

## Common GPU Problems and How To Solve Them

### My current development environment:

I use Microsoft Visual Studio for coding and do almost everything in C/C++. I commonly use OpenGL along with CUDA and Qt to create OpenGL contexts and user interfaces. All of these things work well together but there are a few problems that I find people often run into.

### 1) Are OpenGL functions that you see in books and tutorials giving you “identifier not found” errors?

I have never met a tutorial that actually tells you this: Any OpenGL function introduced after version 1.1 is not included in the gl.h header file. There are two ways to deal with this:

- (a) The hard way – Query the OpenGL library for the appropriate function and map it to a function pointer that you create. I don’t do this.
- (b) The easy way – Get the OpenGL Extension Wrangler (GLEW) library. Just Google it and download the binaries. Link to `glew32.lib` and include `glew.h` (it has to be included before `gl.h`). Next, add the following code:

```
GLenum err = glewInit();
if(GLEW_OK != err)
{
    printf("Error starting GLEW.");
}
```

One more thing! This code has to be included *after* you create an OpenGL context. If you use Qt, this means that you have to create an OpenGL widget first. If you use GLUT, put the above code after your `glutCreateWindow()` call. Initializing GLEW before creating an OpenGL context is the only reason I’ve found for getting a GLEW error. Further GLEW instructions are on the SourceForge website.

## 2) How do I set up a 3D texture in CUDA?

Use the following code as a template (these concepts are also shown in the volume rendering example of the CUDA SDK). This code loads a single-channel 3D 8-bit data set and prints the value in the center.

```
texture<unsigned char, 3, cudaReadModeNormalizedFloat> texDataSet;
cudaArray *d_volumeArray = 0;

__global__ void kernelGetValue(float* val, float x, float y, float z)
{
    //call the tex3D function with the global texture as a parameter
    *val = tex3D(texDataSet, x, y, z);
}
void cudaUploadDataSet(unsigned char* h_Volume, int sx, int sy, int sz)
{
    //upload the data set to the GPU as a 3D texture

    //create a cudaExtent structure, storing the dimensions of the 3D texture
    cudaExtent volumeSize = make_cudaExtent(sx, sy, sz);

    //create a channel format descriptor--defines the format of the texture
    //this code creates a descriptor for a single-channel unsigned char texture
    cudaChannelFormatDesc channelDesc = cudaCreateChannelDesc<unsigned char>();

    //allocate a 3D cuda array
    HANDLE_ERROR( cudaMalloc3DArray(&d_volumeArray, &channelDesc, volumeSize) );

    /*create a cudaMemcpy3DParms structures--this is a structure that tells cuda how
    to copy data using the cudaMemcpy3D function. Basically, this prevents having to
    pass a bunch of parameters in favor of a single complex structure.*/
    cudaMemcpy3DParms copyParms={0};
    copyParms.srcPtr = make_cudaPitchedPtr((void*)h_Volume,
                                           volumeSize.width*sizeof(unsigned char),
                                           volumeSize.width, volumeSize.height);
    copyParms.dstArray = d_volumeArray;
    copyParms.extent = volumeSize;
    copyParms.kind = cudaMemcpyHostToDevice;
    cudaMemcpy3D(&copyParms);

    //set the parameters for the global texture variable
    texDataSet.normalized = true;
    texDataSet.filterMode = cudaFilterModeLinear;
    texDataSet.addressMode[0] = cudaAddressModeClamp;
    texDataSet.addressMode[1] = cudaAddressModeClamp;

    //bind the texture to the array
    HANDLE_ERROR( cudaBindTextureToArray(texDataSet, d_volumeArray, channelDesc) );

    //output the value in the center (0.5, 0.5, 0.5) using a CUDA kernel
    float* d_val;
    cudaMalloc(&d_val, sizeof(float));
    kernelGetValue<<<1, 1>>>(d_val, 0.5, 0.5, 0.5);
    float h_val;
    cudaMemcpy(&h_val, d_val, sizeof(float), cudaMemcpyDeviceToHost);
    printf("The returned value is: %f\n", h_val);
}
```

### 3) How do I deal with “exit() redefinition” errors when using GLUT?

I have seen posts that say this has something to do with the inclusion of glut.h before stdlib.h but I haven't seen a correlation myself. I even get this without including stdlib.h. The fix that I use is to add “GLUT\_BUILDING\_LIB” to the “Project->Properties->C/C++->Preprocessor->Preprocessor Definitions” line in Visual Studio.

### 4) How do I map a 3D domain to CUDA's thread architecture?

While several CUDA enabled graphics cards support 3D thread blocks, the current lack of 3D grid support is inconvenient when dealing with 3D domains. I've found the following code is a good balance between complexity and performance. In particular, it allows the user to take advantage of 2D texture caching.

#### Host Code

```
void cudaHostFunction()
{
    //Input: Sx, Sy, Sz is the domain size along each dimension

    //create and allocate pointers, etc...

    //compute the block and grid sizes
    //Feel free to try alternative block sizes, as performance will vary with GPU
    dim3 block(16, 16);
    dim3 grid(Sx/block.x + 1, (Sy/block.y + 1)* Sz);

    //Call the kernel
    cudaKernelFunction<<<grid, block>>>(param1, param2, Sx, Sy, Sz);
}
```

#### Kernel Code

```
void cudaKernelFunction(int p1, int p2, int Sx, int Sy, int Sz)
{
    int x = blockIdx.x * blockDim.x + threadIdx.x;
    int blocks_per_slice = Sy/blockDim.y + 1;
    int y = (blockIdx.y % blocks_per_slice)*blockDim.y + threadIdx.y;
    int z = blockIdx.y/blocks_per_slice;

    //Make sure that the current thread is in the domain
    //Threads can be outside when Sx and Sy are not evenly divisible by the block size
    if(x >= Sx || y >= Sy || z >= Sz)
        return;

    //Kernel code based on x, y, z...
}
```