

INFOGR – Computer Graphics

J. Bikker - April-July 2015 - Lecture 7: “Visibility”

Welcome!



Perpendicular

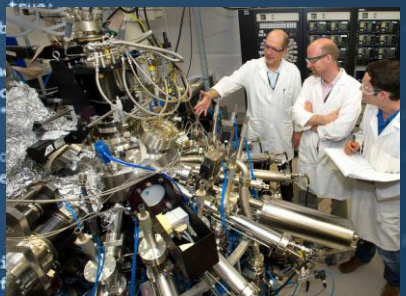
Vectors perpendicular to $\begin{pmatrix} x \\ y \end{pmatrix}$: $\begin{pmatrix} -y \\ x \end{pmatrix}, \begin{pmatrix} y \\ -x \end{pmatrix}$

Calculating a vector perpendicular to $\begin{pmatrix} x \\ y \\ z \end{pmatrix}$: ~~$\begin{pmatrix} -y \\ x \\ z \end{pmatrix}$~~ $\begin{pmatrix} -y \\ x \\ 0 \end{pmatrix}$

**additional rules apply*

Verify:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \cdot \begin{pmatrix} -y \\ x \\ 0 \end{pmatrix} = x * -y + y * x + z * 0 = 0.$$



```
...ics
& (depth < MAXDEPTH)
...
t = inside / (1.0 - refl);
nt = nt / nc; rdd = ...
...
os2t = 1.0f - nnt; ...
D, N );
...
)
...
at a = nt - nc; b = nt; ...
at Tr = 1 - (RB + (1 - RB) * ...
Tr) R = (D * nnt - N * (dd
...
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, M, &light, ...
e.x + radiance.y + radiance.z) > 0) && (depth <
v = true;
at brdfPdf = EvaluateDiffuse( L, N ) * Survive;
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
ive)
;
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf);
urvive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;
```

Today's Agenda:

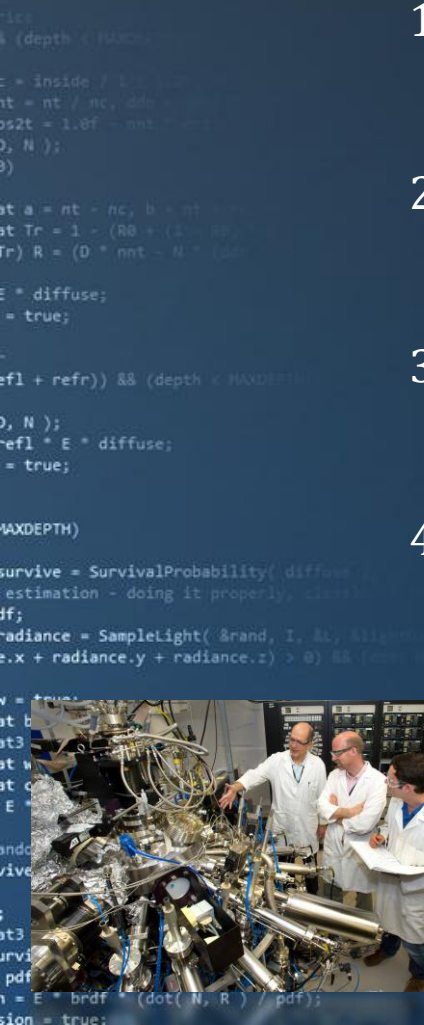
- Depth Sorting
- Clipping
- Visibility
- The Midterm Exam



Depth Sorting

Rendering – Functional overview

1. Transform:
translating / rotating meshes
2. Project:
calculating 2D screen positions
3. Rasterize:
determining affected pixels
4. Shade:
calculate color per affected pixel



Animation, culling,
tessellation, ...

meshes

Transform

vertices

Project

vertices

Rasterize

fragment positions

Shade

pixels

Postprocessing



Depth Sorting

3. Rasterize: *determining affected pixels*

Questions:

- What is the screen space position of the fragment?
- Is that position actually on-screen?
- Is the fragment the nearest fragment for the affected pixel?

How do we efficiently determine visibility of a pixel?

Animation, culling,
tessellation, ...

meshes

Transform

vertices

Project

vertices

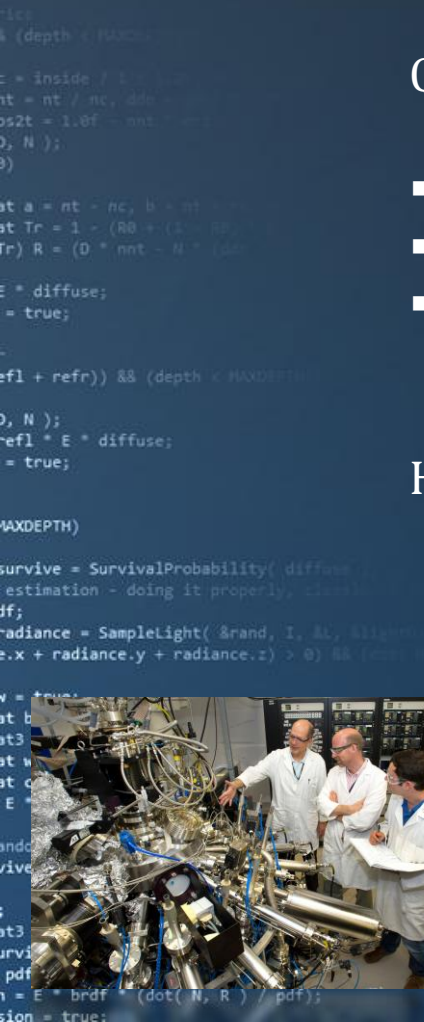
Rasterize

fragment positions

Shade

pixels

Postprocessing



Part of the tree is off-screen

Too far away to draw

Tree requires little detail

City obscured by tree

Torso closer than ground

Tree between ground & sun



Depth Sorting

Old-skool depth sorting: Painter’s Algorithm

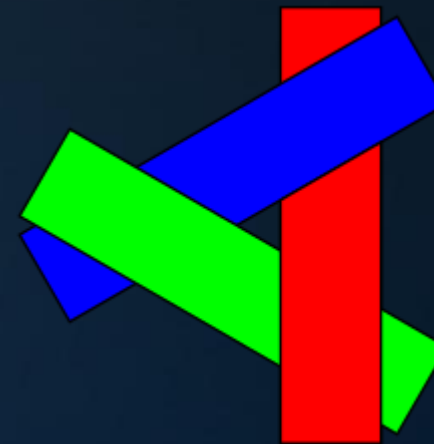
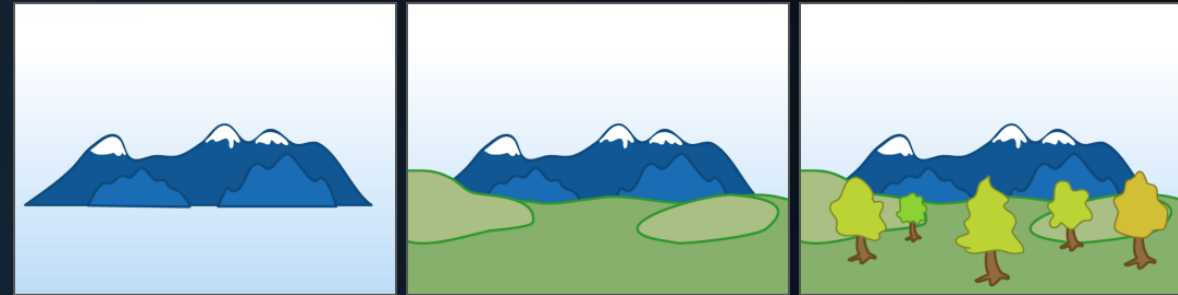
- Sort polygons by depth
- Based on polygon center
- Render depth-first

Advantage:

- Doesn’t require z-buffer

Problems:

- Cost of sorting
- Doesn’t handle all cases
- Overdraw



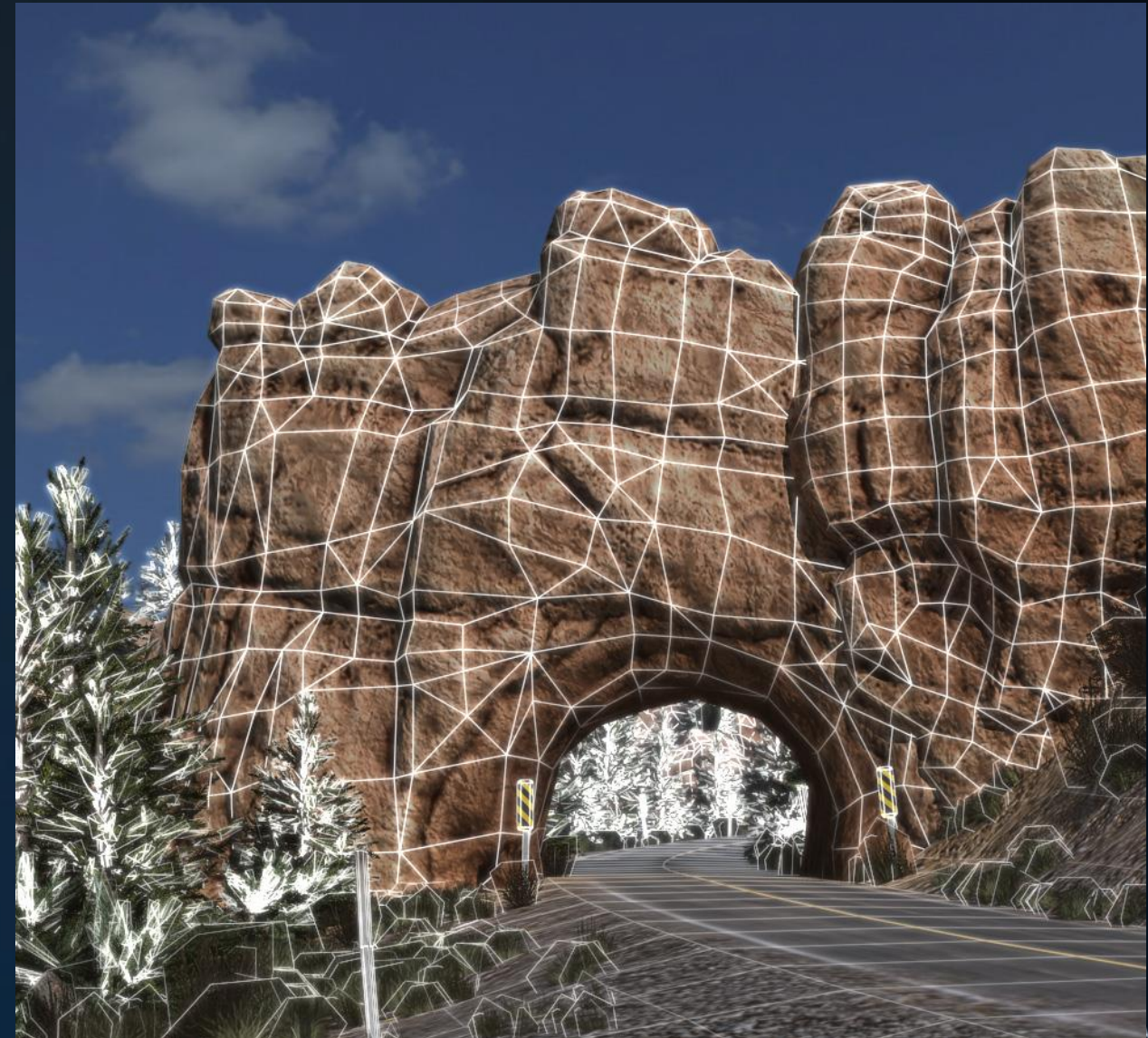
Depth Sorting

Overdraw:

Inefficiency caused by drawing multiple times to the same pixel.

```
...
    & (depth < MAXDEPTH)
...
    if (inside) {
        nt = nt / nc;
        pos2t = 1.0f / nnt;
        D, N );
    }
...
    at a = nt - nc, b = nt;
    at Tr = 1 - (R0 + (1 - R0) * R);
    Tr) R = (D * nnt - N * (a *
...
    E * diffuse;
    = true;
...
    refl + refr) && (depth < MAXDEPTH)
...
    D, N );
    refl * E * diffuse;
    = true;
...
MAXDEPTH)
...
survive = SurvivalProbability( diffuse;
estimation - doing it properly, check
if;
radiance = SampleLight( @rand, I, M, Alignment
e.x + radiance.y + radiance.z) > 0) && (max
...
    = true;
    at b
    at3
    at w
    at c
    at E *
...
and
ive
...
at3
surv
pdf
n = E * brdf * (dot( N, R ) / pdf);
sion = true;
...

