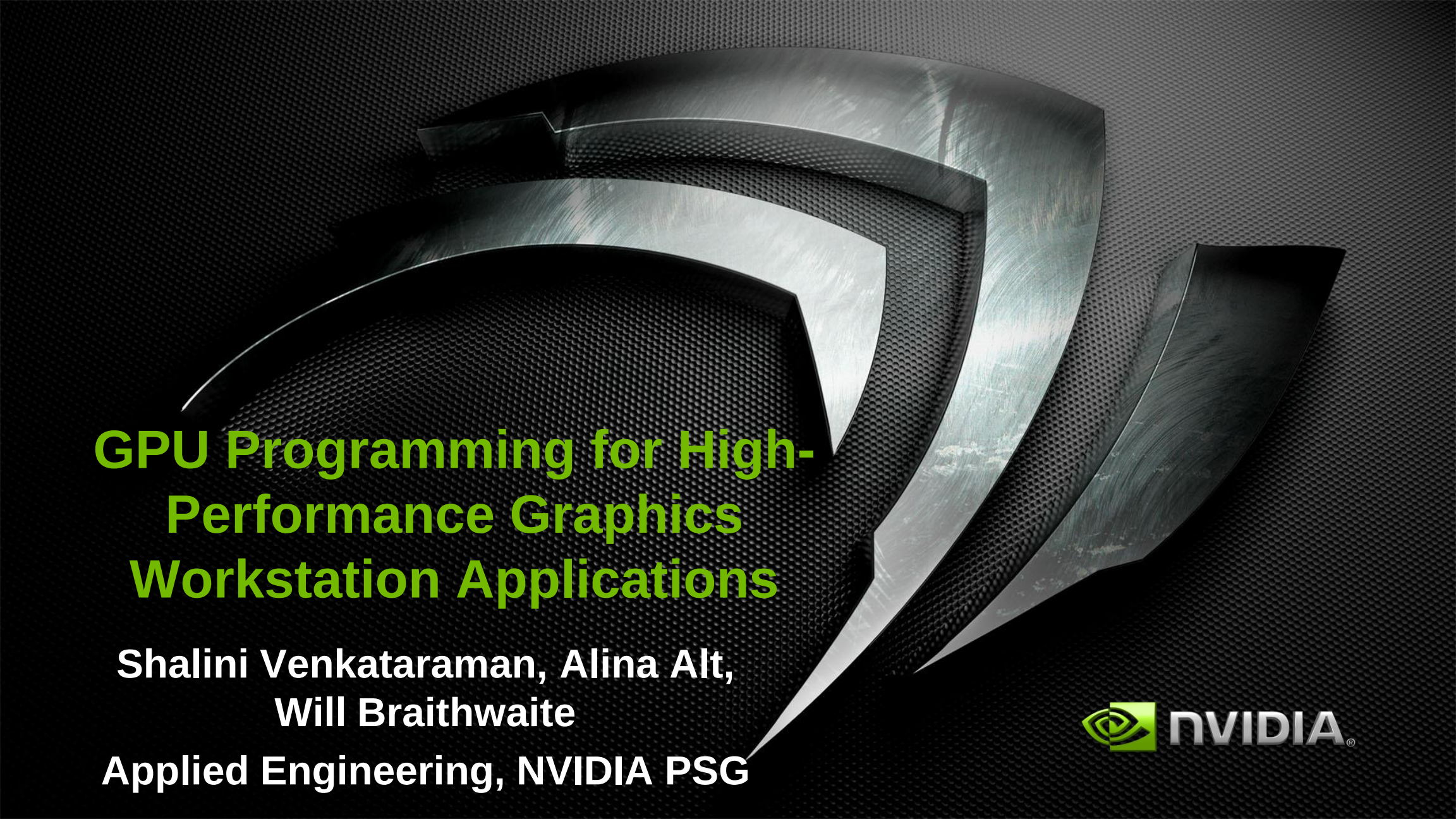


GPU Programming for High-Performance Graphics Workstation Applications

Shalini Venkataraman, Alina Alt,
Will Braithwaite

Applied Engineering, NVIDIA PSG

NVIDIA

Talk Outline

- Scaling transfers and rendering
  - Shalini Venkataraman
- Mixing graphics and compute
  - Alina Alt
- Developing an optimized Maya plugin using CUDA and OpenGL
  - Will Braithwaite
- Questions at the end

Scaling Transfers and Rendering

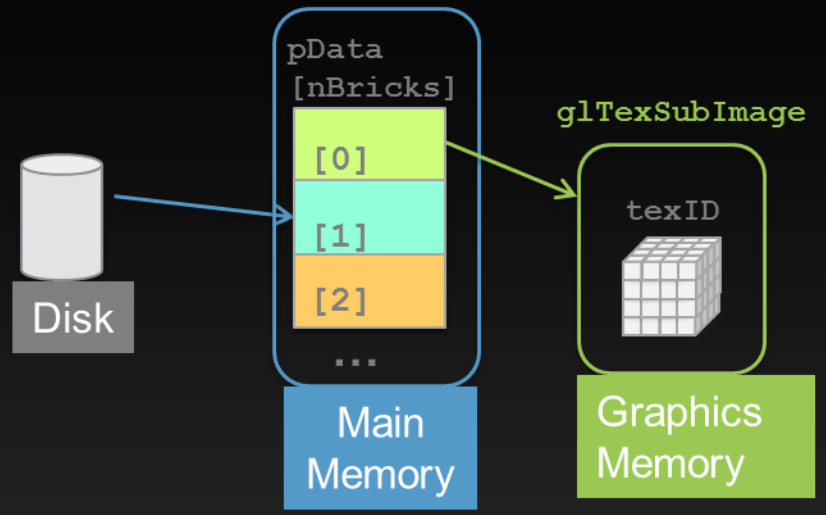
- Overlapping transfers & rendering
  - Implementing various transfer methods
  - Multi-threading and Synchronization
  - Debugging transfers
  - Best Practices & Results
- Scaling to Multi-GPU
  - Pinning OpenGL context to GPU
  - Application structure
  - Optimized inter-GPU transfers

