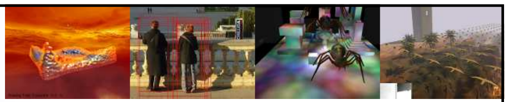


OpenGL Buffer Transfers

Patrick Cozzi
University of Pennsylvania
CIS 565 - Spring 2012

Drawing

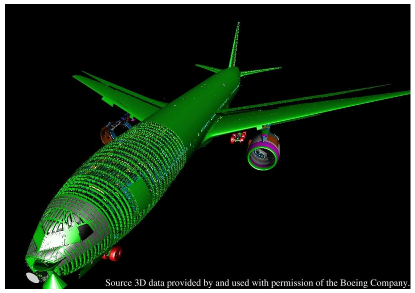
- It doesn't matter if we're using:

OpenGL nVIDIA CUDA OpenCL

- Efficiently transferring data between the CPU and GPU is critical for performance.

How many vertices per second do we need?

Drawing


Source 3D data provided by and used with permission of the Boeing Company.

Boeing 777 model: ~350 million polygons

Image from http://graphics.cs.uni-sb.de/MassiveRT/boeing777.html

Drawing

Procedurally generated model of Pompeii: ~1.4 billion polygons

Image from http://www.vision.ee.ethz.ch/~pmueller/wiki/CityEngine/Documents

Buffer Objects

- *Array buffers* – store vertex attributes
- *Element buffers* – store indices
- Stored in driver-controlled memory, not an array in our application
- Provide hints to the driver about how we will use the buffer

Buffer Objects

```
GLuint vbo;
GLfloat* vertices = new GLfloat[3 * numberOfVertices];

glGenBuffers(1, &vbo);

glBindBuffer(GL_ARRAY_BUFFER_ARB, vbo);

glBufferData(GL_ARRAY_BUFFER_ARB, numberOfBytes, vertices, GL_STATIC_DRAW_ARB);
// Also check out glBufferSubData

delete [] vertices;

glDeleteBuffers(1, &vbo);
```

Buffer Objects

```
GLuint vbo;
GLfloat* vertices = new GLfloat[3 * numberOfVertices];

glGenBuffers(1, &vbo);

glBindBuffer(GL_ARRAY_BUFFER_ARB, vbo);

glBufferData(GL_ARRAY_BUFFER_ARB, numberOfBytes, vertices, GL_STATIC_DRAW_ARB);
// Also check out glBufferSubData

delete [] vertices;

glDeleteBuffers(1, &vbo);
```

Buffer Objects

```
GLuint vbo;
GLfloat* vertices = new GLfloat[3 * numberOfVertices];

glGenBuffers(1, &vbo);

glBindBuffer(GL_ARRAY_BUFFER_ARB, vbo);

glBufferData(GL_ARRAY_BUFFER_ARB, numberOfBytes, vertices, GL_STATIC_DRAW_ARB);
// Also check out glBufferSubData

delete [] vertices;

glDeleteBuffers(1, &vbo);
```

Buffer Objects

```
GLuint vbo;
GLfloat* vertices = new GLfloat[3 * numberOfVertices];

glGenBuffers(1, &vbo);

glBindBuffer(GL_ARRAY_BUFFER_ARB, vbo);

glBufferData(GL_ARRAY_BUFFER_ARB, numberOfBytes, vertices, GL_STATIC_DRAW_ARB);
// Also check out glBufferSubData

delete [] vertices;

glDeleteBuffers(1, &vbo);
```

Copy from application to driver-controlled memory. `GL_STATIC_DRAW` should imply video memory.

Buffer Objects

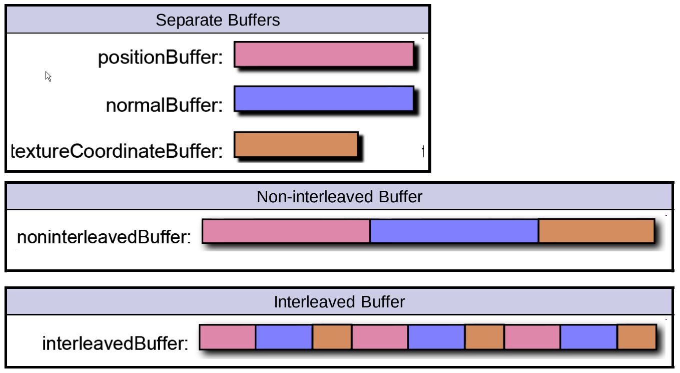
```
GLuint vbo;
GLfloat* vertices = new GLfloat[3 * numberOfVertices];

glGenBuffers(1, &vbo);

glBindBuffer(GL_ARRAY_BUFFER_ARB, vbo);

glBufferData(GL_ARRAY_BUFFER_ARB, numberOfBytes, vertices, GL_STATIC_DRAW_ARB);
// Also check out glBufferSubData

delete [] vertices;

glDeleteBuffers(1, &vbo);
```

- Does `glBufferData` block?
- Does `glBufferSubData` block?

