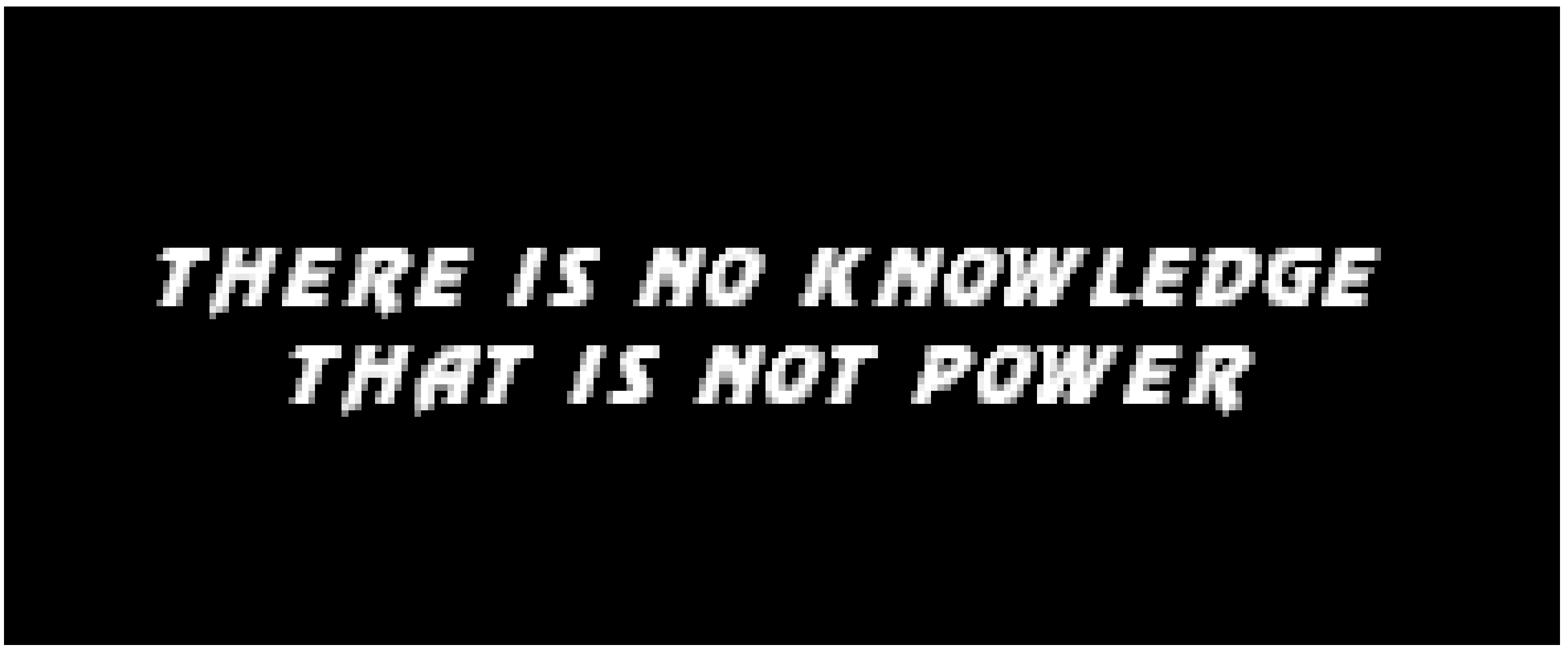
Optimised

Anton Gerdelan gerdela@scss.tcd.ie
Trinity College Dublin

Optimisation Guidelines

- “Don't optimise early”
- Don't optimise unless it's definitely too slow
- Find bottlenecks
- Measure improvements objectively
- Simple and slower > complex and faster

Finding Bottlenecks



Mortal Kombat

Adding FPS Counter

- Hz
- #draw calls per frame
- #uniform updates per frame
- #vertices or triangles per frame
- Run-time feedback on perf of current scene
- **Q. What is the problem with using this system to guide optimisation?**

