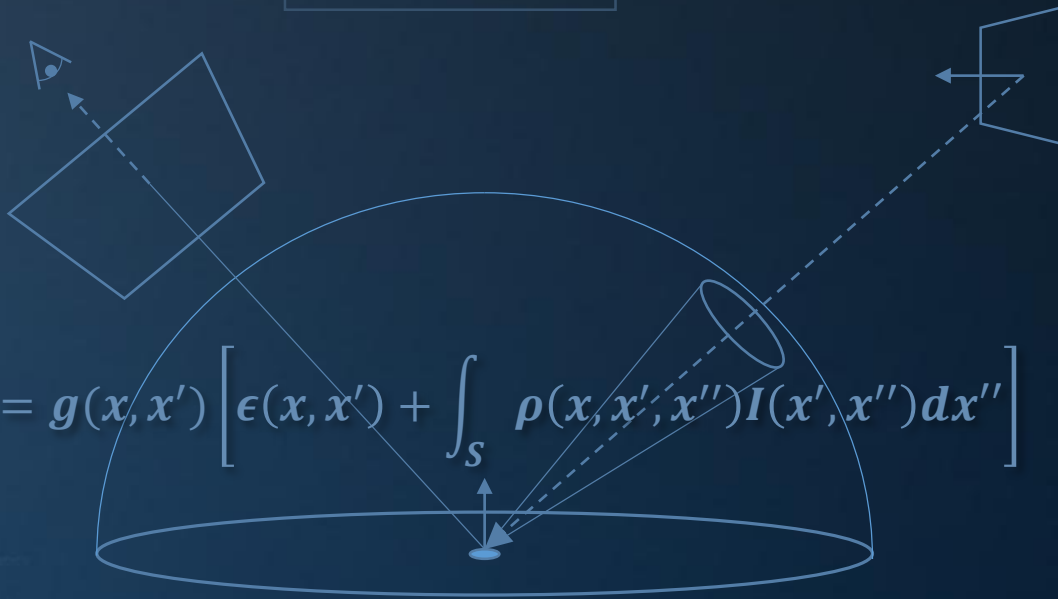


INFOMAGR – Advanced Graphics

Jacco Bikker - February – April 2016

Welcome!



$$I(x, x') = g(x, x') \left[\epsilon(x, x') + \int_S \rho(x, x', x'') I(x', x'') dx'' \right]$$



Today's Agenda:

- Building Better BVHs
- The Problem of Large Polygons
- Refitting
- Fast BVH Construction





MAXDEPTH)



Better BVHs

```

    if (depth < MAXDEPTH) {
        // Inside the sphere
        Vec inside = L - S;
        double nt = nt / nc, dde = dde / nc;
        double r2s2t = 1.0f - nnt * dde;
        Vec D, N );
    }

    Vec a = nt - nc, b = nt + nc;
    double Tr = 1 - (RB + (1 - RB) * r2s2t);
    double Tr) R = (D * nnt - N * dde);

    E * diffuse;
    = true;

    if (refl + refr) && (depth < MAXDEPTH) {
        D, N );
        refl * E * diffuse;
        = true;

    MAXDEPTH)

    survive = SurvivalProbability( diffuse );
    // estimation - doing it properly, close to
    if;
    radiance = SampleLight( &rand, I, &t, &light );
    e.x + radiance.y + radiance.z) > 0) && (depth <
    v = true;
    at brdfPdf = EvaluateDiffuse( L, N ) * Psurvive;
    at3 factor = diffuse * INVPI;
    at weight = Mis2( directPdf, brdfPdf );
    at cosThetaOut = dot( N, L );
    E * ((weight * cosThetaOut) / directPdf) * (radiance
    random walk - done properly, closely following
    (survive)

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



Better BVHs

What Are We Trying To Solve?

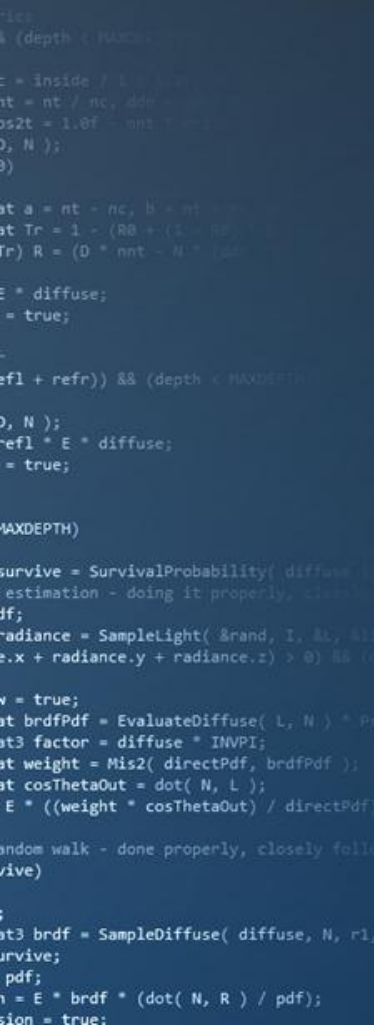
A BVH is used to reduce the number of ray/primitive intersections.

But: it introduces new intersections.

The ideal BVH minimizes:

- # of ray / primitive intersections
- # of ray / node intersections.





Better BVHs

BVH versus kD-tree

The BVH better encapsulates geometry.

→ This reduces the chance of a ray hitting a node.

→ This is all about probabilities!

What is the probability of a ray hitting a random triangle?

What is the probability of a ray hitting a random node?

This probability is proportional to **surface area**.



Better BVHs

Optimal Split Plane Position

The ideal split minimizes the cost of a ray intersecting the resulting nodes.

This cost depends on:

- Number of primitives that will have to be intersected
- Probability of this happening

The cost of a split is thus:

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



Better BVHs

Optimal Split Plane Position

The ideal split minimizes the cost of a ray intersecting the resulting nodes.

This cost depends on:

- Number of primitives that will have to be intersected
- Probability of this happening

The cost of a split is thus:

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



Better BVHs

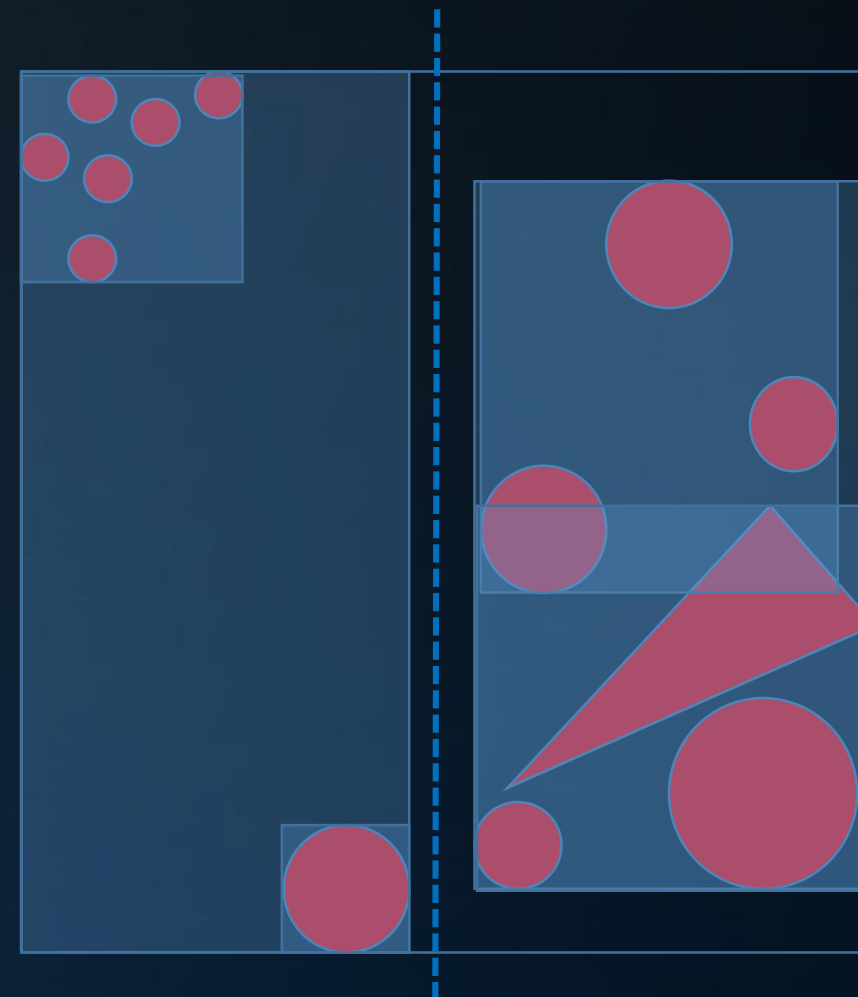
Optimal Split Plane Position

Or, more concisely:

$$A_{left}^0 * (A_{left}^1 * N_{left}^1 + A_{right}^1 * N_{right}^1)$$

+

$$A_{right}^0 * (A_{left}^2 * N_{left}^2 + A_{right}^2 * N_{right}^2)$$



Better BVHs

Optimal Split Plane Position

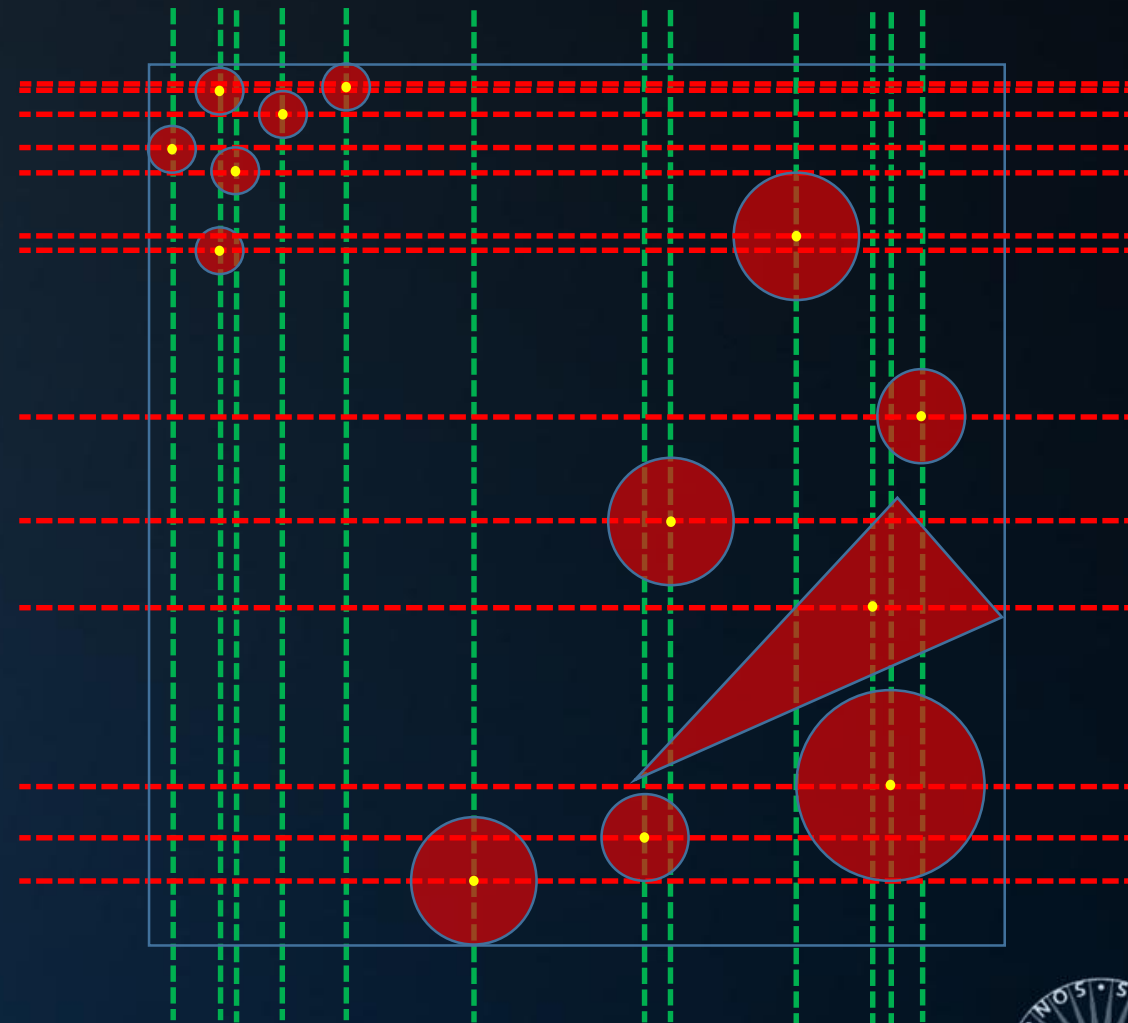
Which positions do we consider?

Object subdivision may happen over x , y or z axis.

The cost function is constant between primitive centroids.

➔ For N primitives: $3(N - 1)$ possible locations

➔ For a 2-level tree: $(3(N - 1))^2$ configurations



Better BVHs

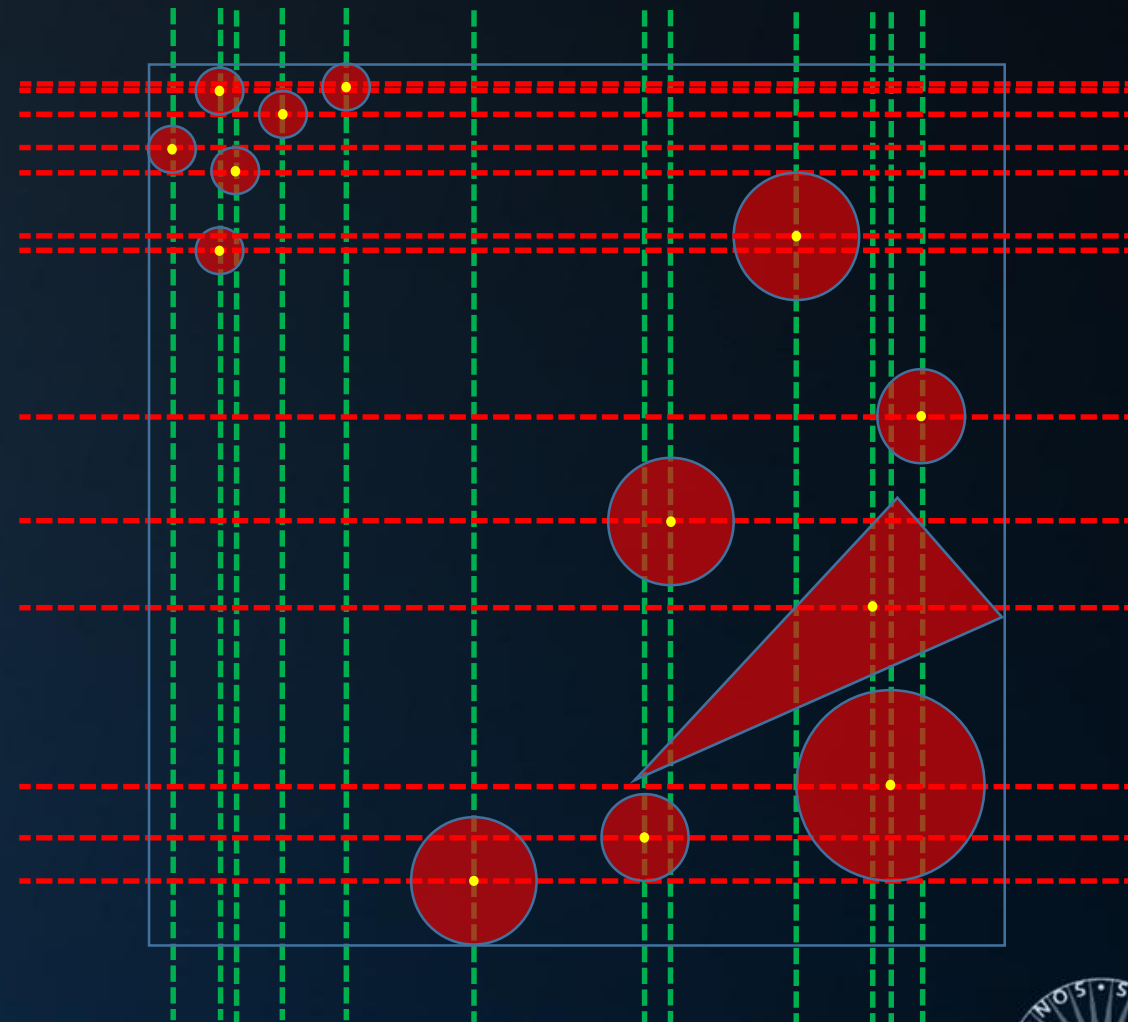
SAH and Termination

A split is ‘not worth it’ if it doesn’t yield a cost lower than the cost of the parent node, i.e.:

$$A_{left} * N_{left} + A_{right} * N_{right} \geq A * N$$

This provides us with a natural and optimal termination criterion.

(and it solves the problem of the Bad Artist)



Better BVHs

Perfect BVHs

