SIGGRAPHASIA2009

革新の波動

the pulse of innovation

OpenCL/GL interop

Timo Stich, NVIDIA

Outline

- Introduction to CL/GL interop
- Setting up the CL context for GL interop
- API overview
- Examples
    - Mesh Animation
    - Post Processing
- Summary

OpenCL kernel vs GLSL shader

- OpenCL has functionality not available in GLSL shaders
  - Scattered writes
  - Local memory
  - Thread synchronization
  - Atomic memory operations

A new level of GPU programmability!

OpenCL/GL interop

- OpenGL can share data with OpenCL
  - Buffer (Vertex/Pixelbuffer)
  - Texture
  - Renderbuffer
- Mapping
  - OpenCL image -> OpenGL texture, renderbuffer
  - OpenCL buffer -> OpenGL buffer

OpenCL/GL interop

- OpenCL extensions
  - **clGetDeviceInfo**(dev, CL_DEVICE_EXTENSIONS,…);
  - cl_khr_gl_sharing (Windows, Linux, other)
  - cl_apple_gl_sharing (MacOS X)
- OpenCL context must be created from
  - OpenGL context (Windows, Linux, other)
  - OpenGL share group (MacOS X)

OpenCL/GL interop

- Query CL devices that can be associated with a GL context
    - **clGetGLContextInfoKHR**

```
cl_context_properties props[] =
{
    CL_GL_CONTEXT_KHR,
    (cl_context_properties) wglGetCurrentContext()
};
cl_device_id cdDeviceID[N]; size_t size;

clGetGLContextInfoKHR(props, CL_DEVICES_FOR_
GL_CONTEXT_KHR, N*sizeof(cl_device_id), cdDeviceID, &size);

// returns the k = size / sizeof(cl_device_id) devices that support interop
```

Setting up OpenCL/GL interop

- Windows WGL

```
cl_context_properties props[] =
{
    CL_GL_CONTEXT_KHR,
    (cl_context_properties) wglGetCurrentContext(),    // HGLRC handle
    CL_WGL_HDC_KHR,
    (cl_context_properties) wglGetCurrentDC(),         // HDC handle
    CL_CONTEXT_PLATFORM,
    (cl_context_properties)cpPlatform, 0
};
cxGPUContext = clCreateContext(props, 1, cdDeviceID, NULL, NULL,
&ciErrNum);
```

Setting up OpenCL/GL interop

- MacOS X

```
CGLContextObj kCGLContext = CGLGetCurrentContext(); // GL Context
CGLShareGroupObj kCGLShareGroup =
CGLGetShareGroup(kCGLContext);                          // Share Group
cl_context_properties props[] =
{
    CL_CONTEXT_PROPERTY_USE_CGL_SHAREGROUP_APPLE,
    (cl_context_properties) kCGLShareGroup,
    CL_CONTEXT_PLATFORM,
    (cl_context_properties) cpPlatform, 0
};
cxGPUContext = clCreateContext(props, 1, cdDeviceID, NULL, NULL,
&ciErrNum);
```

Sharing Data

- OpenCL memory objects are created <u>from</u> OpenGL objects
  - Become invalid when GL object changes
  - Still valid when GL object is deleted

- Must be acquired/released before/after use
  - Need to sync APIs

- Best Practice:
  - Release CL resource <u>before</u> GL resource

Sharing API

- Creating CL objects from GL objects
  - **clCreateFromGLBuffer**
  - **clCreateFromGLTexture2D**
  - **clCreateFromGLTexture3D**
  - **clCreateFromGLRenderbuffer**

Example

- Sharing GL vertex buffer

```
GLuint vbo;
cl_mem vbo_cl;
// create buffer object
glGenBuffers(1, &vbo);
glBindBuffer(GL_ARRAY_BUFFER, vbo);

// initialize buffer object
unsigned int size = mesh_width * mesh_height * 4 * sizeof(float);
glBufferData(GL_ARRAY_BUFFER, size, 0, GL_DYNAMIC_DRAW);

// create OpenCL buffer from GL VBO
vbo_cl = clCreateFromGLBuffer(cxGPUContext,CL_MEM_WRITE_ONLY,
                    vbo, NULL);
```

