

```
ics
& (depth < MAXDEPTH)
{
    if (inside ? 1 : 0.25)
    {
        nt = nt / nc; ddn = ddn * ddn;
        pos2t = 1.0f - nnt * nnt;
        D, N );
    }
    at a = nt - nc, b = nt + nc;
    at Tr = 1 - (R0 + (1 - R0) * pos2t);
    (Tr) R = (D * nnt - N * (ddn * pos2t));
    E * diffuse;
    = true;
    efl + refr)) && (depth < MAXDEPTH)
    D, N );
    refl * E * diffuse;
    = true;
    MAXDEPTH)
    survive = SurvivalProbability( diffuse );
    estimation - doing it properly, closely following
    if;
    radiance = SampleLight( &rand, I, &align,
    e.x + radiance.y + radiance.z );
    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 Small's
    (survive)
    ;
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    sion = true;
```

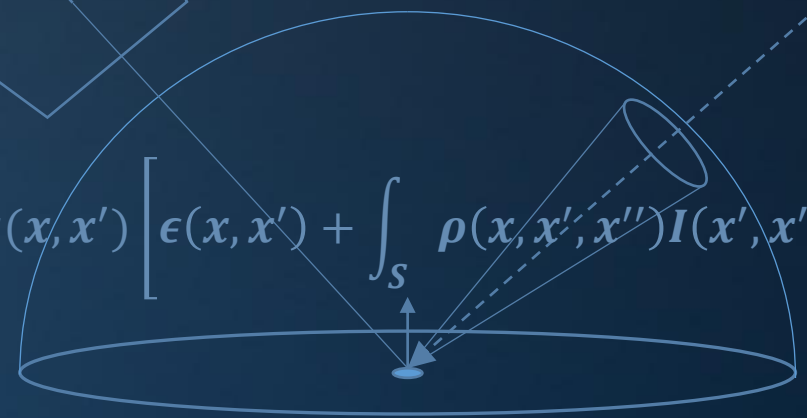


INFOMAGR – Advanced Graphics

Jacco Bikker - November 2022 - February 2023

Lecture 6 - “The Perfect BVH”

Welcome!



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



```

ics
& (depth < MAXDEPTH) {
    // Inside?
    if (inside ? 1 : 0) {
        nt = nt / nc; ddn = ddn * nc;
        cos2t = 1.0f - nnt * ddn;
        D, N );
    }
    // Outside?
    at a = nt - nc, b = nt + nc;
    at Tr = 1 - (R0 + (1 - R0) * ddn);
    (Tr) R = (D * nnt - N * (ddn > 0 ? 1 : -1));
    // Diffuse
    E * diffuse;
    = true;
    // Refractive
    refl + refr)) && (depth < MAXDEPTH) {
        D, N );
        refl * E * diffuse;
        = true;
    }
    // Survival
    MAXDEPTH)
    survive = SurvivalProbability( diffuse );
    // Estimation
    estimation - doing it properly, closely following
    if;
    radiance = SampleLight( &rand, I, &L, &lightPdf );
    e.x + radiance.y + radiance.z) > 0) && (depth < MAXDEPTH) {
        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.x + radiance.y + radiance.z);
    }
    // Random walk
    random walk - done properly, closely following Small's
    (survive)
    ;
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    ion = true;
}

```

Today's Agenda:

- Building Better BVHs
- Refitting
- Fast BVH Construction
- The Top-level BVH

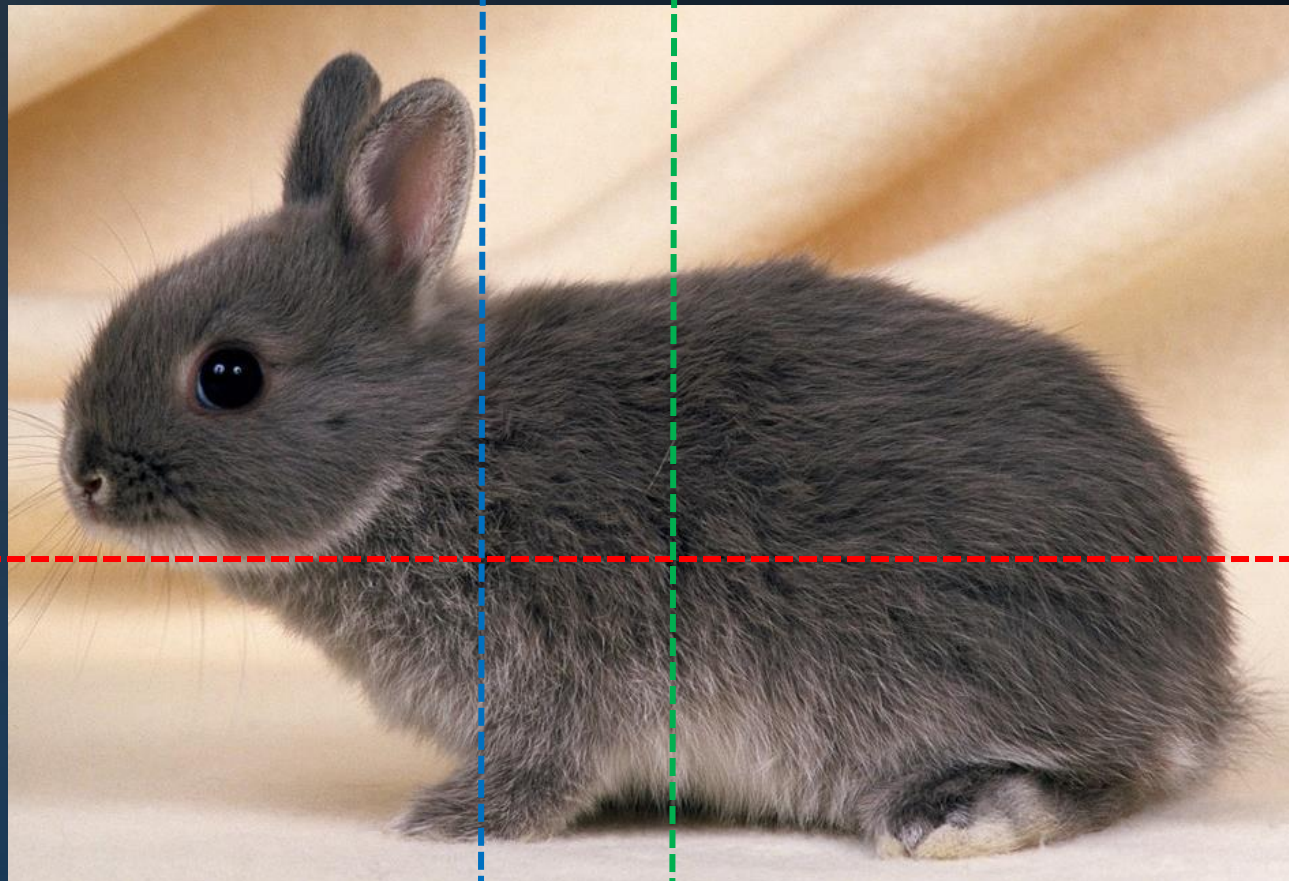


Better BVHs

```

ics
& (depth < MAXDEPTH)
{
    if (nt < nc) {
        nt = nt / nc; ddn = ddn / nc;
        pos2t = 1.0f - nnt * ddn;
        D, N );
    }
    else {
        at a = nt - nc, b = nt + nc;
        at Tr = 1 - (R0 + (1 - R0) * a);
        Tr) R = (D * nnt - N * (ddn * a));
    }
    E * diffuse;
    = true;
    refl + refr)) && (depth < MAXDEPTH)
    D, N );
    refl * E * diffuse;
    = true;
    MAXDEPTH)
    survive = SurvivalProbability( diffuse );
    estimation - doing it properly, closely following
    df;
    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 Small's
    vive)
    ;
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    sion = true;

```



Better BVHs

```

ics
& (depth < MAXDEPTH)
{
    if (inside ? 1 : 1.0f - 0.5f)
    {
        nt = nt / nc, ddn = ddn * ddn;
        cos2t = 1.0f - nnt * ddn;
        D, N );
    }
    {
        a = nt - nc, b = nt + nc;
        at Tr = 1 - (R0 + (1 - R0) * cos2t);
        Tr) R = (D * nnt - N * (ddn * cos2t));
    }
    E * diffuse;
    = true;
}
{
    refl + refr)) && (depth < MAXDEPTH)
    {
        D, N );
        refl * E * diffuse;
        = true;
    }
    MAXDEPTH)
    {
        survive = SurvivalProbability( diffuse );
        estimation - doing it properly, closely following Small's
        if;
        radiance = SampleLight( &rand, I, &L, &light, &N );
        e.x + radiance.y + radiance.z) > 0) && (depth < MAXDEPTH)
        {
            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 Small's
        vive)
    }
    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.

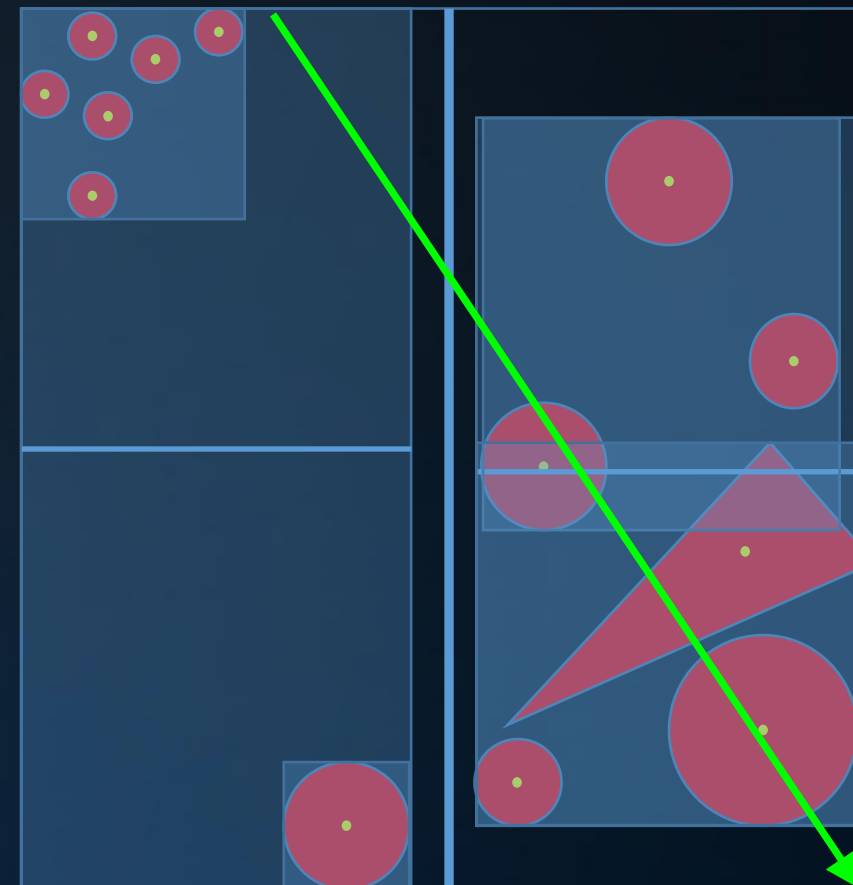
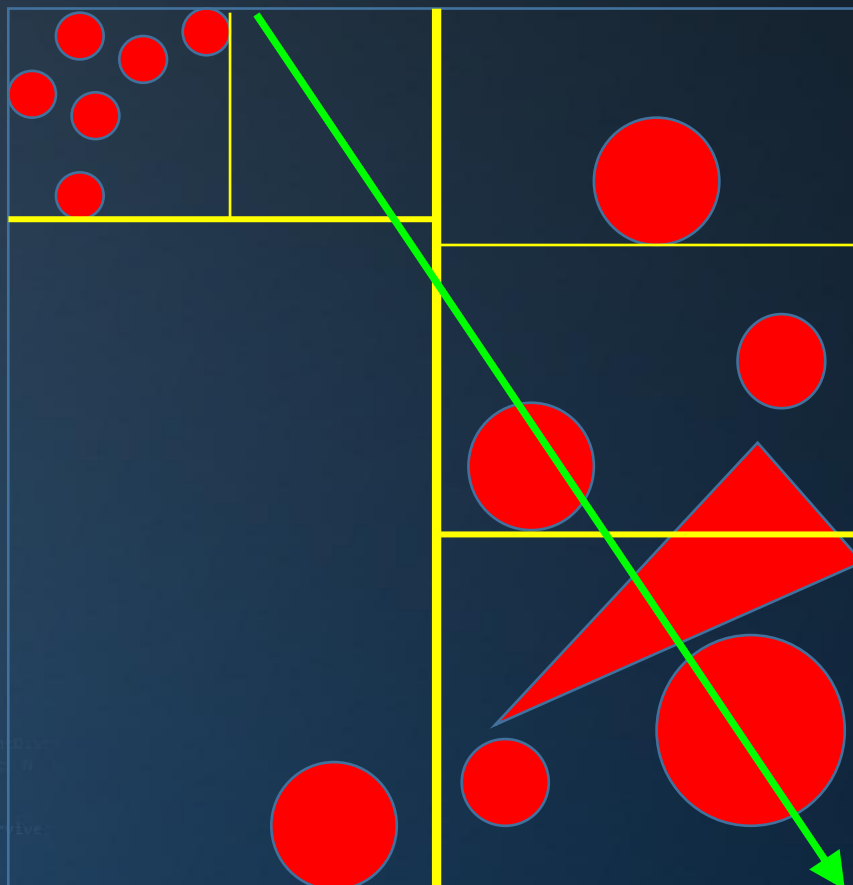


Better BVHs

