Game Technology

Optional Lecture R – 23.01.2016
Memory, GPUs and Memory on GPUs

Dr.-Ing. Florian Mehm
Dipl-Inf. Robert Konrad

Prof. Dr.-Ing. Ralf Steinmetz
KOM - Multimedia Communications Lab

This is bad

```
void Graphics::setFloats(ConstantLocation location, float* values, int count) {
    if (location.shaderType == -1) return;
    int dx9count = (count + 3) / 4;
    if (dx9count == count / 4) {
        if (location.shaderType == 0) device->SetVertexShaderConstantF(location.reg.regindex, values, dx9count);
        else device->SetPixelShaderConstantF(location.reg.regindex, values, dx9count);
    }
    else {
        auto data = new float[dx9count * 4];
        memcpy(data, values, sizeof(float)*count);
        if (location.shaderType == 0) device->SetVertexShaderConstantF(location.reg.regindex, data, dx9count);
        else device->SetPixelShaderConstantF(location.reg.regindex, data, dx9count);
        delete[] data;
    }
}
```

malloc / new

malloc / new in C/C++

- Requests memory from the operating system
- Fragments memory
- Slow (and sometimes super slow)
- Unreliable (and more unreliable with more fragmentation)
- Works differently in subtle details on every system

new in Java

- Usually no OS memory request
- Automatic defragmentation
- Sometimes optimized away
- More stress for the Garbage Collector

The slowness of malloc

- System calls
- Has to manage a list of memory blocks
  - a = malloc(100);
  - b = malloc(100);
  - free(a);
  - a = malloc(150);
  - -> fragmentation
    - Managing the memory blocks becomes more and more complicated

malloc in practice

Heavy use leads to a general performance hit
plus performance decreases further with time

- Unregular, hard to locate frame drops
- Especially bad for VR

This is not as bad

```
void Graphics::setFloats(ConstantLocation location, float* values, int count) {
    if (location.shaderType == -1) return;
    int registerCount = (count + 3) / 4; // round up
    if (registerCount == count / 4) { // round down
            if (location.shaderType == 0) device->SetVertexShaderConstantF(location.reg.regindex, values, registerCount);
            else device->SetPixelShaderConstantF(location.reg.regindex, values, registerCount);
    }
    else {
            float* data = (float*)alloca(registerCount * 4 * sizeof(float));
            memcpy(data, values, count * sizeof(float));
            if (location.shaderType == 0) device->SetVertexShaderConstantF(location.reg.regindex, data, registerCount);
            else device->SetPixelShaderConstantF(location.reg.regindex, data, registerCount);
    }
}
```

Dynamic Stack Allocation

Syntax

- void* data = alloca(size_in_bytes);
- // or
- float data[some_non_const_var];
- frees automatically when out of scope

Has some problems

- Doesn't work everywhere
- Stack size very restricted
- Prevents some compiler optimizations

Why are stack allocations fast?

Stack Allocation

- All memory is pre-allocated
- The actual stack-allocation is just a pointer += allocation_size
- Optimized for specific access patterns
  - No fragmentation

Manual Dynamic Stack Allocation

TECHNISCHE UNIVERSITÄT DARMSTADT

Stack Allocation

- Preallocate: At program start 1x void* memory = malloc(large_size);
- Set a pointer to memory: void* mem_pointer = memory;
- Allocation:
  whatever* data = (whatever*)mem_pointer;
  mem_pointer += sizeof(whatever);
- Deallocation: mem_pointer -= sizeof(whatever);

Stack Allocation Example Usage

TECHNISCHE UNIVERSITÄT DARMSTADT

Load level 1

- Stack-allocate all assets
- Run level

Load level 2

- Move stack pointer back
- Stack-allocate all assets
- Run level

…

Restrict sizes for less fragmentation

TECHNISCHE UNIVERSITÄT DARMSTADT

For example a buffer just for images

- Keep a list of slots that fit 1/4/16 images of certain sizes
- Deallocation: Mark the slot as free
- Allocation: Walk the list, use the first free slot

Cache Structures

Similar as before but

- Restrict size
- Kick out older data

Example: Megatextures in Rage or Trials Fusion

Linked Lists, Trees,…

TECHNISCHE UNIVERSITÄT DARMSTADT

Can usually use fixed sizes

Don't mix with other data (element sizes are typically tiny)