```

-ics
& (depth < MAXDEPTH) {
    // Inside / Outside test
    nt = nt / nc; cde = cde / nc;
    ps2t = 1.0f - nnt * cde;
    D, N );
}

// Inside / Outside test
at a = nt - nc, b = nt - nc;
at Tr = 1 - (RB + (1 - RB) * a);
Tr) R = (D * nnt - N * cde);

// Inside / Outside test
E * diffuse;
= true;

// Inside / Outside test
efl + refr)) && (depth < MAXDEPTH) {
    D, N );
    refl * E * diffuse;
    = true;

// Inside / Outside test
MAXDEPTH)

survive = SurvivalProbability( diffuse, L );
estimation - doing it properly, close to
if;
radiance = SampleLight( &rand, I, &t, &light );
e.x + radiance.y + radiance.z) > 0) && (depth < MAXDEPTH) {
    v = true;
    at brdfPdf = EvaluateDiffuse( L, N ) * Psurvive;
    at3 factor = diffuse * INVPI;
    at weight = Mis2( directPdf, brdfPdf );
    at cosThetaOut = dot( N, L );
    E * ((weight * cosThetaOut) / directPdf) * (radiance
    random walk - done properly, closely following the
    survive)

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

```

Optimal Split Plane Position

Evaluating $(3(N - 1))^2$ configurations?

Solution: apply the *surface area heuristic* (SAH) lazily*.

*: Heuristics for Ray Tracing using Space Subdivision, MacDonald & Booth, 1990.



