Welcome!
INFOGR – Lecture 7 – “Accelerate”

Recap

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

\[ A \cdot x + B \cdot y + C \cdot z + D = 0 \]

\[ A \cdot 6 + B \cdot 3 + C \cdot 3 + D = 0 \]

\[ 0 \cdot 6 + 1 \cdot 3 + 0 \cdot 3 + D = 0 \]

\[ 3 + D = 0 \Rightarrow D = -3 \Rightarrow D = -(P \cdot \hat{N}) \]
Recap

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

\[ Ax + By + Cz + d = \vec{N} \cdot p + d = 0 \] and \[ p(t) = O + t\vec{D} \]

\[ \vec{N} \cdot (O + t\vec{D}) + d = 0 \Rightarrow \vec{N} \cdot O + \vec{N} \cdot (t\vec{D}) + d = 0 \]

\[ \vec{N} \cdot (t\vec{D}) = -(\vec{N} \cdot O + d) \Rightarrow t = -(\vec{N} \cdot O + d)/ (\vec{N} \cdot \vec{D}) \]
Recap

InfoGr - Lecture 7 – “Accelerate”

Schlick:

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

Color Trace( vec3 O, vec3 D )

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

\[ \hat{D} \cdot \hat{N} \]
\[ \hat{D} \]
\[ \hat{N} \]
\[ ? \]
\[ \vec{v} = \vec{D} - \vec{u} \]
\[ R = \vec{v} - \vec{u} \]
\[ R = (\vec{D} - \vec{\hat{N}}(\vec{D} \cdot \hat{N})) - \vec{\hat{N}}(\vec{D} \cdot \hat{N}) \]
\[ R = \vec{D} - 2\vec{\hat{N}}(\vec{D} \cdot \hat{N}) \]
\[ n_1 \sin \theta_1 = n_2 \sin \theta_2 \iff \sin \theta_1 \leq \frac{n_2}{n_1} \]

D.P.: \( \vec{D} \)

\( p_1 \)

\( p_2 \)

\( p_0 \)

\( \vec{v} \)

\( \vec{u} \)

\( \vec{\hat{N}} \)

\( \vec{\hat{D}} \)

\( \vec{\hat{R}} \)

\( \vec{\hat{\hat{N}}} \)

\( \vec{\hat{\hat{D}}} \)

\( \vec{\hat{\hat{R}}} \)

\( \vec{n}_1 \)

\( \vec{n}_2 \)

\( \vec{\hat{n}} \)

\( \vec{n}_1 \)

\( \vec{n}_2 \)

\( \vec{\hat{n}} \)

\( \theta_1 \)

\( \theta_2 \)
Camera looks straight ahead along $z$:

$$\vec{V} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \text{ or } \vec{V} = \begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix}$$

$$C = E + d\vec{V}$$

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

Arbitrary $\vec{V}$:

Let $\vec{u}_p = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$. Then:

$$\hat{r} = \frac{\vec{u}_p \times \vec{V}}{||\vec{u}_p \times \vec{V}||} \quad \hat{p} = \frac{\vec{V} \times \vec{u}_p}{||\vec{V} \times \vec{u}_p||}$$

$$\hat{u} = \frac{\hat{r} \times \hat{p}}{||\hat{r} \times \hat{p}||} \quad \hat{u} = \hat{r} \times \hat{V}$$

$$p_0 = C - \hat{r} + \hat{u}$$
$$p_1 = C + \hat{r} + \hat{u}$$
$$p_2 = C - \hat{r} - \hat{u}$$
Today's Agenda:

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

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

From: Real Shading in Unreal Engine 4, Karis, 2013 (also used in Frostbite).
Loop hoisting:
Take expressions that do not rely on the loop counter outside the loop.

```cpp
define(test)
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.

```csharp
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:

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

Why Do We Care

We can use an AABB to quickly discard objects.
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.
Boxes

Special Case: AABB

AABB: *Axis Aligned Bounding Box*.

Slab test:

Intersect the ray against pairs of planes.

\[ t_{\min} = +\infty, t_{\max} = -\infty \]

\[ t_{\min} = \max(t_{\min}, \min(t_1, t_2)) \]

\[ t_{\max} = \min(t_{\max}, \max(t_1, t_2)) \]

Intersection if: \( t_{\min} < t_{\max} \)

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

\[
t = -\frac{(O \cdot \vec{N} + d) / (\vec{D} \cdot \vec{N})}{ \vec{x}_\text{plane} - O_x / D_x}
\]

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

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

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

\[ d = -P_x = -x_{\text{plane}}, \text{and thus:} \]

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

In pseudo-code:

```c
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:

- Efficiency
- Boxes, AABBs & Groupings
- To Rasterization
- 3D Engine Overview
- 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
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.
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.
INFOGR – Lecture 7 – “Accelerate”

Rasterization

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
- Textures
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
3D Engine

Rendering – Data Overview

camera

world

$T_{\text{camera}}$

$T_{\text{car}1}$

$T_{\text{plane}1}$

$T_{\text{car}2}$

$T_{\text{plane}2}$

car

plane

buggy

wheel

wheel

wheel

wheel

wheel

wheel

turret

turret

dude

dude

wheel

wheel

wheel

wheel

wheel

dude
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).
Rendering – Data Overview

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

- SCREEN
- FRAME BUFFER
- Z-BUFFER
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 10

Lecture 9

Lecture 11

Lecture 14

Lecture 14

P2

P3
Transformations

Rendering a scene graph is done using a recursive function:

```cpp
void SGNode::Render( mat4& M )
{
    mat4 M' = M_local * M;
    mesh->Rasterize( M' );
    for( int i = 0; i < childCount; i++ )
        child[i]->Render( M' );
}
```

Here, matrix concatenation is part of the recursive flow.
Transformations

To transform meshes to world space, we call `SGNode::Render` with an identity matrix.

To transform meshes to camera space, we call it with the inverse transform of the camera.

Remember: the world revolves around the viewer; instead of turning the viewer, we turn the world in the opposite direction.

```cpp
void SGNode::Render(mat4& M)
{
    mat4 M' = Mlocal * M;
    mesh->Rasterize(M');
    for(int i = 0; i < childCount; i++)
        child[i]->Render(M');
}
```
3D Engine

After projection

The output of the projection stage is a stream of vertices for which we know 2D screen positions.

The vertex stream must be combined with connectivity data to form triangles.

‘Triangles’ on a raster consist of a collection of pixels, called fragments.
Today's Agenda:

- Efficiency
- Boxes, AABBS & Groupings
- To Rasterization
- 3D Engine Overview
- Textures
INFOGFR – Computer Graphics
Jacco Bikker & Debabrata Panja - April-July 2017
END OF lecture 7: “Accelerate”

Next lecture: “OpenGL”