```



Depth Sorting

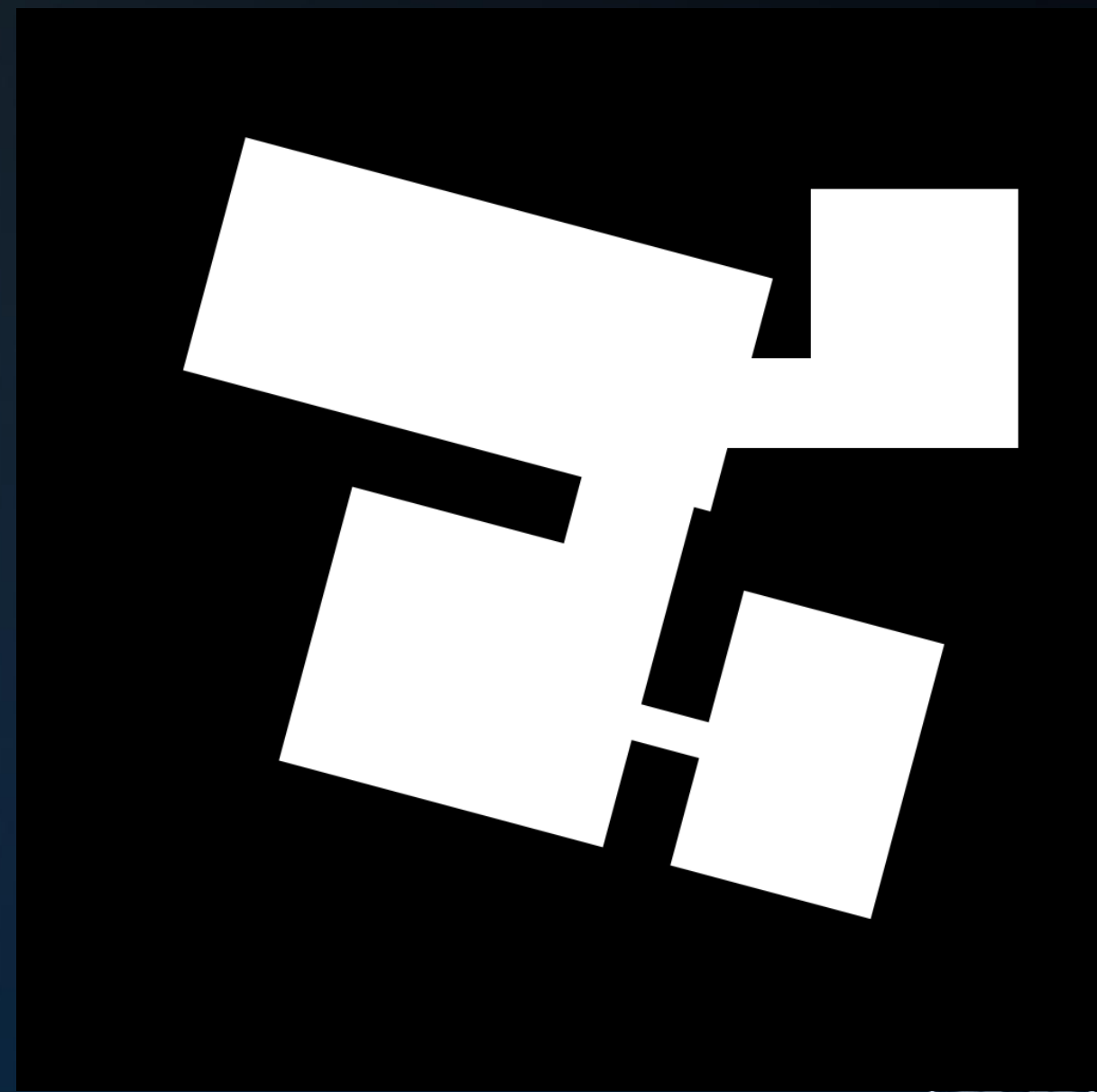
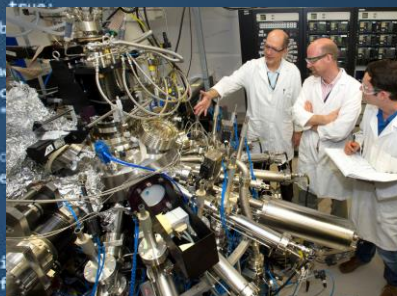
Correct order: BSP

root

```

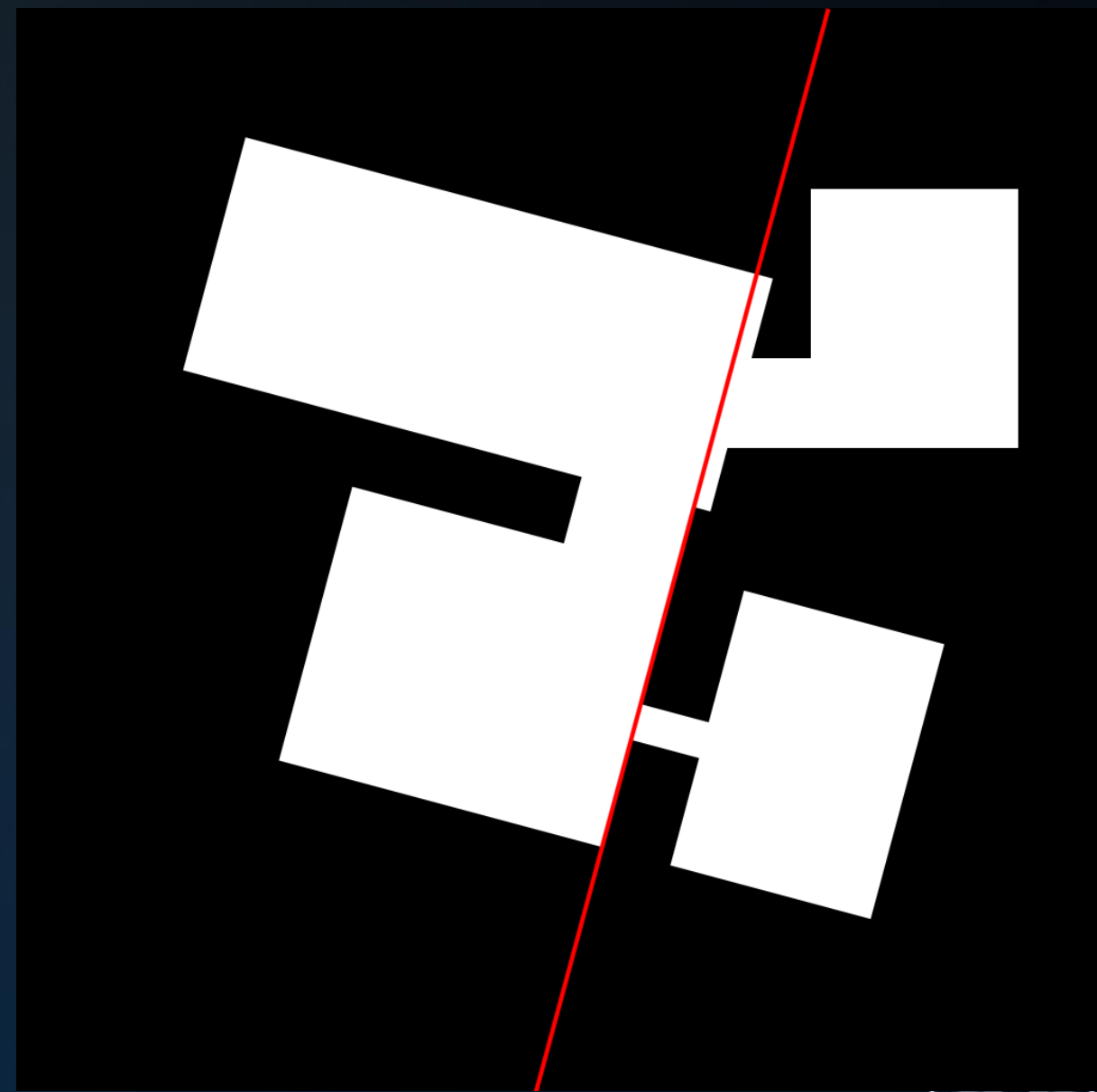
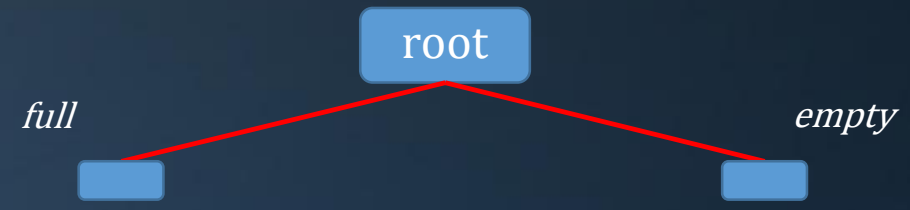
...
    & (depth < MAXDEPTH)
...
    c = inside / 1.0;
    nt = nt / nc; nct = nc;
    cos2t = 1.0f - nnt * nct;
    D, N );
    )
...
    at a = nt - nc; b = nt * nc;
    at Tr = 1 - (R0 + (1 - R0) * c);
    Tr) R = (D * nnt - N * (a *
...
    E * diffuse;
    = true;
...
    refl + refr) && (depth < MAXDEPTH)
...
    D, N );
    -refl * E * diffuse;
    = true;
...
MAXDEPTH)
...
survive = SurvivalProbability( diffuse;
estimation - doing it properly, check
if;
radiance = SampleLight( &rand, I, M, Alignment;
e.x + radiance.y + radiance.z) > 0) && (max
...
y = true;
at b
at3
at w
at c
E *
...
and
yive
...
at3
survi
pdf
n = E * brdf * (dot( N, R ) / pdf);
ion = true;