- ```
0, N);
refl * E * diffuse;
```

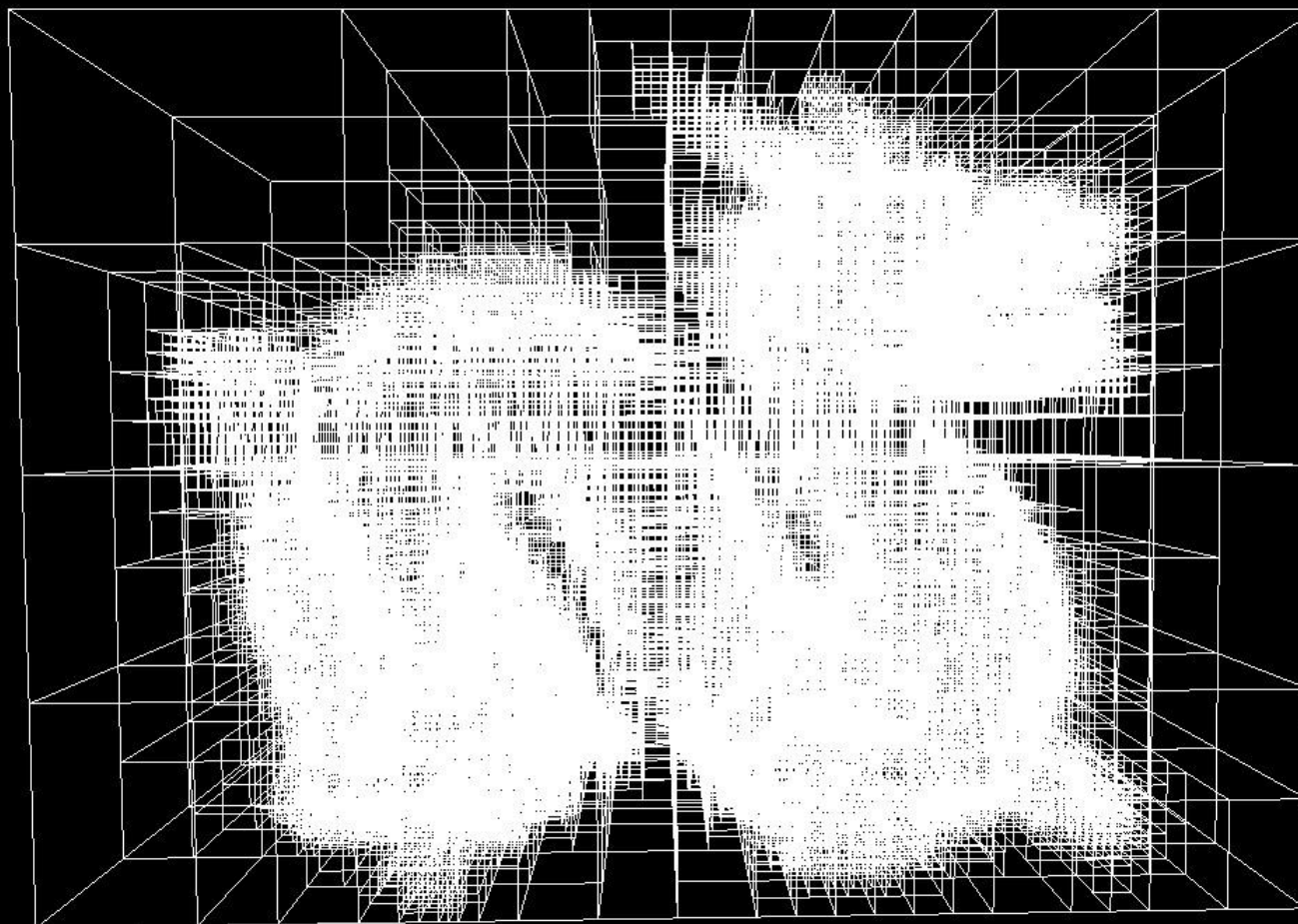


# Better BVHs

```

 if (depth < MAXDEPTH)
 {
 // Inside / Outside
 float inside = 1.0f;
 float nt = nt / nc;
 float cos2t = 1.0f - nt * nt;
 float D, N;
 D = 0;
 N = 0;
 for (int i = 0; i < N; i++)
 {
 float a = nt - nc;
 float b = nt;
 float Tr = 1 - (RB + (L - RB) * cos2t);
 float R = (D * nt - N * (a * cos2t));
 // Diffuse
 // true;
 // Refl + refr
 // (depth < MAXDEPTH)
 D, N;
 refl * E * diffuse;
 // true;
 // MAXDEPTH
 // survive = SurvivalProbability(diffuse
 // estimation - doing it properly, close
 // if;
 // radiance = SampleLight(&rand, 1, &t,
 // e.x + radiance.y + radiance.z) > 0) &&
 // w = true;
 // brdfPdf = EvaluateDiffuse(L, N);
 // factor = diffuse * INVPI;
 // weight = Mis2(directPdf, brdfPdf);
 // cosThetaOut = dot(N, L);
 // E * ((weight * cosThetaOut) / directP
 // random walk - done properly, closely following
 // (survive)
 // brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf
 // survive;
 // pdf;
 // n = E * brdf * (dot(N, R) / pdf);
 // sion = true;

```



## Median Split

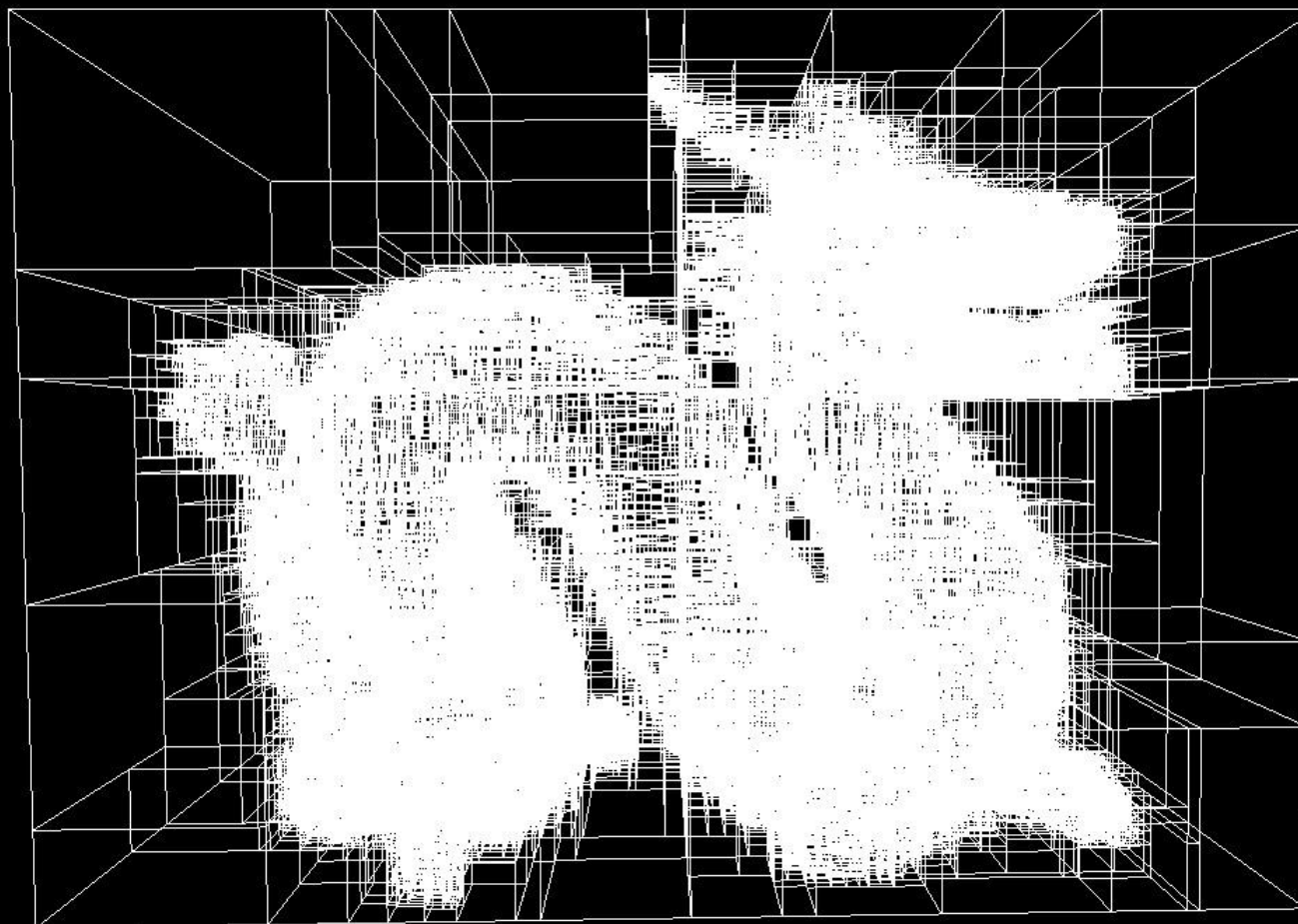


# Better BVHs