Sharing API

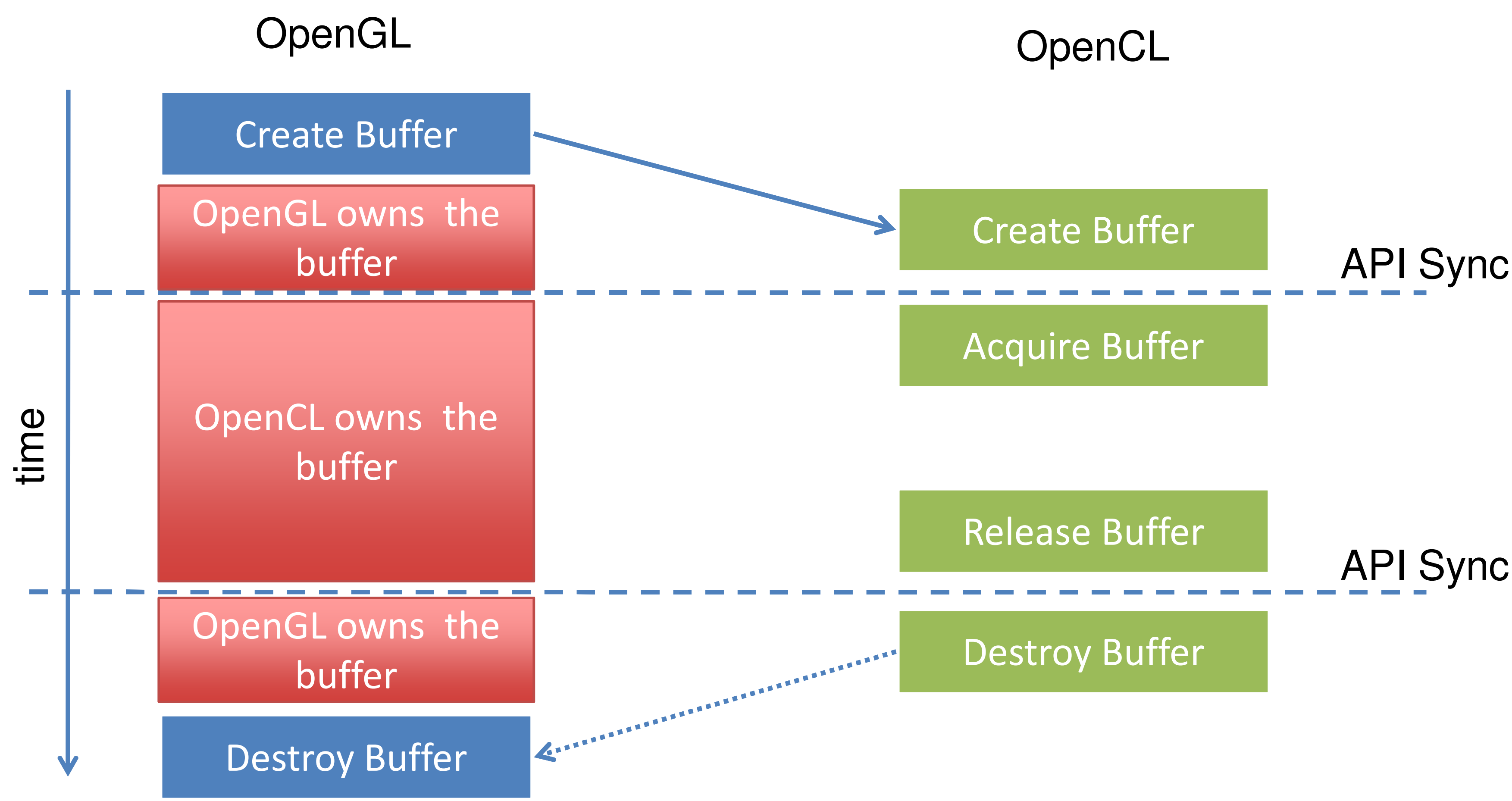
- Locking objects for use with OpenCL
  - **clEnqueueAcquireGLObjects**
  - **clEnqueueReleaseGLObjects**
- Additionally the APIs need to be synchronized
  - **clFinish, clWaitForEvents**
  - **glFinish, glFlush**

Example

- Acquire and Release

```
glFinish();
// All pending GL calls have finished -> safe to acquire the buffer in CL
clEnqueueAcquireGLObjects(cqCommandQueue, 1, vbo_cl, 0,0,0);

<... OpenCL manipulates the buffer ...>

clEnqueueReleaseGLObjects(cqCommandQueue, 1, vbo_cl, 0,0,0);
clFinish(cqCommandQueue);
// All pending CL calls have finished -> safe to make use of buffer in GL
```

Sharing Data, Summary



Multi device OpenCL/GL interop

- Typically only one device will drive the GL context
  - But multiple CL devices can be associated

- Query device associated with a GL context
  - **clGetGLContextInfoKHR**

- Acquire/Release can be posted on <u>any</u> command queue
  - CQ of device driving the GL context will be the fast path
  - All other might trigger implicit copy through host

Multi device OpenCL/GL interop

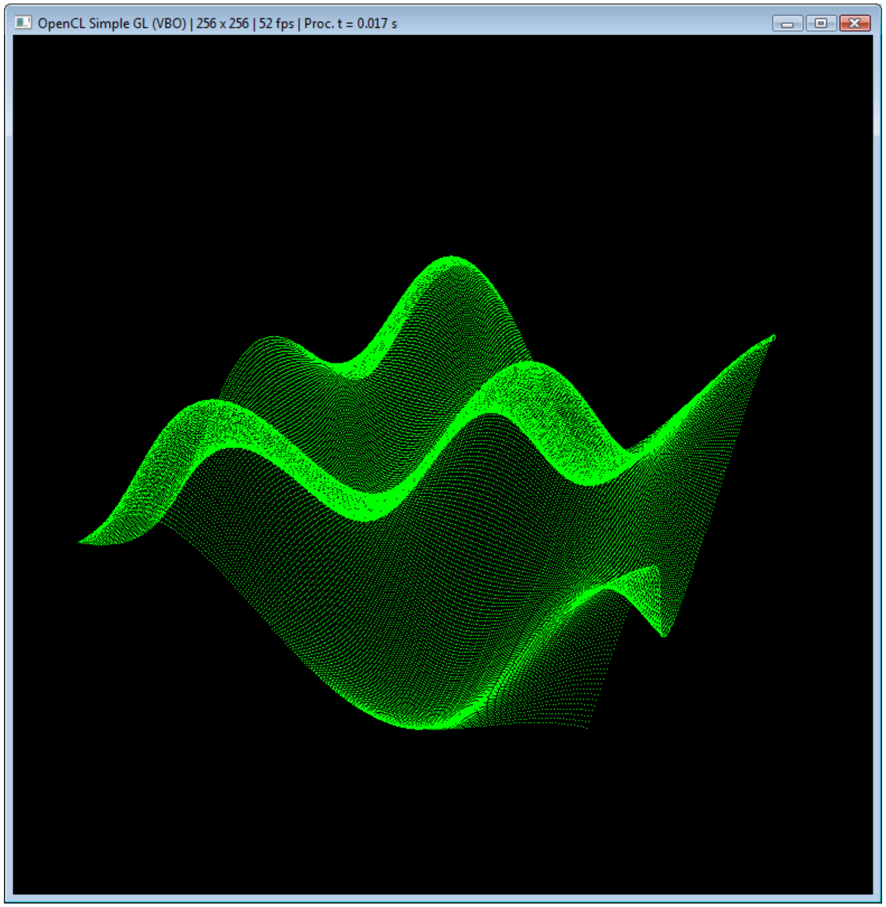
- Example

```
cl_context_properties props[] =
{
    CL_GL_CONTEXT_KHR,
    (cl_context_properties) wglGetCurrentContext()
};
cl_device_id clGLdevice;

clGetGLContextInfoKHR(props, CL_CURRENT_DEVICE_FOR_
GL_CONTEXT_KHR, sizeof(cl_device_id), &clGLdevice, 0);

cl_command_queue cqFastGLinteropQueue =
clCreateCommandQueue(cxGPUContext, clGLdevice, 0,0);
```