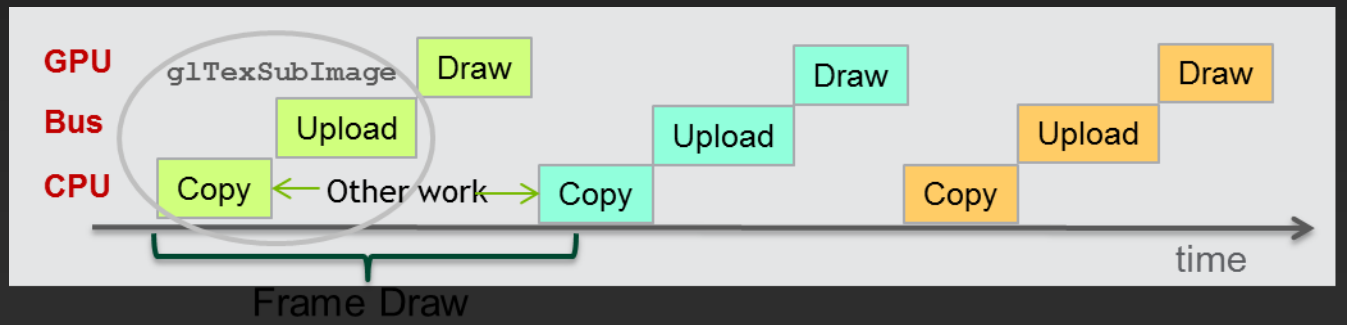
Applications

- 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

Previous Approach – Synchronous Transfers

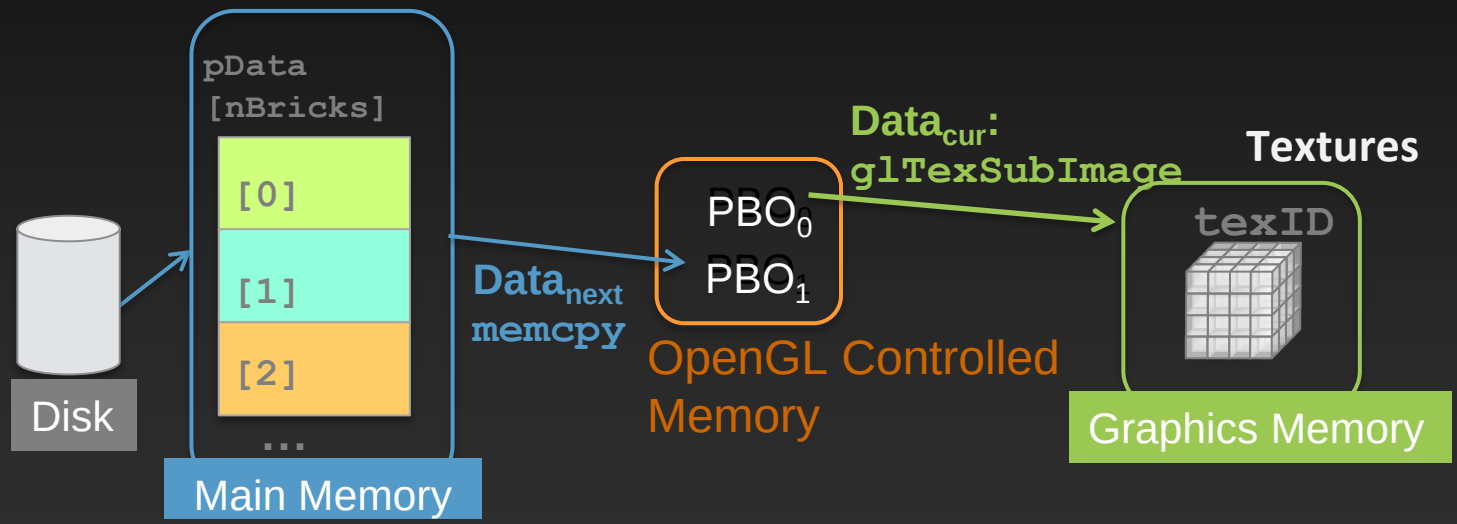
- Straightforward
  - Upload texture every frame
  - Driver does all copy
- Copy, download and draw are sequential



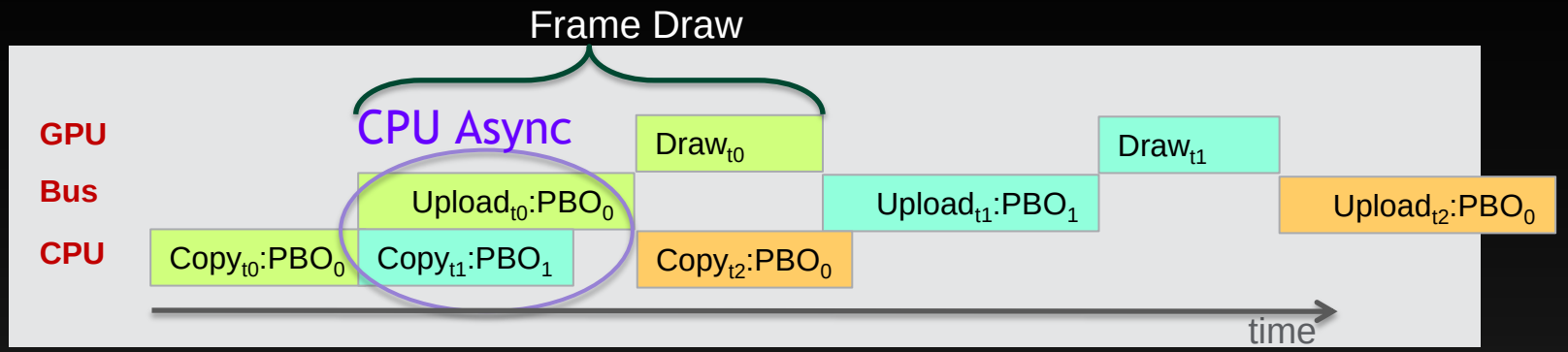


Previous Approach - CPU Asynchronous Transfers

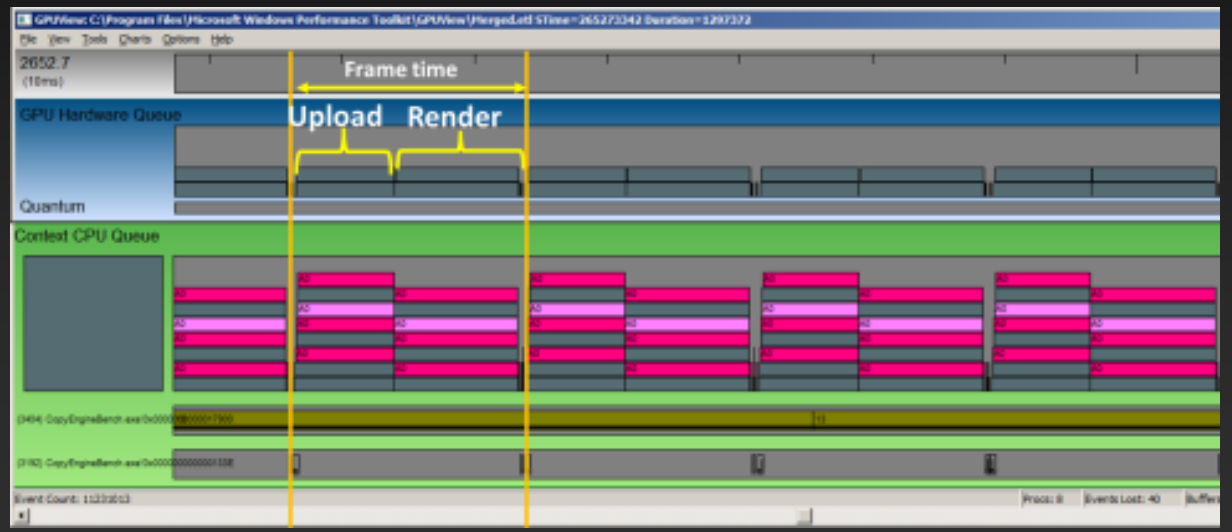
- Non CPU-blocking transfer using Pixel Buffer Objects (PBO)
  - Ping-pong PBOs for optimal throughput
  - Data must be in GPU native format



