Welcome!
Recap

Color Trace( vec3 O, vec3 D )
{
    I, N, mat = IntersectScene( O, D );
    if (!I) return BLACK;
    return DirectIllumination( I, N ) * mat.diffuseColor;
}
Recap

Color Trace (vec3 O, vec3 D)
{
    I, N, mat = IntersectScene( O, D);
    if (!I) return BLACK;
    return DirectIllumination( I, N ) * mat.diffuseColor;
}

Schlick:

\[ F_r = R_0 + (1 - R_0)(1 - \cos\theta)^5, \]
\[ R_0 = \left( \frac{n_1 - n_2}{n_1 + n_2} \right)^2. \]

\[ \hat{N} = \frac{\hat{N}}{n_1} \]
\[ \hat{R} = \frac{\hat{R}}{n_1} \]

\[ \hat{R} = \hat{R} - \hat{R} \]
\[ \hat{R} = \hat{R} - \hat{R} \]

\[ \hat{R} = \hat{R} - \hat{R} \]
\[ \hat{R} = \hat{R} - \hat{R} \]

\[ n_1 \sin\theta_1 = n_2 \sin\theta_2 \iff \sin\theta_1 \leq \frac{n_2}{n_1} \]
Camera looks straight ahead along \( z \):

\[
\hat{v} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \text{ or } \hat{v} = \begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix}
\]

\[
C = E + d\hat{v}
\]

\[
p_0 = C + \begin{pmatrix} -1 \\ 0 \\ 0 \end{pmatrix}, \quad p_1 = C + \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \quad p_2 = C + \begin{pmatrix} -1 \\ 0 \\ 0 \end{pmatrix}
\]

Arbitrary \( \hat{v} \):

let \( \hat{u} \hat{p} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \). Then:

\[
\hat{r} = \frac{\hat{u} \hat{p} \times \hat{v}}{\|\hat{u} \hat{p} \times \hat{v}\|} \quad \hat{r} = \frac{\hat{v} \times \hat{u} \hat{p}}{\|\hat{v} \times \hat{u} \hat{p}\|}
\]

\[
p_0 = C - \hat{r} + \hat{u}
\]

\[
p_1 = C + \hat{r} + \hat{u}
\]

\[
p_2 = C - \hat{r} - \hat{u}
\]
Today’s Agenda:

- Efficiency
- Boxes, AABBS & Groupings
- To Rasterization
- 3D Engine Overview
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

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

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.

```cpp
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) {
        ...
    }
}
```

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
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.

```c
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!
Efficiency

Tricks
Today's Agenda:

- Efficiency
- Boxes, AABBs & Groupings
- To Rasterization
- 3D Engine Overview
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.
Why Do We Care

We can use a box to quickly discard objects.
Boxes

Why Do We Care

We can use a box to quickly discard (parts of) objects.
Why Do We Care

We can use a sphere to quickly discard (parts of) objects.
Hierarchical Grouping

Using boxes, 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.
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.
Boxes

Special Case: AABB

AABB: *Axis Aligned Bounding Box.*

It's a special box:

- Each of the 6 faces is parallel to a pair of axes.
- Each of the 6 faces has a simple normal.
- We can store the AABB in a compact format.
- It is cheap to intersect.
Boxes

Special Case: AABB

AABB: Axis Aligned Bounding Box.

Slab test:

Intersect the ray against pairs of planes.

\[ t_{\text{min}} = +\infty, \quad t_{\text{max}} = -\infty \]
\[ t_{\text{min}} = \max (t_{\text{min}}, \min (t_1, t_2)) \]
\[ t_{\text{max}} = \min (t_{\text{max}}, \max (t_1, t_2)) \]

intersection if: \( t_{\text{min}} < t_{\text{max}} \)

Since the box is axis aligned, calculating \( t \) is cheap:

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

\( d = -(\vec{N} \cdot P) \), where \( P \) is a point on the plane.

In this case, for \( \vec{N} = (1,0,0) \):

\( d = -P_x = -x_{\text{plane}} \), and thus:

\[ t = -(O_x \cdot \vec{N}_x + d)/(\vec{D}_x \cdot \vec{N}_x) \]
\[ = -(O_x - x_{\text{plane}})/\vec{D}_x \]
\[ = (x_{\text{plane}} - O_x)/\vec{D}_x \]
Special Case: AABB

In pseudo-code:

