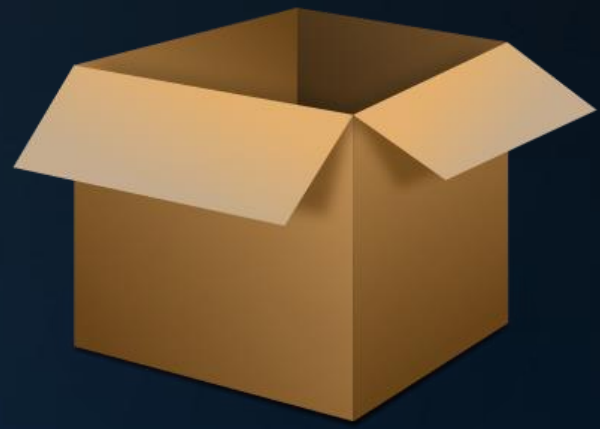
INFOGR – Computer Graphics

Jacco Bikker - April-July 2016 - Lecture 6: “Boxes”

Welcome!

Today’s Agenda:

- Introduction
- Boxes
- AABBs
- Groupings
- Efficiency
- To Rasterization



Introduction

Finalizing the Ray Tracer

… and slowly moving to rasterization:

- Generic scenes: intersecting triangles
- More speed
- Application responsiveness
- Boxes

Introduction

Intersecting a Triangle

Many ways to intersect a triangle…

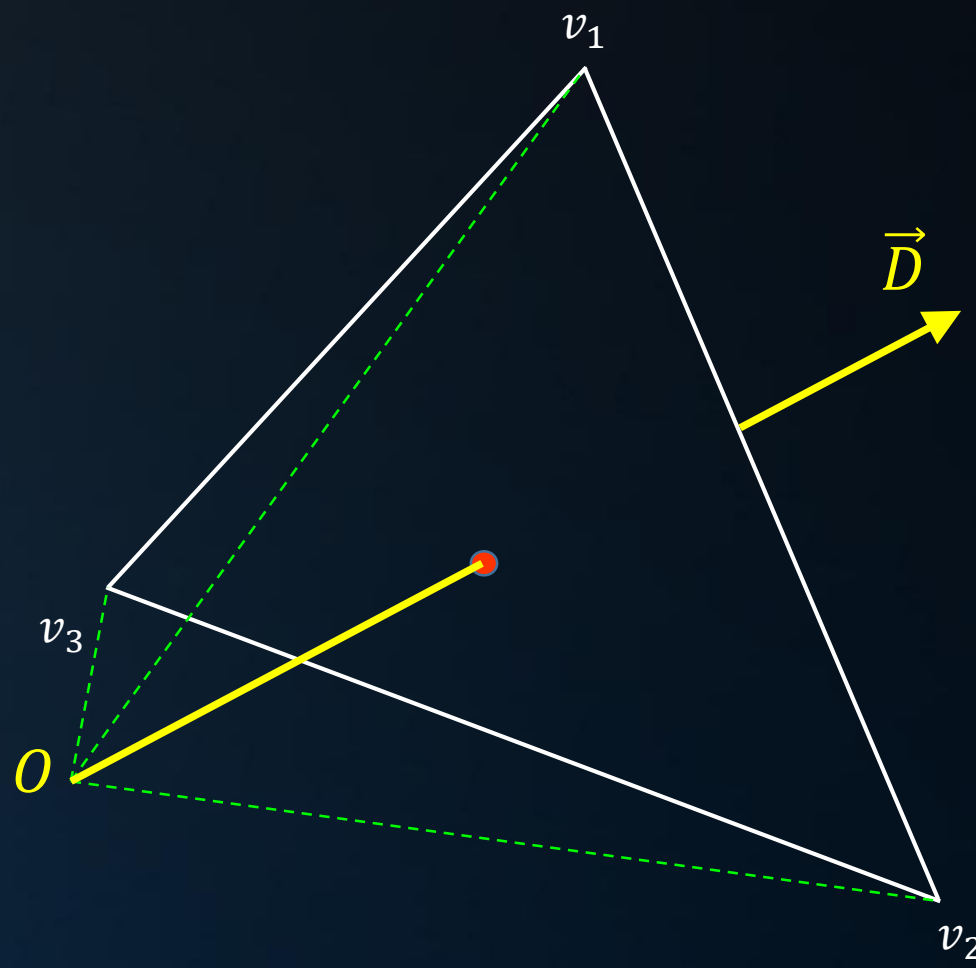
Start with the plane:

$$\vec{N} = normalize\big((v_2 - v_1) \times (v_3 - v_1)\big)$$
$$d = -(\vec{N} \cdot v_1)$$

Calculate the intersection of the ray and the plane:

$$t = -(O \cdot \vec{N} + d)/(\vec{D} \cdot \vec{N})$$
$$P = O + t\vec{D}$$

And finally, see if point P is on the same side of the three planes between the edges and the origin.



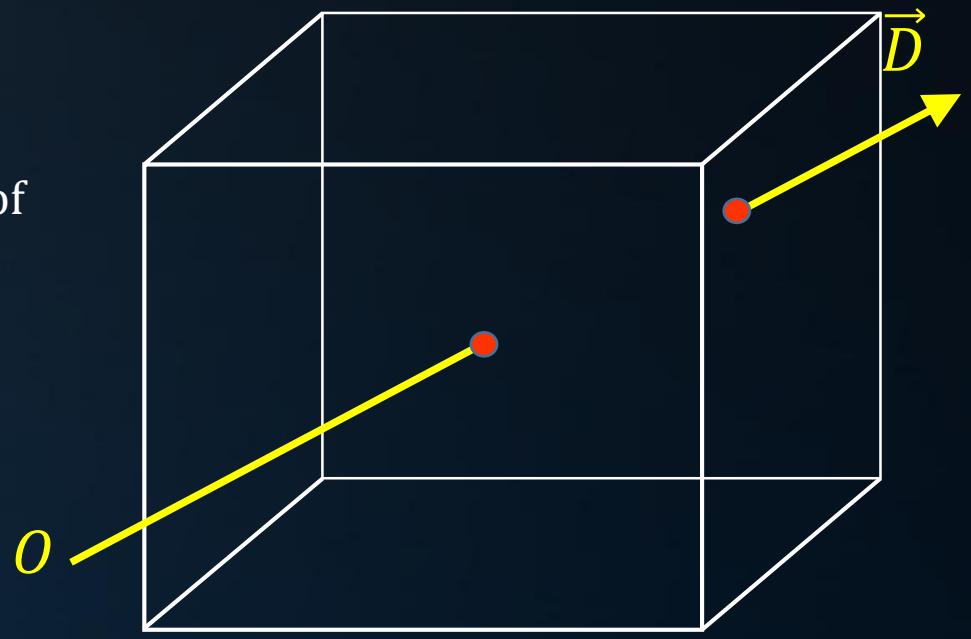
For a more efficient algorithm, see:
Fast, Minimum Storage Ray/Triangle Intersection, Möller & Trumbore. Journal of Graphics Tools, 1997.

Introduction

Intersecting a Box

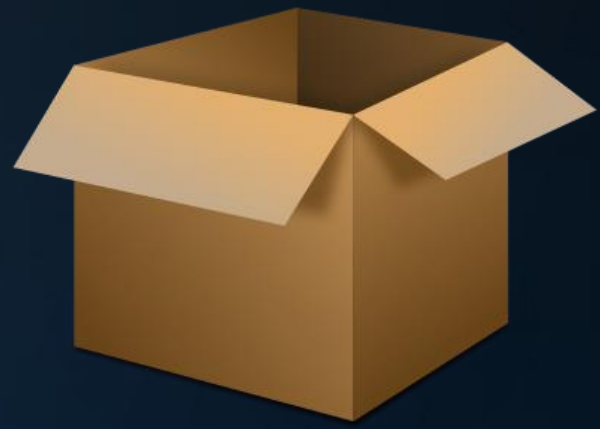
Basic ray/box intersection:

1. Intersect the ray with each of the 6 planes;
2. Keep the intersections that are on the same side of the remaining planes;
3. Determine the closest intersection point.



Today’s Agenda:

- Introduction
- Boxes
- AABBs
- Groupings
- Efficiency
- To Rasterization



Boxes

Special Case: AABB

AABB: *Axis Aligned Bounding Box*.

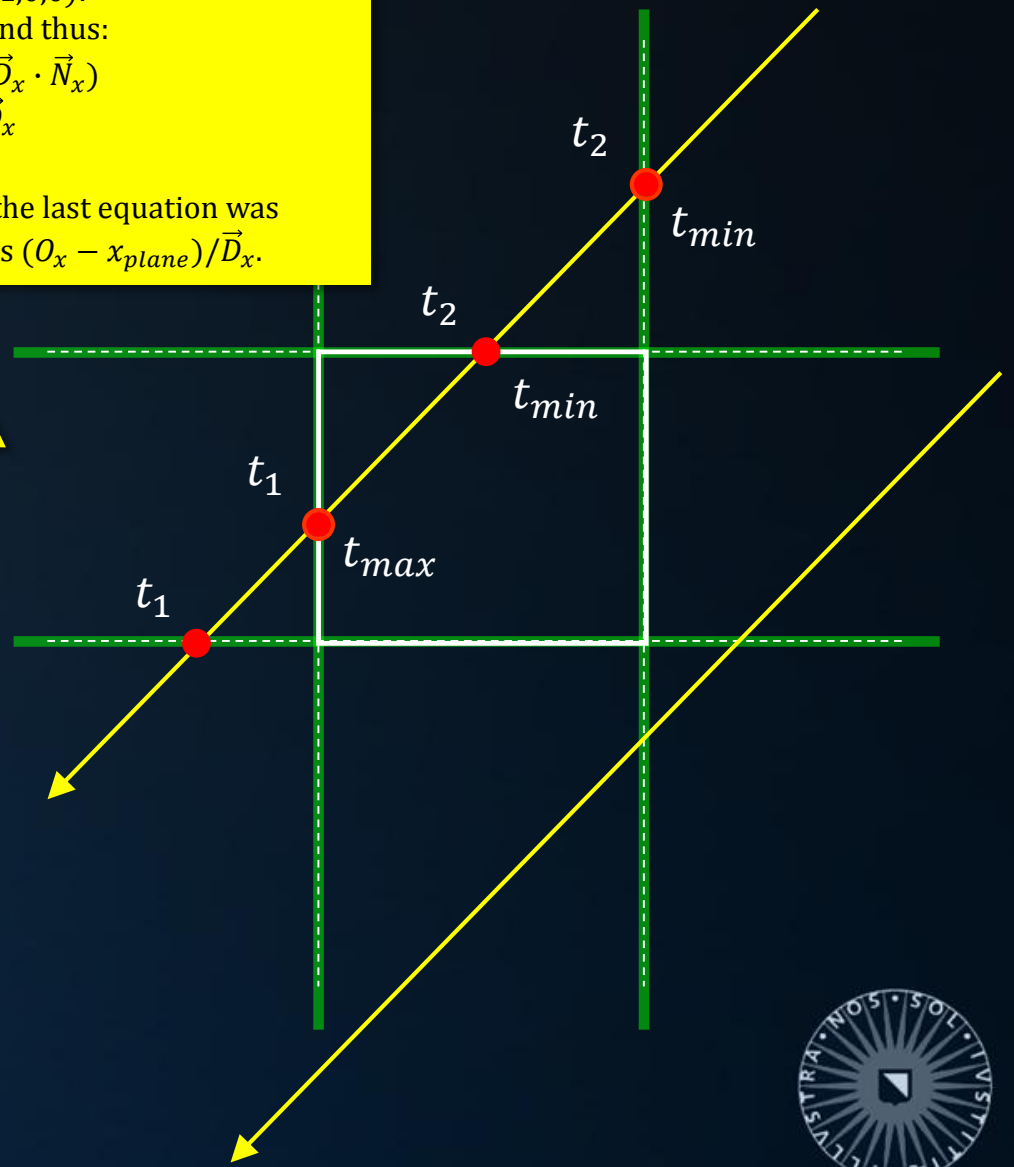
Slab test:

