《Poisson Image Editing》代码阅读报告

李文彬，1120173001

2020 年 1 月 30 日

1 论文概要

1.1 论文要解决的问题

论文中提出了一种基于泊松方程的图像融合方法。泊松重建的实质是通过方程组 Ax = b 的求解得到融合结果的像素颜色值。图像融合算法即计算系数矩阵 A 和散度 b.

该方法可以实现前景图像和背景图像在融合边界上的较为自然过渡效果。

1.2 论文采用的主要方法

对于前景图像兴趣区域 (ROI), 计算 ROI 的梯度场 Isrc 背景图像不被修改的像素区域的梯度场 Idst。然后通过对 Isrc 和 Idst 结果求和得到整幅待重建图像的梯度场，最后才根据梯度场求解散度。

计算泊松方程组散度值 b 的流程如图1

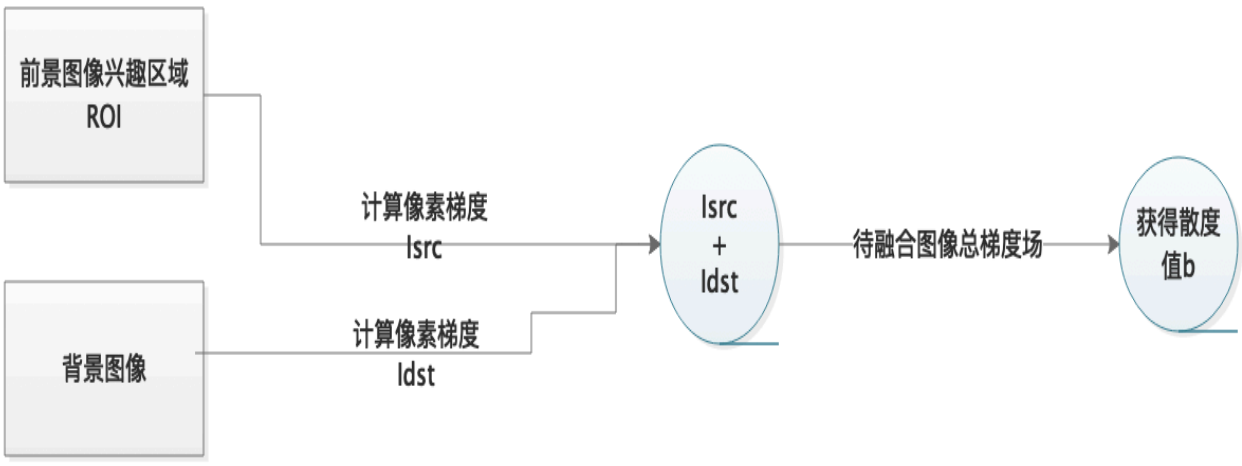

图 1: 散度值 b 求解流程图

2 相关模块功能以及 API 调用

本报告主要分析 Seamless_clone.cpp 的功能实现以及分析相关模块功能，相关 API 调用

2.1 demo 分析

```
{/**
作用:
剪裁，无缝融合，待融合区域先剪裁

*/

#include <blend/clone.h>
#include <opencv2/opencv.hpp>

/**

Naive image cloning by just copying the values from foreground over background

*/
void naiveClone(cv::InputArray background_,
cv::InputArray foreground_,
cv::InputArray foregroundMask_,
int offsetX, int offsetY,
cv::OutputArray destination_)
{
cv::Mat bg = background_.getMat(); //背景图像
cv::Mat fg = foreground_.getMat(); //前景图像
cv::Mat fgm = foregroundMask_.getMat(); //前景mask

destination_.create(bg.size(), bg.type()); // dst兴趣区域的矩形
cv::Mat dst = destination_.getMat();

cv::Rect overlapAreaBg, overlapAreaFg;
blend::detail::findOverlap(background_, foreground_, offsetX, offsetY, overlapAreaBg, overlapAreaFg);

/**
前景图片 矩形中图像 转到背景图片中

*/
bg.copyTo(dst);
fg(overlapAreaFg).copyTo(dst(overlapAreaBg), fgm(overlapAreaFg));

}

/**

Main entry point.

*/
int main(int argc, char **argv)
{
if (argc != 6) {
std::cerr << argv[0] << "␣background␣foreground␣mask␣offsetx␣offsety" << std::endl;
return -1;
}

cv::Mat background = cv::imread(argv[1]);
cv::Mat foreground = cv::imread(argv[2]);
cv::Mat mask = cv::imread(argv[3], CV_LOAD_IMAGE_GRAYSCALE);
int offsetx = atoi(argv[4]);
```

```
int offsety = atoi(argv[5]);


cv::Mat result;

naiveClone(background, foreground, mask, offsetx, offsety, result);
cv::imshow("Naive", result);
cv::imwrite("naive.png", result);

/**
* 输出融合结果
* 步骤一 : 计算组合梯度场
* mixed
* 保留destination 图像的texture 细节
* 目标区域的梯度是由原图像和目的图像的组合计算出来(计算gradient)。