CPU Asynchronous - Timeline



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



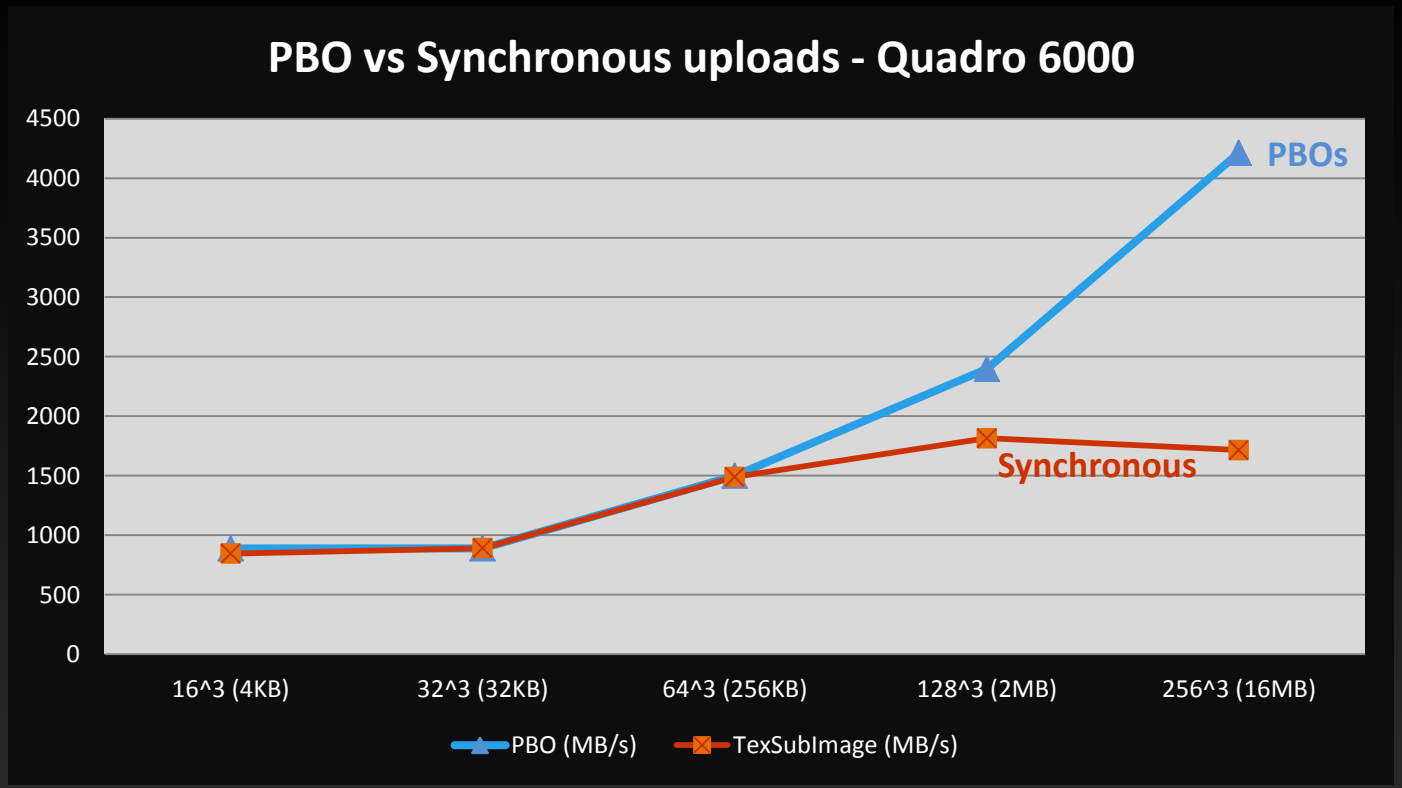
Example - 3D texture upload +Ping-Pong PBOs

```
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
```

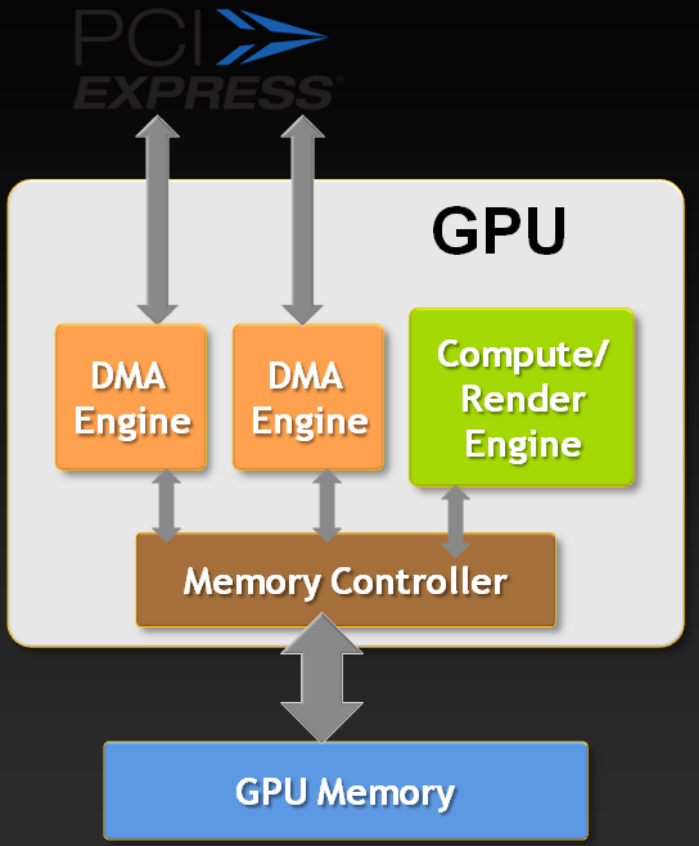
Results - Synchronous vs CPU Async



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

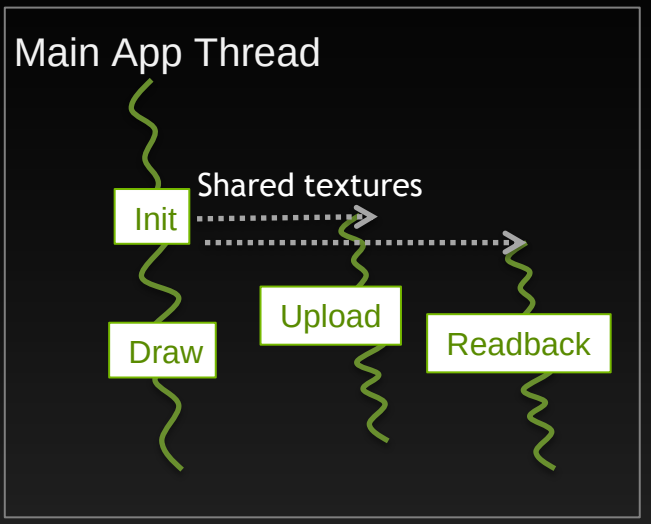
Achieving GPU Overlap - Copy Engines

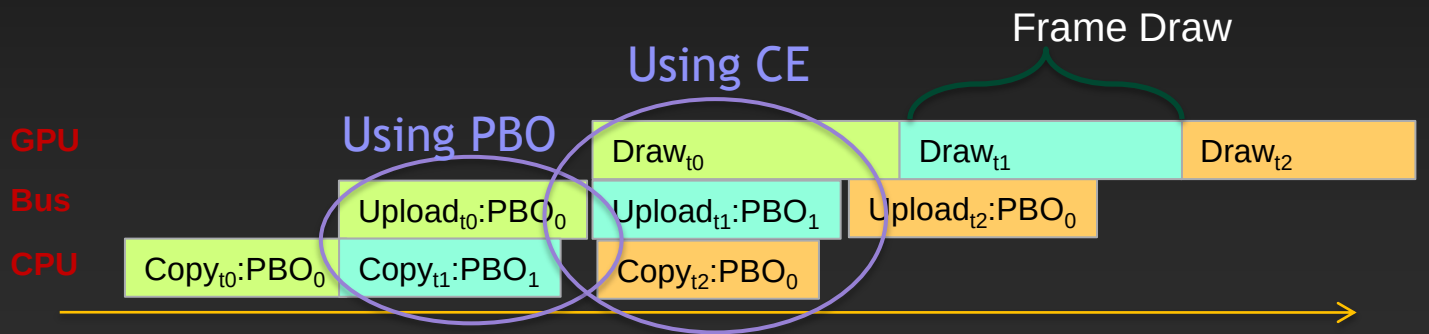
- Fermi+ have copy engines
  - GeForce, low-end Quadro- 1 CE
  - Quadro 4000+ - 2 CEs
- Allows copy-to-host + compute + copy-to-device to overlap simultaneously
- Graphics/OpenGL
  - Using PBO's in multiple threads
  - Handle synchronization



