Developing an efficient Maya plug-in using CUDA & GLInterop.

Wil Braithwaite - NVIDIA Applied Engineering
Our Particle pipeline

- Emit particles
- Simulate
- Render
- Export
Particle Emission

- Upload emitted data using nParticles as offset.

- nParticles += nEmitted

- We use fast host-to-device transfer for emission data.
  - i.e. page-locked & write-combined host memory.

- For large emissions we can double-buffer and use Async API.
The App Pipeline - (Autodesk Maya)

- Maya plug-in callbacks:
  - Plugin::compute(...) (called during data computation)
    - Upload & Simulate the particles using CUDA.
    - Blit simulated data to VBO.
  - Plugin::draw(...) (called during Maya’s scene-graph traversal)
    - Render particle VBO.
CUDA contexts in Maya

- **CUDA context**
  - We must create one. Where?
  - Other plug-ins (or “future” Maya) might change CUDA “state”.

- **Choices:**
  - Push and pop any previous CUDA context (using driver API).
  - Or create our own “persistent” thread with dedicated CUDA context.

- If we want GL interoperability, Maya’s GL context must be passed into our persistent thread via context sharing.
Code: persistent CUDA context thread

// Initialize from within Maya thread ...
void Plugin::_initializeThread() {
    HDC display = getDisplay();
    HGLRC mayaGlCxt = getGlContext();
    HGLRC glCxt = wglCreateContext(display);
    wglShareLists(mayaGlCxt, glCxt);
    _hdl = runThread(_threadFunc, display, glCxt);
    launchJob(jobInit);
    wglMakeCurrent(display, mayaGlCxt);
}

// Clean-up from within Maya thread ...
void Plugin::_destroyThread () {
    launchJob(jobExit);
    killThread(_hdl);
}

// Our persistent thread function...
void _threadFunc(HDC display, HGLRC glCxt) {
    wglMakeCurrent(display, glCxt);
    CUdevice dev;
    cuDeviceGet(&dev, 0);
    CUcontext cuCxt;
    cuGLCtxCreate(&cuCxt, 0, dev);
    while (isRunning) {
        // pop & run job from thread-safe queue...
    }
    wglMakeCurrent(0, 0);
    wglDeleteContext(glCxt);
    cuCtxDestroy(cuCxt);
}

void jobInit(Data* ) { // initialize CUDA things... }
void jobExit(Data*) { // clean-up CUDA things... }
Particle Initialization

- Simulation & Rendering initialization

```c
struct Data {
    int nParticles; // number of particles
    GLuint glVboPositions; // OpenGL VBO of particle positions
    cudaGraphicsResource* cuGlRes; // CUDA registered VBO
    cudaStream_t cuStream; // CUDA stream
    float4* cuPositions; // CUDA buffer of particle positions
};

void jobInit(Data* d) {
    cudaStreamCreate(&d->cuStream);
    cudaMalloc(d->cuPositions, d->nParticles * sizeof(float4));
    glGenBuffers(1, &d->glVboPositions)
    glBindBuffer(GL_ARRAY_BUFFER, d->glVboPositions);
    glBufferData(GL_ARRAY_BUFFER, d->nParticles * sizeof(float4), 0, GL_DYNAMIC_DRAW);
    glBindBuffer(GL_ARRAY_BUFFER, 0);
    cudaGraphicsGLRegisterBuffer(&d->cuGlRes, d->glVboPositions, cudaGraphicsMapFlagsWriteDiscard);
}
```
Particle Simulation

- CUDA kernel operates on CUDA buffers.

```c
void jobSim(Data* d)
{
    // ...
    somethingCrazyKernel<<<dimGrid, dimBlock, 0, d->cuStream>>>(d->nParticles, d->cuPositions);
}
```
Particle Render Blit

- **Blit into VBOs**
  
  - copy the CUDA simulated data into the GL buffers.

```c
void jobBlit(Data* d)
{
    void* glVboPositionsPtr = 0;
    size_t nBytes = 0;
    cudaGraphicsMapResources(1, &d->cuGlRes, d->cuStream);
    cudaGraphicsResourceGetMappedPointer(&glVboPositionsPtr, (size_t*)&nBytes, d->cuGlRes);

    cudaMemcpyAsync(glVboPositionsPtr, d->cuPositions, nBytes, cudaMemcpyDeviceToDevice, d->cuStream);
    cudaGraphicsUnmapResources(1, &d->cuGlRes, d->cuStream);
}

void jobSync(Data* d)
{
    cudaStreamSynchronize(d->cuStream);
}
```
void Plugin::draw()
{
    launchJob(jobSync);
    Data* d = &_data;
    glBindBuffer(GL_ARRAY_BUFFER, d->glVboPositions);
    glVertexPointer(4, GL_FLOAT, 0, 0);
    glEnableClientState(GL_VERTEX_ARRAY);
    glDrawArrays(GL_POINTS, 0, d->nParticles);
    glBindBuffer(GL_ARRAY_BUFFER, 0);
    glDisableClientState(GL_VERTEX_ARRAY);
}
Example 1: Synchronous CUDA thread

- Sync with CUDA thread before we draw the buffers.
  - use a semaphore.
  - or... simply synchronize the CUDA thread: `jobSync()`
Example 1: Synchronous CUDA thread

- Sync with CUDA thread before we draw the buffers.
- We must synchronize our blit with the GL render.
Example 1: Synchronous CUDA thread

- Sync with CUDA thread before we draw the buffers.
- Sync with OpenGL before we blit to the buffers.

