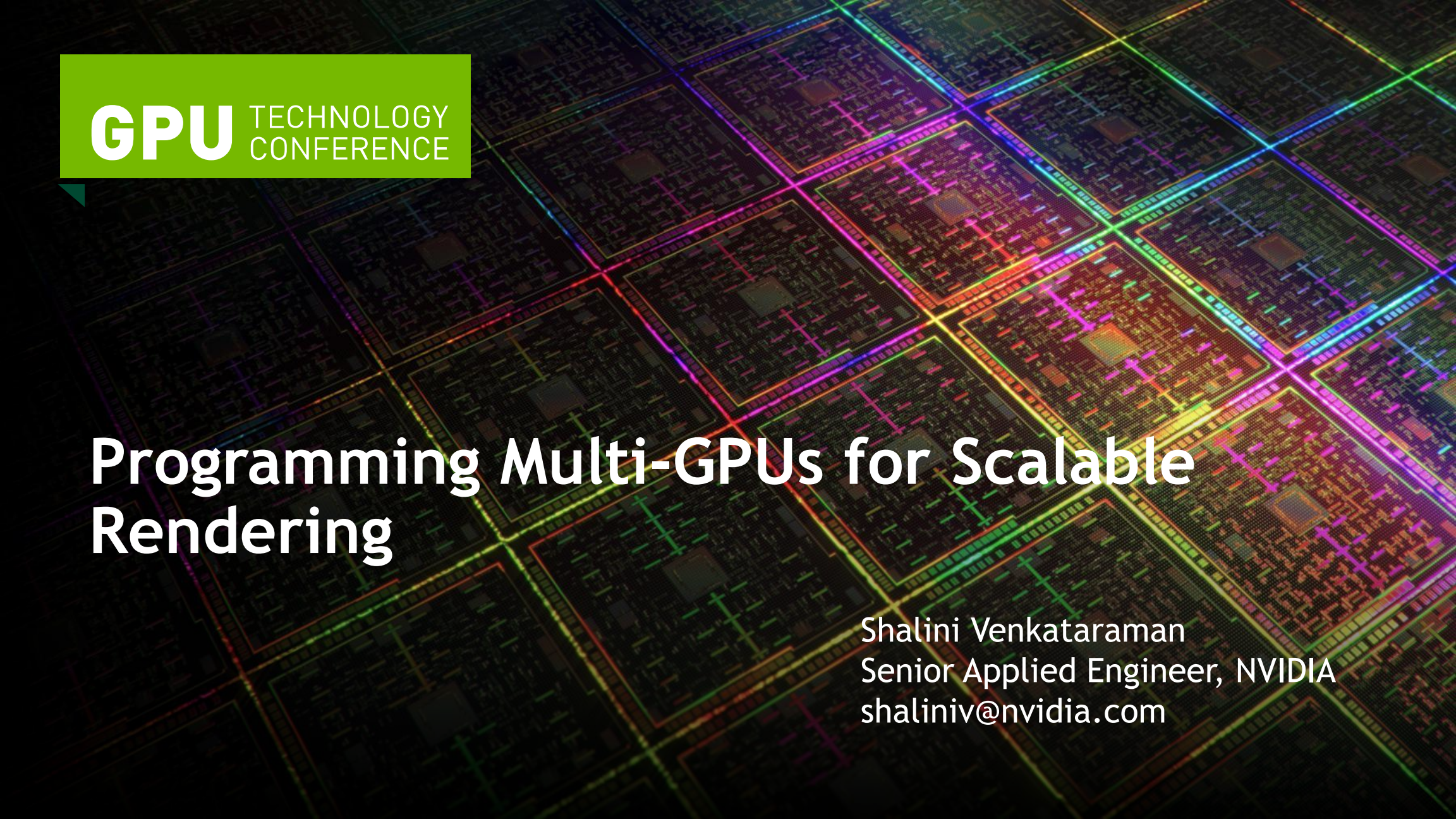
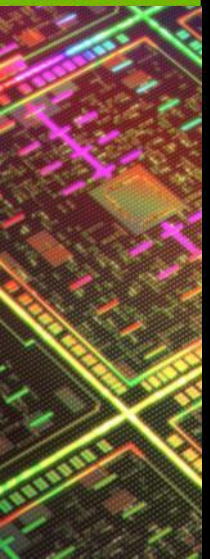


Programming Multi-GPUs for Scalable Rendering

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

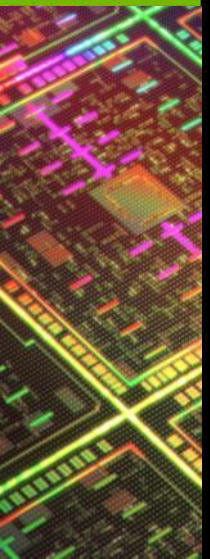


Scaling Graphics

- Focus on OpenGL graphics
- Onscreen Rendering
  - Display scaling for multi-projector, multi-tiled display environments
- 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