Intersect the ray against pairs of planes;
$t_{min} = +\infty, t_{max} = -\infty$
$t_{min} = \max(t_{min}, \min(t1, t2))$
$t_{max} = \min(t_{max}, \max(t1, t2))$
intersection if: $t_{min} < t_{max}$

Since the box is axis aligned, calculating t is cheap:

$$t = -(O \cdot \vec{N} + d)/(\vec{D} \cdot \vec{N})$$
$$= -(O_x \cdot \vec{N}_x + d)/(\vec{D}_x \cdot \vec{N}_x)$$
$$= (x_{plane} - O_x)/\vec{D}_x$$

$d = -(\vec{N} \cdot P)$, where P is a point on the plane.
In this case, for $\vec{N} = (1,0,0)$:
$d = -P_x = -x_{plane}$, and thus:
$t = -(O_x \cdot \vec{N}_x + d)/(\vec{D}_x \cdot \vec{N}_x)$
$= -(O_x - x_{plane})/\vec{D}_x$
$=(x_{plane} - O_x)/\vec{D}_x$
Note: during college, the last equation was erroneously written as $(O_x - x_{plane})/\vec{D}_x$.



Boxes

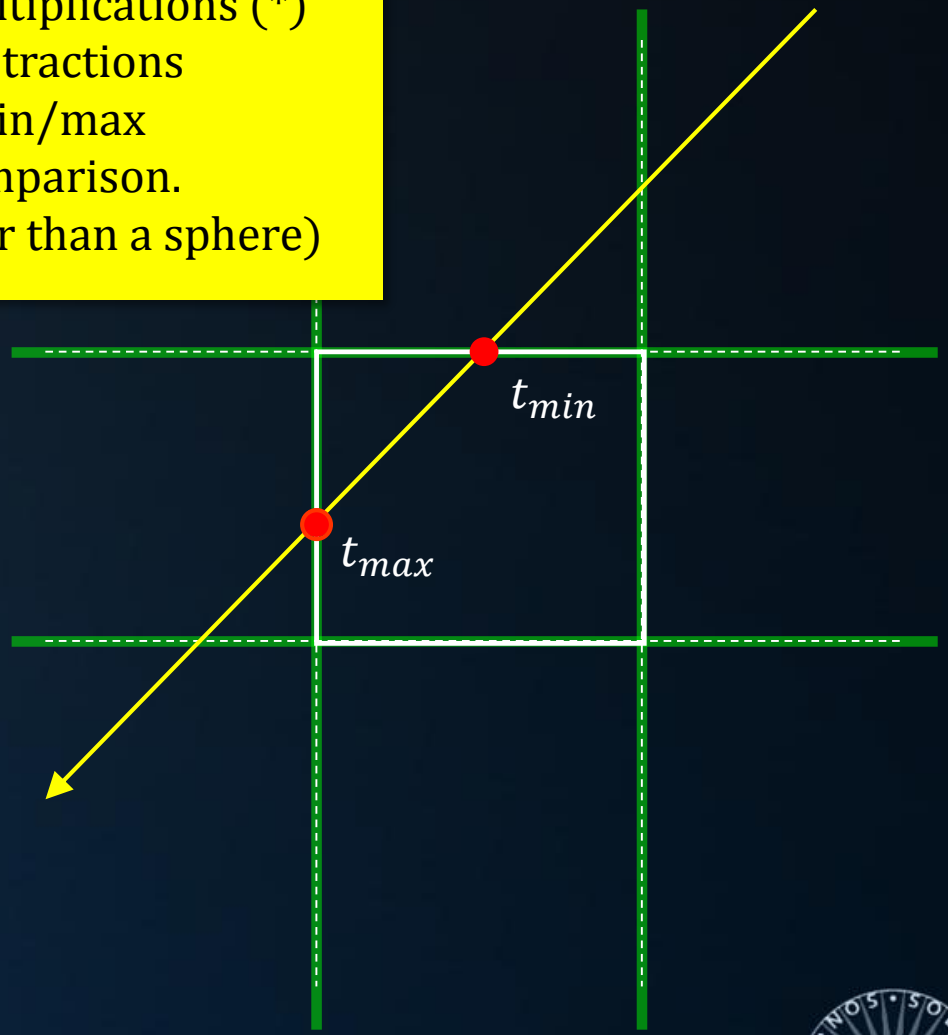
Special Case: AABB

In pseudo-code:

```
bool intersection( box b, ray r )
{
    float tx1 = (b.min.x - r.O.x) / r.D.x;
    float tx2 = (b.max.x - r.O.x) / r.D.x;

    float tmin = min( tx1, tx2 );
    float tmax = max( tx1, tx2 );

    float ty1 = (b.min.y - r.O.y) / r.D.y;
    float ty2 = (b.max.y - r.O.y) / r.D.y;

    tmin = max( tmin, min(ty1, ty2 ) );
    tmax = min( tmax, max(ty1, ty2 ) );

    return tmax >= tmin;
}
```

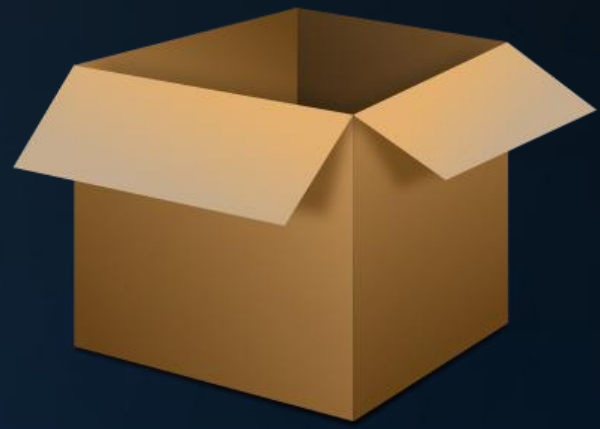
Intersecting a box in 3D:
- 6 multiplications (*)
- 6 subtractions
- 10 min/max
- 1 comparison.

(cheaper than a sphere)



Today’s Agenda:

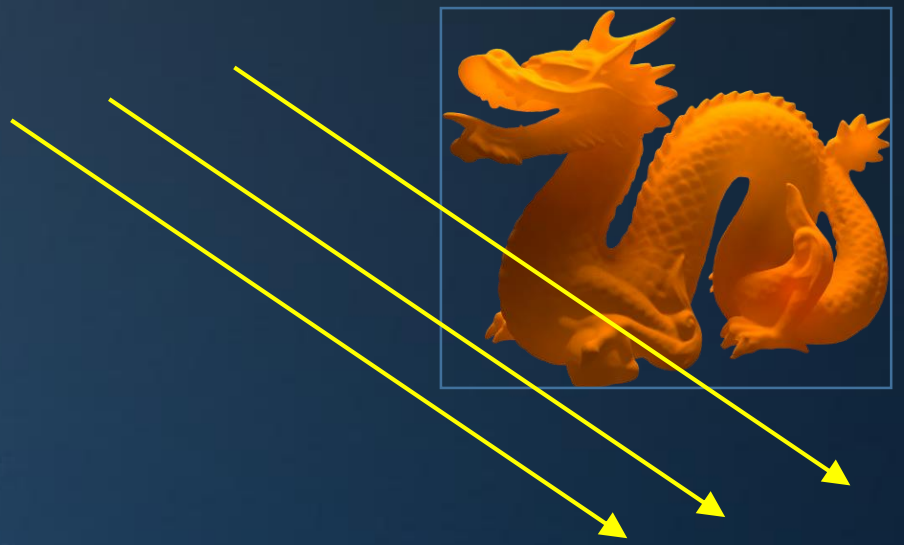
- Introduction
- Boxes
- AABBs
- Groupings
- Efficiency
- To Rasterization



AABBs

Why Do We Care

We can use an AABB to quickly discard objects.



