WebGL pathtracing
Challenges and benchmark

Thomas Kjeldsen and Peter Trier Mikkelsen

Alexandra Instituttet
thomas.kjeldsen@alexandra.dk

May 22, 2013
Outline

- Alexandra Instituttet
- Motivation
- Demo
- Implementation
- Benchmark
Alexandra Instituttet - Mission

- Not-for-profit GTS institute within IT
- Add value to the Danish Industry
- From research to applications in industry
Alexandra Instituttet - Mission

- Not-for-profit GTS institute within IT
- Add value to the Danish Industry
- From research to applications in industry

- Computer Graphics Lab
  - Interactive visualization (fast, high quality)
  - Accurate simulation of materials → photo realistic images
  - Acceleration (using GPUs)
  - Solving numerical problems
  - Physical simulations (fluids, soft bodies)
Motivation - part 1
Motivation

- Interactive, realistic rendering in a web browser
Motivation

- Interactive, realistic rendering in a web browser
  - E.g. realistic preview of customizable products in a web store
Motivation

- Interactive, realistic rendering in a web browser
  - E.g. realistic preview of customizable products in a web store

Raytracing is computationally expensive (not feasible to implement in JavaScript)

WebGL is a new standard that allows us to access the power of the graphics card
Motivation

- Interactive, realistic rendering in a web browser
  - E.g. realistic preview of customizable products in a web store
- Raytracing is computationally expensive (not feasible to implement in javascript)
Motivation

- Interactive, realistic rendering in a web browser
  - E.g. realistic preview of customizable products in a web store
- Raytracing is computationally expensive (not feasible to implement in javascript)
- WebGL is a new standard that allows us to access the power of the graphics card
Motivation - part 2
Motivation

• Have you ever experienced this situation?
Motivation

• Have you ever experienced this situation?
• You receive an email from you friend

Hey, click on this link to view some cool interactive 3D graphics. Link
Motivation

- Or, Flash, Unity, java, etc.
Motivation

- Or, Flash, Unity, java, etc.
- WebGL is natively supported by modern browsers
Demo

Demo 1
Demo 2
Implementation
Implementation

- WebGL is nearly equivalent to OpenGL ES 2.0
Implementation

- WebGL is nearly equivalent to OpenGL ES 2.0
- Programmable pipeline allows us to do complex calculations per pixel
Implementation

- WebGL is nearly equivalent to OpenGL ES 2.0
- Programmable pipeline allows us to do complex calculations per pixel
- In our case: Execute a ray tracing program for each pixel on the screen

Implementation

Simplified pathtracing algorithm

- Upload triangle data to the graphics memory using textures
  - Launch a ray from the camera through the pixel
    - Intersect the ray with all triangles
    - Record the color at the closest hit point
    - Launch a secondary ray from the hit point
      - Repeat until the ray hits a light source
  - Pixel color = BRDF * cos / pdf * light color
Implementation

Simplified pathtracing algorithm

- Upload triangle data to the graphics memory using textures
- Fragment shader for each pixel

Launch a ray from the camera through the pixel

{  
Intersect the ray with all triangles
  
Record the color at the closest hit point
  
Launch a secondary ray from the hit point
  
Repeat until the ray hits a light source
}

Pixel color = BRDF * cos / pdf * light color
Challenge 1 - long shaders fail to compile

- Some shader compilers unroll loops - in particular on Windows (ANGLE OpenGL to DirectX translation)
Challenge 1 - long shaders fail to compile

- Some shader compilers unroll loops - in particular on Windows (ANGLE OpenGL to DirectX translation)

```plaintext
Launch a ray from the camera through the pixel
{
    Intersect the ray with all triangles
    Record the color at the closest hit point
    Launch a secondary ray from the hit point
    Repeat until the ray hits a light source
}
Pixel color = surface colors * light color
```
Challenge 1 - long shaders fail to compile

- Some shader compilers unroll loops - in particular on Windows (ANGLE OpenGL to DirectX translation)

```
Launch a ray from the camera through the pixel
{
    Intersect the ray with all triangles

    Record the color at the closest hit point

    Launch a secondary ray from the hit point

    Repeat until the ray hits a light source
}
Pixel color = surface colors * light color
```

- Inner loop over ray-triangle intersections $\sim 1000$
Challenge 1 - long shaders fail to compile

- Some shader compilers unroll loops - in particular on Windows (ANGLE OpenGL to DirectX translation)

```cpp
Launch a ray from the camera through the pixel  
{
    Intersect the ray with all triangles
    Record the color at the closest hit point
    Launch a secondary ray from the hit point
    Repeat until the ray hits a light source
}
Pixel color = surface colors * light color
```

- Inner loop over ray-triangle intersections \( \sim 1000 \)
- Outer loop over secondary bounces \( \sim 5 \)
Challenge 1 - long shaders fail to compile

- Some shader compilers unroll loops - in particular on Windows (ANGLE OpenGL to DirectX translation)

```c
Launch a ray from the camera through the pixel
{
    Intersect the ray with all triangles

    Record the color at the closest hit point

    Launch a secondary ray from the hit point

    Repeat until the ray hits a light source
}
Pixel color = surface colors * light color
```

- Inner loop over ray-triangle intersections \(\sim 1000\)
- Outer loop over secondary bounces \(\sim 5\)
- Our shader fails to compile on Windows!
Challenge 1 - long shaders fail to compile

- Solution: Trace each secondary bounce in a separate pass

  - We need to store the hit record between each pass
    - Hit position (3 floats)
    - Ray direction (3 floats)
    - Hit material (1 int)
    - Surface normal (3 floats)
    - Accumulated color (3 floats)

