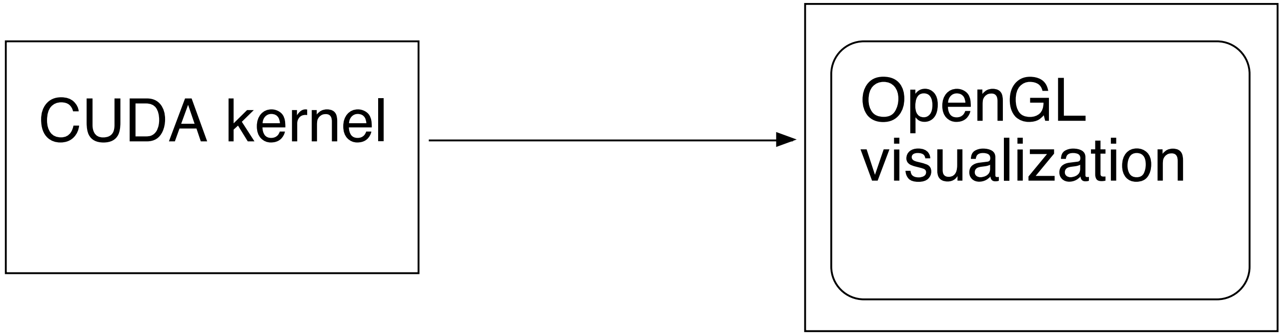
CUDA-OpenGL Interoperability

Visualize results with OpenGL



CUDA kernel

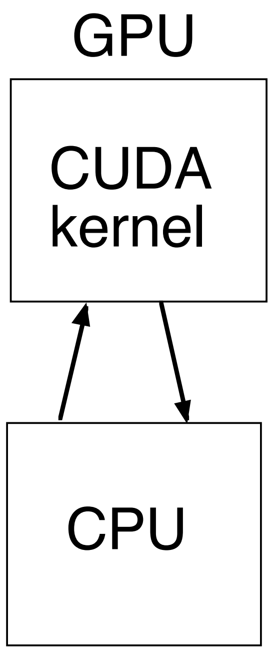
OpenGL visualization

CUDA-OpenGL Interoperability

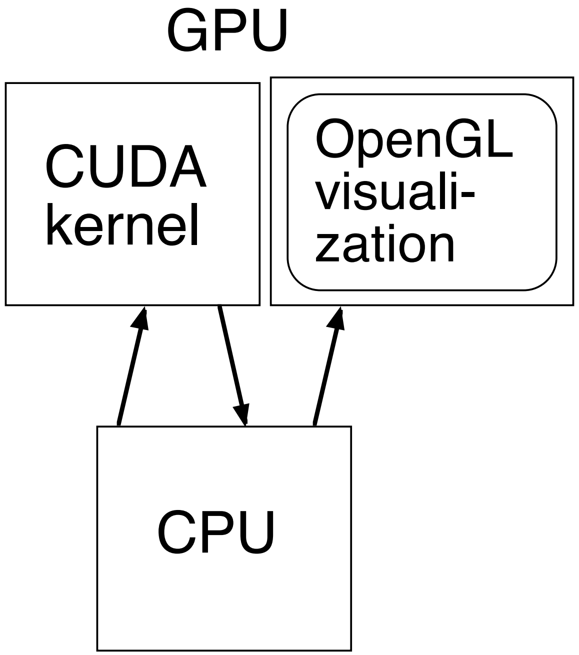
- Great performance
- Possible to visualize without leaving GPU

An output which is not the CPU

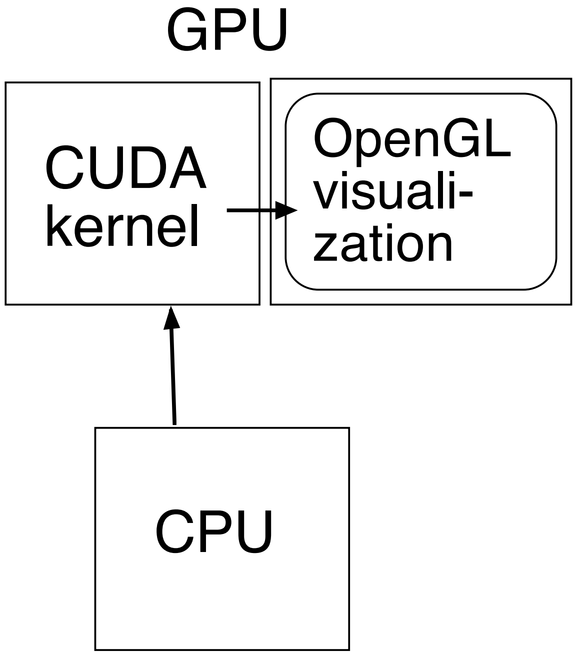
No visualization



Simple visualization



Visualization with OpenGL interoperability



Steps for interoperability

- Decide what data CUDA will process
- Allocate with OpenGL
- Register with CUDA
- Map buffer to get CUDA pointer
- Pass pointer to CUDA kernel
- Release pointer
- Use result in OpenGL graphics