```

ics
& (depth < MAXDEPTH)
{
    if (inside ? 1 : 0.25)
    {
        nt = nt / nc; ddn = ddn * 0.5;
        cos2t = 1.0f - nnt * ddn;
        D, N );
    }
    {
        at a = nt - nc, b = nt * nc;
        at Tr = 1 - (R0 + (1 - R0) * ddn);
        Tr) R = (D * nnt - N * (ddn * 0.5));
    }
    E * diffuse;
    = true;
    {
        refl + refr)) && (depth < MAXDEPTH)
    {
        D, N );
        refl * E * diffuse;
        = true;
    }
    MAXDEPTH)
    survive = SurvivalProbability( diffuse );
    estimation - doing it properly, closely
    if;
    radiance = SampleLight( &rand, I, &L, &align );
    e.x + radiance.y + radiance.z) > 0) && (depth < MAXDEPTH)
    {
        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 Small's
    (survive)
    {
        at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
        survive;
        pdf;
        n = E * brdf * (dot( N, R ) / pdf);
        ion = true;
    }
}

```



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



Route 1: 10% up-time, \$1000 fine



Route 2: 100% up-time, \$100 fine



Better BVHs

Optimal Split Plane Position

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

This expected cost is based on:

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

The cost of a split is thus:

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



```

    * diffuse;
    = true;

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

    MAXDEPTH)

survive = SurvivalProbability( diffuse
estimation - doing it properly, closely fol
df;
radianc = SampleLight( &rand, I, &L, &I
.e.x + radianc.y + radianc.z) > 0) && (
w = true;
at brdfPdf = EvaluateDiffuse( L, N ) * P
at3 factor = diffuse * INVPI;
at weight = Mis2( directPdf, brdfPdf );
at cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf)
random walk - done properly, closely follow
ive)

```

```

    = 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);
    random walk - done properly, closely follows
    (arrive)
;
    at3 brdf = SampleDiffuse( diffuse, N, r1,
    arrive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    sion = true;

```

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

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

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



# Better BVHs

## Optimal Split Plane Position

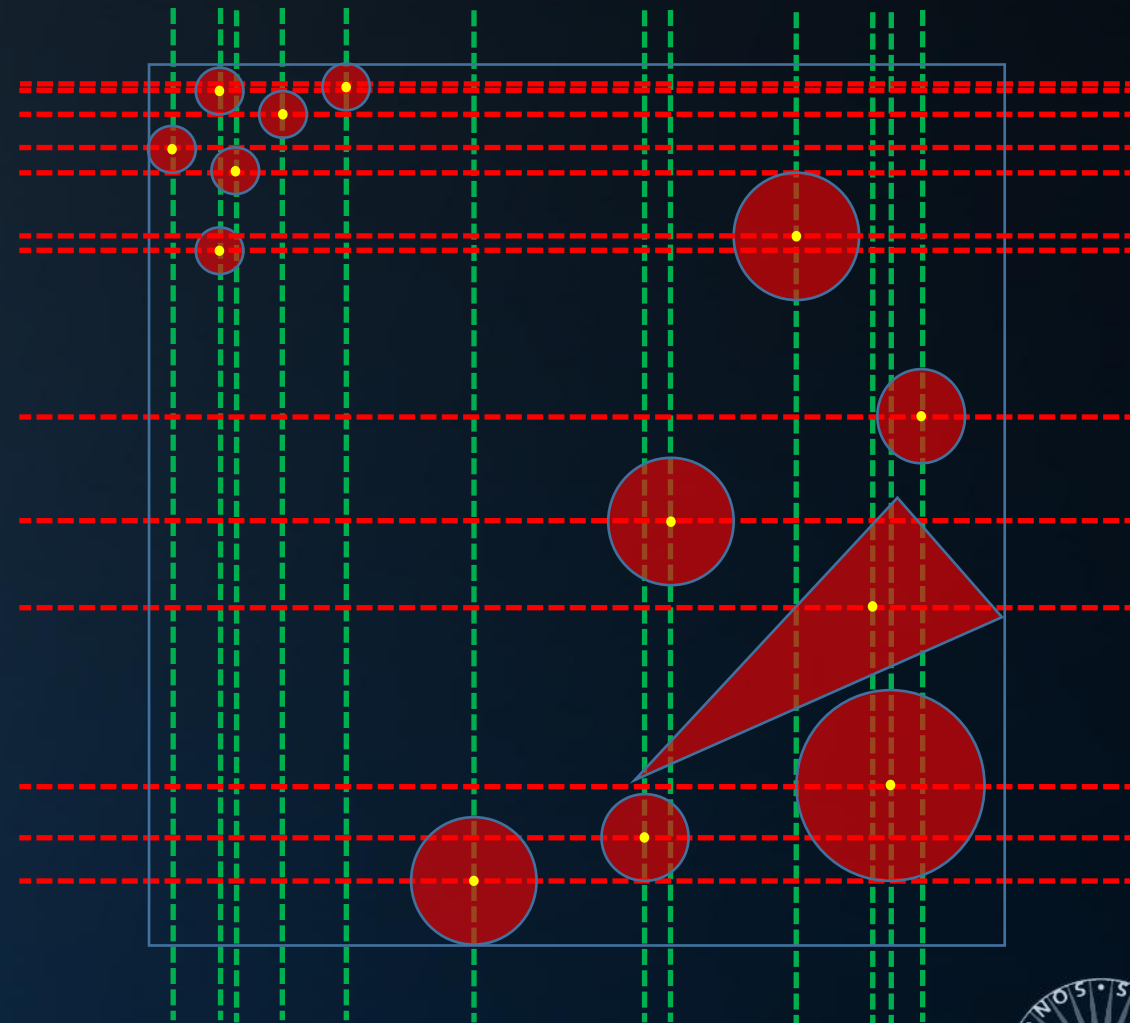
Which positions do we consider?

*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

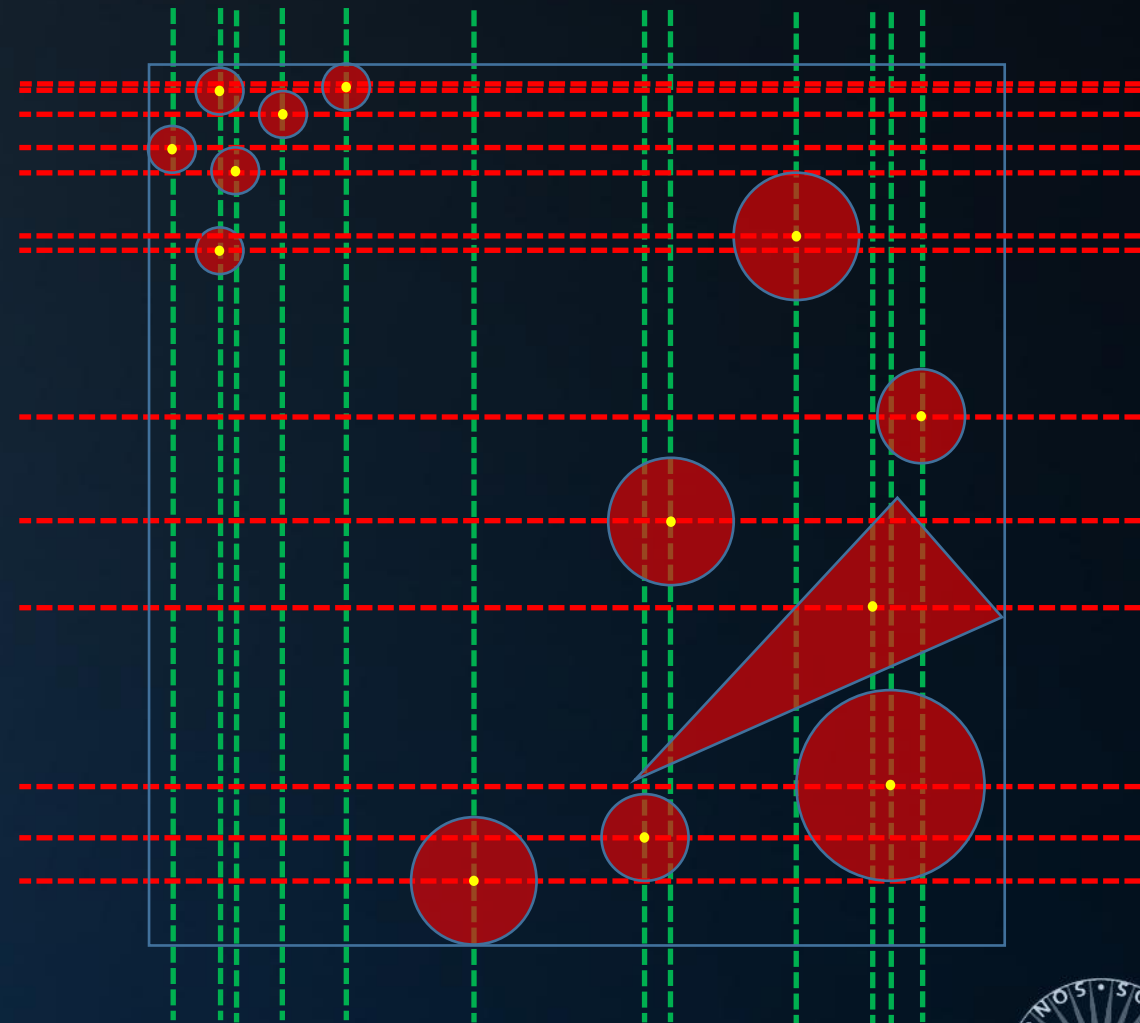
## SAH and Termination

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

## Optimal Split Plane Position

The *surface area heuristic* (SAH) is applied in a greedy manner\*.

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



# Better BVHs

## Optimal Split Plane Position

## Comparing naïve versus SAH:

- SAH will cut #intersections in half;
- expect ~2x better performance.

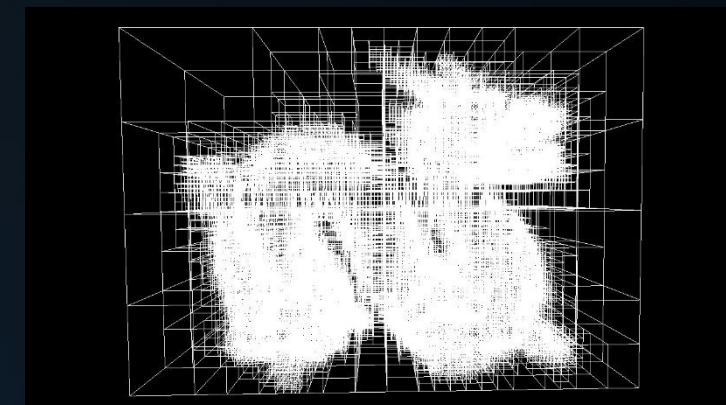
## SAH & kd-trees:

- Same scheme applies.

