Optimisation

CS7GV3 – Real-time Rendering
Introduction

• Talk about lower-level optimization
• Higher-level optimization is ”better algorithms”
  • Example: not using a spatial data structure vs. using one
• After that pipeline optimization
  • A cookbook of tricks and techniques
• One way of optimizing performance of the geometry stage is triangle strips and meshes.
Triangle strips and meshes

• What’s the point?
• We can send a triangle to the rendering pipeline by sending 3 vertices
  • The triangle after uses another 3 vertices, and so on
• Not efficient...
• Triangle strips and meshes are better
  • Uses locality in geometrical data

Based on Tomas Akenine-Möller presentation slides http://www.realtimerendering.com/
Example image of model consisting of triangles

• Note that triangles in general are connected to other triangles
• Example:
Triangle strips

- Without strips: 8 triangles * 3 vertices = 24 vertices
- With strips: use 1 vertex per triangle instead of 3!

What we send to graphics hardware:
- Startup cost: $v_0$, $v_1$ then $v_2$ (T0), $v_3$ (T1), $v_4$ (T2), $v_5$ (T3), $v_6$ (T4), $v_7$ (T5), $v_8$ (T6), $v_9$ (T7).
- 9 vertices $\Rightarrow$ 100*9/24 = 37.5% or 9/8 = 1.125 verts/tri

Based on Tomas Akenine-Möller presentation slides http://www.realtimerendering.com/
Triangle strips

• 9 vertices instead of 24 →
  • 100*9/24= 37.5% of data
  • 9/8=1.125 vertices/tri

• We can expect the geometry stage to run almost 3 times faster!
Swaps in triangle strips

• What can we do for this case?
  
  • Implement a swap!
  • Startup cost: \( v_0, v_1 \) then
    - \( v_2 (T0) \)
    - \( v_3 (T1) \)
    - \( v_2, \)
    - \( v_4 (T2) \)
    - \( v_5 (T3), v_6 (T4) \)
  
  • Degenerate triangle (0-area): \( v_2, v_3, v_2 \)

Based on Tomas Akenine-Möller presentation slides http://www.realtimerendering.com/
Swaps...

- Cost of degenerate triangle is 1 extra vertex
- Still cheaper than restarting triangle strip
- This example: 8 sent vertices / 5 triangles = 1.6 vertices/triangle
- Restarting triangle strip costs more:
  - 4 vertices (2 triangles) +
  - 5 vertices (3 triangles)

Based on Tomas Akenine-Möller presentation slides http://www.realtimerendering.com/
Swaps...

- Also, hardware determines degenerate triangles efficiently and skips these
- Can use swaps to connect non-connected triangles too
  - The idea is to avoid API call overhead
  - Hardware needs caches for this to be efficient

Send these vertices: 0,1,2,3, 3,4, 4,5,6,7
- 10 vertices (sending as 2 strips: 8 vertices)
- If 3 and 4 are cached, then 8 vertices
How to create triangle strips from a 3D model?

• Manually
  • only doable for small models, and not fun...

• Or make your own program
  • Need to know triangle’s neighbors (page 278 for an algorithm for this, http://www.realtimerendering.com/)
  • It’s quite simple to make a working strip-creator
  • To make a really good one is more work
Triangle meshes

- Only specify each vertex once in a buffer
- Then send (e.g., 16 bit) indices to identify vertices in buffer
- Example: make a buffer of 100 unique vertices
  - A triangle is sent as 3 indices into this buffer: 97, 5, 32
- An interesting thing is that we can send those indices in the triangle strip form
Vertex arrays (OpenGL) vertex buffers (DirectX)

• Triangle strips and vertex buffers are (often) fastest way to send triangles today
• Store vertex data sequentially memory
• Just pass the pointer to the API
• The API itself then copies the data from memory
• Avoids expensive copying
• Can specify triangles as strips, single triangles, etc, etc.
• Expensive to switch between vertex buffers, so use the technique with 4 degenerate triangles (presented previously) to connect disconnected triangles

Based on Tomas Akenine-Möller presentation slides http://www.realtimerendering.com/
Pipeline Optimization

• "Premature optimization is the root of all evil", Donald Knuth
• Make it run first, then optimize
• But only optimize where it makes any difference
• This lecture deals with finding bottleneck, and how to make them go away (to another place, which we can optimize)

Based on Tomas Akenine-Möller presentation slides http://www.realtimerendering.com/
Pipeline optimization

• Try to make algorithmic changes first
  • Examples: culling techniques, spatial data structures for faster ray tracing.

• When you’re (nearly) satisfied:
  • Turn to optimization techniques
  • That is, do this last!

