

```
ics
& (depth < MAXDEPTH))
{
    if (inside ? 1 : 1.25)
    {
        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) * ddn);
    Tr) R = (D * nnt - N * (ddn > 0)

    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) && (dot( N, L )
    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;
```

6

# Ray Tracing for Games

Dr. Jacco Bikker - IGAD/BUAS, Breda, February 3

# Welcome!



## Agenda:

- Combining BVHs
- Refitting
- Rigid Motion

```
ics
& (depth < MAXDEPTH))
{
    if (nt < 0)
        nt = inside ? 1 : -1;
    nt = nt / nc; ddn = dot(N, N);
    float r0 = 1 - nt; r1 = 0; r2 = 0;
    float s2t = 1.0f - nnt * nnt;
    float r = D, N );
    if (r < 0)
        r = 0;
    float a = nt - nc, b = nt + nc;
    float Tr = 1 - (R0 + (1 - R0) * r);
    float R = (D * nnt - N * (ddn * nnt));
    if (E * diffuse;
    = true;
    -
    refl + refr)) && (depth < MAXDEPTH))
    {
        D, N );
        refl * E * diffuse;
        = true;
    }
}

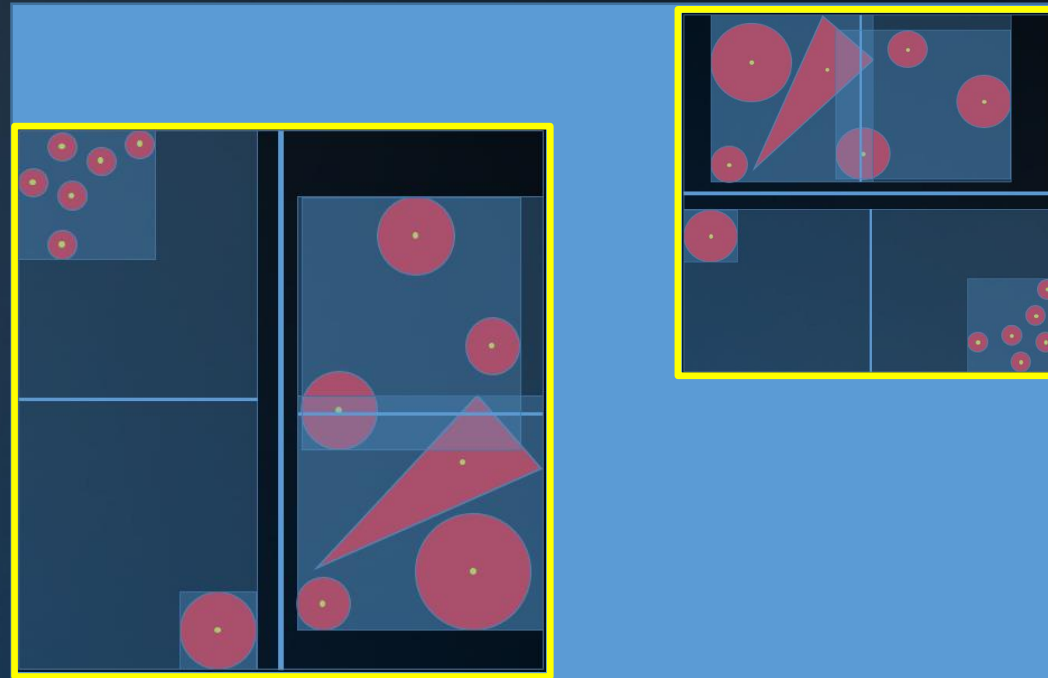
MAXDEPTH)

survive = SurvivalProbability( diffuse );
estimation - doing it properly, closely following Small's
df;
radiance = SampleLight( &rand, I, &L, &lightPdf );
e.x + radiance.y + radiance.z) > 0) && (dot(N, L) > 0)
{
    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;
}
```



