Documentation for CUDA GVF 3D

Quan Wang, Yu Wang
ECSE, Rensselaer Polytechnic Institute, Troy, NY, USA
November 2010

1 Code Description

We have two source files for this project, *GVF3D.cxx* and *CUDAgvf3D.cu*. The *GVF3D.cxx* file is the ITK module and the *CUDAgvf3D.cu* file is the CUDA module. Besides, we have a *CMakeLists.txt* file to build the ITK project with CMake.

The ITK module allocates four memory sections of the image size.

```
    float *f = (float *) malloc(imageSize);
    float *u = (float *) malloc(imageSize);
    float *v = (float *) malloc(imageSize);
    float *o = (float *) malloc(imageSize);
```

The edge map image is loaded to the memory section *f, and the memory sections *u, *v and *o are for the GVF vector field. The *CUDAgvf3D* function is declared in the ITK module as external C function.

```
extern "C" void CUDAgvf3D(float *f, float *u, float *v, float *o, int height, int
width, int frames, float mu, int iteration, float minValue, float range);
```

The *CUDAgvf3D* function is called by the ITK module and implemented in the CUDA module. The arguments of this function are defined as following.

Table 1.1 Argument description of CUDAgvf3D function.

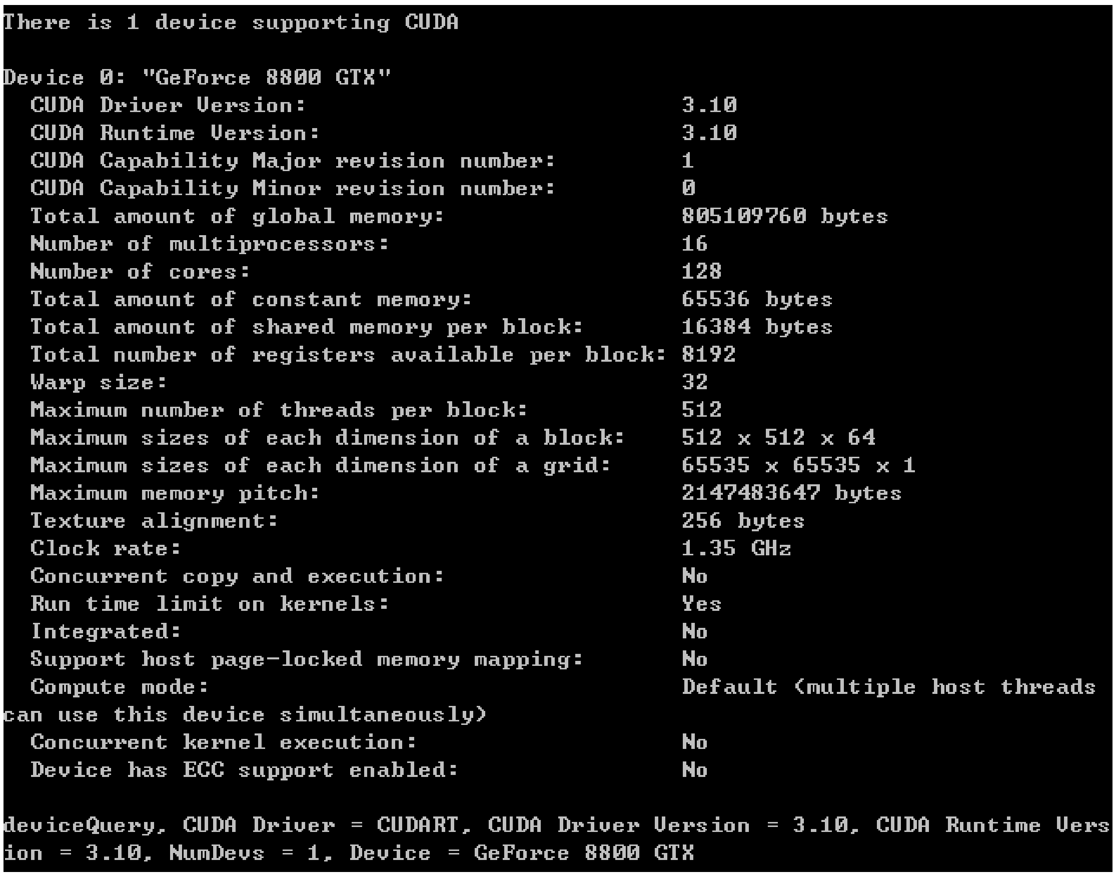
Name of Argument	Description
*f	The address of the memory section where the edge map image is loaded.
*u	The address of the memory section where the *x* direction GVF force field is saved.
*v	The address of the memory section where the *y* direction GVF force field is saved.
*o	The address of the memory section where the *z* direction GVF force field is saved.
height	The height of the edge map image. In ITK, it is the second dimension of the image size type.
width	The width of the edge map image. In ITK, it is the first dimension of the image size type.