* 步骤二 : 计算背景图片梯度场
* 步骤三 : 计算平均梯度场
*/

blend::seamlessClone(background, foreground, mask, offsetx, offsety, result, blend::CLONE_MIXED_GRADIENTS);
cv::imshow("Mixed␣Gradients", result);
cv::imwrite("mixed-gradients.png", result);

blend::seamlessClone(background, foreground, mask, offsetx, offsety, result, blend::CLONE_FOREGROUND_GRADIENTS);
cv::imshow("Foreground␣Gradients", result);
cv::imwrite("foreground-gradients.png", result);

blend::seamlessClone(background, foreground, mask, offsetx, offsety, result, blend::CLONE_AVERAGED_GRADIENTS);
cv::imshow("Averaged␣Gradients", result);
cv::imwrite("averaged-gradients.png", result);

cv::waitKey();

return 0;
}
}
```

2.2 功能实现分析

现假设有一图像 g, 如图2

图 2: 待克隆图像 ROI

有一背景图片 S, 如图3

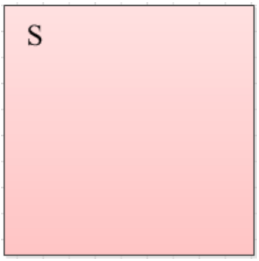

图 3: 背景图片 R

融合目标即为把图片 G 融合粘贴到 s 中，且实现自然粘贴的效果，如图4

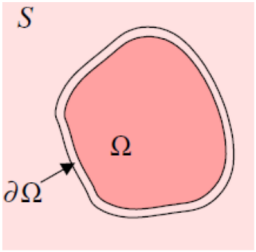

图 4: 融合后图片

2.3 算法实现

——与 Seamless_cloning.cpp 相关的 OpenCV 函数库及 API 分析

2.3.1 计算前景图像 ROI 的梯度场

通过差分方法，可以求得图像 g 的梯度场。OpenCV 中泊松融合计算 g 梯度场的代码如下：

```
void Cloning::computeGradientX( const Mat &img, Mat &gx)
{
Mat kernel = Mat::zeros(1, 3, CV_8S);
kernel.at<char>(0,2) = 1;
kernel.at<char>(0,1) = -1;

if(img.channels() == 3)
{
filter2D(img, gx, CV_32F, kernel);
}
else if (img.channels() == 1)
{
Mat tmp[3];
for(int chan = 0 ; chan < 3 ; ++chan)
{
filter2D(img, tmp[chan], CV_32F, kernel);
}
merge(tmp, 3, gx);
}
}

void Cloning::computeGradientY( const Mat &img, Mat &gy)
{
Mat kernel = Mat::zeros(3, 1, CV_8S);
```

```
kernel.at<char>(2,0) = 1;
kernel.at<char>(1,0) = -1;

if(img.channels() == 3)
{
filter2D(img, gy, CV_32F, kernel);
}
else if (img.channels() == 1)
{
Mat tmp[3];
for(int chan = 0 ; chan < 3 ; ++chan)
{
filter2D(img, tmp[chan], CV_32F, kernel);
}
merge(tmp, 3, gy);
}
}
```

通过上述代码，可以得到函数返回值 V(pathGradientX,VpatchGradientY)，相关函数调用 API 如下：

```
computeGradientX(patch,patchGradientX);//计算ROI区域转换复制到destination一样大小的patch图片x方向梯度
computeGradientY(patch,patchGradientY);//计算y方向梯度
```