AABBs

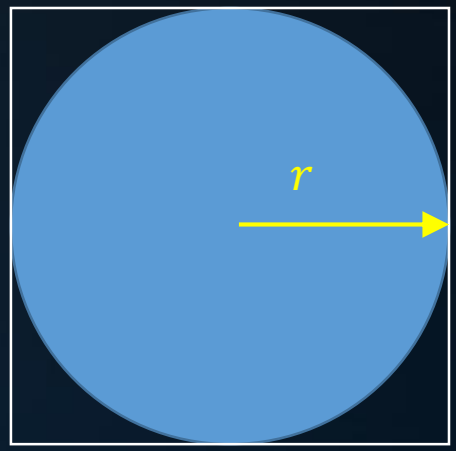
Calculating the AABB

Definition:

```
struct AABB
{
   vec3 bmin, bmax;
};
```

For a sphere:

```
AABB box;
box.bmin = centre – vec3( r, r, r );
box.bmax = centre + vec3( r, r, r );
```

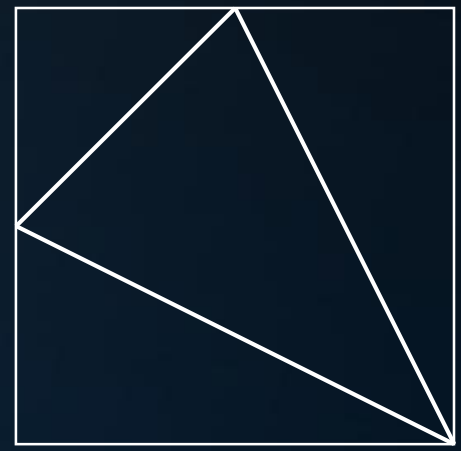


AABBs

Calculating the AABB

For a triangle:

```
AABB box;
box.bmin = vec3( +INF, +INF, +INF );
box.bmax = vec3( -INF, -INF, -INF );
for( int i = 0; i < 3; i++ )
{
   for( int a = 0; a < 3; a++ )
   {
      box.bmin[a] = min( vert[i][a], box.bmin[a] );
      box.bmax[a] = max( vert[i][a], box.bmax[a] );
   }
}
```



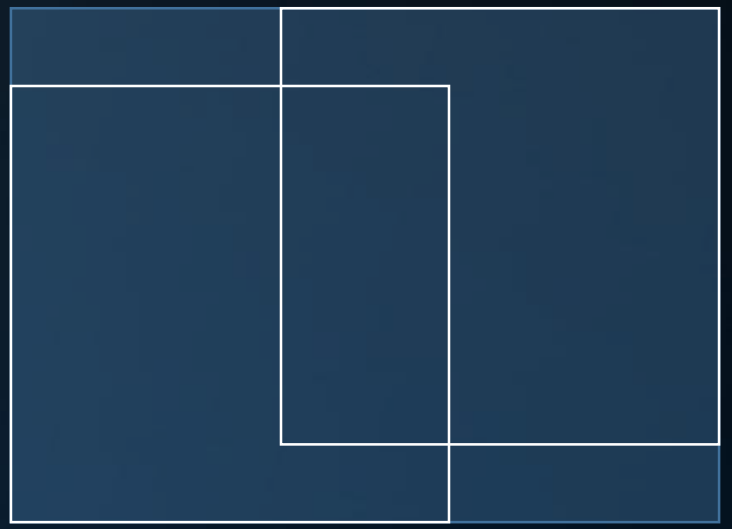
For multiple triangles, the algorithm is the same.

AABBs

Calculating the AABB

For multiple AABBs (union):

```
box.bmin.x = min( A.bmin.x, B.bmin.x );
box.bmax.x = max( A.bmax.x, B.bmax.x );
box.bmin.y = min( A.bmin.y, B.bmin.x );
box.bmax.y = max( A.bmax.y, B.bmax.y );
box.bmin.z = min( A.bmin.z, B.bmin.z );
box.bmax.z = max( A.bmax.z, B.bmax.z );
```



AABBs

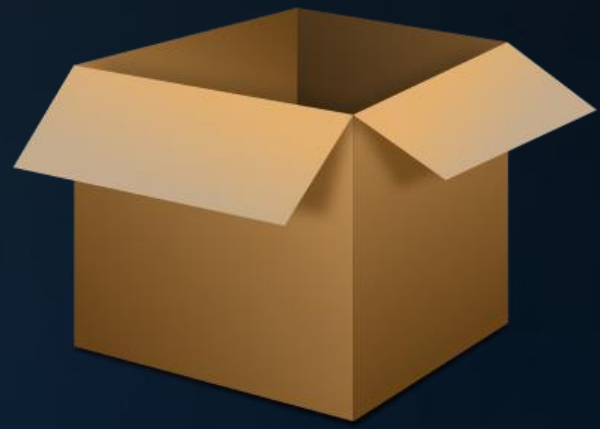
Calculating the AABB

Checking AABB intersection:

```
   (A.bmin.x < B.bmax.x)
&& (A.bmin.y < B.bmax.y)
&& (A.bmin.z < B.bmax.z)
```

Today’s Agenda:

- Introduction
- Boxes
- AABBs
- Groupings
- Efficiency
- To Rasterization

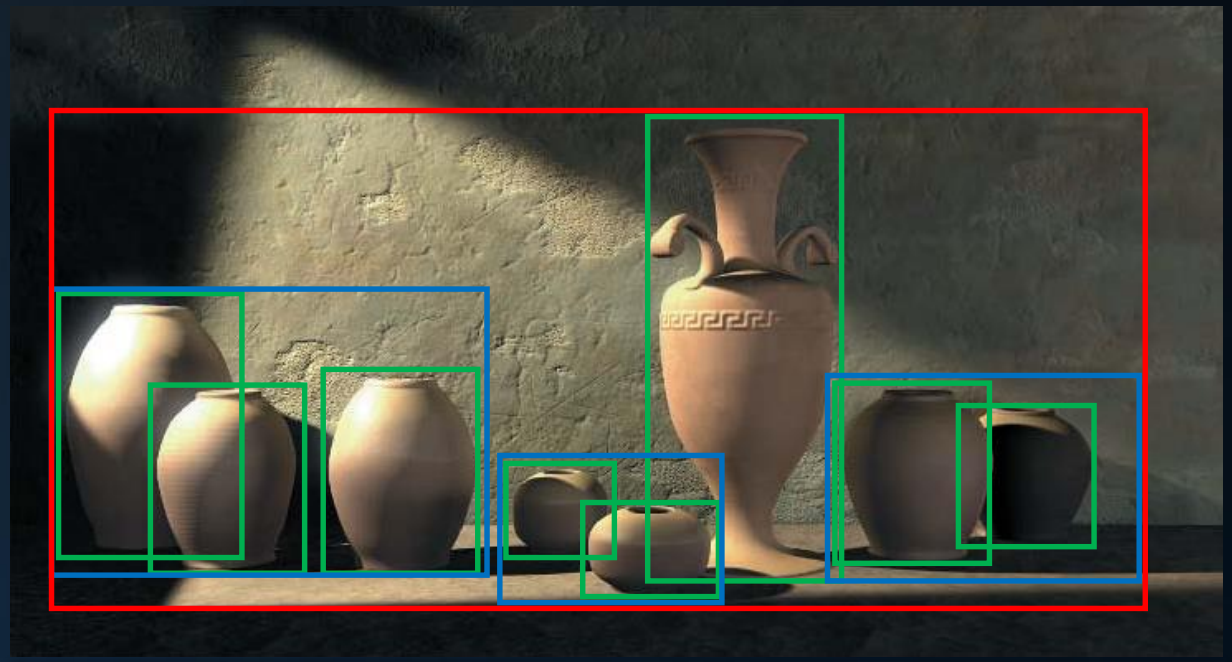


Groupings

Hierarchical Grouping

Using AABBs, we can recursively group objects.

- A ray that misses a green box will not check the triangles inside it;
- A ray that misses a blue box will skip the two green boxes inside it;
- A ray that misses the red box doesn’t hit anything at all.

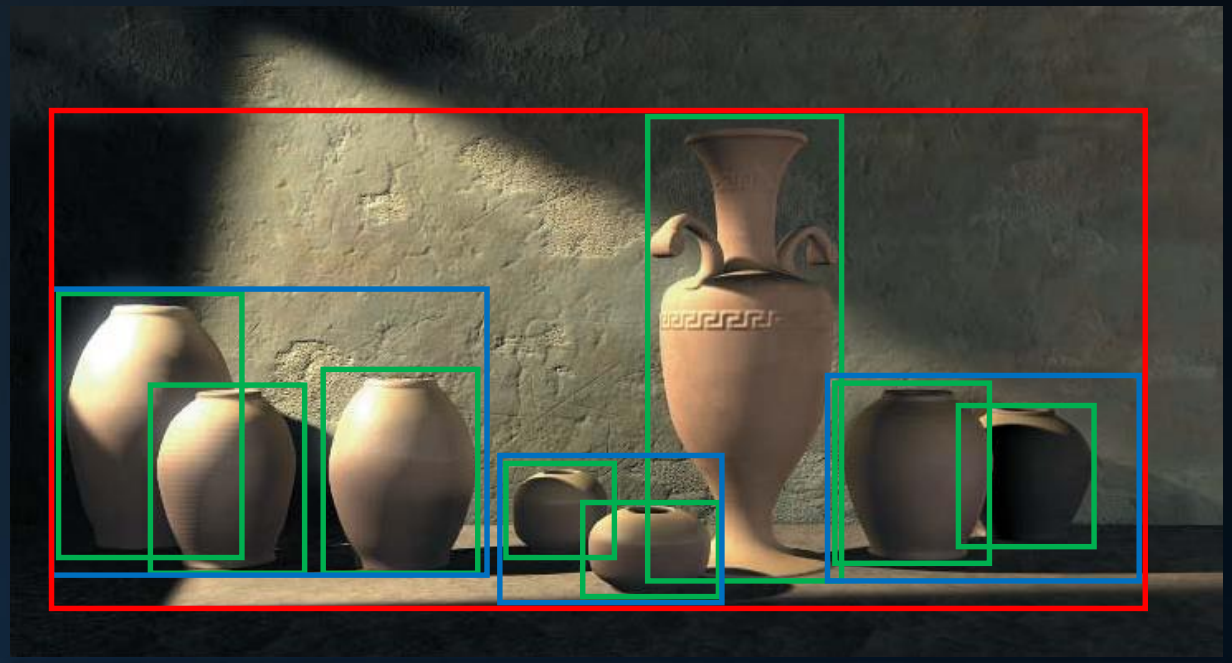


Groupings

Hierarchical Grouping

In a rasterization-based world:

- If a green box is outside the view frustum, we don’t have to render the triangles inside it;
- If a blue box is outside the view frustum, we don’t have to test the green boxes inside it;
- If the red box is outside the view frustum, we don’t see anything.