GPU Asynchronous Transfers

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



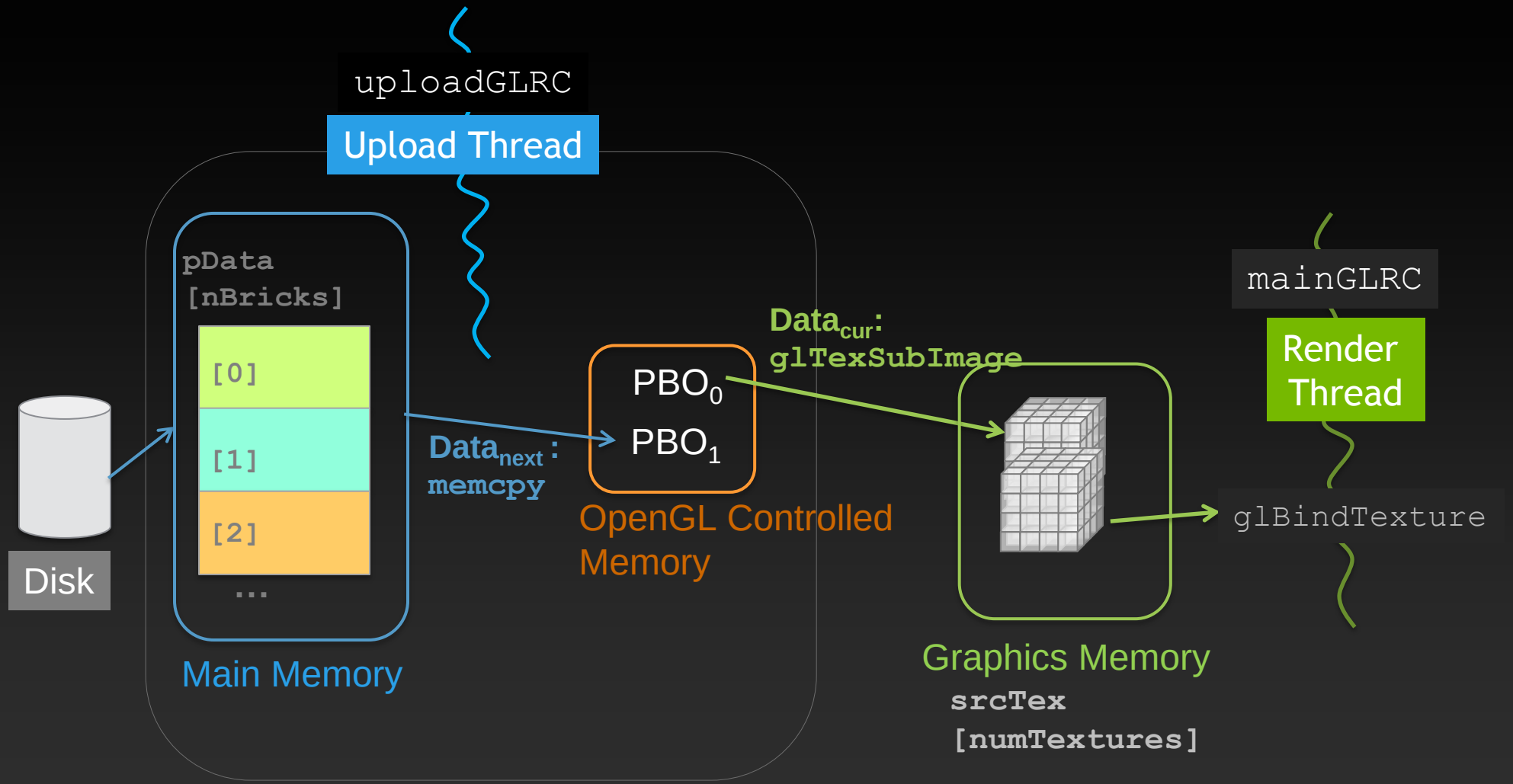


Multi-threaded Context Creation

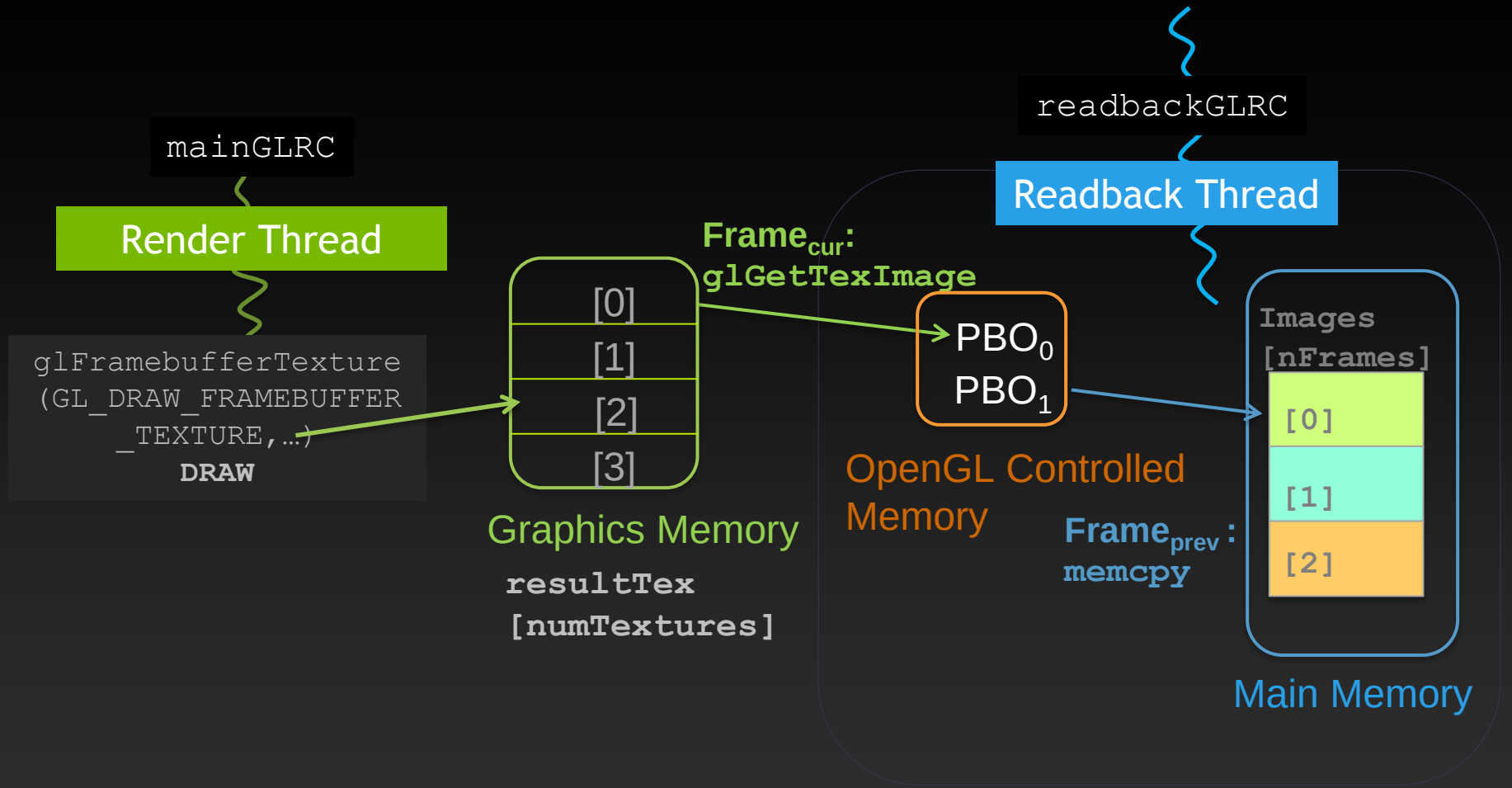
- Sharing textures between multiple contexts
  - Don’t use wglShareLists
  - Use WGL/GLX_ARB_CREATE_CONTEXT instead
  - Set OpenGL debug on

```
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
```

