```
////////////////////////////////////////////////////////////////////////////
//    File:        SimpleSIFT.cpp
//    Author:      Changchang Wu
//    Description : A simple example shows how to use SiftGPU and SiftMatchGPU
//
//
//
//    Copyright (c) 2007 University of North Carolina at Chapel Hill
//    All Rights Reserved
//
//    Permission to use, copy, modify and distribute this software and its
//    documentation for educational, research and non-profit purposes, without
//    fee, and without a written agreement is hereby granted, provided that the
//    above copyright notice and the following paragraph appear in all copies.
//
//    The University of North Carolina at Chapel Hill make no representations
//    about the suitability of this software for any purpose. It is provided
//    'as is' without express or implied warranty.
//
//    Please send BUG REPORTS to ccwu@cs.unc.edu
//
////////////////////////////////////////////////////////////////////////////


#include <stdlib.h>
#include <vector>
#include <iostream>
using std::vector;
using std::iostream;


////////////////////////////////////////////////////////////////////////////
#if !defined(SIFTGPU_STATIC) && !defined(SIFTGPU_DLL_RUNTIME)
// SIFTGPU_STATIC comes from compiler
#define SIFTGPU_DLL_RUNTIME
// Load at runtime if the above macro defined
// comment the macro above to use static linking
#endif

////////////////////////////////////////////////////////////////////////////
// define REMOTE_SIFTGPU to run computation in multi-process (Or remote) mode
// in order to run on a remote machine, you need to start the server manually
// This mode allows you use Multi-GPUs by creating multiple servers
// #define REMOTE_SIFTGPU
// #define REMOTE_SERVER        NULL
// #define REMOTE_SERVER_PORT   7777


////////////////////////////////////////////////////////////////////////////
//#define DEBUG_SIFTGPU  //define this to use the debug version in windows

#ifdef _WIN32
    #ifdef SIFTGPU_DLL_RUNTIME
        #define WIN32_LEAN_AND_MEAN
```

```
        #include <windows.h>
        #define FREE_MYLIB FreeLibrary
        #define GET_MYPROC GetProcAddress
    #else
        //define this to get dll import definition for win32
        #define SIFTGPU_DLL
        #ifdef _DEBUG
            #pragma comment(lib, "../../lib/siftgpu_d.lib")
        #else
            #pragma comment(lib, "../../lib/siftgpu.lib")
        #endif
    #endif
#else
    #ifdef SIFTGPU_DLL_RUNTIME
        #include <dlfcn.h>
        #define FREE_MYLIB dlclose
        #define GET_MYPROC dlsym
    #endif
#endif

#include "../SiftGPU/SiftGPU.h"


int main()
{
#ifdef SIFTGPU_DLL_RUNTIME
    #ifdef _WIN32
        #ifdef _DEBUG
            HMODULE  hsiftgpu = LoadLibrary("siftgpu_d.dll");
        #else
            HMODULE  hsiftgpu = LoadLibrary("siftgpu.dll");
        #endif
    #else
        void * hsiftgpu = dlopen("libsiftgpu.so", RTLD_LAZY);
    #endif

    if(hsiftgpu == NULL) return 0;

    #ifdef REMOTE_SIFTGPU
        ComboSiftGPU* (*pCreateRemoteSiftGPU) (int, char*) = NULL;
        pCreateRemoteSiftGPU = (ComboSiftGPU* (*) (int, char*)) GET_MYPROC(hsiftgpu,
"CreateRemoteSiftGPU");
        ComboSiftGPU * combo = pCreateRemoteSiftGPU(REMOTE_SERVER_PORT, REMOTE_SERVER);
        SiftGPU* sift = combo;
        SiftMatchGPU* matcher = combo;
    #else
        SiftGPU* (*pCreateNewSiftGPU)(int) = NULL;
        SiftMatchGPU* (*pCreateNewSiftMatchGPU)(int) = NULL;
        pCreateNewSiftGPU = (SiftGPU* (*) (int)) GET_MYPROC(hsiftgpu, "CreateNewSiftGPU");
        pCreateNewSiftMatchGPU = (SiftMatchGPU* (*)(int)) GET_MYPROC(hsiftgpu,
"CreateNewSiftMatchGPU");
        SiftGPU* sift = pCreateNewSiftGPU(1);
        SiftMatchGPU* matcher = pCreateNewSiftMatchGPU(4096);
    #endif
```

```
#elif defined(REMOTE_SIFTGPU)
    ComboSiftGPU * combo = CreateRemoteSiftGPU(REMOTE_SERVER_PORT, REMOTE_SERVER);
    SiftGPU* sift = combo;
    SiftMatchGPU* matcher = combo;
#else
    //this will use overloaded new operators
    SiftGPU  *sift = new SiftGPU;
    SiftMatchGPU *matcher = new SiftMatchGPU(4096);
#endif
    vector<float > descriptors1(1), descriptors2(1);
