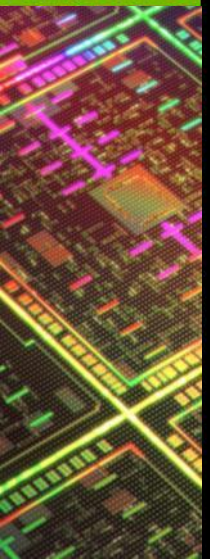
Optimizing Texture Transfers

Shalini Venkataraman
Senior Applied Engineer, NVIDIA
shaliniv@nvidia.com

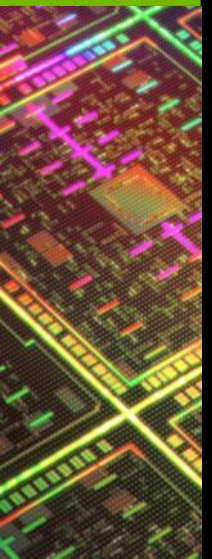


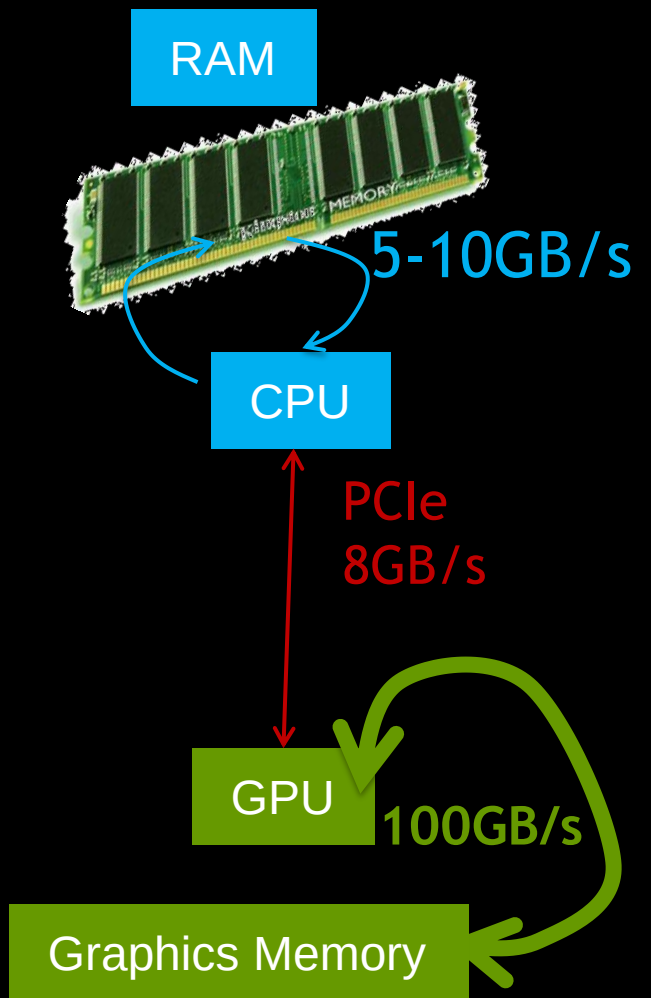
Outline

- Definitions
  - Upload : Host (CPU) -> Device (GPU)
  - Readback: Device (GPU) -> Host (CPU)
- Focus on OpenGL graphics
  - Implementing various transfer methods
  - Multi-threading and Synchronization
  - Debugging transfers
  - Best Practices & Results

Applications

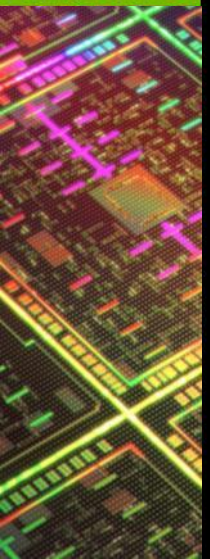
- Streaming videos/time varying geometry or volumes
  - Broadcast, real-time fluid simulations etc
- Level of detailing
  - Out of core image viewers, terrain engines
  - Bricks paged in as needed
- Parallel rendering
  - Fast communication between multiple GPUs for scaling data/render
- Remoting Graphics
  - Readback GPU results fast and stream over network





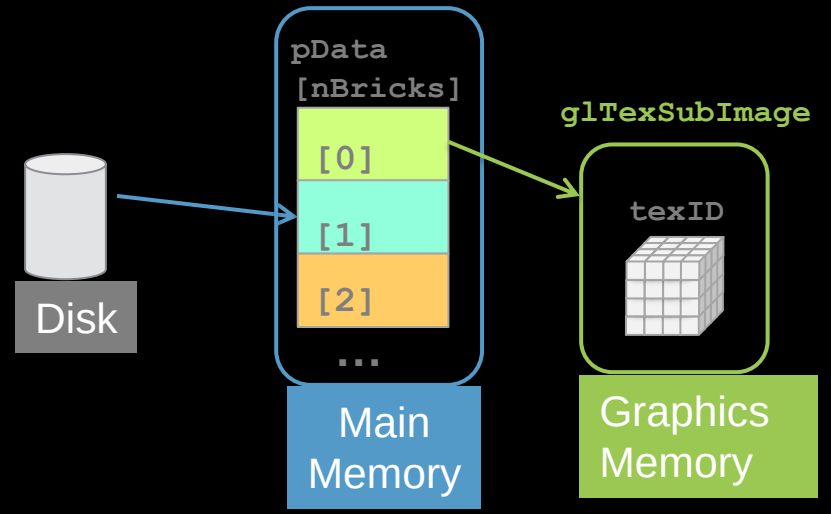
OpenGL Graphics - Streaming Data

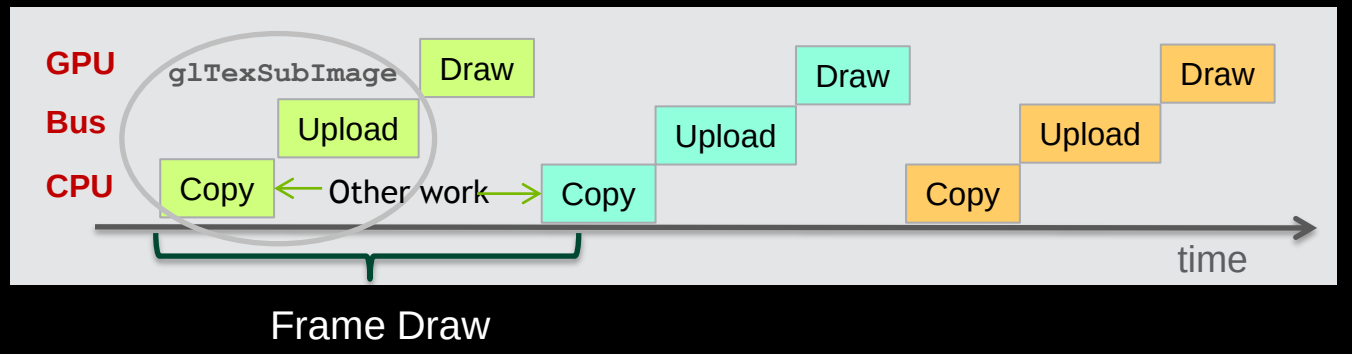
- Previous approaches
  - Synchronous - CPU and GPU idle during transfer
  - CPU Asynchronous
- GPU and CPU Asynchronous with Copy Engines
  - Application layout
  - Use cases
  - Results