Buffer Objects

- Usage Hint
  - *Static*: 1-to-n update-to-draw ratio
  - *Dynamic*: n-to-m update to draw (n < m)
  - *Stream*: 1-to-1 update to draw
- It's a hint. Do drivers take it into consideration?
  - GL_ARB_debug_output tells us where the buffer is stored

Layouts



Images from www.virtualglobebook.com

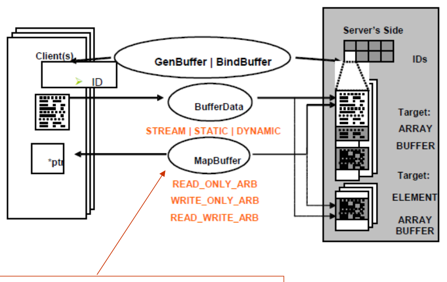
Layout Tradeoffs

- Separate Buffers
  - Flexibility, e.g.:
    - Combination of static and dynamic buffers
    - Multiple objects share the same buffer
- Non-interleaved Buffer
  - How is the memory coherence?
- Interleaved Buffer
  - Faster for static buffers
    - Proportional to the number of attributes
- Hybrid?

Vertex Throughput Tips

- Optimize for the *Vertex Cache*
- Use smaller vertices
  - Use less precision, e.g., `half` instead of `float`
  - Pack, then unpack in vertex shader
  - Derive attributes or components from other attributes
  - How many components do you need to store a normal?

Buffer Objects



Map a pointer to driver-controlled memory
• Also map just a subset of the buffer

Image from http://developer.nvidia.com/object/using_VBOs.html

DMA

- *DMA* – *D*irect *M*emory *A*ccess
  - Asynchronously transfer buffer between CPU and GPU
  - Asynchronous with respect to the CPU, not always the GPU
  - How many copies are made?

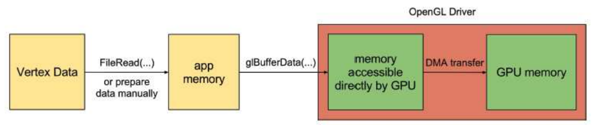
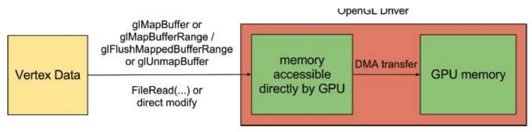


Image from http://www.openglinsights.com/

Buffer Mapping

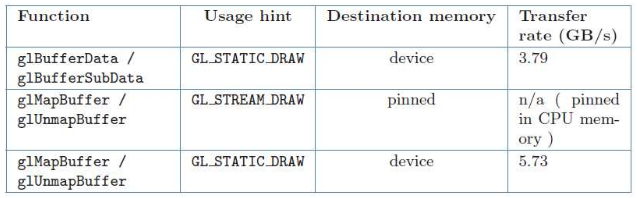
- Use `glMapBuffer`, `glUnmapBuffer`, and friends to save a copy


OpenGL Driver
glMapBuffer or glMapBufferRange / glFlushMappedBufferRange or glUnmapBuffer
FileRead(...) or direct modify
Vertex Data
memory accessible directly by GPU
DMA transfer
GPU memory

- Pointer returned by `glMapBuffer` is valid until `glUnmapBuffer` is called.

Image from http://developer.nvidia.com/object/using_VBOs.html

Buffer Mapping



Function	Usage hint	Destination memory	Transfer rate (GB/s)
glBufferData / glBufferSubData	GL_STATIC_DRAW	device	3.79
glMapBuffer / glUnmapBuffer	GL_STREAM_DRAW	pinned	n/a (pinned in CPU memory)
glMapBuffer / glUnmapBuffer	GL_STATIC_DRAW	device	5.73

Image from http://developer.nvidia.com/object/using_VBOs.html

Buffer Mapping

- Use `glMapBufferRange` to map a subset of a buffer. Why?

Buffer Mapping

- Use `glMapBufferRange` to map a subset of a buffer. Why?
  - Only upload the portion of a buffer that changed
  - Manual double buffering – use one half for updating and the other for rendering

Implicit Synchronization

- Command queue
- Rendering may occur a frame or two later
- Helps hide latency
- However implicit synchronization can occur:

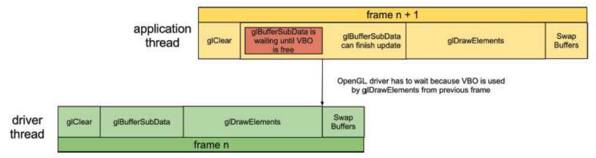

frame n + 1
application thread
glClear
glBufferSubData is waiting until VBO is free
glBufferSubData can finish update
glDrawElements
Swap Buffers
OpenGL driver has to wait because VBO is used by glDrawElements from previous frame
driver thread
glClear
glBufferSubData
glDrawElements
Swap Buffers
frame n