Upload-Render: Application Layout



Adding Render - Readback



Use glGetTexImage, not glReadPixels between contexts/threads

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-Readback Pipeline

Upload 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]

Readback Thread

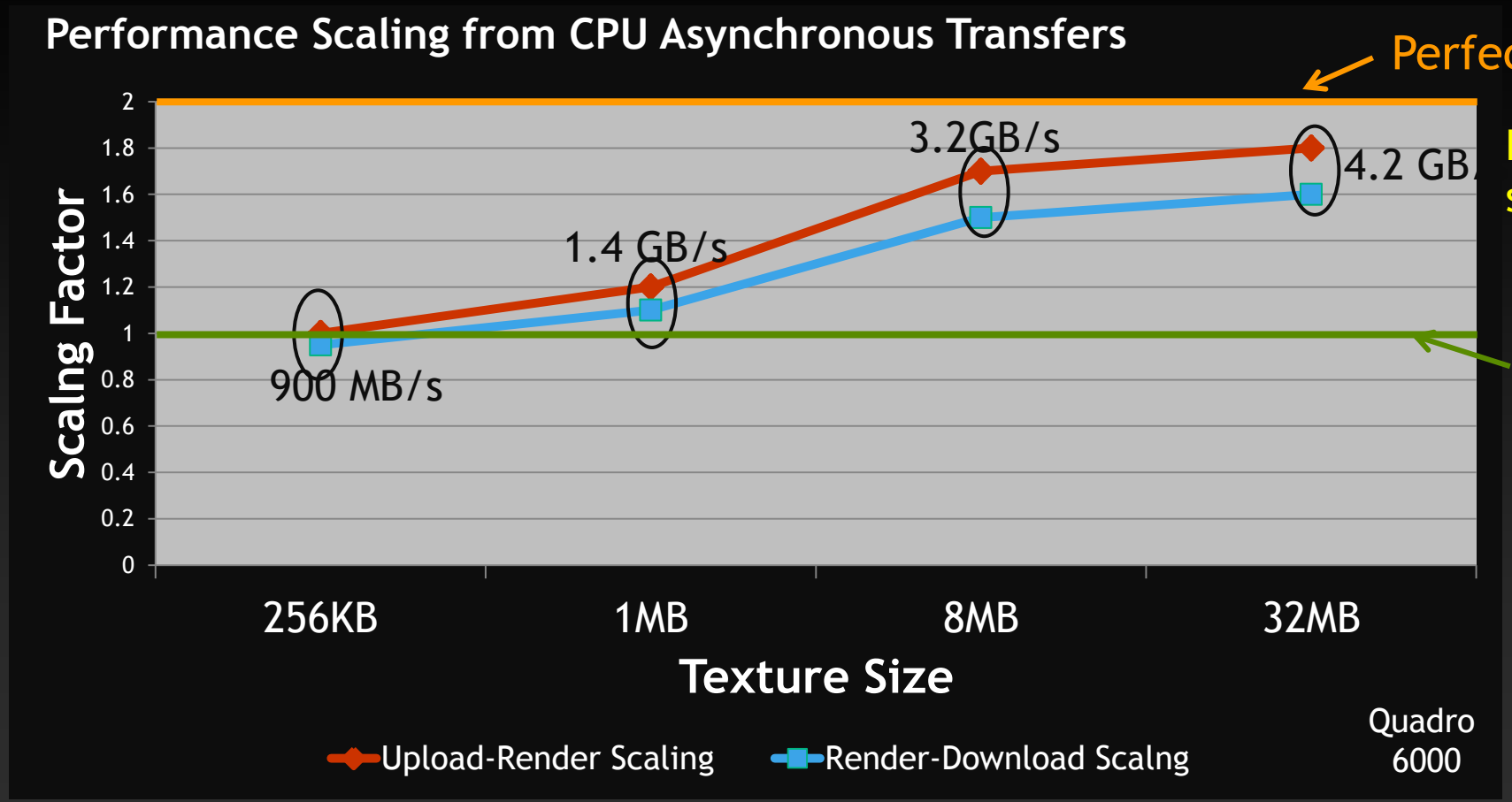
```
// Playout thread
CPUWait(startDownloadValid);
glWaitSync(startDownload[2]);

// Readback
glGetTexImage(playTex[2]);

// Read pixels to PBO

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

Results

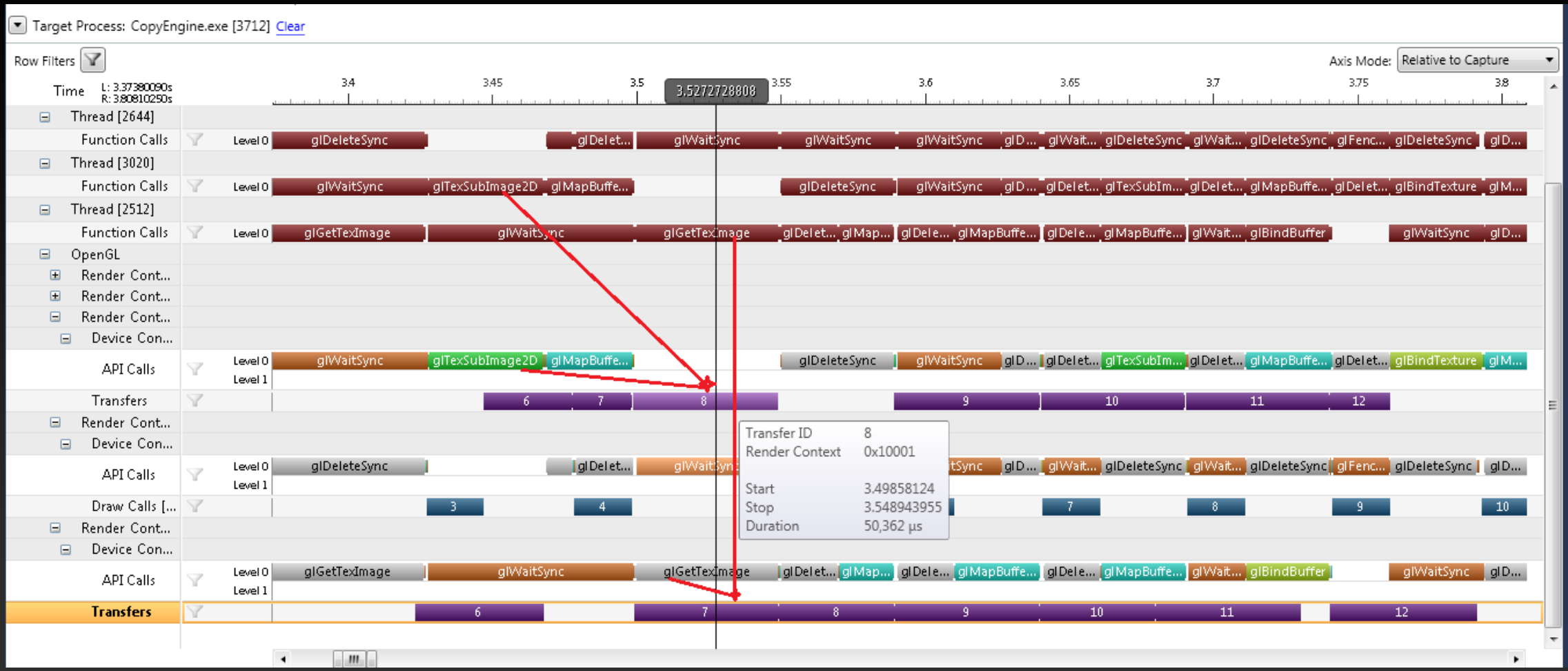

Performance Scaling from CPU Asynchronous Transfers
Perfect Scaling
Larger texture sizes scale better
No Scaling
Scalng Factor
2
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
900 MB/s
1.4 GB/s
3.2GB/s
4.2 GB/
256KB
1MB
8MB
32MB
Texture Size
Upload-Render Scaling
Render-Download Scalng
Quadro 6000

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
- Currently supported for PBOs, not VBOs
- Will serialize on Pre-Fermi hardware

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

Debugging with Nsight Visual Studio



Scaling Rendering on Multi-GPU

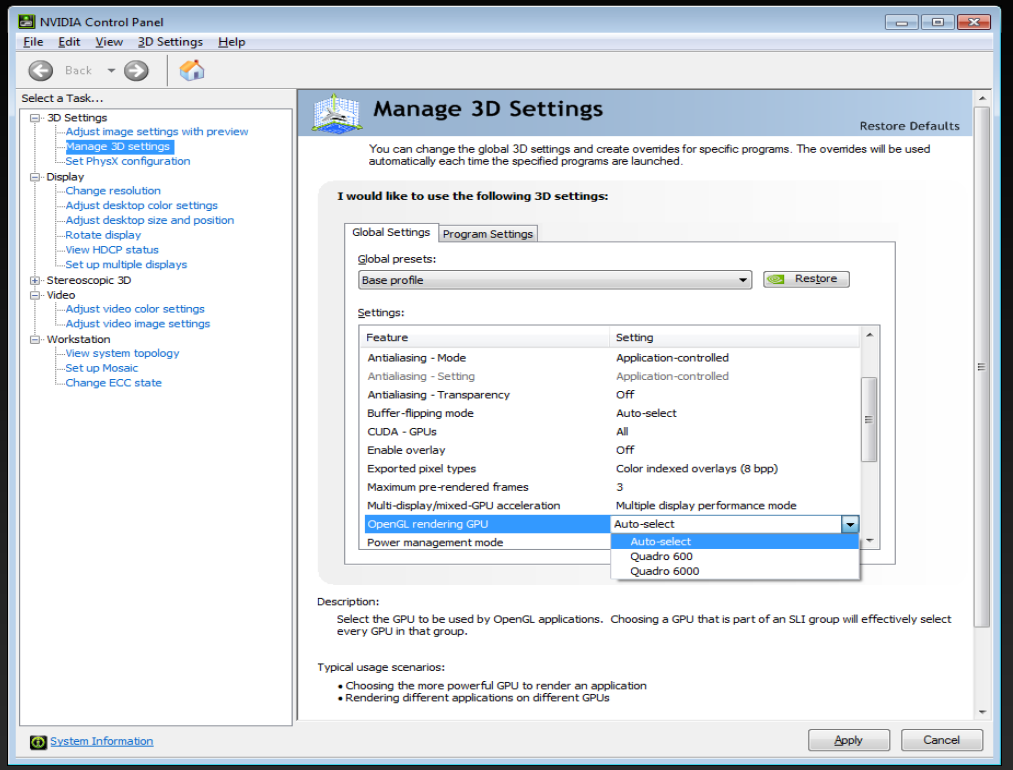
- Focus on OpenGL graphics
- Onscreen Rendering
  - X Display scaling for multi-projector, multi-tiled display environments
    http://developer.download.nvidia.com/GTC/PDF/GTC2012/PresentationPDF/S0353-GTC2012-Multi-GPU-Rendering.pdf
- Offscreen Parallel Rendering
  - Image Scaling - final image resolution
  - Data scaling - texture size, # triangles
  - Task/Process Scaling - eg render farm serving thin clients

Multi-GPU - Transparent Behavior

- Default Behavior of OGL command dispatch
  - Win XP : Sent to all GPUs, slowest GPU gates performance
  - Linux : Only to the GPU attached to screen
  - Win 7: Sent to most powerful GPU and blitted across
- SLI AFR
  - Single threaded application
  - Data and commands are replicated across all GPUs

Specifying OpenGL GPU on NVIDIA Quadro

- Directed GPU Rendering
  - Quadro-only
  - Heuristics for automatic GPU selection
  - Allow app to pick the GPU for rendering, fast blit path to other displays
  - Programmatically using NVAPI or using CPL



Programming for Multi-GPU

- Linux
  - Specify separate X screens using XOpenDisplay