OpenCL Mesh Animation Example

- Animate mesh with sine pattern
  - Coordinates computed with OpenCL kernel
- Render as point cloud with OpenGL
  - OpenCL kernel writes to shared Vertex Buffer Object

OpenCL Mesh Animation



OpenCL Mesh Animation

- OpenCL C kernel

```
__kernel void sine_wave(__global float4* pos, uint width, uint height, float time) {
    uint x = get_global_id(0); uint y = get_global_id(1);

    // calculate uv coordinates
    float u = x / (float) width;
    float v = y / (float) height;
    u = u*2.0f - 1.0f;
    v = v*2.0f - 1.0f;

    // calculate simple sine wave pattern
    float freq = 4.0f;
    float w = sin(u*freq + time) * cos(v*freq + time) * 0.5f;

    // write output vertex
    pos[y*width+x] = (float4)(u, w, v, 1.0f); }
```

OpenCL Mesh Animation

- GL/CL interop

```
// Acquire OpenGL buffer object for writing from OpenCL
glFinish();
clEnqueueAcquireGLObjects(cqCommandQueue, 1, &vbo_cl, 0,0,0);

// Set work size and execute the kernel
szGlobalWorkSize[0] = mesh_width; szGlobalWorkSize[1] = mesh_height;
szLocalWorkSize[0] = 16; szLocalWorkSize[1] = 16;
clSetKernelArg(ckKernel, 3, sizeof(float), &anim);   // Update animation time
clEnqueueNDRangeKernel(cqCommandQueue, ckKernel, 2, NULL, szGlobalWorkSize,
szLocalWorkSize, 0,0,0 );

// Release buffer object from OpenCL
clEnqueueReleaseGLObjects(cqCommandQueue, 1, &vbo_cl, 0,0,0);
clFinish(cqCommandQueue);
```

OpenCL Mesh Animation

- Rendering Loop

```
// run OpenCL kernel to generate vertex positions
runKernel(animation_time);

// clear graphics then render from the vbo
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glBindBuffer(GL_ARRAY_BUFFER, vbo);
glVertexPointer(4, GL_FLOAT, 0, 0);
glEnableClientState(GL_VERTEX_ARRAY);
glColor3f(0.0, 1.0, 0.0);
glDrawArrays(GL_POINTS, 0, mesh_width * mesh_height);
glDisableClientState(GL_VERTEX_ARRAY);

// flip backbuffer to screen
glutSwapBuffers();
glutPostRedisplay();
```

OpenCL Mesh Animation

- Demo

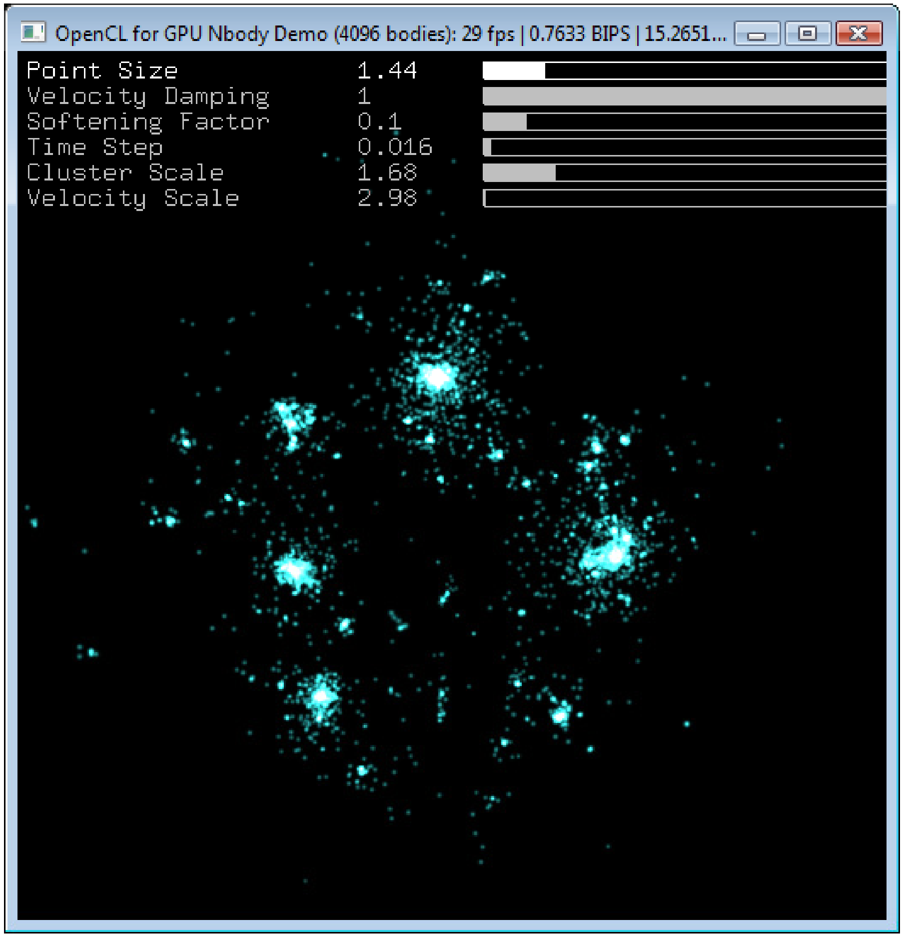
oclSimpleGL.exe

OpenCL N-Body

- Simulating a gravity system
    - Kernel makes use of local memory

- Update system with OpenCL
    - Render with OpenGL

OpenCL N-Body



OpenCL Postprocessing Example

- Postprocessing of OpenGL rendered scene
  - 2D box filter
  - Boost highlights

- Render scene to FrameBufferObject
  - RenderBuffer for Color and Depth

- OpenCL Kernel writes to OpenGL Texture
  - OpenGL renders textured Screen-Quad

OpenCL images

- Optional: Not supported by all OpenCL devices
    - Check with CL_DEVICE_IMAGE_SUPPORT
- Similar to OpenGL textures
- Readable OR Writeable
- Read via Sampler
    - Interpolation (Nearest, Bilinear)
    - Normalized/Non-normalized coordinates
    - Border handling (Clamp, Repeat)