```



Depth Sorting

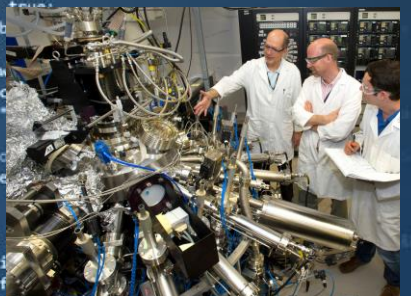
Correct order: BSP



```

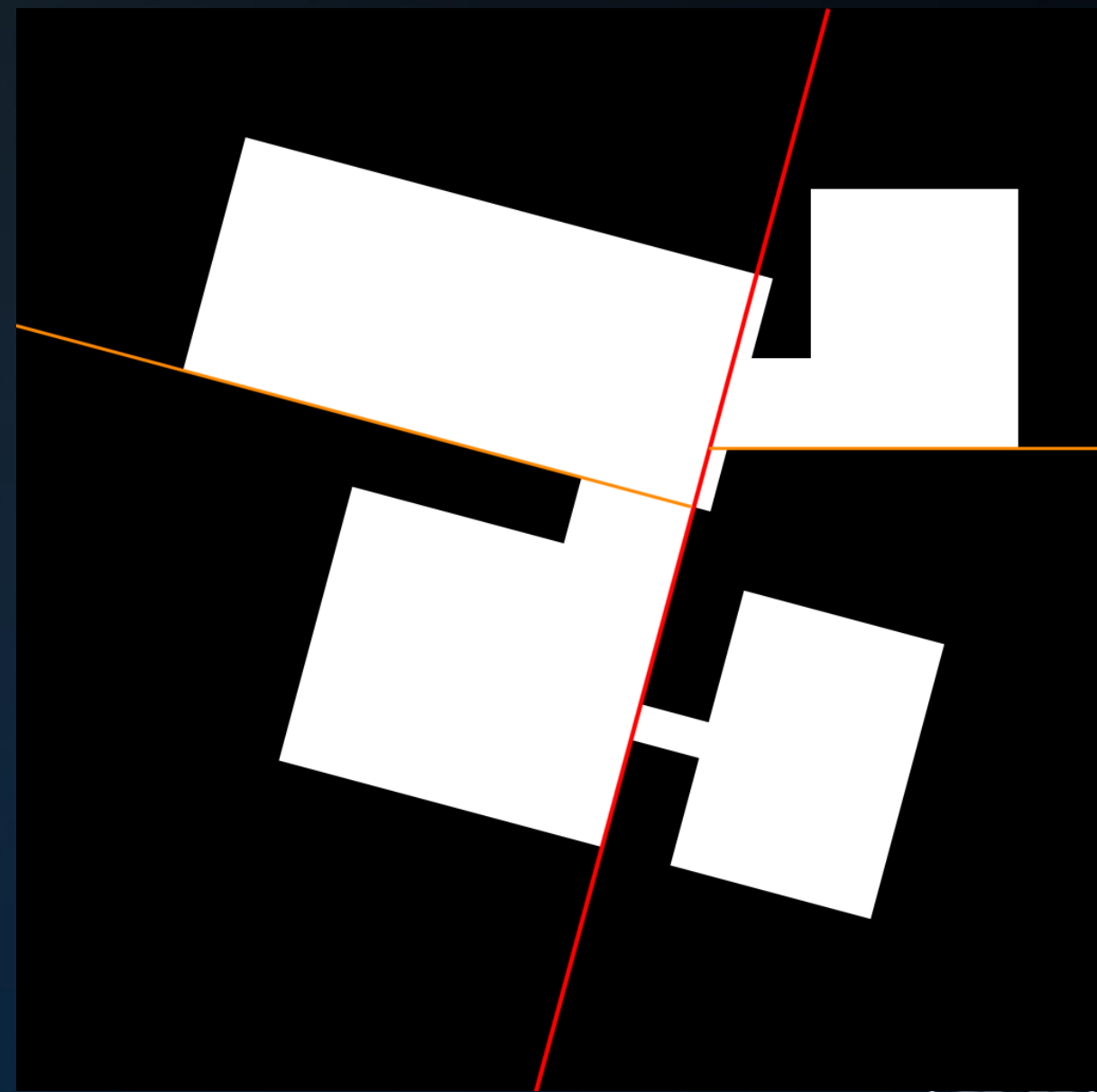
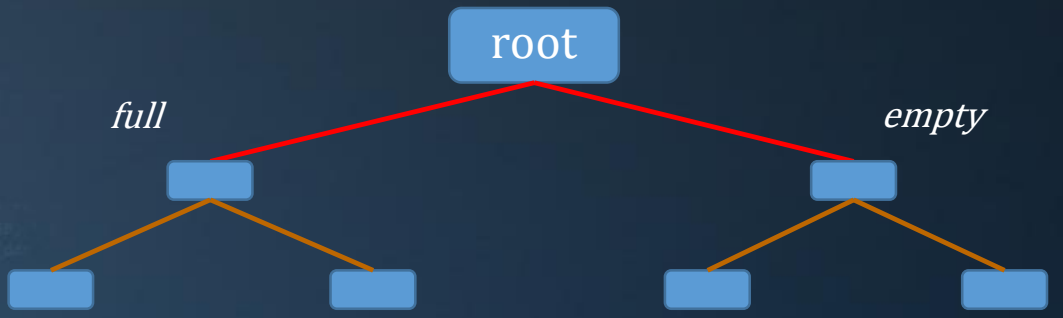
...
    & (depth < MAXDEPTH)
...
    c = inside / 1.0;
    nt = nt / nc; nct = nct / nc;
    cos2t = 1.0f - nnt;
    D, N );
...
    at a = nt - nc; b = nt;
    at Tr = 1 - (R0 + (1 - R0) * c);
    Tr) R = (D * nnt - N * (c *
...
    E * diffuse;
    = true;
...
    refl + refr) && (depth < MAXDEPTH)
...
    D, N );
    refl * E * diffuse;
    = true;
...
MAXDEPTH)
...
survive = SurvivalProbability( diffuse;
estimation - doing it properly, check
if;
radiance = SampleLight( &rand, I, M, Alignment
e.x + radiance.y + radiance.z) > 0) && (max
...
y = true;
at b
at3
at w
at c
E *
...
and
yive
...
at3
survi
pdf
n = E * brdf * (dot( N, R ) / pdf);
ision = true;

```



Depth Sorting

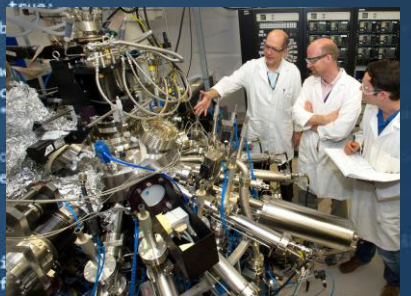
Correct order: BSP



```

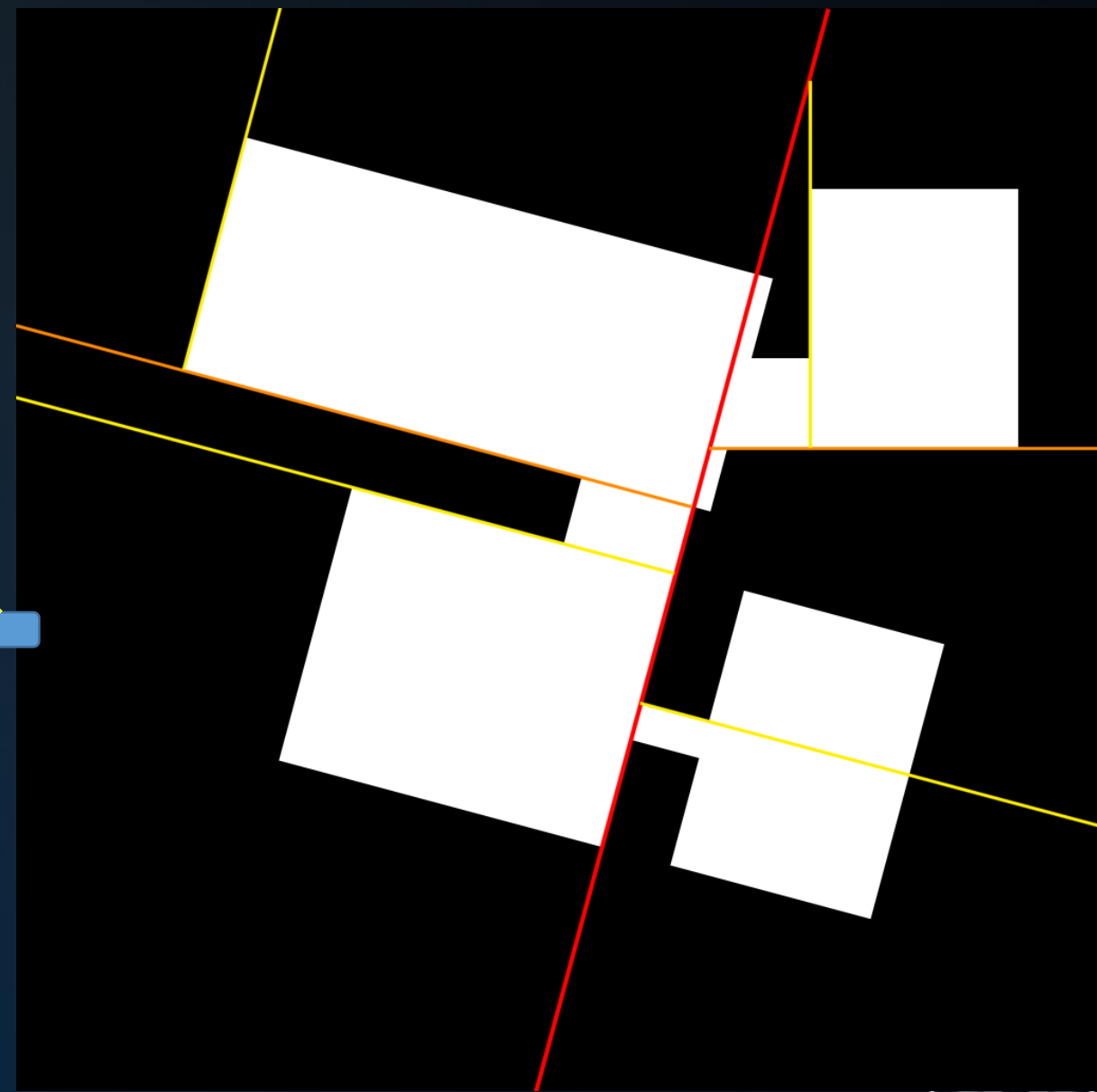
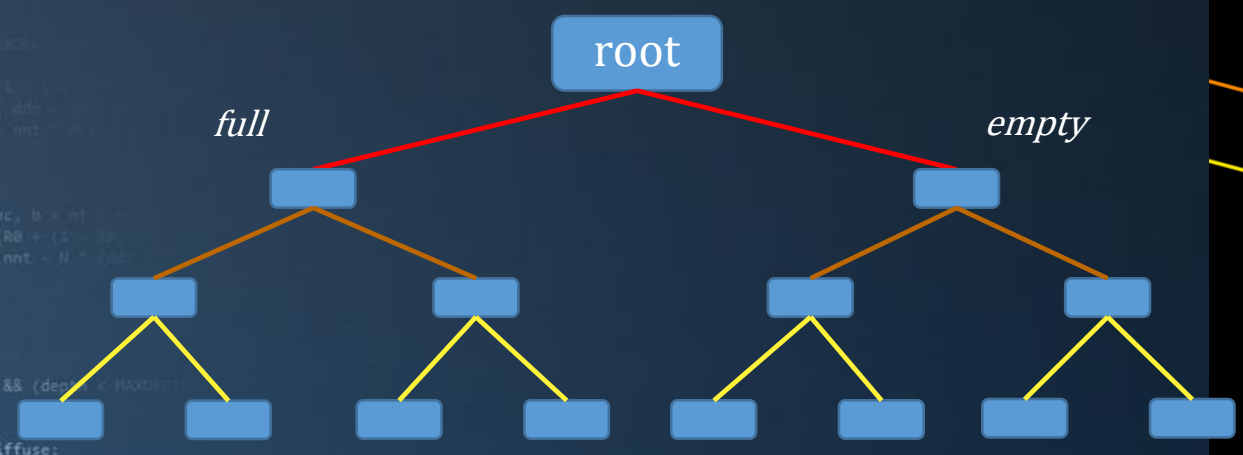
...
    & (depth < MAXDEPTH)
...
    c = inside / 1.0;
    nt = nt / nc;
    nnt = nnt / nc;
    cos2t = 1.0f - nnt;
    D, N );
...
    at a = nt - nc;
    at Tr = 1 - (R0 + (1 - R0) * c);
    Tr) R = (D * nnt - N * (cos2t
...
    E * diffuse;
    = true;
...
    refl + refr) && (depth < MAXDEPTH)
...
    D, N );
    refl * E * diffuse;
    = true;
...
MAXDEPTH)
...
survive = SurvivalProbability( diffuse;
estimation - doing it properly, check
if;
radiance = SampleLight( &rand, I, M, Alignment
e.x + radiance.y + radiance.z) > 0) && (max
...
y = true;
at b
at3
at w
at c
E *
...
and
yive
...
at3
surv
pdf
n = E * brdf * (dot( N, R ) / pdf);
ision = true;