CPU / Memory / Cache Profiling

- Add your own timers around blocks of interest
- Profiling tools
  - Gprof (GNU)
  - Visual Studio profiler
  - AMD Code Analyst and CodeXL
  - Intel VTune
  - Valgrind (memory leaks etc.), Cachegrind
- Finds heavy functions
- Finds small functions called a huge number of times
- **Q. Why might this all be misleading for us?**

GPU Profiling

- **A. <u>Asynchronous</u> processing on the GPU!**
- apitrace with qapitrace
- VOGL (Valve)
- AMD CodeXL
- NVIDIA Perfkit, PerfHUD, Nsight
- Use OpenGL timer queries http://docs.gl/gl4/glQueryCounter
- **NB Results specific to hardware and driver version!**

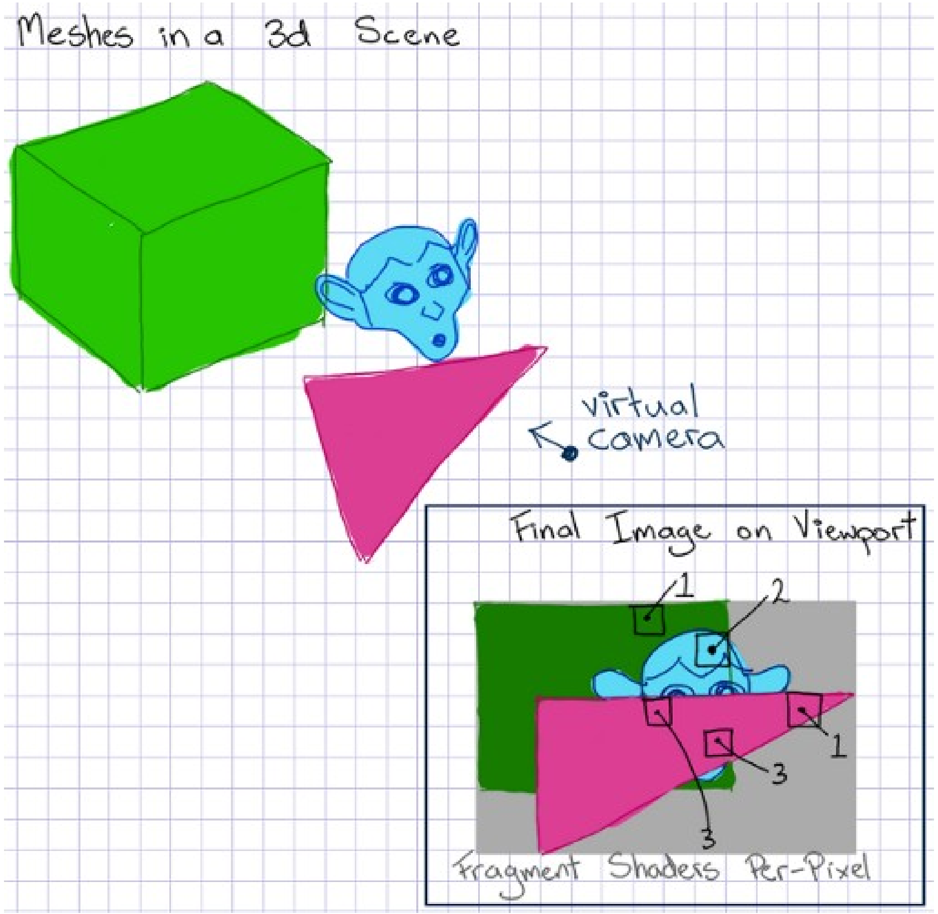
Types of Optimisation

Most of the optimisations that I have made have made performance slightly <u>worse</u> and hugely increased complexity