## Combining BVHs

```
ics
& (depth < MAXDEPTH)
{
    if (nt < nc)
    {
        nt = inside ? 1 + 1.2f * nc : 1.5f * nc;
        nc = nt / nc; ddn = ddn * nc;
        rands2t = 1.0f - nnt * nnt;
        D, N );
    }
    else
    {
        at a = nt - nc, b = nt * nc;
        at Tr = 1 - (R0 + (1 - R0) * a);
        Tr) R = (D * nnt - N * (ddn * nnt));
    }
    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, &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
    }
    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

BVH #1: Car 1

BVH #2: Car 2

BVH #3: Track

Per frame:

- Rebuild BVH for each car
- Combine car 1 and car 2
- Combine track and cars
- Trace.

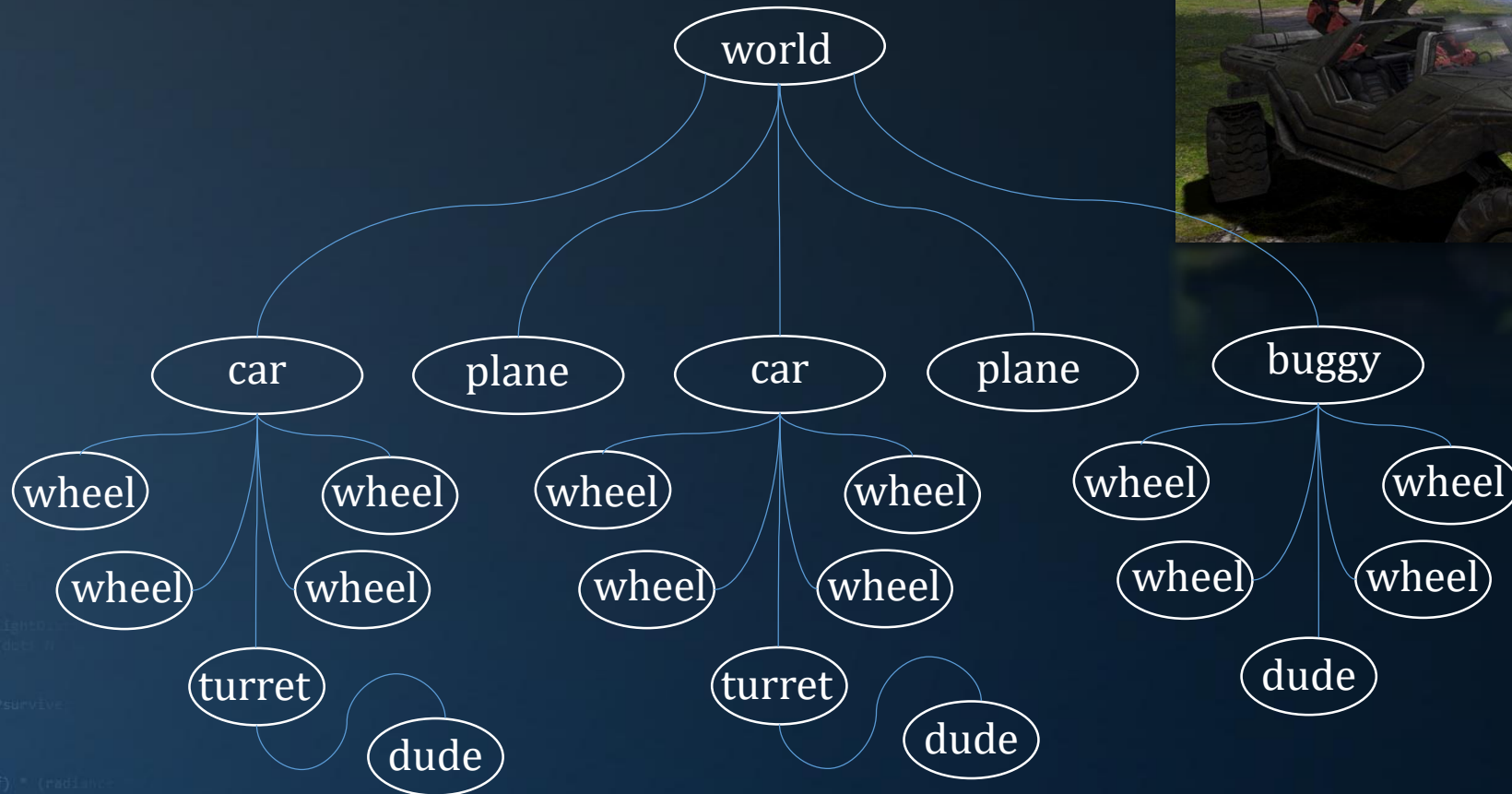








## Scene Graph



## 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*.



## 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  $O(N^3)$ .

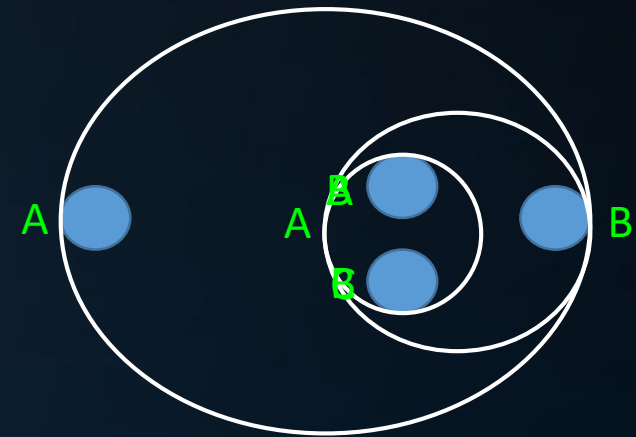




## The Top-level BVH – Faster Construction\*

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



## Agenda:

- Combining BVHs
- Refitting
- Rigid Motion