```



Depth Sorting

Correct order: BSP



```

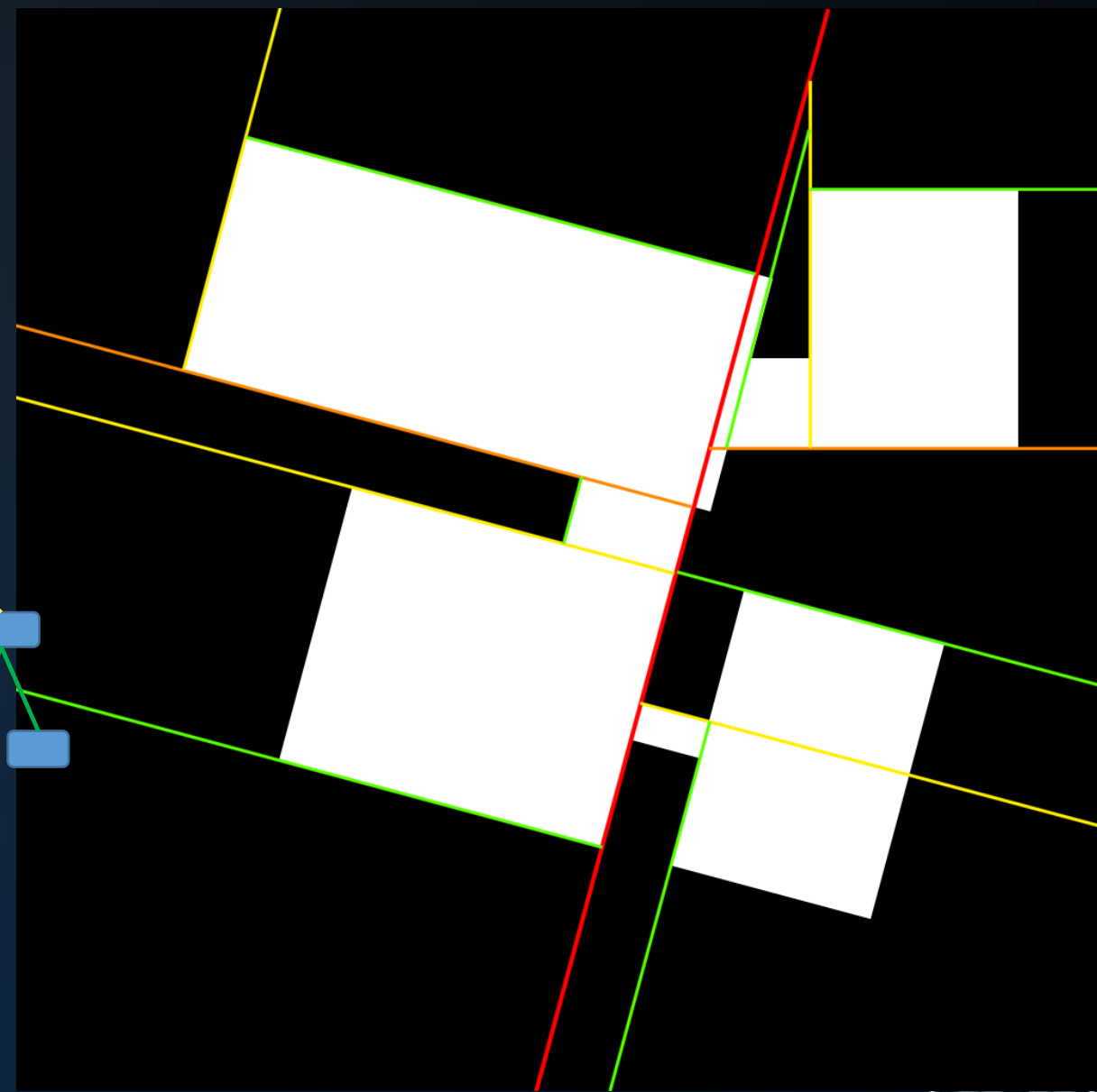
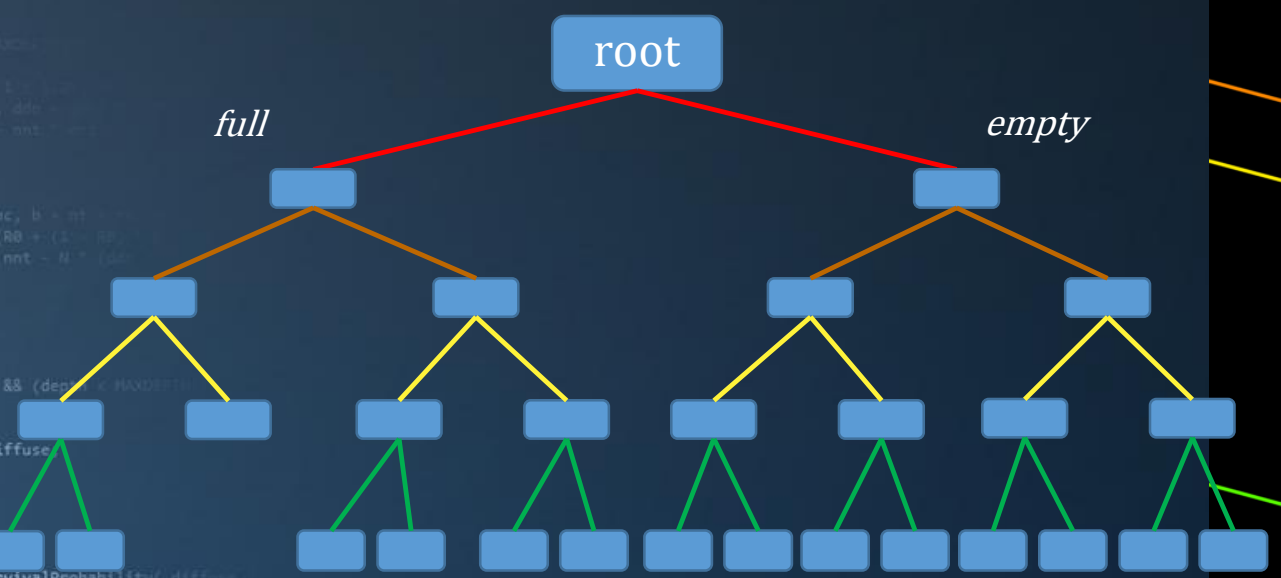
...
    & (depth < MAXDEPTH)
...
    c = inside / (1 + nc);
    nt = nt / nc;
    at = nt / nc;
    pos2t = 1.0f - nnt;
    D, N );
    0);
...
    at a = nt - nc; b = nt;
    at Tr = 1 - (R0 + (1 - R0) * nnt);
    Tr = (D * nnt - N * (1 - R0));
...
    E * diffuse;
    = true;
...
    refl + refr) && (depth < MAXDEPTH)
...
    D, N );
    -refl * E * diffuse;
    = true;
...
MAXDEPTH)
...
survive = SurvivalProbability( diffuse;
estimation - doing it properly, check
if;
radiance = SampleLight( &rand, I, M, Alignment;
e.x + radiance.y + radiance.z) > 0) && (depth <
...
y = true;
at b
at3
at w
at c
E *
...
and
yive
...
at3
surv
pdf
n = E * brdf * (dot( N, R ) / pdf);
ion = true;

```



Depth Sorting

Correct order: BSP



```

...
at (depth < MAXDEPTH)
{
  ...
  if (inside / ...
  {
    nt = nt / nc;
    ...
    pos2t = 1.0f - nnt;
    ...
    D, N );
  }
}

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

E * diffuse;
= true;

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

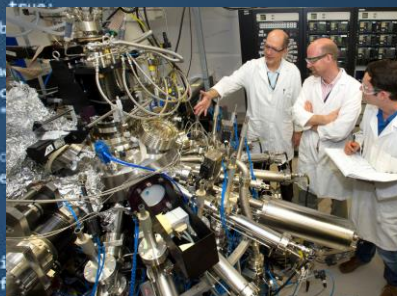
MAXDEPTH)
{
  survive = SurvivalProbability( diffuse;
  estimation - doing it properly, check
  if;
  radiance = SampleLight( &rand, I, M, Alignment;
  e.x + radiance.y + radiance.z) > 0) && (maxDepth
}

y = true;
at b
at3
at w
at c
E *
}

and
yive
}

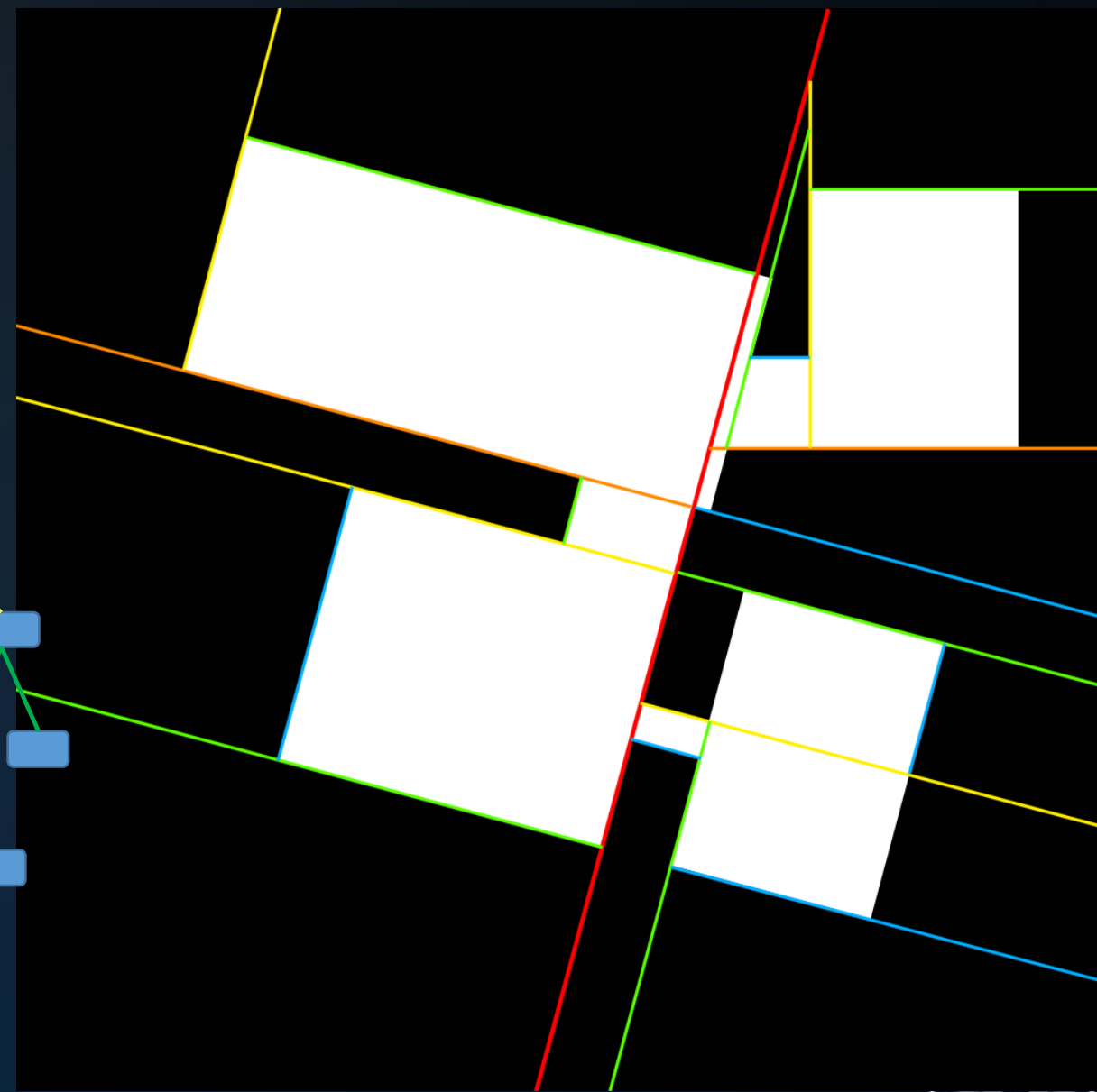
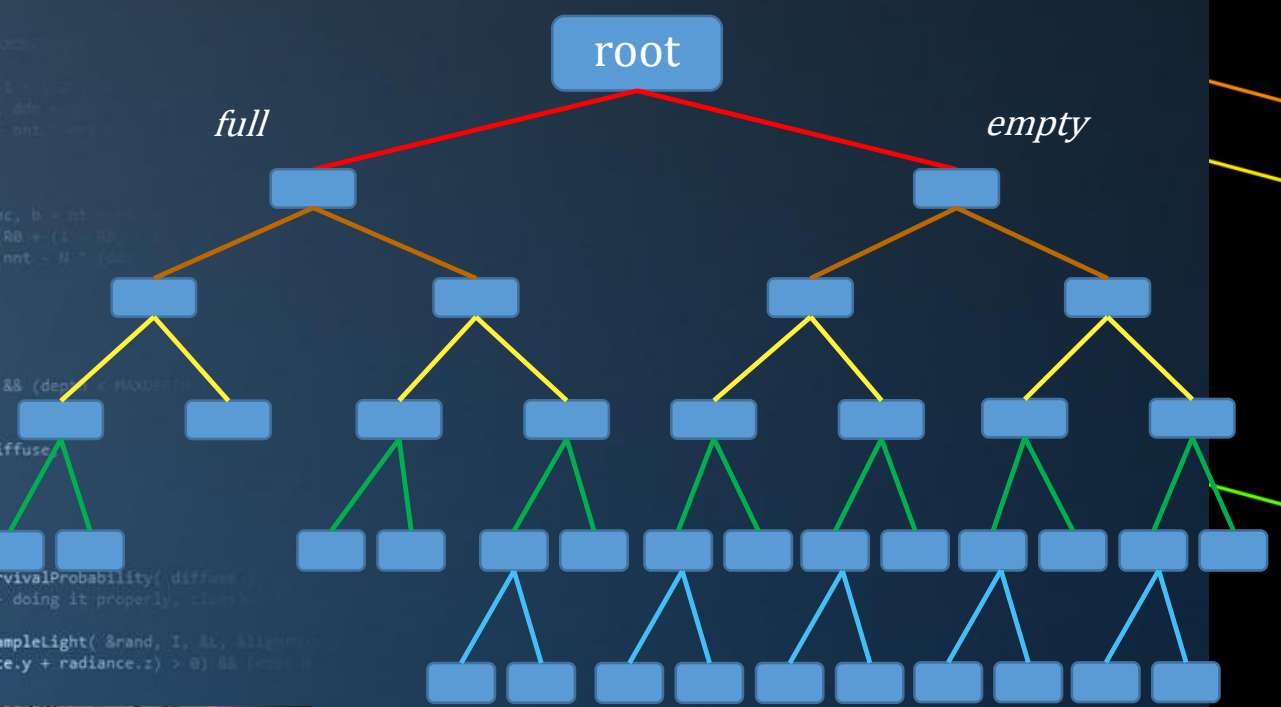
at3
surv
pdf
n = E * brdf * (dot( N, R ) / pdf);
ision = true;