frames	The number of frames of the edge map image. In ITK, it is the third dimension of the image size type.
mu	The regularization parameter of the GVF iterative algorithm.
iteration	The number of iterations of the GVF iterative algorithm.
minValue	The minimum intensity of the edge map image.
range	The range of the intensity of the edge map image. (The maximum intensity minus the minimum intensity)

In the CUDA module, there are some parameters that we should pay attention to.

```
long gridWidth=32768;
long gridHeight=(width*height*frames+gridWidth-1)/gridWidth;
dim3 dimBlock(32,16);
dim3 dimGrid((gridWidth+32-1)/32,(gridHeight+16-1)/16);
```

The dimension of a block and grid in the GPU is limited by the GPU we use. In our work, we use a NVIDIA GeForce 8800 GTX GPU, and the properties are as following.



```
There is 1 device supporting CUDA

Device 0: "GeForce 8800 GTX"
  CUDA Driver Version:                           3.10
  CUDA Runtime Version:                          3.10
  CUDA Capability Major revision number:         1
  CUDA Capability Minor revision number:         0
  Total amount of global memory:                 805109760 bytes
  Number of multiprocessors:                     16
  Number of cores:                               128
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       16384 bytes
  Total number of registers available per block: 8192
  Warp size:                                     32
  Maximum number of threads per block:           512
  Maximum sizes of each dimension of a block:    512 x 512 x 64
  Maximum sizes of each dimension of a grid:     65535 x 65535 x 1
  Maximum memory pitch:                          2147483647 bytes
  Texture alignment:                             256 bytes
  Clock rate:                                    1.35 GHz
  Concurrent copy and execution:                 No
  Run time limit on kernels:                     Yes
  Integrated:                                    No
  Support host page-locked memory mapping:       No
  Compute mode:                                  Default (multiple host threads
can use this device simultaneously)
  Concurrent kernel execution:                   No
  Device has ECC support enabled:                No

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 3.10, CUDA Runtime Vers
ion = 3.10, NumDevs = 1, Device = GeForce 8800 GTX
```

Figure 1.1 GPU properties in this project.

2 Requirements

2.1 Hardware Requirements

A CUDA-enabled GPU is required to run this project, which can be found on the NVIDIA CUDA Web site at http://www.nvidia.com/object/cuda_gpus.html.

2.2 Software Requirements

To use the ITK libraries, ITK toolkit should have been installed on the computer. Visit http://www.itk.org/.

To use CUDA, the following should have been installed:
(1) The CUDA Driver, which can be found at http://www.nvidia.com/drivers.
(2) The CUDA Toolkit, which contains the tools needed to compile and build a CUDA application in conjunction with Microsoft Visual Studio.
(3) The GPU Computing SDK.

The CUDA Toolkit and the GPU Computing SDK can be found at http://developer.nvidia.com/object/cuda_3_2_downloads.html.