```
ics
& (depth < MAXDEPTH))
{
    if ( ! inside )
    {
        nt = nt / nc, ddn = ddn * nc;
        r = 1.0f - nnt * ddn;
        r = (D, N );
        r = (0);
        if ( r < 0 )
        {
            a = nt - nc, b = nt + nc;
            at Tr = 1 - (R0 + (1 - R0) * r);
            Tr) R = (D * nnt - N * (ddn * nnt));
            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, &lightPdf );
        e.x + radiance.y + radiance.z) > 0) && (dot( N, L ) > 0)
        {
            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;
    }
}
```



## Summary of BVH Characteristics

A BVH provides significant freedom.

- 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

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.*

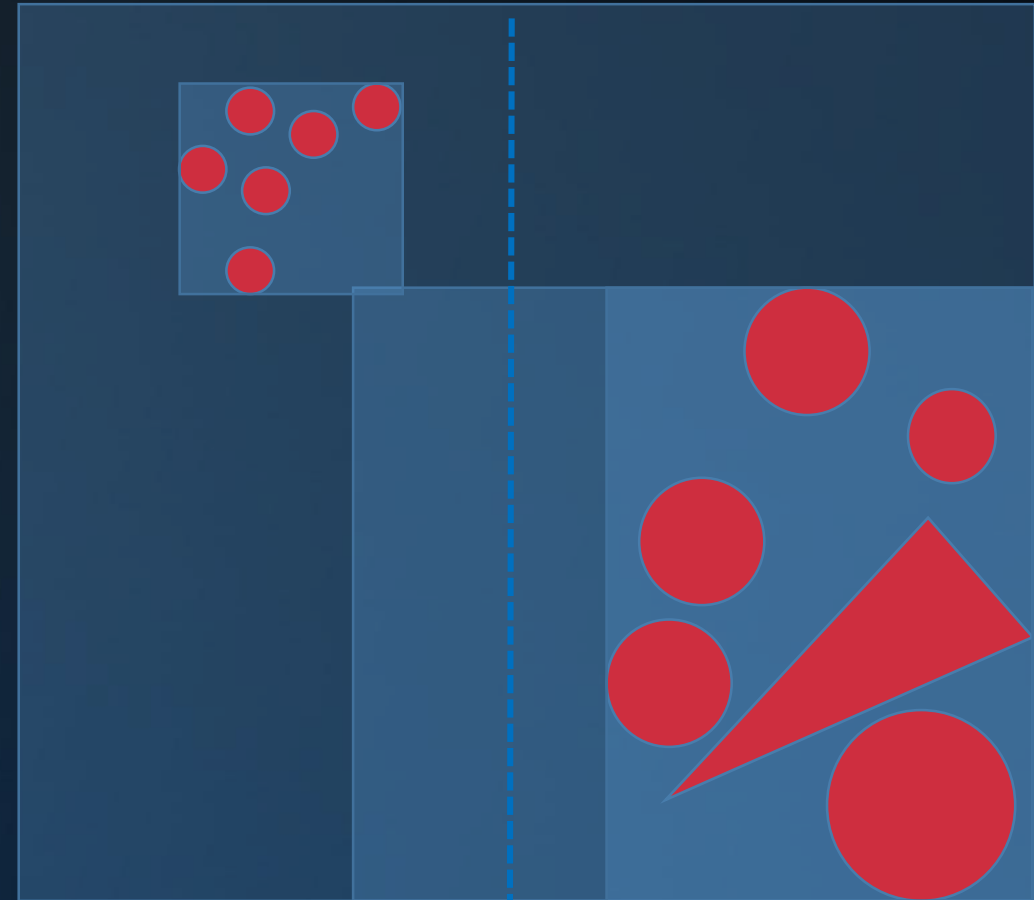


## Refitting

*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 - Suitability

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

