Harnessing OpenCL in Your Application

OpenCL overview

Ian Ollmann, Ph.D.
Senior Scientist

Introduction

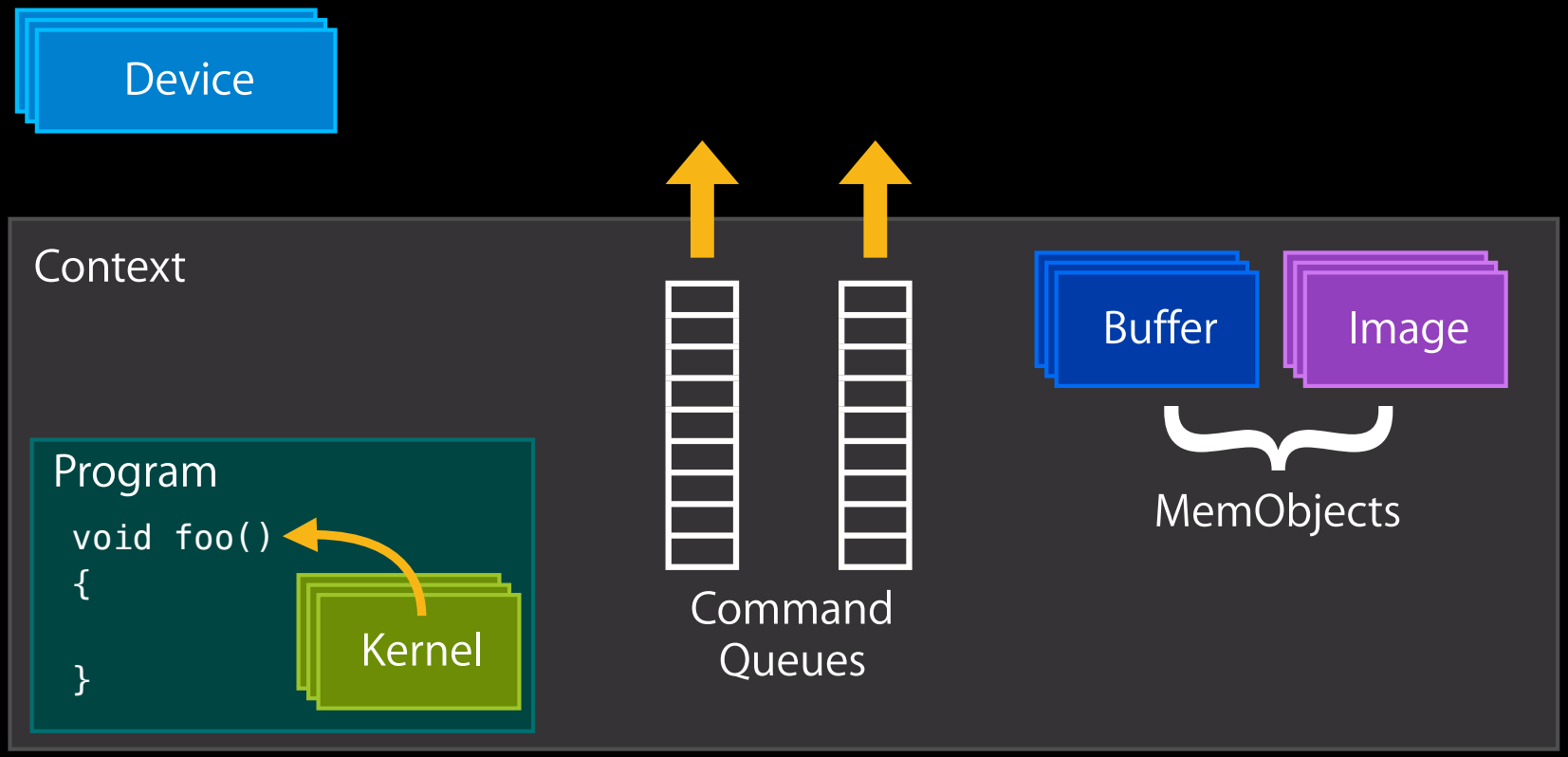
- Discuss OpenCL design and philosophy
- Explore common developer questions
- Debugging tips

- Assumes some experience with OpenCL

OpenCL Design Goals

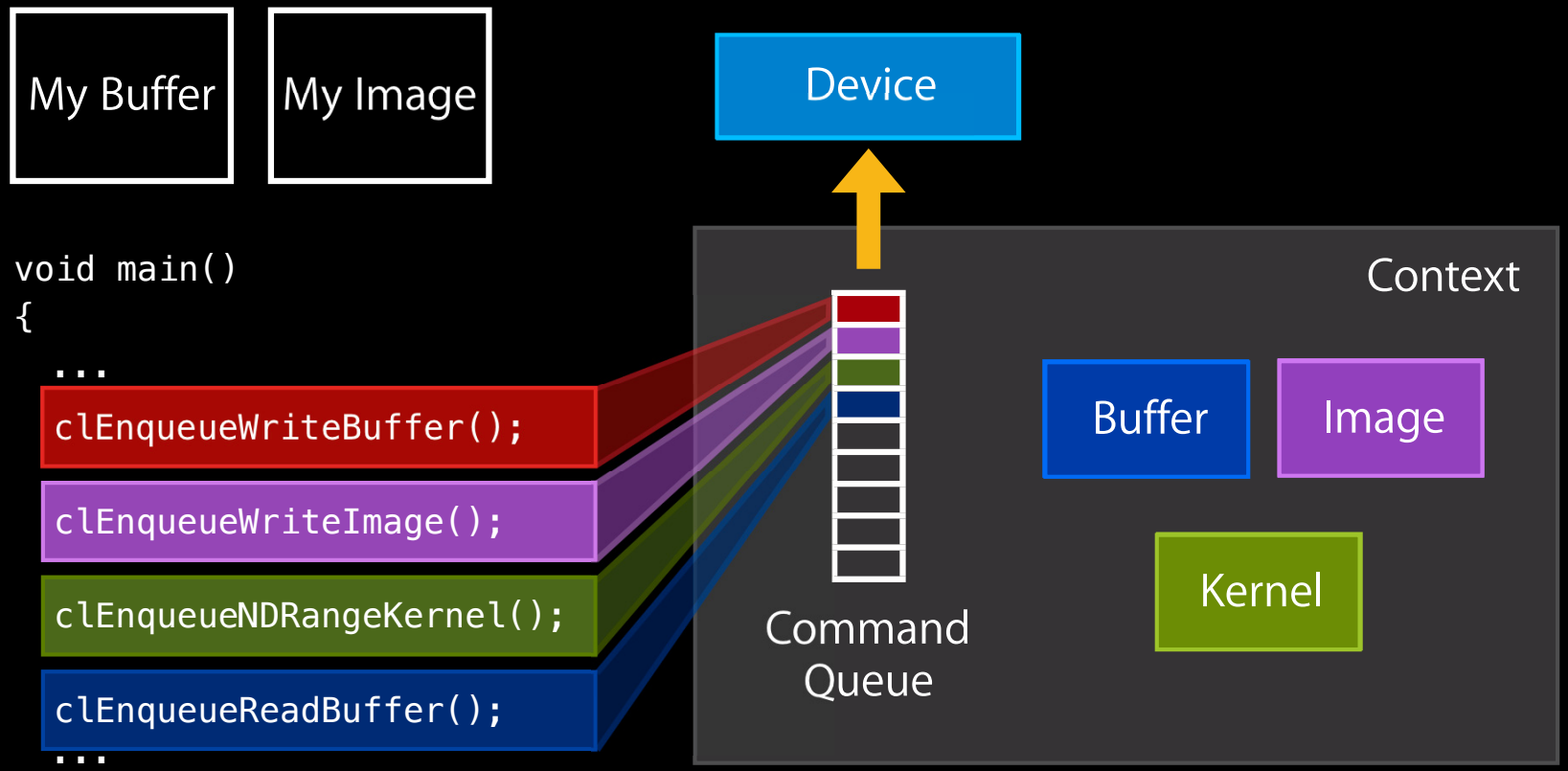
- Bring programmability to all the devices on your system
  - CPU
  - GPU
  - Accelerators
- Multiple devices may mean:
  - Separate address spaces per device
  - Multiple instruction sets

OpenCL Object Design



Keep Command Queues full!

OpenCL Objects in Action




```
void main()
{
  ...
  clEnqueueWriteBuffer();
  clEnqueueWriteImage();
  clEnqueueNDRangeKernel();
  clEnqueueReadBuffer();
  ...
}
```

Consistent Object Interfaces

Object Type
Device
Context
Program
Kernel
Buffer/Image
Sampler
CommandQueue

- Allocators

```
cl_type clCreateObjectType(..., cl_int *err );
```

- Reference counting

```
cl_int clRetainObjectType( cl_type );
cl_int clReleaseObjectType( cl_type );
```

- Getters / setters

```
cl_int clGetObjectTypeInfo( cl_type, selector, size_t, void*, size_t* );
cl_int clSetObjectTypeInfo( cl_type, selector, size_t, void*, size_t* );
```

- Enqueue commands

```
cl_int clEnqueueCommandType( cl_command_queue, ... );
```

Programming Model

How to write code for 100's or 1000's of cores?

