# INFOMAGR – Advanced Graphics

Jacco Bikker - November 2022 - February 2023

# Lecture 5 - "Acceleration Structures"

efl + refr)) && (dept ), N ); refl \* E \* diffuse; (AXDEPTH)

survive = SurvivalProbability( dif

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

andom walk - done properly, closely follo

at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &

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

Welcome!



# Today's Agenda:

- Problem Analysis
- Early Work
- BVH Up Close



```
AXXDEPTH)

Survive = SurvivalProbability( diffuse a sestimation - doing it properly, classes of f;

radiance = SampleLight( &rand, I, &L, &lighter a set and ance.y + radiance.z) > 0) && (duta a set)

v = true;

st brdfPdf = EvaluateDiffuse( L, N ) * Psurvival and factor = diffuse * INVPI;

st weight = Mis2( directPdf, brdfPdf );

st cosThetaOut = dot( N, L );

E * ((weight * cosThetaOut) / directPdf) * (radiance andom walk - done properly, closely following Samularive)

stata brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pd
urvive;

pdf;

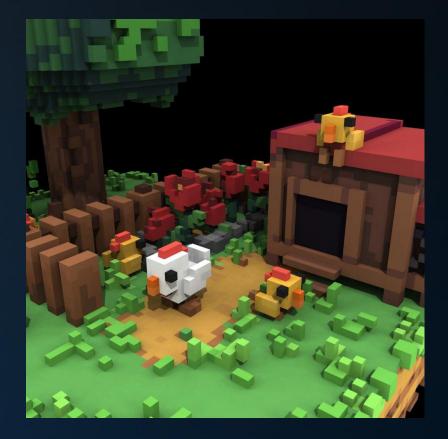
n = E * brdf * (dot( N, R ) / pdf);

sion = true:
```

```
(AXDEPTH)
survive = SurvivalProbabil
radiance = SampleLight( &r
```

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

"Cornell Box"



Voxel game



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

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

v = true;

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



```
w = true;
at brdfPdf = EvaluateDiffuse( L, N ) Resumme
at3 factor = diffuse * INVPI;
at weight = Mis2( directPdf, brdfPdf );
but cosThetaOut = dot( N, L );
but weight * cosThetaOut) / directPdf) * (radianal
andom walk - done properly, closely following security)

andom walk - done properly, closely following security

andom walk - done properly, closely following security

and but = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
```



Avengers Endgame



(AXDEPTH)

survive = SurvivalProbability( diff)

radiance = SampleLight( &rand, I, &L

e.x + radiance.y + radiance.z) > 0) 8

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

#### Characteristics

#### Rasterization:

- Games
- Fast
- Realistic
- Consumer hardware

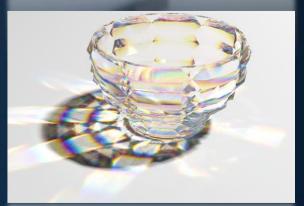
#### Ray Tracing:

- Movies
- Slow
- Very Realistic
- Supercomputers





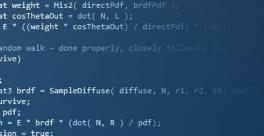












#### Characteristics

#### Reality:

- everyone has a budget
- bar must be raised
- *we need to optimize.*

# NOF THE SHIT OF IT

#THEMARTIAN

Cost Breakdown for Ray Tracing:

- Pixels
- Primitives
- Light sources
- Path segments

Mind scalability as well as constant cost.

Example: scene consisting of 1k spheres and 4 light sources, diffuse materials, rendered to 1M pixels:

 $1M \times 5 \times 1k = 5 \cdot 10^9$  ray/prim intersections. (multiply by desired framerate for realtime)



nt = nt / nc, ddn = 1
ps2t = 1.0f - nnt = nnt
ps2t = nt - nc, b = nt
pst Tr = 1 - (R0 + (1 - n)
pst Tr = 1 - (R0 + (1 - n)
pst Tr = 1 - (R0 + (1 - n)
pst Tr = nnt - N = nnt
pst Tr = nnt - N = nnt
pst Tr = nnt - nnt
pst Tr = nnt
pst Tr = nnt - nnt
pst Tr = nnt - nnt
pst Tr = nnt
pst T

(AXDEPTH)

estimation - doin df; radiance = SampleL e.x + radiance.y +

w = true; at brdfPdf = Evalua at3 factor = diffus at weight = Mis2( d at cosThetaOut = do

andom walk - done properly, clo vive)

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

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

efl + refr)) && (depth o

survive = SurvivalProbability( di

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

at brdfPdf = EvaluateDiffuse( L, N ) at3 factor = diffuse \* INVPI; at weight = Mis2( directPdf, brdfPdf

refl \* E \* diffuse;

), N );

(AXDEPTH)

v = true;

#### Optimizing Ray Tracing

#### Options:

- 1. Faster intersections (reduce constant cost)
- Faster shading (reduce constant cost)
- 3. Use more expressive primitives (trade constant cost for algorithmic complexity)
- 4. Fewer of ray/primitive intersections (reduce algorithmic complexity)

#### Note for option 1:

At 5 billion ray/primitive intersections, we will have to bring down the cost of a single intersection to 1 cycle on a 5Ghz CPU – if we want one frame per second.

```
Crysis, 2007
```

```
at cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf) * (radians)
andom walk - done properly, closely following Season
vive)
;
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf
urvive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;
```

# Today's Agenda:

- Problem Analysis
- Early Work
- BVH Up Close



```
AXXDEPTH)

Survive = SurvivalProbability( diffuse a sestimation - doing it properly, classes of f;

radiance = SampleLight( &rand, I, &L, &lighter a set and ance.y + radiance.z) > 0) && (duta a set)

v = true;

st brdfPdf = EvaluateDiffuse( L, N ) * Psurvival and factor = diffuse * INVPI;

st weight = Mis2( directPdf, brdfPdf );

st cosThetaOut = dot( N, L );

E * ((weight * cosThetaOut) / directPdf) * (radiance andom walk - done properly, closely following Samularive)

stata brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pd
urvive;

pdf;

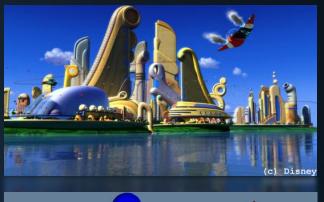
n = E * brdf * (dot( N, R ) / pdf);

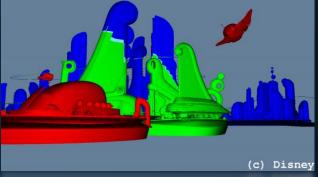
sion = true:
```