```
t = inside  
nt = nt + 1  
os2t = 1  
D, N );  
)  
at a = nt  
at Tr = 1  
Tr) R = (
```

```
E * diffuse  
= true;  
efl + ref
```

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

```
(MAXDEPTH)
```

```
survive =  
estimat  
df;  
radiance  
e.x + rad
```

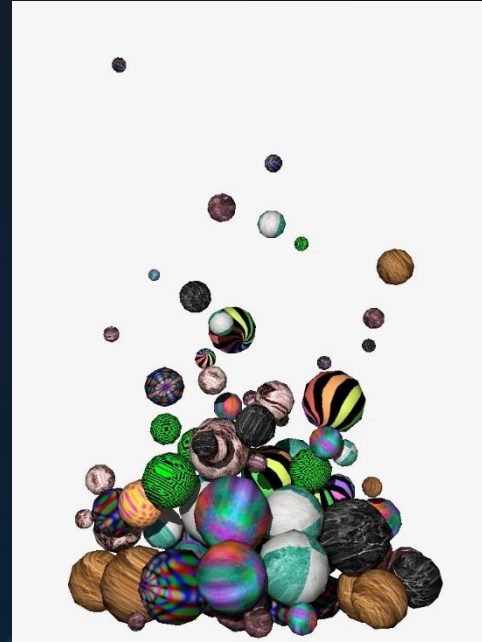
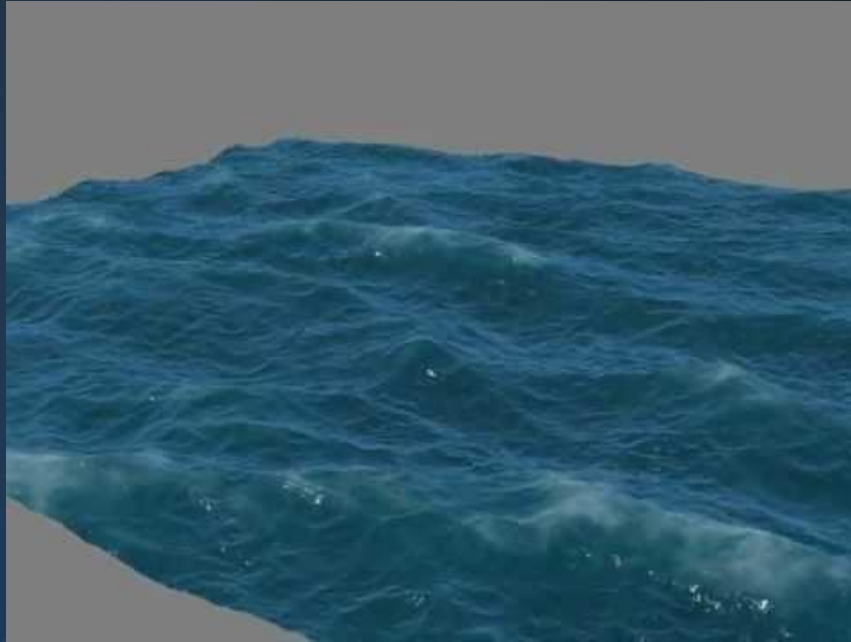
```
w = true;
```

```
at brdfPdf = EvaluateDiffuse( L, N ) * PdfDiff  
at3 factor = diffuse * INVPI;  
at weight = Mis2( directPdf, brdfPdf );  
at cosThetaOut = dot( N, L );  
E * ((weight * cosThetaOut) / directPdf) * (radiance
```