```
Display* dpy = XOpenDisplay(":0."+gpu)
GLXContext = glxCreateContextAttribs(dpy,…);
```

  - Xinerama disabled
- Windows
  - Vendor specific extension
  - NVIDIA : NV_GPU_AFFINITY extension
  - AMD Cards : AMD_GPU_Association

GPU Affinity-
Enumerating and attaching to GPUs

- **Enumerate GPUs**

```
BOOL wglEnumGpusNV(UINT iGpuIndex, HGPUNV *phGPU)
```

- **Enumerate Displays per GPU**

```
BOOL wglEnumGpusDevicesNV(HGPUNV hGPU, UINT iDeviceIndex,
                          PGPU_DEVICE lpGpuDevice);
```

- **Pinning OpenGL context to a specific GPU**

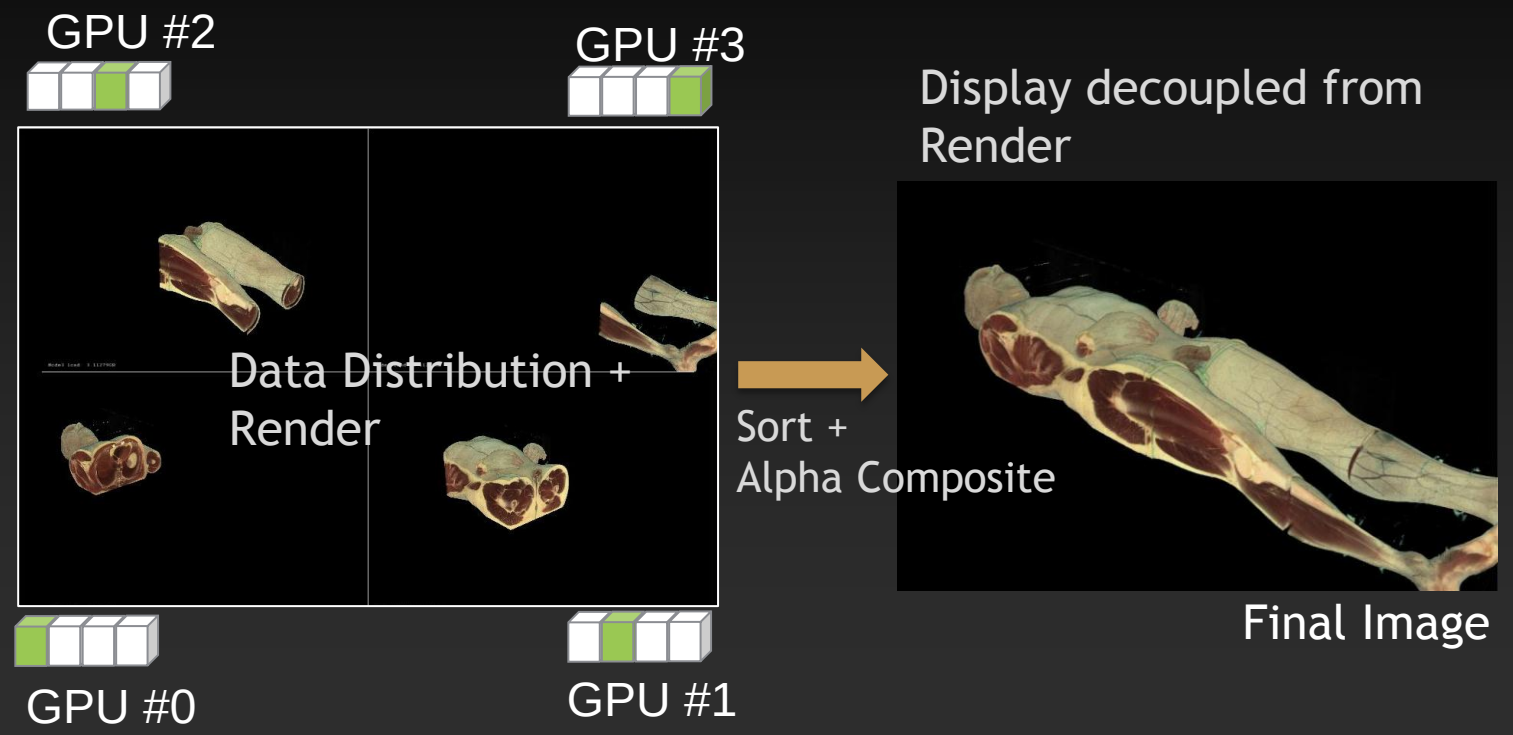
For #GPUs enumerated {