    vector<SiftGPU::SiftKeypoint> keys1(1), keys2(1);
    int num1, num2;

    //process parameters
    //The following parameters are default in V340
    //-m,       up to 2 orientations for each feature (change to single orientation by
using -m 1)
    //-s        enable subpixel subscale (disable by using -s 0)


    char * argv[] = {"-fo", "-1",  "-v", "1"};//
    //-fo -1    staring from -1 octave
    //-v 1      only print out # feature and overall time
    //-loweo    add a (.5, .5) offset
    //-tc <num> set a soft limit to number of detected features

    //NEW:  parameters for  GPU-selection
    //1. CUDA.                   Use parameter "-cuda", "[device_id]"
    //2. OpenGL.                                 Use "-Display", "display_name" to select
monitor/GPU (XLIB/GLUT)
        //                              on windows the display name would be something
like \\.\DISPLAY4

    ////////////////////////////////////////////////////////////////////////////
    //You use CUDA for nVidia graphic cards by specifying
    //-cuda   : cuda implementation (fastest for smaller images)
    //          CUDA-implementation allows you to create multiple instances for multiple
threads
        //          Checkout src\TestWin\MultiThreadSIFT
    ////////////////////////////////////////////////////////////////////////////

    ////////////////////////////////////////////////////////////////////////////
    //////////////////////Two Important Parameters//////////////////////////
    // First, texture reallocation happens when image size increases, and too many
    // reallocation may lead to allocatoin failure.  You should be careful when using
    // siftgpu on a set of images with VARYING imag sizes. It is recommended that you
    // preset the allocation size to the largest width and largest height by using
function
    // AllocationPyramid or prameter '-p' (e.g. "-p", "1024x768").

    // Second, there is a parameter you may not be aware of: the allowed maximum working
    // dimension. All the SIFT octaves that needs a larger texture size will be skipped.
    // The default prameter is 2560 for the unpacked implementation and 3200 for the
packed.
    // Those two default parameter is tuned to for 768MB of graphic memory. You should
```

```
adjust
    // it for your own GPU memory. You can also use this to keep/skip the small featuers.
    // To change this, call function SetMaxDimension or use parameter "-maxd".
        //
        // NEW: by default SiftGPU will try to fit the cap of GPU memory, and reduce the
working
        // dimension so as to not allocate too much. This feature can be disabled by -nomc
    ////////////////////////////////////////////////////////////////////////////


    int argc = sizeof(argv)/sizeof(char*);
    sift->ParseParam(argc, argv);

    ////////////////////////////////////////////////////////////////////
    //Only the following parameters can be changed after initialization (by calling
ParseParam).
    //-dw, -ofix, -ofix-not, -fo, -unn, -maxd, -b
    //to change other parameters at runtime, you need to first unload the dynamically
loaded libaray
    //reload the libarary, then create a new siftgpu instance


    //Create a context for computation, and SiftGPU will be initialized automatically
    //The same context can be used by SiftMatchGPU
    if(sift->CreateContextGL() != SiftGPU::SIFTGPU_FULL_SUPPORTED) return 0;

    if(sift->RunSIFT("../data/800-1.jpg"))
    {
        //Call SaveSIFT to save result to file, the format is the same as Lowe's
        //sift->SaveSIFT("../data/800-1.sift"); //Note that saving ASCII format is slow

        //get feature count
        num1 = sift->GetFeatureNum();

        //allocate memory
        keys1.resize(num1);    descriptors1.resize(128*num1);

        //reading back feature vectors is faster than writing files
        //if you dont need keys or descriptors, just put NULLs here
        sift->GetFeatureVector(&keys1[0], &descriptors1[0]);
        //this can be used to write your own sift file.
    }

    //You can have at most one OpenGL-based SiftGPU (per process).