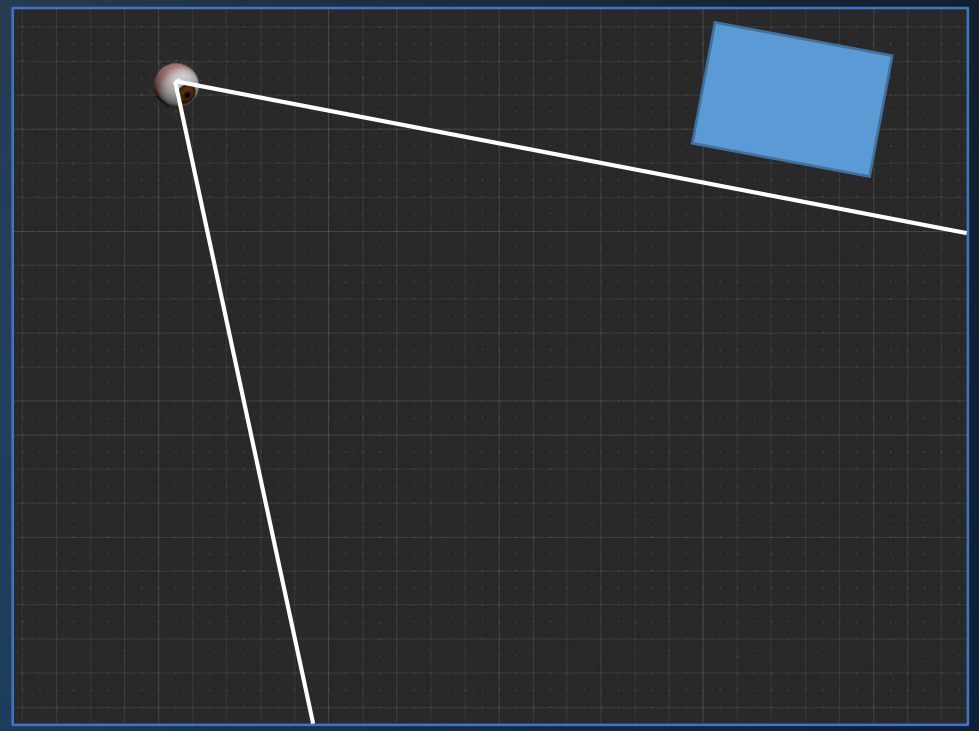


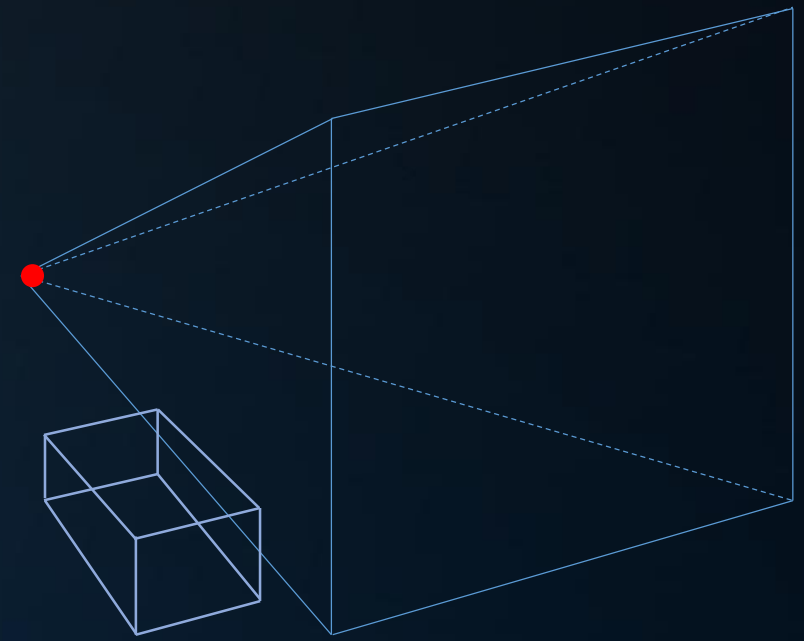
Groupings

Culling a Bounding Box

A bounding box is outside the view frustum when:

- *All it’s vertices on the backside of a single plane.*



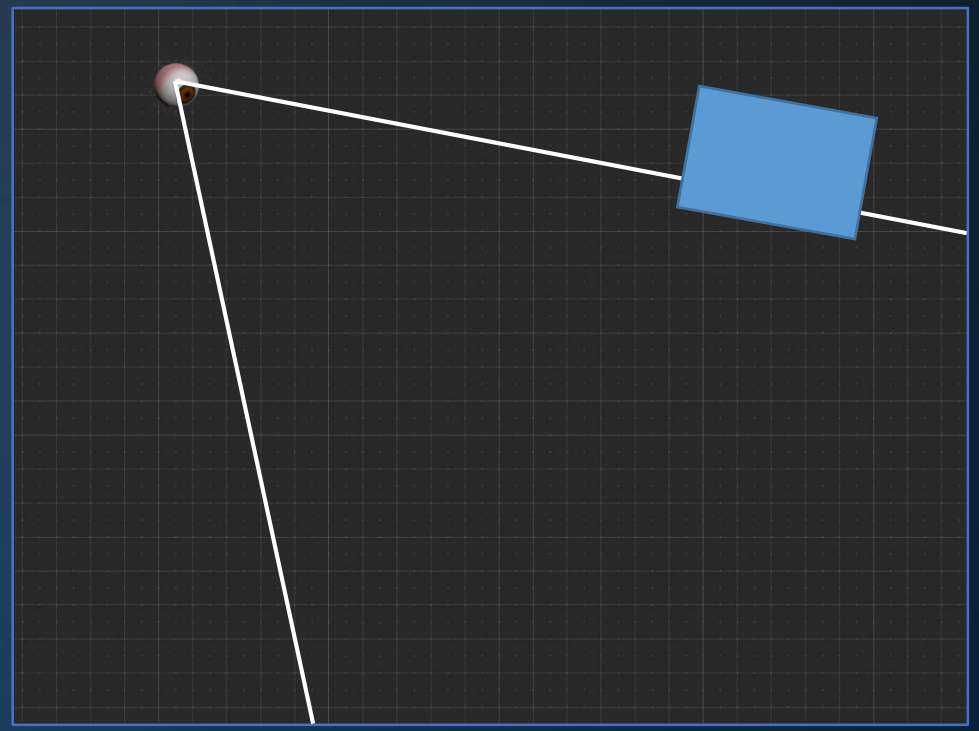


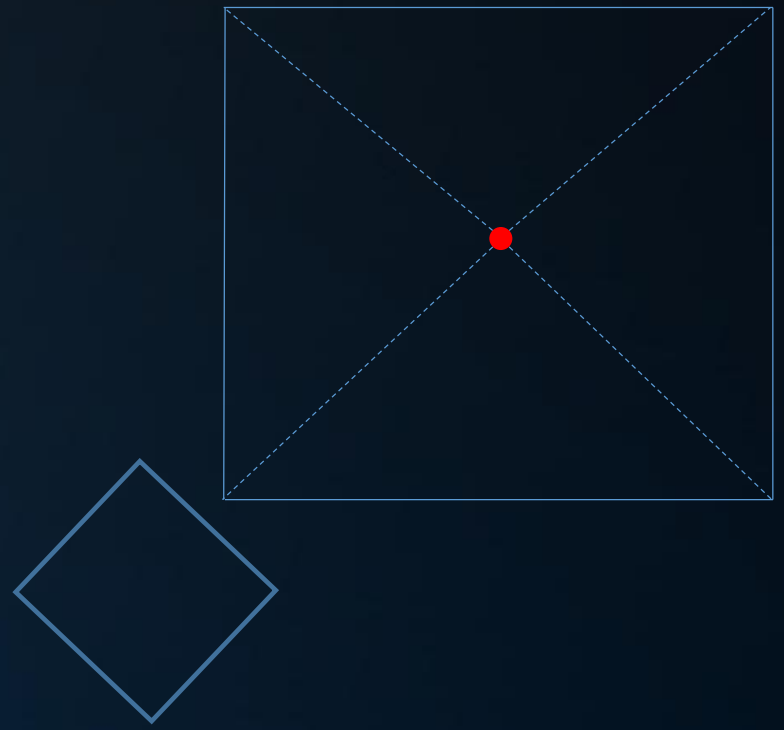
Groupings

Culling a Bounding Box

A bounding box is outside the view frustum when:

- *All it’s vertices on the backside of a single plane.*



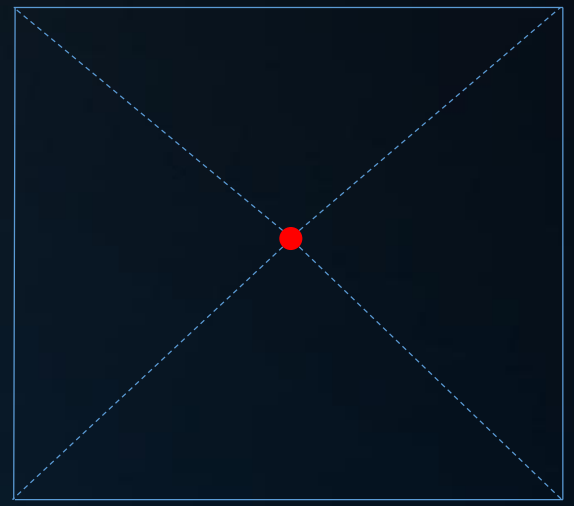


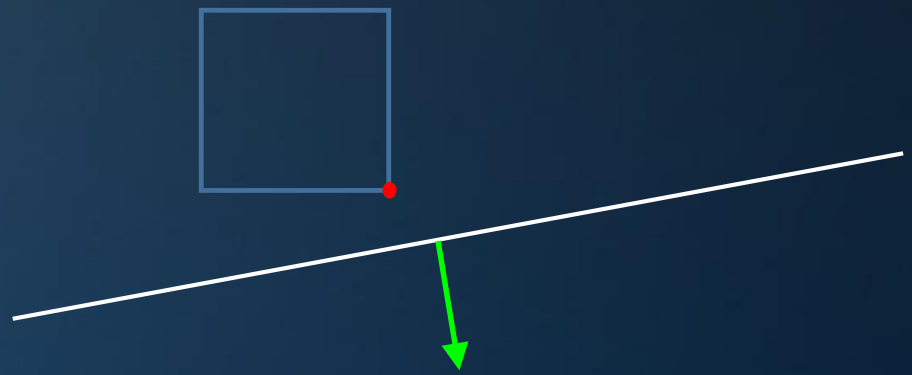
Groupings

Culling a Bounding Box

Instead of checking all eight vertices, we can limit the test to a single vertex.