#### **Complex Primitives**

More expressive than a triangle:

- Sphere
- Torus
- Teapotahedron
- Bézier surfaces
- Subdivision surfaces\*
- Implicit surfaces\*\*
- Fractals\*\*\*



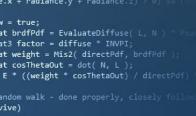






Utah Teapot, Martin Newell, 1975

- \*: Benthin et al., Packet-based Ray Tracing of Catmull-Clark Subdivision Surfaces. 2007.
- \*\*: Knoll et al., Interactive Ray Tracing of Arbitrary Implicits with SIMD Interval Arithmetic. RT'07 Proceedings, Pages 11-18
- \*\*\*: Hart et al., Ray Tracing Deterministic 3-D Fractals. In Proceedings of SIGGRAPH '89, pages 289-296.



survive = SurvivalProbability( dif

(AXDEPTH)

at3 brdf = SampleDiffuse( diffuse, N, r1, urvive; pdf; n = E \* brdf \* (dot( N, R ) / pdf);

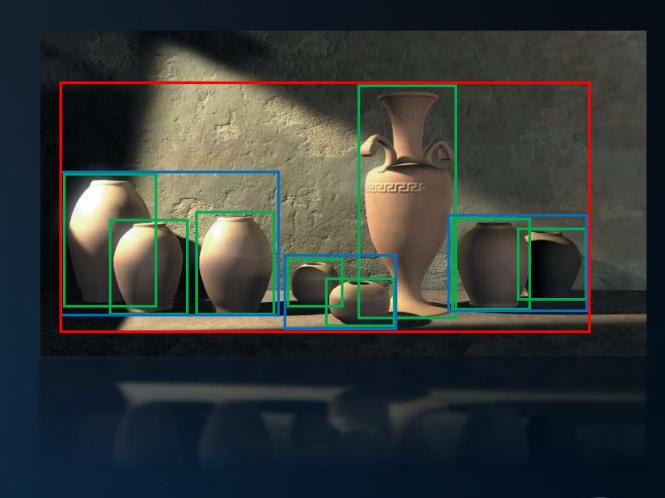
Rubin & Whitted\*

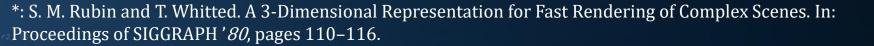
"Hierarchically Structured Subspaces"

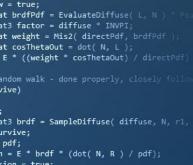
Proposed scheme:

- Manual construction of hierarchy
- Oriented parallelepipeds

A transformation matrix allows efficient Intersection of the skewed / rotated boxes, which can tightly enclose actual geometry.







radiance = SampleLight( &rand)

), N );

(AXDEPTH)

), N );

(AXDEPTH)

v = true;

refl \* E \* diffuse;

survive = SurvivalProbability( dif

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

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

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

E \* ((weight \* cosThetaOut) / directPdf)
andom walk - done properly, closely follo

Amanatides & Woo\*

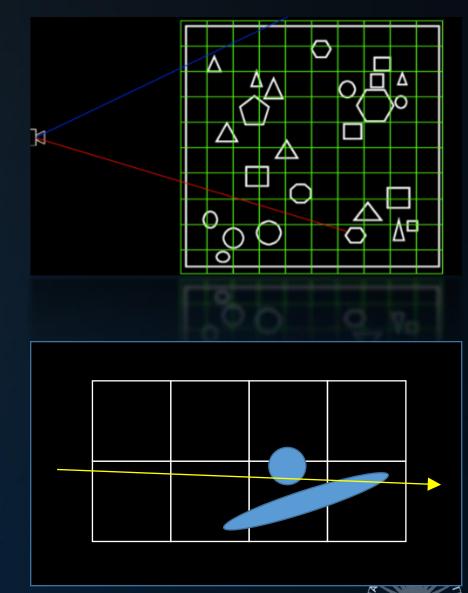
"3DDDA of a regular grid"

The grid can be automatically generated.

Considerations:

- Ensure that an intersection happens in the current grid cell
- Use mailboxing to prevent repeated intersection tests

\*: J. Amanatides and A. Woo. A Fast Voxel Traversal Algorithm for Ray Tracing. In Eurographics '87, pages 3–10, 1987.



Glassner\*

"Hierarchical spatial subdivision"

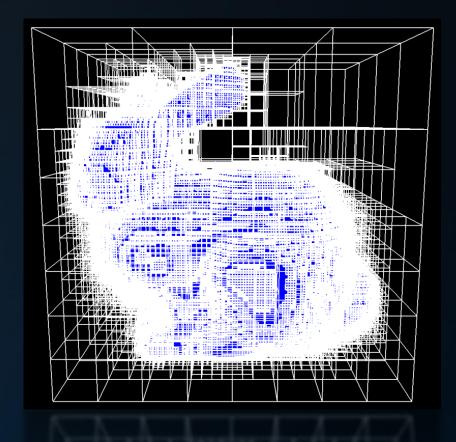
Like the grid, octrees can be automatically generated.