Scaling display - SLI Mosaic Mode

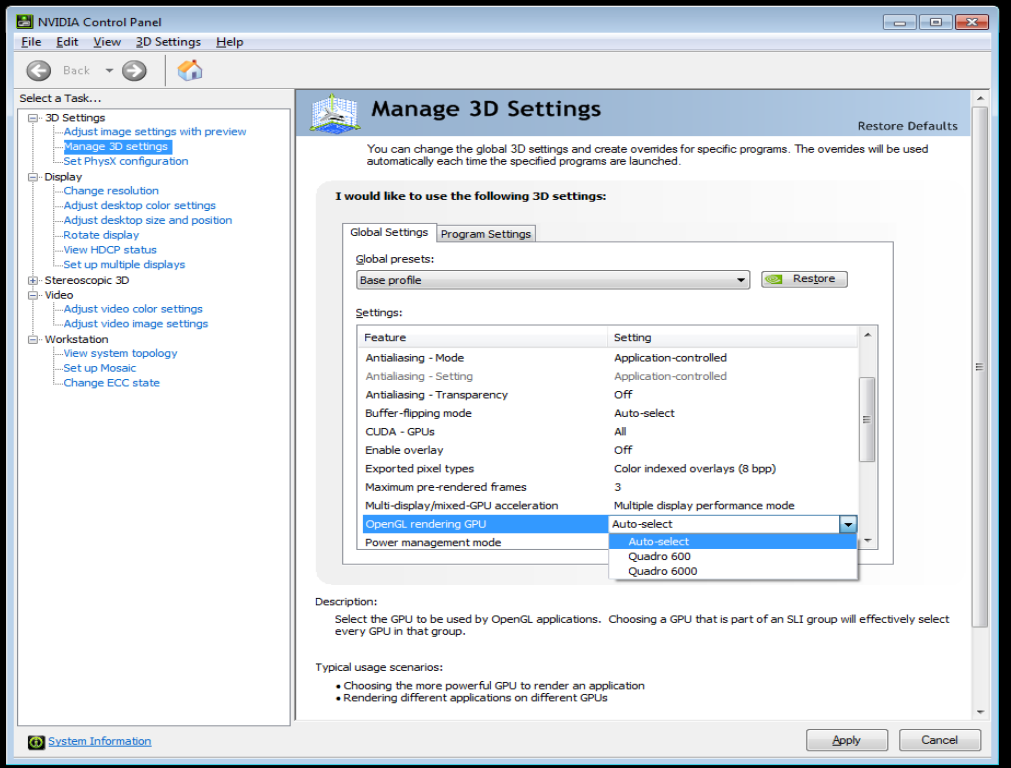
- Transparent
- Does frame synchronization
- Does fragment level clipping
- Disadvantages
  - Single view frustum
  - No geometry/vertex level clipping

Doug Traill, S0341-See the Big Picture Scalable Visualization Solutions for System Integrators, GTC 2012 Recordings

Specifying OpenGL GPU

- 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

http://developer.nvidia.com/nvapi

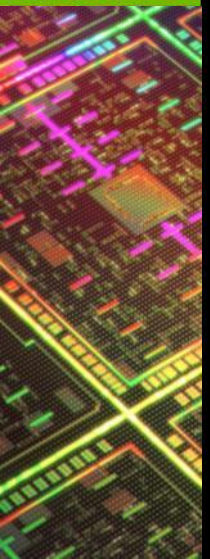


Programming for Multi-GPU

- 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

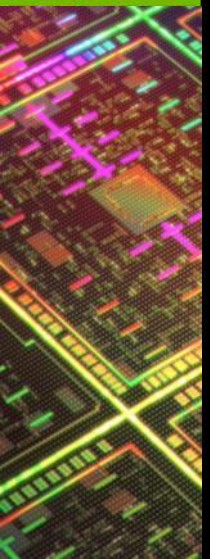
```
BOOL wglEnumGpusNV(UINT iGpuIndex, HGPUNV *phGPU)
```

- Enumerate Displays per GPU

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

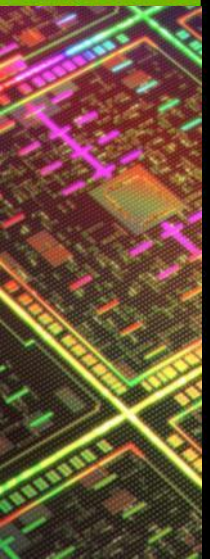
- Pinning OpenGL context to a specific GPU

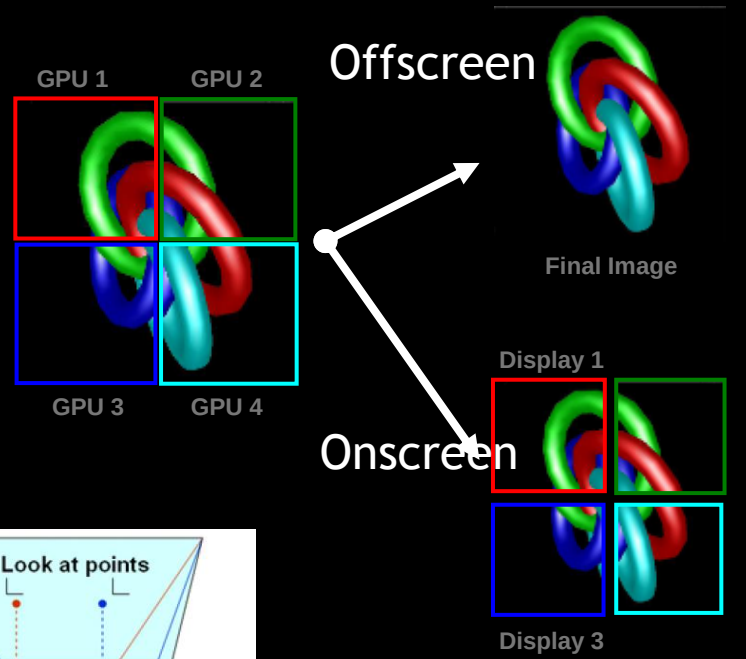
```
For #GPUs enumerated {
    GpuMask[0]=hGPU[0];
    GpuMask[1]=NULL;
    //Get affinity DC based on GPU
    HDC affinityDC = wglCreateAffinityDCNV(GpuMask);
    setPixelFormat(affinityDC);
    HGLRC affinityGLRC = wglCreateContext(affinityDC);
}
```