Synchronous Transfers

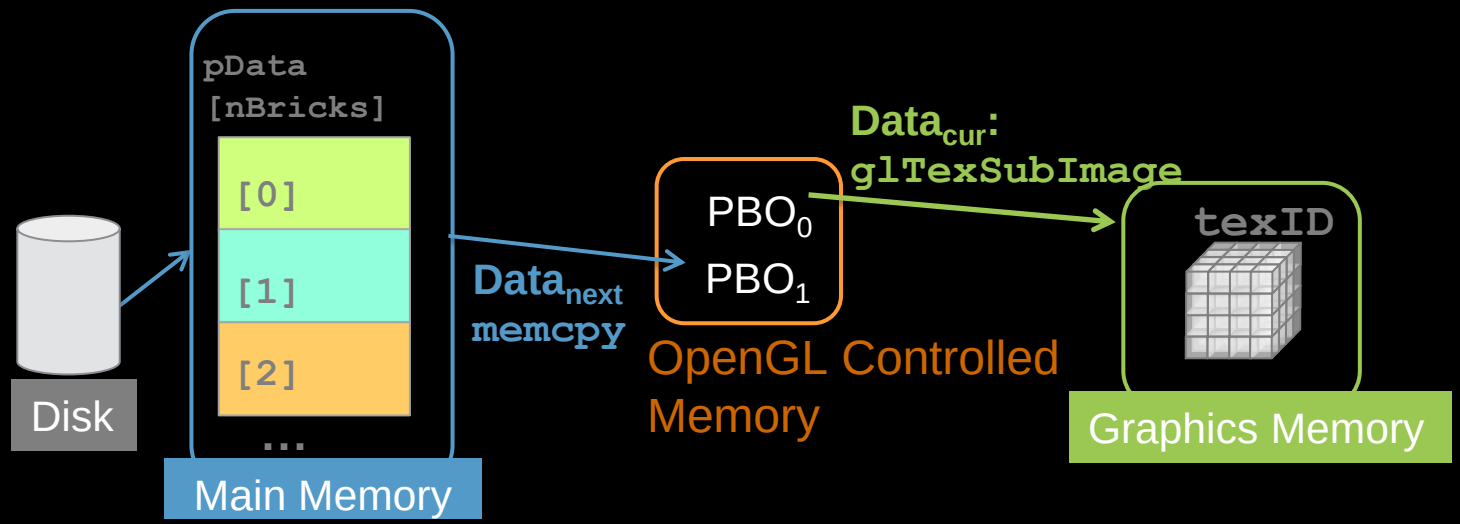
- Straightforward
  - Upload texture every frame
  - Driver does all copy
- Copy, download and draw are sequential





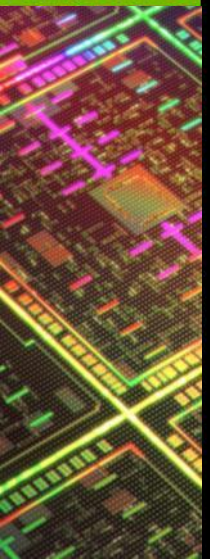
CPU Asynchronous Transfers

- Non CPU-blocking transfer using Pixel Buffer Objects (PBO)
  - Ping-pong PBO's for optimal throughput
  - Data must be in GPU native format

