MULTI-GPU GRAPHICS PROGRAMMING
Take advantage of all the GPU power available

Christopher Mayer, Amaury Le Leyzour
AMD
ISV Engineer
INTRODUCTION | Ways of Using Multiple GPUs

- **AMD Crossfire™**
  - Transparent to the application
  - Alternate Frame Rendering

- **AMD GPU Association**
  - Needs to be implemented by the application
  - In addition to a main context, an associated context on another GPU can be used for off-screen rendering

- **Multiple windows on multiple GPUs**
  - Needs to be implemented by the application
  - A window / context is created per GPU
AMD GPU ASSOCIATION

- Provides functionality to use an additional GPU for off-screen rendering

- Good choice to implement well known techniques like:
  - 2D Decomposition
  - 3D Decomposition
  - Eye Decomposition for stereo rendering
  - …

- Shadow maps can be rendered on slave GPU and blitted into the main context
AMD GPU ASSOCIATION | Multi GPU Shadow Rendering

- Demo from 3DInteractive GmbH
- Rendering of a complex geometry bound scene
- Shadowmaps are rendered on second GPU using AMD GPU Association
To efficiently use multiple GPUs in parallel, one thread per context and GPU should be used.

The Master thread creates a window.

The Slave thread will create the associated context.

Use `WGL_AMD_GPU_association`:
- to pick GPU
- To create associated context
- To blit results between GPUs

Use `GL_ARB_sync` to sync between threads.

Use Semaphores to synchronize execution order:
- e.g. Master needs to finish window creation before the slave can create the context.
### AMD GPU ASSOCIATION | Select a GPU

- **Querying the extension**
  - WGL_AMD_gpu_association
  - Query via `wglGetExtensionStringARB`

- **GPU Count**
  - `UINT wglGetGPUIDsAMD( UINT maxCount, UINT* Ids );`
    - Lists the GPU IDs for all the GPUs

- **Get the GPU ID of a context**
  - `INT wglGetContextGPUIDAMD(HGLRC hglrc);`

- **GPU Properties**
  - `INT  wglGetGPUInfoAMD(UINT id, WGL_ENUM property, ENUM dataType, UINT size, void *data)`
    - Helps get information about a GPU like the fastest GPU, Memory on the GPU or the OpenGL version supported on that specific GPU
**AMD GPU ASSOCIATION | Context Creation**

- Normally a context is associated with the card attached to the display in which the window was created
  - `WGL_AMD_gpu_association` lets you associate a context to a GPU, not a display

- Create the off screen context associated to a GPU we selected as our target
  - Its simple form:
    - `HGLRC wglCreateAssociatedContextAMD( UINT gpuid)`
  - Or using an attribute list:
    - `HGLRC wglCreateAssociatedContextAttribsAMD( UINT gpuid, HGLRC hShareContext, const int *attribList);`

- Now, to make the context current
  - `BOOL wglMakeAssociatedContextCurrentAMD( HGLRC hglrc );`
**AMD GPU ASSOCIATION | Blit between contexts**

- Sharing pixel data between contexts
  - `void wglBlitContextFramebufferAMD(HGLRC dstCtx,
    GLint srcX0, GLint srcY0, GLint srcX1, GLint srcY1,
    GLint dstX0, GLint dstY0, GLint dstX1, GLint dstY1,
    GLbitfield mask, GLenum filter);`

  - Mask lets you specify what to transfer (color, depth and/or stencil)
  - The behavior of this blit follows the specs defined in `EXT_framebuffer_bit`
    - The source context (current context) cannot be used as the destination context
    - Make sure the proper frame buffers are bound (`GL_DRAW_FRAMEBUFFER_EXT` and `GL_READ_FRAMEBUFFER_EXT`)

- The cost in time to copy data from one GPU to another is not insignificant. Because of this, it is important to plan what rendering should be done on remote GPUs, leaving time for copies to the main GPU.
An example:
   – Select an OpenGL 4.1 capable GPU

```c
UINT offscreenGPU = -1, gpuIds[4];
UINT nGPUs = wglGetGPUIDsAMD( sizeof( gpuIds)/sizeof(UINT), gpuIds );
UINT nMainGPUId = wglGetContextGPUIDAMD(g_hRCMain);
for( GLuint i=0; i< nGPUs; i++ ) {
    if( gpuids[i] != nMainGPUId) { // We need a different GPU than main char versionString[64];
        wglGetGPUInfoAMD( gpuIds[i], WGL_GPU_OPENGL_VERSION_STRING, GL_UNSIGNED_CHAR, 64, versionString );
        if(versionString[0] > '4' || (versionString[0]== '4' && versionString[3] >= '1') )
            { // Check for at least OpenGL 4.1
                // This is the one my app needs to select
                offscreenGPU = gpuIds[i];
                break;
            }
}
```