Advantages over grids:

- Adapts to local complexity: fewer steps
- No need to hand-tune grid resolution

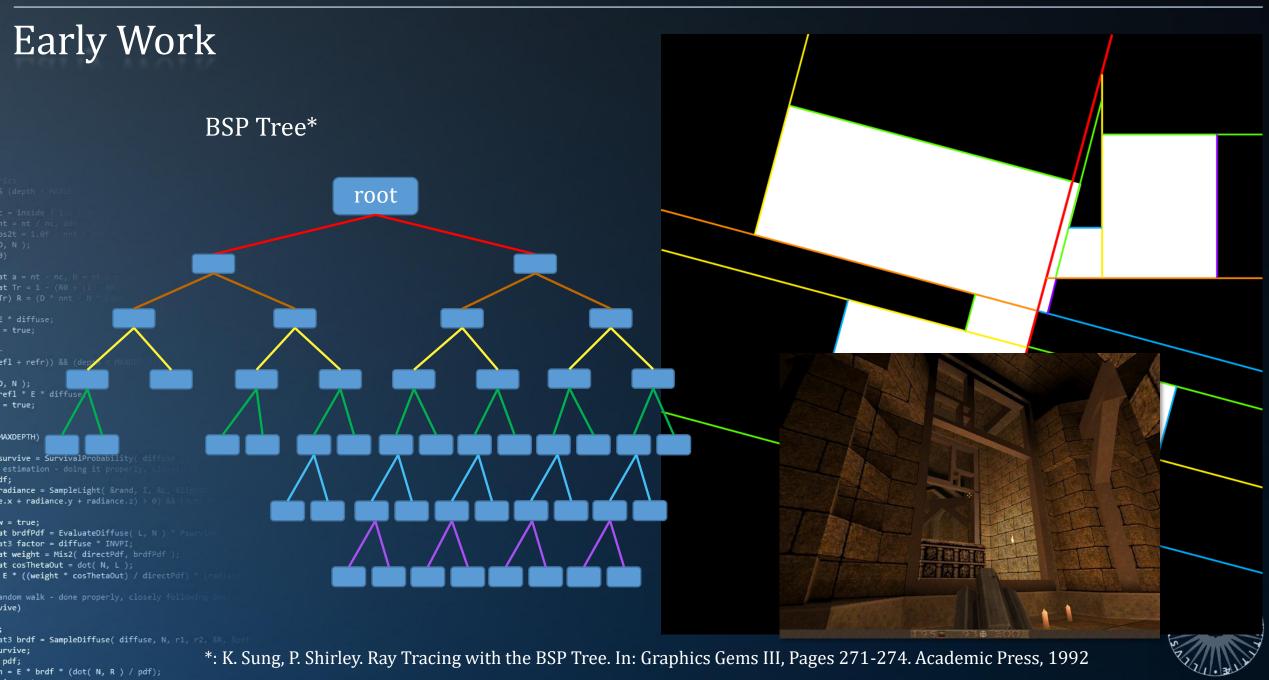
Disadvantage compared to grids:

Expensive traversal steps.



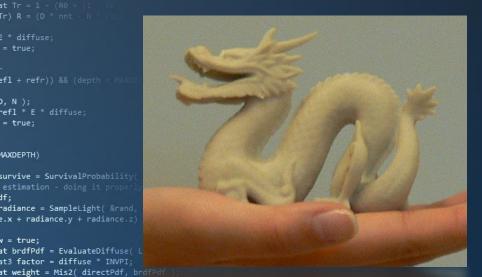


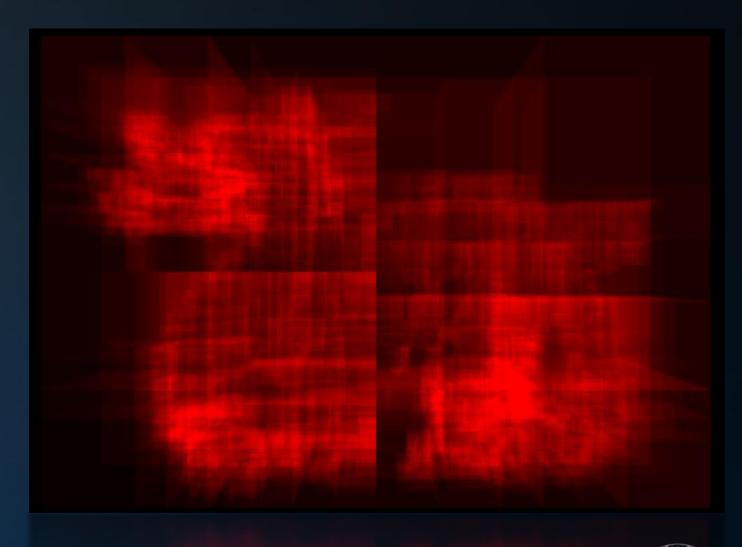
at3 brdf = SampleDiffuse( diffuse, N, r1, urvive;
pdf;
n = E \* brdf \* (dot( N, R ) / pdf);



kD-Tree\*

"Axis-aligned BSP tree"







\*: V. Havran, Heuristic Ray Shooting Algorithms. PhD thesis, 2000.

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

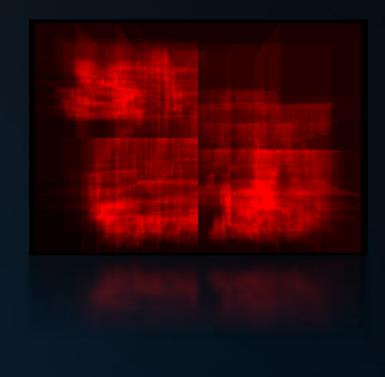
at cosThetaOut = dot( N, L );

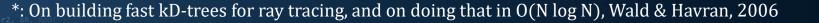
E \* ((weight \* cosThetaOut) / directPdf

#### kD-Tree Construction\*

A kd-tree is a binary tree that recursively subdivides the space occupied by the scene.

