



#include <iostream>

#include <opencv2/opencv.hpp>

#include <cmath>

#include<omp.h>

using namespace cv;

int **main**() {

    Mat inputImage = **imread**("input.jpg", IMREAD\_COLOR);

    if (inputImage.**empty**()) {

        std::cerr **<<** "Error: Could not open or find the image." **<<** std::**endl**;

        return -1;

    }

    int num;

    double start=**omp\_get\_wtime**();

    int kernelSize = 15;

    double sigma = 1;

    Mat **outputImage**(inputImage.**size()**, inputImage.**type**());

    Mat kernel = Mat::**zeros**(kernelSize, kernelSize, **CV\_64F**);

    double sum = 0.0;

*//#pragma omp parallel for reduction (+:sum)*

    for (int y = -kernelSize / 2; y <= kernelSize / 2; y++) {

        for (int x = -kernelSize / 2; x <= kernelSize / 2; x++) {

            double weight = **exp**(-(x \* x + y \* y) / (2.0 \* sigma \* sigma));

            kernel.**at**<double>(y + kernelSize / 2, x + kernelSize / 2) = weight;

            sum += weight;

        }

    }

    kernel **/=** sum;

*//#pragma omp parallel for*

    for (int y = 0; y < inputImage.rows; y++) {

        for (int x = 0; x < inputImage.cols; x++) {

            Vec3d **sum**(0, 0, 0);

            for (int ky = -kernelSize / 2; ky <= kernelSize / 2; ky++) {

                for (int kx = -kernelSize / 2; kx <= kernelSize / 2; kx++) {

                    int newX = x + kx;

                    int newY = y + ky;

                    if (newX >= 0 && newX < inputImage.cols && newY >= 0 && newY < inputImage.rows) {

                        Vec3b pixel = inputImage.**at**<Vec3b>(newY, newX);

                        double weight = kernel.**at**<double>(ky + kernelSize / 2, kx + kernelSize / 2);

                        sum**[**0**]** += pixel**[**0**]** \* weight;

                        sum**[**1**]** += pixel**[**1**]** \* weight;

                        sum**[**2**]** += pixel**[**2**]** \* weight;

                    }

                }

            }

            outputImage.**at**<Vec3b>(y, x) **=** **Vec3b**(sum**[**0**]**, sum**[**1**]**, sum**[**2**]**);

        }

    }

    double end=**omp\_get\_wtime**();

    double time[2];

    time[0]=end-start;

    std::cout**<<**"single thread using time:"**<<**time[0]**<<**std::**endl**;

    cv::**imwrite**("output.jpg", outputImage);

**printf**("Manual Gaussian blur completed.\n");

    start=**omp\_get\_wtime**();

    Mat **outputImage2**(inputImage.**size()**, inputImage.**type**());

    kernel **=** Mat::**zeros**(kernelSize, kernelSize, **CV\_64F**);

    sum = 0.0;

    #pragma omp parallel for reduction (+:sum)

    for (int y = -kernelSize / 2; y <= kernelSize / 2; y++) {

        for (int x = -kernelSize / 2; x <= kernelSize / 2; x++) {

            double weight = **exp**(-(x \* x + y \* y) / (2.0 \* sigma \* sigma));

            kernel.**at**<double>(y + kernelSize / 2, x + kernelSize / 2) = weight;

            sum += weight;

            num=**omp\_get\_num\_threads**();

        }

    }

    kernel **/=** sum;

    #pragma omp parallel for

    for (int y = 0; y < inputImage.rows; y++) {

        for (int x = 0; x < inputImage.cols; x++) {

            Vec3d **sum**(0, 0, 0);

            for (int ky = -kernelSize / 2; ky <= kernelSize / 2; ky++) {

                for (int kx = -kernelSize / 2; kx <= kernelSize / 2; kx++) {

                    int newX = x + kx;

                    int newY = y + ky;

                    if (newX >= 0 && newX < inputImage.cols && newY >= 0 && newY < inputImage.rows) {

                        Vec3b pixel = inputImage.**at**<Vec3b>(newY, newX);

                        double weight = kernel.**at**<double>(ky + kernelSize / 2, kx + kernelSize / 2);

                        sum**[**0**]** += pixel**[**0**]** \* weight;

                        sum**[**1**]** += pixel**[**1**]** \* weight;

                        sum**[**2**]** += pixel**[**2**]** \* weight;

                    }

                }

            }

            outputImage2.**at**<Vec3b>(y, x) **=** **Vec3b**(sum**[**0**]**, sum**[**1**]**, sum**[**2**]**);

        }

    }

    end=**omp\_get\_wtime**();

    time[1]=end-start;

    std::cout**<<**"multi thread using time:"**<<**time[1]**<<**std::**endl**;

    cv::**imwrite**("output.jpg", outputImage2);

**printf**("Manual Gaussian blur completed.\n");

    std::cout**<<**"Speedup:"**<<**time[0]/time[1]**<<**std::**endl**;

    std::cout**<<**"Parallel Efficiency:"**<<**time[0]/(time[1]\*num)**<<**std::**endl**;

    return 0;

}