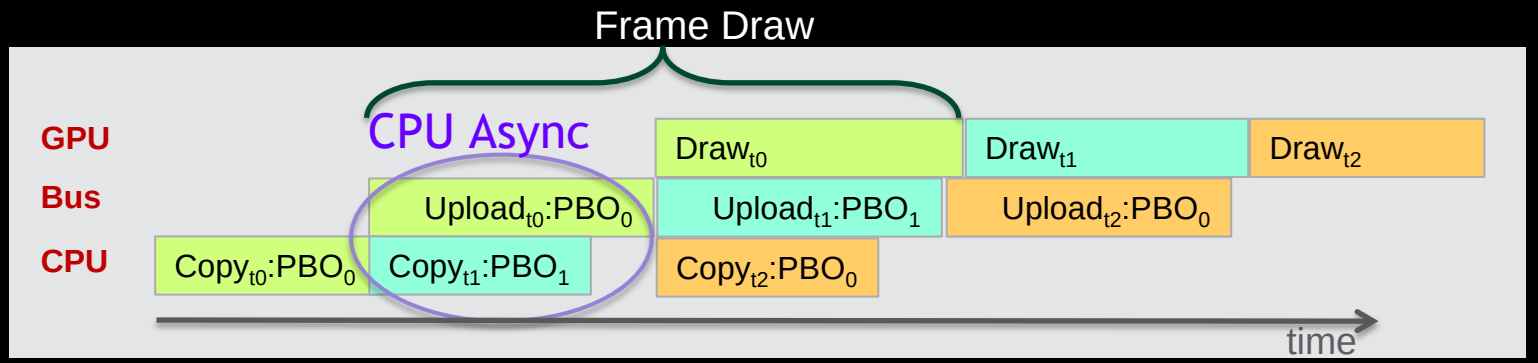


Example - 3D texture +Ping-Pong PBOs

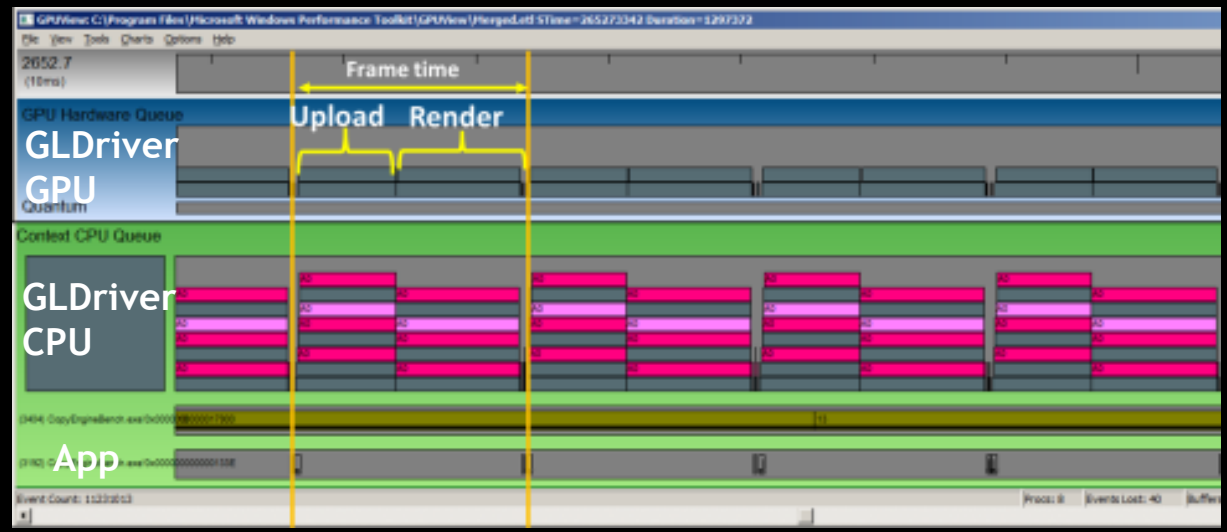
```
Gluint pbo[2] ; //ping-pong pbo generate and initialize them ahead
unsigned int curPBO = 0;

//bind current pbo for app->pbo transfer
glBindBuffer(GL_PIXEL_UNPACK_BUFFER_ARB, pbo[curPBO]); //bind pbo
GLubyte* ptr = (GLubyte*)glMapBufferRange(GL_PIXEL_UNPACK_BUFFER_ARB, 0, size,
          GL_MAP_WRITE_BIT|GL_MAP_INVALIDATE_BUFFER_BIT);
memcpy(ptr,pData[curBrick],xdim*ydim*zdim);
glUnmapBuffer(GL_PIXEL_UNPACK_BUFFER_ARB);
//Copy pixels from pbo to texture object
glBindTexture(GL_TEXTURE_3D,texId);
glBindBuffer(GL_PIXEL_UNPACK_BUFFER_ARB, pbo[1-curPBO]); //bind pbo
glTexSubImage3D(GL_TEXTURE_3D,0,0,0,0,xdim,ydim,zdim,GL_LUMINANCE,GL_UNSIGNED_BYTE,0);
glBindBuffer(GL_PIXEL_UNPACK_BUFFER_ARB,0);
glBindTexture(GL_TEXTURE_3D,0);

curPBO = 1-curPBO;
//Call drawing code here
```



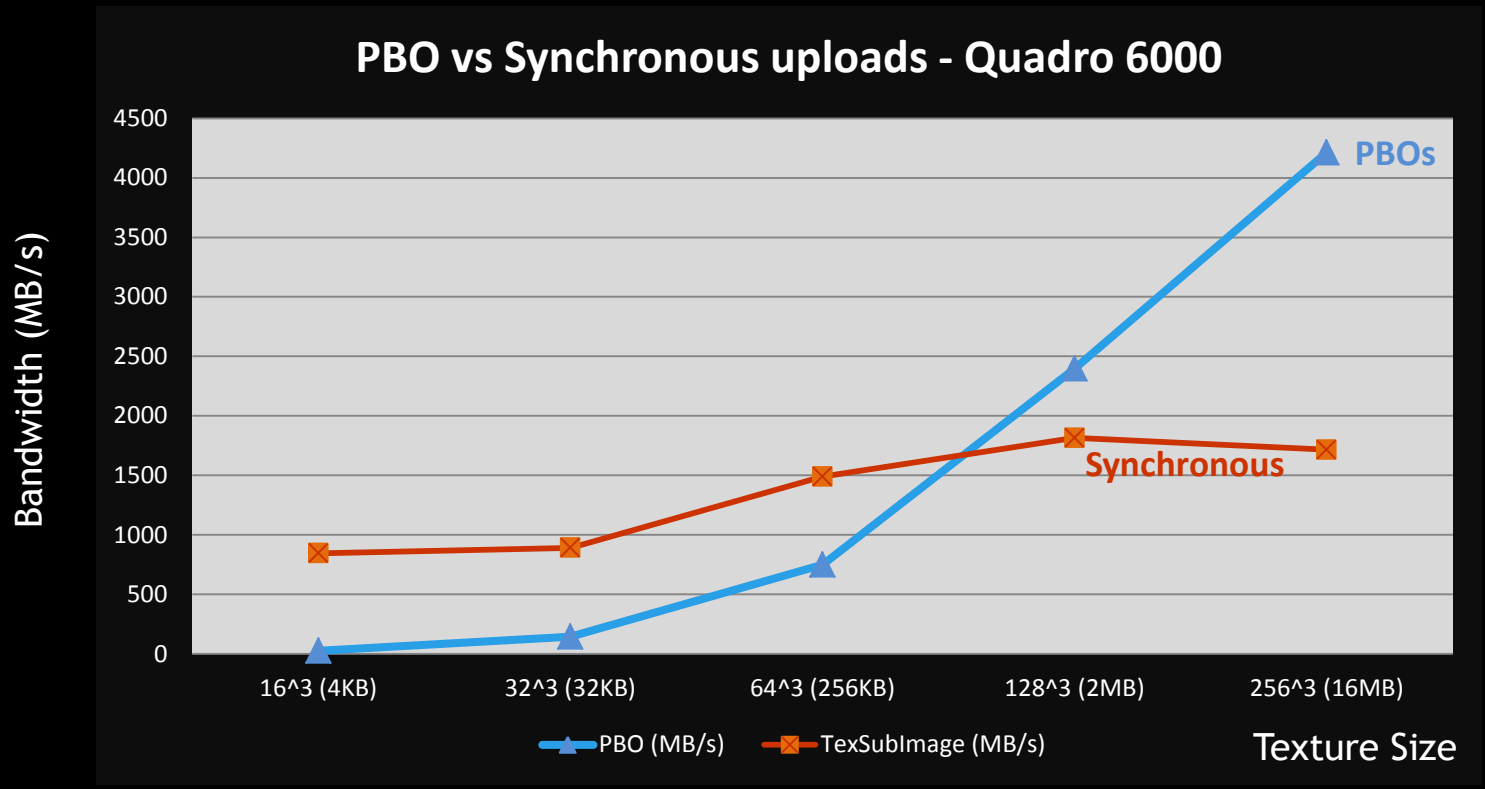
CPU Async - Execution Timeline



Analysis with GPUView (http://graphics.stanford.edu/~mdfisher/GPUView.html)