- If $N_x>0$, we use bmax.x, else bmin.x;
- If $N_y>0$, we use bmax.y, else bmin.y;
- If $N_z>0$, we use bmax.z, else bmin.z.





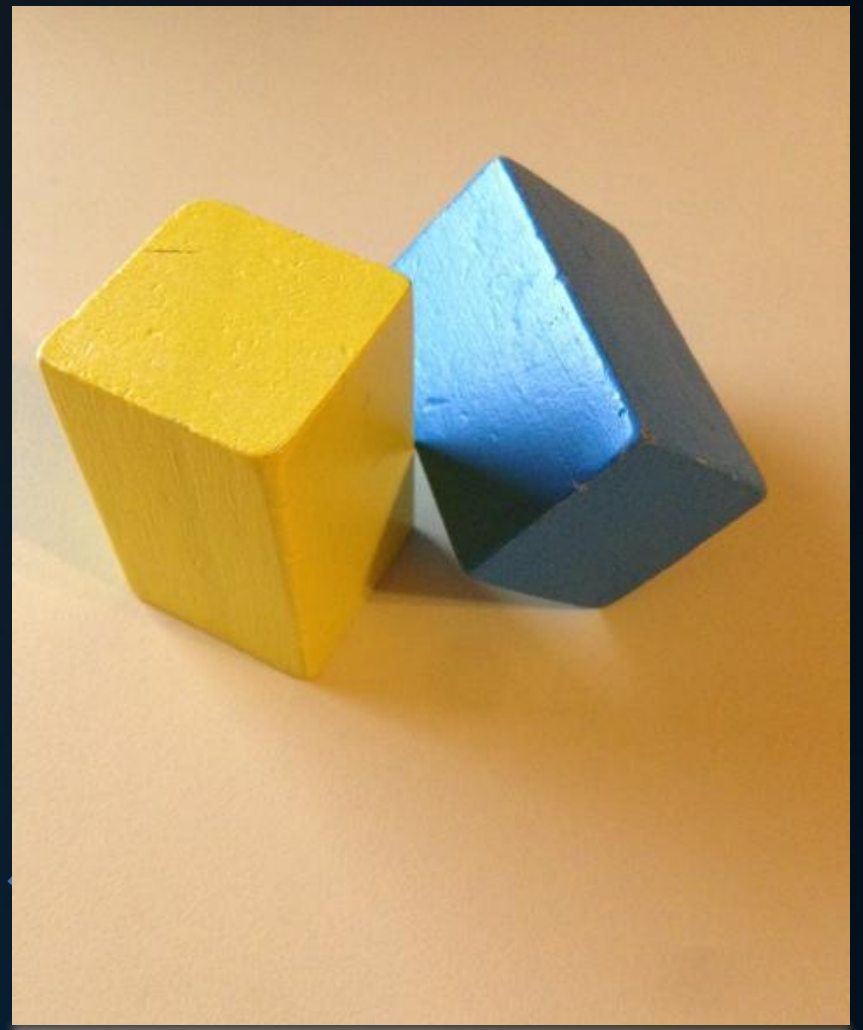
Groupings

Culling a Bounding Box

What about the problematic case?

1. Our test is a conservative test; i.e. it will produce *false negatives*, but no false positives.

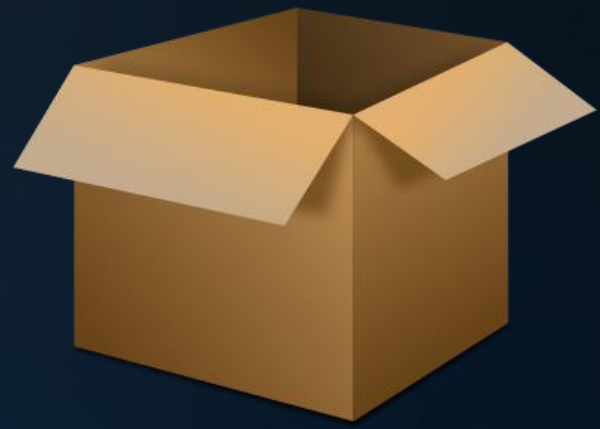
2. We can improve accuracy (at the cost of extra calculations) by reversing roles: use the planes of the AABB to cull the frustum.



Note that this still leaves certain tricky cases. For a perfect solution, check:
http://gamedev.stackexchange.com/questions/44500/how-many-and-which-axes-to-use-for-3d-obb-collision-with-sat

Today’s Agenda:

- Introduction
- Boxes
- AABBs
- Groupings
- Efficiency
- To Rasterization



Efficiency

Measuring Performance

Stopwatch class:

Using System.Diagnostics.Stopwatch;

Useful property:

- `long ElapsedMilliseconds { get; }`

Methods:

- `Reset`
- `Start`
- `Stop`

Note:

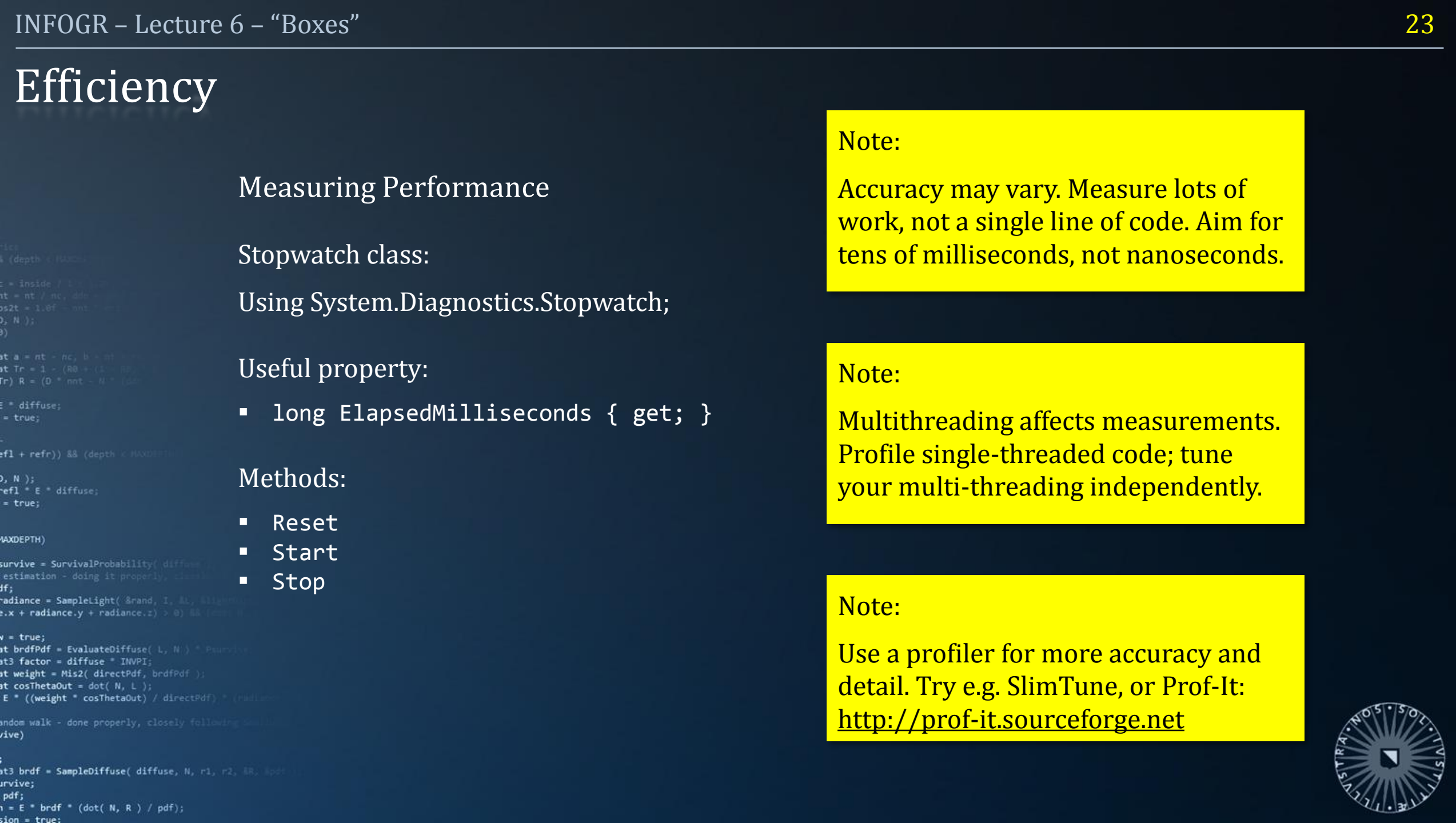
Accuracy may vary. Measure lots of work, not a single line of code. Aim for tens of milliseconds, not nanoseconds.

Note:

Multithreading affects measurements. Profile single-threaded code; tune your multi-threading independently.

Note:

Use a profiler for more accuracy and detail. Try e.g. SlimTune, or Prof-It: http://prof-it.sourceforge.net



Efficiency

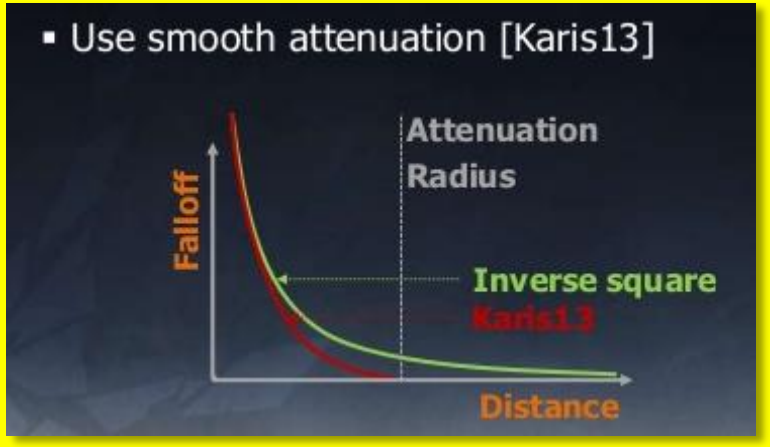
Optimization Primer

Some things to keep in mind:

- Float or double
- Don’t do work you don’t need to do
  - Early out
  - Reduce precision
  - Lights with finite radius
  - Things that can’t occlude light

$$attenuation = \frac{saturate\left(1 - \left(\frac{distance}{radius}\right)^4\right)^2}{distance^2 + 1}$$

From:
Real Shading in Unreal Engine 4, Karis, 2013
(also used in Frostbite).