**AMD GPU ASSOCIATION | Example**

- An example:
  - Create the context on the OpenGL4.1 GPU we selected

```c
int attribList[] = {
    WGL_CONTEXT_MAJOR_VERSION_ARB, 4,
    WGL_CONTEXT_MINOR_VERSION_ARB, 1,
    NULL
};

HGLRC hOffScreenCtx= wglCreateAssociatedContextAttribsAMD(offscreenGPU, NULL, attribList);
wglMakeAssociatedContextCurrentAMD( hOffScreenCtx );
```

- Now let’s render to our off-screen context

```c
UINT nShadowPassFBO, nShadowPassRBO;
genFramebuffers(1, &nShadowPassFBO );
genRenderbuffers(1, &nShadowPassRBO );
bindFramebuffer( GL_DRAW_FRAMEBUFFER, nShadowPassFBO );
bindRenderbuffer( GL_RENDERBUFFER, nShadowPassRBO );
RenderbufferStorage( GL_RENDERBUFFER, 1, DEPTH_COMPONENT24, 1024, 768);
FramebufferRenderbuffer( GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, GL_RENDERBUFFER, nShadowPassRBO );
// Begin off-screen rendering
```
Setup the main context

```c
wglMakeCurrent(g_hDC, hRCMain);
  // Setup Main context
UINT nRemoteDataFBOName, nRemoteDataRBOName;
glGenFramebuffers(1, &nRemoteDataFBO);
glGenRenderbuffers(1, &nRemoteDataRBO);
glBindFramebuffer(GL_DRAW_FRAMEBUFFER, nRemoteDataFBO);
glBindRenderbuffer(GL_RENDERBUFFER, nRemoteDataRBO);
glRenderbufferStorage(GL_RENDERBUFFER, 1, DEPTH_COMPONENT24, 1024, 768);
glfwFramebufferRenderbuffer(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, GL_RENDERBUFFER, nShadowPassRBO);
```
Synchronizing data transfer between contexts
- See GL_ARB_sync (OpenGL 3.3)
- Simply insert a sync object in our example

```c
wglMakeAssociatedContextCurrentAMD(hOffScrCtx);
// Render to FBO
...
// Wait for Master to be ready
WaitForSingleObject(gMasterReady, INFINITE);
// Copy result to main context
wglBlitContextFramebufferAMD(hRCMain, 0, 0, 1024, 768,
                           0, 0, 1024, 768,
                           GL_DEPTH_BUFFER_BIT, GL_NEAREST);

// Insert Fence
remoteFence = glFenceSync(GL_SYNC_GPU_COMMANDS_COMPLETE, 0);
```
In the main thread

```c
// Main rendering loop
...
// Bind FBO to receive data
glBindFramebuffer(GL_DRAW_FRAMEBUFFER, nRemoteDataFBO);
// Signal ready to receive data
ReleaseSemaphore(gMasterReady, 1, 0);
// Now the transfer can take place, wait for fence
GLenum syncResult = glGet waitForSync(remoteFence, GL_SYNC_FLUSH_COMMAND_BIT, 0);
if ( syncResult == GL_CONDITION_SATISFIED || syncResult == GL_ALREADY_SIGNALED) {
    // Rendering complete and result ready
}
else
    if ( syncResult == GL_TIMEOUT_EXPIRED || syncResult == GL_WAIT_FAILED) {
        // Error occurred
    }
```
MULTIPLE WINDOWS ON MULTIPLE GPUS

- Can be used to render to multiple display
- Synchronization of the SwapBuffers is required
- In combination with Eyefinity, a large number of displays can be driven by one system
- On windows
  - When you open a window, it shows up on a display
  - Your context is automatically associated to one of the GPUs on this adapter
  - Can be a DirectX®/D3D or OpenGL window
  - The application can choose which GPU to use based on the coordinates of the window
MULTIPLE WINDOWS ON MULTIPLE GPUs | SIGGRAPH 2010

- 40 Monitors driven by 4 systems
- 2 systems with 2 x AMD FirePro™ V8800
- 2 systems with 3 x AMD FirePro™ V8800
MULTIPLE WINDOWS ON MULTIPLE GPUS | IBC 2010

- 12 Monitors driven by 1 systems
- 1 systems with 2 x AMD FirePro V9800

IBC Demo video on fireuser.com
MULTIPLE WINDOWS ON MULTIPLE GPUs | Synchronization

- To avoid any kind of tearing effects when rendering in parallel to multiple windows on different GPUs, the SwapBuffers needs to be synchronized.
- AMD S400 can be used to synchronize the video signal and the SwapBuffers of multiple GPUs.
- Up to 4 GPUs per system can be synchronized.
- Several systems can be connected.
- All GPUs in all systems will be in sync.
The `WGL_NV_Swap_group` extension provides functionality to synchronize the SwapBuffer.