OpenCL images

- Optional: Not supported by all OpenCL devices
    - Check with CL_DEVICE_IMAGE_SUPPORT
- Similar to OpenGL textures
- Readable OR Writeable
- Read via Sampler
    - Interpolation (Nearest, Bilinear)
    - Normalized/Non-normalized coordinates
    - Border handling (Clamp, Repeat)

OpenCL sampler

- Sampler can be created on the host and passed in as kernel arguments
  - **clCreateSampler**
  - **clGetSamplerInfo**
- Samplers can also be defined as const in kernel code

```
const sampler_t constSampler = CLK_NORMALIZED_COORDS_FALSE |
CLK_ADDRESS_CLAMP | CLK_FILTER_NEAREST;
}
```

OpenCL C image functions

- Images are passed as kernel arguments
  - either as __read_only OR __write_only

```
__kernel void someKernel(__read_only image2d_t inputImage)
```

- Functions for accessing images
  - read_imagei/ui/f(image, sampler, coord);
  - write_image(image, coord, value);

OpenCL images

- Example

```
const sampler_t sampler = CLK_NORMALIZED_COORDS_TRUE |
CLK_ADDRESS_CLAMP | CLK_FILTER_BILINEAR;

__kernel void imageKernel(__read_only image2d_t inputImage,
                          __write_only image2d_t outputImage, int width, int height)
{
    int x = get_global_id(0); int y = get_global_id(1);

    float2 normalizedCoord = (float2)((x + 0.5f)/width, (y+0.5f)/height);
    uint4 value = read_imageui(inputImage, sampler, normalizedCoord);

    int2 unnormalizedCoord = (int2)(x,y);
    write_imageui(outputImage, unnormalizedCoord, value);
}
```

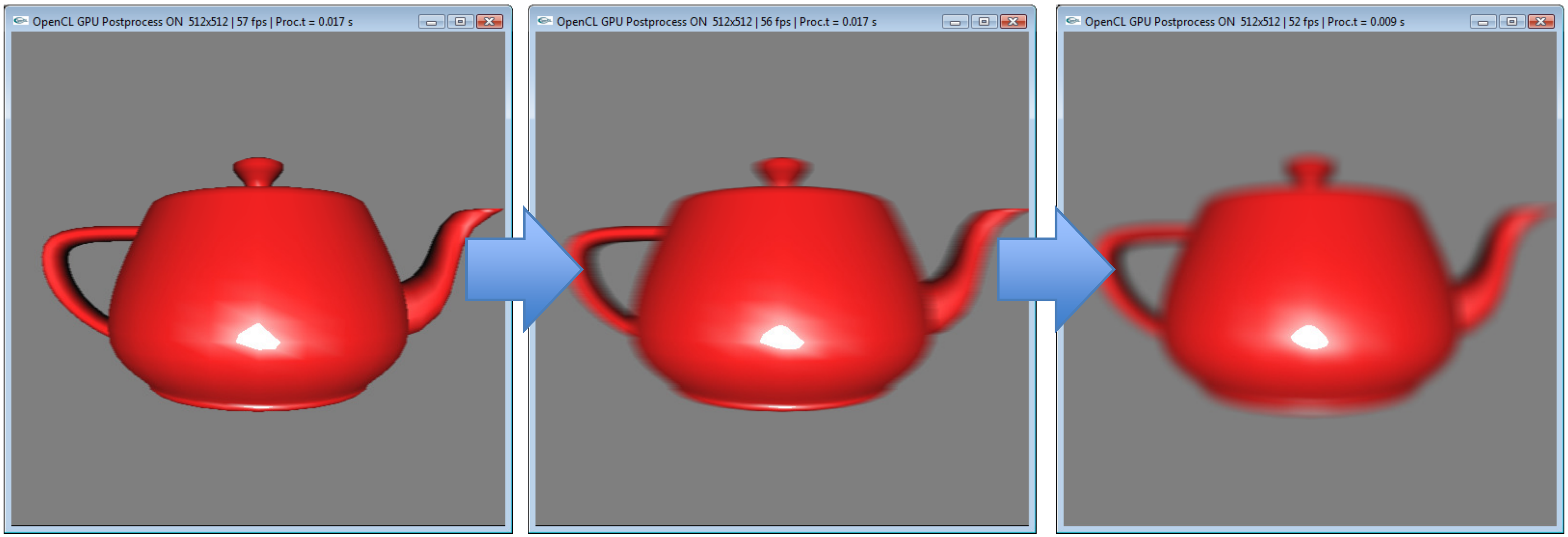
OpenCL Postprocessing Example

- Postprocessing of OpenGL rendered scene
  - 2D box filter, implemented as 2-pass separable filter
  - Boost highlights in final pass

