf;

Bounding Volume Hierarchy: data structure

vector<Primitive*> primitives;

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

```
// 4 or 8 bytes
// 4 or 8 bytes
// 2 * 3 * 4 = 24 bytes
// ?
// ?
```



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

Acceleration Structures

```
Bounding Volume Hierarchy: construction
                             void ConstructBVH( Primitive* primitives )
                                  BVHNode* root = new BVHNode();
                                  root->primitives = primitives;
st a = nt -
                                  root->bounds = CalculateBounds( primitives );
                                  root->isLeaf = true;
                                  root->Subdivide();
f1 + refr)) && (depth
), N );
efl * E * diffuse;
= true;
                             void BVHNode::Subdivide()
MAXDEPTH)
survive = SurvivalProbability( dif
                                  if (primitives.size() < 3) return;</pre>
adiance = SampleLight( &rand, I. .
                                  this.left = new BVHNode(), this.right = new BVHNode();
e.x + radiance.y + radiance.z)
v = true;
                                  ...split 'bounds' in two halves, assign primitives to each half...
at brdfPdf = EvaluateDiffuse( L. N
st3 factor = diffuse * INVPI:
                                  this.left->Subdivide();
st weight = Mis2( directPdf, brdfPdf
st cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf
                                  this.right->Subdivide();
andom walk - done properly, closely foll
                                  this.isLeaf = false;
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, U
```



efl + refr)) && (dept

efl * E * diffuse;

survive = SurvivalProbability dif

st weight = Mis2(directPdf, brdfPdf

E * ((weight * cosThetaOut) / directPdf

andom walk - done properly, closely fell

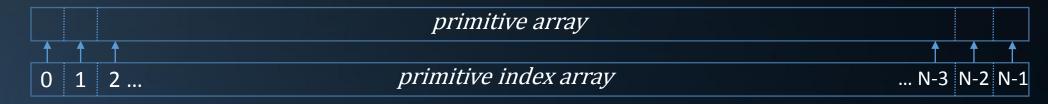
1 = E * brdf * (dot(N, R) / pdf);

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

), N);

Acceleration Structures

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)

<u>Data consequences</u>:

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



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

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

Acceleration Structures

struct BVHNode

Bounding Volume Hierarchy: data structure

```
BVHNode* left;
                                                                                                   uint left;
                                                                                                                                      // 4 bytes
                                     BVHNode* right;
                                                                                                   uint right;
                                                                                                                                      // 4 bytes
                                     aabb bounds;
                                                                                                   aabb bounds;
                                                                                                                                      // 24 bytes
                                     bool isLeaf;
                                                                                                   bool isLeaf;
                                                                                                                                      // 4 bytes
efl + refr)) && (depth k HA
), N );
                                                                                                                                      // 4 bytes
                                     vector<Primitive*> primitives;
                                                                                                   uint first;
efl * E * diffuse;
= true;
                                 };
                                                                                                   uint count;
                                                                                                                                      // 4 bytes
                                                                                               };
MAXDEPTH)
survive = SurvivalProbability( diff.
                                                                                                                                      // 44 bytes
adiance = SampleLight( &rand, I.
e.x + radiance.y + radiance.z) > 0)
                                                                                        BVH node pool
v = true;
at brdfPdf = EvaluateDiffuse( L, N
st3 factor = diffuse * INVPI:
st weight = Mis2( directPdf, brdfPdf )
at cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf)
andom walk - done properly, closely follow
```

struct BVHNode



Acceleration Structures

Bounding Volume Hierarchy: data structure

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

```
struct BVHNode
{
    uint left;
    uint right;
    aabb bounds;
    bool isLeaf;
    uint first;
    uint count;
};
```