- The root corresponds to the axis aligned bounding box (AABB) of the scene:
- Interior nodes represent planes that recursively subdivide space perpendicular to the coordinate axis;
- Leaf nodes store references to all the triangles overlapping the corresponding voxel.





survive = SurvivalProbability( di

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

efl + refr)) && (dept

refl \* E \* diffuse;

(AXDEPTH)

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

at weight = Mis2( directPdf, brdfPdf E \* ((weight \* cosThetaOut) / directPdf

at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &p

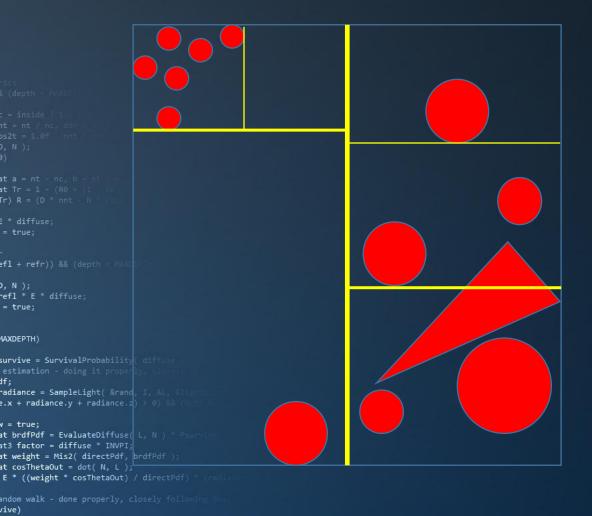
n = E \* brdf \* (dot( N, R ) / pdf);

```
(AXDEPTH)
survive = SurvivalProbability
radiance = SampleLight( &rand
.x + radiance.y + radiance.z
v = true;
at brdfPdf = EvaluateDiffuse(
at3 factor = diffuse * INVPI
at weight = Mis2( directPdf, brdfPdf
at cosThetaOut = dot( N, L )
E * ((weight * cosThetaOut) / directPdf)
andom walk - done properly, closely follow
```

```
function Build( triangles T, voxel V)
   if (Terminate( T , V )) return new LeafNode( T )
   Plane p = FindPlane(T, V)
   Voxel V_L, V_R = Split V with p
   triangles T_L = \{ t \in T \mid (t \cap V_L) \neq 0 \}
   triangles T_R = \{ t \in T \mid (t \cap V_R) \neq 0 \}
   return new InteriorNode(
      p,
      Build(T_L, V_L),
      Build( T_R, V_R )
Function BuildKDTree( triangles T )
   Voxel V = bounds(T)
   return Build( T, V )
```

at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &

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



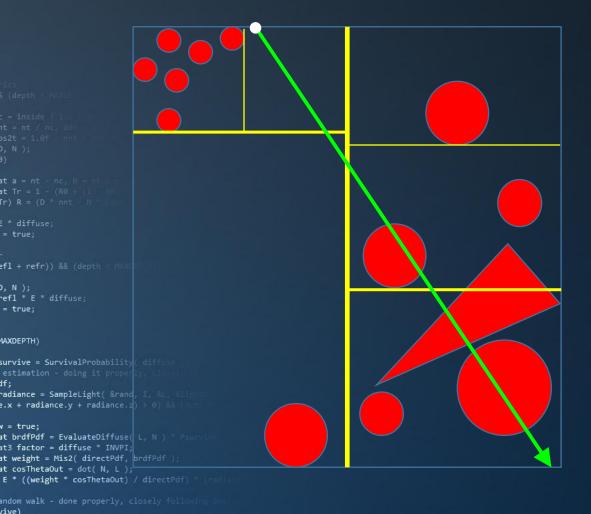
#### Considerations

- Termination
   minimum primitive count, maximum recursion depth
- Storage primitives may end up in multiple voxels: required storage hard to predict
- Empty spaceempty space reduces probability of having to intersect primitives
- Optimal split plane position / axis
   good solutions exist will be discussed later.



at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, I

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



#### Traversal\*

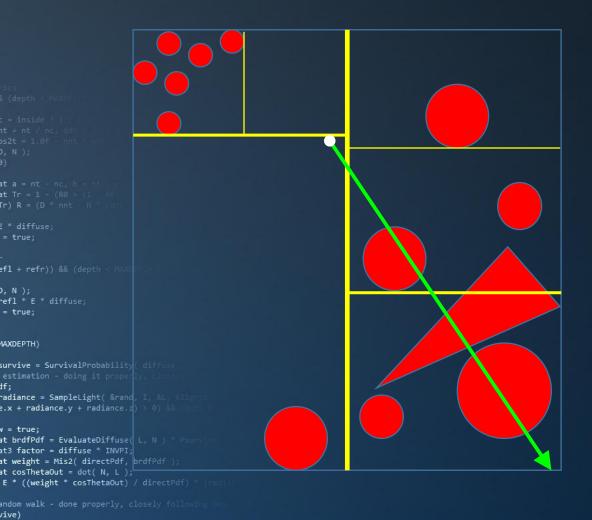
- 1. Find the point *P* where the ray enters the voxel
- 2. Determine which leaf node contains this point
- 3. Intersect the ray with the primitives in the leaf If intersections are found:
  - Determine the closest intersection
  - If the intersection is inside the voxel: done

\*: Space-Tracing: a Constant Time Ray-Tracer, Kaplan, 1994



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

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



#### Traversal\*

- 1. Find the point *P* where the ray enters the voxel
- 2. Determine which leaf node contains this point
- 3. Intersect the ray with the primitives in the leaf If intersections are found:
  - Determine the closest intersection
  - If the intersection is inside the voxel: done
- 4. Determine the point B where the ray leaves the voxel
- 5. Advance P slightly beyond B
- 6. Goto 1.

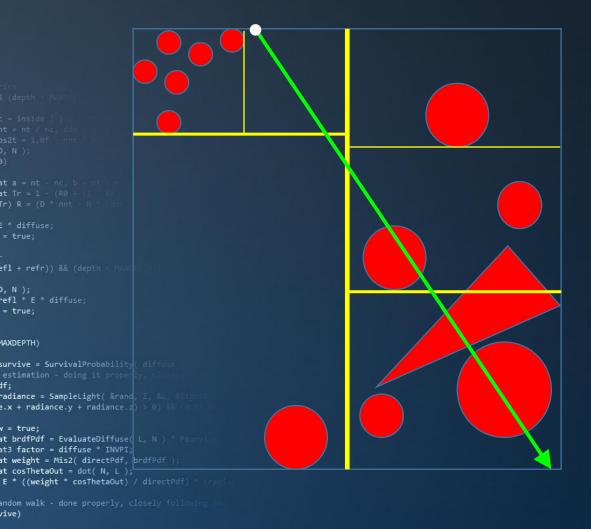
Note: step 2 traverses the tree repeatedly – inefficient.