```

ics
& (depth < MAXDEPTH)
{
 // Inside?
 if (inside ? 1 : 0)
 {
 nt = nt / nc; ddn = ddn * nc;
 cos2t = 1.0f - nnt * nnt;
 D, N);
 }
 // Outside?
 {
 at a = nt - nc, b = nt + nc;
 at Tr = 1 - (R0 + (1 - R0) * a);
 Tr) R = (D * nnt - N * (ddn *
 }
 // Diffuse?
 E * diffuse;
 = true;
 // Refractive?
 {
 refl + refr)) && (depth < MAXDEPTH)
 {
 D, N);
 refl * E * diffuse;
 = true;
 }
 }
 // Survival?
 survive = SurvivalProbability(diffuse);
 // Estimation - doing it properly, closely following
 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 Small's
 (survive)
 {
 at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf);
 survive;
 pdf;
 n = E * brdf * (dot(N, R) / pdf);
 ion = true;
 }
}

```

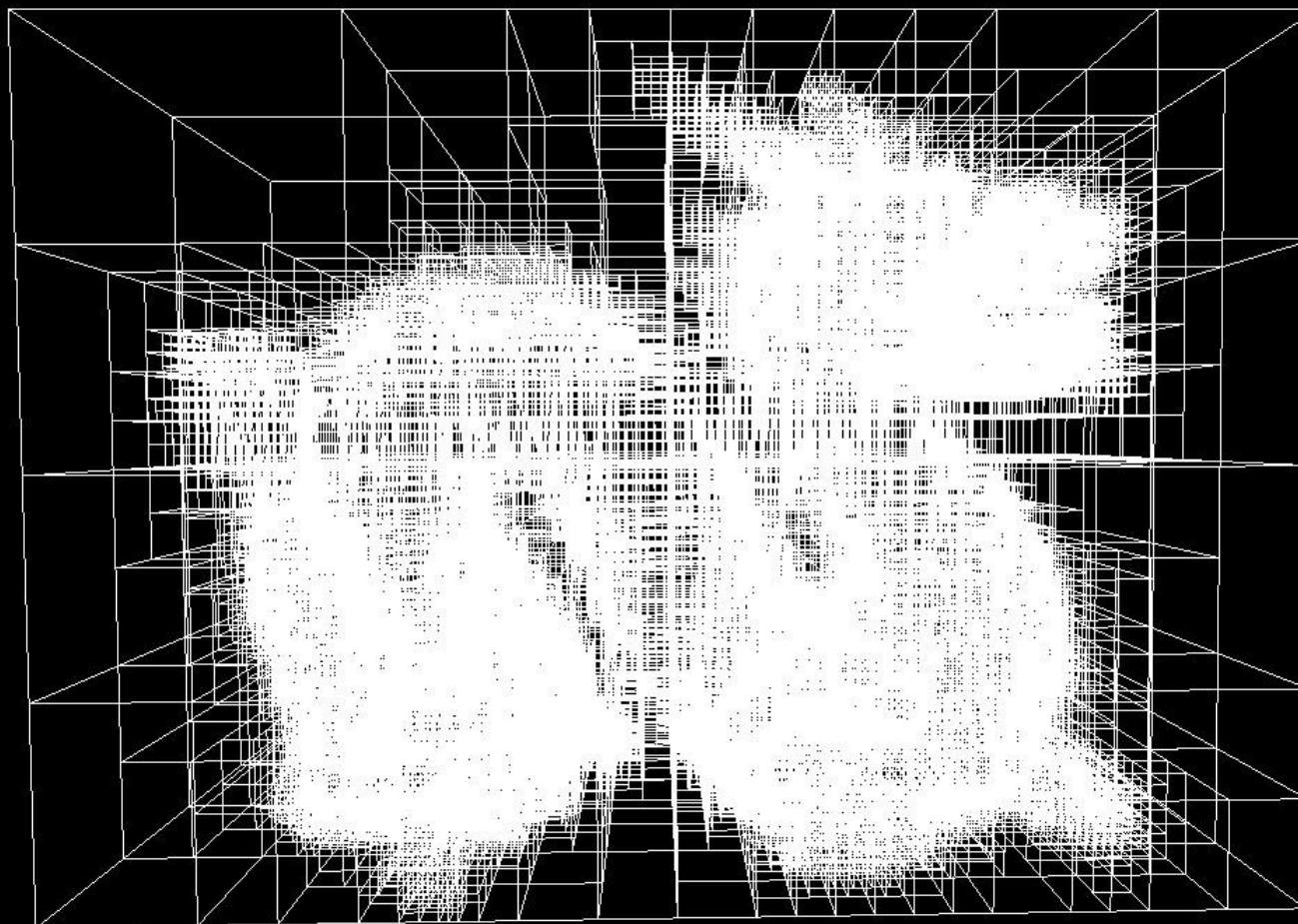


# Better BVHs

```

ics
& (depth < MAXDEPTH)
{
 if (inside ? 1 : 0)
 {
 nt = nt / nc; ddn = ddn * nc;
 cos2t = 1.0f - nnt * ddn;
 D, N);
 }
 {
 at a = nt - nc, b = nt + nc;
 at Tr = 1 - (R0 + (1 - R0) *
 Tr) R = (D * nnt - N * (ddn
 }
 {
 E * diffuse;
 = true;
 }
 {
 refl + refr)) && (depth < MAXDEPTH)
 {
 D, N);
 refl * E * diffuse;
 = true;
 }
 {
 MAXDEPTH)
 {
 survive = SurvivalProbability(diffuse
 estimation - doing it properly, close
 df;
 radiance = SampleLight(&rand, I, &L,
 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) / directPdf
 }
 {
 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;
 }

```



## Median Split





# Better BVHs

```

ics
& (depth < MAXDEPTH)
{
 if (inside ? 1 : 0)
 {
 nt = nt / nc; ddn = ddn * ddn;
 cos2t = 1.0f - nnt * ddn;
 D, N);
 }
 {
 at a = nt - nc, b = nt + nc;
 at Tr = 1 - (R0 + (1 - R0) * ddn);
 Tr) R = (D * nnt - N * ddn);
 }
 E * diffuse;
 = true;
 {
 defl + refr)) && (depth < MAXDEPTH)
 {
 D, N);
 refl * E * diffuse;
 = true;
 }
 MAXDEPTH)
 survive = SurvivalProbability(diffuse
 estimation - doing it properly, close
 df;
 radiance = SampleLight(&rand, I, &L,
 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

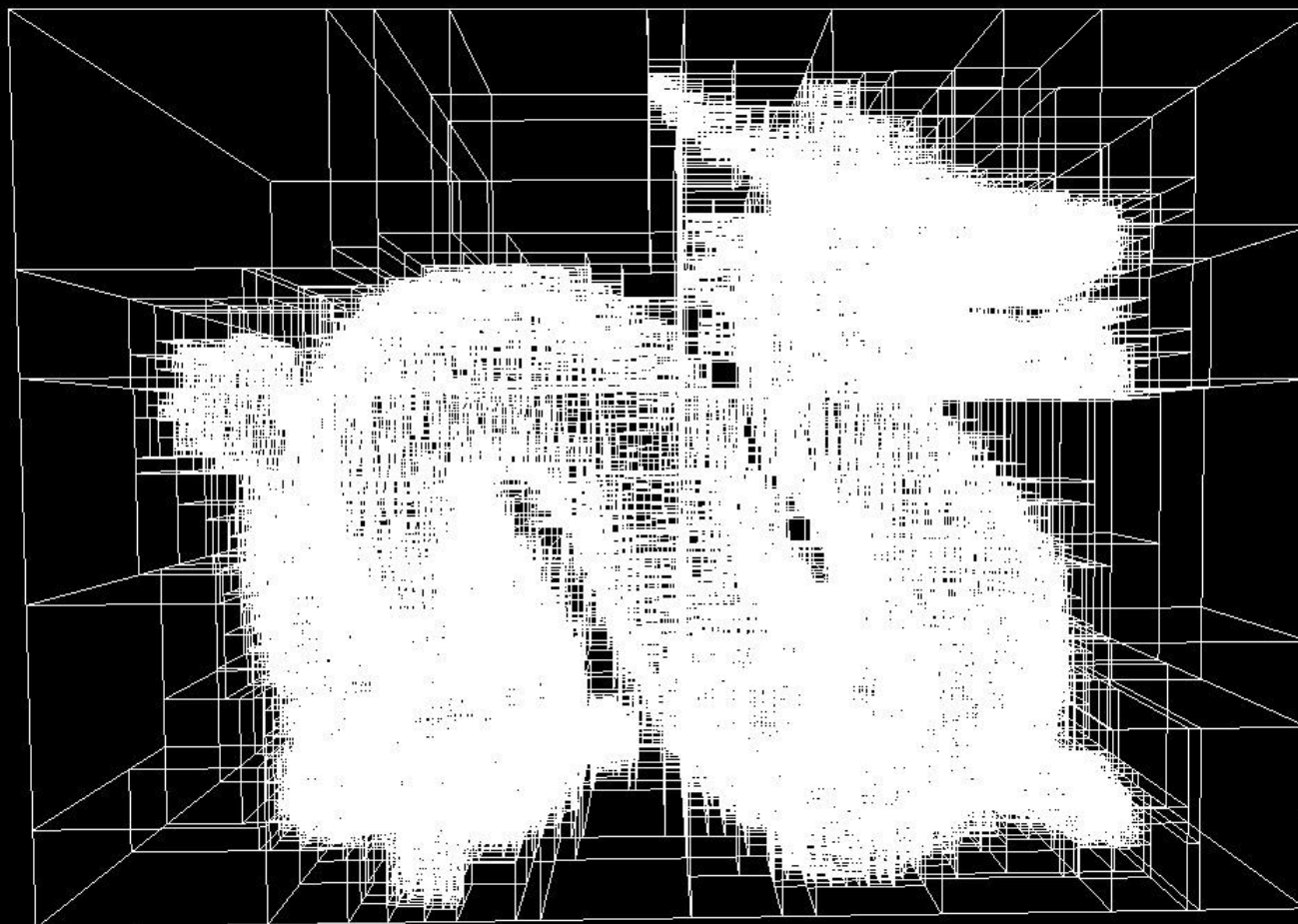
```

random walk - done properly, closely following the path  
 (survive)

```

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

```

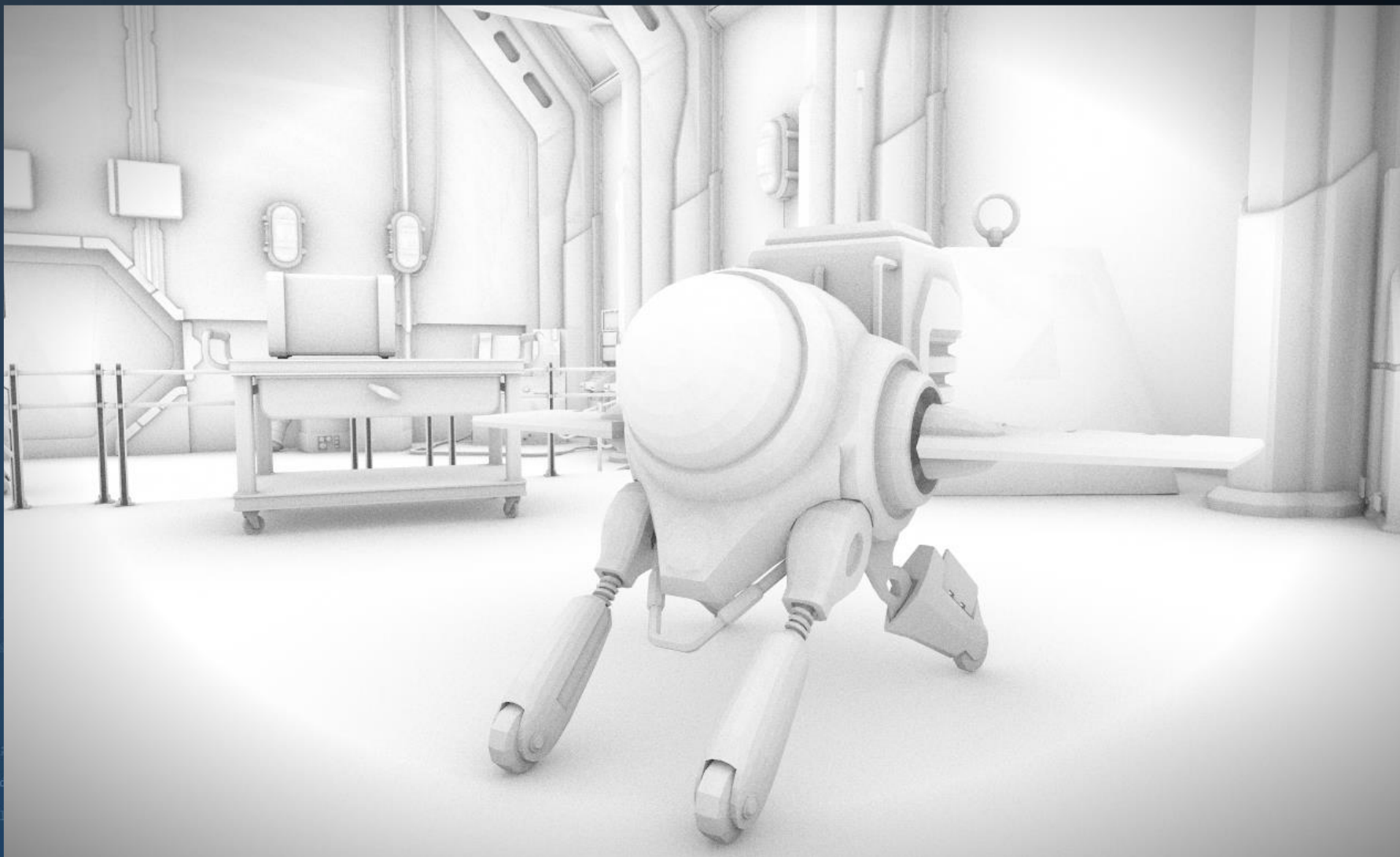


## Surface Area Heuristic



# Better BVHs

