半闲居士

SLAM拾萃(3):siftGPU

前言

本周博客我们给大家介绍一下SiftGPU。由于特征匹配是SLAM中非常耗时间的一步，许多人都想把它的时间降至最短，因此目前ORB成了非常受欢迎的特征。而老牌SIFT，则一直给人一种"很严谨很精确，但计算非常慢"的印象。在一个普通的PC上，计算一个640×480的图中的SIFT大约需要几百毫秒左右。如果特征都要算300ms，加上别的ICP什么的，一个SLAM就只能做成两帧左右的速度了，这是很令人失望的。而ORB，FAST之类的特征，由于计算速度较快，在SLAM这种实时性要求较高的场合更受欢迎。

那么，今天我们来说一个GPU版本的SIFT。它是由Wu Changchang同学写的。它能够明显地提升你的程序提取SIFT的速度。同时，它的代码大部分是基于OpenGL的，即使在没有英伟达显卡的机器上也能运行起来。但另一方面，出于某种（历史或人为的）原因，SiftGPU的代码配置起来并不很容易（特别是在Linux下，似乎SiftGPU作者是在win下开发的），代码新人可能会觉得比较困难。现在我们带着大家实践一下SiftGPU，我会给出一个例程供大家测试。

首先，说说我的运行配置。我用的机器是Thinkpad T450，Intel+Nvidia GetForce 940m显卡。但我个人只用Intel卡，所以我就不编译Cuda了。各位有上好N卡的同学也可以搞个Cuda下来编，可能会提高一点速度（但我不保证）。我使用的操作系统是Ubuntu 14.04，OpenCV3.1版本。所以我假设你OpenCV已经装好啦！（所以c++编译器总有的吧！）不过opencv是不是3.1版本是没关系的，程序在2.x版本上也是能正常运行的。

小萝卜：师兄你这真是宅男标配啊！你到底是在讲配置环境还是在秀桌面啊！

下载SiftGPU与依赖库

SiftGPU主页：http://www.cs.unc.edu/~ccwu/siftgpu/

请找到"SiftGPU-V400"那个下载链接，保存到你的电脑上。然后解压缩，进入压缩后的文件夹。假定你也在用Ubuntu，那么你现在的目录应该是~/Downloads/SiftGPU/ 。注意，为了和我保持一致，请你暂时不要下载github上面那个版本，那个与它稍有不同。如果你就是喜欢github，可以把这个编译好，再考虑用github版本。

现在我们来安装依赖项。首先，确保你机器上有OpenGL，请安装以下几项工具：

```
sudo apt-get install libgl1-mesa-dev libglu1-mesa-dev freeglut3-dev
```

然后，要安装glew1.5.1以上版本。据我个人经验，最好是去下载glew网站的版本。

glew的网址：http://glew.sourceforge.net/

请下载那个1.13.0版本，zip文件或tgz均可。下载到本地并解压，然后进入该文件夹。我的在~/Downloads/glew-1.13.0

glew是用makefile直接编译的，不用cmake。所以我们直接敲：

```
make
sudo make install
```

即可。很快它就编译好了。

注意看make install输出的信息。它默认把编译好的库文件libglew.so.1.13放到了/usr/lib64下。由于之后我们要用cmake去编，但是它可能找不到这个文件夹，所以我们现在先告诉系统，该文件夹下有要找的链接库：

```
sudo ldconfig /usr/lib64/
```

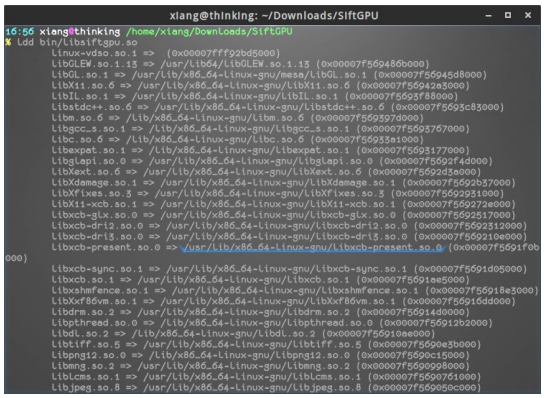
ok，现在我们处理完了glew，转去编译SiftGPU。SiftGPU也是用Makefile编译的。现在转到SiftGPU所在文件夹。调用

```
make
```