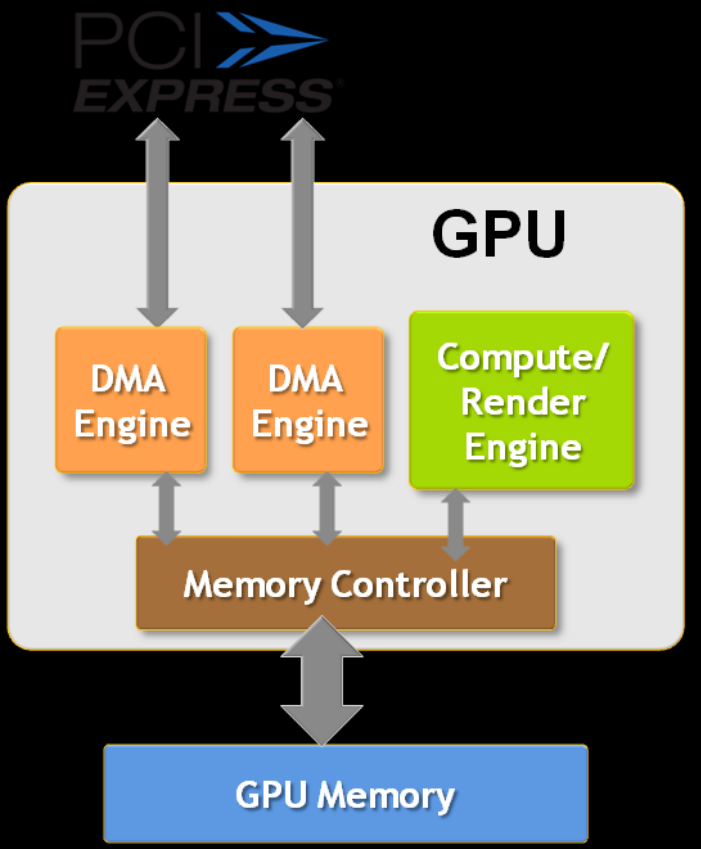
Results - Synchronous vs CPU Async



- Transfers only
- Adding rendering will reduce bandwidth, GPU can't do both
- Ideally – want to sustain bandwidth with render, need GPU overlap

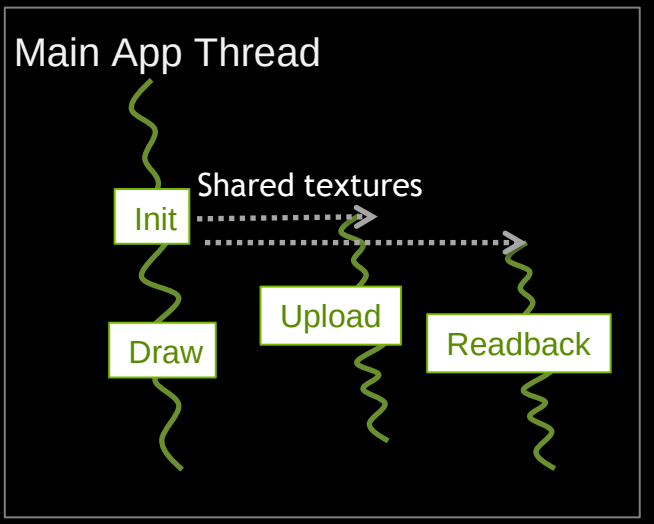
Achieving Overlap - Copy Engines

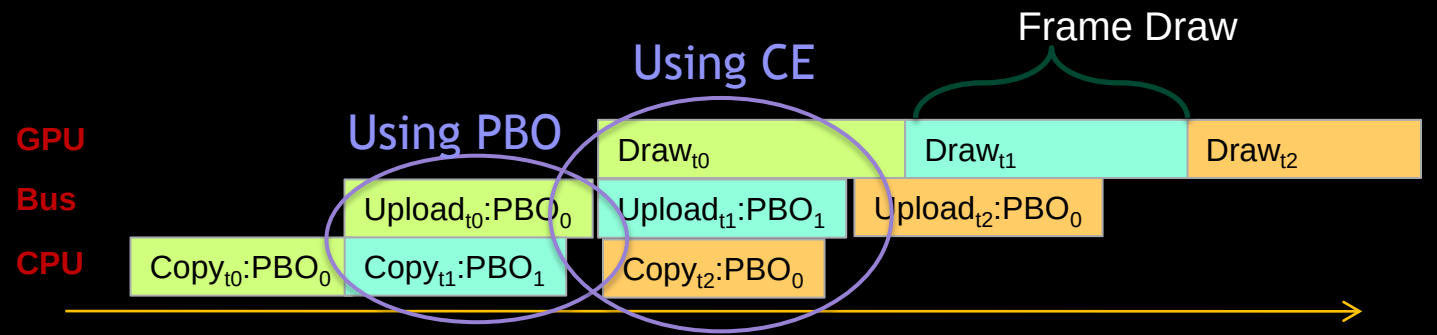
- Fermi+ have copy engines
  - GeForce, low-end Quadro- 1 CE
  - Quadro 4000+ - 2 CEs
- Allows copy-to-host + compute + copy-to-device to overlap simultaneously
- Graphics/OpenGL
  - Using PBO's in multiple threads
  - Handle synchronization