2.3.2 计算背景图片的梯度场

与前景图像计算同理，可以得到背景图片和融合图片的梯度场，函数调用 API 如下:

```
computeGradientX(destination,destinationGradientX);//计算背景图像的x方向梯度
computeGradientY(destination,destinationGradientY);//计算背景图像y方向的梯度
```

2.3.3 计算融合图像的梯度场

```
Mat laplacianX = Mat(destination.size(),CV_32FC3);
Mat laplacianY = Mat(destination.size(),CV_32FC3);

//因为前面已经对destinationGradientX做了固定区域的mask，patchGradientX做了修改区域的mask
laplacianX = destinationGradientX + patchGradientX;//求解整张图片新的梯度场
laplacianY = destinationGradientY + patchGradientY;
```

2.3.4 求解融合图像散度

对拉普拉斯算子在 x 和 y 方向求偏导，求和;

```
lap = laplacianX + laplacianY;//散度
```

3 实验运行

由于原始版 C++demo 使用不熟练，因此选择运行开源的 Python 版 demo，但针对实验结果 API 给出 C++ 版本的解释

开源链接：https://www.learnopencv.com/seamless-cloning-using-opencv-python-cpp/

3.1 Python 版本

可运行程序:

```
# 注意修改路径!
import cv2
import numpy as np

# Read images : src image will be cloned into dst
im = cv2.imread("images/wood-texture.jpg")
obj= cv2.imread("images/iloveyouticket.jpg")

# Create an all white mask
mask = 255 * np.ones(obj.shape, obj.dtype)

# The location of the center of the src in the dst
width, height, channels = im.shape
center = (height/2, width/2)

# Seamlessly clone src into dst and put the results in output
normal_clone = cv2.seamlessClone(obj, im, mask, center, cv2.NORMAL_CLONE)
mixed_clone = cv2.seamlessClone(obj, im, mask, center, cv2.MIXED_CLONE)

# Write results
cv2.imwrite("images/opencv-normal-clone-example.jpg", normal_clone)
cv2.imwrite("images/opencv-mixed-clone-example.jpg", mixed_clone)
```

3.2 API 解释

3.2.1 C++ 版本

```
blend::seamlessClone(background, foreground, mask, offsetx, offsety, result, blend::CLONE_MIXED_GRADIENTS);
//or
blend::seamlessClone(background, foreground, mask, offsetx, offsety, result, blend::CLONE_FOREGROUND_GRADIENTS);
//or
blend::seamlessClone(background, foreground, mask, offsetx, offsety, result, blend::CLONE_AVERAGED_GRADIENTS);
```

foreground	目标影像
background	背景图片
mask	表示目标影像上的 ROI
offsetx	目标影像中心在背景图片上 x 坐标
offsety	目标影像中心在背景图片上 y 坐标
result	输出图像
blend::	模式选择

3.2.2 Python 版本

```
output = cv2.seamlessClone(src, dst, mask, center, flags)
```

（同名参数以及简单命名参数与 C++ 版本比较，易得类似 API 解释）

center	目标影像中心在背景图片上坐标
flags	模式选择

3.3 运行实例

前景图像

图 5: 前景图像

背景图片

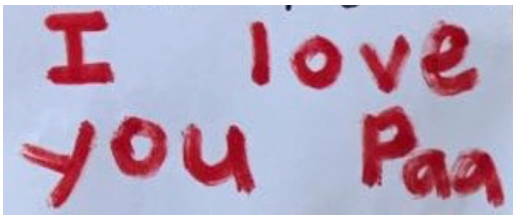

图 6: 背景图片

前景图像

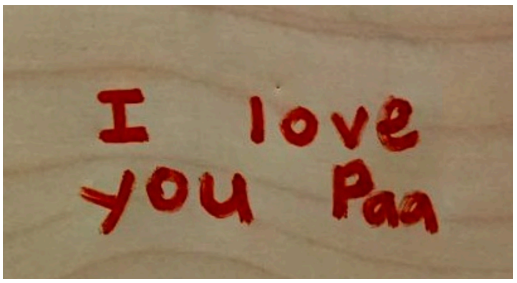

图 7: 融合图片