After binding to a barrier, the SwapBuffers will be synchronized.

```c
const GLuint nGroup = 1;
const GLuint nBarrier = 1;
GLuint nMaxBarriers, nMaxGroups;

if (!wglQueryMaxSwapGroupsNV(mhDC, &nMaxGroups, &nMaxBarriers))
    return false;

if (nMaxGroups > 0 && nMaxBarriers > 0)
{
    // In this sample we always join to SwapGroup 1
    // and Barrier 1
    if (!wglJoinSwapGroupNV(mhDC, nGroup))
        return false;

    if (!wglBindSwapBarrierNV(nGroup, nBarrier))
        return false;
}
```
MULTIPLE WINDOWS ON MULTIPLE GPUS | Select a GPU – Win32

- When you open a window on a display, the GPU associated will automatically be selected to the graphics card the monitor is attached to.
- To perform GPU parallelization choose on which display you want to open your windows
  - One display per graphics card + one window per display = perfect parallelization
- Win32 offers functions to enumerate the displays
  - user32.lib: Xp/Vista/W7
    - BOOL EnumDisplayMonitors( HDC hdc, LPRECT lprcClip, MONITORENUMPROC lpfnEnum, LPARAM dwData );

```c
BOOL CALLBACK EnumMonitor( HMONITOR, HDC hdcMonitor, LPRECT lprcMonitor, LPARAM dwData )
{
    HWND hwnd= CreateWindow( ...,lprcMonitor->left, lprcMonitor->top, lprcMonitor->right - lprcMonitor->left,
                             lprcMonitor->bottom - lprcMonitor->top, ... );
    HGLRC hrc= wglCreateContext( GetDC( hwnd) );
    wglMakeCurrent( hdc, hrc);
}
EnumDisplayMonitors( NULL, NULL, EnumMonitor, NULL );
```

- EnumDisplayMonitors enumerates the monitors attached, but does not know about the hardware devices which control them.
MULTIPLE WINDOWS ON MULTIPLE GPUS | Select a GPU – Win32

- Win32 offers functions to enumerate the displays
  - user32.lib: Xp/Vista/W7
    - BOOL EnumDisplayDevices( LPCSTR lpDevice, DWORD iDevNum,
                               LPDISPLAY_DEVICE lpDisplayDevice, DWORD dwFlags );
    - Along with: BOOL EnumDisplaySettings( LPCSTR lpDevice, DWORD iModeNum, LPDEVMODE lpDevMode );
    - Offers a lot more info:

```c
DISPLAY_DEVICE device;
for( DWORD nMonitor=0; EnumDisplayDevices( NULL, nMonitor, &device, EDD_GET_DEVICE_INTERFACE_NAME ); nMonitor++ )
{
  DEVMODE devMode;
  if( (device.stateFlag & DISPLAY_DEVICE_ACTIVE)== DISPLAY_DEVICE_ACTIVE
      && EnumDisplaySettings( device.DeviceName, ENUM_CURRENT_SETTINGS, &devMode ) ) {
      HWND hwnd= CreateWindow( …., devMode.dmPosition.x, devMode.dmPosition.y,
                               devMode.dmPelsWidth, devMode.dmPelsHeight, … );
      HGLRC hrc= wglCreateContext( GetDC( hwnd ) );
      wglMakeCurrent( hdc, hrc );
  }
}
```

- Still does not know about the graphics devices
MULTIPLE WINDOWS ON MULTIPLE GPUS | Select a GPU – ADL

- ADL (AMD Display Library) provides an interface to all display related information
- ADL is available on all platforms

```c
for (int i = 0; i < nNumAdapters; ++i)
{
    int nAdapterIdx;
    int nAdapterStatus;

    nAdapterIdx = pAdapterInfo[i].iAdapterIndex;
    ADL_Adapter_Active_Get(nAdapterIdx, &nAdapterStatus);

    if (nAdapterStatus)
    {
        LPADLDisplayInfo pDisplayInfo = NULL;

        ADL_Display_DisplayInfo_Get(nAdapterIdx, &nNumDisplays, &pDisplayInfo, 0);

        for (int j = 0; j < nNumDisplays; ++j)
        {
            // check if display is connected and mapped
            if (pDisplayInfo[j].iDisplayInfoValue &
                ADL_DISPLAY_DISPLAYINFO_DISPLAYCONNECTED)
            {
            }

            // check if display is mapped on adapter
            if (pDisplayInfo[j].iDisplayInfoValue &
                ADL_DISPLAY_DISPLAYINFO_DISPLAYMAPPED &&
                pDisplayInfo[j].displayID.iDisplayLogicalAdapterIndex == nAdapterIdx)
            {
                ...
            }
        }
    }
}
```
MULTIPLE WINDOWS ON MULTIPLE GPUs | Select a GPU – ADL
MULTI GPU ENVIRONMENT

