《时空大数据高性能处理》上机实验

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 章节 | **第四次上机：用OpenMP并行编程** | | | | | 序号 | 4 |
| 姓 名 | 甘劲博 | 系院专业 | 计算机 | 班级 | 21计科5 | 学 号 | 2110551202 |
| 日期 | 2024-10-17 | | 指导教师 | 刘舟 | | 成 绩 |  |
| **一、内容**  在理解掌握二维图形的仿射变换模型的基础上。思考给定若干个二维平面点对后（初始坐标-仿射变换坐标），拟合仿射变换模型，并根据该模型实现图像变换（提示：应用间接法双线性重采样）。要求采用OpenMP优化性能。 | | | | | | | |
| **2.1主要c/c++代码**  包含：用OpenMP并行编程实现图像变换  开源地址：<https://github.com/Duxingmengshou/HPCLab>  用OpenMP并行编程实现图像变换：  #include <opencv2/opencv.hpp>  #include <omp.h>  #include <iostream>    using namespace cv;  using namespace std;    // 计算旋转150度的仿射变换矩阵  Mat computeRotationMatrix150(Point2f center) {  double angle = 150.0; // 旋转角度  double radians = angle \* CV\_PI / 180.0; // 转换为弧度    // 计算旋转矩阵  Mat rotationMatrix = (Mat\_<double>(2, 3) <<  cos(radians), -sin(radians), center.x - center.x \* cos(radians) +  center.y \* sin(radians),  sin(radians), cos(radians), center.y - center.x \* sin(radians) - center.y \* cos(radians));    return rotationMatrix;  }    // 双线性插值  Vec3b bilinearInterpolation(const Mat &src, float x, float y) {  int x1 = floor(x);  int y1 = floor(y);  int x2 = ceil(x);  int y2 = ceil(y);    if (x1 < 0 || x2 >= src.cols || y1 < 0 || y2 >= src.rows) {  return Vec3b(0, 0, 0); // 边界处理  }    float a = x - x1;  float b = y - y1;    Vec3b I11 = src.at<Vec3b>(y1, x1);  Vec3b I12 = src.at<Vec3b>(y1, x2);  Vec3b I21 = src.at<Vec3b>(y2, x1);  Vec3b I22 = src.at<Vec3b>(y2, x2);    return (1 - a) \* (1 - b) \* I11 + a \* (1 - b) \* I12 + (1 - a) \* b \* I21 + a \* b \* I22;  }    // 图像变换  void applyAffineTransform(const Mat &src, Mat &dst, const Mat &affine) {  #pragma omp parallel for  for (int y = 0; y < dst.rows; ++y) {  for (int x = 0; x < dst.cols; ++x) {  // 应用仿射变换公式  float x\_src = affine.at<double>(0, 0) \* x + affine.at<double>(0, 1) \* y + affine.at<double>(0, 2);  float y\_src = affine.at<double>(1, 0) \* x + affine.at<double>(1, 1) \* y + affine.at<double>(1, 2);  // 使用双线性插值获取目标像素的颜色  dst.at<Vec3b>(y, x) = bilinearInterpolation(src, x\_src, y\_src);  }  }  }    int main() {  // 读取图像  Mat src = imread("E:\\\_project\\CLion\_project\\openmp-check\\input.jpg");  if (src.empty()) {  cout << "Could not read the image." << endl;  return 1;  }    // 计算旋转 150 度的仿射变换矩阵  Point2f center(src.cols / 2.0f, src.rows / 2.0f); // 旋转中心为图像中心  Mat affine = computeRotationMatrix150(center);    // 创建输出图像  Mat dst(src.rows, src.cols, src.type(), Scalar(0, 0, 0)); // 初始化为黑色背景    // 应用仿射变换  applyAffineTransform(src, dst, affine);    // 显示结果  namedWindow("Source Image", 0);  namedWindow("Transformed Image", 0);  imshow("Source Image", src);  imshow("Transformed Image", dst);  waitKey(0);  return 0;  }  CMakeLists.txt  cmake\_minimum\_required(VERSION 3.26)  project(openmp\_check)    set(CMAKE\_CXX\_STANDARD 17)    # OpenCV  set(OpenCV\_DIR "H:/Programming/C++/OpenCV/OpenCV490/opencv/build/x64/vc16/lib")  find\_package(OpenCV REQUIRED)  include\_directories(${OpenCV\_INCLUDE\_DIRS})    add\_executable(${PROJECT\_NAME} main.cpp)    target\_compile\_options(${PROJECT\_NAME}  PRIVATE  /openmp  )  target\_link\_libraries(${PROJECT\_NAME} ${OpenCV\_LIBS})  target\_link\_libraries(${PROJECT\_NAME} ${pthreads\_ROOT\_LIBRARIES}) | | | | | | | |
| **2.1运行截图**  用OpenMP并行编程实现图像变换：  OpenMP 的性能优化  并行化：通过将每个像素的变换操作并行化，OpenMP可以有效地利用多核处理器的能力，从而加速图像变换的过程。每个线程可以独立计算不同像素的变换，这样可以显著减少执行时间。  负载均衡：在处理大图像时，OpenMP 能够将任务均匀分配到各个线程，避免某些线程空闲而其他线程过载的情况。  减少计算时间：在图像变换过程中，计算每个像素的坐标和颜色是计算密集型的，使用 OpenMP 可以将这些计算分散到多个处理核心上，显著提高性能。 | | | | | | | |