```cpp
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;
    float tmin = max( tmin, min(ty1, ty2) );
    float tmax = min( tmax, max(ty1, ty2) );

    return tmax >= tmin && tmax > 0;
}
```

Intersecting a box in 3D:
- 6 multiplications (*)
- 6 subtractions
- 10 min/max
- 1 comparison.
(cheaper than a sphere)
Calculating the AABB

Definition:

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

For a sphere:

```c
AABB box;
box.bmin = centre - vec3( r, r, r );
box.bmax = centre + vec3( r, r, r );
```
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.
Calculating the AABB

For multiple AABBs (union):

```plaintext
textcode:
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.y );
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 );
```
Boxes

Calculating the AABB

Checking AABB intersection:

\[(A.bmin.x < B.bmax.x) \land (A.bmin.y < B.bmax.y) \land (A.bmin.z < B.bmax.z)\]
Culling a Bounding Box

A bounding box is outside the view frustum when:

- All its vertices on the backside of a single plane.
Groupings

Culling a Bounding Box

A bounding box is outside the view frustum when:

- **All its 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 $b_{\text{max}.x}$, else $b_{\text{min}.x}$;
- If $N_y > 0$, we use $b_{\text{max}.y}$, else $b_{\text{min}.y}$;
- If $N_z > 0$, we use $b_{\text{max}.z}$, else $b_{\text{min}.z}$.
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.

Today’s Agenda:

- Efficiency
- Boxes, AABBs & Groupings
- To Rasterization
- 3D Engine Overview
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
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.
Common Concepts

Many things remain the same:

- Normal interpolation
- Shading
- Texture mapping
- The camera
- Boxes.
Today's Agenda:

- Efficiency
- Boxes, AABBs & Groupings
- To Rasterization
- 3D Engine Overview
3D Engine

Topics covered so far:

Basics:
- Rasters
- Vectors
- Color representation

Ray tracing:
- Light transport
- Camera setup
- Textures

Shading:
- N dot L
- Distance attenuation
- Pure specular
3D Engine

Rendering – Functional overview

1. **Transform:**
   translating / rotating / scaling meshes

2. **Project:**
   calculating 2D screen positions

3. **Rasterize:**
   determining affected pixels

4. **Shade:**
   calculate color per affected pixel

- Animation, culling, tessellation, ...
- meshes
- vertices
- fragment positions
- pixels
- Postprocessing
- Transform
- Project
- Rasterize
- Shade
INFOGR – Lecture 7 – “Accelerate”

3D Engine

Rendering – Data Overview

- Camera
- World
- Car
- Plane
- Buggy

Transformation Matrices:
- $T_{camera}$
- $T_{car1}$
- $T_{plane1}$
- $T_{car2}$
- $T_{plane2}$
- $T_{buggy}$
Rendering – Data Overview

Objects are organized in a hierarchy: the *scenegraph*.

In this hierarchy, objects have translations and orientations relative to their parent node.

Relative translations and orientations are specified using matrices.

Mesh vertices are defined in a coordinate system known as *object space*. 
Rendering – Data Overview

Transform takes our meshes from object space (3D) to camera space (3D).

Project takes the vertex data from camera space (3D) to screen space (2D).
3D Engine

Rendering – Data Overview

The screen is represented by (at least) two buffers:
3D Engine

Rendering – Components

Scenegraph

Culling

Vertex transform pipeline

Matrices to convert from one space to another

Perspective

Rasterization

Interpolation

Clipping

Depth sorting: z-buffer

Shading

Light / material interaction

Complex materials

Lecture 12

Lecture 8, 9, 10

Lecture 11

Lecture 12

Lecture 12

P1

P2
Today’s Agenda:

- Efficiency
- Boxes, AABBs & Groupings
- To Rasterization
- 3D Engine Overview