```
andom walk - done properly, closely following Sussner's  
ive)
```

```
;
```

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





## Refitting – Practical

```
ics
& (depth < MAXDEPTH)
{
    if (inside) {
        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 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;
}
```

0

N-1

↑ ↑  
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();
```



## Agenda:

- Combining BVHs
- Refitting
- Rigid Motion

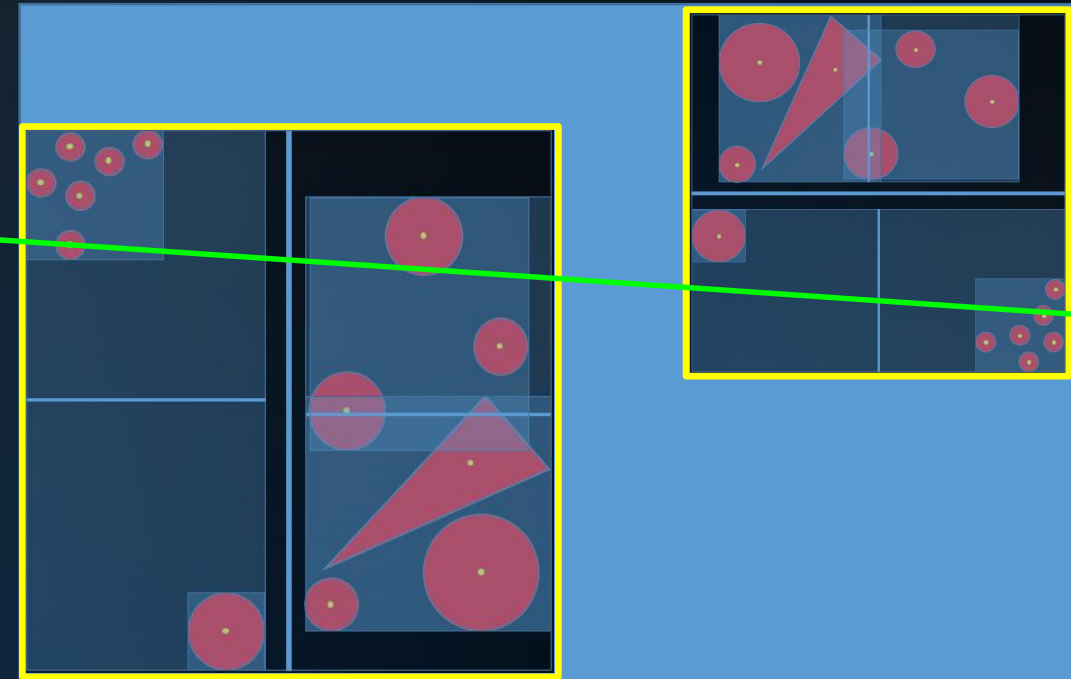
```
ics
& (depth < MAXDEPTH))
{
    if (nt < 0)
        nt = inside ? 1 : -1;
    nt = nt / nc; ddn = dot(N, N);
    float r0 = 1 - nt; r1 = 0; r2 = 0;
    float s2t = 1.0f - nnt * nnt;
    float r = D, N );
    if (r < 0)
        r = 0;
    float a = nt - nc, b = nt + nc;
    float Tr = 1 - (R0 + (1 - R0) * r);
    float R = (D * nnt - N * (ddn * nnt));
    if (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, &lightPdf );
    e.x + radiance.y + radiance.z) > 0) && (dot(N, L) > 0)
    {
        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;
}
```



## Rigid Motion

Applying rigid motion to a BVH:

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



```
ics
& (depth < MAXDEPTH)
{
    if (inside ? 1 : 1.0f - 0.5f * nnt)
    {
        nt = nt / nc; ddn = ddn * nc;
        pos2t = 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 * nnt));
    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, &lightPdf );
    e.x + radiance.y + radiance.z) > 0) && (dot( N, L ) > 0)
    {
        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;
}
```





## Rigid Motion

Applying rigid motion to a BVH:

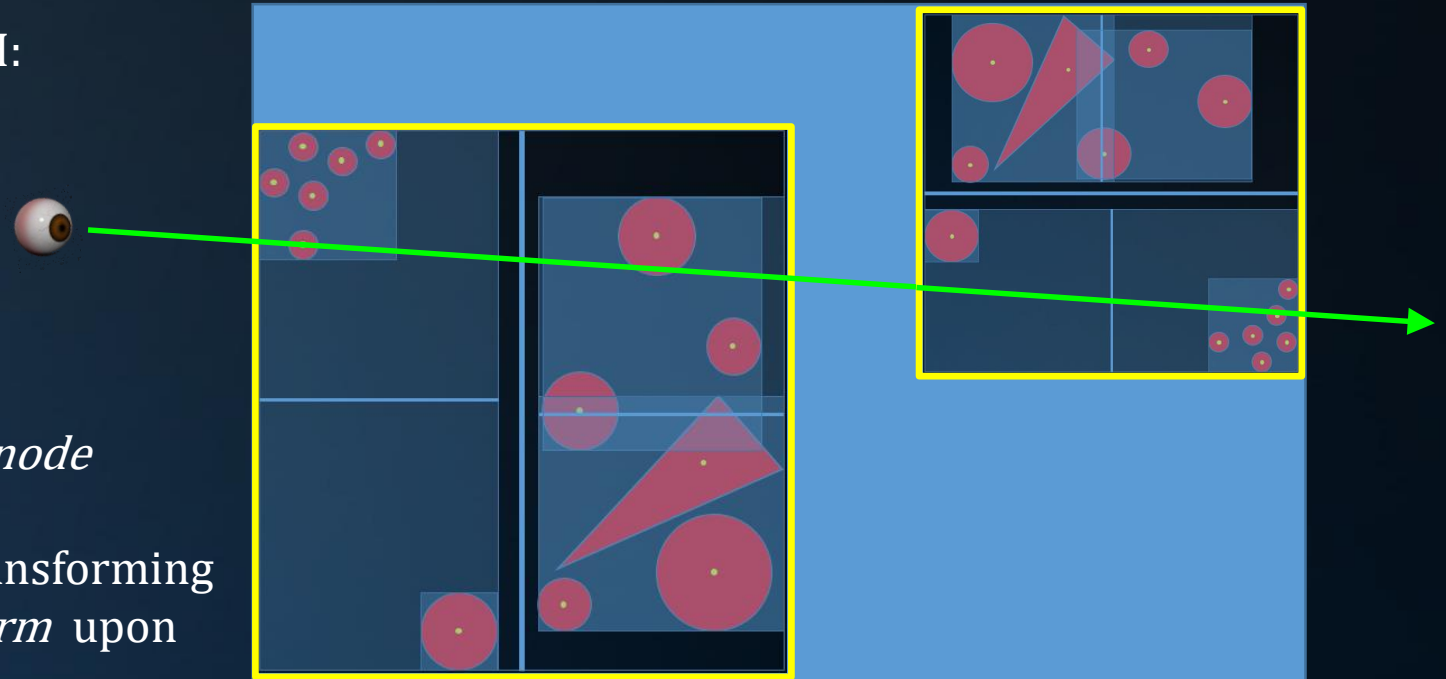
1. Refit the top-level BVH
2. Refit the affected BVH