来完成编译。如果顺利的话，你会在bin/目录里得到几个二进制和一个libsiftgpu.so库文件。我们主要使用这个库文件。现在看一下它的链接是否正确：

```
ldd bin/libsiftgpu.so
```

这个命令会输出与它链接的库的信息。请保证没有出现某个链接（特别是刚才的GLEW）没有找到的情况（否则这里会通过，但后面会出现undefined reference）。像我这样：



如果这步正确无误，恭喜你，SiftGPU已经编译完成了！真是可喜可贺呀！

小萝卜：然后呢？师兄我还没看到什么感觉很厉害的东西啊？

师兄：下面我们来实际找一个图片，写一段小程序调用SiftGPU，提一下特征试试。为测试速度，我们还要记录一下代码运行时间。

测试SiftGPU

现在我们来写一个测试程序。由于它比较短，我就不专门搞个github了。请大家跟着我做即可。

首先，随意新建一个目录，比如test_siftgpu。我们要写一个c++程序，然后用cmake编译它。现在新建一个main.cpp，内容如下：

```
// SiftGPU模块
#include <SiftGPU.h>

//标准C++
#include <iostream>
#include <vector>

// OpenCV图像
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>

// boost库中计时函数
```

```cpp
#include <boost/timer.hpp>

// OpenGL
#include <GL/gl.h>

using namespace std;

int main( int argc, char** argv)
{
    //声明SiftGPU并初始化
    SiftGPU sift;
    char* myargv[4] ={ "-fo", "-1", "-v", "1"};
    sift.ParseParam(4, myargv);

    //检查硬件是否支持SiftGPU
    int support = sift.CreateContextGL();
    if ( support != SiftGPU::SIFTGPU_FULL_SUPPORTED )
    {
        cerr<<"SiftGPU is not supported!"<<endl;
        return 2;
    }

    //测试直接读取一张图像
    cout<<"running sift"<<endl;
    boost::timer timer;
    //在此填入你想测试的图像的路径！不要用我的路径！不要用我的路径！不要用我的路径！
    sift.RunSIFT( "/home/xiang/wallE-slam/data/rgb1.png" );
    cout<<"siftgpu::runsift() cost time="<<timer.elapsed()<<endl;

    // 获取关键点与描述子
    int num = sift.GetFeatureNum();
    cout<<"Feature number="<<num<<endl;
    vector<float> descriptors(128*num);
    vector<SiftGPU::SiftKeypoint> keys(num);
    timer.restart();
    sift.GetFeatureVector(&keys[0], &descriptors[0]);
    cout<<"siftgpu::getFeatureVector() cost time="<<timer.elapsed()<<endl;

    // 先用OpenCV读取一个图像，然后调用SiftGPU提取特征
    cv::Mat img = cv::imread("/home/xiang/wallE-slam/data/rgb1.png", 0);
    int width = img.cols;
    int height = img.rows;
    timer.restart();
    // 注意我们处理的是灰度图，故照如下设置
    sift.RunSIFT(width, height, img.data, GL_INTENSITY8, GL_UNSIGNED_BYTE);
    cout<<"siftgpu::runSIFT() cost time="<<timer.elapsed()<<endl;

    return 0;
}
```

Sift接口还是相当简单的。在这程序里，我们一共做了三件事。一是直接对一个图像路径提Sift，二是获取Sift的关键点和描述子。三是对OpenCV读取的一个图像提取Sift。我们分别测了三者的效果和时间。