- Render scene to FrameBufferObject
  - RenderBuffer for Color and Depth

- OpenCL Kernel writes to OpenGL Texture
  - OpenGL renders textured Screen-Quad

OpenCL Postprocessing



Rendered Scene OpenGL	Rows filtered OpenCL	Columns filtered OpenCL

OpenCL Postprocessing

- FBO Rendertarget

```
// Create and bind the FBO
glGenFramebuffersEXT(1, &fbo);
glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, fbo);

// Create a RGBA8 render buffer
glGenRenderbuffersEXT(1, &rb_color);
glBindRenderbufferEXT(GL_RENDERBUFFER_EXT, rb_color);
glRenderbufferStorageEXT(GL_RENDERBUFFER_EXT, GL_RGBA8, width, height);

// Attach it as color attachment to the FBO
glFramebufferRenderbufferEXT(GL_FRAMEBUFFER_EXT,
GL_COLOR_ATTACHMENT0_EXT, GL_RENDERBUFFER_EXT, rb_color);

// Do the same for the depth attachment
// ...
```

OpenCL Postprocessing

- CL Image from FBO color attachment

```
// Create the CL image from the color renderbuffer – will read from this in the kernel
cl_mem cl_scene;
cl_scene = clCreateFromGLRenderbuffer(cxGPUContext, CL_MEM_READ_ONLY,
rb_color, 0);

// CL can query properties on this image as with normal CL images
cl_image_format image_format;
clGetImageInfo (cl_texture, CL_IMAGE_FORMAT, sizeof(cl_image_format),
&image_format, NULL);

// image_format will be CL_UNSIGNED_INT8, CL_BGRA
```

OpenCL Postprocessing

- GL Texture for final render pass

```
// Create GL texture
glGenTextures(1, &tex_screen); glBindTexture(GL_TEXTURE_2D, tex_screen);

// Set texture parameters
<...>

// Setup data storage
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA8, size_x, size_y, 0, GL_RGBA,
GL_UNSIGNED_BYTE, NULL);

// Create CL image from Screen Texture – the CL kernel will write to this
cl_mem cl_screen;
cl_screen = clCreateFromGLTexture2D(cxGPUContext, CL_MEM_WRITE_ONLY,
GL_TEXTURE_2D, 0, tex_screen, 0);
```

OpenCL Postprocessing

Rendered Scene
OpenGL

Rows filtered
OpenCL

Columns filtered
OpenCL

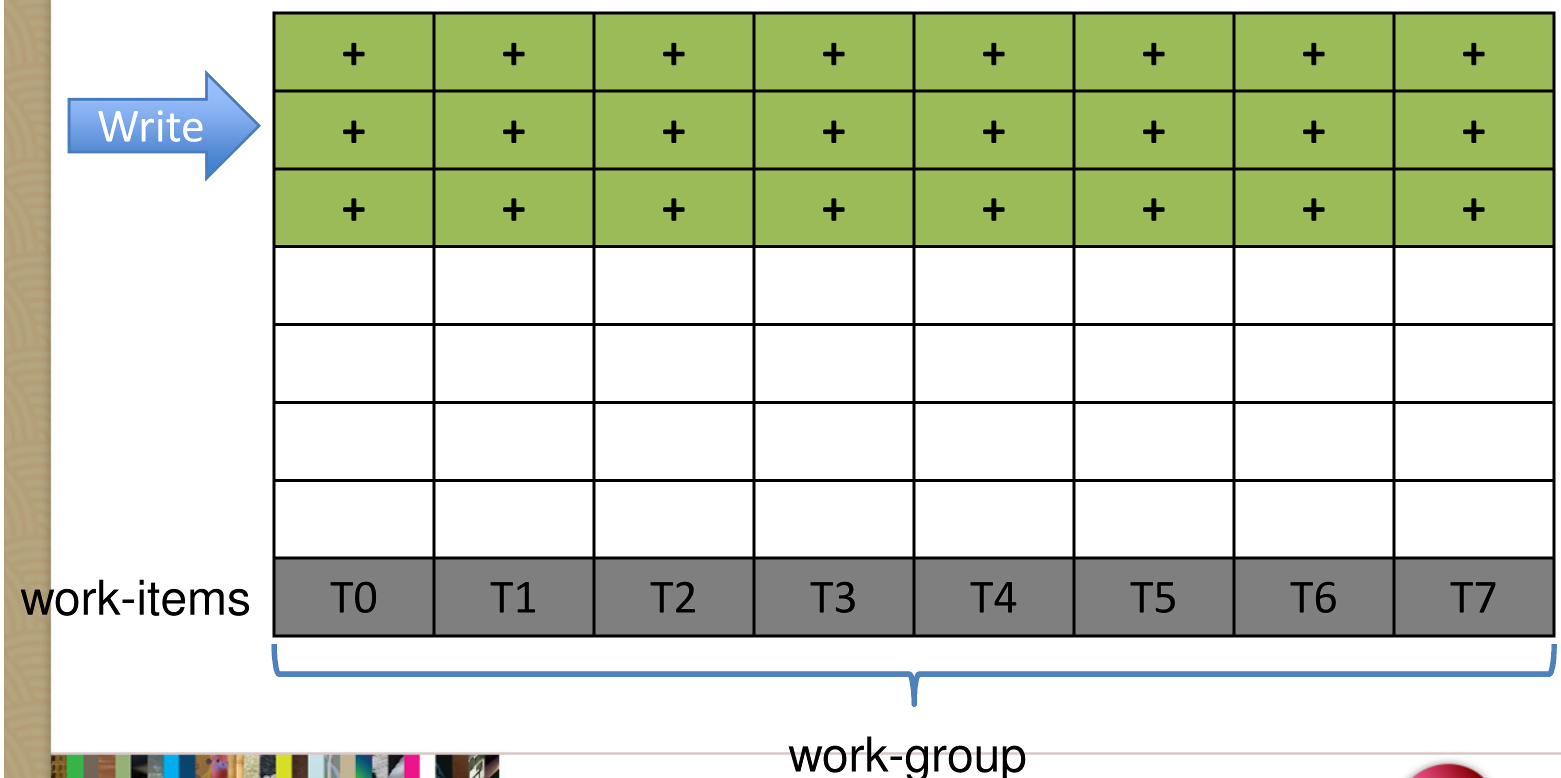
OpenCL Postprocessing

- Shader-like implementation
  - One work-item per output pixel
  - Each thread loops over the radius R of input pixels
  - For N pixels: N **x** (2 x R+1) ops