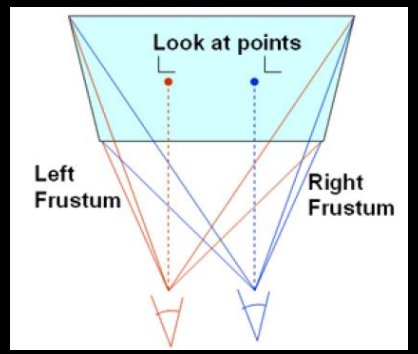


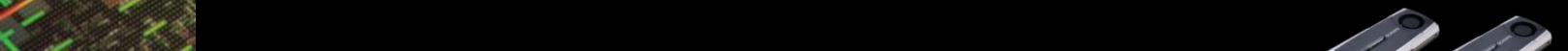
Scaling - Display

- Sort-First
  - Different GPUs render different portions on the screen
  - Data replicated across all GPUs
- Use cases
  - Fill rate bound apps like raytracing
  - Tiled walls, CAVEs
  - Stereo (needs gsync)



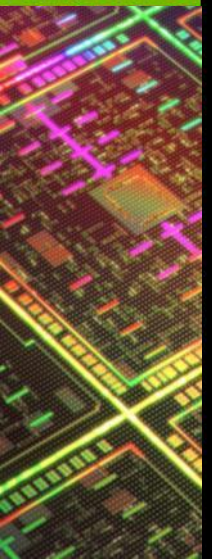


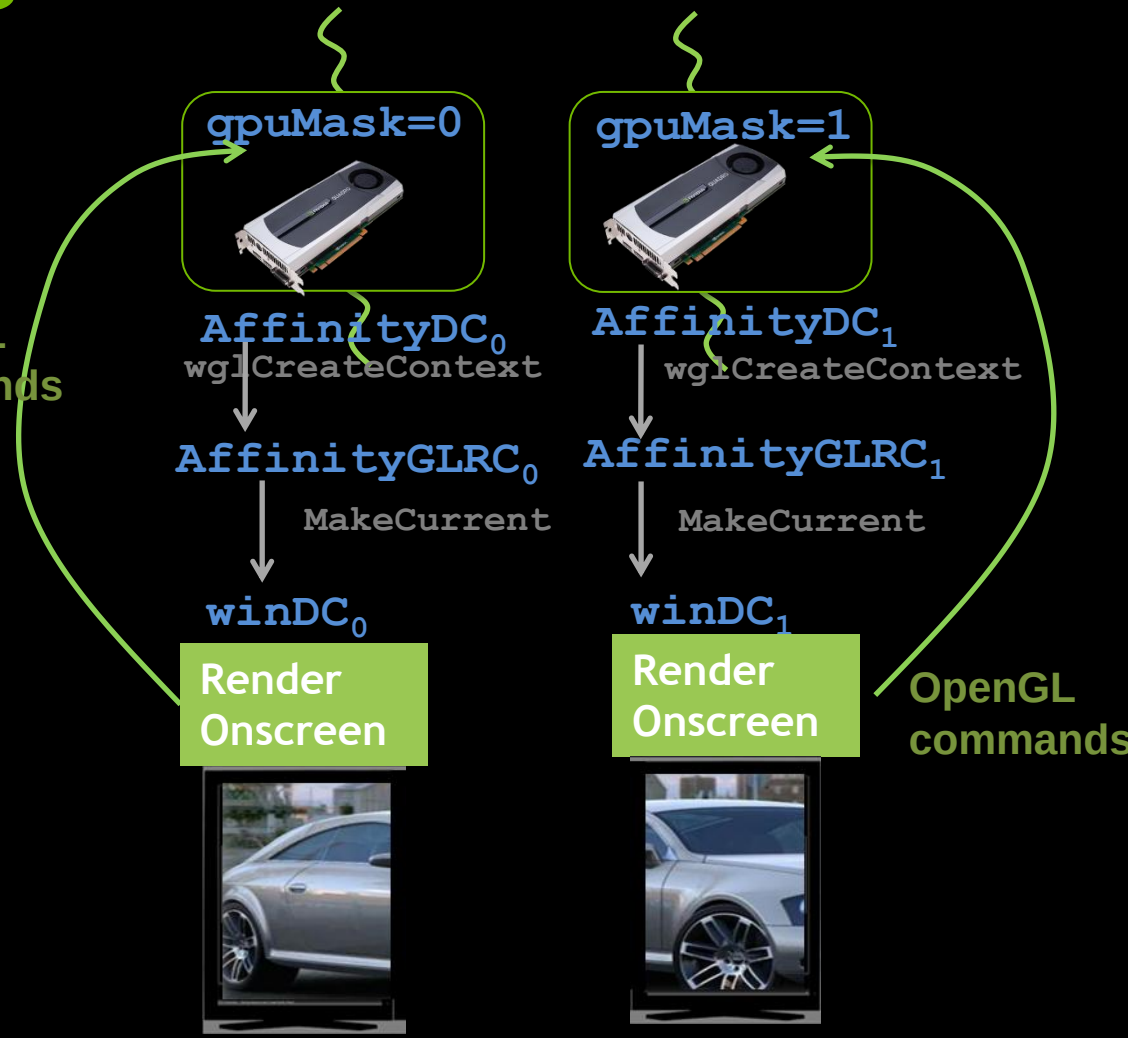




Onscreen Rendering - Overview

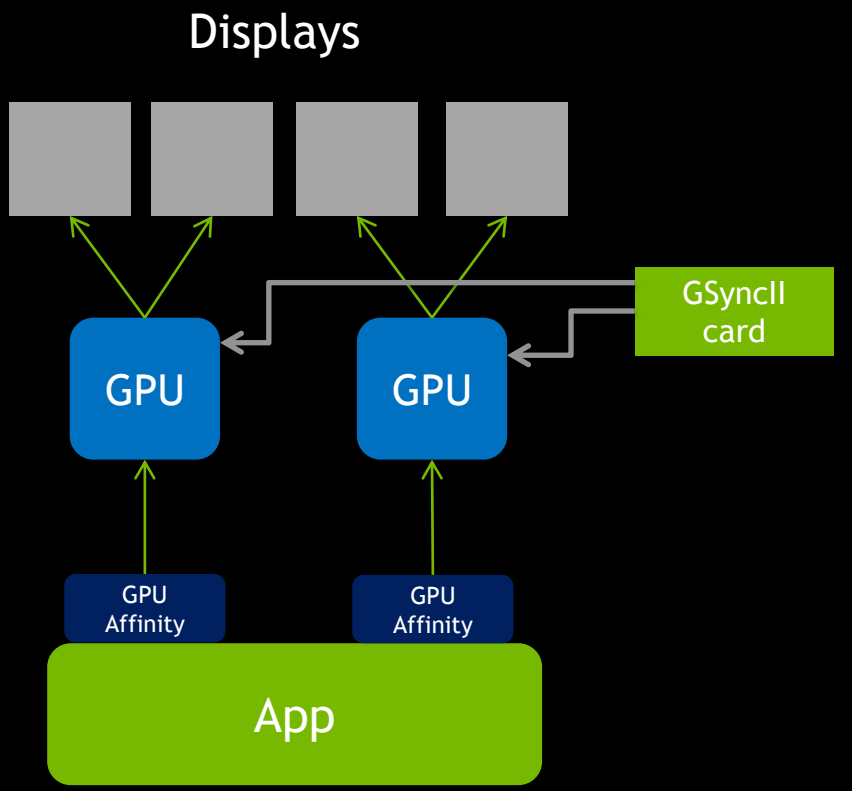
- Simple example of sort-first
- No Inter GPU communication
- Thread per GPU to keep hardware queue busy
- Totally programmable
  - Different view frustums
  - View specific optimizations



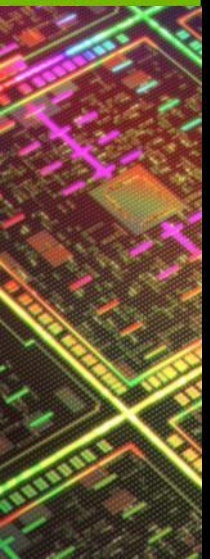