Efficiency

Optimization Primer

Some things to keep in mind:

- Float or double
- Don’t do work you don’t need to do
- Precalculate
  - Loop hoisting
  - Vertex shaders

Loop hoisting:
Take expressions that do not rely on the loop counter outside the loop.

```
for( int i = 0; i < lights; i++ ) {
   vec3 N = intersection->GetNormal();
   vec3 L = light[i]->pos –
            intersection->pos;
   L.Normalize();
   if (dot( L, N ) > 0) {
      …
   }
}
```

➔

```
vec3 N = intersection->GetNormal();
for( int i = 0; i < lights; i++ ) {
   vec3 L = light[i]->pos –
            intersection->pos;
   if (dot( L, N ) > 0) {
      L.Normalize();
      …
   }
}
```

Efficiency

Optimization Primer

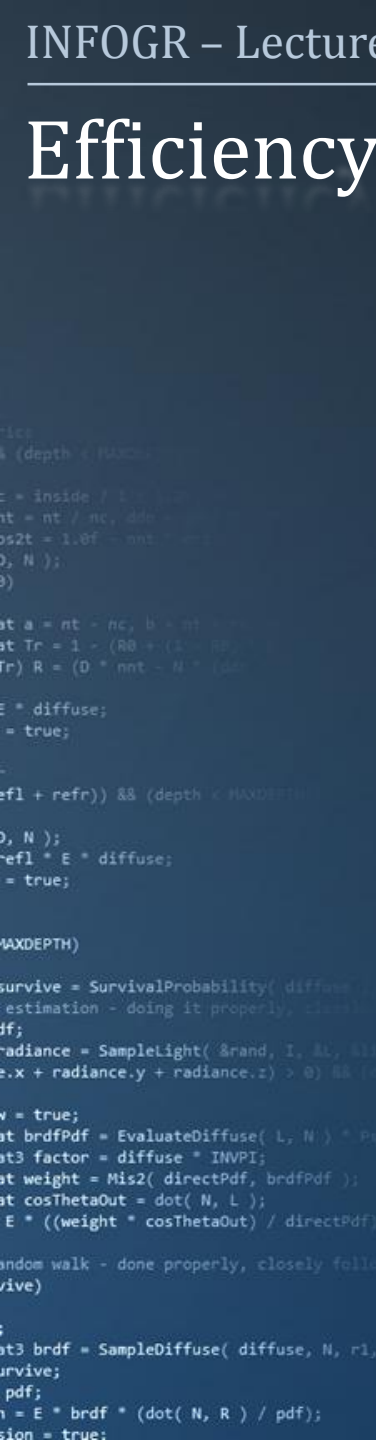
Some things to keep in mind:

- Float or double
- Don’t do work you don’t need to do
- Precalculate
- Expensive operations
  - sin, cos
  - sqrt
  - /
  - *
  - +, -

Look-up tables:

If you need sin/cos, it’s often much faster to use a look-up table.

```
float sintab[3600], costab[3600];
for( int i = 0; i < 3600; i++ )
{
   sintab[i] = Math.Sin( i / 10 );
   costab[i] = Math.Cos( i / 10 );
}

…

float s = sintab[(int)(a * 10)];
float c = costab[(int)(a * 10)];
```



Efficiency

Optimization Primer

Some things to keep in mind:

- Float or double
- Don’t do work you don’t need to do
- Precalculate
- Expensive operations
- Programming Language
  - C#/C++
  - C++/Asm

Efficiency

Perceived Performance

Incremental Rendering

1. Real-time preview:

- Depth map
- Depth map plus materials
- Render without recursive reflections
- Render with very limited recursion

Still not real-time?

- Render half-res
- Adaptive resolution
- Optimize the application a bit

Efficiency

Perceived Performance

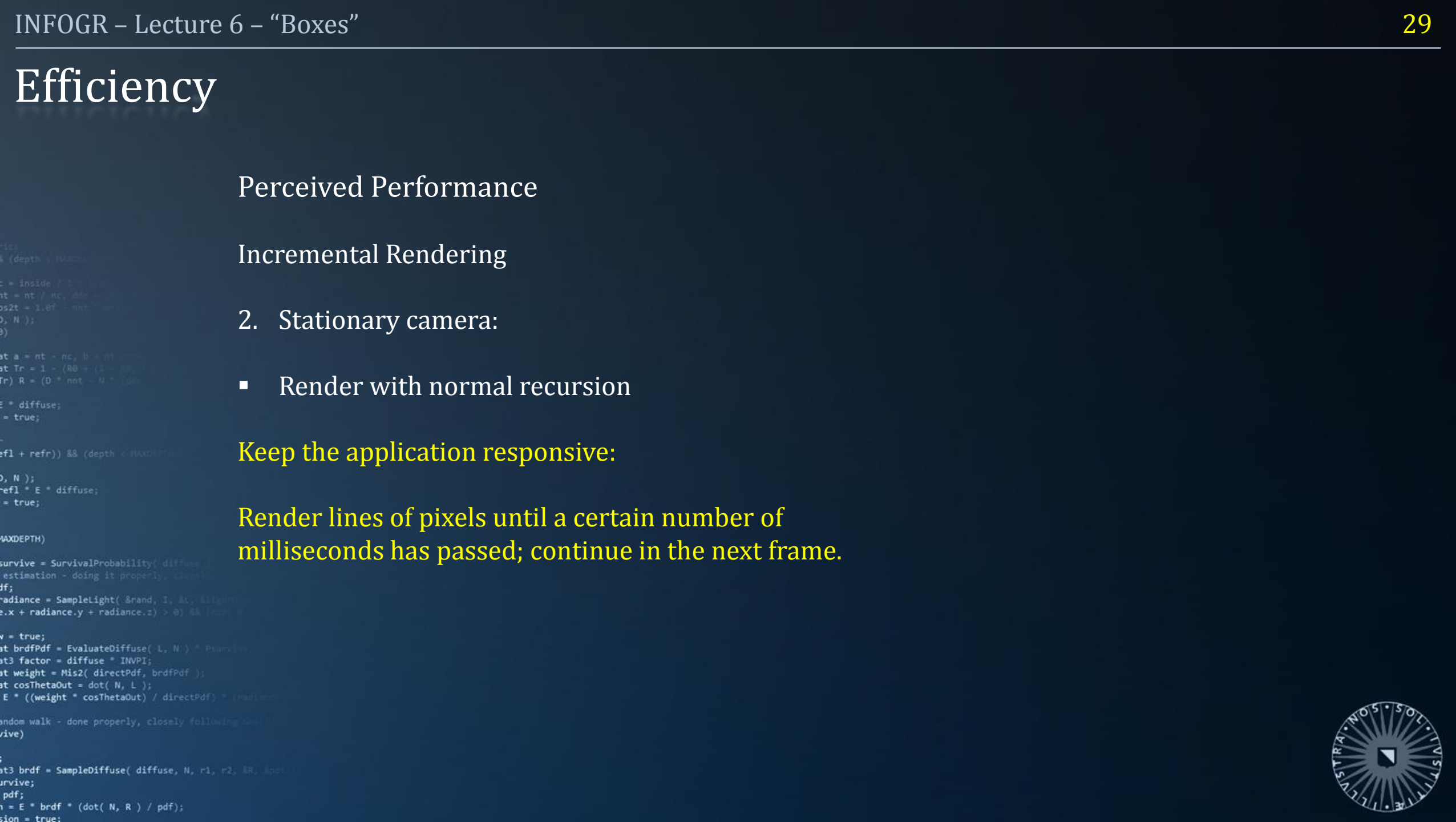
Incremental Rendering

2. Stationary camera:

- Render with normal recursion

Keep the application responsive:

Render lines of pixels until a certain number of milliseconds has passed; continue in the next frame.



Efficiency

Perceived Performance

Incremental Rendering

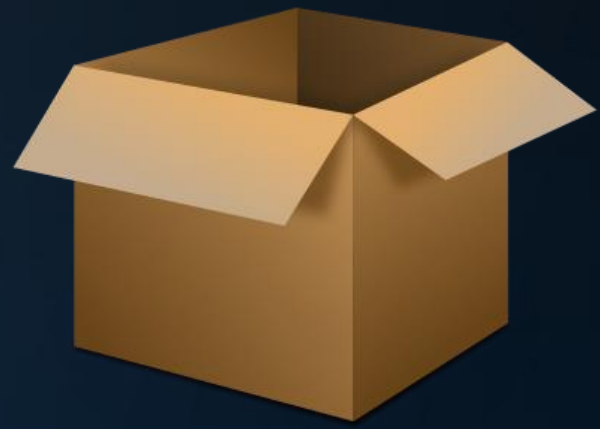
3. ‘Photograph mode’ :

- Invoked with a key
- Render with extreme recursion
- Use anti-aliasing
- Add screenshot feature

Keep the application responsive!