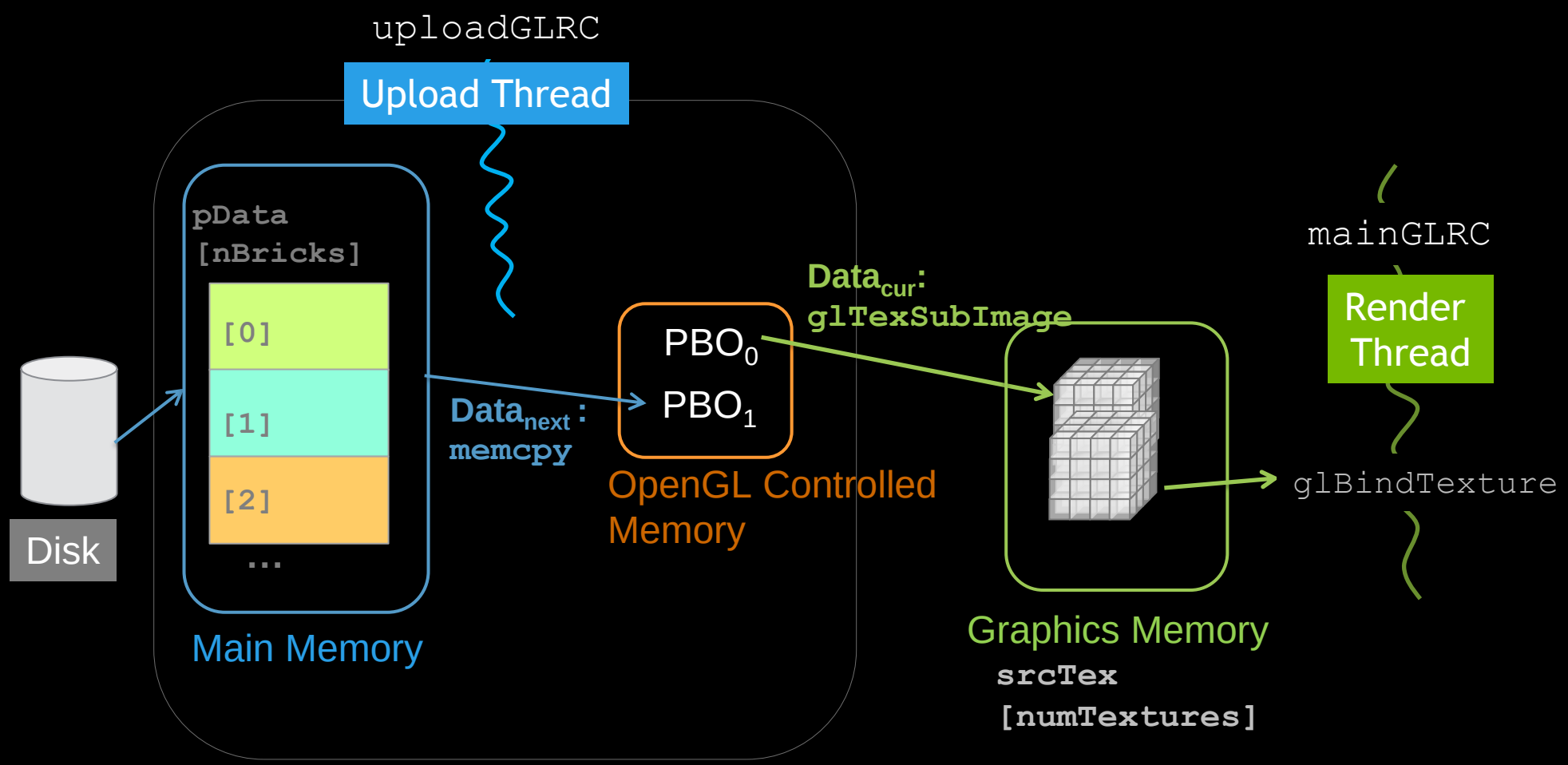
GPU Asynchronous Transfers

- Downloads/uploads in separate thread
  - Using OpenGL PBOs
- ARB_SYNC used for context synchronization





Upload-Render : Application Layout



Multi-threaded Context Creation

- Sharing textures between multiple contexts
  - Don’t use wglShareLists
  - Use WGL/GLX_ARB_CREATE_CONTEXT instead
  - Set OpenGL debug on

```
static const int contextAttribs[] =
{
    WGL_CONTEXT_FLAGS_ARB, WGL_CONTEXT_DEBUG_BIT_ARB,
    0
};
mainGLRC = wglCreateContextAttribsARB(winDC, 0, contextAttribs);
wglMakeCurrent(winDC, mainGLRC);
glGenTextures(numTextures, srcTex);
//uploadGLRC now shares all its textures with mainGLRC
uploadGLRC = wglCreateContextAttribsARB(winDC, mainGLRC, contextAttribs);
//Create Upload thread
//Do above for readback if using
```

Synchronization using ARB_SYNC

- OpenGL commands are asynchronous
  - When glDrawXXX returns, does not mean command is completed
- Sync object glSync (ARB_SYNC) is used for multi-threaded apps that need sync
  - Eg rendering a texture waits for upload completion
- Fence is inserted in a unsignaled state but when completed changed to signaled.

```
//Upload
glTexSubImage(texID,..)
GLSync fence = glFenceSync(..)
```

unsignaled

signaled

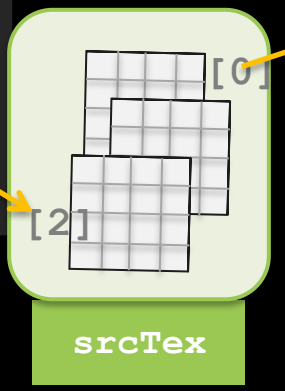
```
//Render
glWaitSync(fence);
glBindTexture(.., texID);
```

Upload-Render Sychronizaton

```
GLsync startUpload[MAX_BUFFERS], endUpload[MAX_BUFFERS]; //GPU fence sync objects
HANDLE startUploadValid, endUploadValid; //cpu event to coordinate wait for GPU sync
```

Upload

```
WaitForSingleObject(startUploadValid)
glWaitSync(startUpload[2])
glBindTexture(srcTex[2])
glTexSubImage(..)
endUpload[2] = glFenceSync(…)
SetEvent(endUploadValid)
```



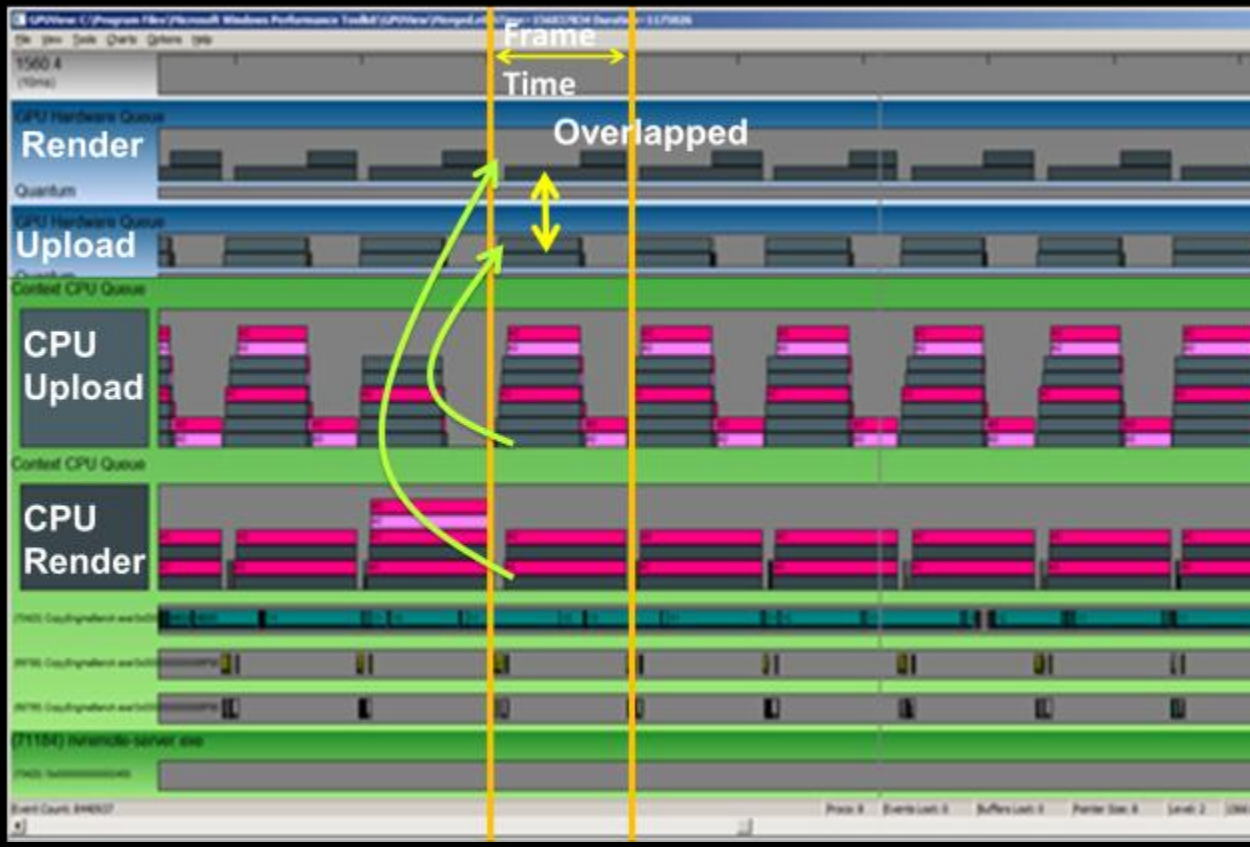
Render

```
WaitForSingleObject(endUploadValid)
glWaitSync(endUpload[0])
glBindTexture(srcTex[0])
//Draw
startUpload[0] = glFenceSync(…)
SetEvent(startUploadValid);
```