```



Depth Sorting

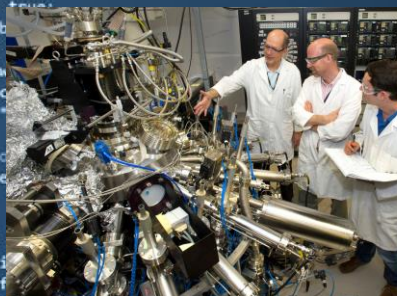
Correct order: BSP



```

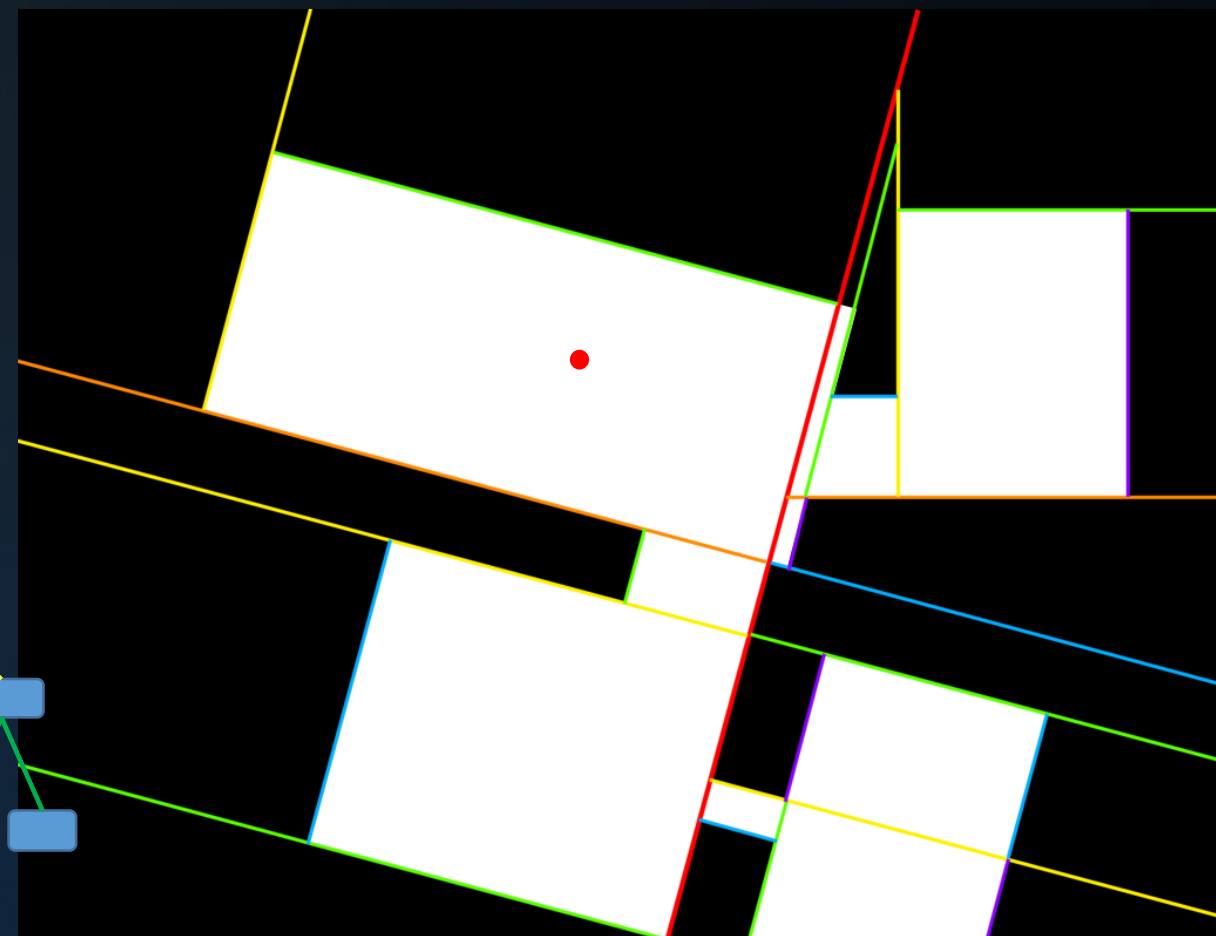
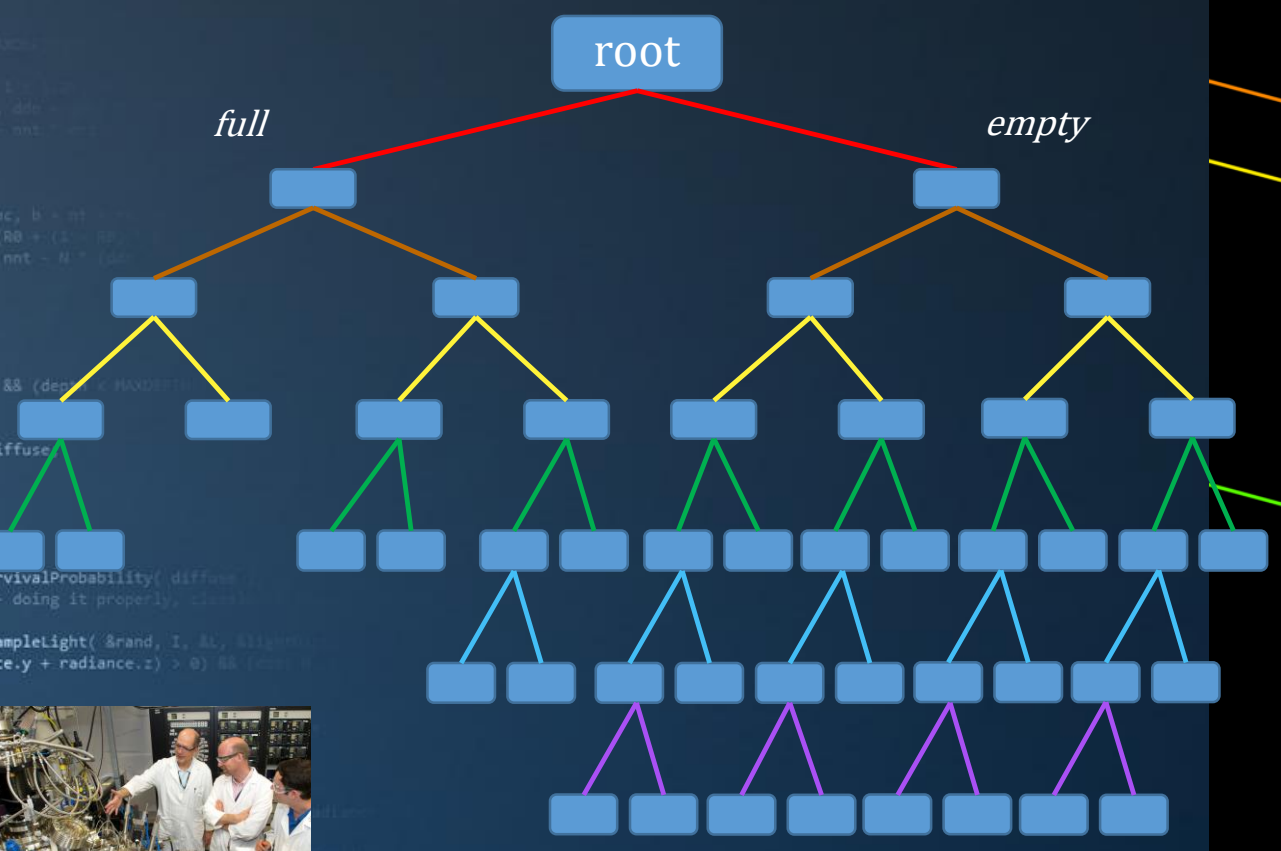
...
at a = nt - nc, b = nt - nc;
at Tr = 1 - (R0 + (1 - R0) * n);
Tr) R = (D * nnt - N * (1 - n));
...
E * diffuse;
= true;
...
efl + refr) && (depth < MAXDEPTH);
...
D, N);
-refl * E * diffuse;
= true;
...
MAXDEPTH);
...
survive = SurvivalProbability( diffuse;
estimation - doing it properly, check);
if;
radiance = SampleLight( &rand, I, M, Alignment);
e.x + radiance.y + radiance.z) > 0) && (survive);
...
y = true;
at b;
at3;
at w;
at c;
E *
...
and;
yive;
...
at3;
surv;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
ision = true;

```



Depth Sorting

Correct order: BSP



Sorting by BSP traversal:
 Recursively
 1. Render far side of plane
 2. Render near side of plane

```

...
at a = nt - nc, b = nt - nc;
at Tr = 1 - (R0 + (1 - R0) * t);
Tr) R = (D * nnt - N * t);
...
E * diffuse;
= true;
...
efl + refr) && (depth < MAXDEPTH);
...
D, N);
-refl * E * diffuse;
= true;
...
MAXDEPTH);
...
survive = SurvivalProbability( diffuse;
estimation - doing it properly, check);
if;
radiance = SampleLight( @rand, I, Rt, Alignment;
e.x + radiance.y + radiance.z) > 0) && (survive);
...
y = true;
at b;
at3;
at w;
at c;
E *
...
and;
vive;
...
at3;
urv;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
ision = true;
...

```



Depth Sorting

Draw order using a BSP:

- Guaranteed to be correct (hard cases result in polygon splits)
- No sorting required, just a tree traversal

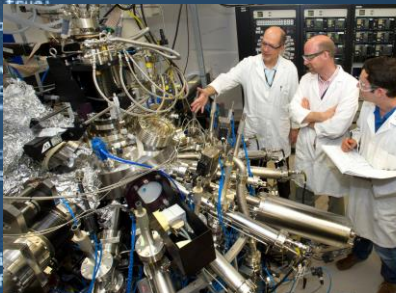
But:

- Requires construction of BSP: not suitable for dynamic objects
- Does not eliminate overdraw

```

...
    & (depth < MAXDEPTH)
...
    t = inside / (1.0 - n * nc);
    nt = nt / nc;
    cos2t = 1.0f - nnt;
    D, N );
    )
...
    at a = nt - nc, b = nt;
    at Tr = 1 - (R0 + (1 - R0) * 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, check
if;
radiance = SampleLight( @rand, I, M, Alignment;
e.x + radiance.y + radiance.z) > 0) && (survive
...
y = true;
at b
at3
at w
at c
E *
...
and
ive
...
at3
surv
pdf
n = E * brdf * (dot( N, R ) / pdf);
ision = true;

```



Depth Sorting

Z-buffer

A z-buffer stores, per screen pixel, a depth value.

The depth of each fragment is checked against this value:

- If the fragment is further away, it is discarded
- Otherwise, it is drawn, and the z-buffer is updated.

The z-buffer requires:

- An additional buffer
- Initialization of the buffer to Z_{max}
- Interpolation of z over the triangle
- A z-buffer read and compare, and possibly a write.





Depth Sorting

Z-buffer

What is the best representation for depth in a z-buffer?

1. Interpolated z (convenient, intuitive);
2. $1/z$ (or: $n + f - \frac{fn}{z}$) (more accurate nearby);
3. $(\text{int})((2^{31}-1)/z)$;
4. $(\text{uint})((2^{32}-1)/-z)$;
5. $(\text{uint})((2^{32}-1)/(-z - 1))$.