```
ics
& (depth < MAXDEPTH)
{
 if (nt < nc) {
 nt = inside ? 1.0f : 0.0f;
 nt = nt / nc; ddn = dot(N, D);
 cos2t = 1.0f - nnt * nnt;
 D = N * (ddn < 0 ? -1 : 1) * sqrt(cos2t);
 D, N);
 }
 else {
 at a = nt - nc, b = nt + nc;
 at Tr = 1 - (R0 + (1 - R0) * sqrt(a));
 Tr) R = (D * nnt - N * (ddn < 0 ? -1 : 1));
 }
 E * diffuse;
 = true;
 refl + refr)) && (depth < MAXDEPTH)
 D, N);
 refl * E * diffuse;
 = true;
 MAXDEPTH)
 survive = SurvivalProbability(diffuse
 estimation - doing it properly, close
 df;
 radiance = SampleLight(&rand, I, &L,
 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 fol
 vive)
 at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf
 survive;
 pdf;
 n = E * brdf * (dot(N, R) / pdf);
 sion = true;
```

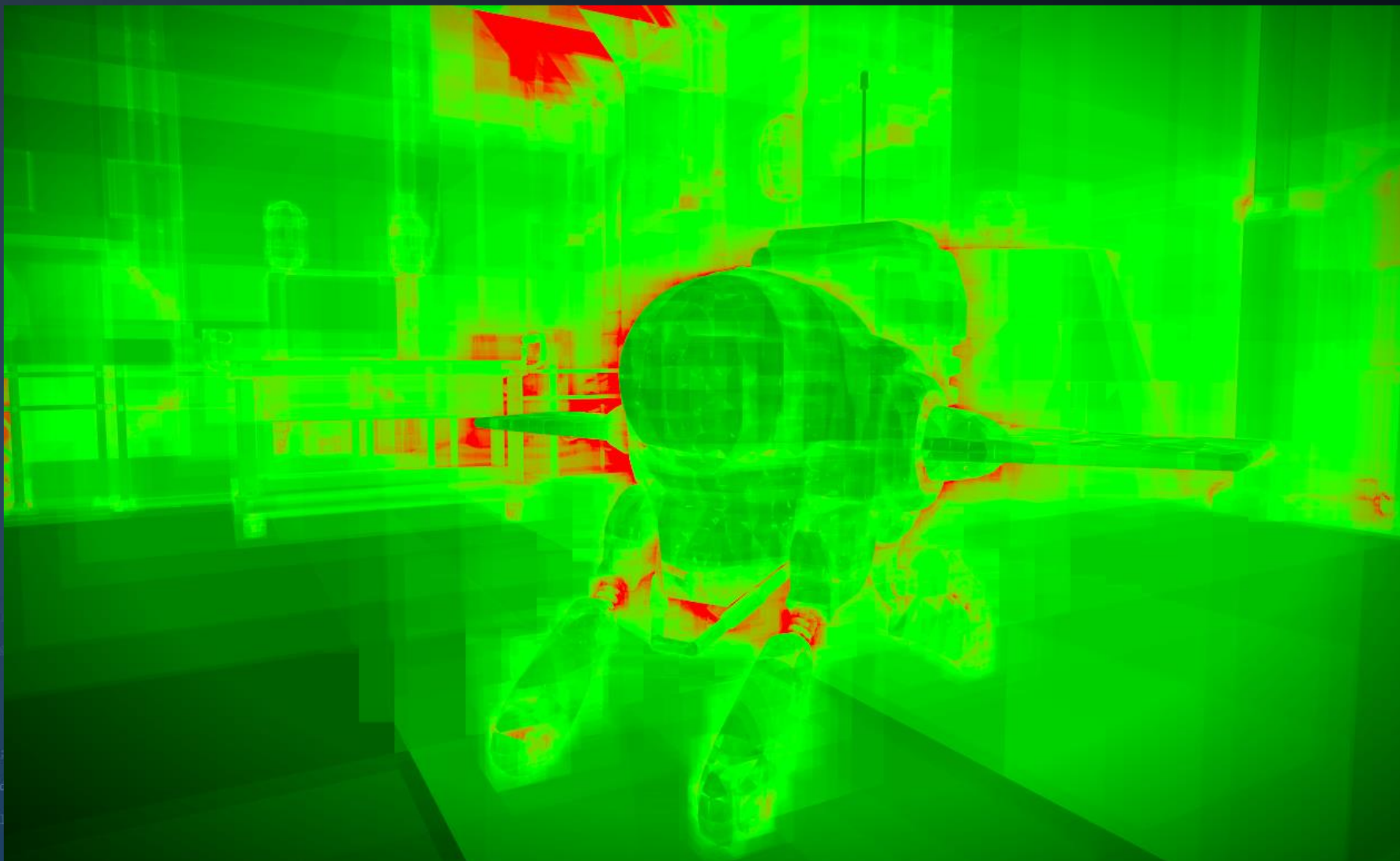


# Better BVHs

```

rics
& (depth < MAXDEPTH)
{
 if (nt < inside ? 1.0f : 0.5f)
 {
 nt = nt / nc; ddn = dot(N, L);
 cos2t = 1.0f - nnt * ddn;
 D, N);
 }
 else
 {
 at a = nt - nc, b = nt + nc;
 at Tr = 1 - (R0 + (1 - R0) * ddn);
 Tr) R = (D * nnt - N * (ddn > 0 ? 1 : -1));
 }
 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) &&
 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
 vive)
 ;
 at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf);
 survive;
 pdf;
 n = E * brdf * (dot(N, R) / pdf);
 sion = true;

```



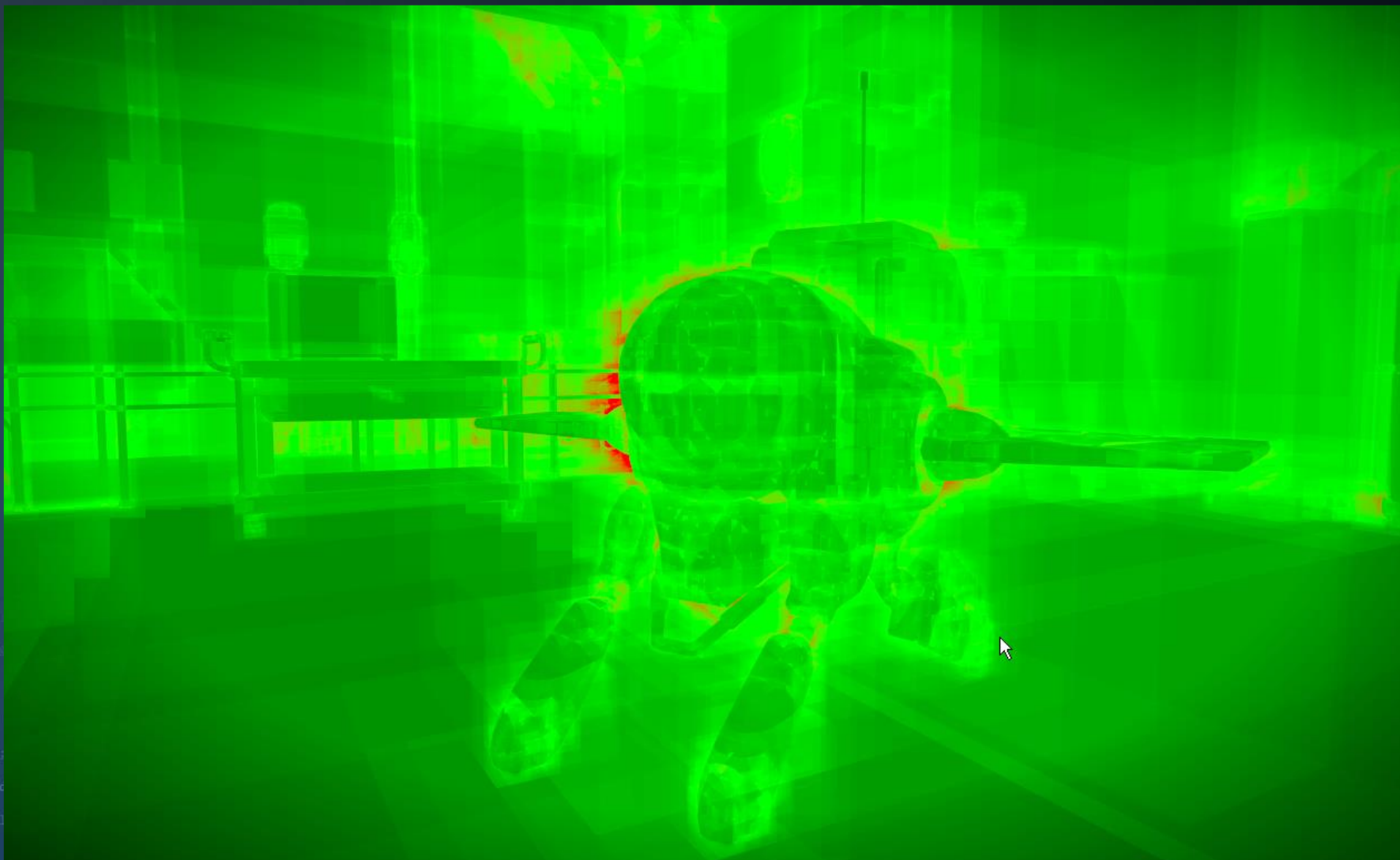


# Better BVHs

```

rics
& (depth < MAXDEPTH)
{
 if (nt < inside ? 1 : 0.25)
 {
 nt = nt / nc; ddn = ddn * nc;
 cos2t = 1.0f - nnt * nnt;
 D, N);
 }
 else
 {
 at a = nt - nc, b = nt + nc;
 at Tr = 1 - (R0 + (1 - R0) * ddn);
 (Tr) R = (D * nnt - N * (ddn > 0 ? 1 : -1));
 E * diffuse;
 = true;
 }
 if (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) &&
 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 fol
 (ive)
 ;
 at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf);
 survive;
 pdf;
 n = E * brdf * (dot(N, R) / pdf);
 ion = true;

```