- **Allocate with OpenGL**
- **Register with CUDA**

```
glGenBuffers(1, &positionsVBO);
glBindBuffer(GL_ARRAY_BUFFER, positionsVBO);
unsigned int size = NUM_VERTS * 4 * sizeof(float);
glBufferData(GL_ARRAY_BUFFER, size, NULL,
GL_DYNAMIC_DRAW);
glBindBuffer(GL_ARRAY_BUFFER, 0);
```

Allocate
VBO (vertex
buffer)

```
cudaGraphicsGLRegisterBuffer(&positionsVBO_CUDA
, positionsVBO, cudaGraphicsMapFlagsWriteDiscard);
```

Register with
CUDA

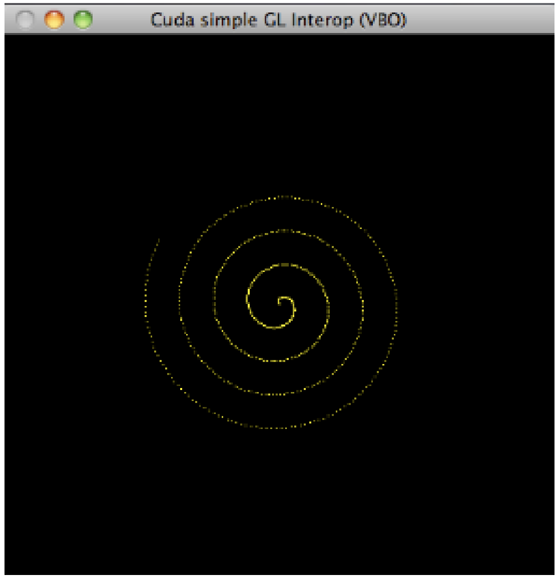
- **Map buffer to get CUDA pointer**
- **Pass pointer to CUDA kernel**
- **Release pointer**

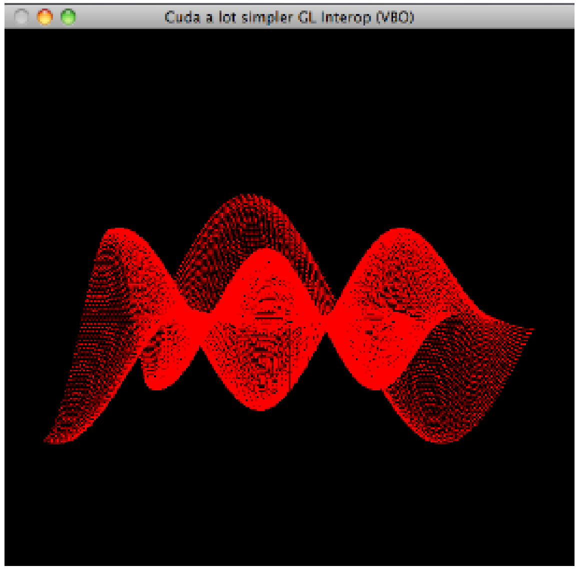
```
cudaGraphicsMapResources(1, &positionsVBO_CUDA, 0);
size_t num_bytes;
cudaGraphicsResourceGetMappedPointer((void**)&positions, &num_bytes,
positionsVBO_CUDA);printError(NULL, err);

// Execute kernel
dim3 dimBlock(16, 1, 1);
dim3 dimGrid(NUM_VERTS / dimBlock.x, 1, 1);
createVertices<<<dimGrid, dimBlock>>>(positions, anim, NUM_VERTS);

// Unmap buffer object
cudaGraphicsUnmapResources(1, &positionsVBO_CUDA, 0);
```

Simple CUDA kernel for producing vertices for graphics

```
// CUDA vertex kernel
__global__ void createVertices(float4* positions, float time, unsigned int num)
{
    unsigned int x = blockIdx.x*blockDim.x + threadIdx.x;

    positions[x].w = 1.0;
    positions[x].z = 0.0;
    positions[x].x = 0.5*sin(kVarv * (time + x * 2 * 3.14 / num)) * x/num;
    positions[x].y = 0.5*cos(kVarv * (time + x * 2 * 3.14 / num)) * x/num;
}
```

Simple examples:





Just vertices - but you can draw surfaces, compute textures, use any OpenGL effects (light, materials)

But should we use CUDA for OpenGL?

Great for visualizing

Faster than going over CPU

but OpenGL has similar functionality built-in! (Compute Shaders.)

Next time....

More to check out

Debugging with cudagdb

Doing printf() from CUDA threads (yes you can!)

Running on multiple GPUs

That’s all folks!

Next time: OpenCL and shaders