HW3 - Ha Hwang and Wasfi Momen



main1.cpp, cuda1.h, and cuda1.cu make up the blurring of resolution for the levels of the image pyramid.



Attempted to do a sequential nearest-neighbor interpolation. Main function to execute this is in interpolate(). Check readme in "hw3\_part2\_sequential.zip" for more info.

**nearest\_neighbor.cpp**

|  |
| --- |
| #include <opencv2/core/mat.hpp>  #include <opencv2/highgui.hpp>  #include <iostream>  #include <fstream>  #include <string>  using namespace std;  using namespace cv;  // User defined variables.  const int SCALE\_FACTOR = 2;  const char\* FILENAME = "kodim02.jpg";  /\*  Do a nearest neighbor interpolation.    - Get the size of the new image with the scaling factor.  - calculate the scaling ratio for all pixels  - loop over all pixels, round pixel postion/scaling factor to get new position  \*/  void interpolate(Mat img)  {  // Get the size and type of the input image  int row = img.rows;  int col = img.cols \* img.channels();  int type = img.type();  // Get the size of the output image  int nrows = row \* SCALE\_FACTOR;  int ncol = col \* SCALE\_FACTOR;  // Allocate new image of kn x km size.  Mat output\_img;  output\_img = Mat::zeros(nrows, ncol, type);  // Get the scaling factors for each pixel.  // It will be the output image width/height divided by the old one.  float c\_X = nrows / row;  float c\_Y = ncol / col;  // Get the nearest neighboring pixel to assign to the output image.  int i, j; // output image indicies  float x, y; // input image indicies (float for approx position)  for (i = 0; i < nrows; ++i)  {  for (j = 0; j < ncol; ++j)  {  // Get the pixel positions.  // current pixel / scaling = new pixel to sample  x = i / c\_X;  y = j / c\_Y;  // x, y are floats, so we round to pick nearest neighbor.  x = round(x);  y = round(y);  // PROBLEM HERE. Pointer illegal access.  //output\_img.at<cv::Vec3b>(i, j) = img.at<cv::Vec3b>(x, y);    }  }  imwrite("output.jpg", output\_img);  }  // From StackOverflow, getting image type to make sure correct data type  // TODO: return struct with [string type, data type to scan with]  string type2str(int type) {  string r;  uchar depth = type & CV\_MAT\_DEPTH\_MASK;  uchar chans = 1 + (type >> CV\_CN\_SHIFT);  switch (depth) {  case CV\_8U: r = "8U"; break;  case CV\_8S: r = "8S"; break;  case CV\_16U: r = "16U"; break;  case CV\_16S: r = "16S"; break;  case CV\_32S: r = "32S"; break;  case CV\_32F: r = "32F"; break;  case CV\_64F: r = "64F"; break;  default: r = "User"; break;  }  r += "C";  r += (chans + '0');  return r;  }  int main()  {  Mat img = imread(FILENAME, 1); // uses opencv to parse in image. note the 1 flag means 3-channel BGR data. (actually BGR not RGB)  if (!img.data) // check for null image data  {  printf("No image data \n");  return -1; // exit with error  }  int rows = img.rows; // get the width of the image  int cols = img.cols \* img.channels(); // get the height of the image, BGR has 3 subcolumn channels  int type = img.type(); // get the type of image  string image\_type = type2str(type);  printf("\nWidth %d ", rows);  printf("\nHeight %d ", cols);  printf("\nMatrix Type: %s ", image\_type.c\_str());    interpolate(img);  // write out the image to the current directory  //imwrite("output.jpg", img);    // GUI stuff  namedWindow(FILENAME);  namedWindow("processed image (Press any key to exit)");  imshow(FILENAME, img); // original  imshow("processed image (Press any key to exit)", img); // processed  waitKey(0);  return 0;  } |