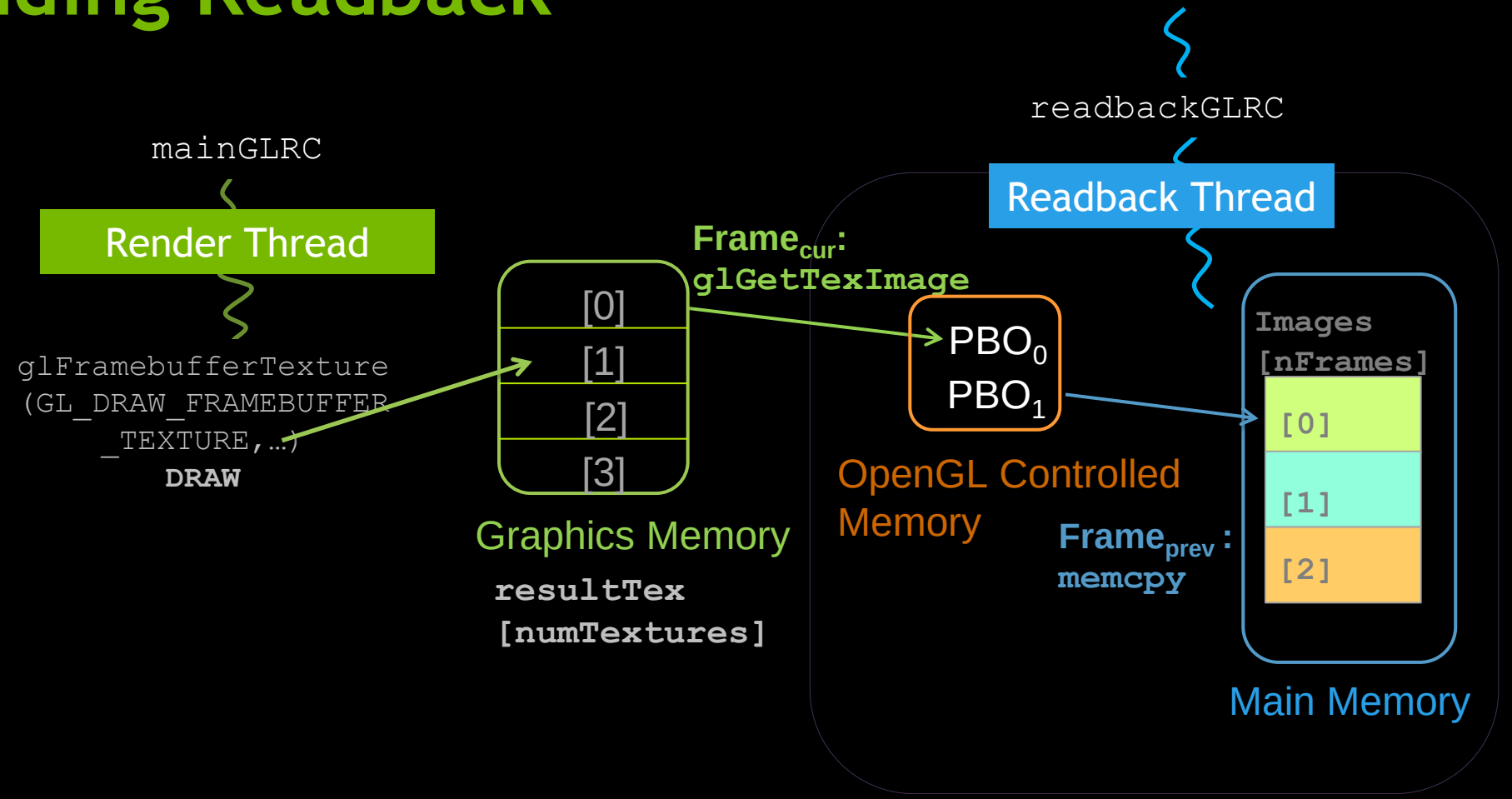
- Need additional CPU event to coordinate waiting for GPU sync!

Analysis with GPUView

- Upload and Render in separate threads
  - Map to distinct hardware queues on GPU
  - Executed concurrently
  - Will serialize on pre-Fermi hardware



Adding Readback



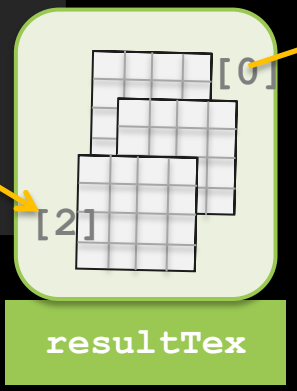
Use glGetTexImage, not glReadPixels between threads

Render-Readback Synchronizaton

```
GLsync startReadback[MAX_BUFFERS],endReadback[MAX_BUFFERS]; //GPU fence sync objects
HANDLE startReadbackValid, endReadbackValid; //cpu event to coordinate wait for GPU sync
```