- Task level parallelism—traditional
  - Divide work into different tasks
    - Get mail from server
    - Identify junk mail
    - Run a mail filter or two
    - Draw UI
    - Get Input
    - Audio
  - Maybe 5 or 10 different tasks → 5 or 10-way parallelism

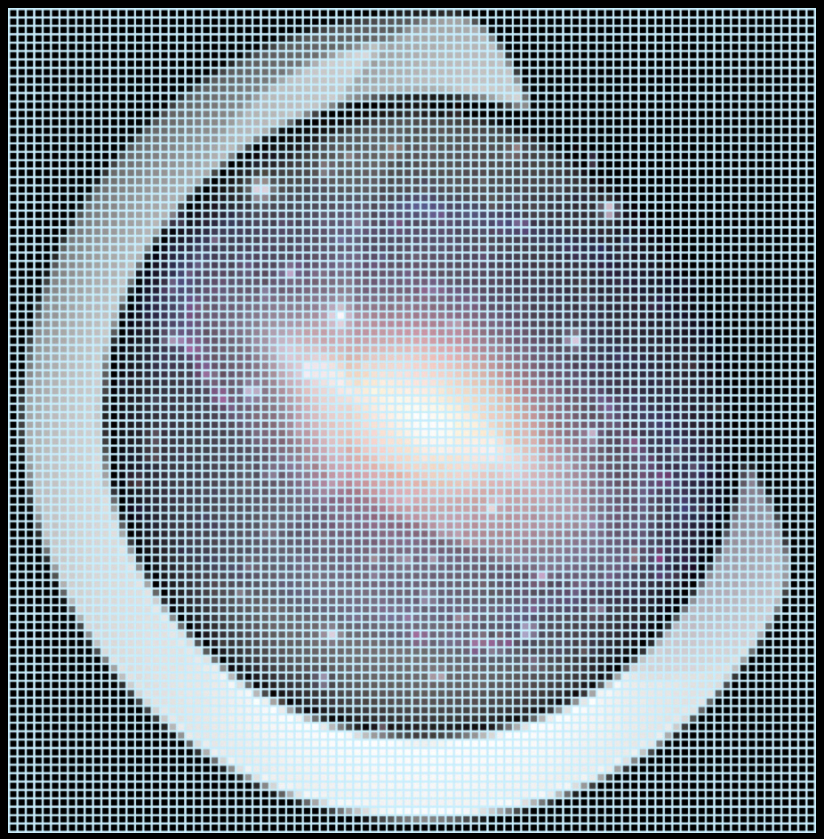
Programming Model

How to write code for 100's or 1000's of cores?

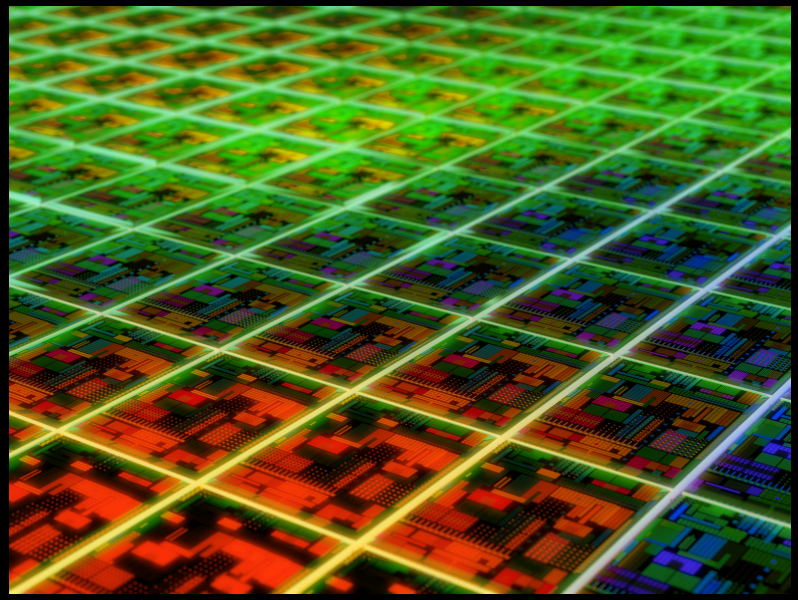
- Data level parallelism—learn from shaders!
  - Parallelize along data boundaries
    - e.g. a different thread for each e-mail
    - Hopefully largely independent computations
  - 1000's of items to do work on → 1000-way parallelism
    - …or better yet millions → million way parallelism!

This is what OpenCL is designed to do

Programming Model

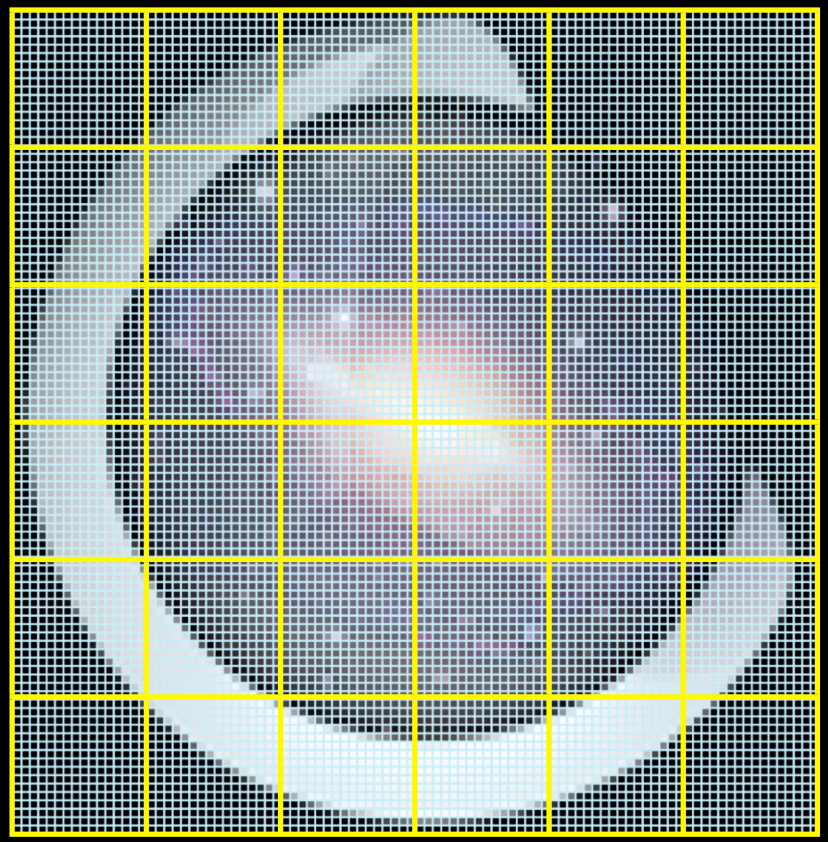


- Break data into a grid of chunks
  - Chunk is called “workitem”
- Write a function to process one workitem
  - OpenCL will call each workitem
  - get_global_id() for workitem id
- Workitem grid can also be orthogonal to data set dimensions



Workgroups and Compute Units

- Complication: real devices are not uniform arrays of simple scalar processors
  - Super scalar
  - SMT
  - Vector units!
- Most cores, "Compute Units", can run many workitems concurrently



Programming Model

Aggregate consecutive workitems into "workgroups"

- Workgroup:
  - One or more workitems working together
  - Workitems run together on a single compute unit
    - Can share resources
      - Local caches
      - Coalesced loads / stores
    - Very cheap to synchronize!
- Synchronization between workgroups is HARD

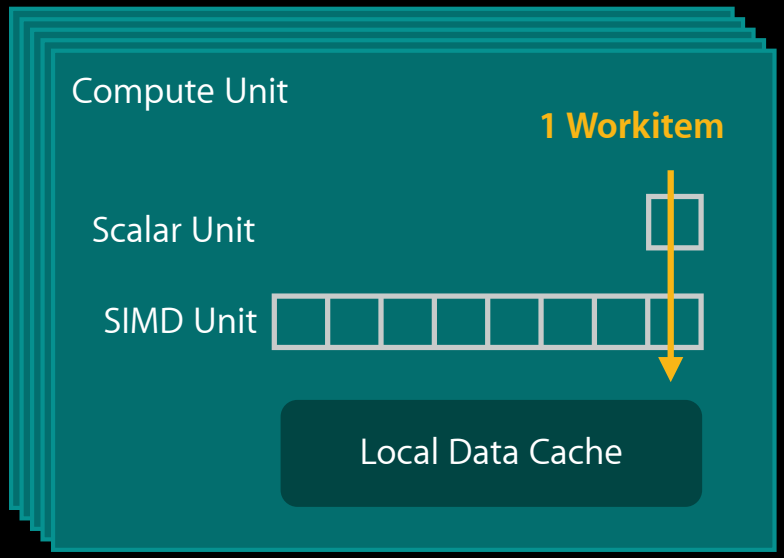
Why Inter-core Synchronization is Hard

- Cores are far apart
  - Speed of light is only so fast!
  - Communication can be slow
- Costly!
  - Would add complexity—cores frequently very simple
    - More complexity ⇒ more area ⇒ fewer cores on die ⇒ less performance!
  - Don't want to get permission to complete each instruction
    - N^2 communications problem?
  - Not enough memory
    - 4 kB stack per workitem* 1 million workitems = 4GB of stack

Direct Model

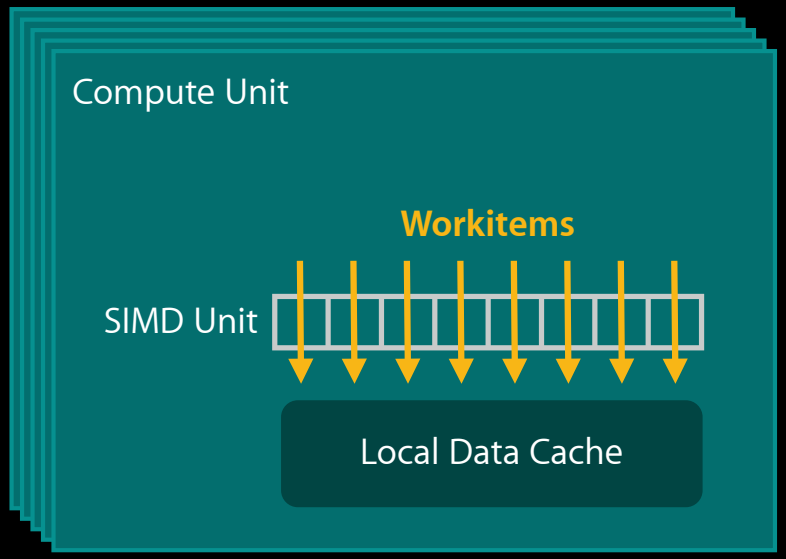
Simple

- One workitem = 1 thread
- Must vectorize your code to get full parallelism within workgroup
- Other sources
  - Superscalarism
  - Reorder buffers