Adding Frame Synchronization

- Needs GSYNC for projection setups to avoid tearing
- *Framelock provides a* common sync signal between graphics cards to insure the vertical sync pulse starts at a common start.



Doug Traill, S0341-See the Big Picture Scalable Visualization Solutions for System Integrators, GTC 2012 Recordings



Onscreen rendering + Framelock

- NV_Swap_Group to sync buffers between GPUs (WGL & GLX)
  - Swap Groups : windows in a single GPU
  - Swap Barrier : Swap Groups across GPUs
- Init per window DC

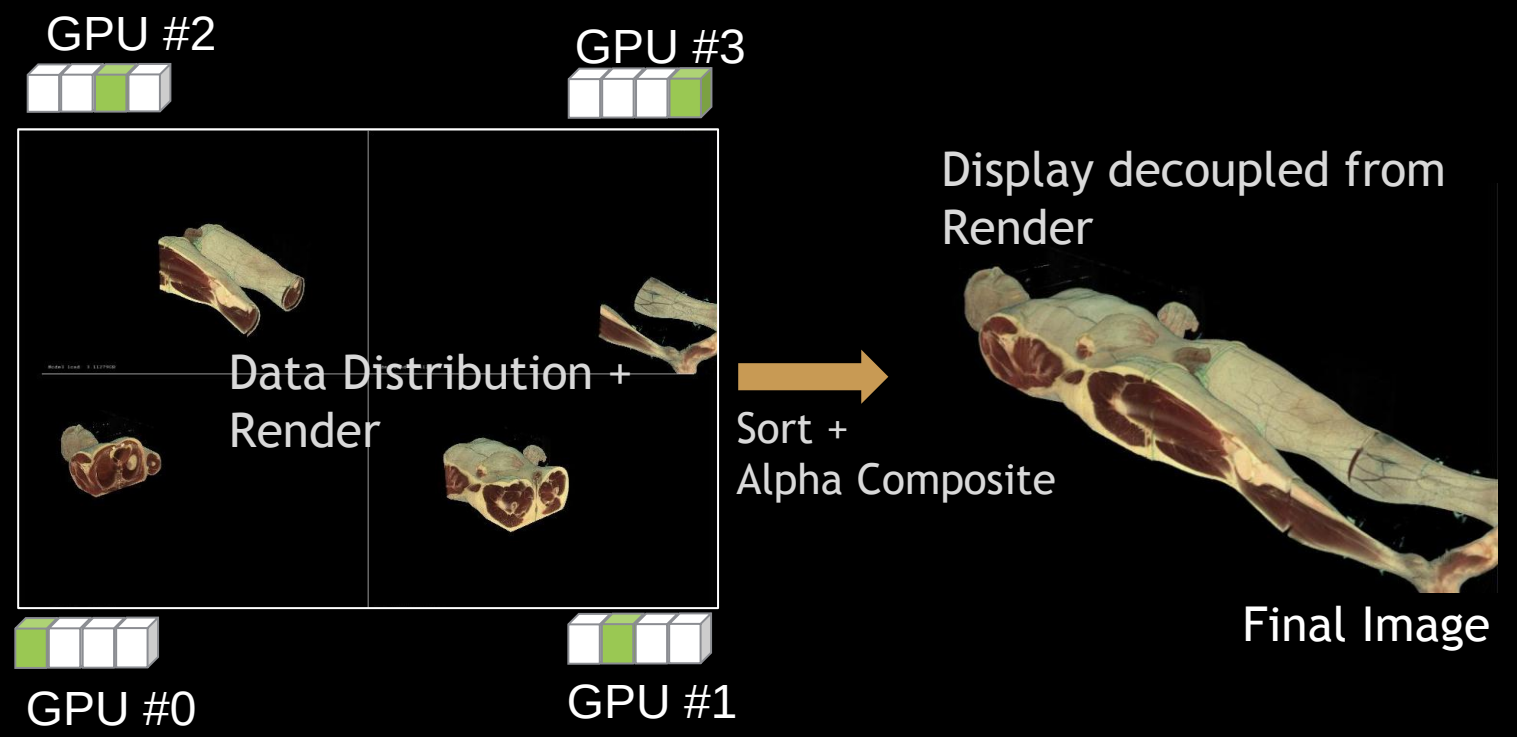
```
for (i=0; i< numWindows ; i++) {
    GLuint swapGroup    = 1;
    wglJoinSwapGroupNV(winDC[i], swapGroup)
    wglBindSwapBarrierNV(swapGroup, 1);
}
```

