OPENGL®/DIRECTX® - OPENCL™ INTEROPERABILITY

Benjamin Coquelle
AMD
AGENDA

- OpenCL + Graphics API running on Discrete GPU
- APU + discrete GPU
- APU + discrete GPU, Pinned Buffer
OPENCL RUNNING ON DISCRETE GPU
With OpenCL arrival, everyone can now use GPU for general purposes.

The GPU can now be used to compute the geometry based on real physics information (CAD/CAE simulation).

Or animation (DCC animation).

As a result, we want to share the same buffer between OpenCL and OpenGL/DX to avoid data transfer throughout PCI express bus with **Zero-copy** data sharing between APIs.
DETECTING OPENGL/D3D SUPPORT

- Verify GL sharing is supported
  - Get extension string for `CL_DEVICE_EXTENSIONS`
  - "cl_khr_gl_sharing" ! // Linux + Windows

- Verify D3D sharing is supported
  - Get extension string for `CL_DEVICE_EXTENSIONS`
  - "cl_khr_d3d10_sharing"

- All of our graphics devices supports these extensions
OPENCL/OPENGL CONTEXT CREATION

- To share buffer with OpenGL or D3D, you need to create an OpenCL context with your OpenGL/D3D context/device before creating any OpenGL/D3D objects:
  - OpenGL windows:
    ```c
    HGLRC glCtx = wglGetCurrentContext();
    cl_context_properties cpsGL[] = { CL_CONTEXT_PLATFORM, (cl_context_properties)platform,
                                     CL_WGL_HDC_KHR, (intptr_t) wglGetCurrentDC(), CL_GL_CONTEXT_KHR, (intptr_t) glCtx, 0};
    context = clCreateContextFromType(cpsGL, dType, NULL, NULL, &status);
    ```
  - OpenGL Linux:
    ```c
    GLXContext glCtx = glXGetCurrentContext();
    cl_context_properties cpsGL[] = { CL_CONTEXT_PLATFORM, (cl_context_properties)platform,
                                     CL_GLX_DISPLAY_KHR, (intptr_t) glXGetCurrentDisplay(), CL_GL_CONTEXT_KHR, (intptr_t) glCtx, 0};
    context = clCreateContextFromType(cpsGL, dType, NULL, NULL, &status);
    ```
OPENCL/D3D CONTEXT CREATION

- Direct3D:

```c
cl_context_properties cps[] =
{
    CL_CONTEXT_PLATFORM, (cl_context_properties)platform, CL_CONTEXT_D3D10_DEVICE_KHR,
    (intptr_t)d3d10DevicePtr, 0
};
context = clCreateContextFromType(cps, dType, NULL, NULL, &status);
```
**SHARING BUFFER (OPENGL)**

- OpenGL and OpenCL can share OpenGL buffer
  - You need to create your OpenCL buffer from an OpenGL buffer

- OpenCL and OpenGL are communicating, so your OpenGL buffer won’t be deleted/update until your corresponding OpenCL buffer is released

- You can now update your buffer using OpenCL and draw it using OpenGL
  - Your VBO certainly needs to be created as GL_DYNAMIC_DRAW, because OpenCL will update it frequently

- This functionality only requires OpenGL 1.5 (Buffer objects have included in OpenGL 1.5)
SHARING BUFFER (OPENGL), CALL STACK

```c
glGenBuffers(1, &MyGL_BO);

MyClBuffer = clCreateFromGLBuffer(MyCL_context, flags, MyGL_BO, status);

cEnqueueAcquireGLObjects(commandqueue, nbGLObjects, pCLObject, NB_EventInWaitList, pWaitingEvent, pEvent);

cEnqueueNDRangeKernel(...);

cEnqueueReleaseGLObjects(commandqueue, NbObjects, pCLObject, NB_EventInWaitList, pWaitingEvent, pEvent);
```
SHARING BUFFER (D3D10)

- You can do the exact same thing on D3D10
  
  ```c
  cl_mem clCreateFromD3D10BufferKHR (cl_context context, cl_mem_flags flags, ID3D10Buffer *resource, cl_int *errcode_ret)
  ```

- You will find the same properties than in OpenGL interoperability

- Like OpenGL, you need to acquire and release the resource before working on it on a command queue:
  
  ```c
  cl_int clEnqueueAcquireD3D10ObjectsKHR (cl_command_queue command_queue, cl_uint num_objects, const cl_mem *mem_objects, cl_uint num_events_in_wait_list, const cl_event *event_wait_list, cl_event *event)
  ```

  ```c
  cl_int clEnqueueReleaseD3D10ObjectsKHR (cl_command_queue command_queue, cl_uint num_objects, const cl_mem *mem_objects, cl_uint num_events_in_wait_list, const cl_event *event_wait_list, cl_event *event)
  ```
SHARING IMAGE (OPENGL)