```

 if (depth < MAXDEPTH)
 {
 float t = inside / (1.0f - inside);
 float nt = nt / nc;
 float cos2t = 1.0f - nnt;
 float r = (D * N);
 float a = nt - nc;
 float b = nt - nc;
 float Tr = 1 - (RB + (1 - RB) * r);
 float R = (D * nnt - N * (a * r));
 E * diffuse;
 = true;
 refl + refr)) && (depth < MAXDEPTH)
 D, N);
 refl * E * diffuse;
 = true;
 MAXDEPTH)
 survive = SurvivalProbability(diffuse
 estimation - doing it properly, close
 if;
 radiance = SampleLight(&rand, 1, &t,
 e.x + radiance.y + radiance.z) > 0) &&
 w = true;
 at brdfPdf = EvaluateDiffuse(L, N) *
 at3 factor = diffuse * INVPI;
 at weight = Mis2(directPdf, brdfPdf);
 at cosThetaOut = dot(N, L);
 E * ((weight * cosThetaOut) / directP
 random walk - done properly, closely following
 (survive)
 at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf
 survive;
 pdf;
 n = E * brdf * (dot(N, R) / pdf);
 ion = true;

```



## Surface Area Heuristic



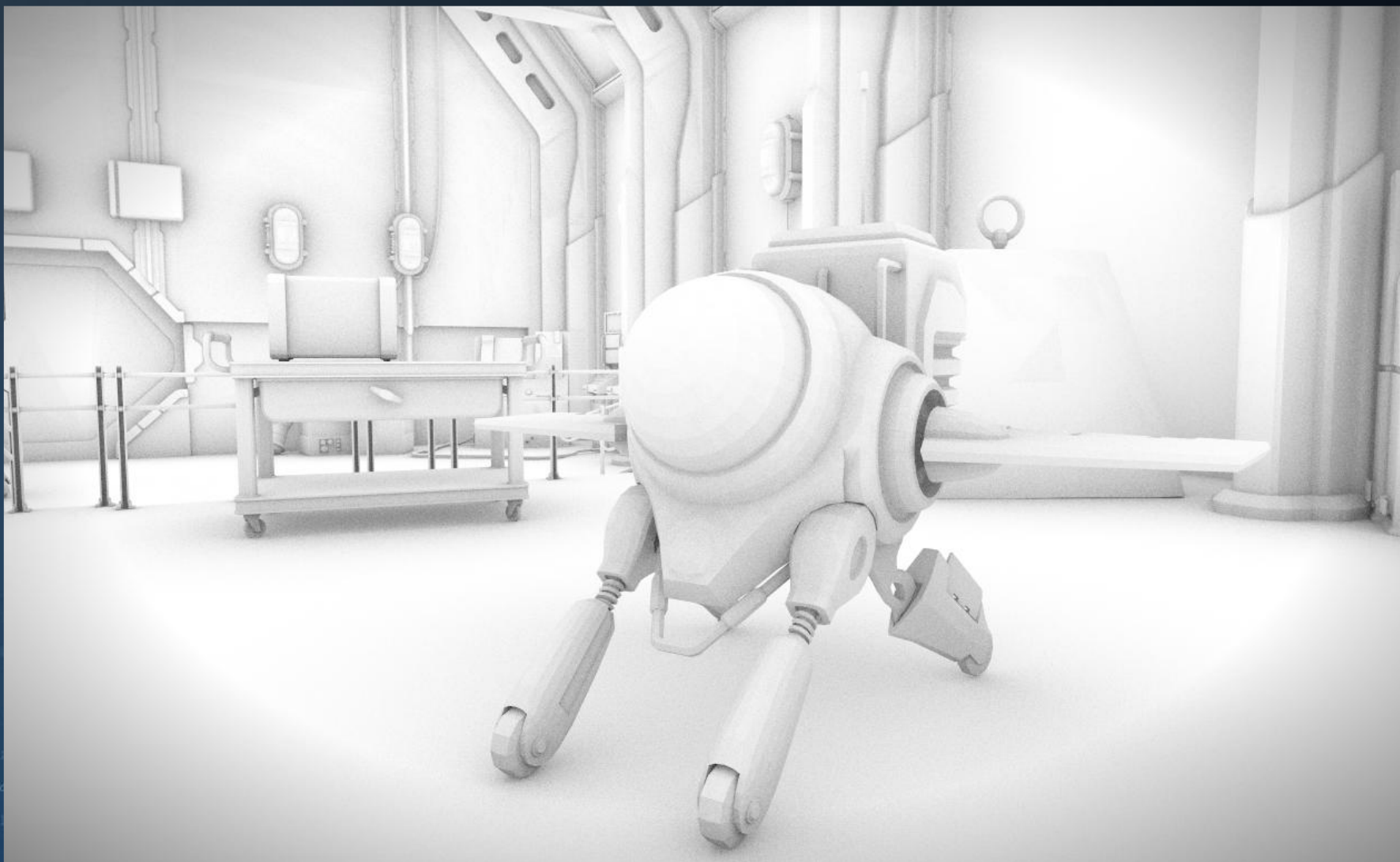


# Better BVHs

```

-ics
& (depth < MAXD)
{
 t = inside / 1.5;
 nt = nt / nc;
 cos2t = 1.0f - nnt;
 D, N);
)
 at a = nt - nc, b = nt - nc;
 at Tr = 1 - (RB + (1 - RB) * t);
 Tr) R = (D * nnt - N * (a * t
 E * diffuse;
 = true;
 efl + refr)) && (depth < MAXDEPTH)
 D, N);
 efl * E * diffuse;
 = true;
 MAXDEPTH)
 survive = SurvivalProbability(diffuse
 estimation - doing it properly, close
 if;
 radiance = SampleLight(&rand, I, &t,
 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) / directPdf
 random walk - done properly, closely fo
 survive)
 at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf
 survive;
 pdf;
 n = E * brdf * (dot(N, R) / pdf);
 sion = true;

```

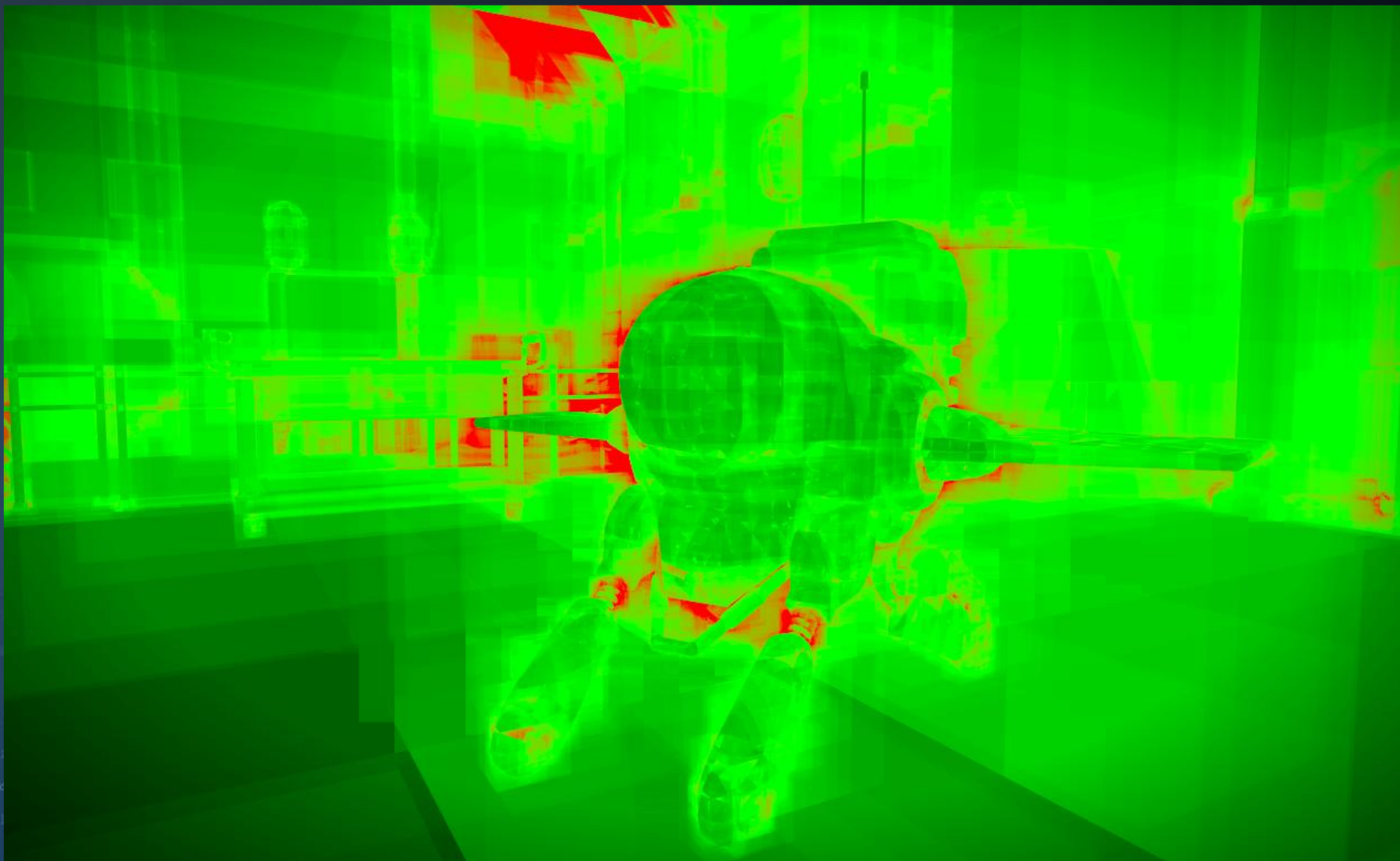


# Better BVHs

```

 if (depth < MAXDEPTH)
 {
 float t = inside / (1.0f - inside);
 float nt = nt / nc;
 float cos2t = 1.0f - nnt;
 float r = sqrt(cos2t);
 float phi = 2.0f * 3.14159265358979323846264338327 * r;
 float a = nt - nc;
 float b = nt + nc;
 float Tr = 1.0f - (R0 + (1.0f - R0) * phi);
 float R = (D * nnt - N * (a * Tr + b * (1.0f - Tr)));
 float E = diffuse;
 bool = true;
 if (refl + refr) && (depth < MAXDEPTH)
 {
 float D, N;
 float refl * E * diffuse;
 bool = true;
 }
 if (MAXDEPTH)
 {
 float survive = SurvivalProbability(diffuse);
 float estimation = doing it properly, closely follow;
 if (survive)
 {
 float radiance = SampleLight(&rand, 1, &t, &phi);
 float x = radiance.x + radiance.y + radiance.z;
 if (x > 0) &&
 {
 bool = true;
 float brdfPdf = EvaluateDiffuse(L, N) *
 float3 factor = diffuse * INVPI;
 float weight = Mis2(directPdf, brdfPdf);
 float cosThetaOut = dot(N, L);
 float E = ((weight * cosThetaOut) / directPdf);
 }
 }
 // random walk - done properly, closely follow survive
 }
 float3 brdf = SampleDiffuse(diffuse, N, r1, r2, R0, R1, R2);
 bool survive;
 float pdf;
 float n = E * brdf * (dot(N, R) / pdf);
 bool = true;
 }