The best optimisations are the simplest ones

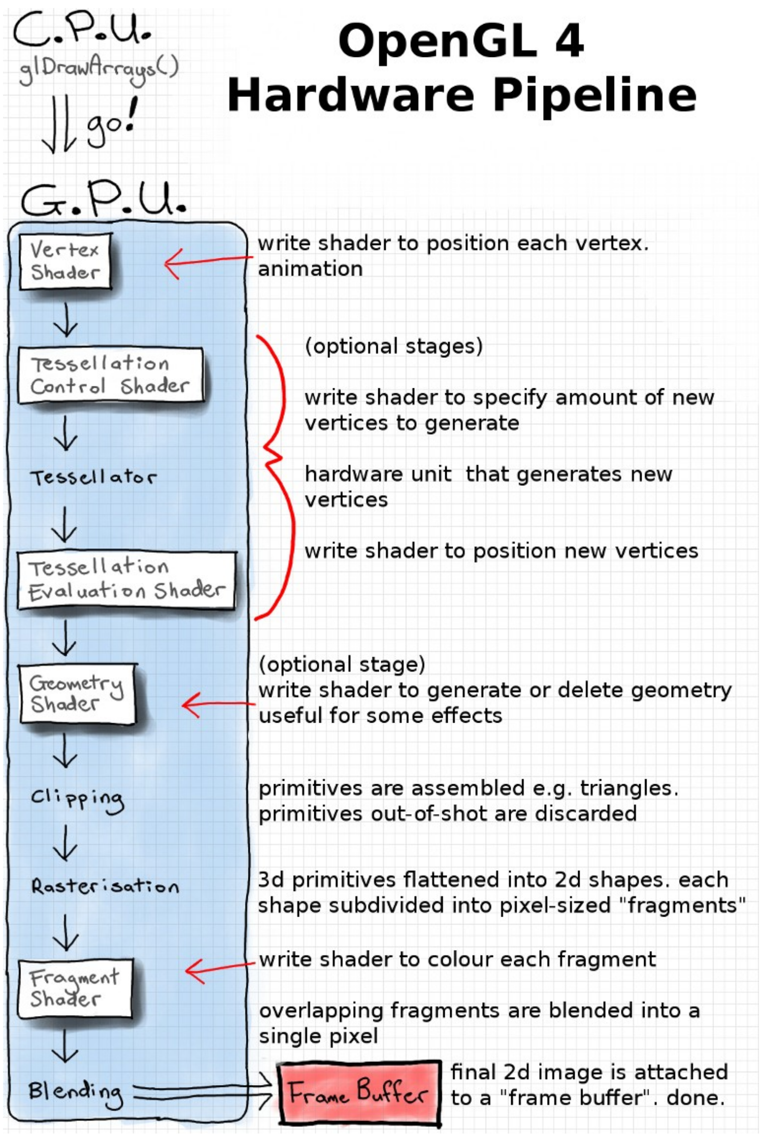
Types of optimisation

- Effectively using built-in or available algorithms
  - Back-face culling
  - Clipping
  - Early z-reject (vs. overdraw)
  - Hardware occlusion culling
- **Q. Why isn't clipping a perfect optimisation?**



Overdraw: >1 fragment per pixel

The Problem with Clipping



- What gets computed before clipping removes out-of-shot geometry?

Types of optimisation

- Scene management
  - Scene graphs
  - Quad-tree, oct-tree
  - Frustum intersections
  - Portal culling
  - Tile-based

Types of optimisation

- Data-oriented design
  - **Memory** is now really slow <u>relative</u> to CPU
  - Arrays and loops instead of lists, object instances, encapsulated data
  - Re-arrange data access to fit better in one cache width
  - Dice: Introduction to DOD
    http://dice.se/wp-content/uploads/Introduction_to_Data-Oriented_Design.pdf

Types of optimisation

- GPU usage
  - Batching geometry
  - Instanced drawing
  - Uniform Buffer Objects
    (share common uniforms between shaders)
  - Hardware tessellation
  - Reduce branching in shaders
  - Shorter fragment shaders
  - **Q. Why are Frag.S more likely the bottleneck than Vert.S?**

Types of optimisation

- CPU usage
  - `sqrt()`
  - Big O complexity: Loops within loops.
  - Threading – not so useful
  - Profiling
  - Small functions used many times
    - inline them (either by keyword or manually)

Uniform Buffer Object (UBO)