Image from http://developer.nvidia.com/object/using_VBOs.html

Implicit Synchronization

- Avoiding implicit synchronization
  - Round-robin
  - Orphan
  - Manual synchronization

Implicit Synchronization

- Round-robin

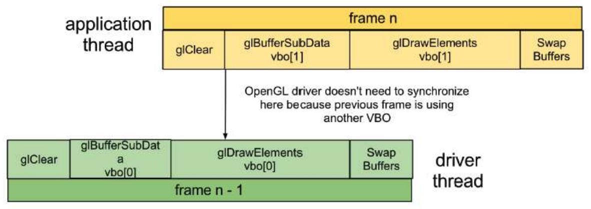


Image from http://developer.nvidia.com/object/using_VBOs.html

Implicit Synchronization

- Orphan – round robin inside the driver?

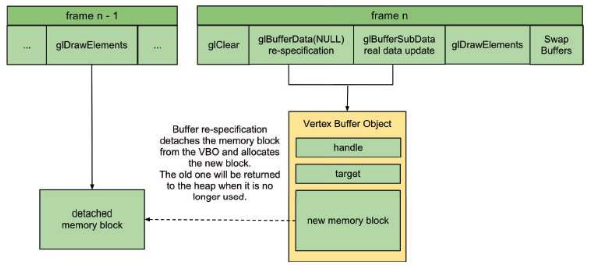

frame n - 1
...
glDrawElements
...
frame n
glClear
glBufferData(NULL) re-specification
glBufferSubData real data update
glDrawElements
Swap Buffers
Buffer re-specification detaches the memory block from the VBO and allocates the new block. The old one will be returned to the heap when it is no longer used.
Vertex Buffer Object
handle
target
new memory block
detached memory block

Image from http://developer.nvidia.com/object/using_VBOs.html

Implicit Synchronization

- Use `glMapBufferRange` with `GL_MAP_UNSYNCHRONIZED_BIT`
  - Manually sync with `glClientWaitSync`

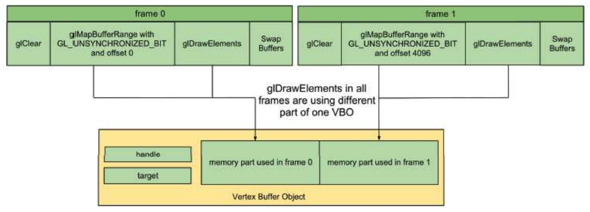
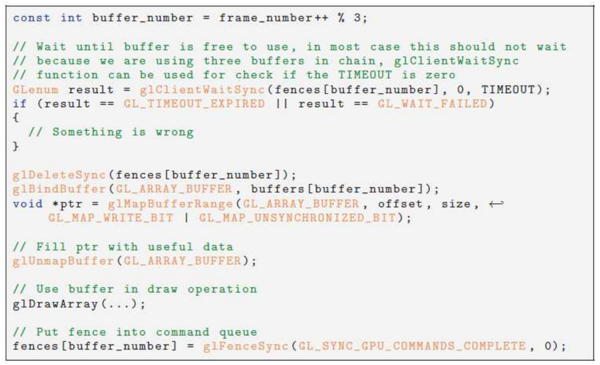

frame 0
glClear
glMapBufferRange with GL_UNSYNCHRONIZED_BIT and offset 0
glDrawElements
Swap Buffers
frame 1
glClear
glMapBufferRange with GL_UNSYNCHRONIZED_BIT and offset 4096
glDrawElements
Swap Buffers
glDrawElements in all frames are using different part of one VBO
handle
target
memory part used in frame 0
memory part used in frame 1
Vertex Buffer Object

Image from http://developer.nvidia.com/object/using_VBOs.html

Implicit Synchronization



```
const int buffer_number = frame_number++ % 3;

// Wait until buffer is free to use, in most case this should not wait
// because we are using three buffers in chain, glClientWaitSync
// function can be used for check if the TIMEOUT is zero
GLenum result = glClientWaitSync(fences[buffer_number], 0, TIMEOUT);
if (result == GL_TIMEOUT_EXPIRED || result == GL_WAIT_FAILED)
{
  // Something is wrong
}

glDeleteSync(fences[buffer_number]);
glBindBuffer(GL_ARRAY_BUFFER, buffers[buffer_number]);
void *ptr = glMapBufferRange(GL_ARRAY_BUFFER, offset, size, ↩
     GL_MAP_WRITE_BIT | GL_MAP_UNSYNCHRONIZED_BIT);

// Fill ptr with useful data
glUnmapBuffer(GL_ARRAY_BUFFER);

// Use buffer in draw operation
glDrawArray(...);

// Put fence into command queue
fences[buffer_number] = glFenceSync(GL_SYNC_GPU_COMMANDS_COMPLETE, 0);
```

Image from http://developer.nvidia.com/object/using_VBOs.html

Other Buffer Objects

- Pixel Buffers
- Texture Buffers
- Uniform Buffers

- These are not in OpenGL ES 2.