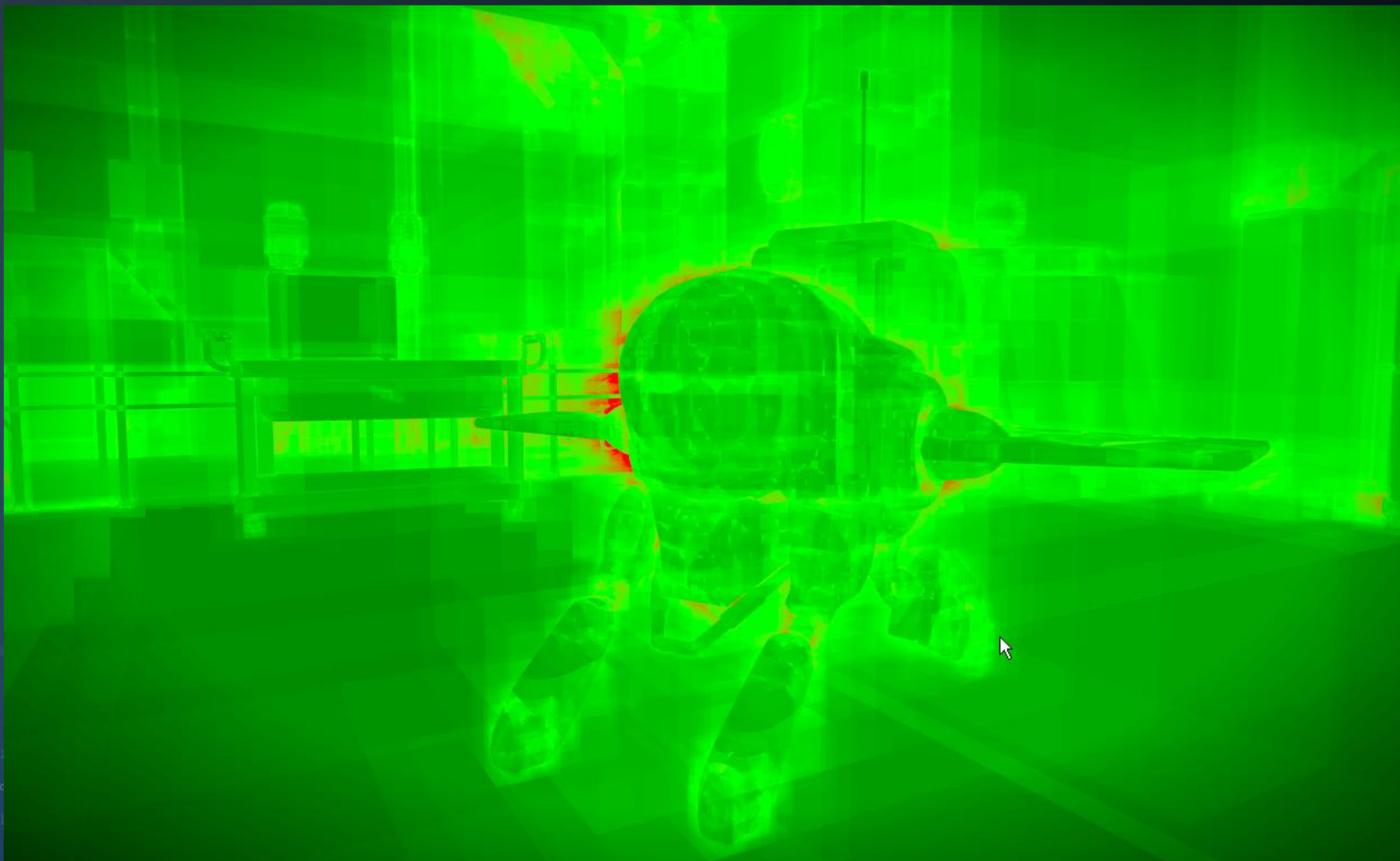
```





# Better BVHs

```
...
 & (depth < MAXDEPTH);
...
 t = inside / (1 + 0.5f * inside);
 nt = nt / nc; nct = nc / nt;
 r2t = 1.0f - nnt * t; r2b = nnt * t;
 D, N);
...
 at a = nt - nc, b = nt - nc;
 at Tr = 1 - (R0 + (1 - R0) * t);
 Tr) R = (D * nnt - N * (a * r2t + b * r2b));
...
 E * diffuse;
 = true;
...
 refl + refr)) && (depth < MAXDEPTH);
...
 D, N);
 refl * E * diffuse;
 = true;
...
MAXDEPTH)
survive = SurvivalProbability(diffuse,
estimation - doing it properly, close
if;
radiance = SampleLight(&rand, I, &L,
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) / directPe
random walk - done properly, closely fo
survive)
;
at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf);
survive;
pdf;
n = E * brdf * (dot(N, R) / pdf);
sion = true;
```



# Today's Agenda:

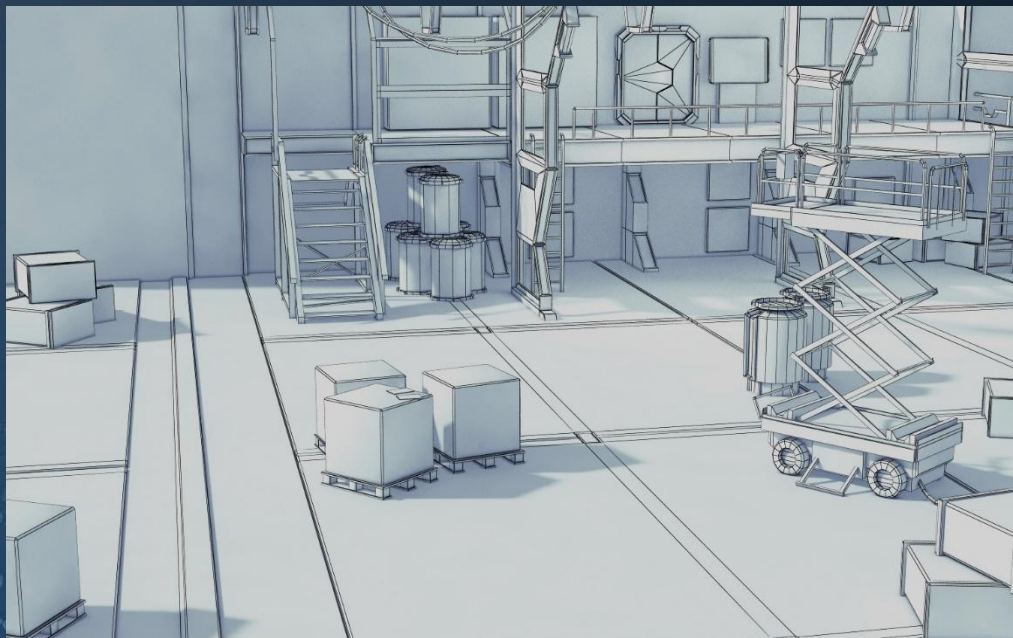
- Building Better BVHs
- The Problem of Large Polygons
- Refitting
- Fast BVH Construction



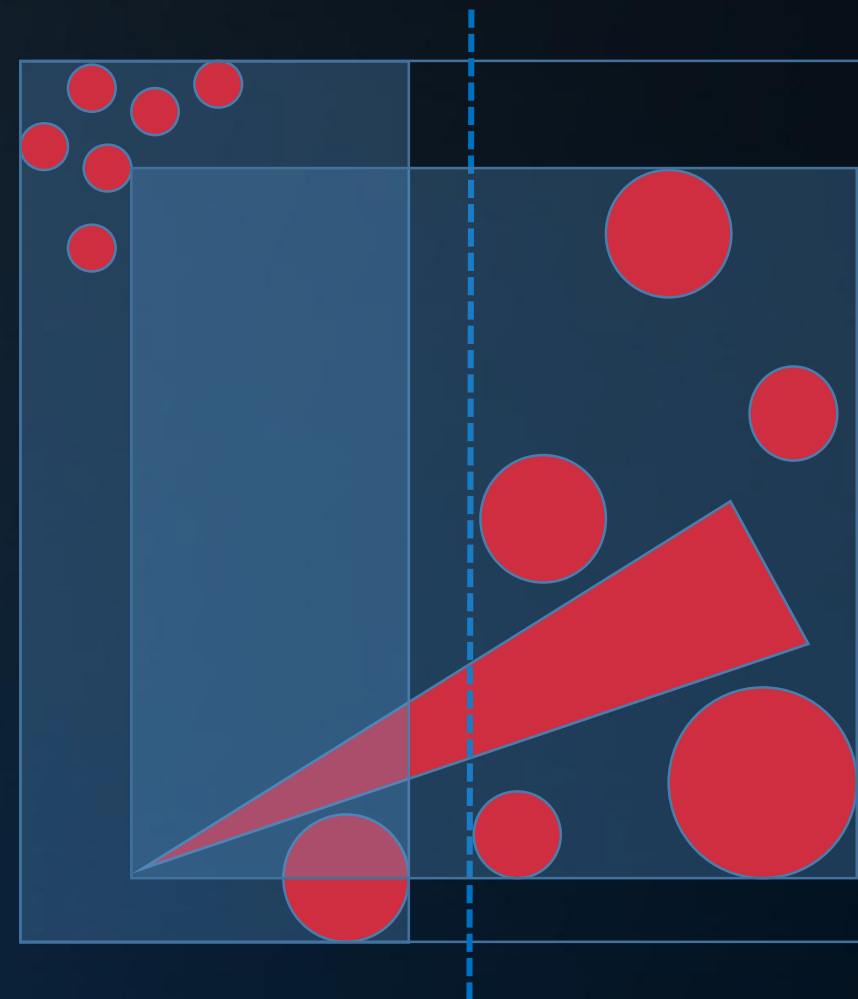
# Splitting

## Problematic Large Polygons

Large polygons lead to poor BVHs.



(far more common than you'd think)



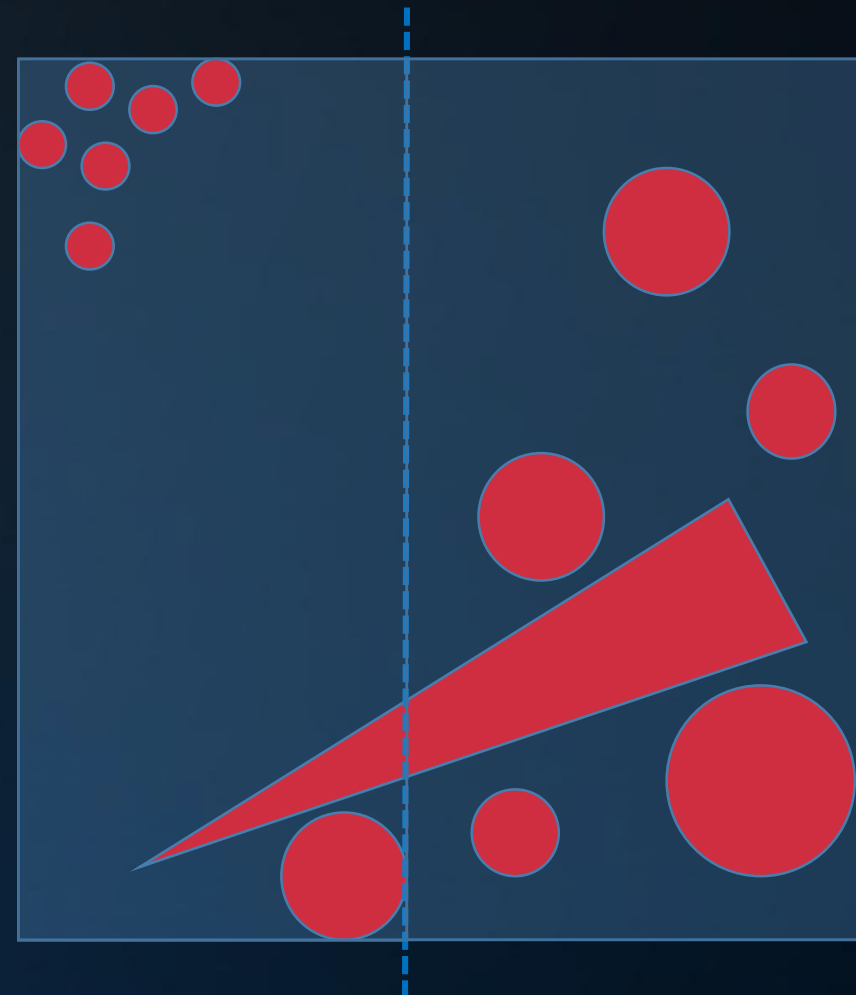
# Splitting

## Problematic Large Polygons

Large polygons lead to poor BVHs.

Using the spatial splits in kD-trees, this is far less of an issue:

The triangle will simply be assigned to each subspace.





# Splitting

## Problematic Large Polygons

Large polygons lead to poor BVHs.

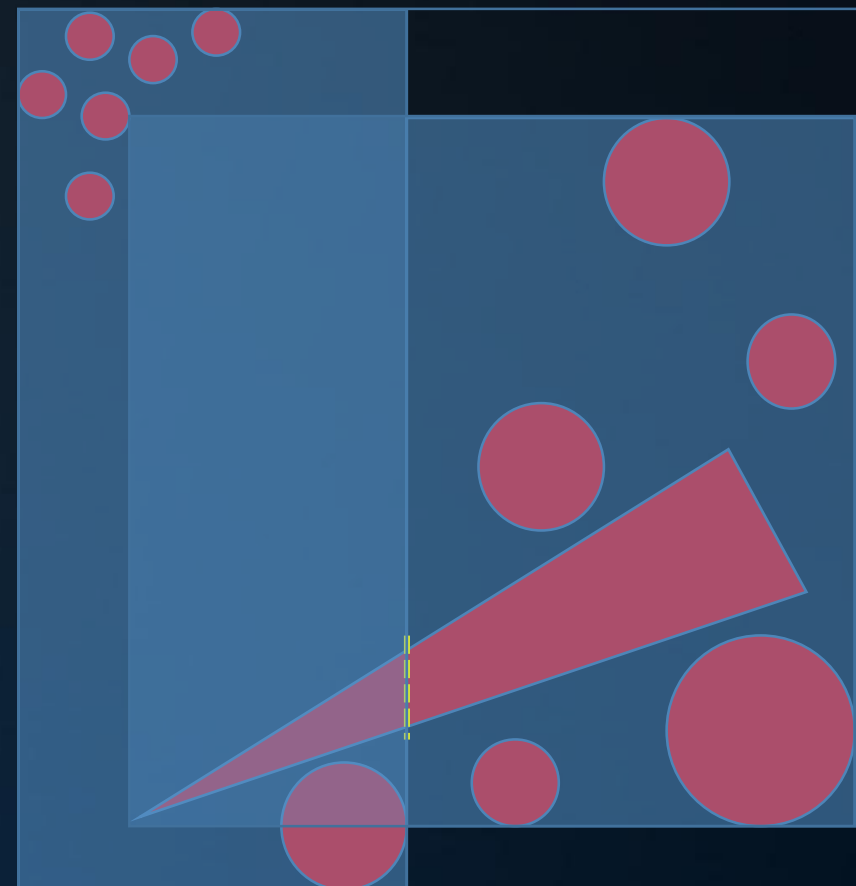
Using the spatial splits in kD-trees, this is far less of an issue:

The triangle will simply be assigned to each subspace.