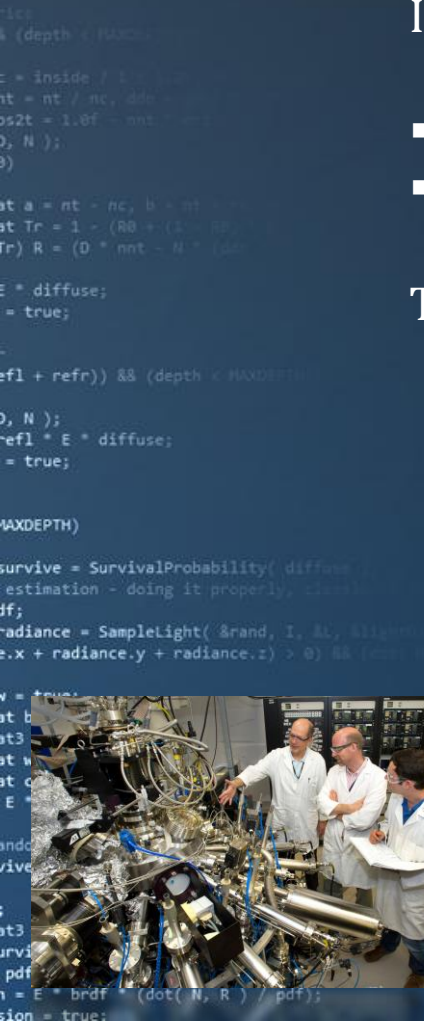
Depth Sorting

Z-buffer optimization

In the ideal case, the nearest fragment for a pixel is drawn first:

- This causes all subsequent fragments for the pixel to be discarded;
- This minimizes the number of writes to the frame buffer and z-buffer.

The ideal case can be approached by using Painter’s to ‘pre-sort’.

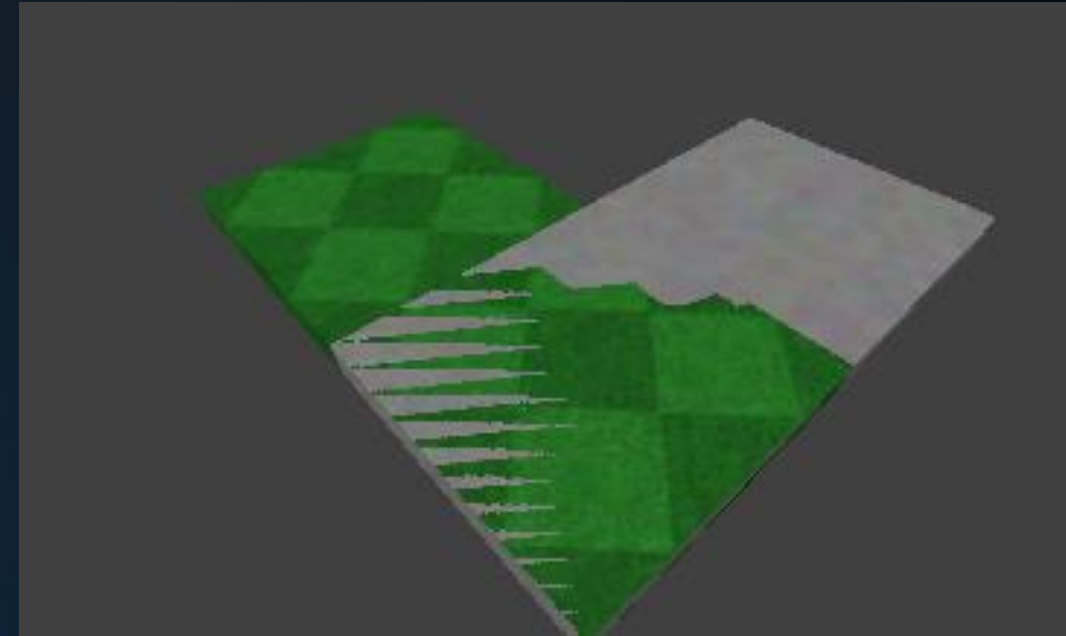


Depth Sorting

‘Z-fighting’:

Occurs when two polygons have almost identical z-values.

Floating point inaccuracies during interpolation will cause unpleasant patterns in the image.



```

...
    & (depth < MAXDEPTH)
...
    t = inside / 2.0;
    nt = nt / nc;
    cos2t = 1.0f - nnt;
    D, N );
    )
...
    at a = nt - nc, b = nt;
    at Tr = 1 - (R0 + (1 - R0)
    Tr) R = (D * nnt - N * (a
...
    E * diffuse;
    = true;
...
    refl + refr) && (depth < MAXDEPTH)
...
    D, N );
    refl * E * diffuse;
    = true;
...
MAXDEPTH)
...
survive = SurvivalProbability( diffuse
estimation - doing it properly, check
if;
radiance = SampleLight( @rand, I, M, Alignment
e.x + radiance.y + radiance.z) > 0) && (max
...
y = true;
at b
at3
at w
at c
at E
...
and
ive
...
at3
surv
pdf
n = E * brdf * (dot( N, R ) / pdf);
ion = true;

```



Part of the tree is off-screen

Stuff that is too far to draw

Tree requires little detail

City obscured by tree

Torso closer than ground

Tree between ground & sun




```
...ics
& (depth < MAXDEPTH)
...
t = inside / (1.0 - refl);
nt = nt / nc; rdd = ...
...
os2t = 1.0f - nnt; ...
D, N );
...
)
...
at a = nt - nc; b = nt; ...
at Tr = 1 - (R0 + (1 - R0) * ...
Tr) R = (D * nnt - N * (dd
...
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, M, &light, ...
e.x + radiance.y + radiance.z) > 0) && (depth <
v = true;
at brdfPdf = EvaluateDiffuse( L, N ) * Survive;
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
ive)
;
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf;
urvive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;
```

Today's Agenda:

- Depth Sorting
- Clipping
- Visibility
- The Midterm Exam



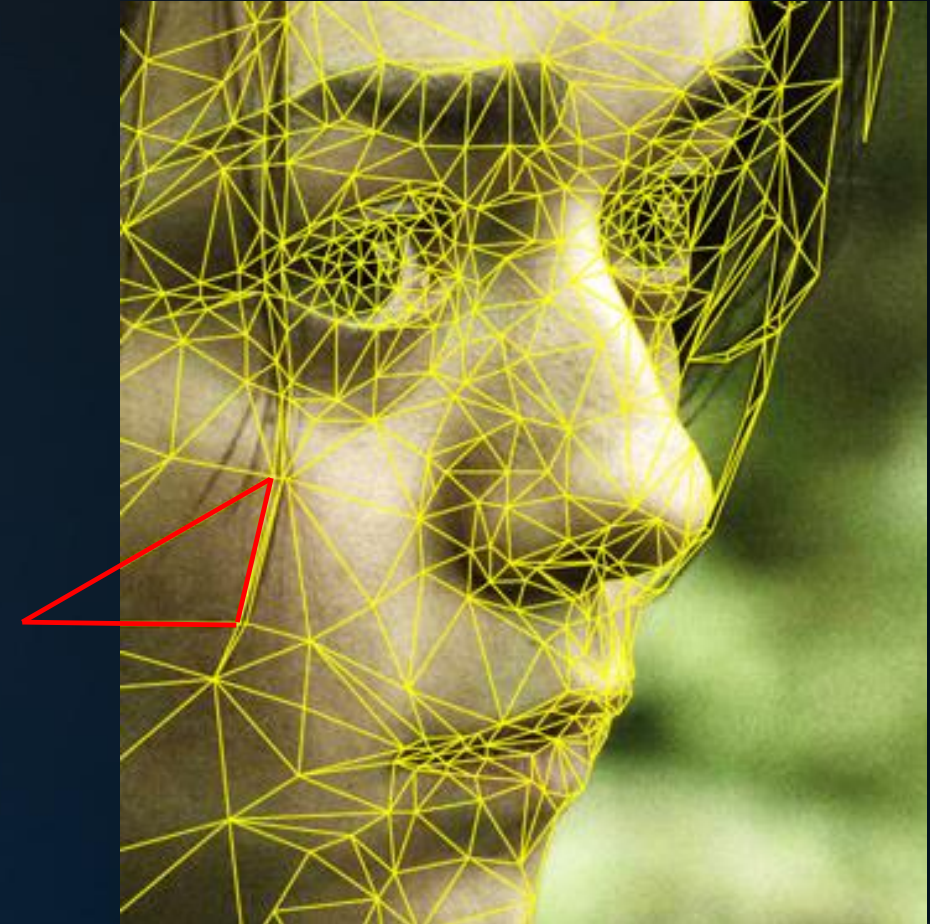
Clipping

Clipping

Many triangles are partially off-screen. This is handled by *clipping* them.

Sutherland-Hodgeman clipping:

Clip triangle against 1 plane at a time;
Emit n-gon (0, 3 or 4 vertices).



Clipping

Sutherland-Hodgeman

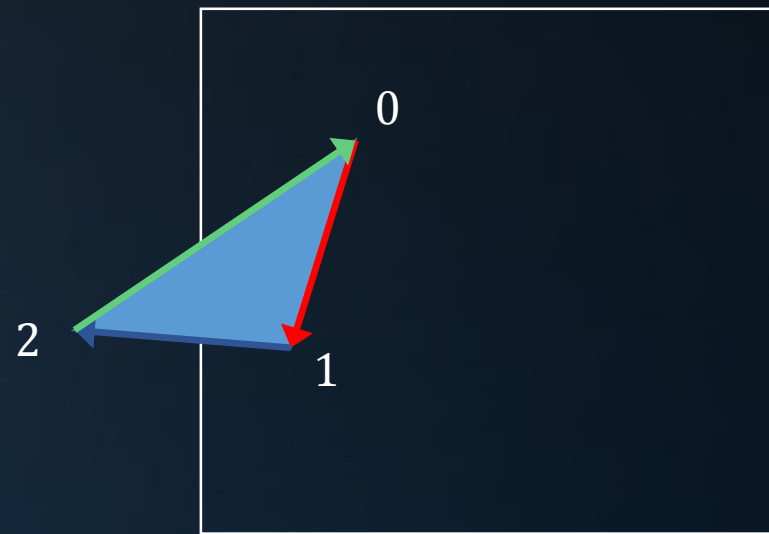
Input: list of vertices

Algorithm:

Per edge with vertices v_0 and v_1 :

- If v_0 and v_1 are ‘in’, emit v_1
- If v_0 is ‘in’, but v_1 is ‘out’, emit C
- If v_0 is ‘out’, but v_1 is ‘in’, emit C and v_1

where C is the intersection point of the edge and the plane.



Output: list of vertices, defining a convex n-gon.

in	out
Vertex 0	Vertex 1
Vertex 1	Intersection 1
Vertex 2	Intersection 2
	Vertex 0



Clipping

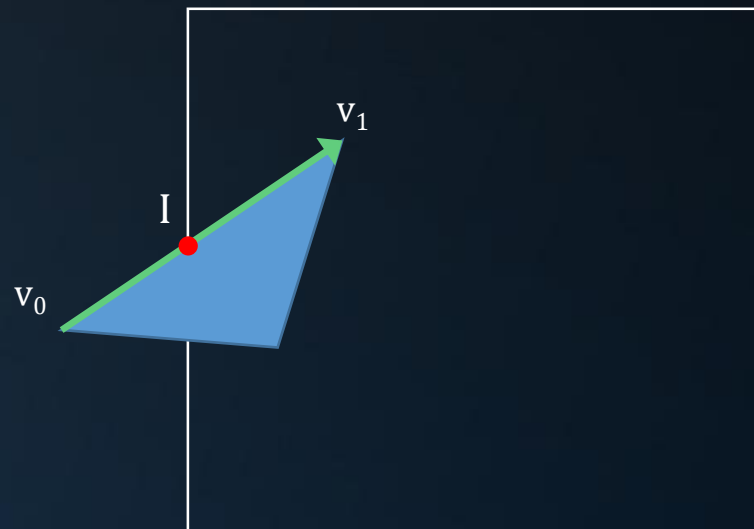
Sutherland-Hodgeman

Calculating the intersections with plane $ax + by + cz + d = 0$:

$$dist_v = v \cdot \begin{pmatrix} a \\ b \\ c \end{pmatrix} + d$$

$$f = \frac{|dist_{v_0}|}{|dist_{v_0}| + |dist_{v_1}|}$$

$$I = v_0 + f(v_1 - v_0)$$



After clipping, the input n-gon may have at most 1 extra vertex. We may have to triangulate it:

0,1,2,3,4 \rightarrow 0, 1, 2 + 0, 2, 3 + 0, 3, 4.

```

...
    & (depth < MAXDEPTH)
...
    inside = inside / 1.0f;
    nt = nt / nc;
    cos2t = 1.0f - nt;
    D, N );
...
    at a = nt - nc, b = nt;
    at Tr = 1 - (R0 + (1 - R0) * inside);
    Tr) R = (D * nnt - N * (a *
...
    E * diffuse;
    = true;
...
    refl + refr) && (depth < MAXDEPTH)
...
    D, N );
    refl * E * diffuse;
    = true;
...
MAXDEPTH)
survive = SurvivalProbability( diffuse
estimation - doing it properly, check
if;
radiance = SampleLight( &rand, I, N, Alignment
e.x + radiance.y + radiance.z) > 0) && (rand
v = true;
at brdfPdf = EvaluateDiffuse( L, N ) * Pearline
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 death
ive)
;
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;