- Unfortunately WebGL only supports a single render target, i.e., we can only transfer four floats between two passes
  - We must encode the hit record to fit in just four floats
Solution: Trace each secondary bounce in a separate pass

We need to store the hit record between each pass

- Hit position (3 floats)
- Ray direction (3 floats)
- Hit material (1 int)
- Surface normal (3 floats)
- Accumulated color (3 floats)
Challenge 1 - long shaders fail to compile

- **Solution:** Trace each secondary bounce in a separate pass
- **We need to store the hit record between each pass**
  - Hit position (3 floats)
  - Ray direction (3 floats)
  - Hit material (1 int)
  - Surface normal (3 floats)
  - Accumulated color (3 floats)

- **Unfortunately WebGL only supports a single render target, i.e., we can only transfer four floats between two passes**
Challenge 1 - long shaders fail to compile

• Solution: Trace each secondary bounce in a separate pass
• We need to store the hit record between each pass
  • Hit position (3 floats)
  • Ray direction (3 floats)
  • Hit material (1 int)
  • Surface normal (3 floats)
  • Accumulated color (3 floats)

• Unfortunately WebGL only supports a single render target, i.e., we can only transfer four floats between two passes
• We must encode the hit record to fit in just four floats
Challenge 2 - acceleration structure

Launch a ray from the camera through the pixel
{
    Intersect the ray with all triangles
    Record the color at the closest hit point
    Launch a secondary ray from the hit point
    Repeat until the ray hits a light source
}
Pixel color = BRDF * cos / pdf * light color

- Linear scaling with number of triangles
Launch a ray from the camera through the pixel
{
    Intersect the ray with all triangles
    Record the color at the closest hit point
    Launch a secondary ray from the hit point
    Repeat until the ray hits a light source
}
Pixel color = BRDF * cos / pdf * light color

- Linear scaling with number of triangles
- Use an acceleration structure
Challenge 2 - acceleration structure

Bounding volume hierarchy

Challenge 2 - acceleration structure

Bounding volume hierarchy


Traversal:
  • Intersect with A
Challenge 2 - acceleration structure

Bounding volume hierarchy


Traversal:

- Intersect with A
  - Descent through B and push C on a stack
Challenge 2 - acceleration structure

Bounding volume hierarchy

Traversals:
- Intersect with A
  - Descent through B and push C on a stack
  - Intersection test with the primitives in the leafs of B

Challenge 2 - acceleration structure

Bounding volume hierarchy

Traversal:
- Intersect with A
- Descent through B and push C on a stack
- Intersection test with the primitives in the leafs of B
- Fetch the node from the top of the stack (C)

Challenge 2 - acceleration structure

Bounding volume hierarchy

Traversal:

- Intersect with A
  - Descent through B and push C on a stack
  - Intersection test with the primitives in the leafs of B
  - Fetch the node from the top of the stack (C)
  - Intersection test with the primitives in the leafs of C

Challenge 2 - acceleration structure

Bounding volume hierarchy - tree traversal

- Shaders are suited for parallel execution of simple tasks
Challenge 2 - acceleration structure

Bounding volume hierarchy - tree traversal

- Shaders are suited for parallel execution of simple tasks
- No support for dynamic memory allocation needed for a stack
Challenge 2 - acceleration structure

Bounding volume hierarchy - tree traversal

- Shaders are suited for parallel execution of simple tasks
- No support for dynamic memory allocation needed for a stack
- Fixed-size stack can be implemented in recent versions of the OpenGL shading language - but not in WebGL
Bounding volume hierarchy - tree traversal

- Shaders are suited for parallel execution of simple tasks
- No support for dynamic memory allocation needed for a stack
- Fixed-size stack can be implemented in recent versions of the OpenGL shading language - but not in WebGL
- We have implemented two stackless BVH traversal algorithms
  
  Laine HPG 2010: Restart Trail for Stackless BVH traversal
  
  Hapala SCCG 2011: Efficient Stack-less BVH Traversal for Ray Tracing
Benchmark
Benchmark setup

- Nvidia GeForce 470GTX
- Xeon E5620 Quad, 2.4 GHz
- Firefox 21 on linux
- 512px x 512px
- 4 secondary bounces
Benchmark - traversal

• Stackless traversal revisits internal nodes
• Shortstack algorithm depends on efficient bitwise operations

![Bar chart showing samples per second for Stack, Stackless (Hapala), and Shortstack (Laine)]
Benchmark - traversal

- Stackless traversal revisits internal nodes

Bar chart showing performance comparison:
- Stack
- Stackless (Hapala)
- Shortstack (Laine)

- OpenGL 3.3
Benchmark - traversal

- Stackless traversal revisits internal nodes
- Shortstack algorithm depends on efficient bitwise operations
Benchmark - multiple passes

![Benchmark Chart]

**Comparison of Samples per Second for Stackless (Hapala) and Shortstack (Laine) between Singlepass and Multipass:**

- **Stackless (Hapala):**
  - Singlepass: Approximately 14 samples per second.
  - Multipass: Approximately 16 samples per second.

- **Shortstack (Laine):**
  - Singlepass: Approximately 10 samples per second.
  - Multipass: Approximately 12 samples per second.
Outlook

• Looking for real-world applications
Outlook

- Looking for real-world applications
- Next version of WebGL will probably be based on OpenGL ES 3.0, hopefully enables support for
  - Bitwise operations (efficient shortstack BVH traversal)
  - Multiple render targets (save hit record between passes)
  - Full array support (stack implementation)
Thank you

- Visit our blog:
  http://cg.alexandra.dk

- Demos:
  http://cg.alexandra.dk/files/pathtracer/?scene=XmasScene
  http://cg.alexandra.dk/files/pathtracer/?scene=motorcycle