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



Acceleration Structures

Bounding Volume Hierarchy: data structure

```
struct BVHNode
                                           float3 bmin;
                                                                             // bounds: minima
                                           uint leftFirst;
                                                                            // or a union
                                           float3 bmax;
                                                                            // bounds: maxima
                                           uint count;
                                                                             // leaf if 0
efl + refr)) && (depth x HA
), N );
                                      };
= true;
                                                                             // 32 bytes
MAXDEPTH)
survive = SurvivalProbability( diffo
adiance = SampleLight( &rand, I. ...
e.x + radiance.y + radiance.z) > 0) [
v = true;
at brdfPdf = EvaluateDiffuse( L, N )
st3 factor = diffuse * INVPI:
st weight = Mis2( directPdf, brdfPdf ):
at cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf) * ( )
andom walk - done properly, closely follow
st3 brdf = SampleDiffuse( diffuse, N, r1, r2, NR, NR
1 = E * brdf * (dot( N, R ) / pdf);
```



Today's Agenda:

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



```
st weight = Mis2( directPdf, brdfPdf );
st cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf) * (rudling sindom walk - done properly, closely following sindom walk - done properly sindom walk - done pr
```

efl + refr)) && (depth x HA

survive = SurvivalProbability(diff.

radiance = SampleLight(&rand, I, 8.) e.x + radiance.y + radiance.z) > 0) 88

st brdfPdf = EvaluateDiffuse(L, N) = st3 factor = diffuse = INVPI:

), N);

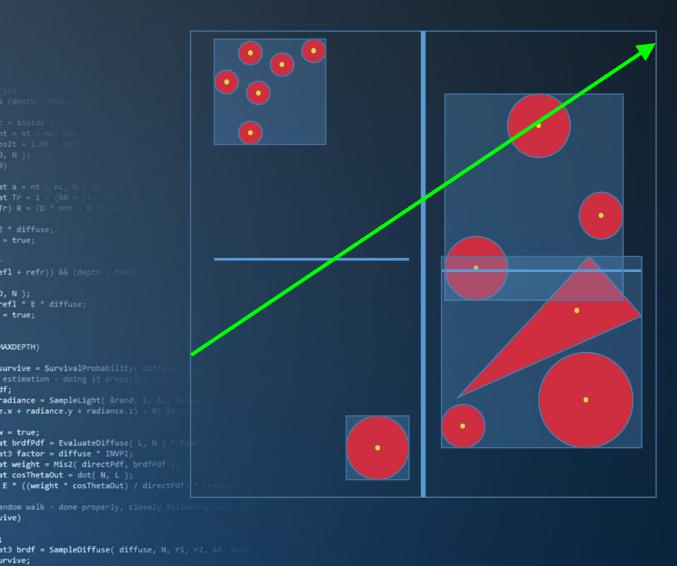
= true;

MAXDEPTH)

v = true;

BVH Traversal

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



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

BVH Efficiency

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

- 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.



; st3 brdf = SampleDiffuse(diffuse, N, r1, r2, NR, 1995) urvive; pdf; n = E * brdf * (dot(N, R) / pdf);

efl + refr)) && (depth

survive = SurvivalProbability(diff

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

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

E * ((weight * cosThetaOut) / directPdf

andom walk - done properly, closely fell

refl * E * diffuse; = true;

), N);

Today's Agenda:

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



```
st weight = Mis2( directPdf, brdfPdf );
st cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf) * (rudling sindom walk - done properly, closely following sindom walk - done properly sindom walk - done pr
```

efl + refr)) && (depth x HA

survive = SurvivalProbability(diff.

radiance = SampleLight(&rand, I, 8.) e.x + radiance.y + radiance.z) > 0) 88

st brdfPdf = EvaluateDiffuse(L, N) = st3 factor = diffuse = INVPI:

), N);

= true;

MAXDEPTH)

v = true;

Optimizing Construction

```
depth (Puxos)

= inside / 1

int = nt / nc, dde

sszt = 1.0f = nnt

sszt a = nt - nc, b = nt

sit Tr = 1 - (R0 + (1 R0)

Fr) R = (D * nnt - N * das

E * diffuse;
= true;

eff + refr)) && (depth < NAVORENIA

O, N );

ref1 * E * diffuse;
= true;

**AXXDEPTH)

survive = SurvivalProbability
```

radiance = SampleLight(&ran e.x + radiance.y + radiance.