- Image can be created from 2D and 3D texture and renderbuffer
- If you are using TBO, you need to create an OpenCL buffer and you cannot use images

```c
cl_mem clCreateFromGLTexture2D (cl_context context, cl_mem_flags flags, GLenum texture_target, GLint mipmaplevel, GLuint texture, cl_int *errcode_ret)
```

```
texture_target: GL_TEXTURE_{2D, RECTANGLE}, GL_TEXTURE_CUBE_MAP_POSITIVE_{X, Y, Z},
GL_TEXTURE_CUBE_MAP_NEGATIVE_{X, Y, Z}
```

```
cl_mem clCreateFromGLTexture3D (cl_context context, cl_mem_flags flags, GLenum texture_target, GLint mipmaplevel, GLuint texture, cl_int *errcode_ret)
```

```
cl_mem clCreateFromGLRenderbuffer (cl_context context, cl_mem_flags flags, GLuint renderbuffer, cl_int *errcode_ret)
```
SHARING TEXTURE-IMAGE (D3D10)

- Again you have the same features than in OpenGL

```c
cl_mem clCreateFromD3D10Texture2DKHR (cl_context context, cl_mem_flags flags, ID3D10Texture2D *resource, UINT subresource, cl_int *errcode_ret)
```

```c
cl_mem clCreateFromD3D10Texture3DKHR (cl_context context, cl_mem_flags flags, ID3D10Texture3D *resource, UINT subresource, cl_int *errcode_ret)
```
OPENCL AND OPENGL SYNCHRONIZATION

- `clFinish` and `glFinish` are for the moment the best way to ensure all OpenCL and OpenGL command have been executed and we will not read or write from a part of memory which is currently use for something else.

- The extension `cl_khr_gl_event` and `GL_ARB_cl_event`, written against OpenCL 1.1 and OpenGL 3.2, will bring a more efficient way to synchronize buffer shared between OpenGL and OpenCL:
  
  - `sync glCreateSyncFromCLEventARB(cl_context context, cl_event event, bitfield flags)` create a sync OpenGL object from a OpenCL event. Therefore we can make sure a specific action in a command queue is over before running OpenGL calls.
  
  - `cl_event clCreateEventFromGLSyncKHR (cl_context context, GLSync sync, cl_int *errcode_ret)` provides the complementary functionality.

APU + DISCRETE GPU
BENEFIT OF APU AND DISCRETE GPU

- What we presented will give you a boost in performance if OpenCL and your graphics API work on the same data.
  - No data transfer through PCI-E and zero copy inside GPU memory
- If your OpenCL kernel runs on data your graphics API don’t need it will slow down your graphics performance
- Because of the new APU technology we can now also use it to offload some OpenCL work on it. This will relieve the GPU.
- On such systems the discrete GPU can perform drawing while the APU is running OpenCL kernels.
- Data transfer for OpenCL is no longer an issue since the APU is accessing directly the system memory.
  - We can achieve 17GB/s compare to the 8GB/s theoretical limit of PCI-E
USING WITH APU/DISCRETE GPU COMBO

- OpenCL does not update memory used for immediate drawing
  - i.e OpenCL works on data not related to drawing or data that will be used later for drawing (2-3 frames)

- APU can be used to run OpenCL kernel

- Discrete GPU will be used for drawing
APU + DISCRETE GPU
PINNED BUFFER
**What it is**
- GPU can access any system memory with asynchronous DMA transfer

**Benefit**
- Increase memory transfer to GPU
- Increase performance when application cannot create static VBO during animation
- Flexibility of immediate mode, buffers are modified on system memory

**Details**
- Requires OpenGL 2.1
- Based on new AMD extension

![Diagram showing UNB / MC, DDR3 DIMM Memory, PCIe, and GPU Chip with GPU connected to memory with an ~17 GB/sec bandwidth.](image-url)
PINNED BUFFER

- GPU can access any system memory with asynchronous DMA transfer

APU-based Platform

- 3X bandwidth between GPU and memory
- Eliminate latency and power associated with the extra chip crossing
It is now easy to understand how APU + discrete GPU can take advantage of this new technology.

The APU will compute data needed to be drawn by the discrete GPU:
- APU will only need to create an OpenCL context
- APU and discrete GPU will not share the same "OpenCL/OpenGL resources"
- GPU can access directly the system memory where the APU wrote into

Developers only need to make sure the APU finishes its work before using GPU on that memory and also make sure the GPU finishes drawing before using the APU to make any update.
CONTEXT WHERE SUCH ARCHITECTURE CAN BE USEFUL

- Animation in DCC space
- Simulation in CAD/CAE
- Oil & gas (streaming data)
- Medical images (streaming data)
- ...

[Image: Fusion 11]

[Logo: AMD Developer Summit]
QUESTIONS ?
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 HEREIN, EVEN IF EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

AMD, 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.

OpenCL is a trademark of Apple Inc. used with permission by Khronos.

2011 Advanced Micro Devices, Inc. All rights reserved.