- Display for each window in a separate thread

```
void renderThreadFunc(int idx) {
    MakeCurrent(winDC[idx], affinityGLRC[idx])
    //Do Drawing, only on GPU idx
    SwapBuffers(winDC[idx]); //SYNC here for buffer swaps
}
```

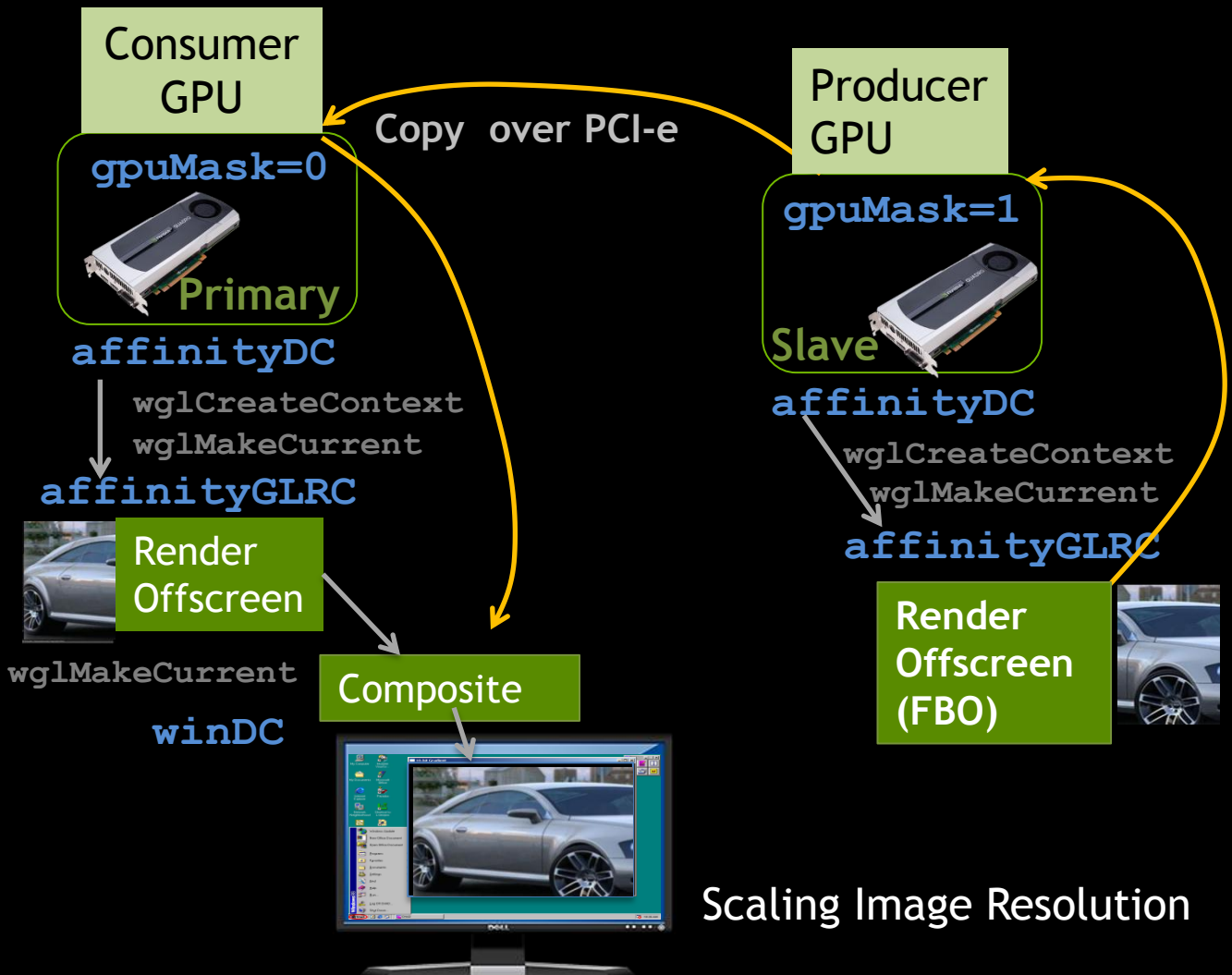
Offscreen Rendering

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



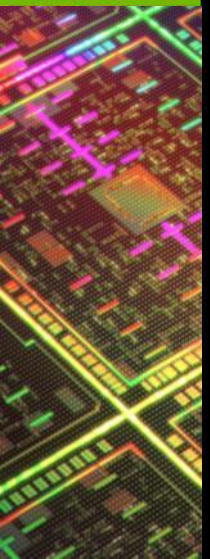
Offscreen Rendering

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



