OpenCL: Graphics Interop

The best of both worlds -- graphics and compute
Graphics & Compute

Best of both worlds

- Choose the most appropriate API for your app
- Add a kernel anywhere in your graphics pipeline
- Zero-copy data sharing between APIs
- Minor overhead associated with context switching
Detecting OpenGL Support

Verify GL sharing is supported

- Get extension string for **CL_DEVICE_EXTENSIONS**
- Check for the appropriate platform extension string:
  - "cl_khr_gl_sharing"   // Linux + Windows
  - "cl_APPLE_gl_sharing" // Apple OSX / iOS
#if defined (__APPLE__) || defined(MACOSX)
  static const char* CL_GL_SHARING_EXT = "cl_APPLE_gl_sharing";
#else
  static const char* CL_GL_SHARING_EXT = "cl_khr_gl_sharing";
#endif

// Get string containing supported device extensions
int ext_size = 1024;
char* ext_string = (char*)malloc(ext_size);
err = clGetDeviceInfo(device_id, CL_DEVICE_EXTENSIONS, ext_size, ext_string, &ext_size);

// Search for GL support in extension string (space delimited)
int supported = IsExtensionSupported(CL_GL_SHARING_EXT, ext_string, ext_size);
if( supported )
{
  // Device supports context sharing with OpenGL
  printf("Found GL Sharing Support!\n");
}

Code Example: check for GL sharing support
```c
int IsExtensionSupported(
    const char* support_str, const char* ext_string, size_t ext_buffer_size)
{
    size_t offset = 0;
    const char* space_substr = strstr(ext_string + offset, " ", ext_buffer_size - offset);
    size_t space_pos = space_substr ? space_substr - ext_string : 0;
    while (space_pos < ext_buffer_size)
    {
        if( strncmp(support_str, ext_string + offset, space_pos) == 0 )
        {
            // Device supports requested extension!
            printf("Info: Found extension support ‘%s’!\n", support_str);
            return 1;
        }
        // Keep searching -- skip to next token string
        offset = space_pos + 1;
        space_substr = strstr(ext_string + offset, " ", ext_buffer_size - offset);
        space_pos = space_substr ? space_substr - ext_string : 0;
    }
    printf("Warning: Extension not supported ‘%s’!\n", support_str);
    return 0;
}
```

**Code Example:** verify extension is supported
Setting Up a Shared Context

Use GL devices in OpenCL context

- Get the handle for your windowing framework
- Use platform context properties to indicate GL interop
- Create an OpenCL context using active GL devices
// Create CL context properties, add WGL context & handle to DC
cl_context_properties properties[] = {
    CL_GL_CONTEXT_KHR, (cl_context_properties)wglGetCurrentContext(), // WGL Context
    CL_WGL_HDC_KHR, (cl_context_properties)wglGetCurrentDC(), // WGL HDC
    CL_CONTEXT_PLATFORM, (cl_context_properties)platform, // OpenCL platform
    0
};

// Find CL capable devices in the current GL context
cl_device_id devices[32]; size_t size;
clGetGLContextInfoKHR(properties, CL_DEVICES_FOR_GL_CONTEXT_KHR,
    32 * sizeof(cl_device_id), devices, &size);

// Create a context using the supported devices
int count = size / sizeof(cl_device_id);
cl_context context = clCreateContext(properties, devices, count, NULL, 0, 0);
```c
// Create CL context properties, add GLX context & handle to DC
cl_context_properties properties[] = {
    CL_GL_CONTEXT_KHR, (cl_context_properties)glXGetCurrentContext(), // GLX Context
    CL_GLX_DISPLAY_KHR, (cl_context_properties)glXGetCurrentDisplay(), // GLX Display
    CL_CONTEXT_PLATFORM, (cl_context_properties)platform,                // OpenCL platform
    0
};

// Find CL capable devices in the current GL context
cl_device_id devices[32]; size_t size;
cGetGLContextInfoKHR(properties, CL_DEVICES_FOR_GL_CONTEXT_KHR,
    32 * sizeof(cl_device_id), devices, &size);

// Create a context using the supported devices
int count = size / sizeof(cl_device_id);
cl_context context = clCreateContext(properties, devices, count, NULL, 0, 0);
```
Apple Shared Context Creation