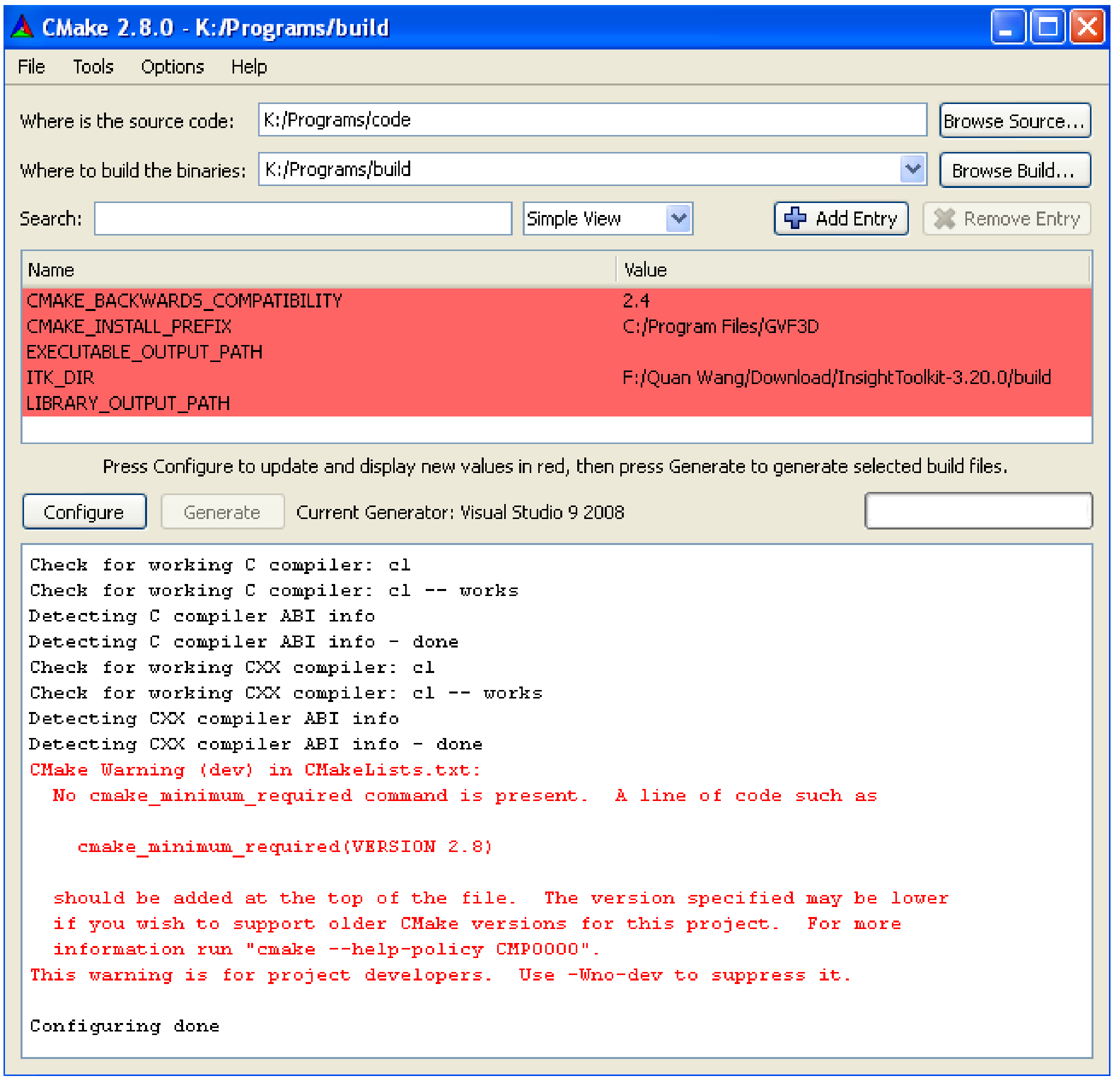
To configure a project in Windows, these softwares are required:
(1) CMake, which is used to build the project. It can be found at http://www.cmake.org/cmake/resources/software.html.
(2) Microsoft Visual Studio.

To view 3D images and GVF force field, ImageJ and Paraview might be used.

3 Project Configuration for Windows

3.1 Build Project for ITK module

If ITK toolkit and CMake have already been installed, we can build a Microsoft Visual Studio project for the ITK module. First we create a new folder, and copy the *GVF3D.cxx* file and the *CMakeLists.txt* file to that folder. Then we run CMake, and set the source code location as this folder. Set the binaries location, and click configure button. You should be asked to specify what generator you will use for this project. Click the configure button again and click the generate button, and the project will be generated. For example, if you are using Microsoft Visual Studio 2005 or 2008, a project file named *GVF3D.sln* will be generated.



Where is the source code: K:/Programs/code

Where to build the binaries: K:/Programs/build

Name	Value
CMAKE_BACKWARDS_COMPATIBILITY	2.4
CMAKE_INSTALL_PREFIX	C:/Program Files/GVF3D
EXECUTABLE_OUTPUT_PATH	
ITK_DIR	F:/Quan Wang/Download/InsightToolkit-3.20.0/build
LIBRARY_OUTPUT_PATH	

Press Configure to update and display new values in red, then press Generate to generate selected build files.