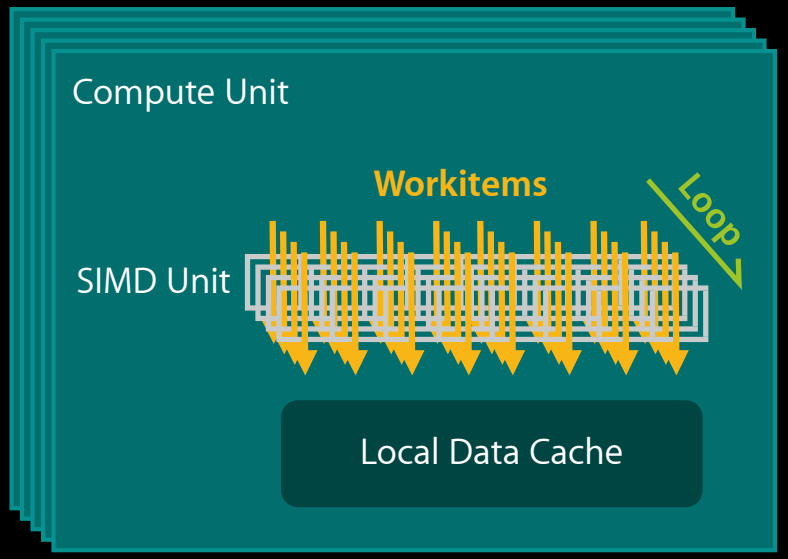
Example

Parallelize through SIMD engine



Example

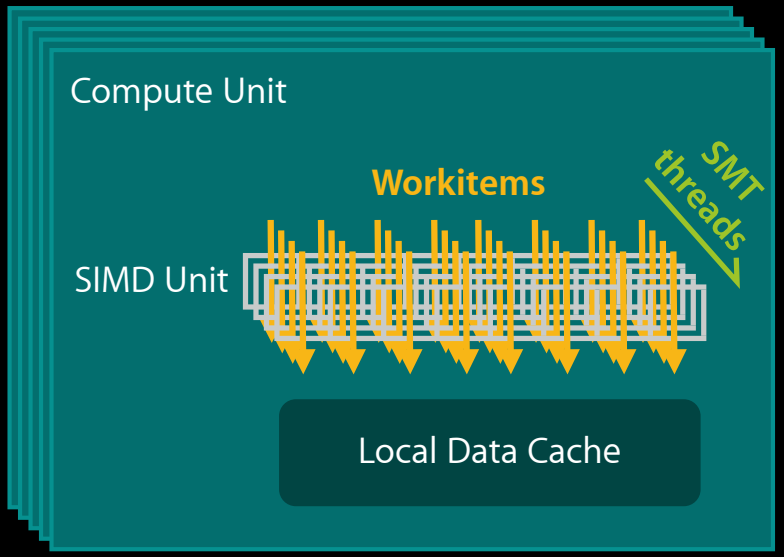
SIMD engine and software loop




```
float8 a0, a1, a2, a3;

a0 += 1.0f;
a1 += 1.0f;
a2 += 1.0f;
a3 += 1.0f;

a0 += b0;
a1 += b1;
a2 += b2;
a3 += b3;
...
```

Example

SIMD engine and SMT



Synchronization

mem_fence() vs. barrier() vs wait_group_events()

- mem_fence
  - Within workitem only
  - Largely unnecessary on MacOS X
- Barrier
  - Within workgroup only
  - Key to successful local memory usage
- wait_group_events
  - Like barrier
  - Used with async_workgroup_copy()

Finding the Right Workgroup Size

size_t[3]	clGetDevinceInfo(CL_DEVICE_MAX_WORK_ITEM_SIZES)
size_t	clGetDevinceInfo(CL_DEVICE_MAX_WORK_GROUP_SIZE)
size_t[3]	clGetKernelWorkGroupInfo(CL_KERNEL_WORK_GROUP_SIZE)
size_t[3]	clGetKernelWorkGroupInfo(CL_KERNEL_COMPILE_WORK_GROUP_SIZE)
size_t[3]	global work size % workgroup size = 0

size_t	clKernelWorkGroupInfo(CL_KERNEL_LOCAL_MEM_SIZE)
size_t	clGetDevinceInfo(CL_DEVICE_LOCAL_MEM_SIZE)
size_t	user allocated local memory per workgroup

Finding the Right Workgroup Size

Solution #1: Give up

- Pass NULL workgroup size
  - Pass the problem to us!
  - Hope for magic
    - Magic is Wonderful
      - Solves all my problems!
      - Does fabulous things!
    - We'll try—Really!
    - Not well grounded in reality
      - You probably have more info than we do

Finding the Right Workgroup Size

Solution #2: Divide and conquer

- Find minimum of 1D limits:

size_t	clGetDevinceInfo(CL_DEVICE_MAX_WORK_GROUP_SIZE)
size_t	clGetDevinceInfo(CL_DEVICE_LOCAL_MEM_SIZE)
size_t	clKernelWorkGroupInfo(CL_KERNEL_LOCAL_MEM_SIZE)
size_t	user allocated local memory per workgroup
size_t	clGetKernelWorkGroupInfo(CL_KERNEL_WORK_GROUP_SIZE)

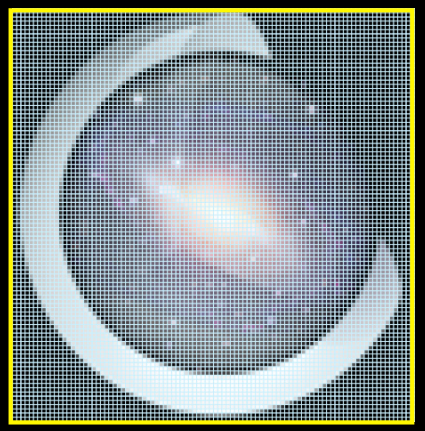
- Then apply minimum 3D limits:

size_t[3]	clGetKernelWorkGroupInfo(CL_KERNEL_COMPILE_WORK_GROUP_SIZE)
size_t[3]	clGetDevinceInfo(CL_DEVICE_MAX_WORK_ITEM_SIZES)
size_t[3]	global work size % workgroup size = 0

Finding the Right Workgroup Size

When global work size is a prime number

- Change the global work dimensions
  - Widen the global work dimension



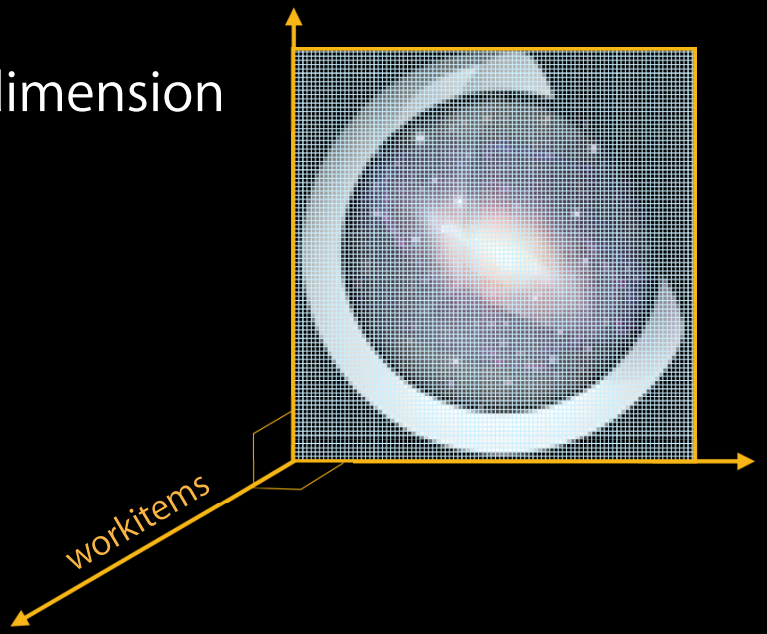
  - Early out for workitems that are outside the original size
    - Kernel:

```
if( get_global_id(0) > width ||
    get_global_id(1) > height )
     return;
```

Finding the Right Workgroup Size

When global work size is a prime number

- Move workgroup size to an abstract dimension



- Map the abstract dimension to problem size in __kernel

```
x_pos = my_fx( get_global_id() );
y_pos = my_fy( get_global_id() );
```

Developer Questions

Common Developer Questions

- Half precision
- Local memory
- Watchdog timer
- Kernel binaries
- Buffers vs images
- Blocking too much
- Half-full vectors
- Thread safety

Common Developer Questions

Half precision—16-bit floating point

- Storage only—convert to float
  - Buffers—vload_halfn / vstore_half*n*
  - Images—read_imagef / write_imagef

1	5	10
±	$2^{-14..15}$	1.m m = 10-bit

- cl_khr_fp16 extension for half arithmetic unsupported
  - Native hardware doesn't do it
  - Half precision arithmetic too lossy

Common Developer Questions

Local memory

- User managed cache
  - Copy data in—by hand or async_workgroup_copy
  - Read out as needed
  - Weakly ordered
    - Requires barrier to exchange data between workitems
- Only useful if you touch data more than once
  - GPU—use images
  - CPU—temporally local data should be spatially local too

Common Developer Questions

Local memory

- Can help both CPU and GPU
  - L1 cache on CPU
  - AoS to SoA transforms very helpful

AoS—Array of Structures

```
struct aos
{
    float x;
    float y;
    float z;
    char  telephone_number[16];
};
global struct aos[512];
```

SoA—Structure of Arrays

```
local struct soa
{
    float x[512];
    float y[512];
    float z[512];
};
```

Common Developer Questions

Watchdog timer

- Will kill GPU tasks that run longer than a few seconds
  - Console will have a message, such as:

```
Channel Exception type 0x8 = Fifo = Watchdog Timeout Error
```

- Divide large tasks into smaller ones

Common Developer Questions

Kernel binaries—precompiled OpenCL kernels

- clGetProgramInfo(CL_PROGRAM_BINARIES)
- Used as a program managed compiler cache
- Not for code obfuscation
- Don't ship these!
  - Apple hasn't committed to kernel binary compatibility yet
  - clBuildProgram will fail:

```
cl_program p = clCreateProgramWithBinary(...);
cl_int err = clBuildProgram( p, ... );      // FAILS!
```