Sharing data between GPUs

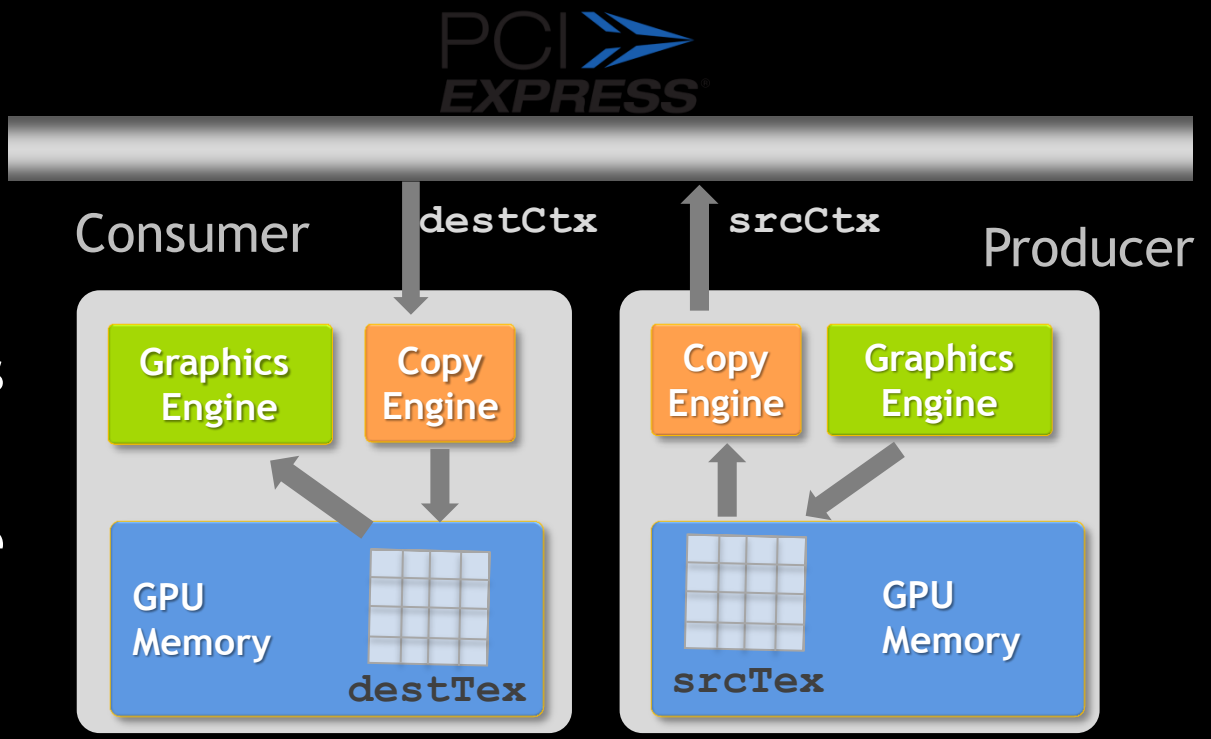
- For multiple contexts on same GPU
  - `ShareLists & GL_ARB_Create_Context`
- For multiple contexts across multiple GPU
  - Readback (GPU_1-Host) → Copies on host → Upload (Host-GPU_0)
- `NV_copy_image` extension for OGL 3.x
  - Windows - `wglCopyImageSubData`
  - Linux - `glXCopyImageSubDataNV`
  -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