    //Normally, you should just create one, and reuse on all images.
    if(sift->RunSIFT("../data/640-1.jpg"))
    {
        num2 = sift->GetFeatureNum();
        keys2.resize(num2);    descriptors2.resize(128*num2);
        sift->GetFeatureVector(&keys2[0], &descriptors2[0]);
    }

    //Testing code to check how it works when image size varies
    //sift->RunSIFT("../data/256.jpg");sift->SaveSIFT("../data/256.sift.1");
    //sift->RunSIFT("../data/1024.jpg"); //this will result in pyramid reallocation
```

```
    //sift->RunSIFT("../data/256.jpg"); sift->SaveSIFT("../data/256.sift.2");
    //two sets of features for 256.jpg may have different order due to implementation

    //////////////////////////////////////////////////////////////////////////
    /////compute descriptors for user-specified keypoints (with or without orientations)

    //Method1, set new keypoints for the image you've just processed with siftgpu
    //say vector<SiftGPU::SiftKeypoint> mykeys;
    //sift->RunSIFT(mykeys.size(), &mykeys[0]);
    //sift->RunSIFT(num2, &keys2[0], 1);         sift->SaveSIFT("../data/640-1.sift.2");
    //sift->RunSIFT(num2, &keys2[0], 0);        sift->SaveSIFT("../data/640-1.sift.3");

    //Method2, set keypoints for the next coming image
    //The difference of with method 1 is that method 1 skips gaussian filtering
    //SiftGPU::SiftKeypoint mykeys[100];
    //for(int i = 0; i < 100; ++i){
    //    mykeys[i].s = 1.0f;mykeys[i].o = 0.0f;
    //    mykeys[i].x = (i%10)*10.0f+50.0f;
    //    mykeys[i].y = (i/10)*10.0f+50.0f;
    //}
    //sift->SetKeypointList(100, mykeys, 0);
    //sift->RunSIFT("../data/800-1.jpg");                    sift->SaveSIFT("../data
/800-1.sift.2");
    //### for comparing with method1:
    //sift->RunSIFT("../data/800-1.jpg");
    //sift->RunSIFT(100, mykeys, 0);                          sift->SaveSIFT("../data
/800-1.sift.3");
    //////////////////////////////////////////////////////////////////////////////////


    //**********************GPU SIFT MATCHING*********************************
    //****************************select shader language************************
    //SiftMatchGPU will use the same shader lanaguage as SiftGPU by default
    //Before initialization, you can choose between glsl, and CUDA(if compiled).
    //matcher->SetLanguage(SiftMatchGPU::SIFTMATCH_CUDA); // +i for the (i+1)-th device

    //Verify current OpenGL Context and initialize the Matcher;
    //If you don't have an OpenGL Context, call matcher->CreateContextGL instead;
    matcher->VerifyContextGL(); //must call once

    //Set descriptors to match, the first argument must be either 0 or 1
    //if you want to use more than 4096 or less than 4096
    //call matcher->SetMaxSift() to change the limit before calling setdescriptor
    matcher->SetDescriptors(0, num1, &descriptors1[0]); //image 1
    matcher->SetDescriptors(1, num2, &descriptors2[0]); //image 2

    //match and get result.
    int (*match_buf)[2] = new int[num1][2];
    //use the default thresholds. Check the declaration in SiftGPU.h
    int num_match = matcher->GetSiftMatch(num1, match_buf);
    std::cout << num_match << " sift matches were found;\n";

    //enumerate all the feature matches
    for(int i  = 0; i < num_match; ++i)
    {
```

```
        //How to get the feature matches:
        //SiftGPU::SiftKeypoint & key1 = keys1[match_buf[i][0]];
        //SiftGPU::SiftKeypoint & key2 = keys2[match_buf[i][1]];
        //key1 in the first image matches with key2 in the second image
    }

    //*****************GPU Guided SIFT MATCHING***************
    //example: define a homography, and use default threshold 32 to search in a 64x64
window
    //float h[3][3] = {{0.8f, 0, 0}, {0, 0.8f, 0}, {0, 0, 1.0f}};
    //matcher->SetFeatureLocation(0, &keys1[0]); //SetFeatureLocaiton after SetDescriptors
    //matcher->SetFeatureLocation(1, &keys2[0]);
    //num_match = matcher->GetGuidedSiftMatch(num1, match_buf, h, NULL);
    //std::cout << num_match << " guided sift matches were found;\n";
    //if you can want to use a Fundamental matrix, check the function definition

    // clean up..
    delete[] match_buf;
#ifdef REMOTE_SIFTGPU
    delete combo;
#else
    delete sift;
    delete matcher;
#endif

#ifdef SIFTGPU_DLL_RUNTIME
    FREE_MYLIB(hsiftgpu);
#endif
    return 1;
}
```