Solution 1: split large polygons\*.

Observations:

1. A polygon can safely reside in multiple leafs;
2. The bounds of a leaf do not have to include the entire polygon.



\*: Early Split Clipping for Bounding Volume Hierarchies, Ernst & Greiner, 2007





# Splitting

## Early Split Clipping

### Observations:

1. A polygon can safely reside in multiple leafs;
2. The bounds of a leaf do not have to include the entire polygon.
3. BVH construction only uses primitive bounding boxes.

### Algorithm:

Prior to BVH construction, we recursively subdivide any polygon with a surface area that exceeds a certain threshold.

### Issues:

- Threshold parameter
- Individual polygons are split, regardless of surrounding geometry
- Primitives may end up multiple times in the same leaf

(some of these issues are resolved in: The Edge Volume Heuristic - Robust Triangle Subdivision for Improved BVH Performance, Dammertz & Keller, 2008)



# Splitting

## Spatial Splits for BVHs

$$C_{split} = A_{left} * N_{left} + A_{right} * N_{right} < A * N$$

Observation: spatial splits are not limited to kD-trees.

But: spatial splits tend to increase the cost of a split.

Idea:

1. Determine cost of optimal object partition;
2. Determine cost of optimal spatial split;
3. Apply spatial split if cost is lower than object partition\*.

\*: Spatial Splits in Bounding Volume Hierarchies, Stich et al., 2009



## State of the Art: SBVH

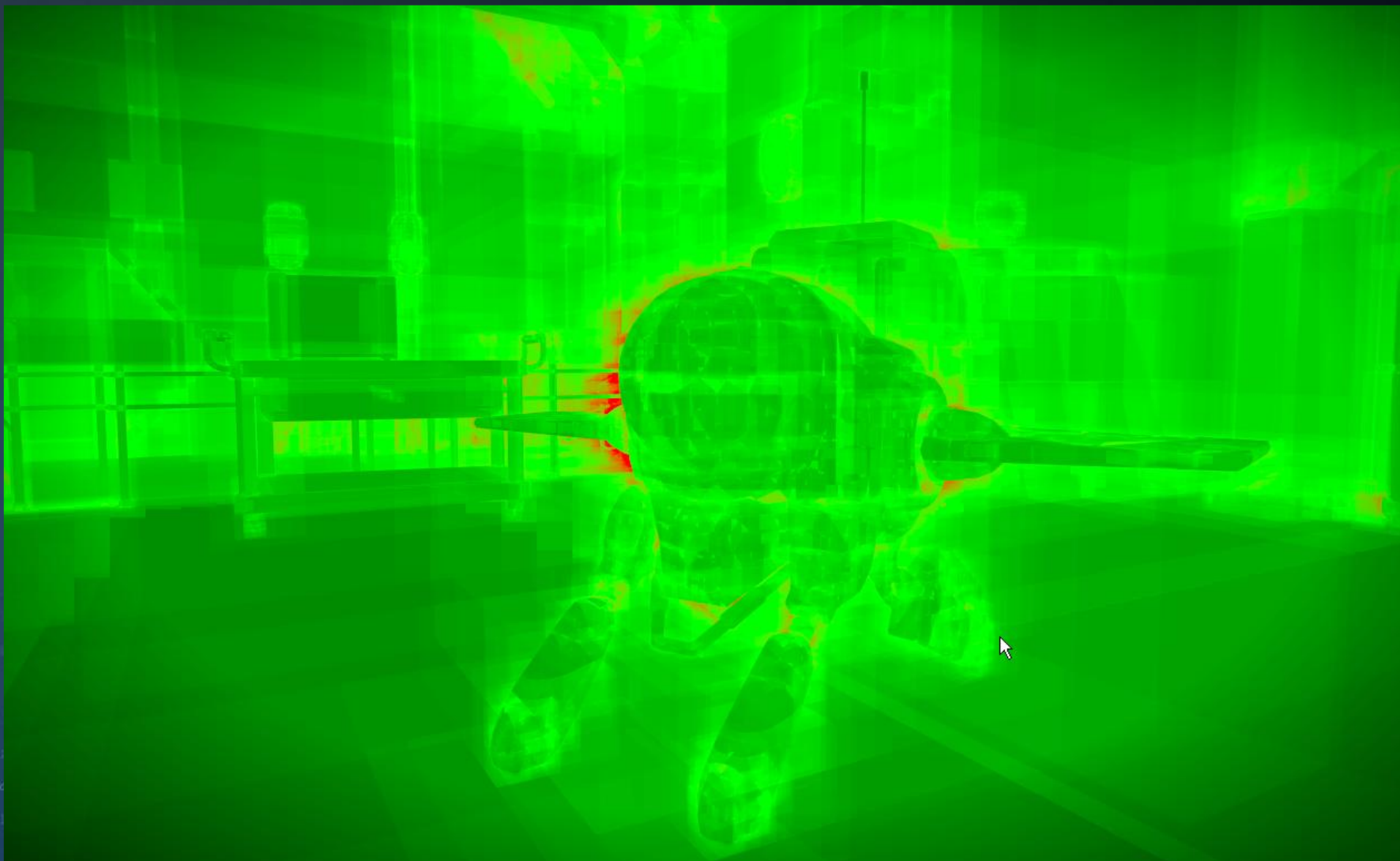
Compared to a regular SAH BVH, spatial splits improve the BVH by  $\sim 25\%$  (see paper for scenes and figures).



# Better BVHs