接下来，写一个CMakeLists.txt来编译上面的文件。

```
cmake_minimum_required(VERSION 2.8.3)
project(test_siftgpu)

# OpenCV依赖
find_package( OpenCV REQUIRED )

# OpenGL
find_package(OpenGL REQUIRED)

# GLUT
find_package(GLUT REQUIRED)

# Glew
find_package(Glew REQUIRED)

# SiftGPU:手动设置其头文件与库文件所在位置
include_directories("/home/xiang/Downloads/SiftGPU/src/SiftGPU/" ${OpenGL_INCLUDE_DIR})
set(SIFTGPU_LIBS "/home/xiang/Downloads/SiftGPU/bin/libsiftgpu.so")

add_executable( testSIFTGPU main.cpp )

target_link_libraries( testSIFTGPU
    ${OpenCV_LIBS}
    ${SIFTGPU_LIBS}
    ${GLEW_LIBRARIES} ${GLUT_LIBRARIES} ${OPENGL_LIBRARIES}
)
```

对于SiftGPU，由于它本身没有提供cmake的配置，我们手动去设置了它的头文件与库文件的链接方式。大家可以学习一下这种比较土的办法……然后就是常见的cmake啦：

```
mkdir build
cd build
cmake ..
make
```

等一下！是不是还忘了些什么呢？嗯，如果你直接去cmake的话，会报一个find_package找不到glew的错！因为我们装glew的时候是直接用make install装的嘛，cmake怎么会知道我们干了这件事呢？所以此时find_package(Glew REQUIRED)就会出错啦！

小萝卜：为什么出错了你还是很高兴的样子……

师兄：对！现在呢我们要自己写一个FindGlew.cmake文件哦。请打开你的编辑器，输入：

```cpp
#
# Try to find GLEW library and include path.
# Once done this will define
#
# GLEW_FOUND
# GLEW_INCLUDE_PATH
# GLEW_LIBRARY
#

IF (WIN32)
    FIND_PATH( GLEW_INCLUDE_PATH GL/glew.h
        $ENV{PROGRAMFILES}/GLEW/include
        ${PROJECT_SOURCE_DIR}/src/nvgl/glew/include
        DOC "The directory where GL/glew.h resides")
    FIND_LIBRARY( GLEW_LIBRARY
        NAMES glew GLEW glew32 glew32s
        PATHS
        $ENV{PROGRAMFILES}/GLEW/lib
        ${PROJECT_SOURCE_DIR}/src/nvgl/glew/bin
        ${PROJECT_SOURCE_DIR}/src/nvgl/glew/lib
        DOC "The GLEW library")
ELSE (WIN32)
    FIND_PATH( GLEW_INCLUDE_PATH GL/glew.h
        /usr/include
        /usr/local/include
        /sw/include
        /opt/local/include
        DOC "The directory where GL/glew.h resides")
    FIND_LIBRARY( GLEW_LIBRARY
```

讲师吗，官网，了解更多可以添加微信GYH-xiaogu咨询。

--古月居

5. Re:一起做RGB-D SLAM (3)

请问为什么我的旋转矩阵和您的差一个负号呀

--Mai_Qi

```
        NAMES GLEW glew
        PATHS
        /usr/lib64
        /usr/lib
        /usr/local/lib64
        /usr/local/lib
        /sw/lib
        /opt/local/lib
        DOC "The GLEW library")
ENDIF (WIN32)

IF (GLEW_INCLUDE_PATH)
    SET( GLEW_FOUND 1 CACHE STRING "Set to 1 if GLEW is found, 0 otherwise")
ELSE (GLEW_INCLUDE_PATH)
    SET( GLEW_FOUND 0 CACHE STRING "Set to 1 if GLEW is found, 0 otherwise")
ENDIF (GLEW_INCLUDE_PATH)

MARK_AS_ADVANCED( GLEW_FOUND )
```

然后呢，把这个文件放到cmake的modules文件夹中去！这样cmake就会知道你在调用find_package(Glew)时怎么找啦！

```
sudo cp ./FindGlew.cmake /usr/share/cmake-2.8/Modules/
```