```
GpuMask[0]=hGPU[0];
GpuMask[1]=NULL;
//Get affinity DC based on GPU
HDC affinityDC = wglCreateAffinityDCNV(GpuMask);
setPixelFormat(affinityDC);
HGLRC affinityGLRC = wglCreateContext(affinityDC);
```

}

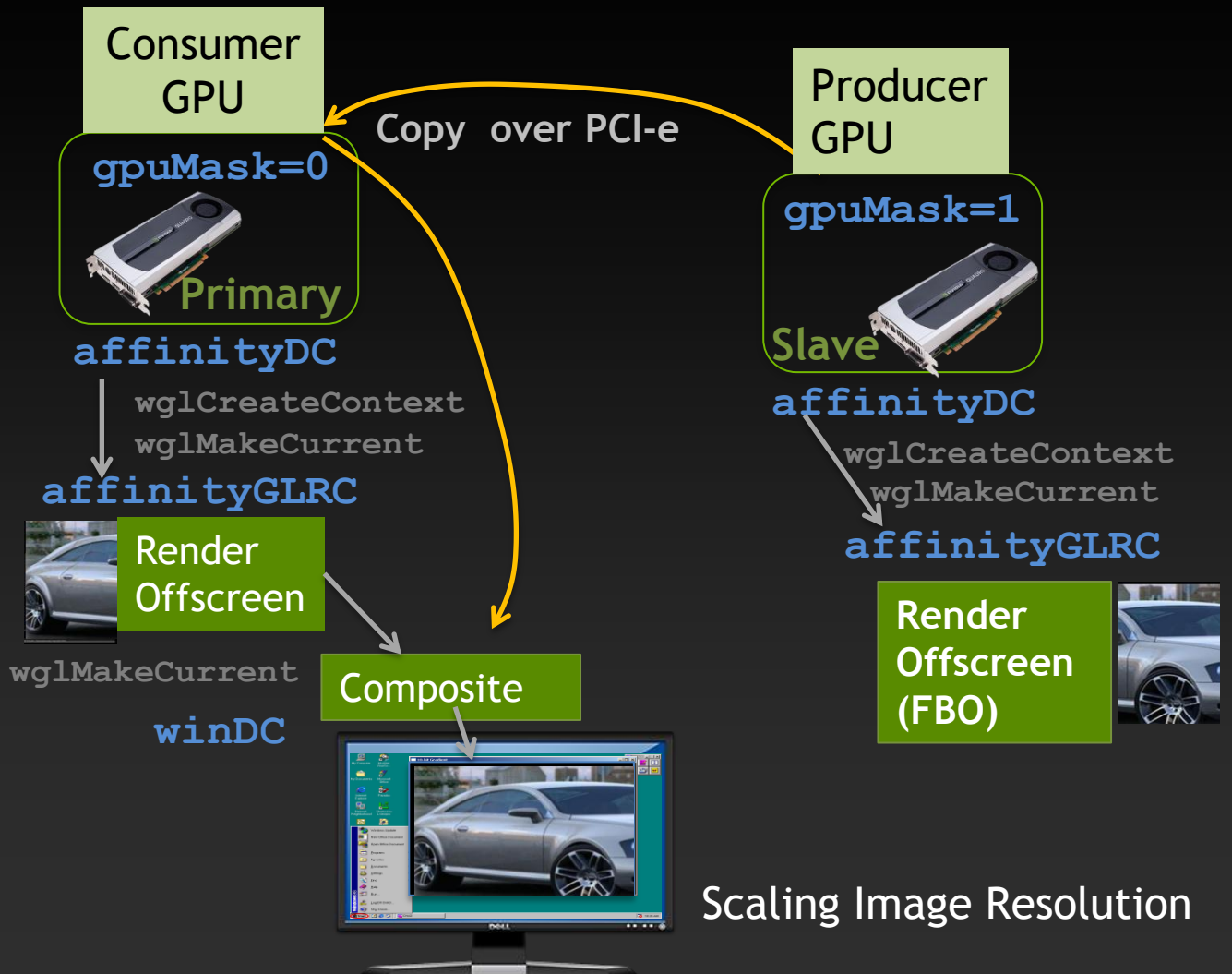
Scaling Rendering

- Scaling data size using Sort-Last approach
  - Eg Visible Human Dataset : 14GB 3D Texture rendered across 4GPUs



Using GPU Affinity

- App manages
  - Distributing render workload
  - implementing various composition methods for final image assembly
- InterGPU communication
- Data, image & task scaling

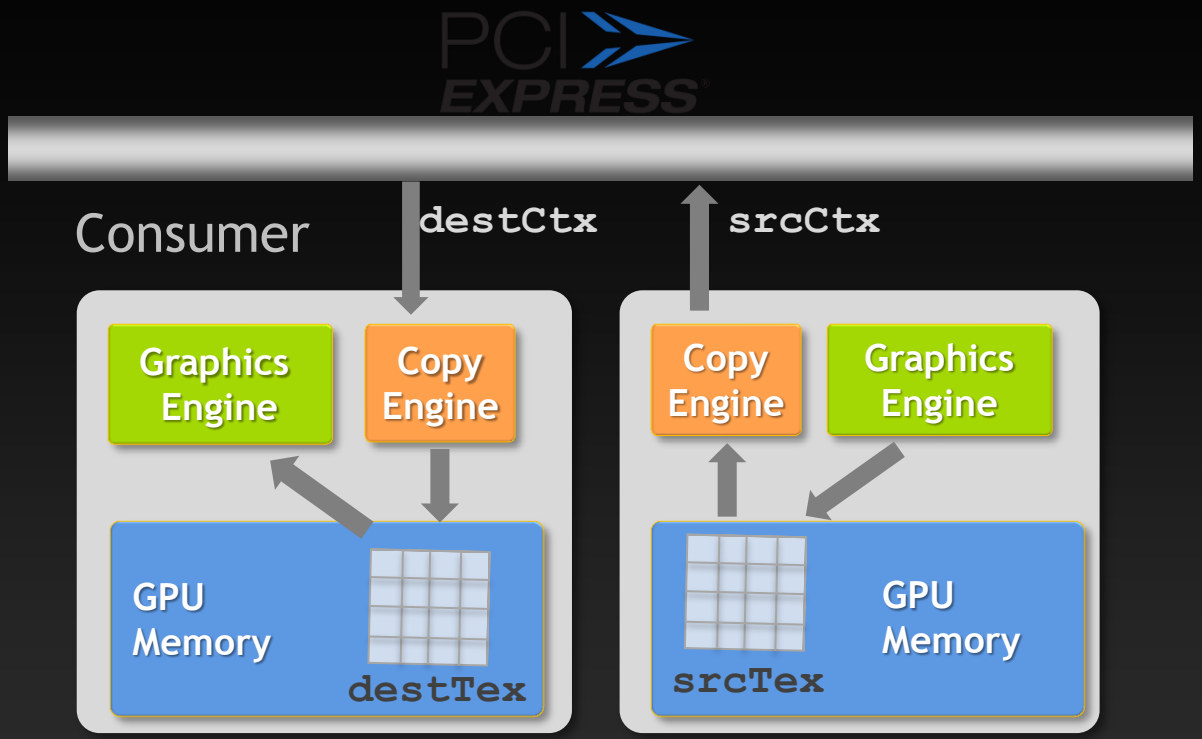


Sharing data between GPUs

- For multiple contexts on same GPU
  - `ShareLists & GL_ARB_Create_Context`
- For multiple contexts across multiple GPU
  - Readback (GPU_1-Host) → Copies on host → Upload (Host-GPU_0)
- `NV_copy_image` extension for OGL 3.x
  - Windows - `wglCopyImageSubData`
  - Linux - `glXCopyImageSubDataNV`
  - Avoids extra copies, same pinned host memory is accessed by both GPUs

NV_Copy_Image Extension

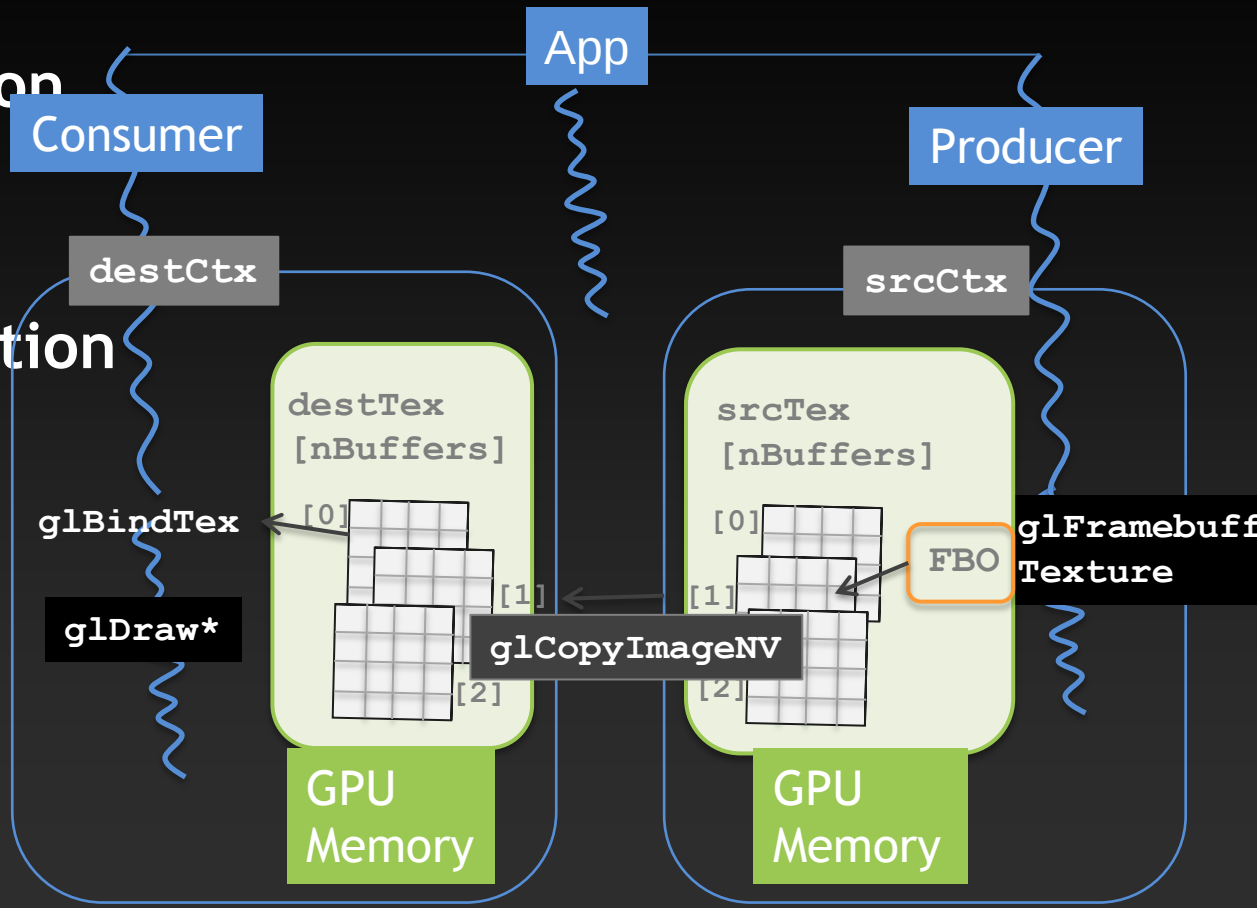
- Transfer in single call
  - No binding of objects
  - No state changes
  - Supports 2D, 3D textures & cube maps
- Async for Fermi & above
  - Requires programming




```