```
...
 & (depth < MAXDEPTH)
{
 t = inside / (1 + 0.5f * inside);
 nt = nt / nc; nct = nc / nt;
 r2t = 1.0f - nnt * nct;
 D, N);
}

// ...
// at a = nt - nc, b = nt * nc;
// at Tr = 1 - (RB + (1 - RB) * t);
// Tr) R = (D * nnt - N * (1 - nnt));
// ...
// E * diffuse;
// = true;
// ...
// refl + refr)) && (depth < MAXDEPTH)
// ...
// D, N);
// refl * E * diffuse;
// = true;
// ...
// MAXDEPTH)
// ...
// survive = SurvivalProbability(diffuse
// estimation - doing it properly, close
// if;
// radiance = SampleLight(&rand, 1, <
// e.x + radiance.y + radiance.z) > 0) &&
// ...
// w = true;
// at brdfPdf = EvaluateDiffuse(L, N) *
// at3 factor = diffuse * INVPI;
// at weight = Mis2(directPdf, brdfPdf);
// at cosThetaOut = dot(N, L);
// E * ((weight * cosThetaOut) / directPe
// ...
// random walk - done properly, closely fo
// survive)
// ...
// at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf);
// survive;
// pdf;
// n = E * brdf * (dot(N, R) / pdf);
// sion = true;
// ...
```





# Better BVHs

```

 if (depth < MAXDEPTH)
 {
 float t = inside / (1 - nc);
 float nt = nt / nc; dde = dde * nt;
 float r2t = 1.0f - nnt; r2b = nnt;
 float D, N);
 float a = nt - nc; b = nt * nc;
 float Tr = 1 - (R0 + (1 - R0) * r2t);
 float R = (D * nnt - N * (a * r2t + b * r2b));
 float E * diffuse;
 bool = true;
 float refl + refr)) && (depth < MAXDEPTH))
 float D, N);
 float refl * E * diffuse;
 bool = true;
 MAXDEPTH)
 survive = SurvivalProbability(diffuse
 estimation - doing it properly, closely
 if;
 radiance = SampleLight(&rand, I, &L,
 e.x + radiance.y + radiance.z) > 0) &&
 w = true;
 float brdfPdf = EvaluateDiffuse(L, N) *
 float3 factor = diffuse * INVPI;
 float weight = Mis2(directPdf, brdfPdf);
 float cosThetaOut = dot(N, L);
 float E * ((weight * cosThetaOut) / directPdf);
 random walk - done properly, closely for
 survive)
 float3 brdf = SampleDiffuse(diffuse, N, r1, r2, R0, &rand);
 survive;
 pdf;
 float n = E * brdf * (dot(N, R) / pdf);
 bool = true;

```





# Today's Agenda:

- Building Better BVHs
- The Problem of Large Polygons
- Refitting
- Fast BVH Construction



# Refitting

## Summary of BVH Characteristics

A BVH provides significant freedom compared to e.g. a kD-tree:

- No need for a 1-to-1 relation between bounding boxes and primitives
- Primitives may reside in multiple leafs
- Bounding boxes may overlap
- Bounding boxes can be altered, as long as they fit in their parent box
- A BVH can be very bad but still valid

Some consequences / opportunities:

- We can rebuild part of a BVH
- We can combine two BVHs into one
- We can refit a BVH



# Refitting

## Refitting

Q: What happens to the BVH of a tree model, if we make it bend in the wind?

A: Likely, only bounds will change; the topology of the BVH will be the same (or at least similar) in each frame.

Refitting:

*Updating the bounding boxes stored in a BVH to match changed primitive coordinates.*



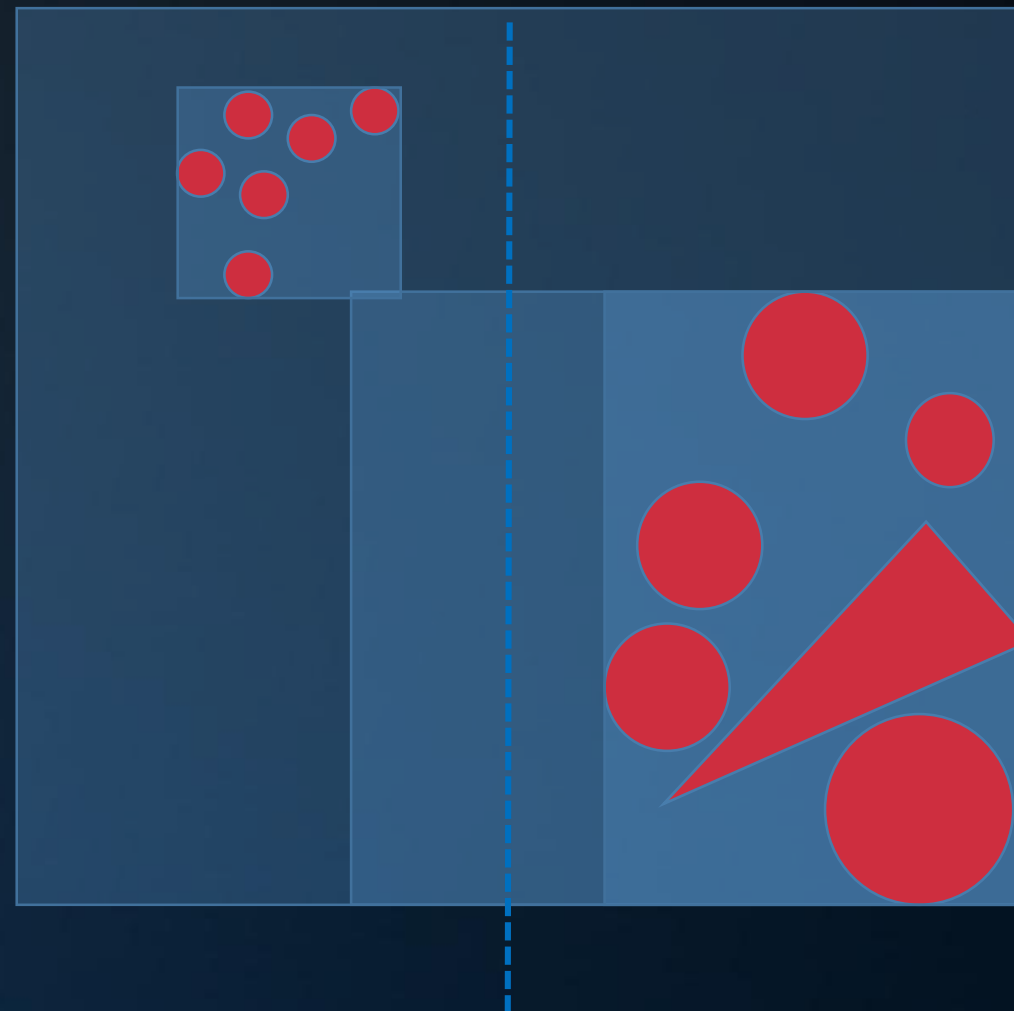
# Refitting

## Refitting

*Updating the bounding boxes stored in a BVH to match changed primitive coordinates.*

### Algorithm:

1. For each leaf, calculate the bounds over the primitives it represents
2. Update parent bounds



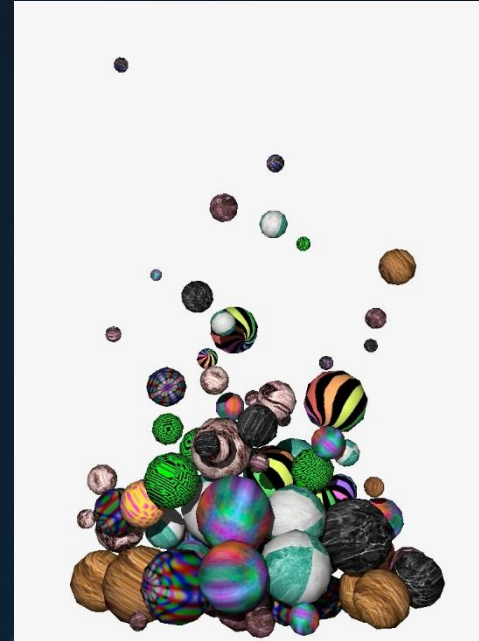
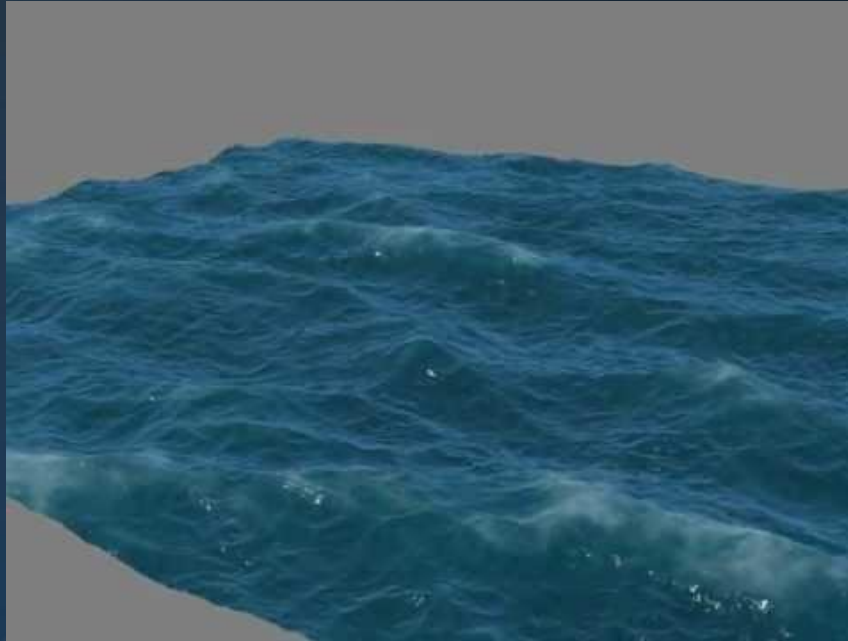


# Refitting

## Refitting - Suitability