\*: Space-Tracing: a Constant Time Ray-Tracer, Kaplan, 1994



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

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



Traversal – Alternative Method\*

#### For interior nodes:

- 1. Determine 'near' and 'far' child node
- Determine if ray intersects 'near' and/or 'far'
   If only one child node intersects the ray:
  - Traverse the node (goto 1)Else (both child nodes intersect the ray):
    - Push 'far' node to stack
    - Traverse 'near' node (goto 1)

#### For leaf nodes:

- 1. Determine the nearest intersection
- 2. Return if intersection is inside the voxel.

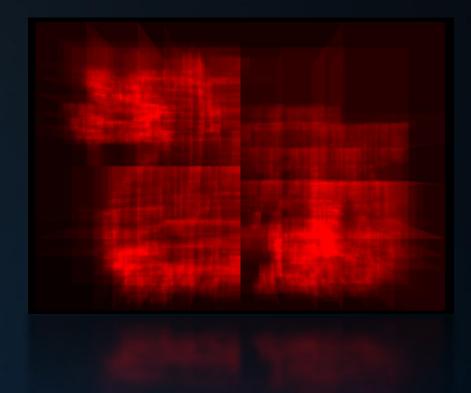


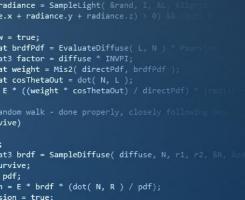
<sup>\*:</sup> Data Structures for Ray Tracing, Jansen, 1986.

kD-Tree Traversal

Traversing a kD-tree is done in a strict order.

Ordered traversal means we can stop as soon as we find a valid intersection.





survive = SurvivalProbability( diff.

(AXDEPTH)



andom walk - done properly, closely followi vive)

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

at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &po urvive;

	Acceleration Structures	Partitioning	Construction	Quality	
ties				Quarty	
& (depth < PMXDEFTH	2				
: = inside	<ul><li>Grid</li></ul>	space	0(n)	low	
os2t = 1.0f - nmt	<ul><li>Octree</li></ul>	space	$O(n \log n)$	medium	
ata = nt - nc, b = nt + r	■ BSP	space	$O(n^2)$	good	
at Tr = 1 - (R0 + (1 - R0) [r] R = (D * not - N * (ddn * )	■ kD-tree	space	O(n log n)	good	
: * diffuse; = true;	■ BVH	object	O(n log n)	good	
- efl + refr)) && (depth < MAXDERSON					
), N ); refl * E * diffuse;	<ul><li>Tetrahedralization</li></ul>	space	?	low	
= true;	• BIH	object	$O(n \log n)$	medium	
1AXDEPTH)					
<pre>survive = SurvivalProbability( diffuse ) estimation - doing it properly, close) if;</pre>					
radiance = SampleLight( &rand, I, &L, ≪ .x + radiance.y + radiance.z) > 0) && c					
<pre>v = true; st brdfPdf = EvaluateDiffuse( L, N ) * Ps st3 factor = diffuse * INVPI; st weight = Mis2( directPdf, brdfPdf ); st cosThetaOut = dot( N, L ); E * ((Weight * cosThetaOut) / directPdf)</pre>					



# Today's Agenda:

- Problem Analysis
- Early Work
- BVH Up Close



```
AXXDEPTH)

Survive = SurvivalProbability( diffuse a sestimation - doing it properly, classes of f;

radiance = SampleLight( &rand, I, &L, &lighter a set and ance.y + radiance.z) > 0) && (duta a set)

v = true;

st brdfPdf = EvaluateDiffuse( L, N ) * Psurvival and factor = diffuse * INVPI;

st weight = Mis2( directPdf, brdfPdf );

st cosThetaOut = dot( N, L );

E * ((weight * cosThetaOut) / directPdf) * (radiance andom walk - done properly, closely following Samularive)

stata brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pd
urvive;

pdf;

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

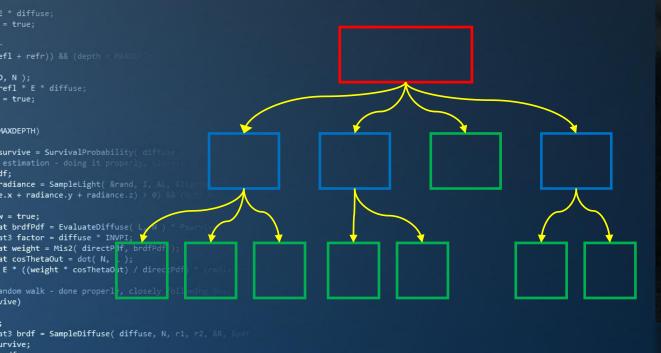
sion = true:
```

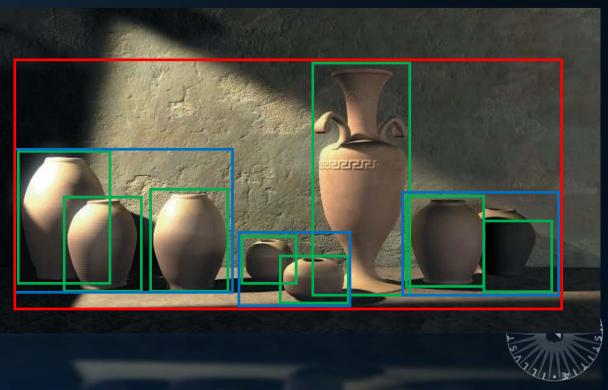
1 = E \* brdf \* (dot( N, R ) / pdf);

#### Automatic Construction of Bounding Volume Hierarchies

BVH: tree structure, with:

- a bounding box per node
- pointers to child nodes
- geometry at the leaf nodes





at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &

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

#### Automatic Construction of Bounding Volume Hierarchies

BVH: tree structure, with:

- a bounding box per node
- pointers to child nodes
- geometry at the leaf nodes

```
= true;

cfl + refr)) && (depth < MAXDEPIN)

O, N );

refl * E * diffuse;
= true;

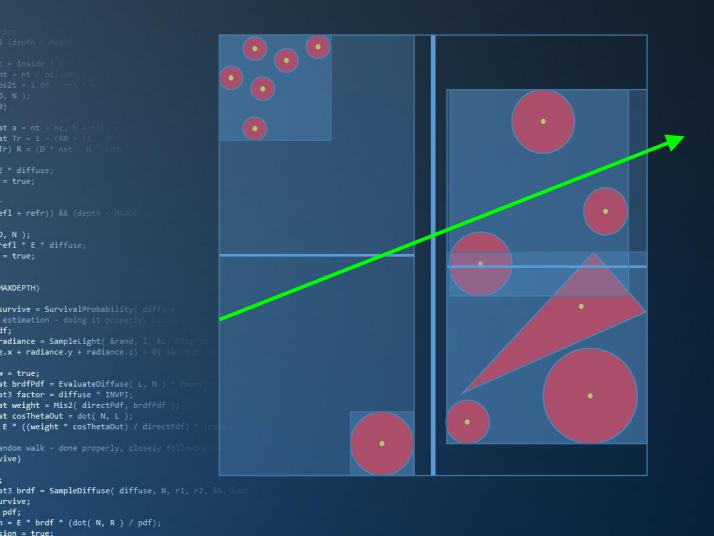
MAXDEPTH)

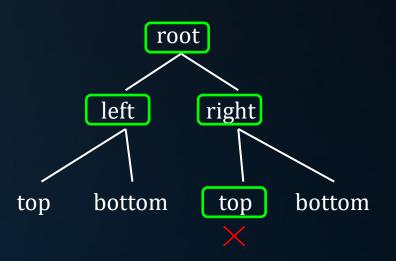
Survive = SurvivalProbability( diffuse;
estination - doing it properly, closes and
if;
radiance = SampleLight( &rand, I, &L, &light
e.x + radiance.y + radiance.z) > 0) && (dot n)

v = true;
at brdfdf = EvaluateDiffuse( L, I) * Psurviv
sit weight = Mis2( directPlf, brdfPdf);
at cosThetaOut = dot( N, .);
E * ((Weight * cosThetaOut) / directPdf) * (radiandom walk - done properly, closely following Ses
```

```
struct BVHNode
{
    AABB bounds;
    bool isLeaf;
    BVHNode*[] child;
    Primitive*[] primitive;
};
```









), N );

(AXDEPTH)

v = true;

refl \* E \* diffuse;

survive = SurvivalProbability( diff

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

at brdfPdf = EvaluateDiffuse( L, N

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

at3 factor = diffuse \* INVPI; at weight = Mis2( directPdf, brdfPdf ) at cosThetaOut = dot( N, L );

#### Automatic Construction of Bounding Volume Hierarchies



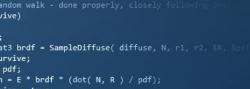
- 1. Determine AABB for primitives in array
- 2. Determine split axis and position
- 3. Partition
- 4. Repeat steps 1-3 for each partition

#### Note:

Step 3 can be done 'in place'.

This process is identical to QuickSort: the split plane is The 'pivot'.





andom walk - done properly, closely follow

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

at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pd

```
12
                           struct BVHNode
                                 AABB bounds;
                                                                                 // 24 bytes
refl * E * diffuse;
                                 bool isLeaf;
                                                                                // 4 bytes
                                 BVHNode* left, *right; // 8 or 16 bytes
(AXDEPTH)
survive = SurvivalProbability( diff
                                 Primitive** primList; // ? bytes
radiance = SampleLight( &rand, I, &L
e.x + radiance.y + radiance.z) > 0) 8
v = true;
at brdfPdf = EvaluateDiffuse( L, N )
at3 factor = diffuse * INVPI;
at weight = Mis2( directPdf, brdfPdf )
at cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf)
```



at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pd

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





at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, A

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

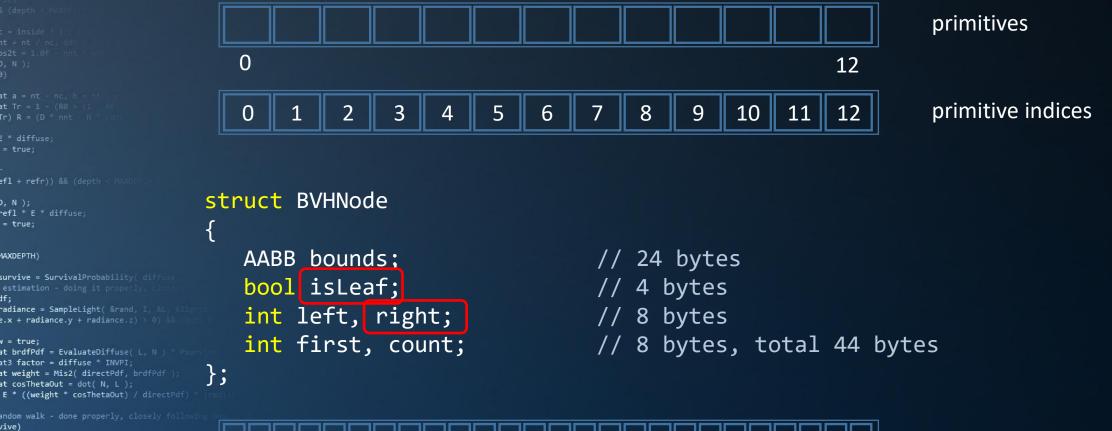
```
void BVH::ConstructBVH( Primitive* primitives )
                                                                                                      void BVHNode::Subdivide()
                            // create index array
                                                                                                          if (count < 3) return;</pre>
                            indices = new uint[N];
                                                                                                          this.left = new BVHNode();
                            for( int i = 0; i < N; i++ ) indices[i] = i;</pre>
                                                                                                          this.right = new BVHNode();
                                                                                                          Partition();
                            // allocate BVH root node
                                                                                                          this.left->Subdivide();
efl + refr)) && (depth :
), N );
                            root = new BVHNode();
                                                                                                          this.right->Subdivide();
refl * E * diffuse;
                                                                                                          this.isLeaf = false;
(AXDEPTH)
                            // subdivide root node
survive = SurvivalProbability( dif
                            root->first = 0;
                            root->count = N;
radiance = SampleLight( &rand, I,
e.x + radiance.y + radiance.z) >
                            root->bounds = CalculateBounds( primitives, root->first, root->count );
v = true;
at brdfPdf = EvaluateDiffuse( |
at3 factor = diffuse * INVPI
                            root->Subdivide();
at weight = Mis2( directPdf, brdfPdf
at cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf)
andom walk - done properly, closely follo
```

```
void BVH::ConstructBVH( Primitive* primitives )
                                                                                                         void BVHNode::Subdivide()
                            // create index array
                                                                                                             if (count < 3) return;</pre>
                            indices = new uint[N];
                                                                                                             this.left = &pool[poolPtr++];
                                                                                                             this.right = &pool[poolPtr++];
                            for( int i = 0; i < N; i++ ) indices[i] = i;</pre>
                                                                                                             Partition();
efl + refr)) && (depth <
                            // allocate BVH root node
                                                                                                             this.left->Subdivide();
), N );
                             pool = new BVHNode[N * 2 - 1];
                                                                                                             this.right->Subdivide();
refl * E * diffuse;
                            root = &pool[0];
                                                                                                             this.isLeaf = false;
(AXDEPTH)
                             poolPtr = 2;
survive = SurvivalProbability( dif
radiance = SampleLight( &rand, I, &
                             // subdivide root node
e.x + radiance.y + radiance.z) >
                             root->first = 0;
v = true;
at brdfPdf = EvaluateDiffuse( L.
at3 factor = diffuse * INVPI
                             root->count = N;
at weight = Mis2( directPdf, brdfPdf )
at cosThetaOut = dot( N, L );
                             root->bounds = CalculateBounds( primitives, root->first, root->count );
E * ((weight * cosThetaOut) / directPdf
                            root->Subdivide();
andom walk - done properly, closely follo
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, )
n = E * brdf * (dot( N, R ) / pdf);
```