// Get current CGL Context and CGL Share group
CGLContextObj kCGLContext = CGLGetCurrentContext();
CGLShareGroupObj kCGLShareGroup = CGLGetShareGroup(kCGLContext);

// Create CL context properties, add handle & share-group enum
cl_context_properties properties[] = {
    CL_CONTEXT_PROPERTY_USE_CGL_SHAREGROUP_APPLE,
    (cl_context_properties)kCGLShareGroup, 0
};

// Create a context with device in the CGL share group
cl_context context = clCreateContext(properties, 0, 0, NULL, 0, 0);
Apple Shared Context Creation

w/CPU + GPU

// Get current CGL Context and CGL Share group
CGLContextObj kCGLContext = CGLGetCurrentContext();
CGLShareGroupObj kCGLShareGroup = CGLGetShareGroup(kCGLContext);

// Create CL context properties, add handle & share-group enum
cl_context_properties properties[] = {
  CL_CONTEXT_PROPERTY_USE_CGL_SHAREGROUP_APPLE,
  (cl_context_properties)kCGLShareGroup, 0
};

// Optional: Get the CPU device (we can request this in addition to GPUs in Share Group)
cl_device_id cpu_device; int count;
clGetDeviceIds(platform, CL_DEVICE_TYPE_CPU, 1 * sizeof(cl_device_id), &cpu_device, & count);

// Create a context from a CGL share group (note: only use CPU if software renderer is enabled!)
cl_context context = clCreateContext(properties, count, cpu_device, NULL, 0, 0);
Sharing Resources

Using GL objects in OpenCL

• Create objects in GL like normal
• Create reference object in OpenCL
• Switch ownership from GL to OpenCL to use
• Release reference in OpenCL first then destroy in GL
OpenGL Texture: OpenCL Image

// Create a texture in OpenGL and allocate space
glGenTextures(1, &gl_texture_id);
glBindTexture(gl_texture_target, gl_texture_id);
glTexImage2D(gl_texture_target, 0, gl_texture_internal, width, height, 0,
             gl_texture_format, gl_texture_type, NULL);

// Create a reference mem object in OpenCL from GL texture
cl_mem cl_image = clCreateFromGLTexture2D(cl_context, CL_MEM_READ_WRITE,
                                           gl_texture_target, 0, gl_texture_id, &err);

if (!cl_image || err != CL_SUCCESS)
{
    printf("Failed to create OpenGL texture reference! %d\n", err);
    return -1;
}
GL buffer sharing

// Create a buffer object in OpenGL and allocate space
glGenBuffers(1, &gl_buffer_id);
glBindBuffer(GL_ARRAY_BUFFER_ARB, gl_buffer_id);

// Note: specify GL_STATIC_DRAW_ARB to modify outside of GL
glBufferData(GL_ARRAY_BUFFER_ARB, bytes, NULL,
GL_STATIC_DRAW_ARB);

// Note: could use colors, normals, etc
glVertexPointer(4, GL_FLOAT, 0, 0);
glBindBuffer(GL_ARRAY_BUFFER_ARB, 0);

// Create a reference cl_mem object from GL buffer object
cl_mem cl_buffer = clCreateFromGLBuffer(cl_context, CL_MEM_READ_WRITE,
gl_buffer_id, &err);

OpenGL Buffer: OpenCL Buffer

Code Example: GL buffer sharing
OpenCL + GL Execution

Intermixing command streams

- Synchronising commands between OpenCL & GL

OpenCL v1.0: `glFlush` to exec pending commands

OpenCL v1.1: Create `cl_event` from `GL_ARB_sync`
Switching Ownership from GL to OpenCL

// Force pending GL commands to get executed so memory is up-to-date
glFlush();

// Acquire ownership of GL texture for OpenCL Image
err = clEnqueueAcquireGLObjects(cl_cmd_queue, 1, &cl_image, 0, 0, 0);

// ... execute kernel or other OpenCL operations ...