- OpenCL introduces scattered writes!
  - One work-item per N output pixels
  - Each thread can reuse result from last pixel
  - For N pixels: N **+** 2 x R ops

OpenCL Postprocessing

- Initialization

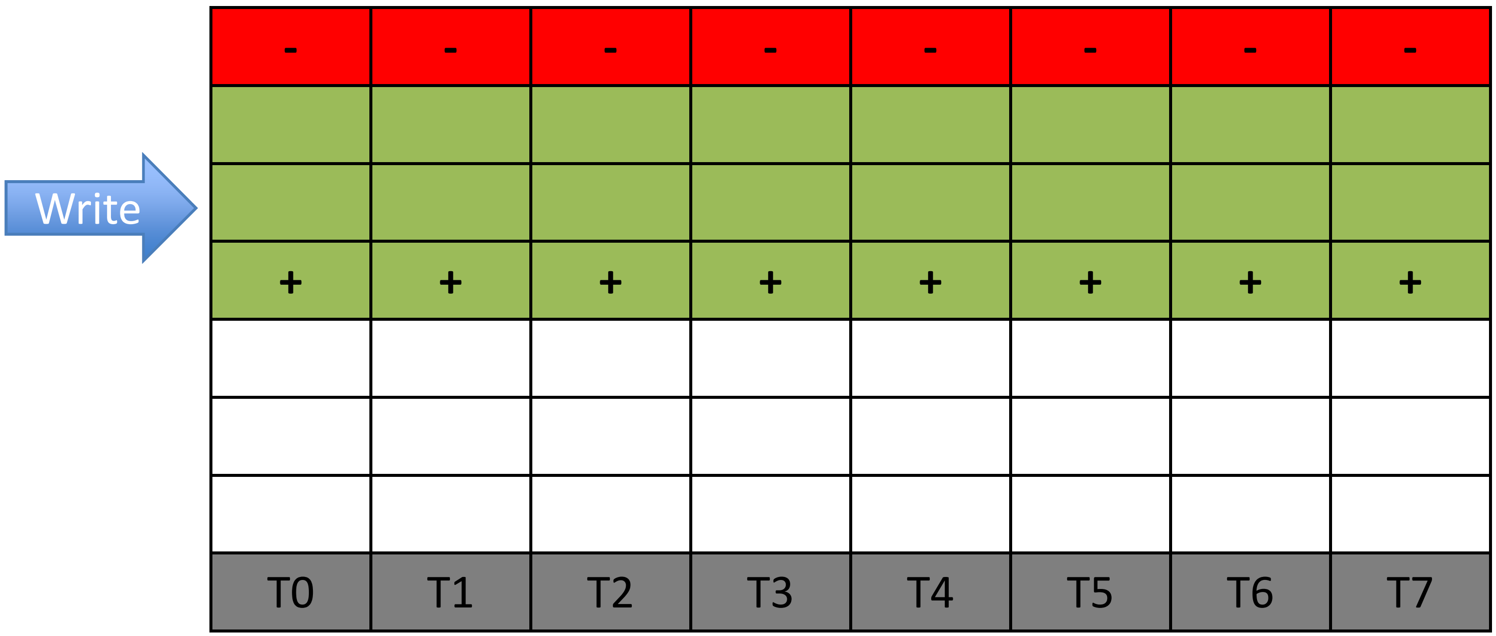


OpenCL Postprocessing

- OpenCL Kernel for filtering columns

```
int x = get_global_id(0);
int y = TILE_Y * get_group_id(1);     // Global ID != Y coord

foat4 color = (float4)(.0f,.0f,.0f,.0f);

// Initialize the sum
for( int i=-radius; i<=radius; ++i ) {
        if( y+i > 0 && y+i < imgh ) {
                uchar4 c = g_data[(y+i)*imgw+x];
                color.x += c.x; color.y += c.y; color.z += c.z;
                color.w += 1.0f;
        }
}
write_imageui( g_odata, (int2)(x,y),
                        (uint4)(color.z/color.w , color.y/color.w, color.x/color.w, 255));
```