at3 brdf = SampleDiffuse( diffuse, N, r1)

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

#### Automatic Construction of Bounding Volume Hierarchies



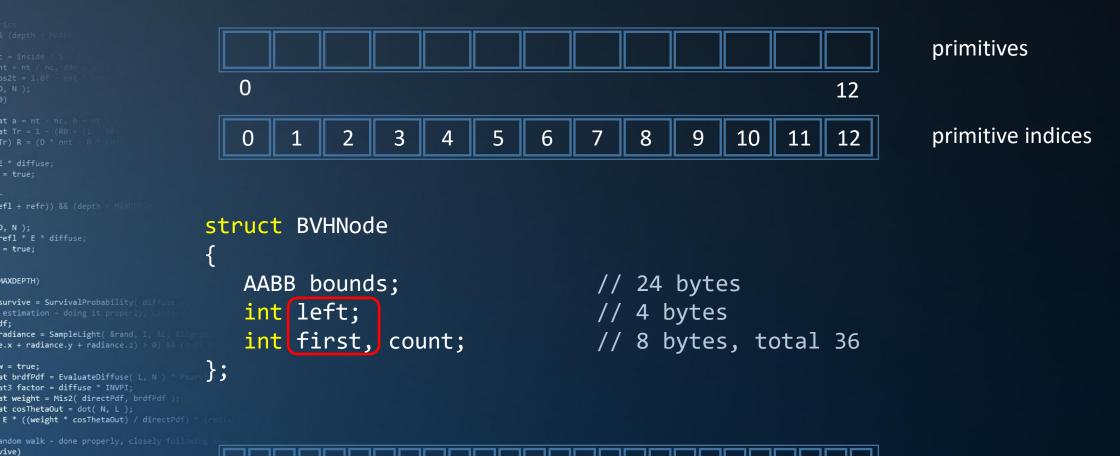
BVH nodes



at3 brdf = SampleDiffuse( diffuse, N, r1,

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

#### Automatic Construction of Bounding Volume Hierarchies



**BVH** nodes



(AXDEPTH)

v = true;

survive = SurvivalProbability( diff

radiance = SampleLight( &rand, I, &L

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

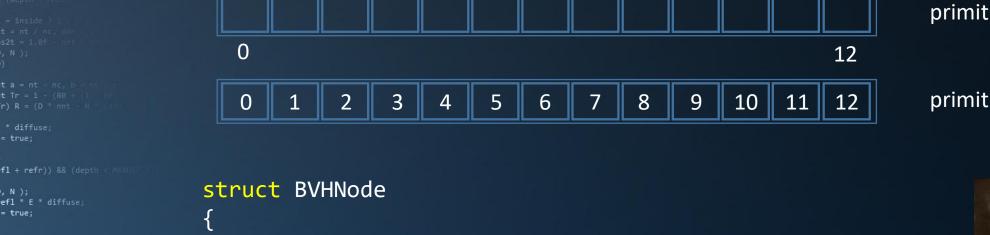
E \* ((weight \* cosThetaOut) / directPdf) andom walk - done properly, closely follo

at3 brdf = SampleDiffuse( diffuse, N, r1)

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

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

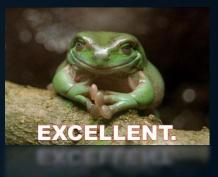
#### Automatic Construction of Bounding Volume Hierarchies



primitives

primitive indices

```
// 24 bytes
AABB bounds;
int leftFirst;
                           // 4 bytes
                           // 4 bytes, total 32 😊
int count;
```



**BVH** nodes



), N );

(AXDEPTH)

v = true;

refl \* E \* diffuse;

survive = SurvivalProbability( diff

radiance = SampleLight( &rand, I, &L e.x + radiance.y + radiance.z) > 0) Automatic Construction of Bounding Volume Hierarchies

Optimal BVH representation:

- Partitioning of array of indices pointing to original triangles
- Using indices of BVH nodes, and assuming right = left + 1
- BVH nodes use exactly 32 bytes (2 per cache line)
- BVH node pool allocated in cache aligned fashion
- AABB splitted in 2x 12 bytes; 1<sup>st</sup> followed by 'leftFirst', 2<sup>nd</sup> by 'count'.