- Uniforms shared between many shaders
  - Camera matrices
  - Light position, colour, etc.
- Have an incidental overhead cost when camera moves
  - `glUniformMatrix4fv()` - repeat for all shaders
- UBO
  - bind each shader to a UBO
  - update a single buffer with camera matrices
  - shaders then refer to the same memory for the uniforms
- Short example in my tutorial book

Batching

- Many small, static objects in scene that use same shader, texture, etc.
- Each object requires a separate draw call
- Not making good use of GPU parallelism
- Combine into fewer, larger objects
- Art or pre-processing code into VBOs
- Balancing act with other optimisations
- Nvidia “Batch, batch, batch” (GDC ...~2007?)
  http://www.nvidia.com/docs/IO/8228/BatchBatchBatch.pdf

Hardware Instancing

- Similar to batching except don't combine
- Less memory used
- Can move independently *via* array of uniforms
- glDrawArraysInstanced(…,num_instances)
- To move each to a separate position

```
uniform mat4 M[MAX_INSTANCES];

gl_Position = P * V * M[gl_InstanceID] * v;
```

Spatial Data Structures

- **Create structure**: Divide 3d scene up somehow
  - Lists of visible items in nodes or for different views
  - Reduce list of items to **traverse** for visibility
- **Traverse** based on camera pos,angle for visibility
- **Test** items against camera frustum for **visibility**
  - Extract frustum planes
- Reference: “Real Time Rendering”, chapter 14

Bounding Volumes

- Approximate mesh by a bounding sphere, or box.
  - Sphere
  - AABB – axis-aligned bounding box
  - OBB – oriented bounding box
- Easier to test against than all triangles in a mesh

Binary Space Partitioning (BSP)

- Recursively sub-divide 3d space in half by a plane
  - 1. In front of current plane
  - 2. Behind current plane
  - Creates a **sorted** front-to-back **binary tree**
  - Used in Doom before depth buffer existed to sort by depth
  - When traversing each sector knows if it is in-front or behind of A…etc.

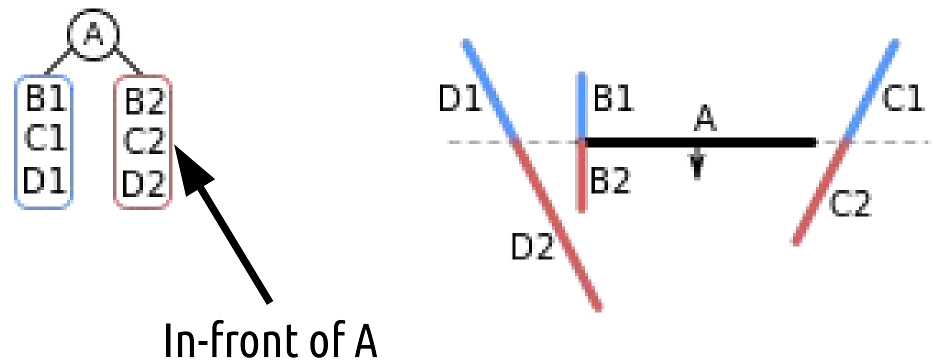
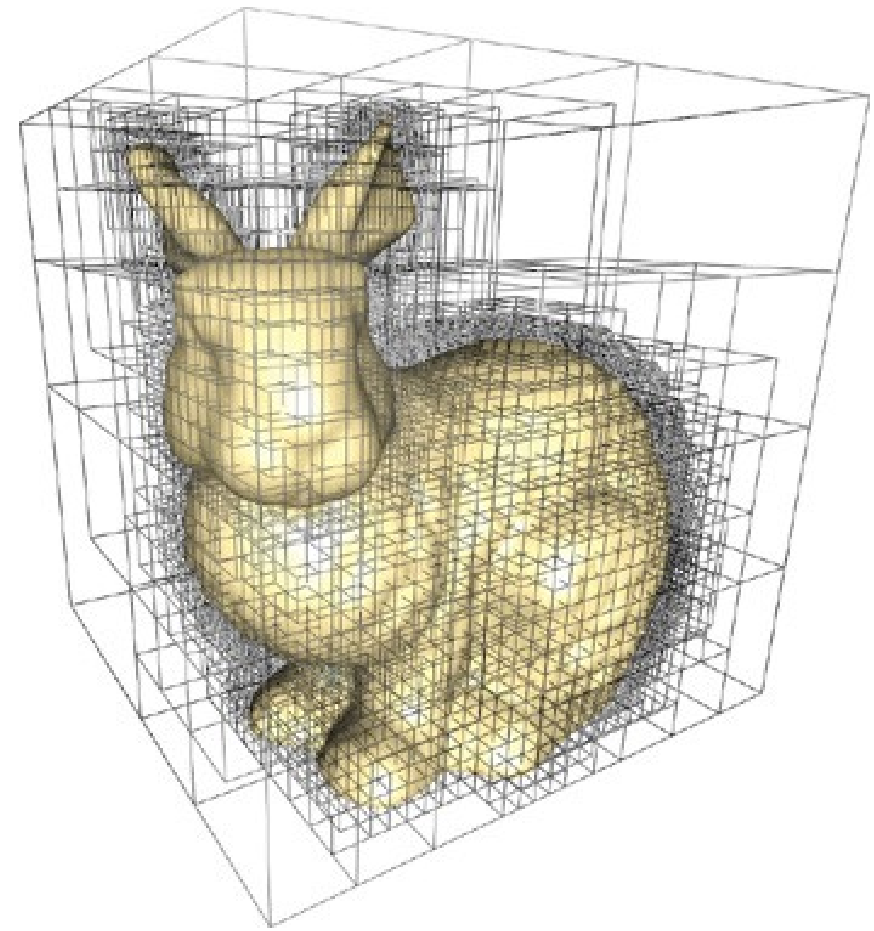