Render

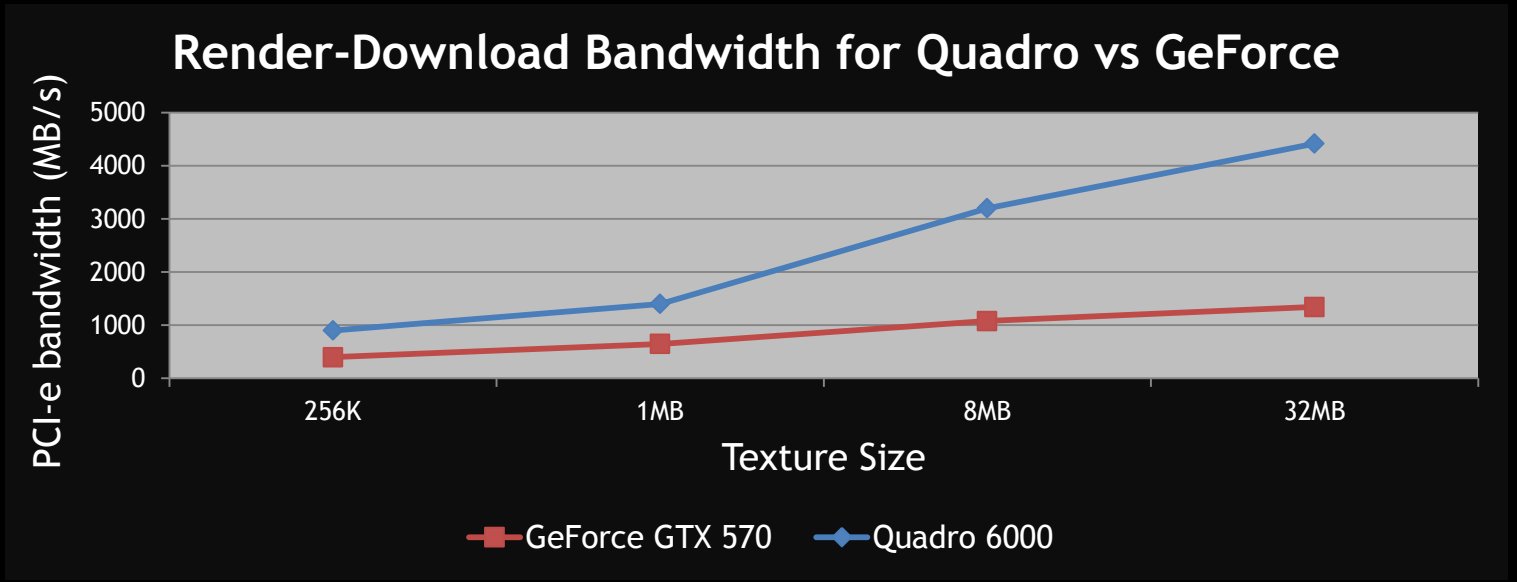
```
WaitForSingleObject(endReadbackValid)
glWaitSync(endReadback[2])
glFramebufferTexture(resultTex[2])
//Draw
startReadback[3] = glFenceSync(…)
SetEvent(startReadbackValid)
```



Readback

```
WaitForSingleObject(startReadbackValid)
glWaitSync(startReadback[0])
glGetTexImage(resultTex[0])
//Read pixels to png-pong pbo
endReadback[0] = glFenceSync(…)
SetEvent(endReadbackValid);
```

GeForce vs Quadro Readbacks



Readbacks on GeForce are 3x slower than Quadro

Upload-Render-Readback pipeline

Capture Thread

```
// Wait for signal to start upload
CPUWait(startUploadValid);
glWaitSync(startUpload[2]);

// Bind texture object
BindTexture(capTex[2]);

// Upload
glTexSubImage(texID…);

// Signal upload complete
GLSync endUpload[2]= glFenceSync(…);
CPUSignal(endUploadValid);
```

[0]
[1]
[2]
[3]

Render Thread

```
// Wait for download to complete
CPUWait(endDownloadValid);
glWaitSync(endDownload[3]);

// Wait for upload to complete
CPUWait(endUploadValid);
glWaitSync(endUpload)[0]);

// Bind render target
glFramebufferTexture(playTex[3]);

// Bind video capture  source texture
BindTexture(capTex[0]);

// Draw

// Signal next upload
startUpload[0] = glFenceSync(…);
CPUSignal(startUploadValid);
// Signal next download
startDownload[3] = glFenceSync(…);
CPUSignal(startDownloadValid);
```

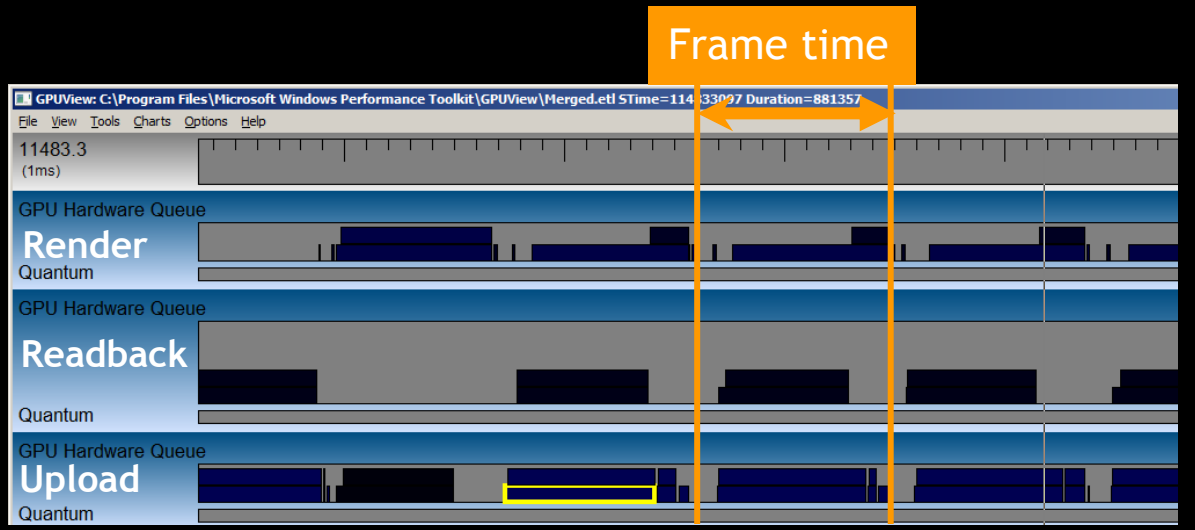
[0]
[1]
[2]
[3]

Playout Thread

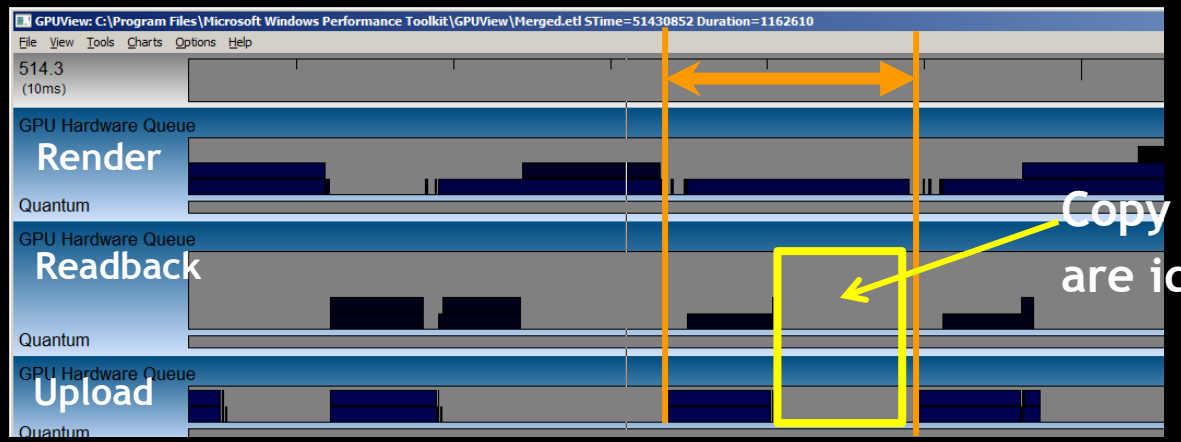
```
// Playout thread
CPUWait(startDownloadValid);
glWaitSync(startDownload[2]);

// Readback
glGetTexImage(playTex[2]);

// Read pixels to PBO

// Signal download complete
endDownload[2] = glFenceSync(…);
CPUSignal(endDownloadValid);
```