```
ices
& (depth < MAXD
```

```
at a = nt
at nt = nt
os2t = 1
D, N);
)
at a = nt
at Tr = 3
Tr) R = (E
E * diffu
= true;
efl + ref
D, N);
refl * E
= true;
MAXDEPTH)
survive =
estimat
df;
radiance
e.x + rad
v = true;
at brdfPdf = EvaluateDiffuse(L, N) * Pdf
at3 factor = diffuse * INVPI;
at weight = Mis2(directPdf, brdfPdf);
at cosThetaOut = dot(N, L);
E * ((weight * cosThetaOut) / directPdf) * (radiance
random walk - done properly, closely following the
ive)
at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf);
survive;
pdf;
n = E * brdf * (dot(N, R) / pdf);
sion = true;
```





# Refitting

## Refitting – Practical

```

"ice
& (depth < MAXDEPTH) {
 // Inside / Outside
 nt = nt / nc; pdd = pdd / nc;
 pos2t = 1.0f - nnt * pdd;
 D, N);
)
 at a = nt - nc, b = nt - nc;
 at Tr = 1 - (RB + (1 - RB) * pdd);
 Tr) R = (D * nnt - N * pdd);
 E * diffuse;
 = true;
 -
 efl + refr)) && (depth < MAXDEPTH) {
 D, N);
 -refl * E * diffuse;
 = true;
 MAXDEPTH)
 survive = SurvivalProbability(diffuse, 1);
 estimation - doing it properly, close to
 if;
 radiance = SampleLight(&rand, I, &t, &light);
 e.x + radiance.y + radiance.z) > 0) && (depth <
 v = true;
 at brdfPdf = EvaluateDiffuse(L, N) * Psurvive;
 at3 factor = diffuse * INVPI;
 at weight = Mis2(directPdf, brdfPdf);
 at cosThetaOut = dot(N, L);
 E * ((weight * cosThetaOut) / directPdf) * (radiance
 random walk - done properly, closely following Monte Carlo
 vive)
 ;
 at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf);
 survive;
 pdf;
 n = E * brdf * (dot(N, R) / pdf);
 sion = true;

```



BVH node array

Level 1

Root node

Order of nodes in the node array:

*We will never find the parent of node  $X$  at a position greater than  $X$ .*

Therefore:

```

for(int i = N-1; i >= 0; i--)
 nodeArray[i].AdjustBounds();

```



# Today's Agenda:

- Building Better BVHs
- The Problem of Large Polygons
- Refitting
- Fast BVH Construction



# Binning

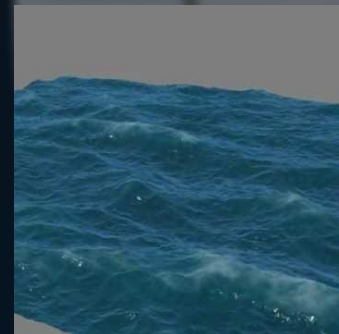
## Rapid BVH Construction

Refitting allows us to update hundreds of thousands of primitives in real-time. But what if topology changes significantly?

Rebuilding a BVH requires  $3N \log N$  split plane evaluations.

## Options:

1. Do not use SAH (significantly lower quality BVH)
2. Do not evaluate all 3 axes (minor degradation of BVH quality)
3. Make split plane selection independent of  $N$



# Binning

```

-ics
& (depth < MAXDEPTH)
{
 t = inside / 1.5;
 nt = nt / nc; dde = dde / nc;
 pos2t = 1.0f - nnt * t;
 D, N);
}

at a = nt - nc, b = nt - nc;
at Tr = 1 - (RB + (1 - RB) * t);
Tr) R = (D * nnt - N * (dd

E * diffuse;
= true;

efl + refr)) && (depth < MAXDEPTH)
{
 D, N);
 refl * E * diffuse;
 = true;

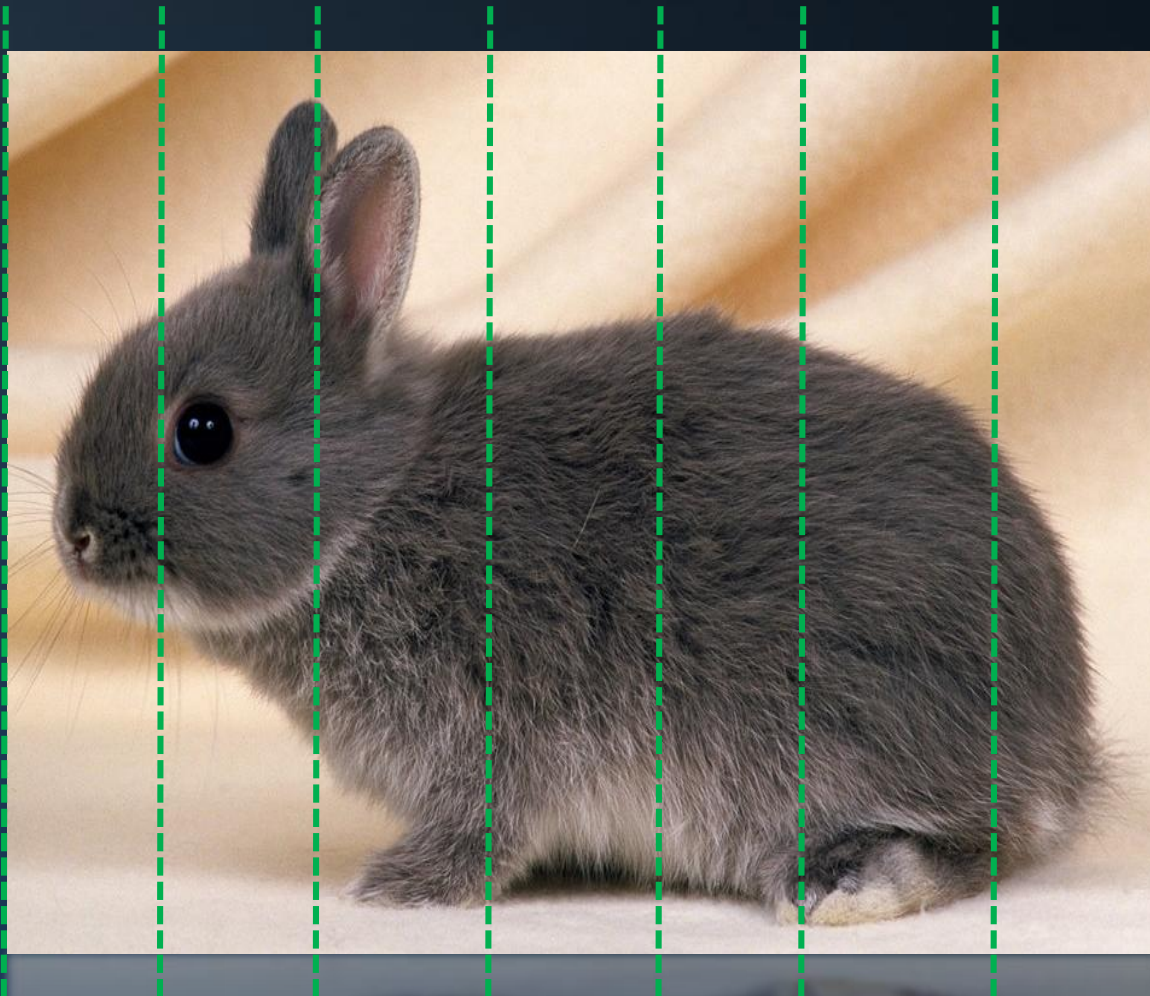
MAXDEPTH)

survive = SurvivalProbability(diffuse, L);
estimation - doing it properly, close
if;
radiance = SampleLight(&rand, I, &L, &light;
e.x + radiance.y + radiance.z) > 0) && (depth <
w = true;
at brdfPdf = EvaluateDiffuse(L, N) * Psurvive;
at3 factor = diffuse * INVPI;
at weight = Mis2(directPdf, brdfPdf);
at cosThetaOut = dot(N, L);
E * ((weight * cosThetaOut) / directPdf) * (radiance

random walk - done properly, closely following real
ive)

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

```





## Binned BVH Construction\*

*Evaluate SAH at  $N$  discrete intervals.*



\*: On fast Construction of SAH-based Bounding Volume Hierarchies, Wald, 2007

## Binned BVH Construction

1. Calculate spatial bounds
2. Calculate object centroid bounds
3. Calculate intervals (efficiently and accurately!)
4. Populate bins
5. Sweep: evaluate cost, keep track of counts
6. Use best position







# Binning





# INFOMAGR – Advanced Graphics

Jacco Bikker - February – April 2016

## END of “The Perfect BVH”

next lecture: “Real-time Ray Tracing”