  - Rebuild from source and overwrite cache

Common Developer Questions

Buffers vs. images

- Buffers
  - Fast on CPU—global L1, L2, L3 caches
  - Less fast on GPU—no cache
    - Local memory
    - Coalesced reads—contiguous addresses may run faster
- Images
  - Fast on GPU—texture unit
  - Slow on CPU
    - Software emulated
    - Accurate!—useful for debugging

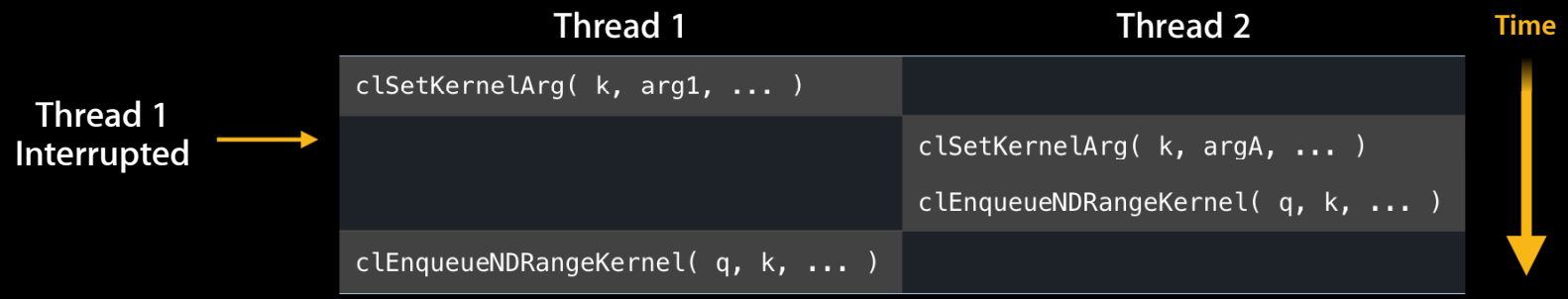
```
__read_2d_ff_normalized_linear_clamp_unorm8_rgba:
00000000000014f0        mulps   0x10(%rdi),%xmm0
00000000000014f4        movq    0x38(%rdi),%rax
00000000000014f8        subps   0x30(%rax),%xmm0
00000000000014fc        cvttps2dq       %xmm0,%xmm1
0000000000001500        cvtdq2ps        %xmm1,%xmm2
0000000000001503        movaps  %xmm0,%xmm3
0000000000001506        cmpps   $0x1,%xmm2,%xmm3
000000000000150a        paddd   %xmm1,%xmm3
000000000000150e        movaps  (%rdi),%xmm1
0000000000001511        movaps  %xmm3,%xmm4
0000000000001514        pcmpgtd %xmm1,%xmm4
0000000000001518        orps    %xmm3,%xmm4
000000000000151b        movd    %xmm4,%rcx
0000000000001520        movq    %rcx,%rdx
0000000000001523        sarq    $0x20,%rdx
0000000000001527        movq    0x20(%rdi),%rsi
000000000000152b        imulq   %rsi,%rdx
000000000000152f        shlq    $0x20,%rcx
0000000000001533        sarq    $0x1e,%rcx
0000000000001537        addq    %rdx,%rcx
000000000000153a        cmpps   $0x4,%xmm0,%xmm2
000000000000153e        movaps  %xmm3,%xmm5
0000000000001541        psubd   %xmm2,%xmm5
0000000000001545        movaps  %xmm5,%xmm2
0000000000001548        pcmpgtd %xmm1,%xmm2
000000000000154c        orps    %xmm5,%xmm2
000000000000154f        movmskps        %xmm2,%edx
0000000000001552        movmskps        %xmm4,%r8d
0000000000001556        movl    %edx,%r9d
0000000000001559        orl     %r8d,%r9d
000000000000155c        testl   $0x00000003,%r9d
0000000000001563        pcmpeqd %xmm2,%xmm4
0000000000001567        movmskps        %xmm4,%r9d
000000000000156b        cvtdq2ps        %xmm3,%xmm1
000000000000156e        subps   %xmm1,%xmm0
0000000000001571        jeq     0x0000167f
0000000000001577        testl   $0x00000002,%r8d
000000000000157e        jneq    0x00001716
0000000000001584        testl   $0x00000001,%r8d
000000000000158b        jneq    0x0000171e
0000000000001591        movq    0x30(%rdi),%r10
0000000000001595        movd    (%r10,%rcx),%xmm1
000000000000159b        xorps   %xmm2,%xmm2
000000000000159e        punpcklbw       %xmm2,%xmm1
00000000000015a2        punpcklbw       %xmm2,%xmm1
00000000000015a6        cvtdq2ps        %xmm1,%xmm1
00000000000015a9        mulps   0x90(%rax),%xmm1
00000000000015ad        testl   $0x00000001,%r9d
00000000000015ad        testl   $0x00000001,%r9d
00000000000015b4        jne     0x000015ed
00000000000015b6        testl   $0x00000001,%edx
00000000000015bc        jneq    0x00001726
00000000000015c2        movq    0x30(%rdi),%r10
00000000000015c6        movd    0x04(%rcx,%r10),%xmm2
00000000000015cd        xorps   %xmm3,%xmm3
00000000000015d0        punpcklbw       %xmm3,%xmm2
00000000000015d4        punpcklbw       %xmm3,%xmm2
00000000000015d8        cvtdq2ps        %xmm2,%xmm2
00000000000015db        mulps   0x90(%rax),%xmm2
00000000000015df        subps   %xmm1,%xmm2
00000000000015e2        pshufd  $0x0,%xmm0,%xmm3
00000000000015e7        mulps   %xmm2,%xmm3
00000000000015ea        addps   %xmm3,%xmm1
00000000000015ed        testl   $0x00000002,%r9d
00000000000015f4        jneq    0x00001712
00000000000015fa        testl   $0x00000002,%edx
0000000000001600        jneq    0x0000172e
0000000000001606        addq    %rsi,%rcx
0000000000001609        testl   $0x00000001,%r8d
0000000000001610        jneq    0x00001736
0000000000001616        movq    0x30(%rdi),%rsi
000000000000161a        movd    (%rsi,%rcx),%xmm2
000000000000161f        xorps   %xmm3,%xmm3
0000000000001622        punpcklbw       %xmm3,%xmm2
0000000000001626        punpcklbw       %xmm3,%xmm2
000000000000162a        cvtdq2ps        %xmm2,%xmm2
000000000000162d        mulps   0x90(%rax),%xmm2
0000000000001631        testl   $0x00000001,%r9d
0000000000001638        jne     0x00001670
000000000000163a        testl   $0x00000001,%edx
0000000000001640        jneq    0x0000173e
0000000000001646        movq    0x30(%rdi),%rsi
000000000000164a        movd    0x04(%rcx,%rsi),%xmm3
0000000000001650        xorps   %xmm4,%xmm4
0000000000001653        punpcklbw       %xmm4,%xmm3
0000000000001657        punpcklbw       %xmm4,%xmm3
000000000000165b        cvtdq2ps        %xmm3,%xmm3
000000000000165e        mulps   0x90(%rax),%xmm3
0000000000001662        subps   %xmm2,%xmm3
0000000000001665        pshufd  $0x0,%xmm0,%xmm4
000000000000166a        mulps   %xmm3,%xmm4
000000000000166d        addps   %xmm4,%xmm2
0000000000001670        subps   %xmm1,%xmm2
0000000000001673        pshufd  $0x55,%xmm0,%xmm0
0000000000001678        mulps   %xmm2,%xmm0
000000000000167b        addps   %xmm1,%xmm0
000000000000167e        ret
000000000000167f        testl   $0x00000001,%r9d
0000000000001686        movq    0x30(%rdi),%rdx
000000000000168a        je      0x000016d7
000000000000168c        movd    (%rdx,%rcx),%xmm1
0000000000001691        xorps   %xmm2,%xmm2
0000000000001694        punpcklbw       %xmm2,%xmm1
0000000000001698        punpcklbw       %xmm2,%xmm1
000000000000169c        cvtdq2ps        %xmm1,%xmm1
000000000000169f        movaps  0x90(%rax),%xmm2
00000000000016a3        mulps   %xmm2,%xmm1
00000000000016a6        testl   $0x00000002,%r9d
00000000000016ad        jne     0x00001712
00000000000016af        addq    %rsi,%rcx
00000000000016b2        movd    (%rdx,%rcx),%xmm3
00000000000016b7        xorps   %xmm4,%xmm4
00000000000016ba        punpcklbw       %xmm4,%xmm3
00000000000016be        punpcklbw       %xmm4,%xmm3
00000000000016c2        cvtdq2ps        %xmm3,%xmm3
00000000000016c5        mulps   %xmm2,%xmm3
00000000000016c8        subps   %xmm1,%xmm3
00000000000016cb        pshufd  $0x55,%xmm0,%xmm0
00000000000016d0        mulps   %xmm3,%xmm0
00000000000016d3        addps   %xmm1,%xmm0
00000000000016d6        ret
00000000000016d7        movq    (%rdx,%rcx),%xmm1
00000000000016dc        xorps   %xmm2,%xmm2
00000000000016df        punpcklbw       %xmm2,%xmm1
00000000000016e3        movaps  %xmm1,%xmm3
00000000000016e6        punpcklbw       %xmm2,%xmm3
00000000000016ea        cvtdq2ps        %xmm3,%xmm3
00000000000016ed        punpckhbw       %xmm2,%xmm1
00000000000016f1        cvtdq2ps        %xmm1,%xmm1
00000000000016f4        subps   %xmm3,%xmm1
00000000000016f7        pshufd  $0x0,%xmm0,%xmm2
00000000000016fc        mulps   %xmm2,%xmm1
00000000000016ff        addps   %xmm3,%xmm1
0000000000001702        movaps  0x90(%rax),%xmm3
0000000000001706        mulps   %xmm3,%xmm1
0000000000001709        testl   $0x00000002,%r9d
0000000000001710        je      0x00001746
0000000000001712        movaps  %xmm1,%xmm0
0000000000001715        ret
0000000000001716        xorps   %xmm1,%xmm1
0000000000001719        jmpq    0x1000015ed
000000000000171e        xorps   %xmm1,%xmm1
0000000000001721        jmpq    0x1000015ad
0000000000001726        xorps   %xmm2,%xmm2
0000000000001729        jmpq    0x1000015df
000000000000172e        xorps   %xmm2,%xmm2
0000000000001731        jmpq    0x100001670
0000000000001736        xorps   %xmm2,%xmm2
0000000000001739        jmpq    0x100001631
000000000000173e        xorps   %xmm3,%xmm3
0000000000001741        jmpq    0x100001662
0000000000001746        addq    %rsi,%rcx
0000000000001749        movq    (%rdx,%rcx),%xmm4
000000000000174e        xorps   %xmm5,%xmm5
0000000000001751        punpcklbw       %xmm5,%xmm4
0000000000001755        movaps  %xmm4,%xmm6
0000000000001758        punpcklbw       %xmm5,%xmm6
000000000000175c        cvtdq2ps        %xmm6,%xmm6
000000000000175f        punpckhbw       %xmm5,%xmm4
0000000000001763        cvtdq2ps        %xmm4,%xmm4
0000000000001766        subps   %xmm6,%xmm4
0000000000001769        mulps   %xmm2,%xmm4
000000000000176c        addps   %xmm6,%xmm4
000000000000176f        mulps   %xmm3,%xmm4
0000000000001772        subps   %xmm1,%xmm4
0000000000001775        pshufd  $0x55,%xmm0,%xmm0
000000000000177a        mulps   %xmm4,%xmm0
000000000000177d        addps   %xmm1,%xmm0
0000000000001780        ret
```