注意到这个文件所在的目录通常是没有写权限的的哦！所以我们要用sudo提升到管理员权限才行呢。

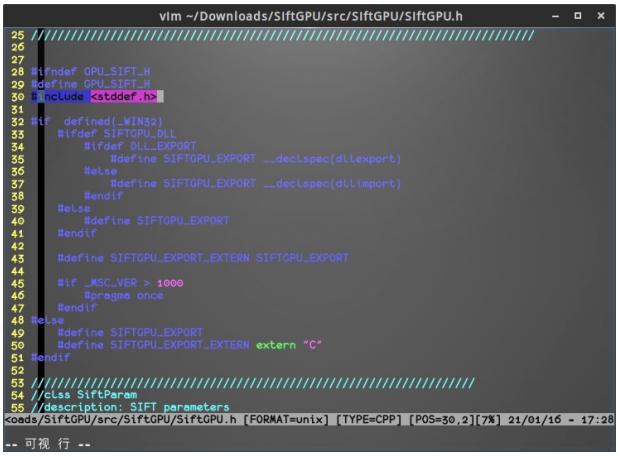
这时，再调用cmake ... 就不会报上面的错误啦！而编译也得以顺利进行下去了。

但是！但是！编译还是出错了，错误如下：

```
/home/xiang/Downloads/SiftGPU/src/SiftGPU/SiftGPU.h:336:40: error: declaration of ‘operator new’ as non-function SIFTGPU_EXPORT void* operator new (size_t size);
```

这是什么原因呢？g++的编译错误很难懂，一直为人诟病。师兄仔细查了查，发现SiftGPU作者重载了new运算符，但是它的参数"size_t size"中的"size_t"类型，在linux下编译是需要指定一个头文件的！所以我们打开~/Downloads/SiftGPU/src/SiftGPU/SiftGPU.h文件，在上头加入一个

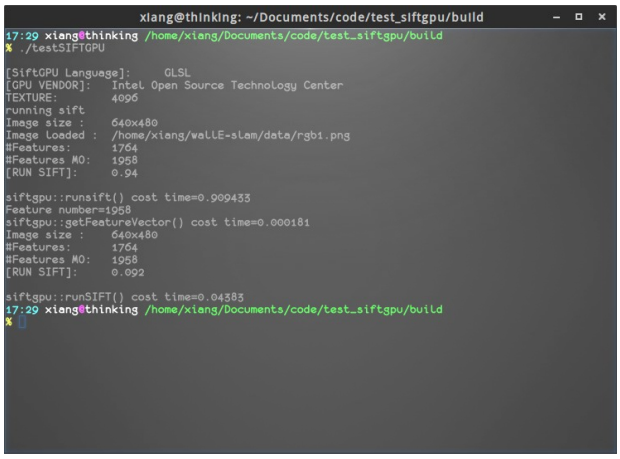
```
#include <stddef.h>
```



这样编译器就会找到size_t类型啦！编译就能通过喽！

SiftGPU运行结果

以下就是在师兄电脑上的运行结果啦，大家可以看一下：



对于OpenCV已经读入的数据，在640x480的分辨率下，用SiftGPU只需40多毫秒即可完成计算了呢！GPU真的是很强大啊！即使在没有Cuda的情况下都取得了近十倍的加速啊！效果拔群！

小萝卜：我的ORB只要30毫秒就行了，哼，

小结

本篇介绍了SiftGPU，我们带领读者完成了它的编译，并在自己的程序内实现了调用。可以看到它的加速效果还是不错的！

另外，这也是我的一次尝试，告诉读者在编译过程中遇到问题该如何处理。我本可以直接跳过这些buggy的部分，告诉大家运行的结果。但我觉得这样子讲可能对读者更有帮助啦！

标签： 视觉SLAM, Sift, SiftGPU


半闲居士
粉丝 - 3062 关注 - 0
+加关注

« 上一篇：视觉SLAM中的数学基础 第三篇 李群与李代数
» 下一篇：视觉SLAM实战(二):ORB-SLAM2 with Kinect2

posted @ 2016-01-21 17:37 半闲居士 阅读(24745) 评论(7) 编辑 收藏 举报