OpenCL Events

An event is an object that communicates the status of OpenCL commands.

From the OpenCL Notes:
11. Enqueue the Kernel Object for Execution

```c
size_t globalWorkSize[3] = { NUM_ELEMENTS, 1, 1 };
size_t localWorkSize[3] = { LOCAL_SIZE, 1, 1 };
status = clEnqueueNDRangeKernel( cmdQueue, kernel, 1, NULL, globalWorkSize, localWorkSize, 0, NULL, NULL );
```

status = clEnqueueNDRangeKernel( cmdQueue, kernel, 1, NULL, globalWorkSize, localWorkSize, 0, NULL, &waitKernelC);

Creating an Event

event that will be thrown when this kernel is finished executing

```c
cl_event waitKernelA, waitKernelB, waitKernelC;
status = clEnqueueNDRangeKernel( cmdQueue, kernel, 1, NULL, globalWorkSize, localWorkSize, 0, NULL, &waitKernelC);
```
Waiting for Events from Previously-Executed Kernels

cl_event waitKernelA, waitKernelB, waitKernelC;
...;
cl_event dependenciesAB[2];
dependenciesAB[0] = waitKernelA;
dependenciesAB[1] = waitKernelB;
status = clEnqueueNDRangeKernel(cmdQueue, kernelC, 1, NULL, globalWorkSize, localWorkSize, 2, dependenciesAB, NULL);

Creating an Execution Graph Structure

A

B

C

D

E

Waiting for One Event

cl_event waitKernelA, waitKernelB;
...;
status = clEnqueueNDRangeKernel(cmdQueue, kernelC, 1, NULL, globalWorkSize, localWorkSize, 1, &waitKernelA, NULL);

Creating the Full Execution Graph Structure

A

B

C

D

E

Computer Graphics
Placing a Barrier in the Command Queue

```c
status = clEnqueueBarrier( cmdQueue );
```

Note: this cannot throw its own event

This does not complete until all commands enqueued before it have completed.

---

Placing an Event Marker in the Command Queue

```c
cl_event waitMarker;
status = clEnqueueMarker( cmdQueue, &waitMarker );
```

Note: this can throw its own event

This does not complete until all commands enqueued before it have completed.

This is just like a barrier, but it can throw an event to be waited for.

---

Waiting for Events Without Enqueuing Another Command

```c
status = clWaitForEvents( 2, dependencies );
```

This blocks until the specified events are thrown, so use it carefully!

---

I Like Synchronizing Things This Way

```c
// wait until all queued tasks have taken place:
void
Wait( cl_command_queue queue )
{
    cl_event wait;
    cl_int status;
    status = clEnqueueMarker( queue, &wait );
    if( status != CL_SUCCESS )
        fprintf( stderr, "Wait: clEnqueueMarker failed
" );
    status = clWaitForEvents( 1, &wait ); // blocks until everything is done!
    if( status != CL_SUCCESS )
        fprintf( stderr, "Wait: clWaitForEvents failed
" );
}
```

Call this before starting the timer, before ending the timer, and before retrieving data from an array computed in an OpenCL program.
Getting Event Statuses Without Blocking

CL_EVENT_COMMAND_QUEUE
CL_EVENT_CONTEXT
CL_EVENT_COMMAND_TYPE
CL_EVENT_COMMAND_EXECUTION_STATUS

Specify one of these

cl_int eventStatus;
status = clGetEventInfo( waitKernelC, CL_EVENT_COMMAND_EXECUTION_STATUS, sizeof(cl_int),
&eventStatus, NULL );

CL_EVENT_COMMAND_EXECUTION_STATUS
returns one of these

CL_QUEUED
CL_SUBMITTED
CL_RUNNING
CL_COMPLETE

Note that this a nice way to check on event statuses without blocking. Thus, you could put
this in a loop and go get some other work done in between calls.