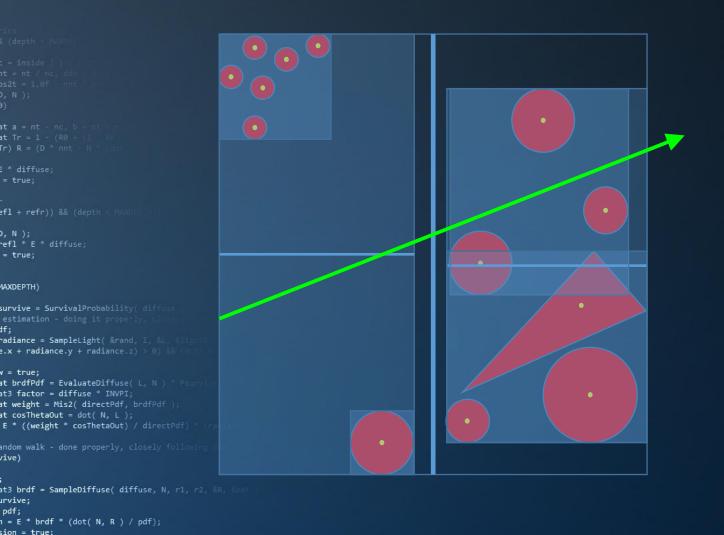
Note: the BVH is now 'relocatable' and thus 'serializable'.

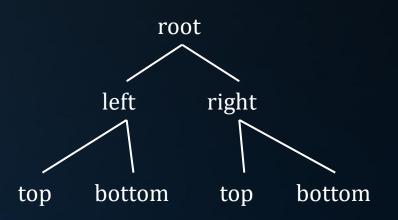
```
at brdfPdf = EvaluateDiffuse( L, N ) * Psurvivality factor = diffuse * INVPI; at weight = Mis2( directPdf, brdfPdf); at cosThetaOut = dot( N, L ); E * ((weight * cosThetaOut) / directPdf) * (radial andom walk - done properly, closely following servive)

at 3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, furvive; pdf; a = E * brdf * (dot( N, R ) / pdf); sion = true:
```



#### **BVH Traversal**







```
BVH Traversal
                                                                                       Ray:
                              Basic process:
                                                                                       vec3 O, D
                              BVHNode::Traverse( Ray r )
                                                                                       float t
                                   if (!r.Intersects( bounds )) return;
                                   if (isleaf())
                                        IntersectPrimitives();
                                   else
(AXDEPTH)
survive = SurvivalProbability( diff)
                                        pool[left].Traverse( r );
radiance = SampleLight( &rand, I, &L
                                        pool[left + 1].Traverse( r );
e.x + radiance.y + radiance.z) > 0)
v = true;
at brdfPdf = EvaluateDiffuse( L
at3 factor = diffuse * INVPI;
at 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, &R, &pd
1 = E * brdf * (dot( N, R ) / pdf);
```

efl + refr)) && (depth

survive = SurvivalProbability( dif

radiance = SampleLight( &rand, I

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

E \* ((weight \* cosThetaOut) / directPdf andom walk - done properly, closely foll

refl \* E \* diffuse;

), N );

(AXDEPTH)

#### **BVH Traversal**

#### Ordered traversal, option 1:

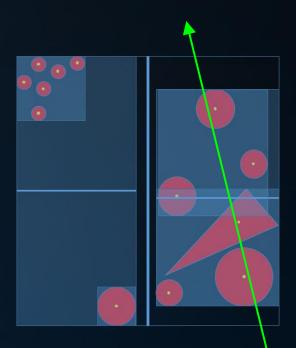
- Calculate distance to both child nodes
- Traverse the nearest child node first

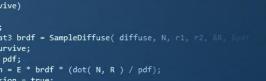
#### Ordered traversal, option 2:

- For each BVH node, store the axis along which it was split
- Use ray direction sign for that axis to determine near and far

#### Ordered traversal, option 3:

- Determine the axis for which the child node centroids are furthest apart
- Use ray direction sign for that axis to determine near and far.





refl \* E \* diffuse;

survive = SurvivalProbability( diff

radiance = SampleLight( &rand, I,

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

at brdfPdf = EvaluateDiffuse( L, N ) ' at3 factor = diffuse \* INVPI; at weight = Mis2( directPdf, brdfPdf )

(AXDEPTH)

v = true;

#### **BVH Traversal**

Ordered traversal of a BVH is approximative.

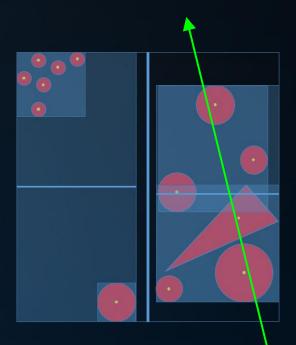
Nodes may overlap.

#### And:

We may find a closer intersection in a node that we visit later.

#### However:

 We do not have to visit nodes beyond an already found intersection distance.



```
at cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf) * (radian);
andom walk - done properly, closely following series
/ive)
;
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &g
urvive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true:
```

### Today's Agenda:

- Problem Analysis
- Early Work
- BVH Up Close



```
AXXDEPTH)

Survive = SurvivalProbability( diffuse a sestimation - doing it properly, classes of f;

radiance = SampleLight( &rand, I, &L, &lighter a set and ance.y + radiance.z) > 0) && (duta a set)

v = true;

st brdfPdf = EvaluateDiffuse( L, N ) * Psurvival and factor = diffuse * INVPI;

st weight = Mis2( directPdf, brdfPdf );

st cosThetaOut = dot( N, L );

E * ((weight * cosThetaOut) / directPdf) * (radiance andom walk - done properly, closely following Samularive)

stata brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pd
urvive;

pdf;

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

sion = true:
```

# INFOMAGR – Advanced Graphics

Jacco Bikker - November 2022 - February 2023

# END of "Acceleration Structures"

next lecture: "The Perfect BVH"

```
at cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf) * (radial
andom walk - done properly, closely following Same
vive)
;
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, i
urvive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;
```

fl + refr)) && (depth <

survive = SurvivalProbability( diff

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

at brdfPdf = EvaluateDiffuse( L, N )
at3 factor = diffuse \* INVPI;
at weight = Mis2( directPdf, brdfPdf

), N );

(AXDEPTH)

v = true;