Best solution: Avoid completely

- Allocation problems
- Cache misses
- Mostly not worth it, please profile

Implement standard allocation

TECHNISCHE UNIVERSITÄT DARMSTADT

(or use a lib)

Pros:

- Full control
- Always the same behavior

Cons

- Probably slower than system allocation

Allocation functions

new

- Calls malloc
- Then calls constructor
- Can be redefined (placement new)

malloc

- Requests memory from the OS

LPVOID VirtualAlloc(lpAddress, dwSize, flAllocationType, flProtect);

- Allocate memory pages in Windows (dwSize is rounded up)
- Provide base address (great for debugging)
- Many more options…

Vulkan / Direct3D 12 / Metal

TECHNISCHE UNIVERSITÄT DARMSTADT

“Direct3D 12 is new territory, for the inquisitive expert to explore.”

Vulkan / Direct3D 12 / Metal

New low level graphics APIs

- Direct3D 12 -> Windows, Xbox One
- Metal -> iOS, OSX
- Vulkan (open standard) -> Windows (with vendor drivers), Android, Linux

Mostly no new features

Designed to solve specific performance problems

Problems: Draw Calls are Expensive

DrawTriangles,…

Internally

- Maybe recompiles shaders

- Commands are converted into internal command lists format
  - No control over when and where that happens
  - Typically not multithreaded

Shader recompiles

GPU shader representation

- Can depend on render state (blend modes, depth tests,…)

Typical driver strategy

- Compile on first use with specific render state and cache

Command Lists

GPUs contain a command processor

- Commands ala
  - set program
  - draw triangles
  - …

Different for each GPU

- API calls are converted into command lists at some point
- Driver usually runs one additional thread for all work like that
- Work happens implicitely at undefined points in time
- Work reuse might or might not happen

Problems: Draw Calls are Expensive

TECHNISCHE UNIVERSITÄT DARMSTADT

DrawTriangles,…

Internally

- Maybe recompiles shaders
- -> PipelineState objects
  - Packs all shaders and all render states
  - Compiles only triggered manually
- Commands are converted into internal command lists format
  - No control over when and where that happens
  - Typically not multithreaded
- -> Compose command lists manually
  - Trigger conversion to internal format manually
  - Pure user mode operation, can run on any thread

Problems: Automatic Buffers

Example: Particle rendering

- Create a large vertex buffer
- Fill with particle data (positions, color,…)
- When buffer is full
  - Draw call
  - Restart at the beginning

Problems: Automatic Buffers

Example: Particle rendering

- Create a large vertex buffer
- Fill with particle data (positions, color,…)
- When buffer is full
  - Draw call
  - Restart at the beginning <- Wait for previous draw call to finish?
    Allocate multiple vbuffers internally and switch?

Problems: Automatic Buffers

TECHNISCHE UNIVERSITÄT DARMSTADT

Example: Particle rendering

- Create a large vertex buffer
- Fill with particle data (positions, color,…)
- When buffer is full
  - Draw call
  - Restart at the beginning <- Wait for previous draw call to finish?

    Allocate multiple vbuffers internally and switch?

    Heuristic based driver decisions,
    optimized for big game releases.

Manual Buffers

Buffers are handled manually

- Buffers can be created in write-combined memory (for PCIe reads) or GPU memory
  - And on some systems it‘s all the same (UMA), which can also be used
- Work with GPU pointers
- Events for completed draw calls

Textures are still special

- Require swizzling for good performance

Simultaneous Command Lists

Multiple command lists at the same time

- Some modern GPUs support this
- Make use of underused ressources
  - Typically run graphics shaders and unrelated compute shaders

Shader constants

(aka uniforms)

Super complex in D3D12

- Can put some data into command lists

Mostly traditional in Metal

Vulkan AFAIK like D3D12

Modern Graphics in Kore

TECHNISCHE UNIVERSITÄT DARMSTADT

Does not yet expose a modern style API

Does include a D3D12 and a Metal backend

- Works on some assumptions that can easily break
  - Vertex buffers are expected to be reused x times
- Feel free to look around in the sources

Waiting for Vulkan