// Release ownership of GL texture for OpenCL Image
err = clEnqueueReleaseGLObjects(cl_cmd_queue, 1, &cl_image, 0, 0, 0);

// Force pending CL commands to get executed
err = clFlush(cl_cmd_queue);

// Bind GL texture and use for rendering
glBindTexture(gl_texture_target, gl_texture_id);
Using an OpenCL Buffer to update an OpenGL texture

```c
// Acquire ownership of GL texture for OpenCL Image
err = clEnqueueAcquireGLObjects(cl_cmd_queue, 1, &cl_image, 0, 0, 0);

size_t origin[] = { 0, 0, 0 };
size_t region[] = { Width, Height, 1 };

// Copy contiguous buffer to formatted image bound to GL texture
err = clEnqueueCopyBufferToImage(cl_cmd_queue, gl_texture, cl_image,
                                  0, origin, region, 0, NULL, 0);

// Release ownership of GL texture for OpenCL Image
err = clEnqueueReleaseGLObjects(cl_cmd_queue, 1, &cl_image, 0, 0, 0);
if (err != CL_SUCCESS)
{
    printf("Failed to release GL object! %d\n", err);
    return EXIT_FAILURE;
}
```
OpenCL/OpenGL Sharing APIs

Creating OpenCL memory objects from OpenGL objects using clCreateFromGLBuffer, clCreateFromGLTexture2D, clCreateFromGLTexture3D, and clCreateFromGLRenderbuffer ensures that the storage of the OpenGL object will not be deleted while the corresponding OpenCL memory object exists.

CL Buffer Objects > GL Buffer Objects [9.8.2]
cl_mem clCreateFromGLBuffer (cl_context context, cl_mem_flags flags, GLuint bufobj, int *errcode_ret)
flags: CL_MEM_READ_WRITE, CL_MEM_WRITEOnly

CL Image Objects > GL Textures [9.8.3]
cl_mem clCreateFromGLTexture2D (cl_context context, cl_mem_flags flags, GLenum texture_target, GLint mipmaplevel, cl_int *errcode_ret)
flags: See clCreateFromGLBuffer
texture_target: GL_TEXTURE_2D, GL_TEXTURE_CUBE_MAP, GL_TEXTURE_CUBE_MAP_POSITIVE_X, GL_TEXTURE_CUBE_MAP_NEGATIVE_X

CL Image Objects > GL Renderbuffers [9.8.4]
cl_mem clCreateFromGLRenderbuffer (cl_context context, cl_mem_flags flags, GLuint renderbuffer, cl_int *errcode_ret)
flags: clCreateFromGLBuffer

Query Information [9.8.5]
cl_int clGetGLObjectInfo (cl_mem memobj, cl_gl_object_type *gl_object_type, GLuint *gl_object_name, *gl_object_type_returns: GL_OBJECT_BUFFER, GL_OBJECT_RENDERBUFFER, GL_OBJECT_TEXTURE_2D, GL_OBJECT_TEXTURE_3D)

cl_int clGetGLTextureInfo (cl_mem memobj, cl_gl_texture_info param_name, size_t *param_value_size, void *param_value, size_t *param_value_size_ret)
param_name: CL_GL_TEXTURE_TARGET, CL_GL_TEXTURE_LEVEL, CL_GL_TEXTURE_MAXLEVEL

Share Objects [9.8.6]
cl_int clEnqueueAcquireGLObjects (cl_command_queue command_queue, cl_uint num_objects, const cl_mem *mem_objects, cl_uint num_events_in_wait_list, cl_event *event_wait_list, cl_event *event)

CL Event Objects > GL Sync Objects [9.9]
cl_event clCreateEventFromGLSyncKHR (cl_context context, GLsync sync, cl_int *errcode_ret)

CL Context > GL Context, Sharegroup [9.7]
cl_int clGetGLContextInfoKHR (const cl_context_properties *properties, size_t param_value_size, void *param_value, size_t *param_value_size_ret)
Questions?

Derek Gerstmann
University of Western Australia
http://local.wasp.uwa.edu.au/~derek