Diagram (also a nice article) on Wikipedia

Oct-Tree, Quad-Tree, k-d Tree

- Creating:
  - Box covering entire scene
  - If >1 object in box, split into 4 boxes
  - Recurse
- Traversing:
  - test main box for visibility
  - recurse with sub-boxes
- Draw all items in visible boxes

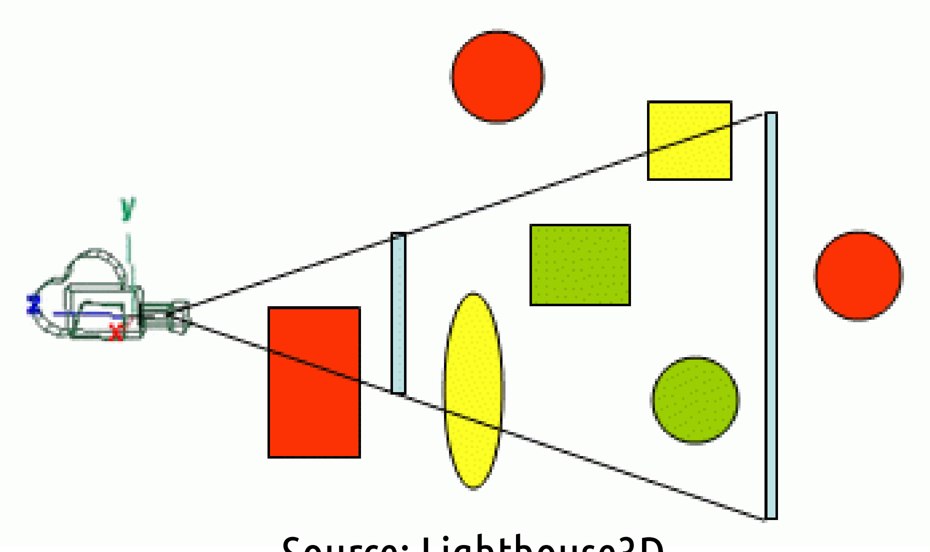


Oct -Tree *Source: GPU Gems 2, Ch.37*

Frustum Cull: A Test for Visibility

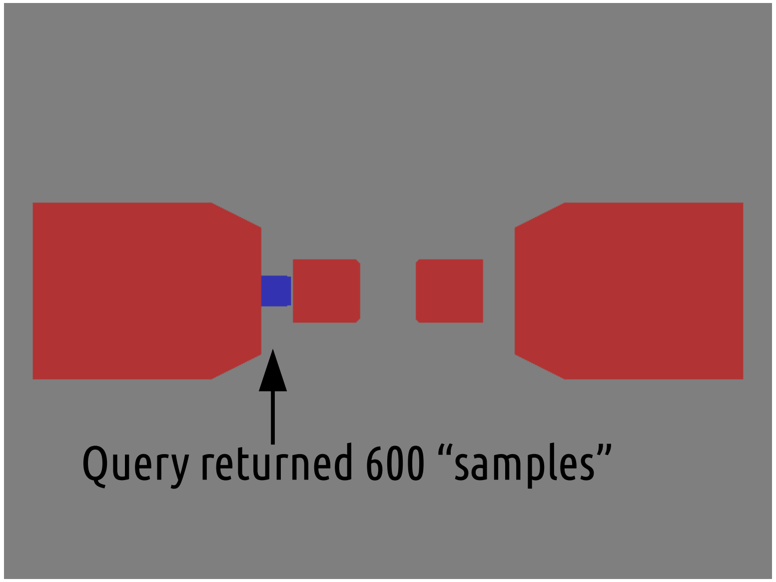
- Extract planes from frustum shape
- Test all bounding volumes (or quad-tree boxes) against frustum planes
- For each plane:
  - dot product of plane's inward normal with
  - distance to a point on the object
  - If < 0.0 → FAIL TEST

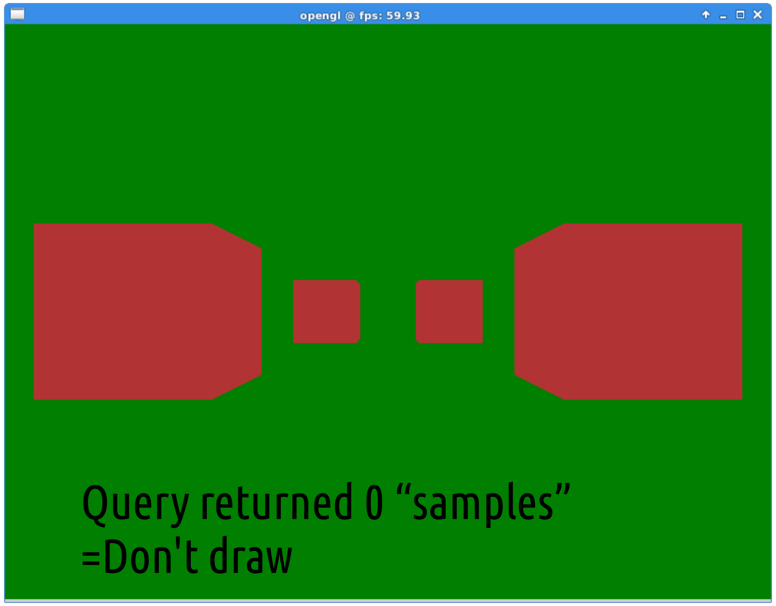
- **Q. What unseen geometry is <u>still not removed</u> despite oct-tree and a frustum?**



Source: Lighthouse3D

Hardware Occlusion Culling





- Split objects into
  - Big "occluders"
  - Small "occludees"
- Draw all occluders
- BeginQuery()
- Draw bounding box of each occludee
- EndQuery()
- Before drawing occludees, check #samples visible in its query

Summary

- Find bottlenecks first
- Determine simplest improvements first
- i.e. Too many draws
  - Can I just test if objects are behind camera?
  - Is a spatial structure like quad-tree appropriate for my scene?
  - Is a frustum culling function appropriate for my camera?
  - Would batching or instancing be a good idea?
- Are my bottlenecks and rendering rate actually acceptable? i.e. if so don't touch it

Advanced Data Buffers

Portal Culling