```

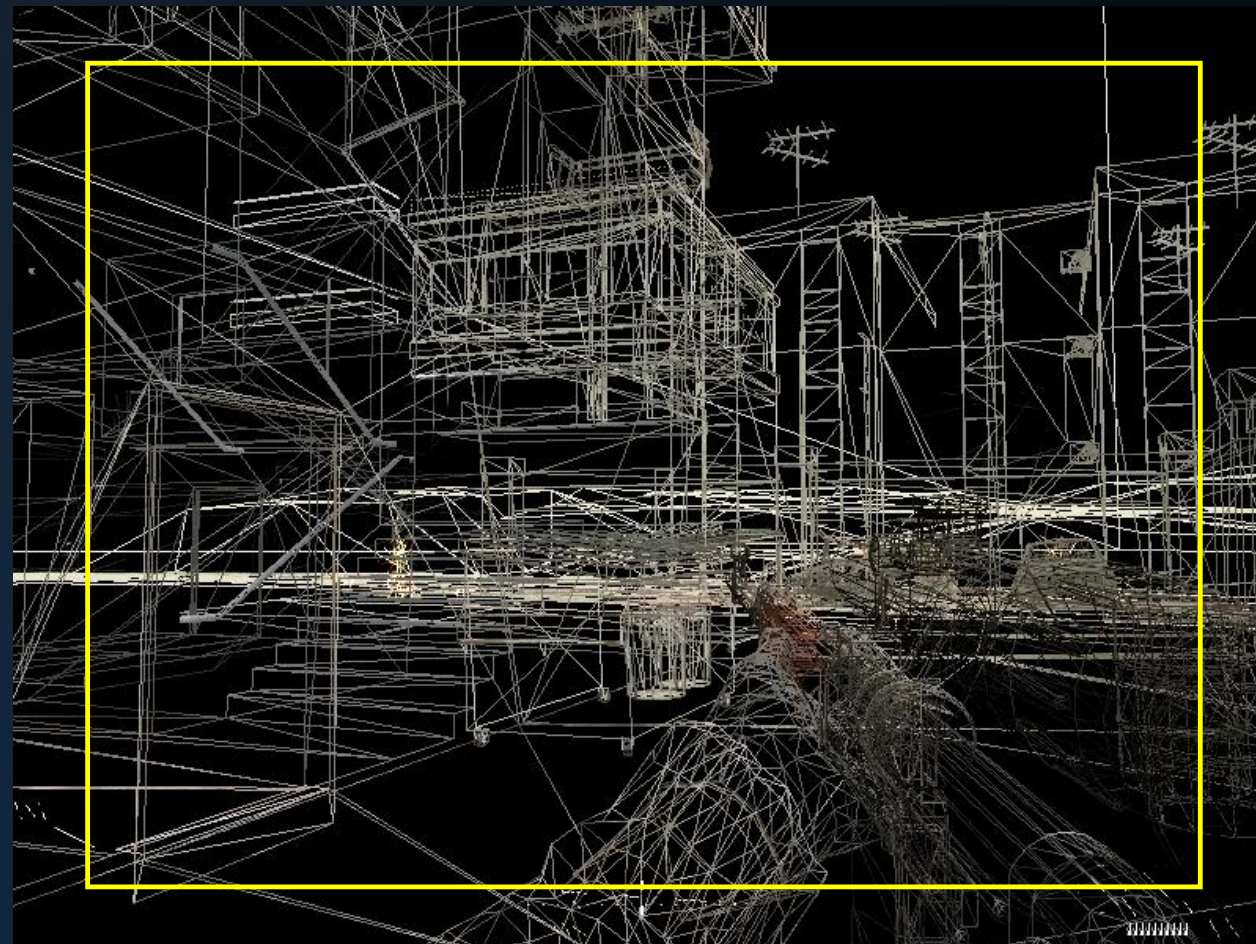


Clipping

Guard bands

To reduce the number of polygons that need clipping, some hardware uses *guard bands*: an invisible band of pixels outside the screen.

- Polygons outside the screen are discarded, even if they touch the guard band;
- Polygons partially inside, partially in the guard band are drawn without clipping;
- Polygons partially inside the screen, partially outside the guard band are clipped.



Clipping

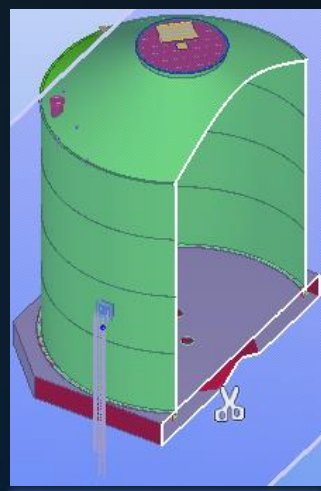
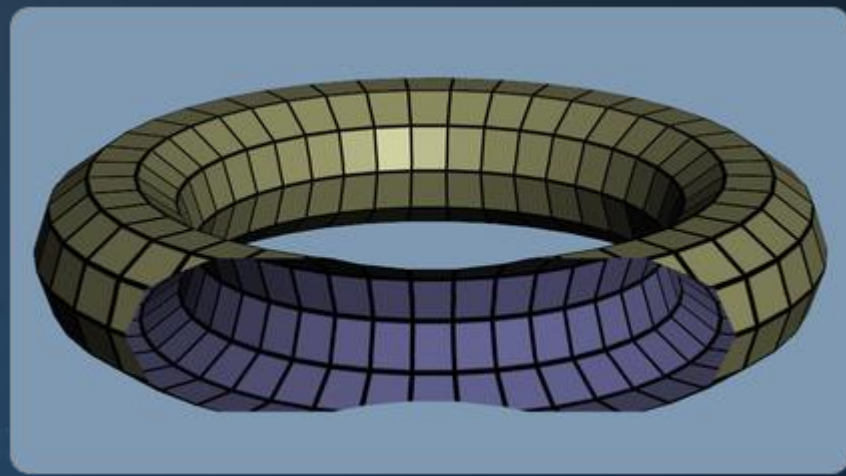
Sutherland-Hodgeman

Clipping can be done against arbitrary planes.

```

...
    & (depth < MAXDEPTH)
...
    t = inside / (inside - outside);
    nt = nt / nc;
    cos2t = 1.0f - nnt;
    D, N );
    )
...
    at a = nt - nc, b = nt;
    at Tr = 1 - (RB + (1 - RB) * t);
    Tr) R = (D * nnt - N * (a *
...
    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, M, Align
e.x + radiance.y + radiance.z) > 0) && (survive)
...
v = true;
at brdfPdf = EvaluateDiffuse( L, N ) * Pearls;
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
ive)
...
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf);
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;

```



```
...ics
& (depth < MAXDEPTH)
...
t = inside / (1.0 - refl);
nt = nt / nc; rdd = ...
...
os2t = 1.0f - nnt; ...
D, N );
...
)
...
at a = nt - nc; b = nt; ...
at Tr = 1 - (R0 + (1 - R0) * ...
Tr) R = (D * nnt - N * (dd
...
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, M, &light, ...
e.x + radiance.y + radiance.z) > 0) && (depth <
v = true;
at brdfPdf = EvaluateDiffuse( L, N ) * Survive;
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
ive)
;
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf, ...
urvive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;
```

Today's Agenda:

- Depth Sorting
- Clipping
- Visibility
- The Midterm Exam



Part of the tree is off-screen

Stuff that is too far to draw

Tree requires little detail

City obscured by tree

Torso closer than ground

Tree between ground & sun







Visibility

Only rendering what’s visible:

“Performance should be determined by visible geometry, not overall world size.”

- Do not render geometry outside the view frustum
- Better: do not *process* geometry outside frustum
- Do not render occluded geometry
- Do not render anything more detailed than strictly necessary



Visibility

Culling

Observation:

50% of the faces of a cube are not visible.

On average, this is true for all meshes.

Culling ‘backfaces’:

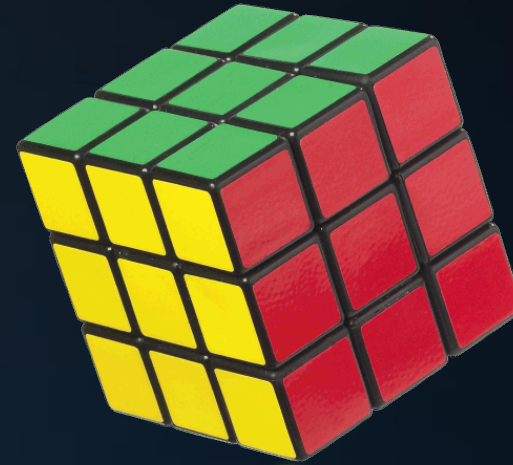
Triangle: $ax + by + cz + d = 0$

Camera: (x, y, z)

Visible: fill in camera position in plane equation.

$ax + by + cz + d > 0$: *visible*.

Cost: 1 dot product per triangle.



Visibility

Culling

Observation:

If the *bounding sphere* of a mesh is outside the view frustum, the mesh is not visible.

But also:

If the *bounding sphere* of a mesh intersects the view frustum, the mesh may be not visible.

View frustum culling is typically a *conservative test*: we sacrifice accuracy for efficiency.

Cost: 1 dot product per mesh.



Visibility

Culling

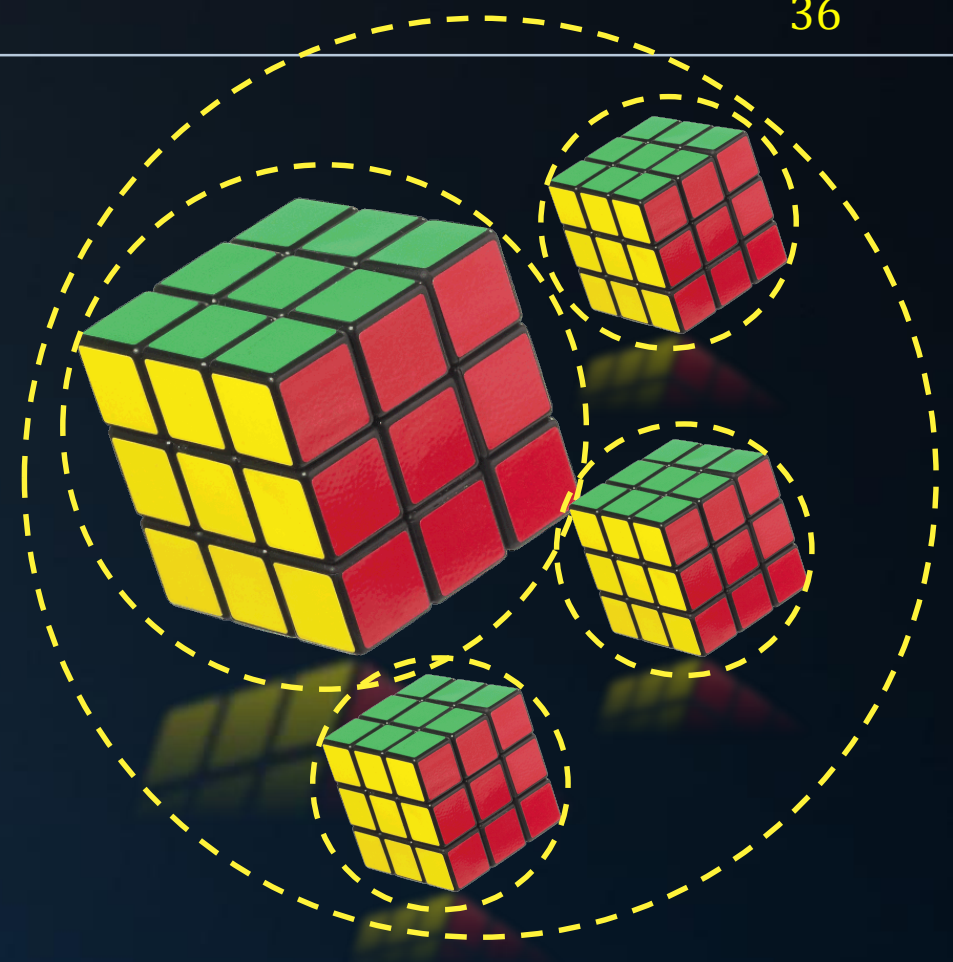
Observation:

If the *bounding sphere* over a group of bounding spheres is outside the view frustum, a group of meshes is invisible.