168 Instructions

Common Developer Questions

Thread safety

- Use a separate queue for each user thread
- Avoid reentrantly calling APIs, with the same object
  - e.g. two threads using the same kernel:



- Could use locks but making a second kernel is cheaper

Common Performance Problems

Blocking too much

- clFinish
  - Almost never needed
  - Use to shut down OpenCL completely
- clEnqueueRead<Buffer/Image>
  - Often blocking
  - Only last one usually needs to be blocking—in order queue
- clEnqueueWrite<Buffer/Image>
  - Usually doesn't need to block—block on later read
  - Block when you need to free() the source buffer right away
- clEnqueueMap/Unmap<Buffer/Image>

Common Performance Problems

Half-full vectors

- CPU
  - Empty lanes, vector too small
  - Floating point hazards from denormal, inf, NaN
  - Wasted arithmetic bandwidth for all types
- GPU—reverse problem
  - Empty lanes, vector too big
  - Doing useless work

Recycle Buffers and Images

- OpenCL objects can be heavy
  - Programs—have to compile
  - Images/buffers
    - Backing store
      - Bunch of driver calls to set up
      - Zero fill
    - State—who used it last
- Recycle old images and buffers
  - Save time recreating the object
  - Avoid zero fill
  - Only useful if approximately the same size

Debugging Tips

Debugging Tips

- CL_LOG_ERRORS
  - Export CL_LOG_ERRORS = ”stdout” or “stderr” or “console”
  - Also clCreateContext() with pfn_notify:
    - clLogMessagesToStdoutAPPLE
    - clLogMessagesToStderrAPPLE
    - clLogMessagesToSystemLogAPPLE (console)
    - Roll your own!
- CPU
  - Shark
  - Instruments

Profiling Demo

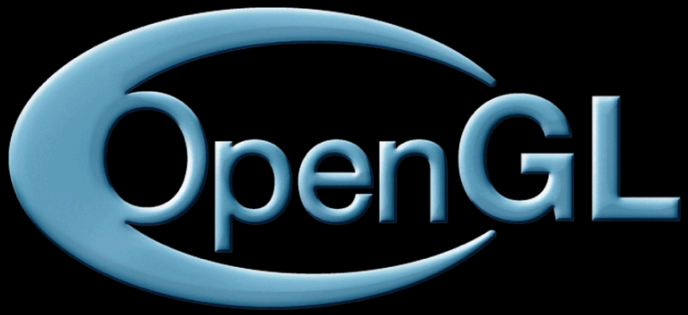
Ian Ollmann, Ph.D.

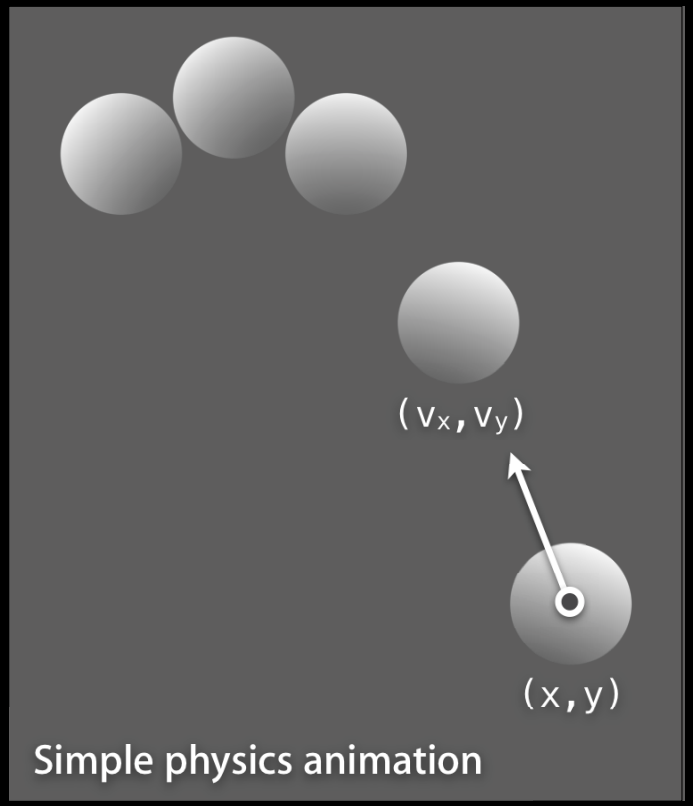
OpenCL/OpenGL Sharing

Abe Stephens
OpenCL Engineer

Motivation

Combine graphics and compute capability



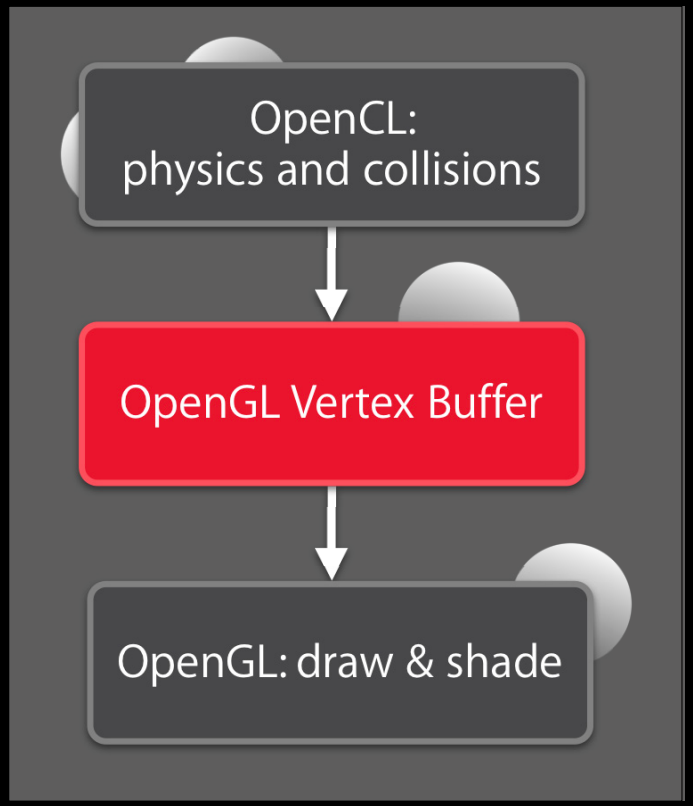


Motivation

Combining compute and graphics capability

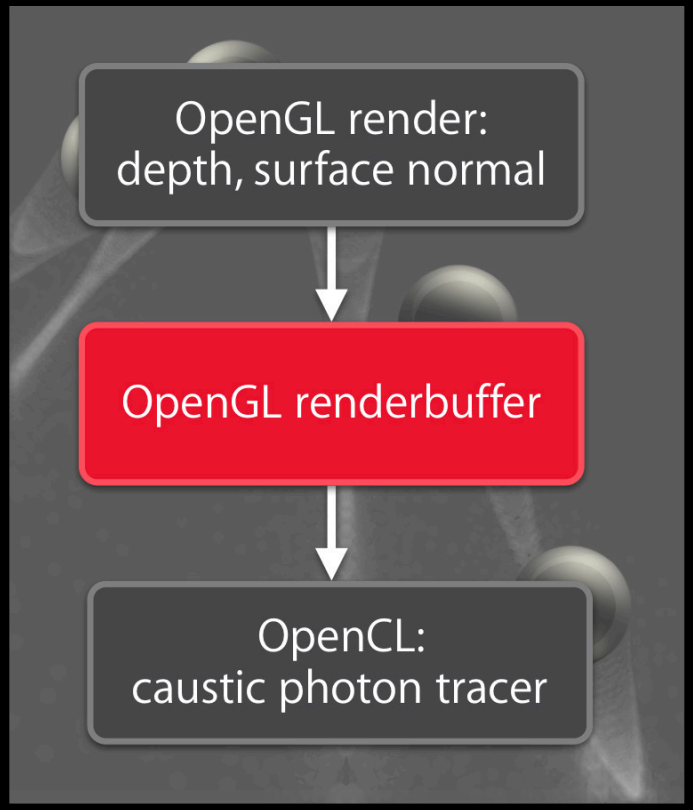
- Compute position and velocity
- Render spheres