• Conceptual pipeline:
  • Application
  • Geometry
  • Rasterizer

• We’ll look into techniques for optimizing each stage
Repeat: The pipeline

• Compare to:
  • Oil pipeline
  • Car building
  • Skilift

• Stages execute in parallel
• Always one stage (slowest) that is the bottleneck of the pipeline
• The bottleneck determines throughput (i.e., maximum speed)
• The bottleneck is the ”average bottleneck over a frame”
  • Cannot measure ”intra-frame” bottlenecks easily

• Most important in optimization: find bottleneck
• Then optimize that stage!
Finding the bottleneck

- Two bottleneck location techniques

- First technique:
  - Make a certain stage work less
  - If performance is the better, then that stage is the bottleneck

- Second technique (similar):
  - Make the other two stages work less or (better) not at all
  - If performance is the same, then the stages not included above is the bottleneck

- Complication: the bus between CPU and graphics card may be bottleneck (not a typical stage)

Based on Tomas Akenine-Möller presentation slides http://www.realtimerendering.com/
Application (CPU) stage is bottleneck?

• Application (CPU) is bottleneck?
  • Use `top` on Unix systems, TaskManager on Windows.
  • If your app uses (near) 100% of CPU time, then it is very likely that the application is the bottleneck
  • Make CPU do less work (e.g., turn off collision-detection)

• Your program is then ”CPU-bound”, or ”CPU-limited”

Based on Tomas Akenine-Möller presentation slides http://www.realtimerendering.com/
Geometry stage is bottleneck?

• Trickiest stage to test

• Because, if you change workload here, then the workload of application and rasterizer often changes as well

• However, number of light sources affects geometry stage

• So, disable light sources
  • If performance goes up, then geometry is bottleneck, and program is ”transform-limited”

• Another test: enable all light sources
  • If performance stays the same, then geometry stage is NOT the bottleneck

• Another: test CPU and rasterizer instead

Based on Tomas Akenine-Möller presentation slides http://www.realtimerendering.com/
Rasterizer stage is bottleneck?

• The easiest, and fastest to test
• Simply, decrease the size of the window you render to
• This does not change the workload for the application or the geometry stage
• But the rasterizer needs to fill fewer pixels
• If the performance goes up, then the program is ”fill-limited” or ”fill-bound”
• You can also make the rasterizer work less
  • Turn of texturing, fog, blending, depth buffering etc (if your architecture have performance penalties for these)

Based on Tomas Akenine-Möller presentation slides http://www.realtimerendering.com/
NVIDIA® Nsight™ Graphics (Range-Based GPU Profiling)

https://developer.nvidia.com/nsight-graphics
NVIDIA® Nsight™ Graphics (GPU Trace)

https://developer.nvidia.com/nsight-graphics
NVIDIA® Nsight™ Graphics (Shader Performance Comparison)

https://developer.nvidia.com/nsight-graphics
Let’s turn to Optimization

Single-most important ”trick”:

KNOW YOUR ARCHITECTURES!

Based on Tomas Akenine-Möller presentation slides http://www.realtimerendering.com/
Optimization

• Optimize the bottleneck stage
• Only put enough effort, so that the bottleneck stage moves.
• Did you get enough performance?
• Yes! Quit optimizing
• NO! Continue optimize the new bottleneck (if it has moved)
Illustration of optimization

- Height of bar is equal to time it takes for that stage for one image (frame)
- Highest bar is bottleneck
- After optimization: bottleneck has moved to APP
- No use in optimizing GEOM, turn to optimizing APP instead

Based on Tomas Akenine-Möller presentation slides http://www.realtimerendering.com/
Application optimization: Some general ideas

• Two major ways:
  • Efficient code
    • Use fewer instructions
    • Use more efficient instructions
    • Recode algorithmically
  • Efficient memory access pattern

• But first:
  • Turn on optimization flags in compiler
  • Code profilers are very useful
    • Helps finding the places where most of the time is spent
    • It is (often) useless to optimize code that uses 1% of the time, or code only used in preprocessing

• This is time consuming stuff
  • Detailed code optimization & profiling out of scope...
  • Don’t do for your project...

Based on Tomas Akenine-Möller presentation slides http://www.realtimerendering.com/
Various code optimization tricks

• SIMD instructions sets are perfect for vector operations
  • Often 2-4 operations are performed in parallel
  • SSE, SSE2, 3DNow! are examples

• Division is an expensive operation
  • Between 4-39 times slower than most other instructions
  • Example of good usage: normalize a vector, \( \mathbf{v} \)
    • Instead of \( \mathbf{v} = (v_x/d, v_y/d, v_z/d) \)
    • \( d = \mathbf{v} \cdot \mathbf{v} \)
    • \( i = 1/d \)
    • \( \mathbf{v} = \mathbf{v} \cdot i \)
  • On some CPUs there are low-precision reciprocal (1/x) and square root reciprocal (1/sqrt(x))
More code optimization tricks

• Conditional branches are often expensive
  • If you can avoid if-cases, to that
  • However, sometimes branch prediction on CPUs work remarkably well

• Math functions, such as, sin, cos, tan, sqrt, exp, etc are expensive
  • Sometimes a rough approximation is sufficient
  • If so, use first few terms in Taylor series