OpenCL Postprocessing

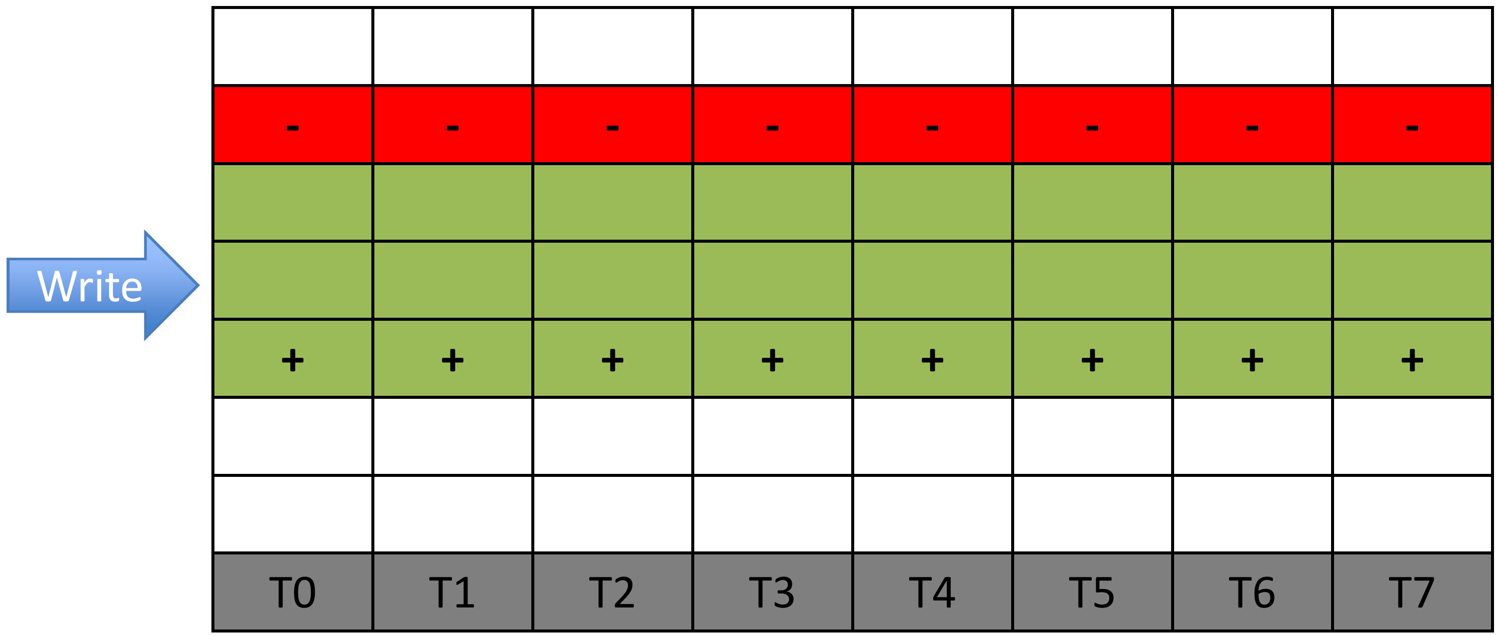


OpenCL Postprocessing

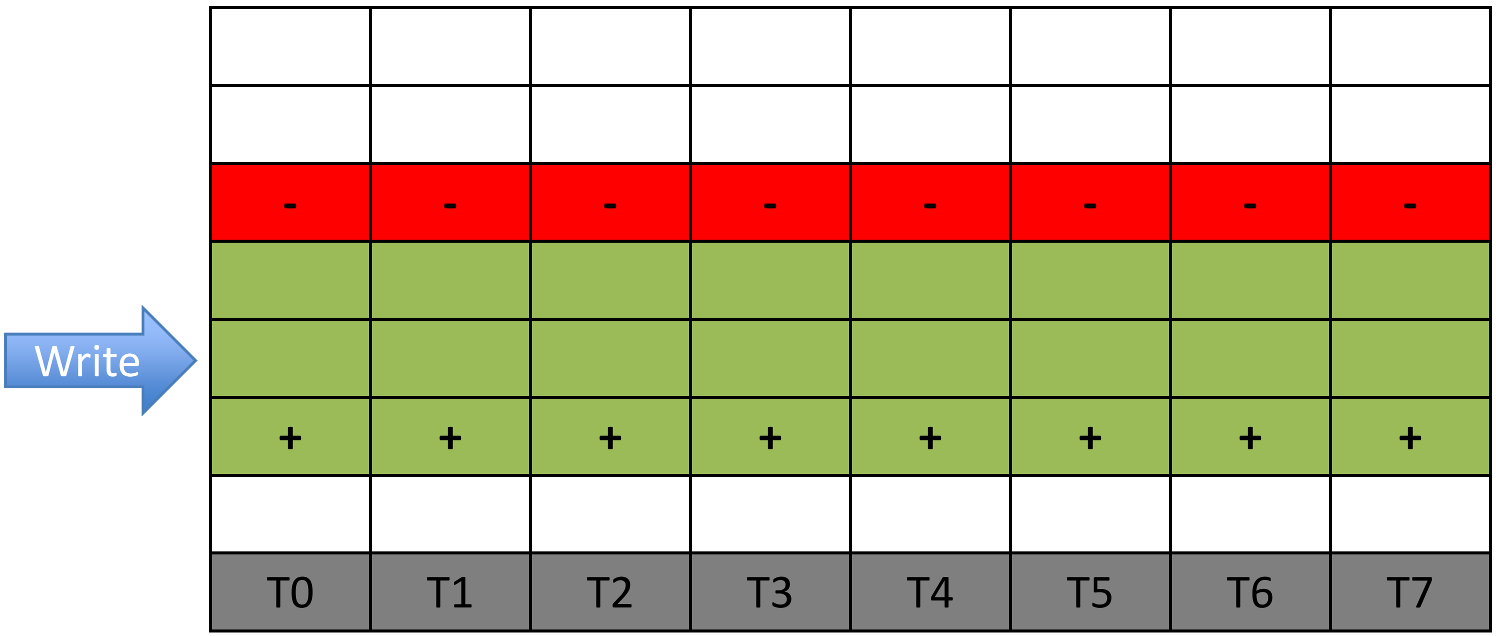
- OpenCL kernel: Loop over tile

```
for( int i=0; i<TILE_Y; ++i, ++y ) {
        // Update sum
        if( y-radius > 0 ) {
                uchar4 c = g_data[(y-radius)*imgw+x];
                color.x -= c.x; color.y -= c.y; color.z -= c.z;
                color.w -= 1.0f;
        }
        if( y+radius+1 < imgh ) {
                uchar4 c = g_data[(y+radius+1)*imgw+x];
                color.x += c.x; color.y += c.y; color.z += c.z;
                color.w += 1.0f;
        }
        // Scattered write to image
        write_imageui(g_odata, (int2)(x,y), (uint4)(color.z/color.w , color.y/color.w,
                                color.x/color.w, 255)); }
```