Geometry from OpenCL

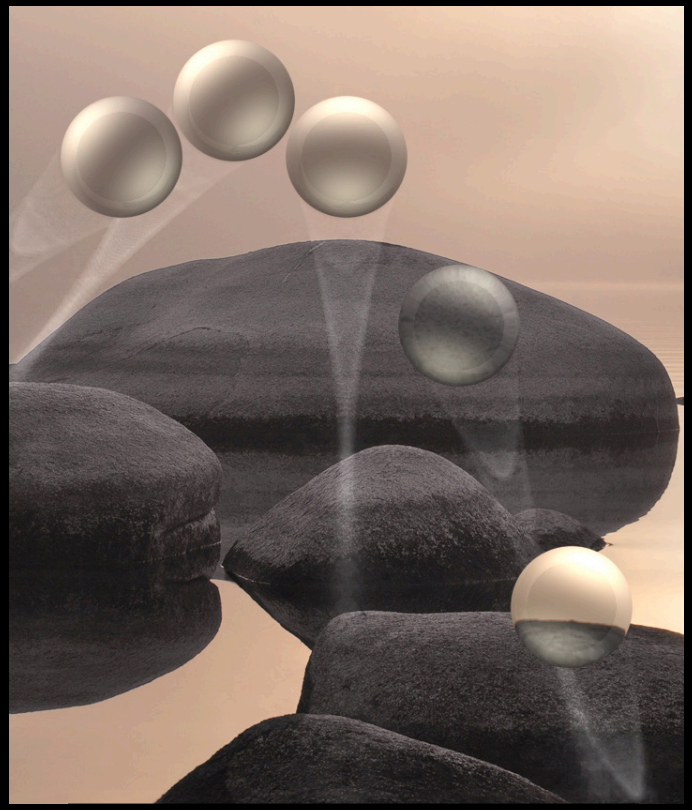


Draw with shader in OpenGL

OpenCL Post-process



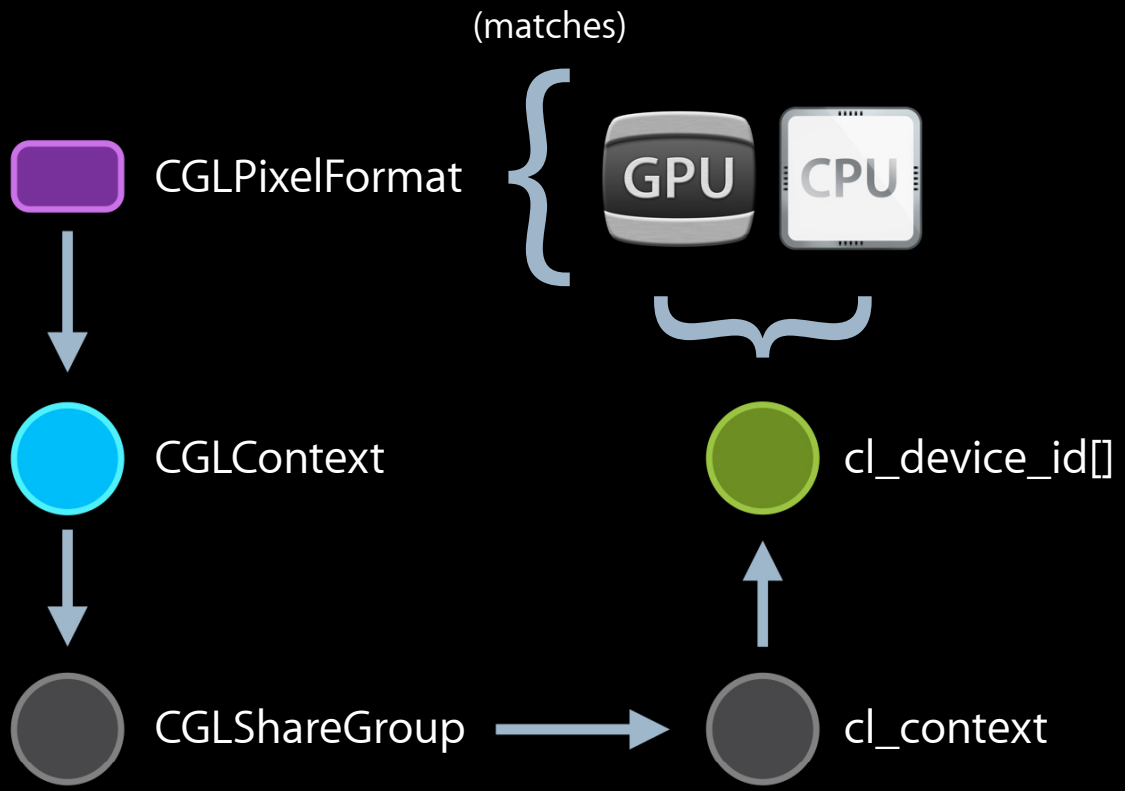
Post-process effect in CL



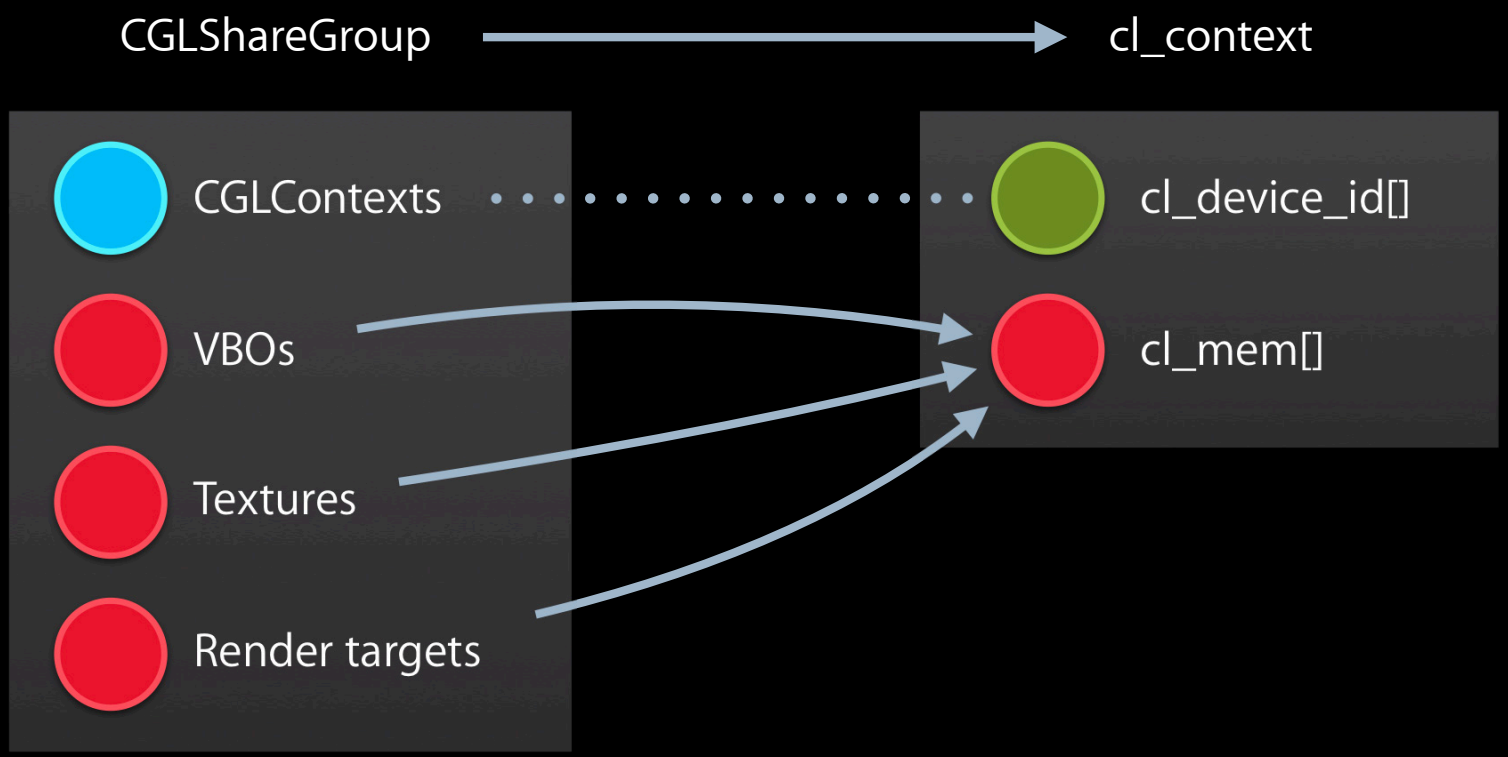
Composite result in OpenGL

Sharing Fundamentals

Setup


(matches)
CGLPixelFormat
GPU
CPU
CGLContext
cl_device_id[]
CGLShareGroup
cl_context

Similar Structures



Steps for Sharing with CL

1. Obtain CGL context
2. Create cl_context
3. Import data
4. Flush and acquire
5. Release safely

Getting Started

Step 1: Obtain CGLContext

Obtaining Share Group with Cocoa/GL

```
@interface BounceView : NSOpenGLView { ... }
@implementation BounceView



- (void)applicationWillFinishLaunching:
    (NSNotification *)aNotification {
// ...


// Obtain the context associated with the Cocoa view.
CGLContextObj cgl_context =
  (CGLContextObj)[[view openGLContext] CGLContextObj]


// Obtain the sharegroup.
CGLShareGroupObj shared = CGLGetShareGroup(cgl_context);
```