```

ics
& (depth < MAXDEPTH) {
 // Inside?
 if (inside ? 1 : 0) {
 nt = nt / nc; ddn = ddn * nc;
 cos2t = 1.0f - nnt * ddn;
 D, N);
 }
 // Outside?
 at a = nt - nc, b = nt + nc;
 at Tr = 1 - (R0 + (1 - R0) * ddn);
 (Tr) R = (D * nnt - N * (ddn > 0 ? 1 : -1));
 // Diffuse
 E * diffuse;
 = true;
 // Refractive
 refl + refr)) && (depth < MAXDEPTH) {
 D, N);
 refl * E * diffuse;
 = true;
 }
 // Survival
 MAXDEPTH)
 survive = SurvivalProbability(diffuse);
 // Estimation
 estimation - doing it properly, closely following Small's
 if;
 radiance = SampleLight(&rand, I, &L, &lightPdf);
 e.x + radiance.y + radiance.z) > 0) && (depth < MAXDEPTH) {
 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.x + radiance.y + radiance.z);
 }
 // Random walk
 random walk - done properly, closely following Small's
 (survive)
 ;
 at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf);
 survive;
 pdf;
 n = E * brdf * (dot(N, R) / pdf);
 ion = true;
}

```

# Today's Agenda:

- Building Better BVHs
- Refitting
- Fast BVH Construction
- The Top-level BVH



# 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
- 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.*



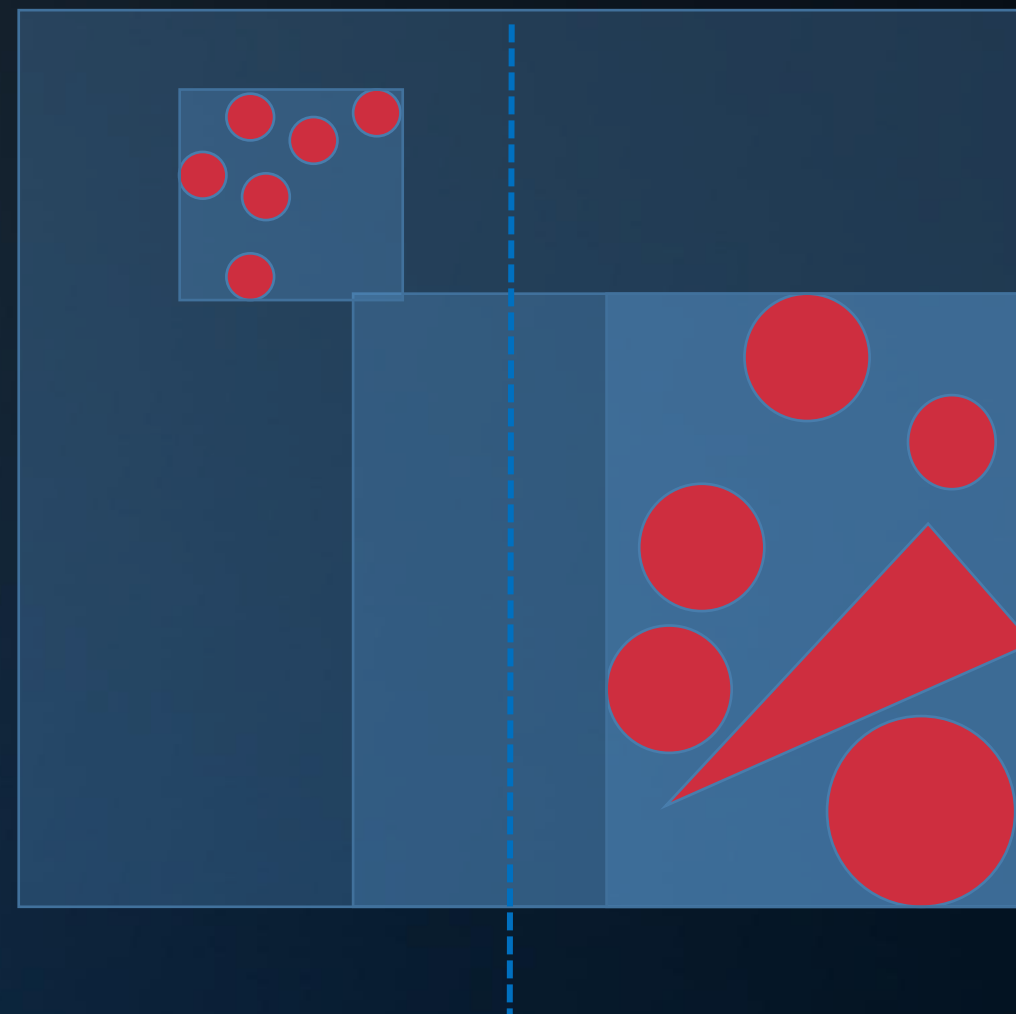
# Refitting

## 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



```

ics
& (depth < MAXDEPTH)
{
 if (inside ? 1 : 0)
 {
 nt = nt / nc; ddn = ddn / nc;
 cos2t = 1.0f - nnt * ddn;
 D, N);
 }
}

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

E * diffuse;
= true;

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

MAXDEPTH)

survive = SurvivalProbability(diffuse);
estimation - doing it properly, closely
df;
radiance = SampleLight(&rand, I, &L, &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 Small's
vive)
;
at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf);
survive;
pdf;
n = E * brdf * (dot(N, R) / pdf);
sion = true;

```



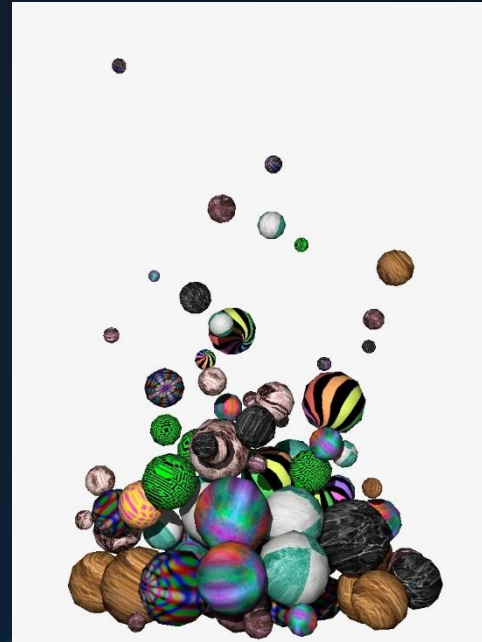
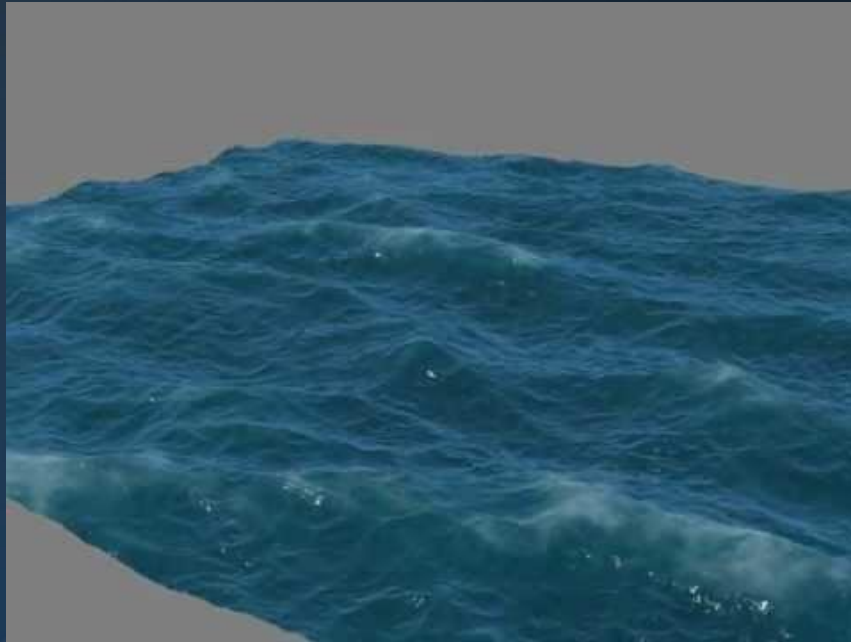


# Refitting

## Refitting - Suitability

```
ics
& (depth < MAXDEPTH)
```

```
nt = inside
nt = nt
os2t = 1
D, N);
)
at a = nt
at Tr = a
Tr) R = (E * diffuse
= true;
efl + refl * E
= true;
MAXDEPTH)
survive =
estimation
df;
radiance
e.x + rad
w = true;
at brdfPdf = EvaluateDiffuse(L, N) * PdfDiffuse
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

```

ics
& (depth < MAXDEPTH)
{
 if (inside ? 1 : 0)
 {
 nt = nt / nc; ddn = ddn * nc;
 pos2t = 1.0f - nnt * pos;
 D, N);
 }
 at a = nt - nc; b = nt * nc;
 at Tr = 1 - (R0 + (1 - R0) *
 Tr) R = (D * nnt - N * (ddn
 E * diffuse;
 = true;
 -
 efl + refr)) && (depth < MAXDEPTH)
 D, N);
 refl * E * diffuse;
 = true;
 MAXDEPTH)
 survive = SurvivalProbability(diffuse);
 estimation - doing it properly, closely following
 if;
 radiance = SampleLight(&rand, I, &L, &align,
 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 Small's
 vive)
 ;
 at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf);
 survive;
 pdf;
 n = E * brdf * (dot(N, R) / pdf);
 sion = true;

```



↑  
 ↑  
 Level 1  
 Root node

BVH node array

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();

```



```

ics
& (depth < MAXDEPTH) {
 // Inside?
 if (inside ? 1 : 0) {
 nt = nt / nc; ddn = ddn * nc;
 cos2t = 1.0f - nnt * ddn;
 D, N);
 }
 // Outside?
 at a = nt - nc, b = nt + nc;
 at Tr = 1 - (R0 + (1 - R0) * ddn);
 (Tr) R = (D * nnt - N * (ddn > 0 ? 1 : -1));
 // Diffuse
 E * diffuse;
 = true;
 // Refractive
 refl + refr)) && (depth < MAXDEPTH) {
 D, N);
 refl * E * diffuse;
 = true;
 }
 // Survival
 MAXDEPTH)
 survive = SurvivalProbability(diffuse);
 // Estimation
 estimation - doing it properly, closely following
 if;
 radiance = SampleLight(&rand, I, &L, &lightPdf);
 e.x + radiance.y + radiance.z) > 0) && (depth < MAXDEPTH) {
 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.x + radiance.y + radiance.z);
 }
 // Random walk
 random walk - done properly, closely following Small's
 (survive)
 ;
 at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf);
 survive;
 pdf;
 n = E * brdf * (dot(N, R) / pdf);
 ion = true;
}

```

# Today's Agenda:

- Building Better BVHs
- Refitting
- Fast BVH Construction
- The Top-level BVH



# Binning

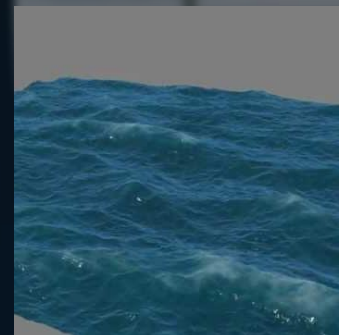
## Rapid BVH Construction

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  $N$



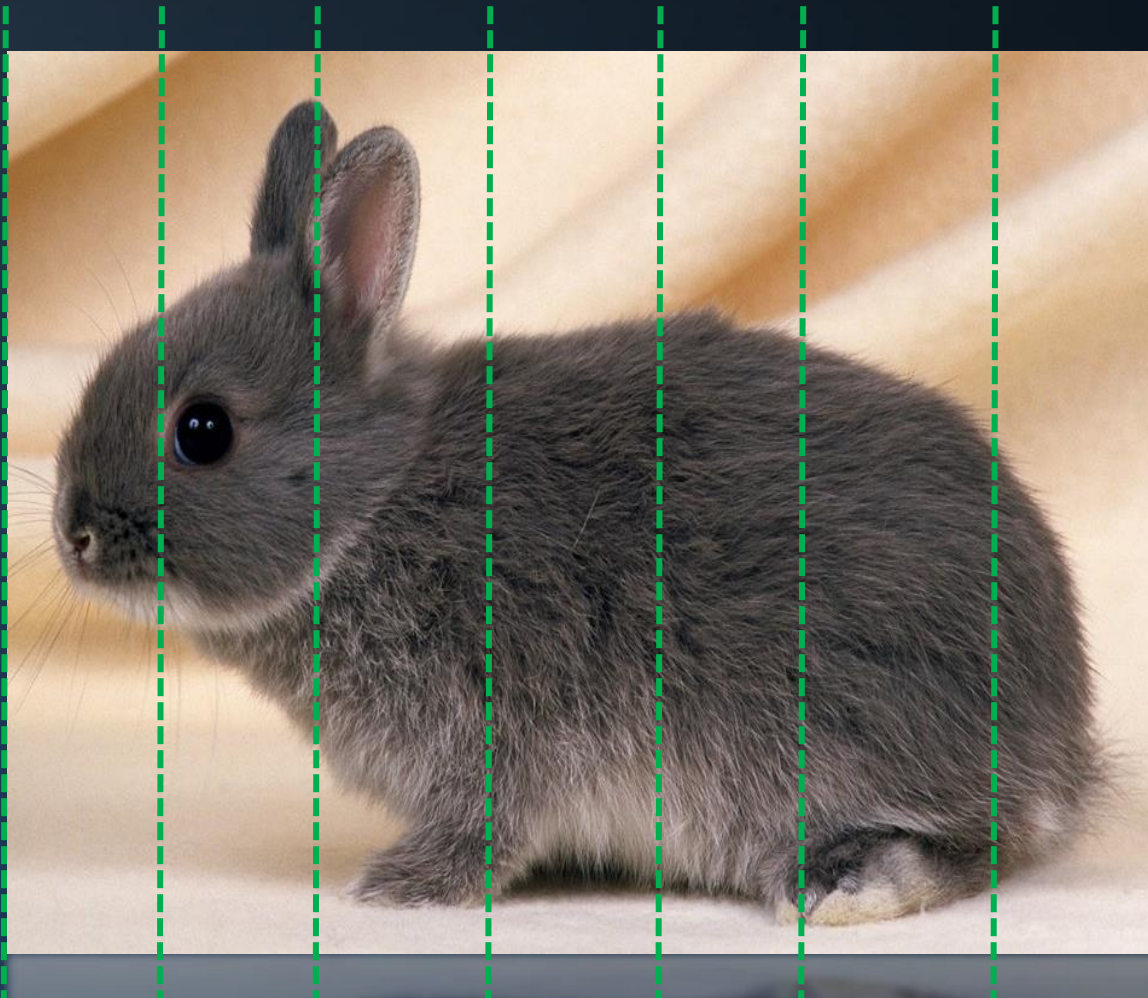


# Binning