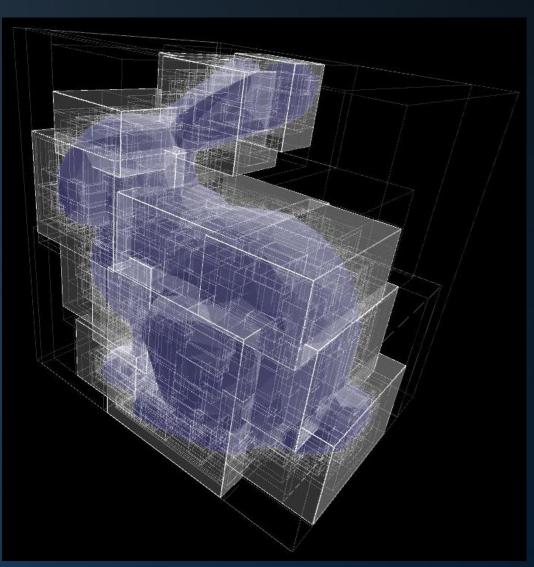
st brdfPdf = EvaluateDiffuse
st3 factor = diffuse * INVPI,
st weight = Mis2(directPdf,
st cosThetaOut = dot(N, L),
E * ((weight * cosThetaOut)
sndom walk - done properly,

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

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

v = 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

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.



f1 + refr)) && (depth), N); survive = SurvivalProbability/ difst weight = Mis2(directPdf, brdfPdf E * ((weight * cosThetaOut) / directPdf andom walk - done properly, closely fell at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &

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

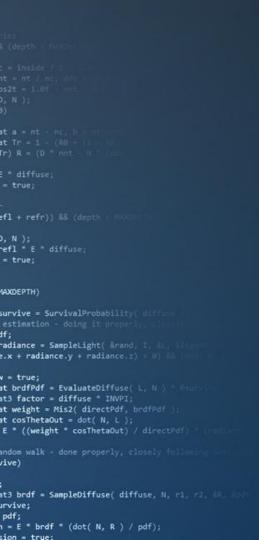
Binned BVH Construction

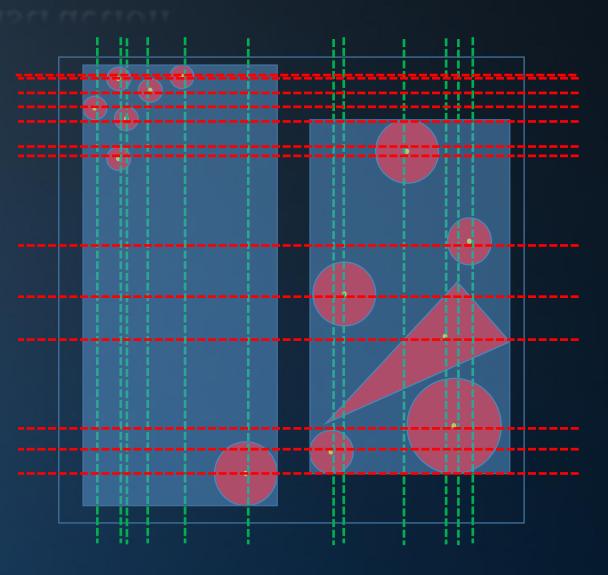
Surface Area Heuristic (Or: what is the best way to slice a bunny?)

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



Binned BVH Construction





Cost:

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

Select the split with the lowest cost.