Today’s Agenda:

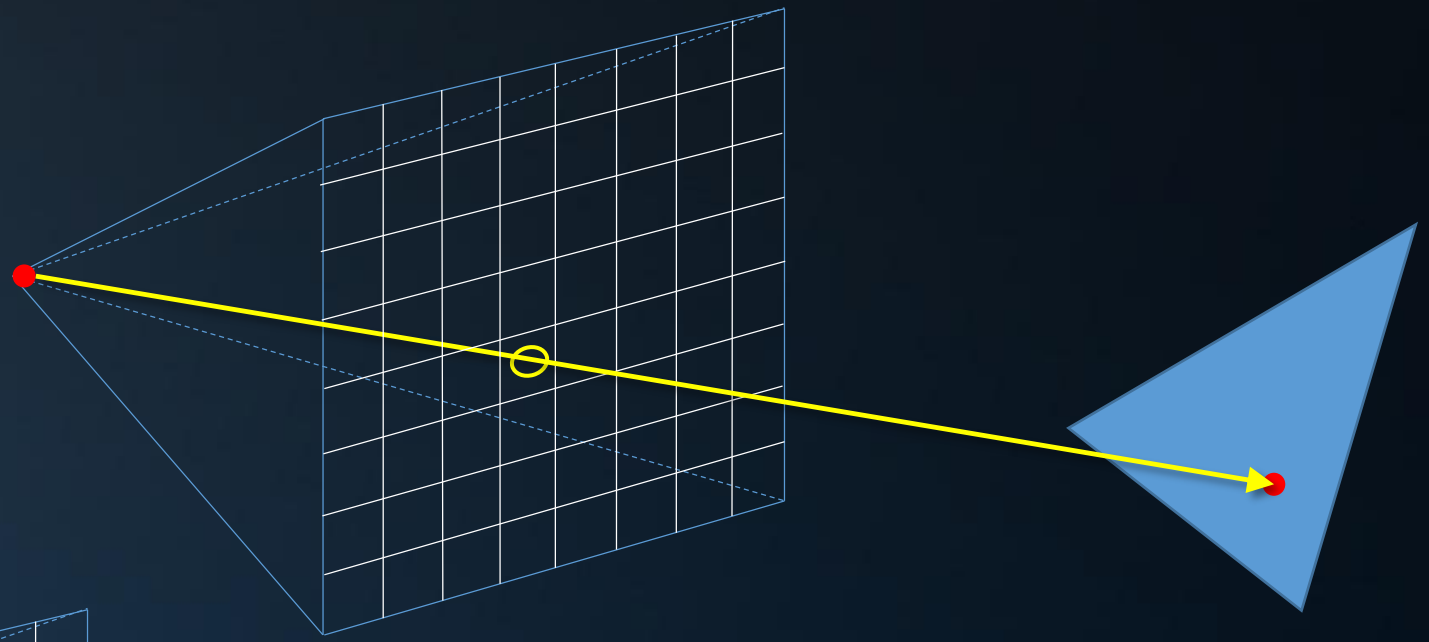
- Introduction
- Boxes
- AABBs
- Groupings
- Efficiency
- To Rasterization

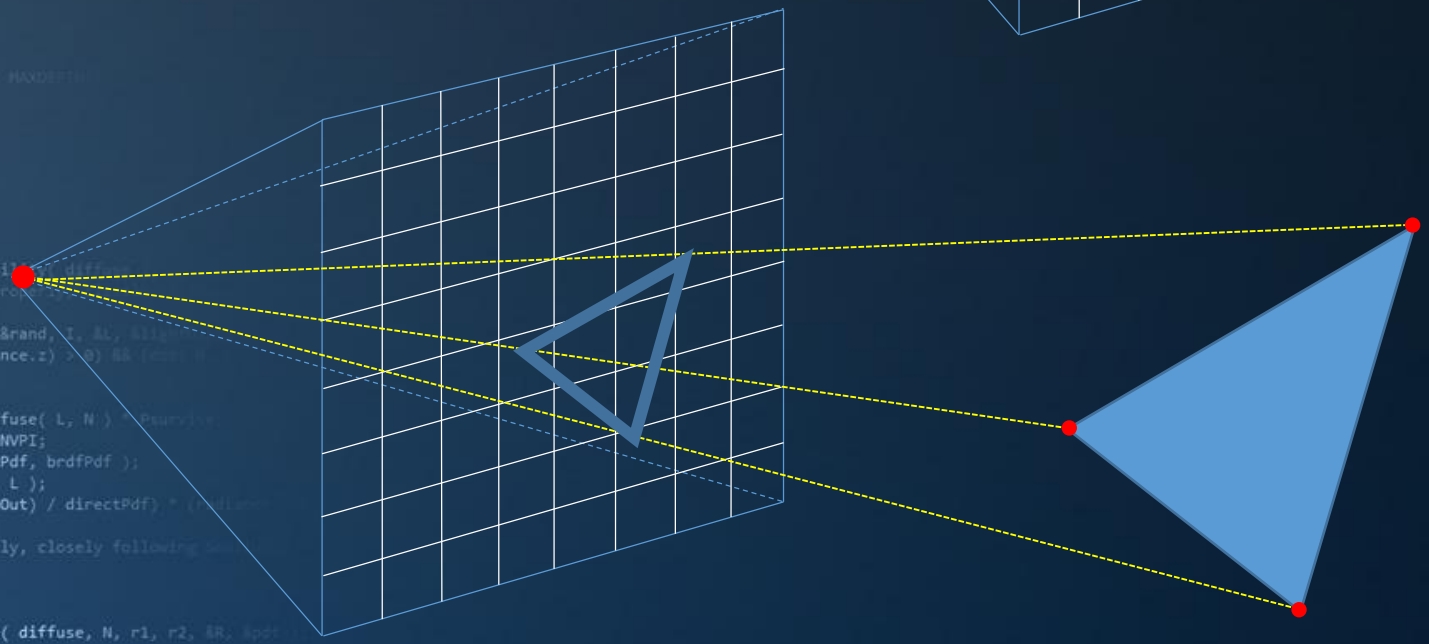


Rasterization

Primary Rays

Ray tracing versus Rasterization





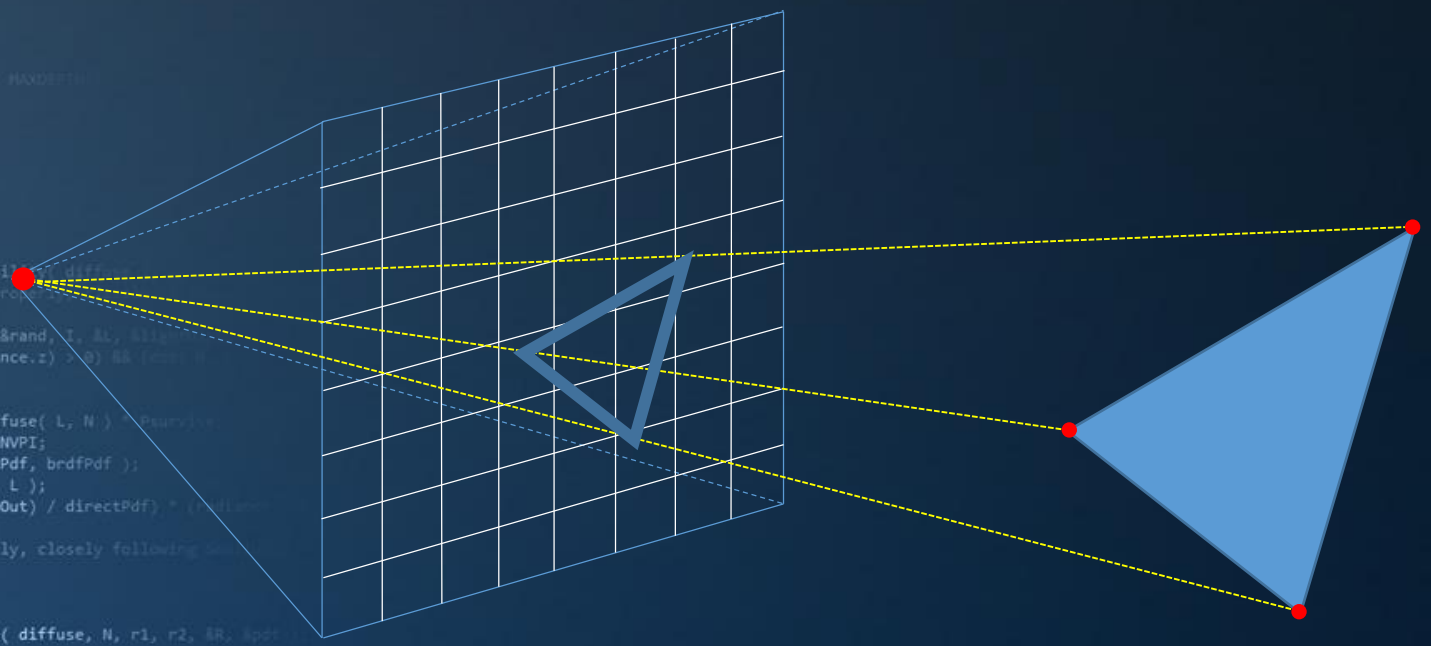
Rasterization

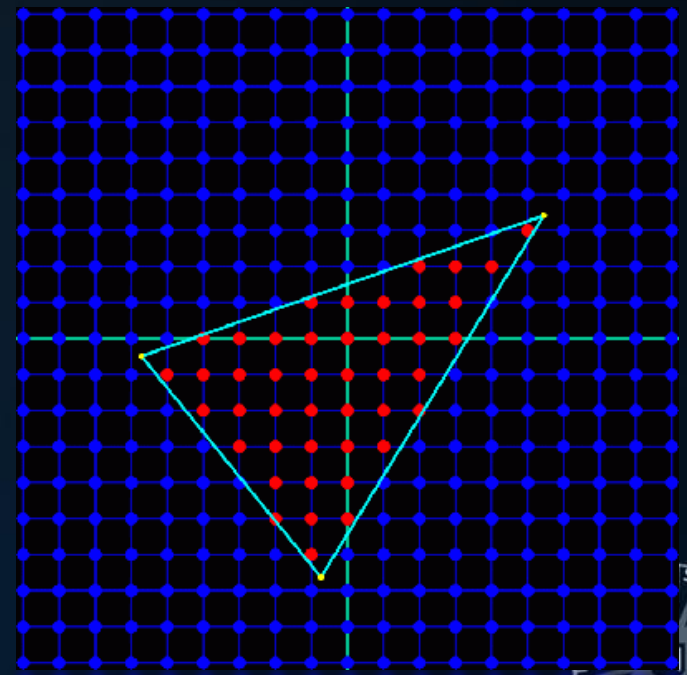
Primary Rays

Ray tracing versus Rasterization

Rasterization:

1. Transform primitive into camera space
2. Project vertices into 2D screen space
3. Determine which pixels are affected
4. Use z-buffer to sort (pixels of) primitives
5. Clip against screen boundaries