Current Generator: Visual Studio 9 2008

```
Check for working C compiler: cl
Check for working C compiler: cl -- works
Detecting C compiler ABI info
Detecting C compiler ABI info - done
Check for working CXX compiler: cl
Check for working CXX compiler: cl -- works
Detecting CXX compiler ABI info
Detecting CXX compiler ABI info - done
CMake Warning (dev) in CMakeLists.txt:
  No cmake_minimum_required command is present.  A line of code such as

    cmake_minimum_required(VERSION 2.8)

  should be added at the top of the file.  The version specified may be lower
  if you wish to support older CMake versions for this project.  For more
  information run "cmake --help-policy CMP0000".
This warning is for project developers.  Use -Wno-dev to suppress it.

Configuring done
```

Figure 3.1.1 CMake interface.

3.2 Add CUDA File to Project

Assume that we have opened the ITK project with Microsoft Visual Studio. First, we copy the *CUDAgvf3D.cu* file to the folder where the project has been built. Then we add this file as a source file to this project. There are several ways to configure a *.cu* file in the project. We will introduce two ways, using the CUDA rules from SDK and manually modifying the properties.

To use the CUDA rules of SDK, first right click on the GVF3D project, and select "Custom Build Rules". Then click on "New Rule File", add the *SDK/C/common/Cuda.rules* file to the rule files, and enable this rule. Now we right click on the *.cu* file and select "Properties". Set *Configuration Properties - General -Tool* as the CUDA Build Rule.

Another way is to modify the properties manually. Right click on the *.cu* file and select "Properties". Set *Configuration Properties - Custom Build Step - General -*

Command Line as

```
“$(CUDA_BIN_PATH)\nvcc.exe” -ccbin “$(VCInstallDir)bin” -c -D_DEBUG -DWIN32 -D_CONSOLE -D_MBCS -Xcompiler /EHsc,/W3,/nologo,/Wp64,/Od,/Zi,/MTd -I”$(CUDA_INC_PATH)” -I./ -o $(ConfigurationName)\CUDAgvf3D.obj CUDAgvf3D.cu
```

Set *Configuration Properties* - *Custom Build Step* - *General* - *Outputs* as

```
$(ConfigurationName)\CUDAgvf3D.obj
```

3.3 Set Project Properties

Right click on the GVF3D project and select "Properties". Set *Configuration Properties* - *C/C++* - *General* - *Additional Include Directories* as

```
$(CUDA_INC_PATH);”C:\Program Files\NVIDIA Corporation\NVIDIA CUDA SDK\common\inc”
```

Add “C:\CUDA\lib” and ”C:\Program Files\NVIDIA Corporation\NVIDIA CUDA SDK\common\lib” to *Configuration Properties* - *Linker* - *General* - *Additional Library Directories*. Add cudart.lib and cutil32D.lib to *Configuration Properties* - *Linker* - *Input* - *Additional Dependencies*. And set *Configuration Properties* - *Linker* - *Optimization* - *Enable COMDAT folding* as

```
Do Not Remove Redundant COMDATs (/OPT:NOICF)
```

Now build the solution, and we should have the executable file *GVF3D.exe* generated in the Debug folder or Release folder.

If the building process ends with library conflict errors, right click on the GVF3D project and select "Properties", then add the conflicting libraries to *Configuration Properties* - *Linker* - *Input* - *Ignore Specific Library*. For example, the *libcmtd.lib* might conflict with some ITK or CUDA libraries.

4 Running the Programs

There should be at least one and at most three input arguments for the *GVF3D.exe* main function. The order of arguments is the file name of edge map image, the regularization parameter μ, and the number of iterations. If we only input two arguments (file name and μ), the default number of iterations is 50. If we only input one argument (file name), the default regularization parameter μ is 0.2. For example, if we want to compute the GVF force field of the 3D edge map image *01.tif* with μ=0.2 for 50 iterations, we can type the command line in these three ways:

```
GVF3D   01.tif
GVF3D   01.tif   0.2
GVF3D   01.tif   0.2   50
```

After running the GVF3D program, six files will be generated, *u.mhd*, *v.mhd*, *o.mhd*, *u.raw*, *v.raw*, and *o.raw*. The *.mhd* files are the image information files, and the *.raw* files are the raw data.

The program will also record and display the time needed for the CUDA module, covering the normalization of edge map image, initialization, and iterative algorithm.

If the image size is too large, there might be not enough device memory on the GPU to perform the parallel computing. The program will display memory allocation status before the iterative algorithm. If the memory allocation fails, the program will automatically terminate, implying that the GPU memory is not enough for the input image, and a GPU with larger memory should be used.