# 使用OpenCL处理OpenCV图像

## 几个问题

使用OpenCL的流程相同，不再赘述。这里着重说明容易忽略的问题。

使用Mat彩色图像，最好转成4通道图像。

使用二维的数据划分，虽然图像数据是连续的，对于大小为307\*413的图像，全局序号为(307, 0)的程没有某个像素与之对应，而不是处理第2行，第1个像素。

## 代码

### 2.1核函数functions.cl

|  |
| --- |
| \_\_kernel void rgb2gray(\_\_global uchar4\* srcImg, \_\_global uchar\* dstImg, int height, int width)  {  int2 point = (int2) (get\_global\_id(0), get\_global\_id(1));  if (point.x < width && point.y < height)  {  uchar4 pixel = srcImg[point.y \* width + point.x];  uchar grayval = 0.33 \* pixel.x + 0.33 \* pixel.y + 0.33 \* pixel.z;  dstImg[point.y \* width + point.x] = grayval;  }  } |

### 2.2 OpenclHelper.h

|  |
| --- |
| #include <stdio.h>  #include <random>  #include <vector>  using namespace std;  cl::Program CreateProgram(std::string filename, int platform\_idx, cl\_int err)  {  vector<cl::Platform> platforms;  err = cl::Platform::get(&platforms);  auto platform = platforms[platform\_idx];  vector<cl::Device> devices;  err|= platform.getDevices(CL\_DEVICE\_TYPE\_GPU, &devices);  cl::Device device = devices.front();  cl::Context context(devices);  ifstream file(filename.c\_str());  std::string src(istreambuf\_iterator<char>(file), (istreambuf\_iterator<char>()));  cl::Program::Sources sources(1, std::make\_pair(src.c\_str(), src.length() + 1));  cl::Program program(context, sources);  err|= program.build("-cl-std=CL1.2");  return program;  } |

### 2.3 opencv\_cl.cpp

|  |
| --- |
| #include "stdafx.h"  #include "OpenclHelper.h"  #include <opencv2\opencv.hpp>  #include <CL\cl.hpp>  using namespace cv;  #define LOCAL\_SIZE 32  int main()  {  cl\_int err = 0;  //读取图像  Mat image = imread("lena2.png");  cvtColor(image, image, CV\_BGR2BGRA);  //创建程序  cl::Program program = CreateProgram("functions.cl", 1, err);  auto context = program.getInfo<CL\_PROGRAM\_CONTEXT>();  auto devices = context.getInfo<CL\_CONTEXT\_DEVICES>();  auto& device = devices.front();  //创建内核  cl::Kernel kernel(program, "rgb2gray");  //创建内存对象  int pix\_num = image.cols \* image.rows;  uchar \*res\_data = new uchar[pix\_num];  memset(res\_data, 0, pix\_num);  cl::Buffer inputBuf(context, CL\_MEM\_READ\_ONLY | CL\_MEM\_COPY\_HOST\_PTR, sizeof(uchar) \* pix\_num \* 4, (void\*)image.data);  cl::Buffer outputBuf(context, CL\_MEM\_WRITE\_ONLY | CL\_MEM\_USE\_HOST\_PTR, sizeof(uchar) \* pix\_num, (void\*)res\_data);  //设置内核参数  kernel.setArg(0, inputBuf);  kernel.setArg(1, outputBuf);  kernel.setArg(2, image.rows);  kernel.setArg(3, image.cols);  //创建queue  cl::CommandQueue queue(context,device);  int g\_x = (image.cols + LOCAL\_SIZE - 1) / LOCAL\_SIZE \* LOCAL\_SIZE;  int g\_y = (image.rows + LOCAL\_SIZE - 1) / LOCAL\_SIZE \* LOCAL\_SIZE;  int l\_x = LOCAL\_SIZE;  int l\_y = LOCAL\_SIZE;  err = queue.enqueueNDRangeKernel(kernel, NULL, cl::NDRange(g\_x, g\_y), cl::NDRange(l\_x, l\_y));  cl::finish();  //读取处理后的数据  queue.enqueueReadBuffer(outputBuf, CL\_TRUE, 0, sizeof(uchar) \* pix\_num, res\_data);  //显示处理结果  Mat out(Size(image.cols, image.rows), CV\_8UC1, res\_data);  imshow("input", image);  imshow("result", out);  waitKey(0);  return 0;  } |

 