Optimizing Construction

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.



```
efl + refr)) && (depth
= true;
MAXDEPTH)
survive = SurvivalProbability di
st weight = Mis2( directPdf, brdfPdf
E * ((weight * cosThetaOut) / directPdf)
andom walk - done properly, closely fello
st3 brdf = SampleDiffuse( diffuse, N, r1, r2, R
```

Optimizing Construction

Efficiency of the Surface Area Heuristic

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

efl + refr)) && (depth < HA

efl * E * diffuse;

), N);



Today's Agenda:

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



```
st weight = Mis2( directPdf, brdfPdf );
st cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf) * (rudling sindom walk - done properly, closely following sindom walk - done properly sindom walk - done pr
```

efl + refr)) && (depth x HA

survive = SurvivalProbability(diff.

radiance = SampleLight(&rand, I, 8.) e.x + radiance.y + radiance.z) > 0) 88

st brdfPdf = EvaluateDiffuse(L, N) = st3 factor = diffuse = INVPI:

), N);

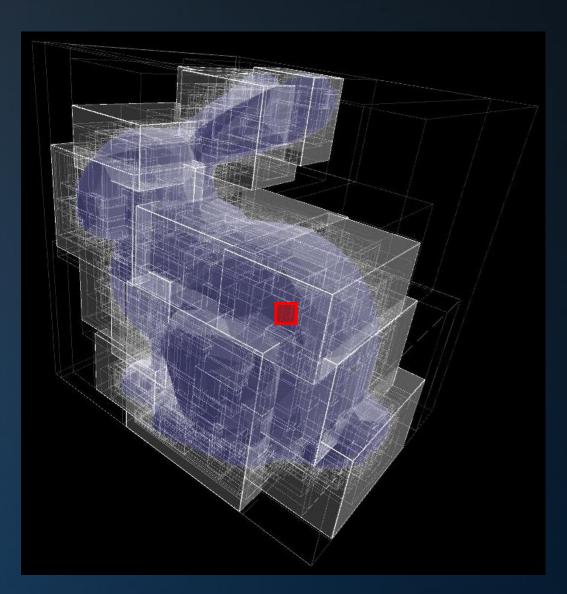
= true;

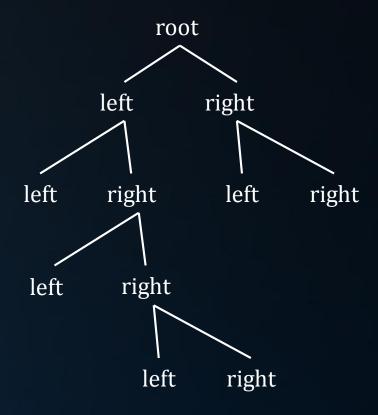
MAXDEPTH)

v = true;

Fast Traversal

```
), N );
= true;
        at = dot(13);
ght * cosThetaOut)
MAXDEPTH)
survive = SurvivalProbability( dif
st3 factor = diffuse IN (1)
st weight = Mis2( dir (1));
st cosThetaOut = dot( (1));
andom walk - done pro
/ive)
st3 brdf = SampleDiffuse( diffuse, N, r1, r2, NR, No.
pdf;
n = E * brdf * (dot( N, R ) / pdf);
```







Fast Traversal

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*.

```
AXXDEPTH)

Survive = SurvivalProbability difference estimation - doing it properly diff;

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

y = true;
st brdfPdf = EvaluateDiffus
stat weight = Mis2( directly) difference diffuse * IN (I)
st weight = Mis2( directly) difference dot (N);
E * ((weight * cosThetaOut) / d.
sendom walk - done properly dive)

stat brdf = SampleDiffuse( diffuse, N, r1, r2, &R, page diffuse)
```

efl + refr)) && (depth <)