Initializing OpenCL

Step 2: Create CL context

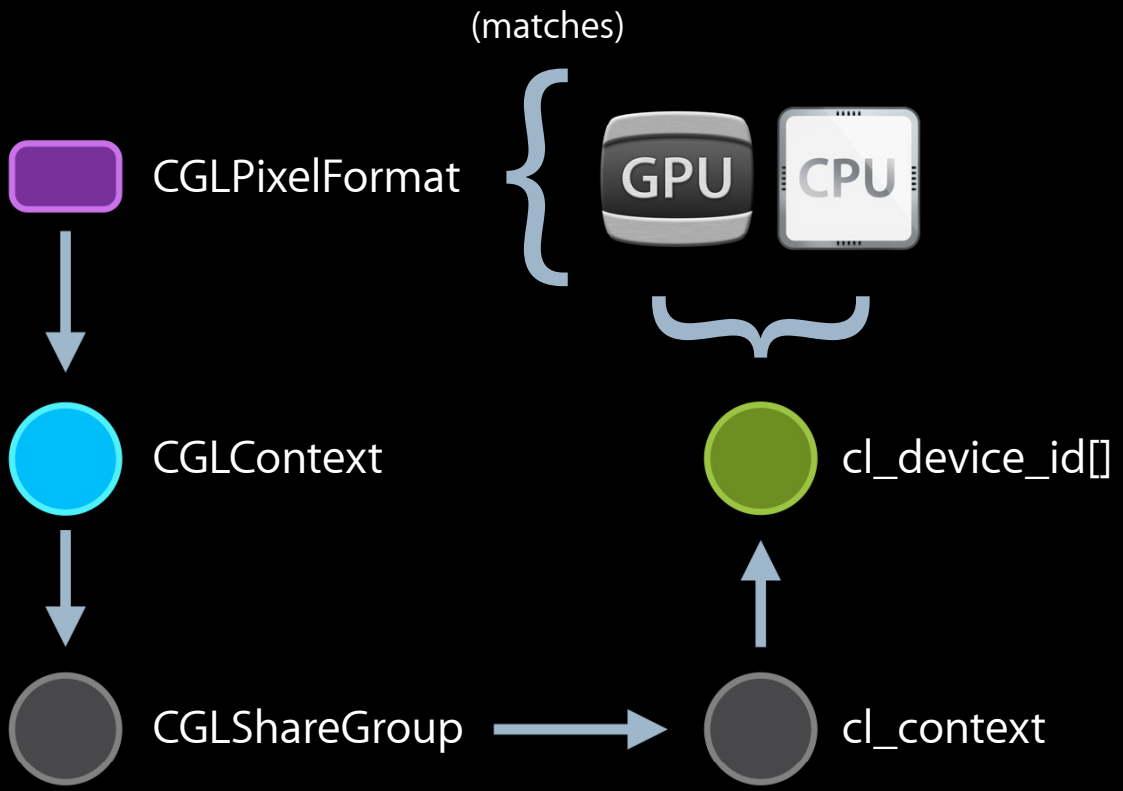
Create an OpenCL Context

```
// Create OpenCL context
cl_context_properties properties[] = {
  CL_CONTEXT_PROPERTY_USE_CGL_SHAREGROUP_APPLE,
    (cl_context_properties)shared,
  0
};
cl_context cl_ctx =
  clCreateContext(properties, 0, NULL, NULL, NULL, &err);
```

Obtaining cl_device_ids

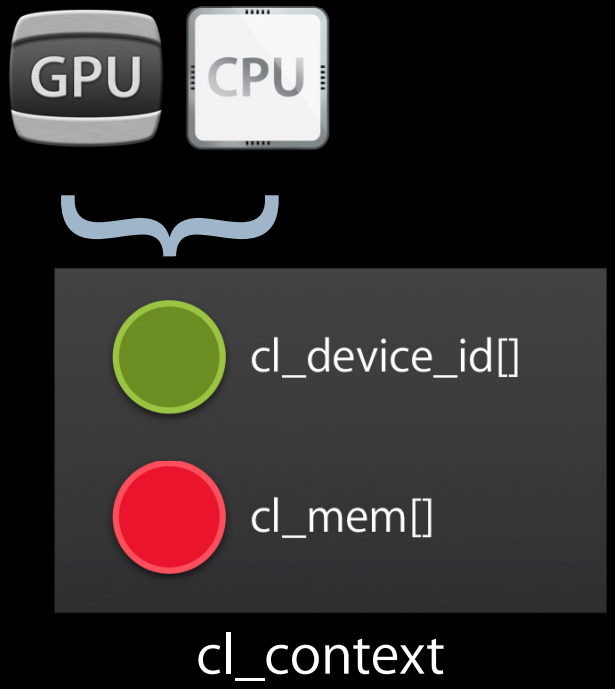
```
// Obtain all of the devices
cl_device_id all_devices[bytes/sizeof(cl_device_id)];
clGetContextInfo(cl_ctx,CL_CONTEXT_DEVICES,bytes,all_devices,NULL);
```

```
// Obtain only the device for the current virtual screen
cl_device_id cl_device;
clGetGLContextInfoAPPLE( cl_ctx, cgl_context,
  CL_CGL_DEVICE_FOR_CURRENT_VIRTUAL_SCREEN_APPLE,
  sizeof(cl_device_id), &cl_device, NULL);
```


(matches)
CGLPixelFormat
GPU
CPU
CGLContext
cl_device_id[]
CGLShareGroup
cl_context

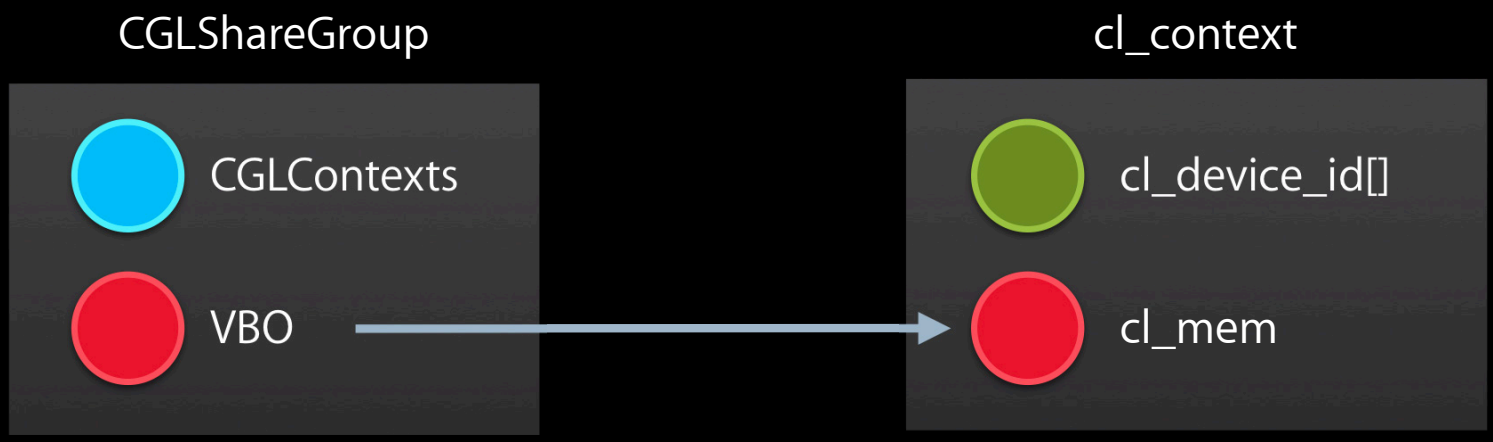
Using the CPU device

```
cl_device_id cpu;
clGetDeviceIDs(NULL,CL_DEVICE_TYPE_CPU,
  1,&cpu,NULL);

cl_context cl_ctx =
  clCreateContext(properties,
    1, &cpu,NULL, NULL, &err);
```

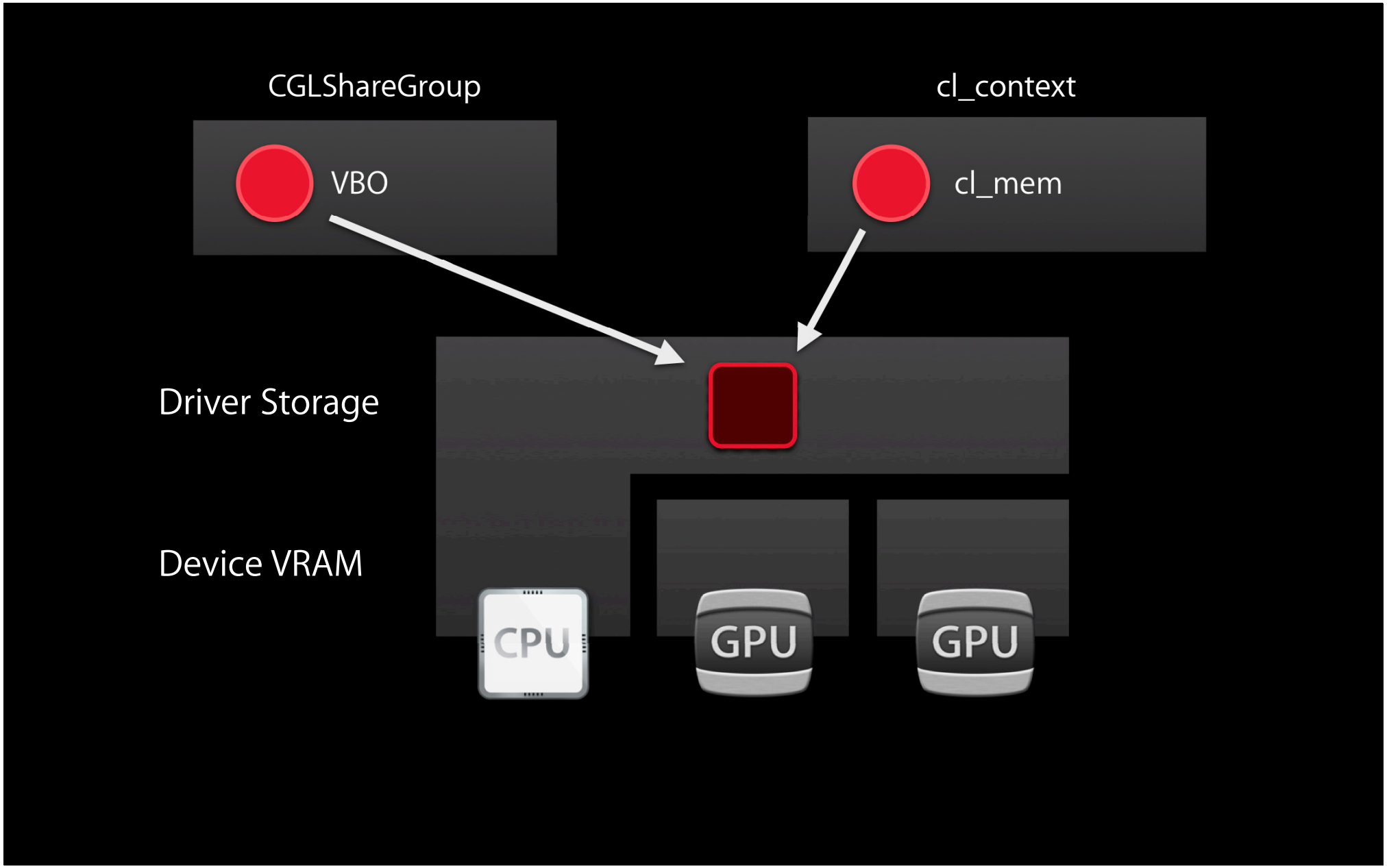


Now We Have a CL Context and Device

Step 3: Import shared objects




```
// Create cl_mem object from GL VBO
cl_int err;
buffer_cl =
 clCreateFromGLBuffer(cl_ctx,CL_MEM_READ_WRITE,buffer_gl,&err);
```


