**Lab 5: Chroma Key**

**2020 Spring**

**Multi-Media & Lab**

**Name/ID: 201533661 이승수**

**Dept: Software**

**[My Code]**

//#include <opencv2/core/core.hpp>

//#include <opencv2/imgcodecs.hpp>

//#include <opencv2/highgui/highgui.hpp>

#include <opencv2/opencv.hpp>

#include <iostream>

#include <string>

using namespace cv;

using namespace std;

Scalar bgr2ycrcb(Scalar bgr);

Mat1b chromaKey(const cv::Mat3b& imageBGR, cv::Scalar chromaBGR, double tInner, double tOuter);

Mat3b replaceBackground(const cv::Mat3b& image, const cv::Mat1b& mask, cv::Mat bgImage);

int main(int argc, char\*\* argv) {

//Lab5: Chroma Key

string src\_Path = "D:\\repos\_VS\\Project\_sourceIMG\\";//image source file path(folder)

//Load image

if (argc > 1) {

src\_Path = argv[1];

}

Mat src = imread((src\_Path + "Girl\_in\_front\_of\_a\_green\_background.jpg").c\_str(), IMREAD\_COLOR);

Mat backgroundImage = imread((src\_Path + "sky\_background.jpg").c\_str(), IMREAD\_COLOR); //Mat::zeros(src.size(), src.type());

if (src.empty() || backgroundImage.empty()) {

cout << "Could not open or find the image" << std::endl;

return -1;

}

Mat resized\_image2;

resize(backgroundImage, resized\_image2, Size(src.cols,src.rows));

//std::cout << "src image size: " << src.size << std::endl;

Scalar chroma(0, 255, 0, 0); //chroma key background color=green(BGR)

double tInner = 30;//hyper parameter

double tOuter = 100; //hyper parameter

std::cout << "tInner: " << tInner << ", tOuter: " << tOuter << std::endl;

Mat1b mask = chromaKey(src, chroma, tInner, tOuter);

//std::cout << "mask image size: " << mask.size << "mask type: " << mask.type() << std::endl;

Mat chromaKeyImage = replaceBackground(src, mask, resized\_image2);

//std::cout << "newBG image size: " << chromaKeyImage.size << "newBackround size: " << mask.size << std::endl;

//Display results

//imshow("source Image", src);

//imshow("chromaImage", mask);

imshow("new Image", chromaKeyImage);

//Wait until user exits program

std::cout << "program Ended. press any Key. ";

waitKey(0);

return 0;

}

Scalar bgr2ycrcb(cv::Scalar bgr)

{

double R = bgr[2];

double G = bgr[1];

double B = bgr[0];

double delta = 128; // Note: change this value if image type isn't CV\_8U.

double Y = 0.299 \* R + 0.587 \* G + 0.114 \* B;

double Cr = (R - Y) \* 0.713 + delta;

double Cb = (B - Y) \* 0.564 + delta;

return cv::Scalar(Y, Cr, Cb, 0 /\* ignored \*/);

}

Mat1b chromaKey(const Mat3b& imageBGR, Scalar chromaBGR, double tInner, double tOuter)

{

assert(tInner <= tOuter);

// Convert to YCrCb.

assert(!imageBGR.empty());

cv::Size imageSize = imageBGR.size();

//std::cout << "chroma\_mask image size: " << imageBGR.size << std::endl;

cv::Mat3b imageYCrCb;

cv::cvtColor(imageBGR, imageYCrCb, cv::COLOR\_BGR2YCrCb);

cv::Scalar chromaYCrCb = bgr2ycrcb(chromaBGR); // Convert a single BGR value to YCrCb.

// Build the mask.

cv::Mat1b mask = cv::Mat1b::zeros(imageSize);

const cv::Vec3d key(chromaYCrCb[0], chromaYCrCb[1], chromaYCrCb[2]);

for (int y = 0; y < imageSize.height; ++y)

{

for (int x = 0; x < imageSize.width; ++x)

{

const cv::Vec3d color(imageYCrCb(y, x)[0], imageYCrCb(y, x)[1], imageYCrCb(y, x)[2]);

double distance = cv::norm(key - color);

if (distance < tInner)

{

// Current pixel is fully part of the background.

mask(y, x) = 0;

}

else if (distance > tOuter)

{

// Current pixel is fully part of the foreground.

mask(y, x) = 255;

}

else

{

double d1 = distance - tInner;

double d2 = tOuter - tInner;

uint8\_t alpha = static\_cast<uint8\_t>(255. \* (d1 / d2));

mask(y, x) = alpha;

//mask(y, x) = saturate\_cast<uchar>((1-alpha)\*backgroundImage.at<Vec3b>(y,x) + alpha \* imageBGR.at<Vec3b>(y,x));

}

}

}

return mask;

}

Mat3b replaceBackground(const cv::Mat3b& image, const cv::Mat1b& mask, cv::Mat bgImage)

{

cv:Mat3b newImage= cv::Mat3b::zeros(Size(image.cols,image.rows));

//std::cout << "func\_test image size: " << newImage.size << std::endl;

//std::cout << "func\_test bgImage size: " << bgImage.size << std::endl;

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

{

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

{

uint8\_t maskValue = mask(y, x);

if (maskValue >= 255)

{

newImage(y, x) = image(y, x);

}else if (maskValue <= 0)

{

newImage(y, x) = bgImage.at<Vec3b>(y,x);

}else

{

double alpha = 1. / static\_cast<double>(maskValue);

newImage(y, x) = alpha \* image(y, x) + (1. - alpha) \* bgImage.at<Vec3b>(y, x);

}

}

}

return newImage;

}

**[Result]**

**case(tInner:30, tOutter:100)**