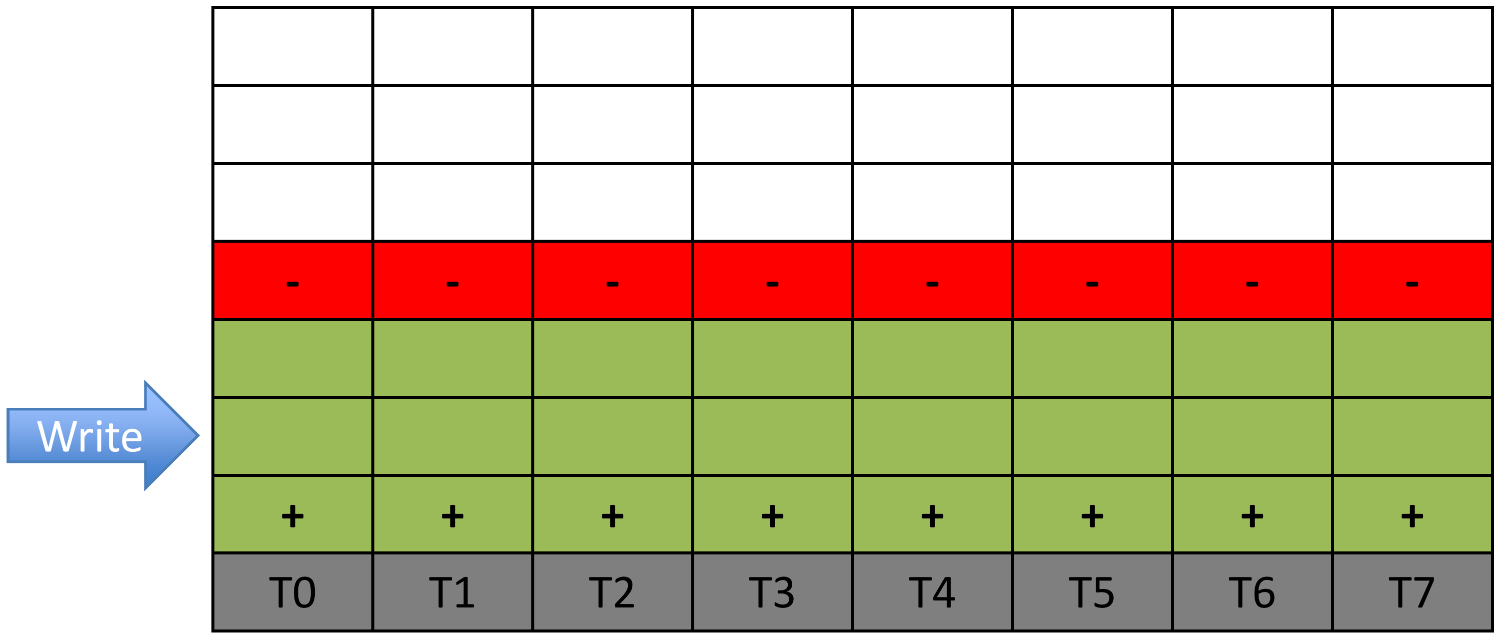
OpenCL Postprocessing



OpenCL Postprocessing

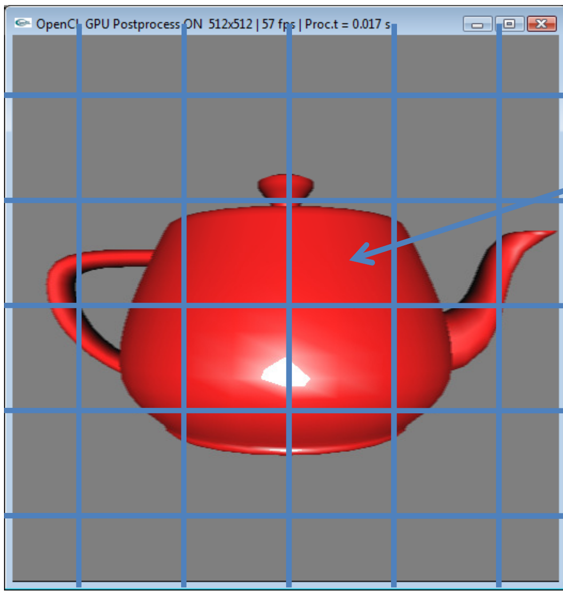


OpenCL Postprocessing

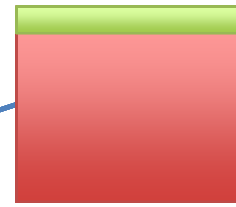


OpenCL Postprocessing

Each work-group consists of 64 work-items (threads)



Renderbuffer is split into tiles of 64x64 pixels



Each Tile is processed by one work-group

GlobalWorkSize = [Width, Height/64]
LocalWorkSize = [64,1]

Each work-item processes 64 pixels

OpenCL Postprocessing

Renderbuffer & Texture

- Host side

```
// Acquire the Renderbuffer and the output Texture
glFinish();
clEnqueueAcquireGLObjects(cqCommandQueue,2, &cl_glObjects, 0, NULL, NULL);

// Row Filtering Pass
clEnqueueNDRangeKernel(cqCommandQueue, ckFilterRows, 2, NULL,
                       szGlobalWorkSize, szLocalWorkSize,
                       0, NULL, NULL);
// Column Filtering Pass
clEnqueueNDRangeKernel(cqCommandQueue, ckKernel, 2, NULL,
                       szGlobalWorkSize, szLocalWorkSize,
                       0, NULL, NULL);
// Release the GL objects
clEnqueueReleaseGLObjects(cqCommandQueue,2, &cl_glObjects, 0, NULL, NULL);
clFinish(cqCommandQueue);
```

OpenCL Postprocessing

- Rendering Loop

```
// Render the 3D scene OpenGL to the FBO
renderScene();

// Postprocess with OpenCL
postprocess();

// Render the Texture on a full screen quad with GL to the backbuffer
drawTexturedFullScreenQuad(tex_screen);

// flip backbuffer to screen
glutSwapBuffers();
glutPostRedisplay();
```

OpenCL Postprocessing

- Demo

oclPostprocessGL.exe

Summary

- OpenCL and OpenGL can share data efficiently
  - OpenCL objects are created from OpenGL objects
  - Acquire/Release mechanism

- OpenCL vs GLSL shaders
  - Scattered writes, Local memory, Thread sync, Atomics,...

- Typical use cases:
  - Animation, Postprocessing, Physical Simulation, ...