CGLShareGroup
cl_context
VBO
cl_mem
Driver Storage
Device VRAM
CPU
GPU
GPU

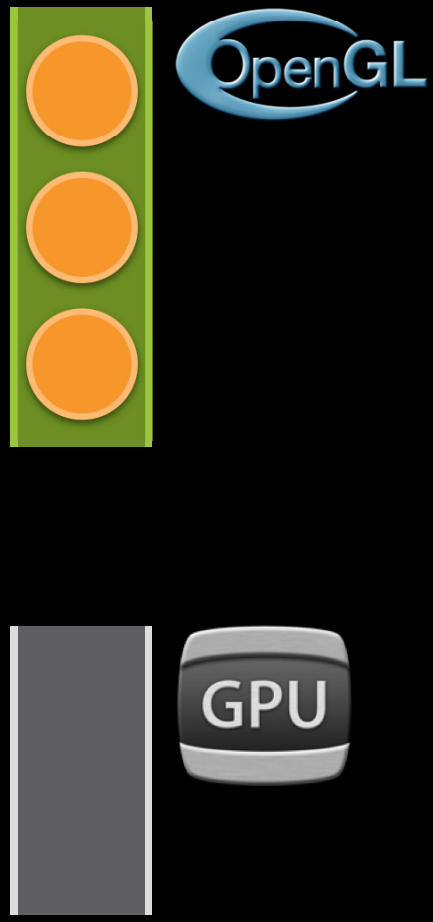
Three Other Creation Functions Available

```
clCreateFromGLTexture2D(context,flags,target,miplevel,texture,&err)

clCreateFromGLTexture3D(context,flags,target,miplevel,texture,&err)

clCreateFromGLRenderbuffer(context,flags,renderbuffer,&err)
```

Launching Commands

Step 4: Flush and acquire

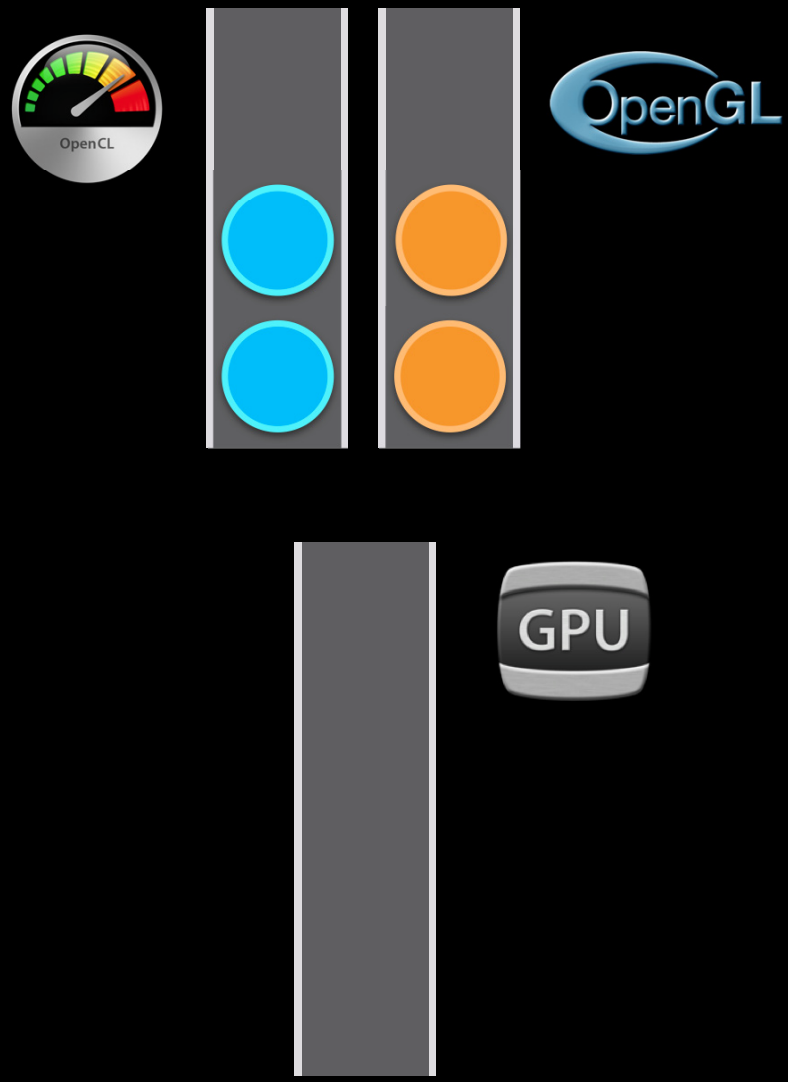
Graphics Alone

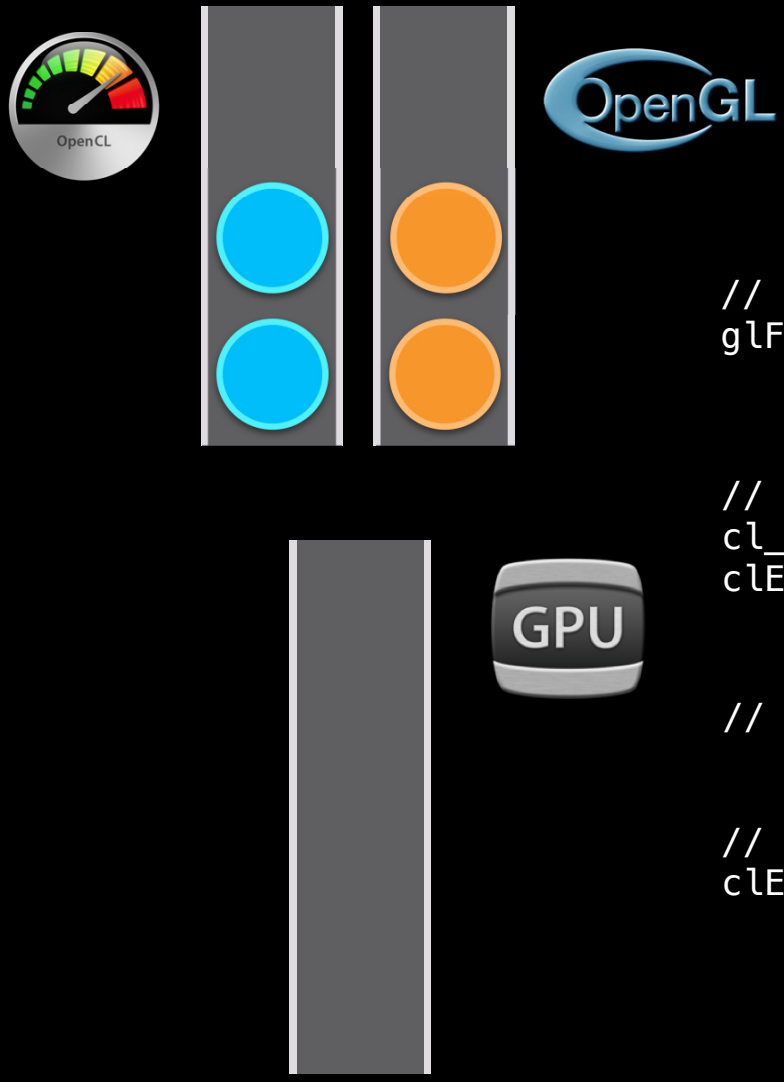


- CGLContext contains an implicit command queue
- Order maintained within queue

Graphics and Compute

No synchronization





Graphics and Compute

With synchronization

```
// Done with previous GL commands
glFlush();


// Update geometry in OpenCL
cl_mem lt[] = { buffer_cl };
clEnqueueAcquireGLObjects(queue,1,lt,...);


// ...


// Done with CL commands
clEnqueueReleaseGLObjects(queue,1,lt,...);
```

Release and Cleanup

Step 5: Safe shutdown

Release on compute side first

```
clReleaseMemObject(buffer_cl)
```

Sharing Demo

Demo Details

Steps to render frame

- Update vertex positions
- Photon trace
- Render scene

Summary

- Use pixel format to select devices
- Pass share group to clCreateContext
- Create objects in OpenGL, then import to OpenCL
- glFlush before acquiring shared objects
- Release in OpenCL before destroying in OpenGL

More Information

Allan Schaffer
Graphics and Game Technology Evangelist
aschaffer@apple.com

Apple Developer Forums
http://devforums.apple.com

Related Sessions

Maximizing OpenCL Performance	Russian Hill Wednesday 4:30PM
OpenGL for Mac OS X	Nob Hill Thursday 9:00AM
Taking Advantage of Multiple GPUs	Nob Hill Thursday 10:15AM

Labs

OpenCL Lab	Graphics & Media Lab C Thursday, 9:00AM
OpenGL for Mac OS X Lab	Graphics & Media Lab C Thursday, 2:00PM


WWDC10