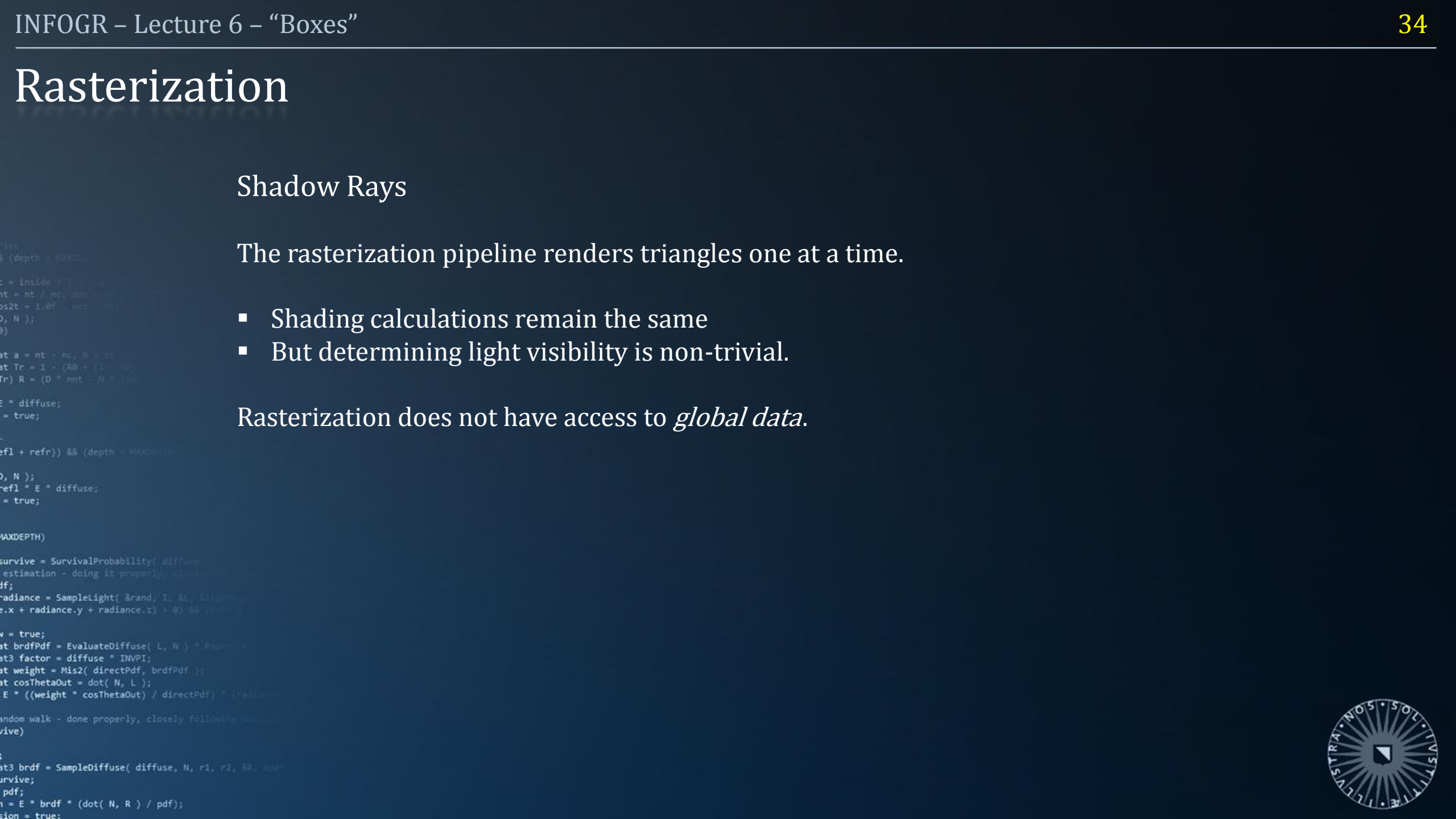
Rasterization

Shadow Rays

The rasterization pipeline renders triangles one at a time.

- Shading calculations remain the same
- But determining light visibility is non-trivial.

Rasterization does not have access to *global data*.



Rasterization

Spaces

Ray tracing typically happens in a single 3D coordinate system.

In rasterization, we use many coordinate systems:

- Camera space
- Clip space
- 2D screen space
- Model space
- Tangent space

We need efficient tools to get from one space to another. We will make extensive use of matrices to do this.

Rasterization

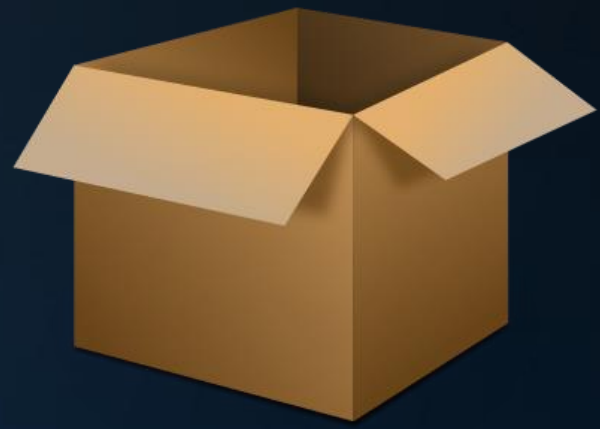
Common Concepts

Many things remain the same:

- Normal interpolation
- Shading
- Texture mapping
- The camera
- Boxes.

Today’s Agenda:

- Introduction
- Boxes
- AABBs
- Groupings
- Efficiency
- To Rasterization



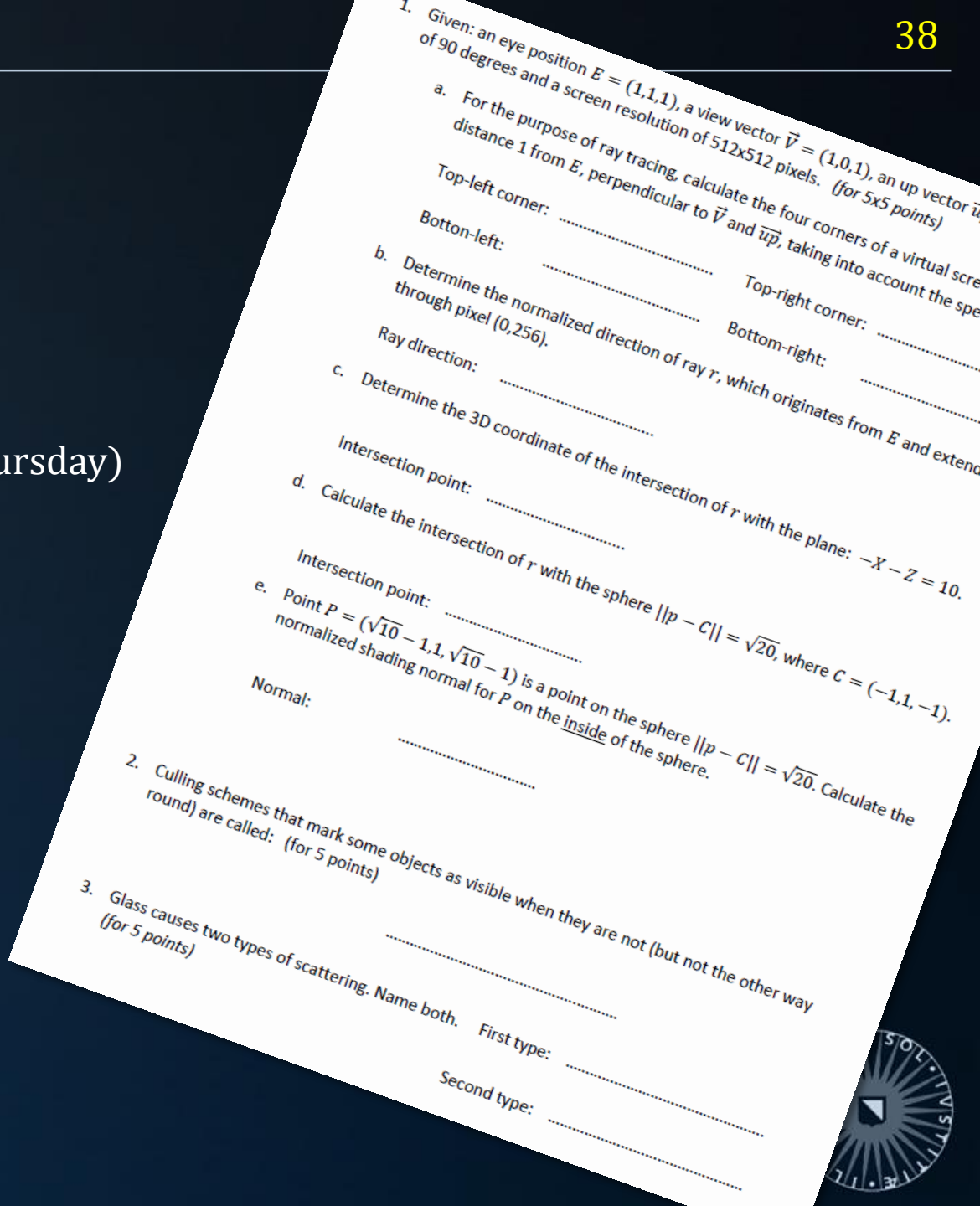
Mid-term Exam

What to study for the exam?

1. Slides (mind cursive terminology!)
2. Example exam (now online, discuss next Thursday)
3. Tutorial sheets

Expectations:

- Fluency with vectors, including dot product, cross product, normalization and all combinations thereof.
- Good understanding of the ray tracing algorithm and light transport.
- Knowledge of terminology used in the lectures.


1. Given: an eye position $E = (1,1,1)$, a view vector $\vec{V} = (1,0,1)$, an up vector $\vec{u}$
of 90 degrees and a screen resolution of 512x512 pixels. *(for 5x5 points)*

a. For the purpose of ray tracing, calculate the four corners of a virtual scre
distance 1 from E, perpendicular to $\vec{V}$ and $\vec{up}$, taking into account the spe

Top-left corner: ..... Top-right corner: .....

Botton-left: ..... Bottom-right: .....

b. Determine the normalized direction of ray r, which originates from E and extend
through pixel (0,256).

Ray direction: .....

c. Determine the 3D coordinate of the intersection of r with the plane: $-X - Z = 10$.

Intersection point: .....

d. Calculate the intersection of r with the sphere $||p - C|| = \sqrt{20}$, where $C = (-1,1,-1)$.

Intersection point: .....

e. Point $P = (\sqrt{10} - 1,1,\sqrt{10} - 1)$ is a point on the sphere $||p - C|| = \sqrt{20}$. Calculate the
normalized shading normal for P on the inside of the sphere.

Normal: .....

2. Culling schemes that mark some objects as visible when they are not (but not the other way
round) are called: *(for 5 points)*

.....

3. Glass causes two types of scattering. Name both. *(for 5 points)*

First type: .....

Second type: .....

INFOGR – Computer Graphics

Jacco Bikker - April-July 2016 - Lecture 6: “Boxes”

END of “Boxes”

next lecture: “Acceleration Structures”