wglCopyImageSubDataNV(srcCtx, srcTex, GL_TEXTURE_2D,0, 0, 0, 0,
        destCtx, destTex, GL_TEXTURE_2D, 0, 0, 0, 0,
        width, height, 1);
```

Producer-Consumer Application Structure

- One thread per GPU to maximize CPU core utilization
- OpenGL commands are asynchronous
- Need GPU level synchronization
  - Use GL_ARB_SYNC
- Can scale to multiple producers/consumers



Applications : Texture/Geometry Scaling

- Adding more GPUs increases transfer time
  - But scales data size
- Full-res images transferred between GPUs
- Volumetric Data
  - Transfer RGBA images
- Polygonal Data (2X transfer overhead)
  - Transfer RGBA and Depth (32bit) images

Applications : Task Scaling

- Render scaling
  - Flight simulation, raytracing
- Server-side rendering
  - Assign GPU for a user depending on heuristics
  - Eg using `GL_NVX_MEMORY_INFO` to assign GPU

References

- **OpenGL Insights chapters**
  - Chapter 29 Fermi Asynchronous Texture Transfers
  - Chapter 27 - Multi-GPU Rendering on NVIDIA Quadro
  - Source Code - https://github.com/OpenGLInsights/OpenGLInsightsCode
- **GTC 2012 On-demand talks**
  - http://www.gputechconf.com/gtcnew/on-demand-gtc.php
  - S0353 - Programming Multi-GPUs for Scalable Rendering
  - S0356 - Optimized Texture Transfers