NV_Copy_Image

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

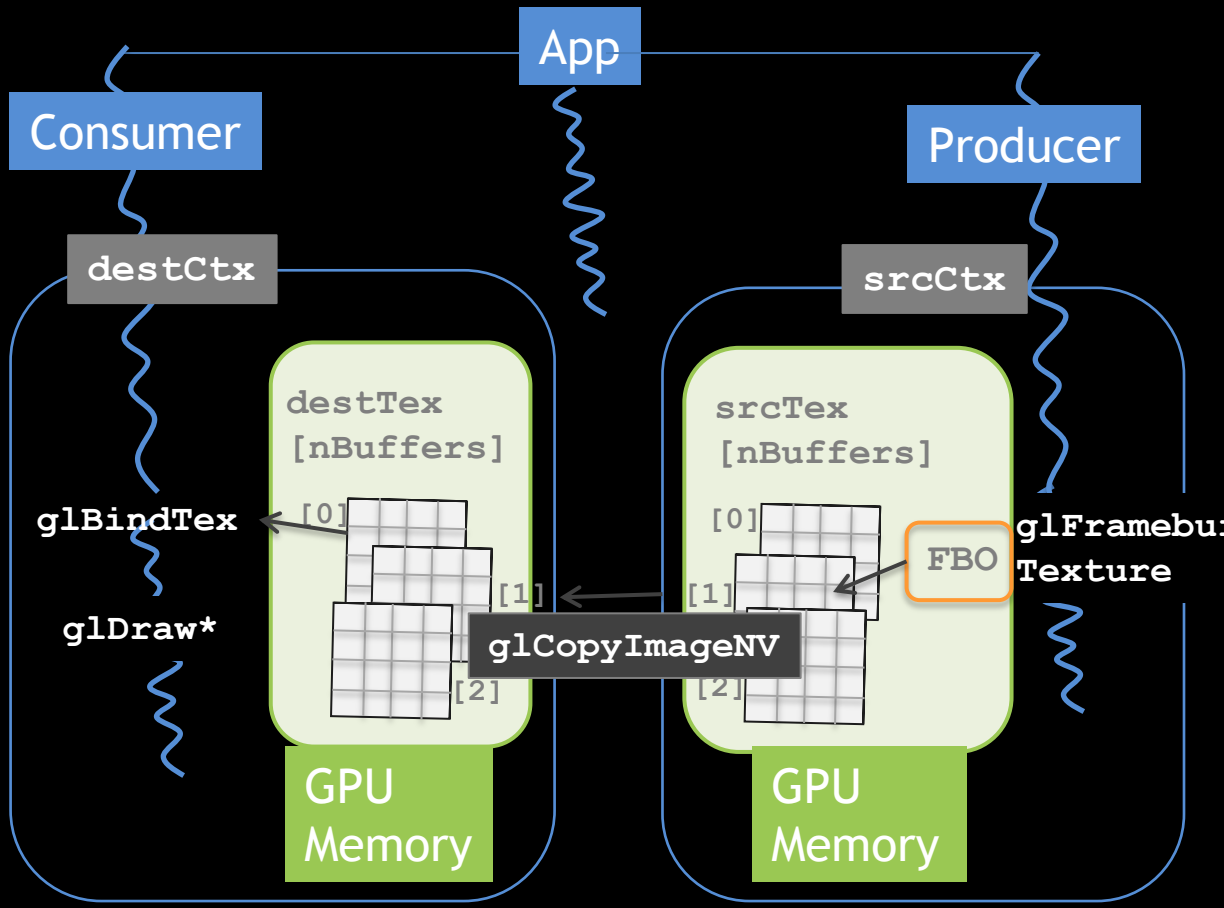
S0356 - Optimized Texture Transfers. GTC 2012 Recordings




```
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

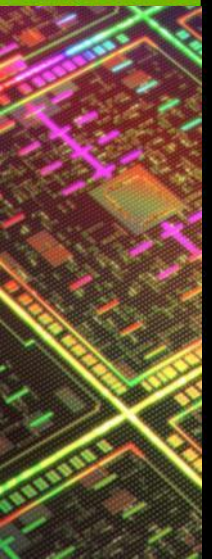
- One thread per GPU to maximize CPU core utilization
- OpenGL commands are asynchronous
- Need GPU level synchronization
  - Use GL_ARB_SYNC



OpenGL Synchronization

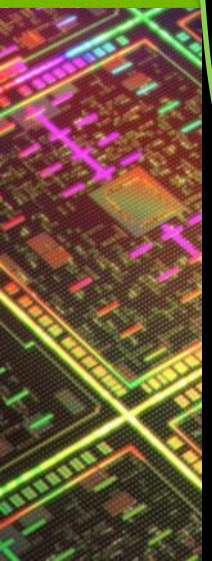
- OpenGL commands are asynchronous
  - When glDrawXXX returns, does not mean command is completed
- Sync object glSync (ARB_SYNC) is used for multi-threaded apps that need sync, Since OpenGL 3.2
  - Eg compositing texture on gpu-consumer waits for rendering completion on gpu-producer
- Fence is inserted in a nonsignaled state but when completed changed to signalled.

```
//Producer Context                          //Consumer Context
glDrawXX
NvCopyImage
GLSync fence = glFenceSync(..)   unsignalled   glWaitSync(fence)
                                 signalled     glBind
                                               Composite & draw
cpu work eg memcpy
```



Multi-GPU Synchronization

```
GLsync consumedFence[MAX_BUFFERS]; producedFence[MAX_BUFFERS];
HANDLE consumedFenceValid, producedFenceValid;
```



Consumer GPU

MakeCurrent(destCtx)

Render subtask offscreen