• Inline code is good (avoids function calls)

• float (32 bits) is faster than double (64 bits)
  • Also less data is sent down the pipeline

Based on Tomas Akenine-Möller presentation slides http://www.realtimerendering.com/
More code optimization tricks

• Try different strategies
  • Compiler optimization is hard to predict
  • E.g., sometimes \(--\text{counter};\) is faster than \(\text{counter}--;\)

• Use \texttt{const} in C and C++ to help to compiler with optimization

• These often incur a overhead
  • Dynamic casting (C++)
  • Virtual methods
  • Inherited constructors
  • Passing \texttt{struct}'s by value
Memory optimization

• Be aware of memory hierarchy (caches) in a modern computer
• Bad memory access pattern can ruin performance
• This is not really about using less memory, though that can help...
More memory optimization tricks

• If memory is accesses sequentially in program, then store data in that order in memory
  • Tex coords #0, position #0, tex coords #1, position #1, tex coords #2, position #2, etc.

• Cache prefetching is nice
  • But hard to control

• `malloc()` and `free()` may be slow
  • Sometimes better to allocate memory to a pool at startup of program
More memory optimization tricks

• Align data with size of cache line
  • Example: the cache line size if 32 bytes
  • Now, assume that it takes 30 bytes to store a vertex
  • Padding with another 2 bytes → 32 bytes
  • This will very likely give better cache performance

• Following pointers (linked list) is expensive (if memory is allocated arbitrarily)
  • Does not use coherence well that cache usually exploits
  • That is, the address after the one we just used is likely to be used soon
  • Paper by Smits on ray tracing shows this
Geometry stage optimization

• Geometry stage does per-vertex ops
• Best way to optimize:
  • Triangle strips!!!
• Lighting optimization
  • Spot lights expensive, point light cheaper, directional light cheapest
  • Disable lighting if possible
  • Use as few light sources as possible
  • If you use $1/d^2$ falloff, then if $d>10$ (example), disable light
Geometry stage optimization

• Normals must be normalized to get correct lighting
  • Normalize them before, and disable normalizing if possible

• Lighting can be computed for both sides of a triangle
  • If not needed, disable

• If light sources are static with respect to geometry, and material is only diffuse
  • Then precompute lighting on CPU
  • Send only precomputed colors (not normals)
Rasterizer stage optimizations

• Rasterizer stage does per-pixel ops
• Simple way: turn on backface culling if possible
• Turn off Z-buffering if possible
  • Example: after screen clear, draw large background polygon
  • Using polygon-aligned BSP trees
• Draw in front-to-back order
• Try disable features: texture filtering mode, fog, blending, multisampling

Based on Tomas Akenine-Möller presentation slides http://www.realtimerendering.com/
Rasterizer stage optimization

• To make rasterization faster, need to rasterize fewer (or cheaper) pixels
  • Make window smaller
  • Or render to a smaller texture, and then enlarge texture onto screen

• Depth complexity is number of times a pixel has been written to
  • Good for understanding behaviour of application
  • Examples follows
Depth complexity

Based on Tomas Akenine-Möller presentation slides http://www.realtimerendering.com/
Overall optimization General techniques

• Reduce number of primitives (polygon simplification algorithms can do this)
• Preprocess geometry and data for your particular architecture
• Turn off features not in use
• Even if no visible effect can be seen, there may be a performance drop
• Examples:
  • Depth buffering
  • Blending
  • Fog
  • Texturing

Based on Tomas Akenine-Möller presentation slides http://www.realtimerendering.com/
Overall optimization

• Minimize state changes by grouping objects after state
  • Example: objects with the same texture should be rendered together
• If all pixels always are drawn, then avoid color buffer clear
• Frame buffer reads are expensive

Based on Tomas Akenine-Möller presentation slides http://www.realtimerendering.com/
Balancing the pipeline

• The bottleneck stage sets the frame rate
• Therefore, the two other stages will be idle for some time
• Also, to sync with monitor, there might be idle time for all stages (see)
• Exploit this time to make quality of images better if possible

Based on Tomas Akenine-Möller presentation slides http://www.realtimerendering.com/
Balancing pipeline, how?

- Increase number of triangles (affects all stages)
- More lights, more expensive (geometry)
- More realistic animation, more accurate collision detection (application)
- More expensive texture filtering, blending, etc. (rasterizer)
- If not fill-limited, increase window size
- Note: there are FIFOs between stages (and at many other places too) to smooth out idleness of stages
Multiprocessing

• Turn to if application is bottleneck, and if you can afford it
• Two major ways:
  • Multiprocessor pipelining
  • Parallel processing
In summary

• Only do pipeline optimization after you have implemented good algorithms!
• This is something one does for products that should be shipped
• Don’t spend too much time on this for your project (unless you really really want to)
• However, most often good to use triangle strips!