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

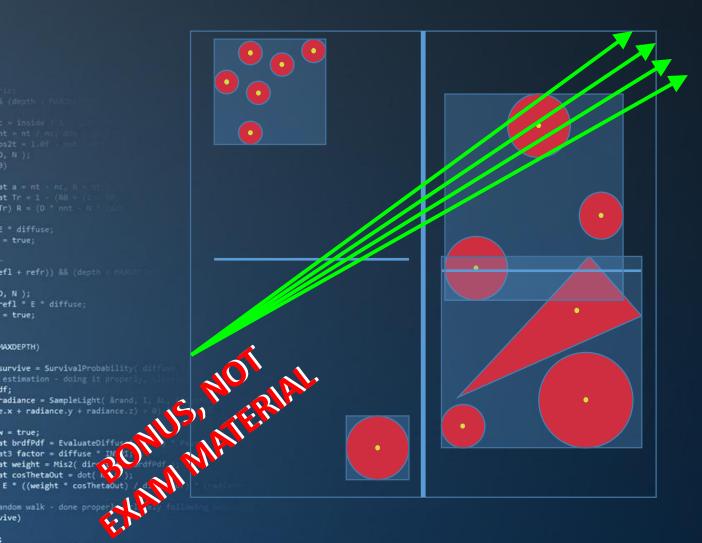
), N);



Fast Traversal

st3 brdf = SampleDiffuse(diffuse, N, r1, r2, RR)

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



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

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:

ray we check first while processing the child nodes.



efl * E * diffuse; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, U

Fast Traversal

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

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.





Today's Agenda:

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



```
st weight = Mis2( directPdf, brdfPdf );
st cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf) * (rudling sindom walk - done properly, closely following sindom walk - done properly sindom walk - done pr
```

efl + refr)) && (depth x HA

survive = SurvivalProbability(diff.

radiance = SampleLight(&rand, I, 8.) e.x + radiance.y + radiance.z) > 0) 88

st brdfPdf = EvaluateDiffuse(L, N) = st3 factor = diffuse = INVPI:

), N);

= true;

MAXDEPTH)

v = true;

Mid-term Exam

f1 + refr)) && (depth

andom walk - done properly, closely fol

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

st3 brdf = SampleDiffuse(diffuse, N, r1, r2, LR)

efl * E * diffuse;

= true;

MAXDEPTH)

What to study for the exam?

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

Expectations:

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

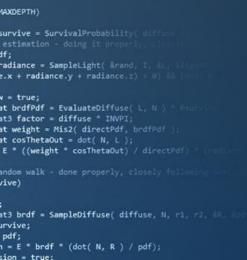
```
of 90 degrees and a screen resolution of 512x512 pixels. (1,0,1), an up vector \vec{i}
                                                                                                                                                                                                                                            Given: an eye position E = (1,1,1), a view vector V = (1,0,1), an up vector V = (1,0,1), and up vector V = (1,0,1), and V = (1,0,1), an
                                                                                                                                                                                                                                                     a. For the purpose of ray tracing, calculate the four corners of a virtual scre
                                                                                                                                                                                                                                                                    distance 1 from E, perpendicular to \vec{V} and \vec{U}\vec{p}, taking into account the specific the specific three specific to \vec{V} and \vec{V}
                                                                                                                                                                                                       b. Determine the normalized direction of ray r, which originates from E and extensi
                                                                                                                                                                                          Determine the 3D coordinate of the intersection of r with the plane: -X-Z=10.
                                                                                                                                        d. Calculate the intersection of r with the sphere ||p-C|| = \sqrt{20}, where C = (-1,1,-1).
                                                                                                           e. Point P = (\sqrt{10} - 1.1, \sqrt{10} - 1) is a point on the sphere ||P - C|| = \sqrt{20}. Calculate the
          2. Culling schemes that mark some objects as visible when they are not (but not the other way
Glass causes two types of scattering. Name both. First type:
```

INFOGR – Computer Graphics

J. Bikker - April-July 2016 - Lecture 7: "Accelerate"

END of "Accelerate"

next up: "Mid-term Exam"



t a = nt - nc

), N); refl * E = true;

efl + refr)) && (depth < HA