```

ics
& (depth < MAXDEPTH)
{
 if (inside ? 1 : 0.25)
 {
 nt = nt / nc; ddn = ddn * ddn;
 cos2t = 1.0f - nnt * ddn;
 D, N);
 }
 at a = nt - nc, b = nt * nc;
 at Tr = 1 - (R0 + (1 - R0) * b);
 Tr) R = (D * nnt - N * (ddn * cos2t));
 E * diffuse;
 = true;
 refl + refr)) && (depth < MAXDEPTH)
 {
 D, N);
 refl * E * diffuse;
 = true;
 }
 MAXDEPTH)
 survive = SurvivalProbability(diffuse);
 estimation - doing it properly, closely following Small
 df;
 radiance = SampleLight(&rand, I, &L, &align, &pdf);
 e.x + radiance.y + radiance.z) > 0) && (depth < MAXDEPTH)
 {
 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 Small
 vive)
 ;
 at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf);
 survive;
 pdf;
 n = E * brdf * (dot(N, R) / pdf);
 sion = true;

```



# Binning

## Binned BVH Construction\*

Binned construction:

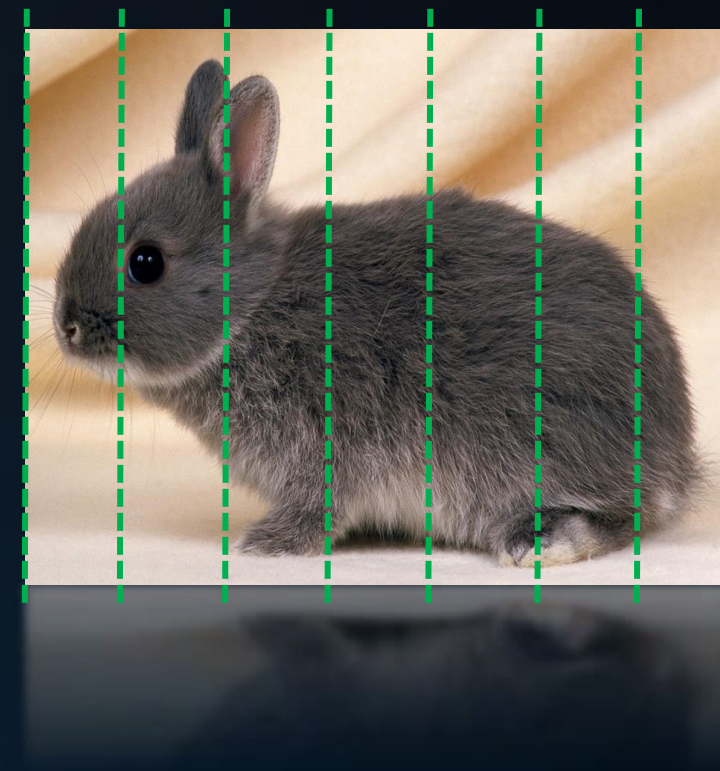
*Evaluate SAH at  $N$  discrete intervals.*

```

ics
& (depth < MAXDEPTH)
{
 if (inside ? 1 : 0)
 {
 nt = nt / nc; ddn = ddn * nc;
 cos2t = 1.0f - nnt * ddn;
 D, N);
 }
 {
 at a = nt - nc, b = nt + nc;
 at Tr = 1 - (R0 + (1 - R0) * a);
 Tr) R = (D * nnt - N * (ddn *
 }
 E * diffuse;
 = true;
 -
 efl + refr)) && (depth < MAXDEPTH)
 {
 D, N);
 efl * E * diffuse;
 = true;
 }
 MAXDEPTH)
 survive = SurvivalProbability(diffuse);
 estimation - doing it properly, closely following
 if;
 radiance = SampleLight(&rand, I, &L, &lightDir);
 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 Small's
 vive)
 ;
 at3 brdf = SampleDiffuse(diffuse, N, r1, r2, R
 survive;
 pdf;
 n = E * brdf * (dot(N, R) / pdf);
 ion = true;

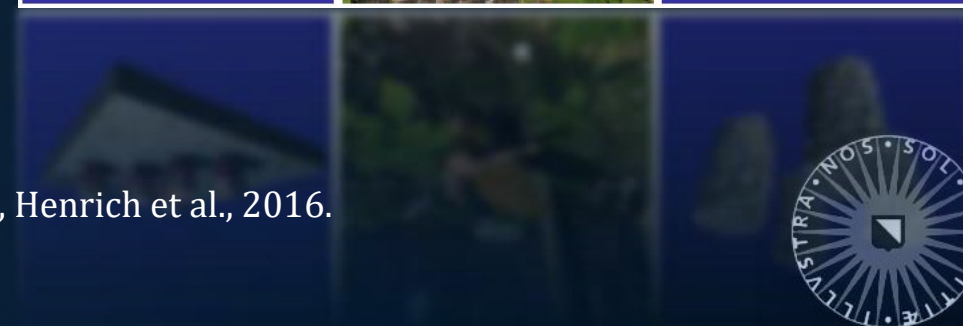
```

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



## Binned BVH Construction

472ms 7.88M triangles (12 cores @ 2Ghz)\*.



\*: Parallel BVH Construction using Progressive Hierarchical Refinement, Henrich et al., 2016.





```

ics
& (depth < MAXDEPTH) {
 // Inside?
 if (inside ? 1 : 0) {
 nt = nt / nc; ddn = ddn * nc;
 cos2t = 1.0f - nnt * ddn;
 D, N);
 }
 // Outside?
 at a = nt - nc, b = nt + nc;
 at Tr = 1 - (R0 + (1 - R0) * ddn);
 (Tr) R = (D * nnt - N * (ddn > 0 ? 1 : -1));
 // Diffuse
 E * diffuse;
 = true;
 // Refractive
 refl + refr)) && (depth < MAXDEPTH) {
 D, N);
 refl * E * diffuse;
 = true;
 }
 // Survival
 MAXDEPTH)
 survive = SurvivalProbability(diffuse);
 // Estimation - doing it properly, closely following
 if;
 radiance = SampleLight(&rand, I, &L, &lightPdf);
 e.x + radiance.y + radiance.z) > 0) && (depth < MAXDEPTH) {
 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.x + radiance.y + radiance.z);
 }
 // 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;
}

```

# Today's Agenda:

- Building Better BVHs
- Refitting
- Fast BVH Construction
- The Top-level BVH





# Top-level BVH

```
ics
& (depth < MAXDEPTH)
{
 if (inside ? 1 : 0.25)
 {
 nt = nt / nc; ddn = ddn * ddn;
 cos2t = 1.0f - nnt * ddn;
 D, N);
 }
 {
 at a = nt - nc, b = nt + nc;
 at Tr = 1 - (R0 + (1 - R0) * cos2t);
 (Tr) R = (D * nnt - N * (ddn * cos2t));
 }
 E * diffuse;
 = true;
 {
 refl + refr)) && (depth < MAXDEPTH)
 {
 D, N);
 refl * E * diffuse;
 = true;
 }
 MAXDEPTH)
 survive = SurvivalProbability(diffuse,
 estimation - doing it properly, closely
 df;
 radiance = SampleLight(&rand, I, &L, Align
 e.x + radiance.y + radiance.z) > 0) && (de
 w = true;
 at brdfPdf = EvaluateDiffuse(L, N) * Psur
 at3 factor = diffuse * INVPI;
 at weight = Mis2(directPdf, brdfPdf);
 at cosThetaOut = dot(N, L);
 E * ((weight * cosThetaOut) / directPdf);
 random walk - done properly, closely following
 (vive)
 ;
 at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf
 survive;
 pdf;
 n = E * brdf * (dot(N, R) / pdf);
 sion = true;
}
```





# Top-level BVH

```
ics
& (depth < MAXDEPTH)
{
 if (inside ? 1 : 0.5)
 {
 nt = nt / nc; ddn = ddn * ddn;
 r = 1.0f - nnt * ddn;
 D, N);
 }
 {
 a = nt - nc; b = nt + nc;
 at Tr = 1 - (R0 + (1 - R0) * r);
 Tr) R = (D * nnt - N * (ddn * r));
 }
 E * diffuse;
 = true;
 {
 refl + refr)) && (depth < MAXDEPTH)
 {
 D, N);
 refl * E * diffuse;
 = true;
 }
 MAXDEPTH)
 {
 survive = SurvivalProbability(diffuse, I,
 estimation - doing it properly, closely
 df;
 radiance = SampleLight(&rand, I, &L, Align
 e.x + radiance.y + radiance.z) > 0) && (de
 w = true;
 at brdfPdf = EvaluateDiffuse(L, N) * Psurf
 at3 factor = diffuse * INVPI;
 at weight = Mis2(directPdf, brdfPdf);
 at cosThetaOut = dot(N, L);
 E * ((weight * cosThetaOut) / directPdf);
 }
 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;
 }
}
```





321 20:31 368

A B C D E

D  
86 m  
AMMO DEPOT

G  
19 m

30 45 60



GERMANY

DEFEND  
AMMO DEPOT

bkevend72 16426 HOLD

Aoi\_Fuuka

Din0Rock

rocktoni1

68 m 17 m

100 m 9 17 | 11

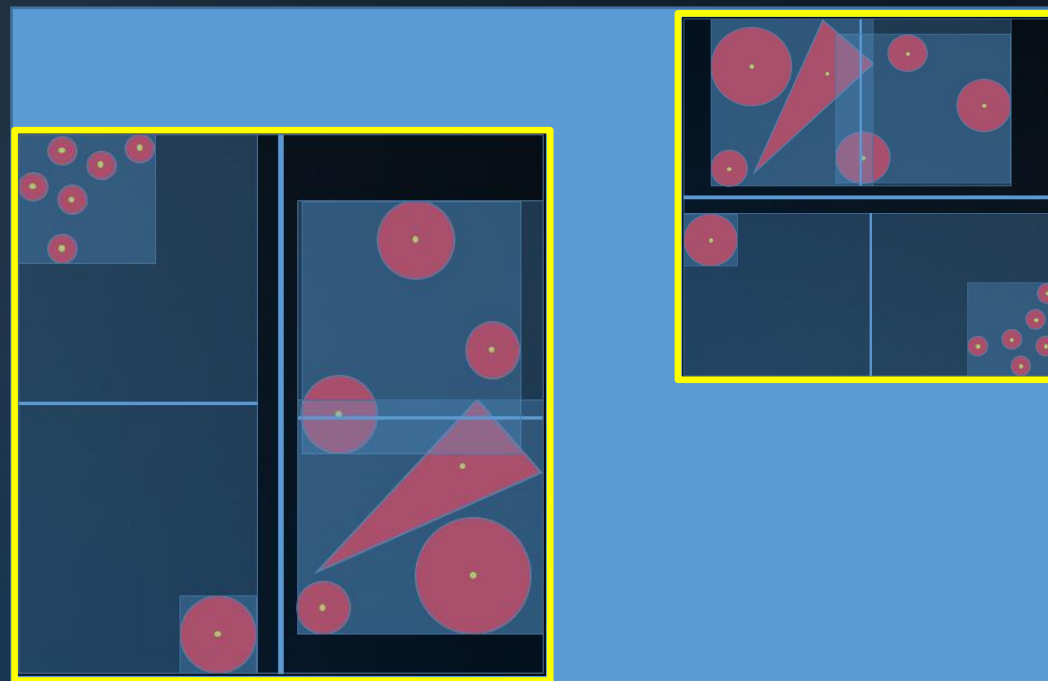
# Top-level BVH

## Combining BVHs