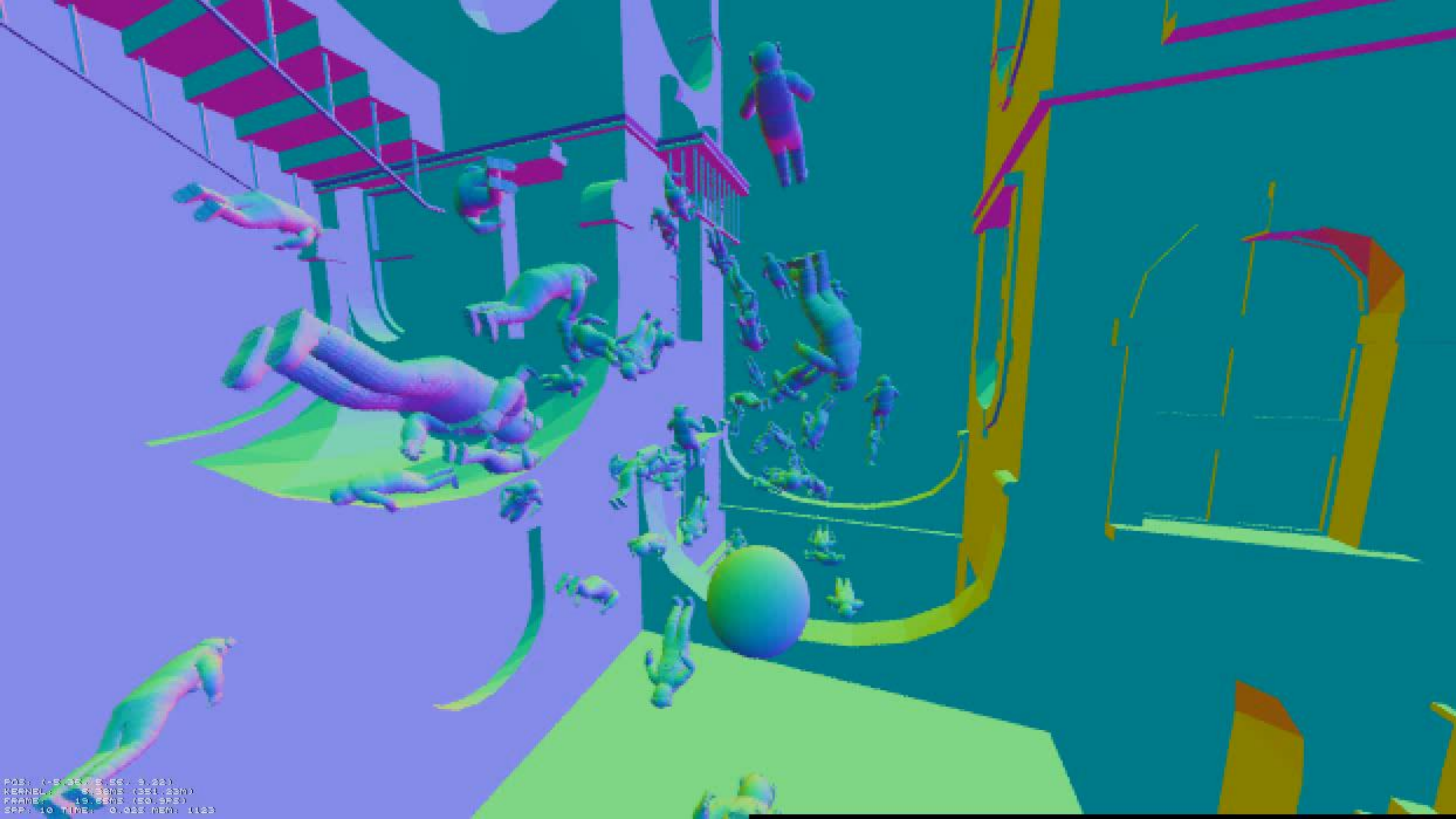
or:

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)*





Pos: (-5.35, 5.55, 9.22)  
Kernel: 8.35ms (351.33ms)  
Frame: 19.85ms (50.9FPS)  
SPP: 10 Time: 0.025 Mem: 1123

## Agenda:

- Combining BVHs
- Refitting
- Rigid Motion

```
ics
& (depth < MAXDEPTH))
{
    if (nt < 0)
        nt = inside ? 1 : -1;
    nt = nt / nc; ddn = dot(N, N);
    float r0 = 1 - nt; r1 = 0; r2 = 0;
    float s2t = 1.0f - nnt * nnt;
    float r = D, N );
    if (r < 0)
        r = 0;
    float a = nt - nc, b = nt + nc;
    float Tr = 1 - (R0 + (1 - R0) * r);
    float R = (D * nnt - N * (ddn * nnt));
    if (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, &lightPdf );
        e.x + radiance.y + radiance.z) > 0) && (dot(N, L) > 0)
        {
            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;
        }
    }
}
```





# End of PART 6.

```
ics
& (depth < MAXDEPTH)
{
    if (inside ? 1 : 1.25 * nnt)
    {
        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) * ddn);
        Tr) R = (D * nnt - N * (ddn < 0));
    }
    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, &lightDir );
    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);
    sion = true;
}
```