We can store a bounding volume hierarchy in the scene graph:

- Leaf nodes store the bounds of the meshes they represent;
- Interior nodes store the bounds over their child nodes.

Cost: 1 dot product per scene graph subtree.



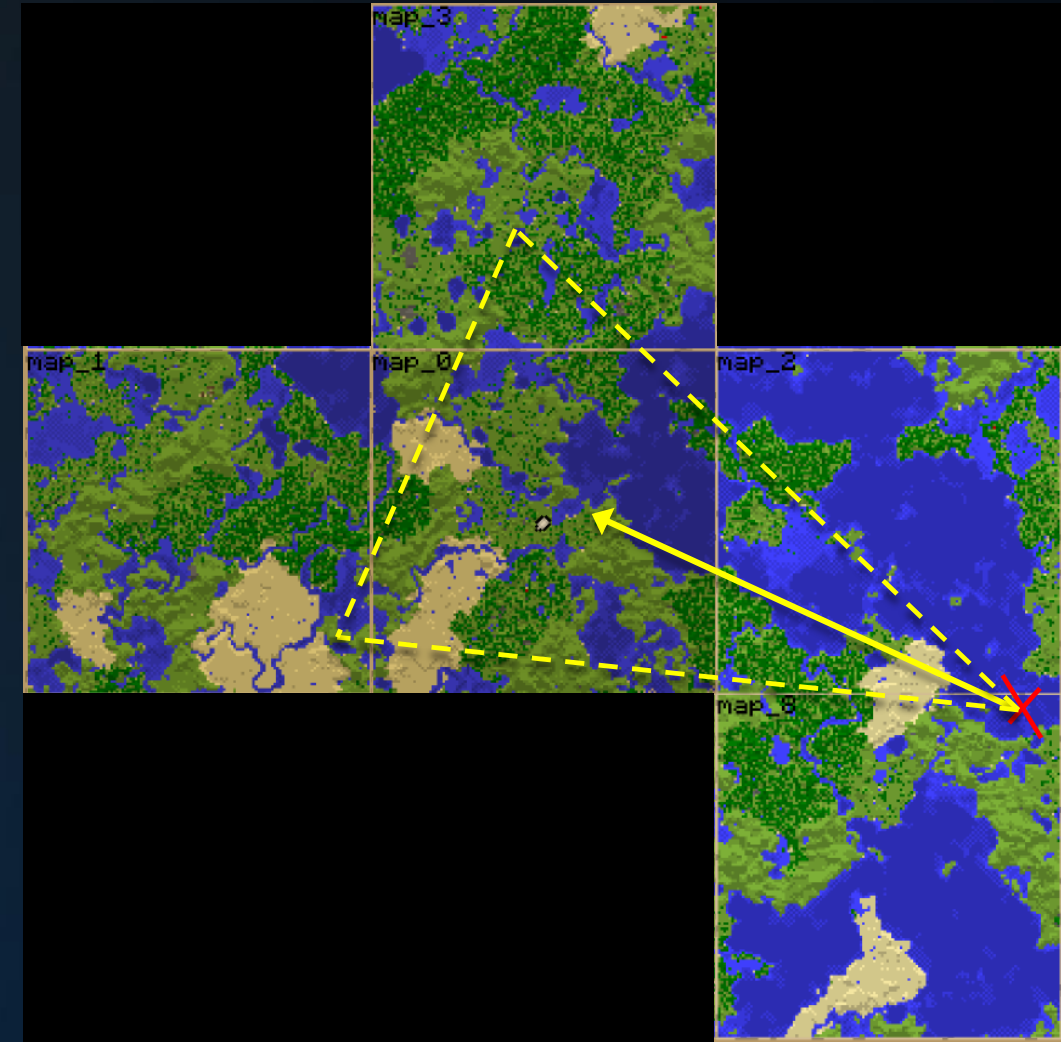
Visibility

Culling

Observation:

If a grid cell is outside the view frustum, the contents of that grid cell are not visible.

Cost: 0 for out-of-range grid cells.



Visibility

Indoor visibility: Portals

Observation: if a window is invisible, the room it links to is invisible.

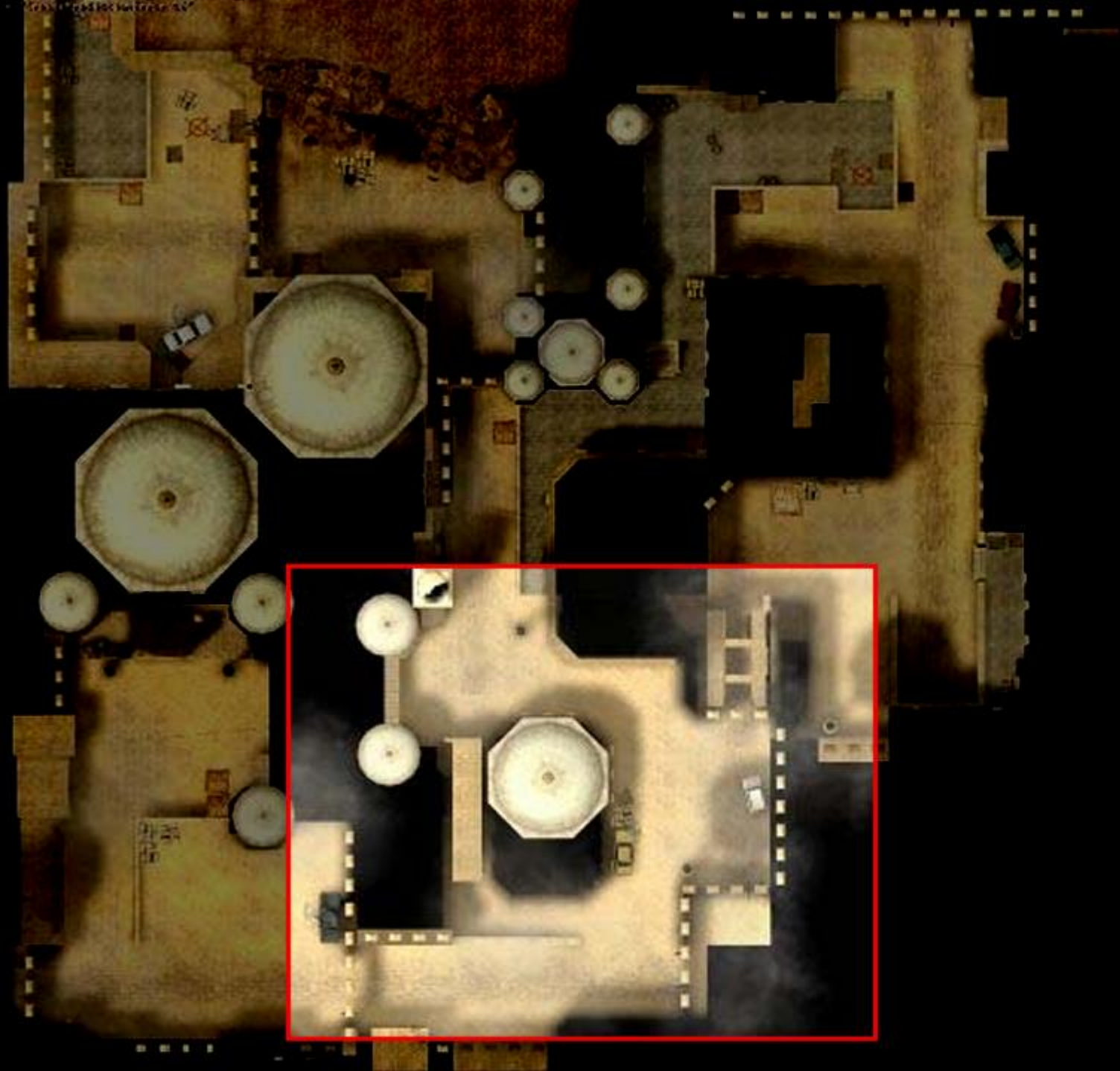
```

...
    & (depth < MAXDEPTH)
...
    t = inside / (nc + nd);
    nt = nt / nc;
    nd = nd / nc;
    cos2t = 1.0f - nnt;
    D, N );
    )
...
    at a = nt - nc, b = nt - nd;
    at Tr = 1 - (RB + (1 - RB) * a);
    Tr) R = (D * nnt - N * (a *
...
    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, M, Alignment
e.x + radiance.y + radiance.z) > 0) && (max
...
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 death
ive)
...
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;

```

















1

2

3

4





Visibility

Visibility determination

Coarse:

- Grid-based (typically outdoor)
- Portals (typically indoor)

Finer:

- Frustum culling
- Occlusion culling

Finest:

- Backface culling
- Clipping
- Z-buffer




```
...ics
& (depth < MAXDEPTH)
...
t = inside / (1.0 - refl);
nt = nt / nc; rdd = rdd / nc;
cos2t = 1.0f - nnt; rnt = sqrt(cos2t);
D, N );
)
...
at a = nt - nc, b = nt + nc;
at Tr = 1 - (RB + (1 - RB) * a);
Tr) R = (D * nnt - N * (a *
...
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, M, &light,
e.x + radiance.y + radiance.z) > 0) && (depth <
v = true;
at brdfPdf = EvaluateDiffuse( L, N ) * Survive;
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
ive)
;
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf);
urvive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;
```

Today's Agenda:

- Depth Sorting
- Clipping
- Visibility
- The Midterm Exam



Midterm Exam

Time for your examination.

Where: EDUC-GAMMA

When: Thursday, May 21st, 2015, at 13.30

Duration: Two hours, three for dyslexia

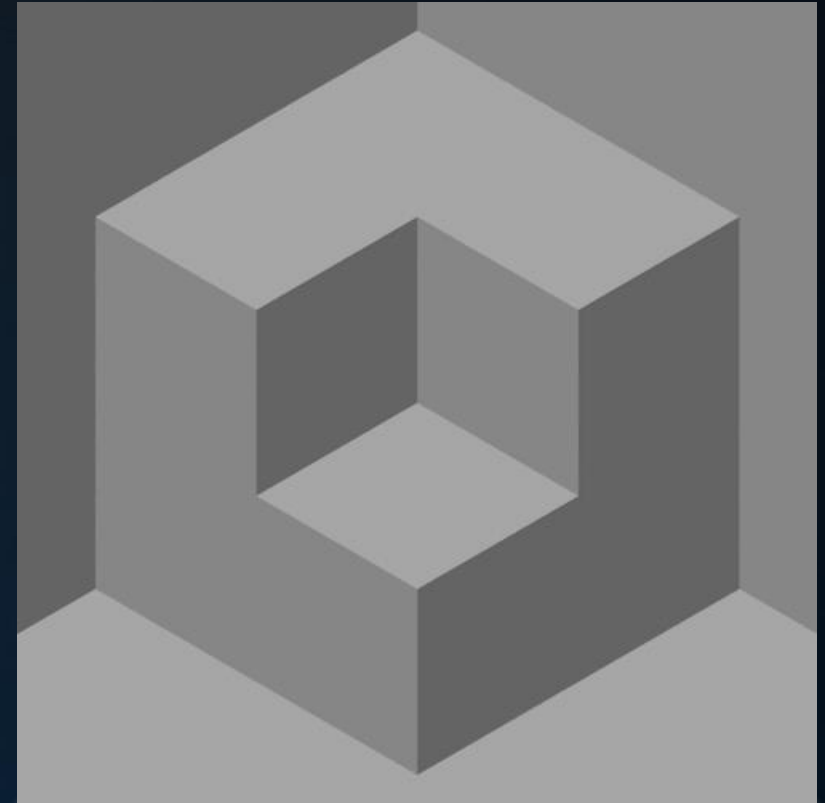
Contents:

- Mathematics (lectures 1..5 + slides, tutorial sheets)
- Graphics theory (lectures 1..7 + slides)

What to study:

- Slides
- Book can be helpful too
- Tutorial sheets

Need extra time? Entitled to it? Notify me!





**WAARSCHUWING
BESMETTE ZONE**



**GASMASKER
VERPLICHT**



BIOHAZARD

OPLETING, ONTOEGANG EN VERBOD
OP TOEGANG EN VERBOD OP
ONTOEGANG EN VERBOD OP
TOEGANG EN VERBOD OP



RADIOACTIE

**RADIOACTIEVE
BESMET**



**WAARSCHUWING
BESMETTE ZONE**