```

ics
& (depth < MAXDEPTH) {
 if (nt < nc) {
 nt = inside ? 1 + 1.25 * nc : 1.25 * nc;
 nt = nt / nc; ddn = ddn * nt;
 cos2t = 1.0f - nnt * ddn;
 D, N);
 }
 at a = nt - nc, b = nt * nc;
 at Tr = 1 - (R0 + (1 - R0) * a);
 Tr) R = (D * nnt - N * (ddn * nnt));
 E * diffuse;
 = true;
 defl + refr)) && (depth < MAXDEPTH) {
 D, N);
 refl * E * diffuse;
 = true;
 }
 MAXDEPTH)
 survive = SurvivalProbability(diffuse);
 estimation - doing it properly, closely following Small's
 if;
 radiance = SampleLight(&rand, I, &L, &align);
 e.x + radiance.y + radiance.z) > 0) && (depth < MAXDEPTH) {
 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 Small's
 (survive)
 ;
 at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf);
 survive;
 pdf;
 n = E * brdf * (dot(N, R) / pdf);
 ion = true;

```





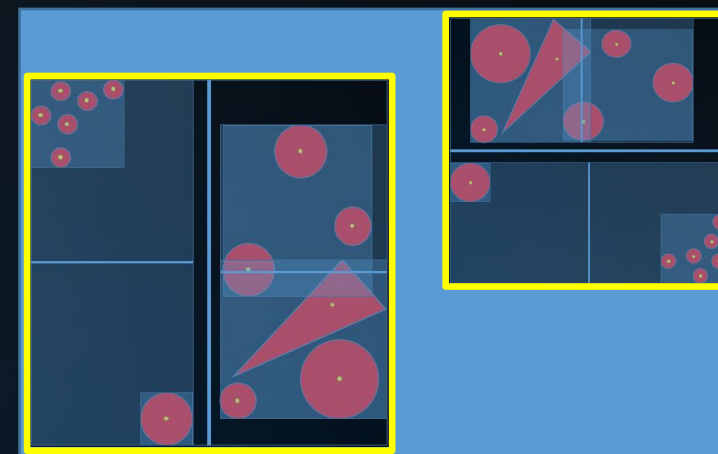
## Combining BVHs

```

145
k (depth < MAXDEPTH)
{
 nc = inside ? 1 : (1 - R0) * n;
 nt = nt / nc; ddn = ddn * nc;
 cos2t = 1.0f - nnt * nnt;
 D, N);
)
 at a = nt - nc, b = nt + nc;
 at Tr = 1 - (R0 + (1 - R0) * a);
 Fr) R = (D * nnt - N * ddn) * a;
 E * diffuse;
 = true;
 -
 efl + refr)) && (depth < MAXDEPTH))
 D, N);
 refl * E * diffuse;
 = true;
 MAXDEPTH)
 survive = SurvivalProbability(diffuse
 estimation - doing it properly, close
 df;
 radiance = SampleLight(&rand, I, &L
 .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) / direc
 random walk - done properly, closely
 (ive)
 ;
 at3 brdf = SampleDiffuse(diffuse, N
 survive;
 pdf;
 n = E * brdf * (dot(N, R) / pdf);
 sion = true;

```

- This works regardless of the method used to build each BVH
- This can be applied repeatedly to combine many BVHs

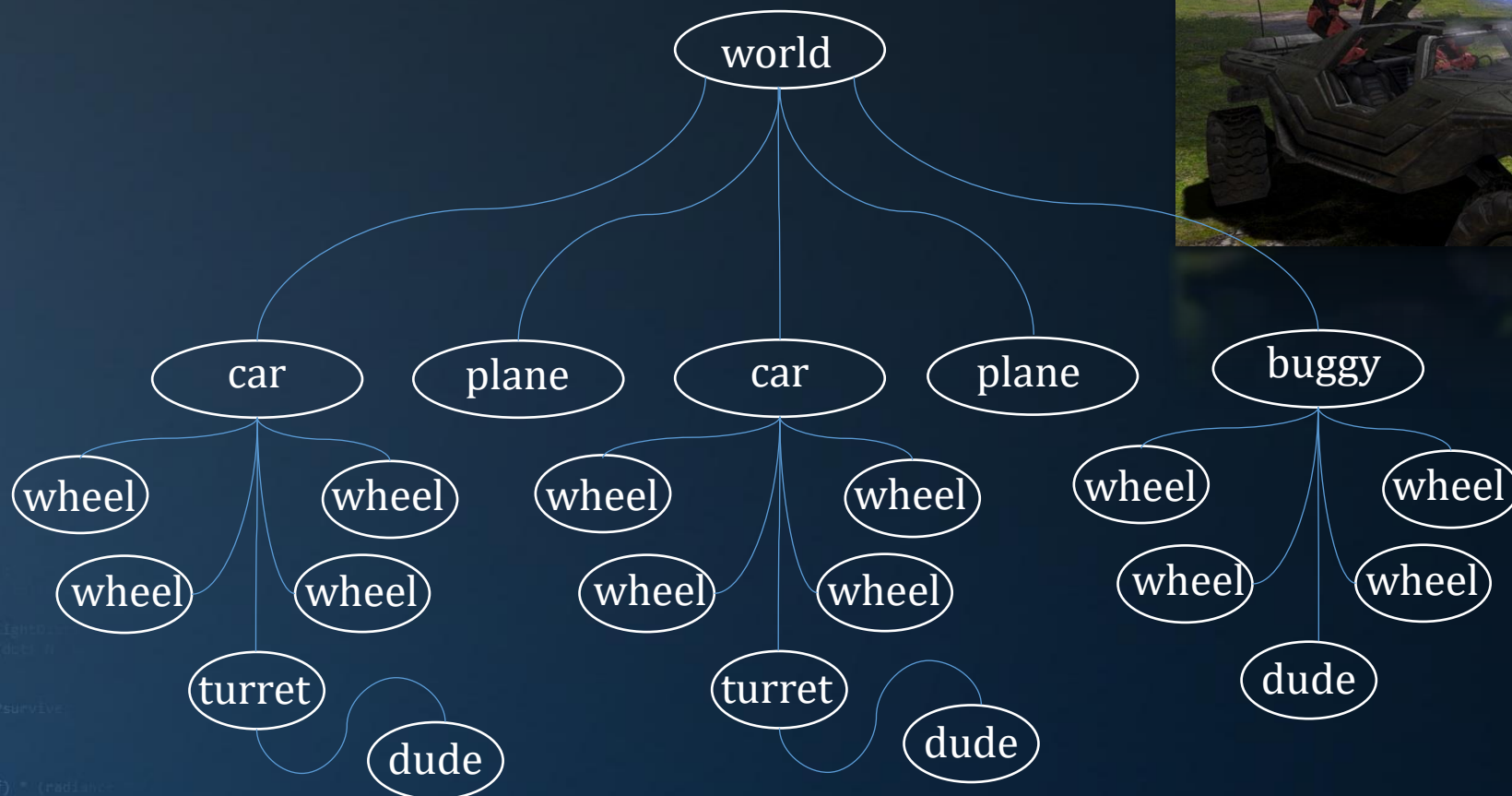






# Top-level BVH

## Scene Graph





# Top-level BVH

## Scene Graph

If our application uses a scene graph, we can construct a BVH for each scene graph node.

The BVH for each node is built using an appropriate construction algorithm:

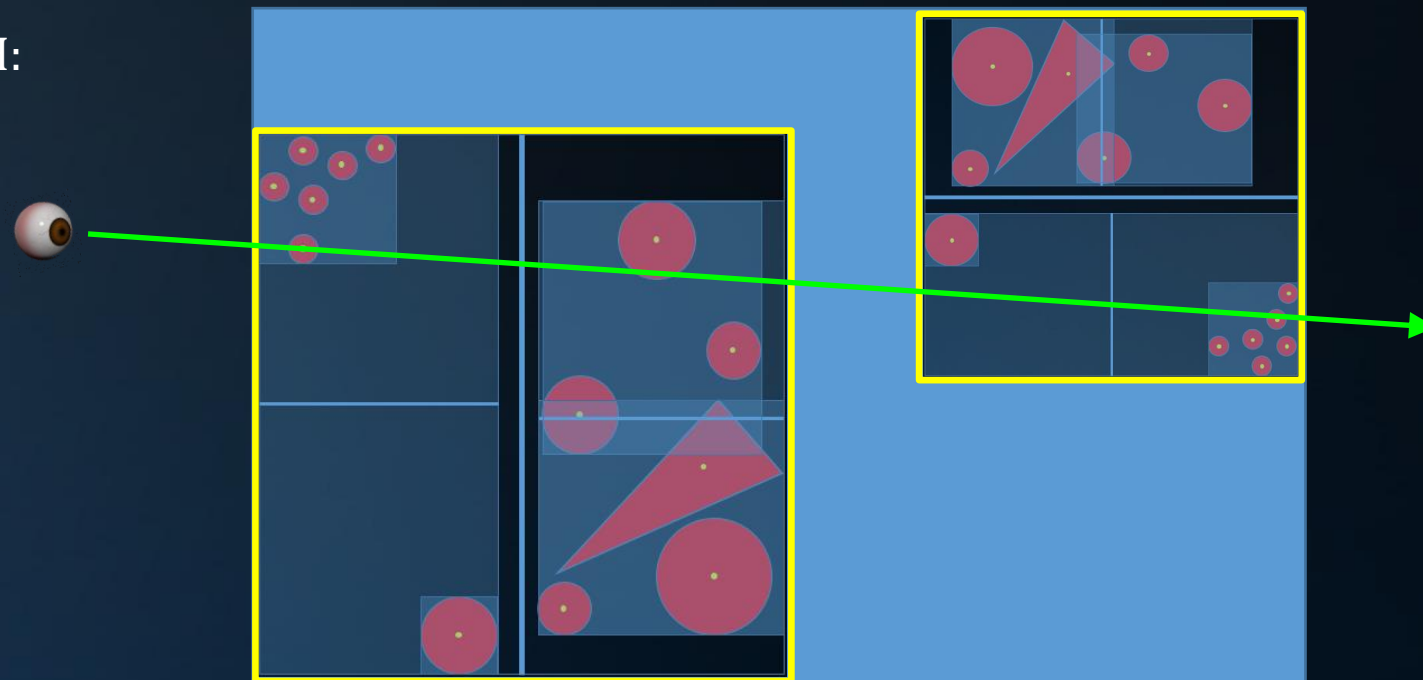
- High-quality SBVH for static scenery (offline)
- Fast binned SAH BVHs for dynamic scenery

The extra nodes used to combine these BVHs into a single BVH are known as the *Top-level BVH*.



# Rigid Motion

1. Refit the top-level BVH
2. Refit the affected BVH



# Top-level BVH

## Rigid Motion

Applying rigid motion to a BVH:

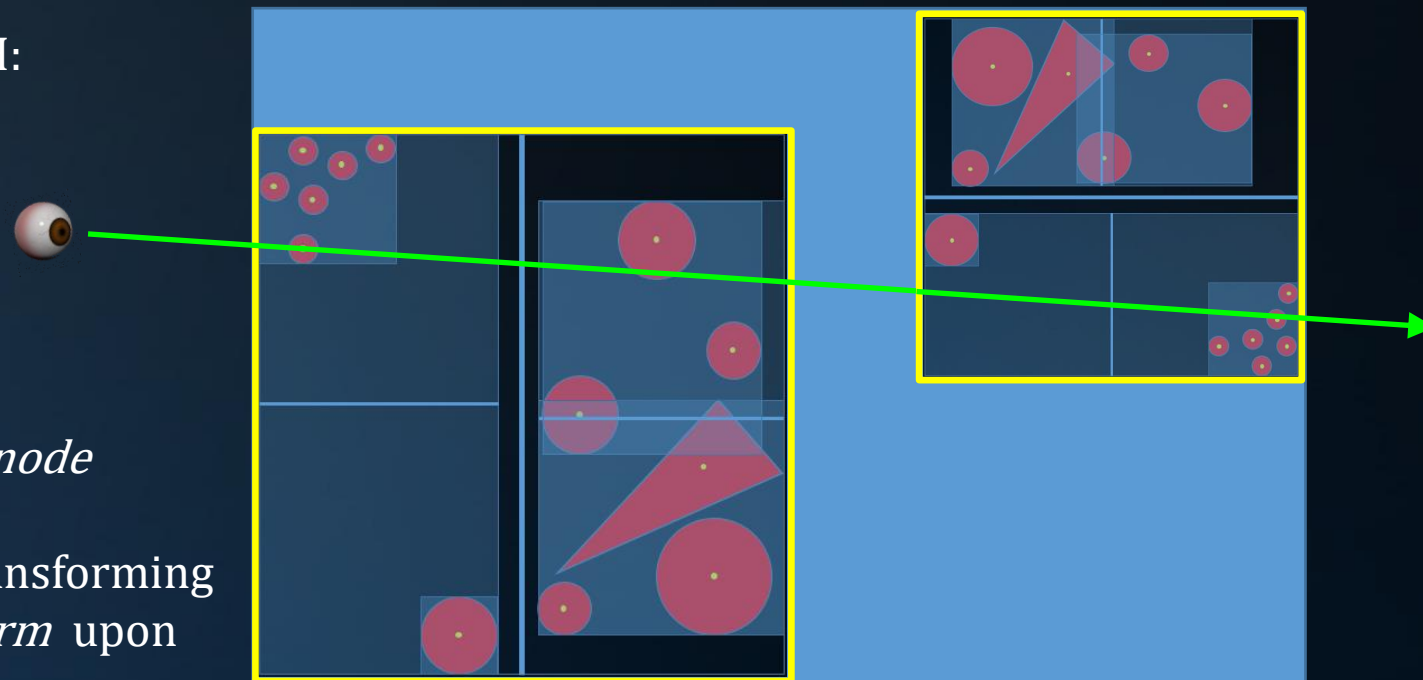
1. Refit the top-level BVH
2. Refit the affected BVH

or:

2. *Transform the ray, not the node*

Rigid motion is achieved by transforming the rays by the *inverse transform* upon entering the sub-BVH.

*(this obviously does not only apply to translation)*





# Top-level BVH

## The Top-level BVH - Construction

Input: *list of axis aligned bounding boxes for transformed scene graph nodes*

Algorithm:

1. Find the two elements in the list for which the AABB has the smallest surface area
2. Create a parent node for these elements
3. Replace the two elements in the list by the parent node
4. Repeat until one element remains in the list.

Note: algorithmic complexity is  $O(N^3)$ .

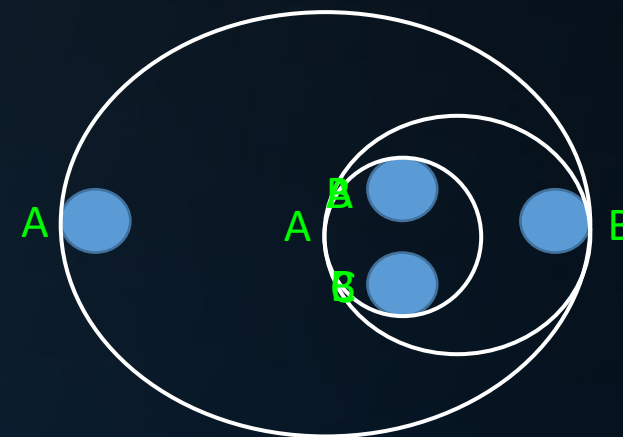


# Top-level BVH

## The Top-level BVH – Faster Construction\*

Algorithm:

```
Node A = list.GetFirst();
Node B = list.FindBestMatch(A);
while (list.size() > 1)
{
 Node C = list.FindBestMatch(B);
 if (A == C)
 {
 list.Remove(A);
 list.Remove(B);
 A = new Node(A, B);
 list.Add(A);
 B = list.FindBestMatch(A);
 }
 else A = B, B = C;
}
```



\*: Fast Agglomerative Clustering for Rendering, Walter et al., 2008



The leafs of the top-level BVH contain the sub-BVHs.

Once the sub-BVH has been traversed, we transform the ray again, this time by the transform matrix of the sub-BVH.

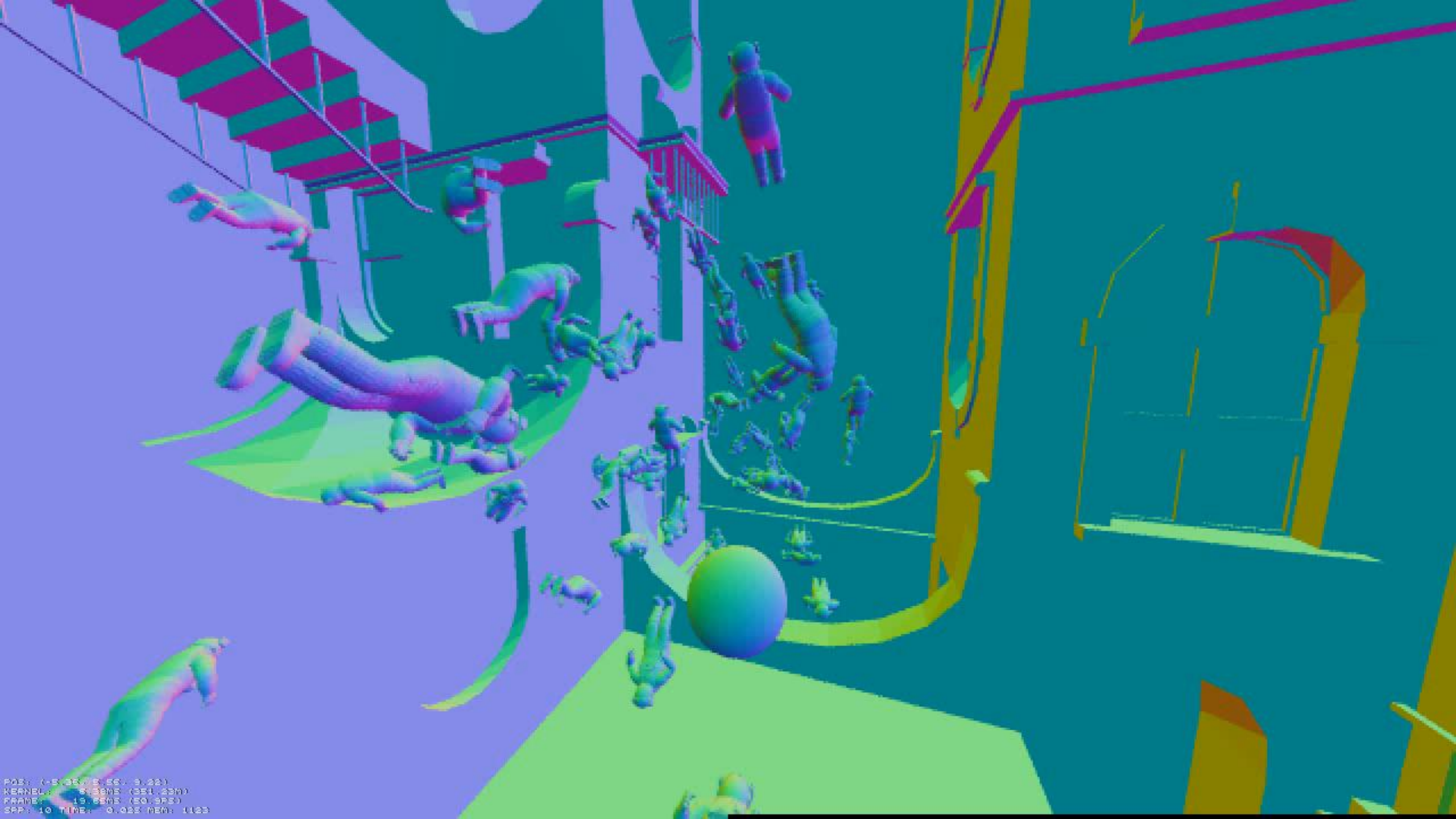
```

 & (depth < MAXDEPTH))
{
 if (nt > inside > 1.0) { // inside
 nc = nt / nc, ddn = ddn / nc;
 rns2t = 1.0f - nnt * nnt * nc;
 D, N);
 }
)
 {
 at a = nt - nc, b = nt + nc;
 at Tr = 1 - (R0 + (1 - R0) * nc);
 Tr) R = (D * nnt - N * (ddn *
)
 E * diffuse;
 = true;
)
 {
 refl + refr)) && (depth < MAXDEPTH))
 {
 D, N);
 refl * E * diffuse;
 = true;
 }
 MAXDEPTH)
 {
 survive = SurvivalProbability(diffuse,
 estimation - doing it properly, closely
 df;
 radiance = SampleLight(&rand, I, &l, &l
 2.x + radiance.y + radiance.z) > 0) && (
 w = true;
 at brdfPdf = EvaluateDiffuse(L, N) * P
 at3 factor = diffuse * INVPI;
 at weight = Mis2(directPdf, brdfPdf);
 at cosThetaOut = dot(N, L);
 E * ((weight * cosThetaOut) / directPdf)
 random walk - done properly, closely follow
 (survive)
 }
 ;
 at3 brdf = SampleDiffuse(diffuse, N, r1,
 survive;
 pdf;
 n = E * brdf * (dot(N, R) / pdf);
 sion = true;
}

```







POS: (-5.35, 15.55, 3.22)  
KERNEL: 4.58MS (351,230)  
FRAME: 15.58MS (50,995)  
SPP: 10 TIME: 0.025 MEM: 1123

# Top-level BVH

## The Top-level BVH – Summary

The top-level BVH enables complex animated scenes:

- for static objects, it contains high-quality sub-BVHs;
- for objects undergoing rigid motion, it also contains high-quality sub-BVHs, with a transform matrix and its inverse;
- for deforming objects, it contains sub-BVHs that can be refitted;
- for arbitrary animations, it contains lower quality sub-BVHs.

Combined, this allows for efficient maintenance of a global BVH.

```

ics
& (depth < MAXDEPTH)
{
 if (nt < inside ? 1.f : 0.f)
 {
 nt = nt / nc; ddn = dot(N, D);
 cos2t = 1.0f - nnt * nnt;
 D, N);
 }
 else
 {
 at a = nt - nc, b = nt + nc;
 at Tr = 1 - (R0 + (1 - R0) * f);
 Tr) R = (D * nnt - N * (ddn *
 }
 E * diffuse;
 = true;
 -
 refl + refr)) && (depth < MAXDEPTH)
 {
 D, N);
 refl * E * diffuse;
 = true;
 }
 MAXDEPTH)
 survive = SurvivalProbability(diffuse);
 estimation - doing it properly, closely following
 if;
 radiance = SampleLight(&rand, I, &L, &lightPos);
 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 Small's
 vive)
 ;
 at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf);
 survive;
 pdf;
 n = E * brdf * (dot(N, R) / pdf);
 sion = true;

```



# INFOMAGR – Advanced Graphics

Jacco Bikker - November 2022 - February 2023

## END of “The Perfect BVH”

next lecture: “Path Tracing”