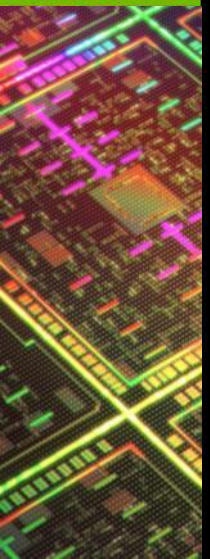
GPUView trace showing 3-way overlap

Balanced render, upload and readback times



Render time larger than upload and readback



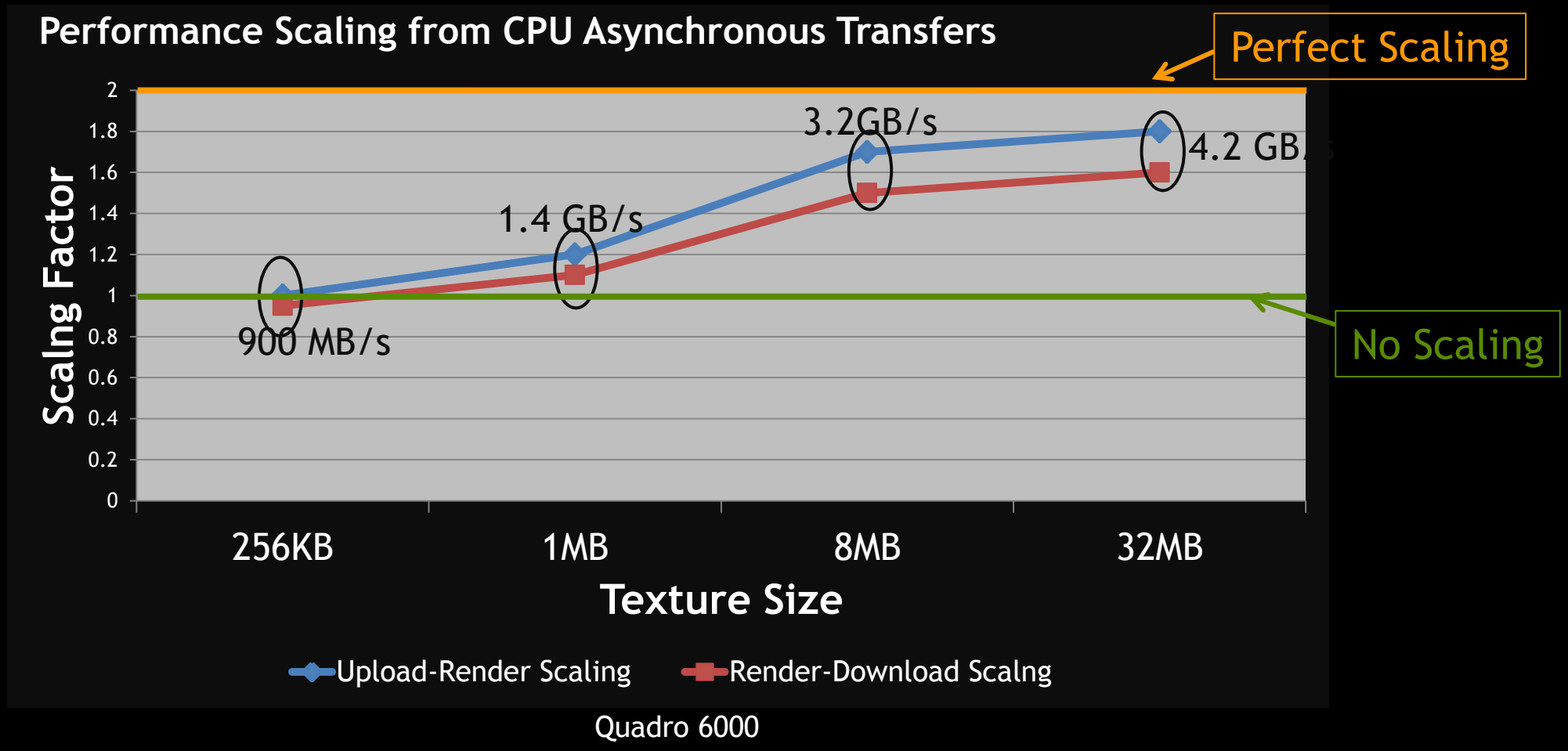


Debugging Transfers

- Some OGL calls may not overlap between transfer/render thread
  - Eg non-transfer related OGL calls in transfer thread
  - Driver generates debug message
    - “Pixel transfer is synchronized with 3D rendering”
  - Application uses ARB_DEBUG_OUTPUT to check the OGL debug log
  - OpenGL 4.0 and above

GL_ARB_debug_output - http://www.opengl.org/registry/specs/ARB/debug_output.txt

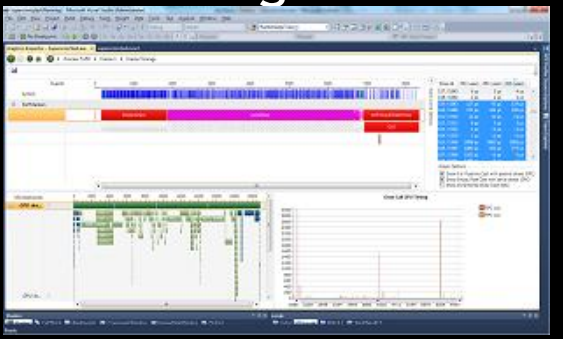
Copy Engine Results - Best Case



Quadro 6000

Conclusion

- Presented different transfer methods
- Keep the transfer method simple
  - Look at your application transfer needs and render times
  - Tradeoff in scaling vs application complexity
- Future
  - Debugging multi-threaded transfers made much easier with Nsight Visual studio http://developer.nvidia.com/nvidia-nsight-visual-studio-edition)



References

- Venkataraman, Fermi Asynchronous Texture Transfers, OpenGL Insights, 2012
  - Source code (around SIGGRAPH 2012) - https://github.com/organizations/OpenGLInsights
- Related GTC Talks
  - S0328, Thomas True, Best Practices in GPU-based video processng
  - S0049, Alina Alt &Tom True, Using the GPU Direct for Video API
  - S0353, S Venkataraman, Programming multi-gpus for scalable rendering