- References
  - WGL_AMD_gpu_association
    - [http://www.opengl.org/registry/specs/AMD/wgl_gpu_association.txt](http://www.opengl.org/registry/specs/AMD/wgl_gpu_association.txt)
  - WGL_NV_swap_group
    - [http://www.opengl.org/registry/specs/NV/wgl_swap_group.txt](http://www.opengl.org/registry/specs/NV/wgl_swap_group.txt)
  - ADL
    - [http://developer.amd.com/sdks/ADLSDK/Pages/default.aspx](http://developer.amd.com/sdks/ADLSDK/Pages/default.aspx)
Disclaimer & Attribution
The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions and typographical errors.

The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. There is no obligation to update or otherwise correct or revise this information. However, we reserve the right to revise this information and to make changes from time to time to the content hereof without obligation to notify any person of such revisions or changes.

NO REPRESENTATIONS OR WARRANTIES ARE MADE WITH RESPECT TO THE CONTENTS HEREOF AND NO RESPONSIBILITY IS ASSUMED FOR ANY INACCURACIES, ERRORS OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION.

ALL IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE ARE EXPRESSLY DISCLAIMED. IN NO EVENT WILL ANY LIABILITY TO ANY PERSON BE INCURRED FOR ANY DIRECT, INDIRECT, SPECIAL OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREFOR, EVEN IF EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

AMD, AMD CrossFire, AMD FirePro, the AMD arrow logo, and combinations thereof are trademarks of Advanced Micro Devices, Inc. All other names used in this presentation are for informational purposes only and may be trademarks of their respective owners.

DirectX is a registered trademark of Microsoft Corporation.

© 2011 Advanced Micro Devices, Inc. All rights reserved.
MULTI GPU ENVIRONMENT | Select a GPU – DXGI

- Win32/COM offers functions to enumerate the displays
  - dxgi.lib on Vista/W7
    - Offers adapters enumeration
      - BOOL EnumAdapters(UINT adapter, IDXGIAdapter** ppAdapter);
      - Enumerates the graphics adapters
    - BOOL IDXGIAdapter::GetDesc( LPDXGI_ADAPTER_DESC pAdapterDesc);
      - Gets some information about this adapter
    - BOOL IDXGIAdapter::EnumOutputs( UINT Output, IDXGIOutput **ppOutput);
      - Enumerates the monitors attached to this adapter
    - BOOL IDXGIOutput::GetDesc( LPDXGI_OUTPUT_DESC pOutputDesc );
      - Gets a bunch of info about this monitor

- DXGI is part of DirectX, but does not create any D3D context or anything DirectX specific
  - And is totally compatible with an OpenGL application
  - Can be used for D3D programming as WGL_AMD_gpu_association does not exist in DirectX
MULTI GPU ENVIRONMENT | Select a GPU – DXGI

- Win32 offers functions to enumerate the displays
  - D3gi.lib on Vista/W7 – An example:

```c
IDXGIFactory *pFactory;
CreateDXGIFactory(__uuid(IDXGIFactory), (void**)&pFactory);
IDXGIAdapter* pAdapter;
for( DWORD nAdapter=0; pFactory->EnumAdapters(nAdapter, &pAdapter)!=DXGI_ERROR_NOT_FOUND; nAdapter++ )
  { DXGI_ADAPTER_DESC adapterDesc;
    if( pAdapter->GetDesc( & adapterDesc ) == S_OK ) {
      IDXGIOutput* pOutput;
      for( DWORD nOutput=0; pAdapter->EnumOutputs( nOutput, &pOutput )!=DXGI_ERROR_NOT_FOUND; nOutput++ ) {
        DXGI_OUTPUT_DESC outputDesc;
        if( pOutput->GetDesc(&outputDesc) == S_OK )
          CreateWindow( ..., outputDesc.DesktopCoordinates.left, outputDesc.DesktopCoordinates.top,
          outputDesc.DesktopCoordinates.right - outputDesc.DesktopCoordinates.left,
          outputDesc.DesktopCoordinates.bottom- outputDesc.DesktopCoordinates.top, ... );
        }
      }
    }
```