Blit waits for GL render to finish
void Plugin::draw()
{
    Data* d = &_data;
    d->blitMutex.claim();

    glBindBuffer(GL_ARRAY_BUFFER, d->glVboPositions);
    glVertexPointer(4, GL_FLOAT, 0, 0);
    glEnableClientState(GL_VERTEX_ARRAY);
    glDrawArrays(GL_POINTS, 0, d->nParticles);
    glBindBuffer(GL_ARRAY_BUFFER, 0);
    glDisableClientState(GL_VERTEX_ARRAY);
    d->glVboSync = glFenceSync(GL_SYNC_GPU_COMMANDS_COMPLETE, 0);
    d->blitMutex.release();
}
Particle Render Blit v2

- Wait for GL to finish rendering from the vbo.

```c
void jobBlit(Data* d)
{
    d->blitMutex.claim();

    if (glIsSync(d->glVboSync))
    {
        glClientWaitSync(d->glVboSync, GL_SYNC_FLUSH_COMMANDS_BIT, GL_FOREVER);
        glDeleteSync(d->glVboSync);
        d->glVboSync = 0;
    }

    void* glVboPositionsPtr = 0;
    size_t nBytes = 0;
    cudaGraphicsMapResources(1, &d->cuGlRes, d->cuStream);
    cudaGraphicsResourceGetMappedPointer(&glVboPositionsPtr, (size_t*)&nBytes, d->cuGlRes);
    cudaMemcpyAsync(glVboPositionsPtr, d->cuPositions, nBytes, cudaMemcpyDeviceToDevice, d->cuStream);
    cudaGraphicsUnmapResources(1, &d->cuGlRes, d->cuStream);

    d->blitMutex.release();
}
```
Example 1: Synchronous CUDA thread

- Advantages:
  - Simple to implement.
  - Multi-threading protects our plug-in / host app.

- Disadvantages:
  - We are relying on Maya calling our update at the start, and our render at the end!
Example 2: Double-buffered CUDA thread

- Allow latency and overlap sim with host app.
- Ping-pong into double-buffered VBOs.
  - (We still require sync objects if draw[readIndex] overlaps with blit[writelIndex].)

Diagram:
- Thread: sim, blit
- Maya: compute, draw the previous frame, draw
- OpenGL: GL render
- Frame: 2, 3
void jobBlit(Data* d)
{
    d->blitMutex.claim();

    if (glIsSync(d->glVboSync[d->current]))
    {
        glClientWaitSync(d->glVboSync[d->current], GL_SYNC_FLUSH_COMMANDS_BIT, GL_FOREVER);
        glDeleteSync(d->glVboSync[d->current]);
        d->glVboSync[d->current] = 0;
    }

    void* glVboPositionsPtr = 0;
    size_t nBytes = 0;
    cudaGraphicsMapResources(1, &d->cuGlRes[d->current], d->cuStream);
    cudaGraphicsResourceGetMappedPointer(&glVboPositionsPtr, (size_t*)&nBytes, d->cuGlRes[d->current]);
    cudaMemcpyAsync(glVboPositionsPtr, d->cuPositions, nBytes, cudaMemcpyDeviceToDevice, d->cuStream);
    cudaGraphicsUnmapResources(1, &d->cuGlRes[d->current], d->cuStream);
    d->current = (++d->current) % 2; // ping-pong

    d->blitMutex.release();
}
```cpp
void Plugin::draw()
{
    Data* d = &_data;
    d->blitMutex.claim();

    glBindBuffer(GL_ARRAY_BUFFER, d->glVboPositions[d->current]);
    glVertexPointer(4, GL_FLOAT, 0, 0);
    glEnableClientState(GL_VERTEX_ARRAY);
    glDrawArrays(GL_POINTS, 0, d->nParticles);
    glBindBuffer(GL_ARRAY_BUFFER, 0);
    glDisableClientState(GL_VERTEX_ARRAY);

    d->glVboSync[d->current] = glFenceSync(GL_SYNC_GPU_COMMANDS_COMPLETE, 0);

    d->blitMutex.release();
}
```
Asynchronous CUDA streams

- Batching the data means...
  - We hide the cost of data transfer between device and host.
  - Extension to Multi-GPU is now trivial.
- NB. Not all algorithms can batch data.
- Each batch’s stream requires resource allocation.
  - Be sure to do this up-front, (before OpenGL gets it all!)
  - Number of batches can be chosen based on available resources.
NVIDIA Maximus

- Quadro AND Tesla with 6GB & 448 cores.

- Benefits of Maximus
  - Uses GL-interop for efficient data movement to the Quadro GPU
  - DCC apps like Maya already use many GPU resources.
  - Share the data between two cards allowing for larger simulations.
  - Multi-GPU is scalability for the future.
  - “batch” simulation on your workstation.
Demo
GTC 2013 | March 18-21 | San Jose, CA
The Smartest People. The Best Ideas. The Biggest Opportunities.

Opportunities for Participation:

**SPEAK** - Showcase your work among the elite of graphics computing
- Call for Sessions: August 2012
- Call for Posters: October 2012

**REGISTER** - learn from the experts and network with your peers
- Use promo code **GM10SIGG** for a 10% discount

**SPONSOR** - Reach influential IT decision-makers

Learn more at [www.gputechconf.com](http://www.gputechconf.com)