```
WaitForSingleObject(producedFenceValid)
glWaitSync(producedFence)
glBindTexture2D(destTex)
//Composite with offscreen image
consumedFence = glFence(..);
SetEvent(consumedFenceValid)
```

Draw onscreen

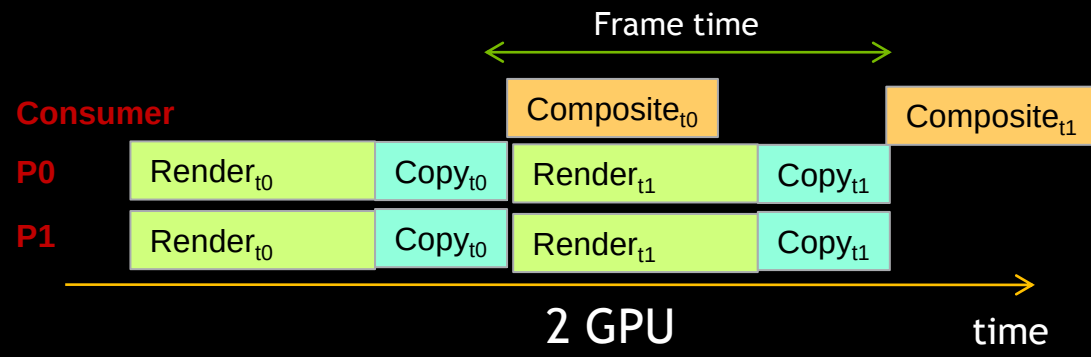
Producer GPU

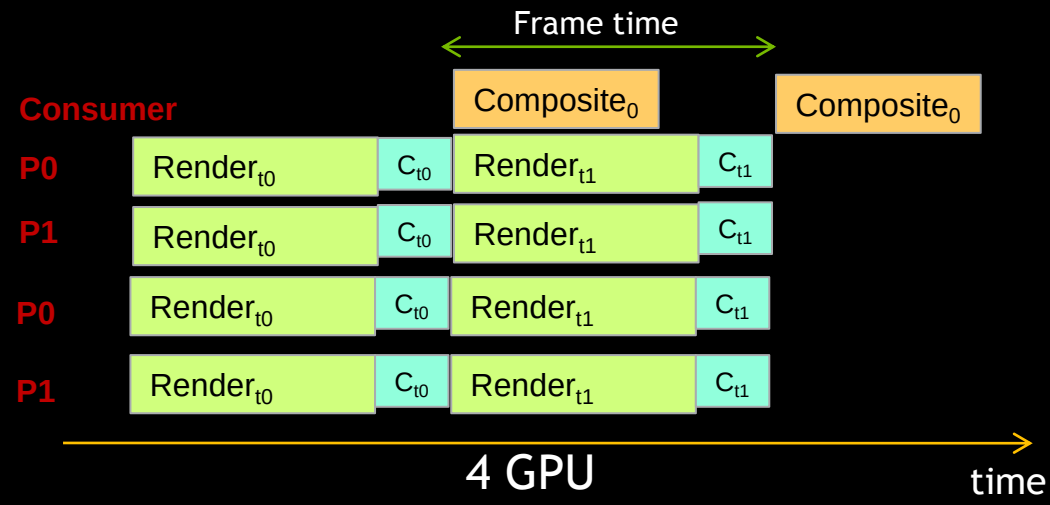
MakeCurrent(srcCtx)

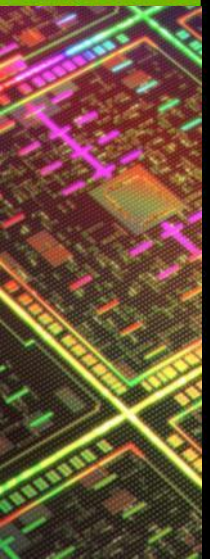
```
WaitForSingleObject(consumedFenceValid);
glWaitSync(consumedFence);
glFramebufferTexture2D(srcTex);
//Offscreen render to texture
glFramebufferTexture2D(0);
wglCopyImageSubDataNV(srcCtx,srcTex,
                      ..destCtx,destTex);
producedFence = glFence(..);
SetEvent(producedFenceValid);
```

Sort-first : Image Scaling

- Image Scaling
  - Each GPU works on a smaller subregion of final image
  - Adding more GPUs reduces transfer time per GPU
  - Total data transferred remains constant





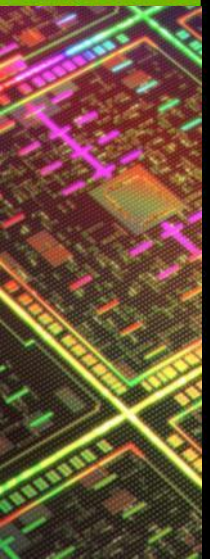


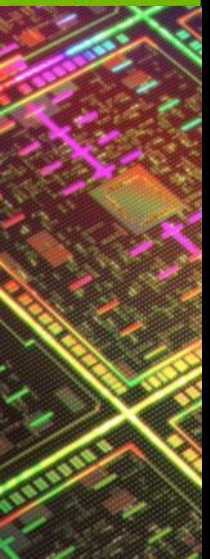
Sort-Last : Data Scaling

- 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

Task Scaling

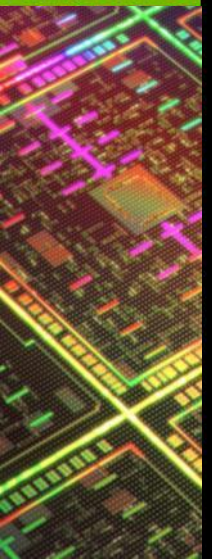
- Render scaling
  - Flight simulation, raytracing
- Server-side rendering
  - Assign GPU for a user depending on heuristics
  - Eg using `GL_NVX_MEMORY_INFO` to assign GPU





Middleware

- Equalizer
  - Scales from single-node multi-gpu to a multi-node cluster
  - Implements various load-balancing, image reassembly and composition optimization
  - Open Source - www.equalizergraphics.com
- CompleX
  - NVIDIA's implementation
  - Single system multi-GPU only
  - http://developer.nvidia.com/complex



Conclusions

- Multi-GPUs for scaling OpenGL graphics for
  - Multi-display cases (onscreen rendering)
  - Various image/data/task parallelism (offscreen rendering)
- Other relevant multi-gpu talks
  - **S0342 - Volumetric Processing and Visualization on Heterogeneous Architecture**
  - **S0507 - Interactive and Scalable Subsurface Data Visualization Framework**
  - **S0267A - Mixing Graphics and Compute with Multiple GPUs**