OpenCV图像处理专栏十四 | 基于Retinex成像原理的自动色彩均衡算法(ACE)

BBuf

前言

这个算法是IPOL上一篇名为《Automatic Color Equalization(ACE) and its Fast Implementation》提出的，这实际上也是《快速ACE算法及其在图像拼接中的应用》这篇论文中使用的ACE算法，这个算法主要是基于Retinex成像理论做的自动彩色均衡，我用C++ OpenCV实现了，来分享一下。

算法原理

在论文介绍中提到，高动态图像是指在一幅图像中，既有明亮的区域又有阴影区域，为了使细节清晰，需要满足以下几点：

- （1）对动态范围具有一定的压缩能力
- （2）对亮暗区域的细节有一定的显示能力
- （3）在满足(1)，(2)的条件下不破坏图像的清晰度

Rizzi等人根据Retinex理论提出自动色彩均衡算法，该算法考虑了图像中颜色和亮度的空间位置关系，进行局部的自适应滤波，实现具有局部和非线性特征的图像亮度，色彩与对比度调整，同时满足灰度世界理论和白斑点假设。

获得空域重构图像

对图像进行色彩/空域调整，完成图像的色差矫正，得到空域重构图像：

$$R_c(p) = \sum_{j \in Subset, j \neq p} \frac{r(I_c(p) - I_c(j))}{d(p,j)}$$

其中 R_c 是中间结果，$I_c(p) - I_c(j)$ 为2个点的亮度差，$d(p,j)$ 表示距离度量函数，$r(*)$ 为亮度表现函数，需要是奇函数，这一步可以适应局部图像对比度，$r(*)$ 可以放大较小的差异，并丰富大的差异，根据局部内容扩展或者压缩动态范围。一般的，$r(*)$ 为：

$$r(x) = \left\{ \begin{array}{c} 1, x < -T \\ x/T, -T <= x <= T \\ -1, x > T \end{array} \right\}$$

对矫正后的图像进行动态扩展

对矫正后的图像进行动态扩展，一种简单的线性扩展为：

$O_c(p) = round[127.5 + s_c R_c(p)]$，

其中 s_c 是 $[(m_c, 0), (M_c, 255)]$ 的斜率，其中：

$M_c = max_p[R_c(p)]$，$m_c = min_p[R_c(p)]$

我们还可以将其映射到 $[0, 255]$ 的空间中：

$$L(x) = \frac{R(x) - minR}{maxR - minR}$$

我实现代码时选择了后者，效果会好一点。

快速ACE算法实现

在查阅资料[参考2]的时候看到一个非常有趣的改进方法，可以让ACE算法速度更快，更利于实际应用。

快速ACE算法基于两个基本假设：(1)对一副图像ACE增强后得到输出 Y，如果对 Y 再进行一次ACE增强，输出仍然是 Y 本身；(2）对一副图像的ACE增强结果进行尺寸缩放得到 Y，对 Y 进行ACE增强，输出仍然是 Y 本身。

如果上面假设成立，我们就可以对图像进行缩放得到 $I1$，对 $I1$ 的ACE增强结果进行尺度放大（与 I 尺寸一样）得到 $Y1$，那么 Y 和 $Y1$ 是非常接近的，我们只需要在 $Y1$ 基础上进一步处理即可。这里就又引申了两个细节问题：

- **「如何快速的求 $I1$ 的ACE增强结果？」** 其实很简单，对它再次缩放得到 $I2$，求 $I2$ 的增强结果，依次类推就是金字塔结构思想。
- **「如何在 $Y1$ 基础上进一步处理得到 Y？」** 因为是在整个图像域进行差分比较运算，与近处邻域像素的比较构成了 Y 的细节信息，与远处像素的比较构成了 Y 的全局背景信息，那么我们合理假设，Y 和 $Y1$ 的全局背景信息相同，只更新细节信息即可。所以，我们需要在 $Y1$ 的基础上加上 I 中邻近像素的差分结果，并减去 $Y1$ 中邻近像素的差分结果就是最终的输出 Y。

注意，这种方法不是论文中使用的改进方法，之所以要介绍这种方法是因为它操作起来很简单，同

可以去看原论文。

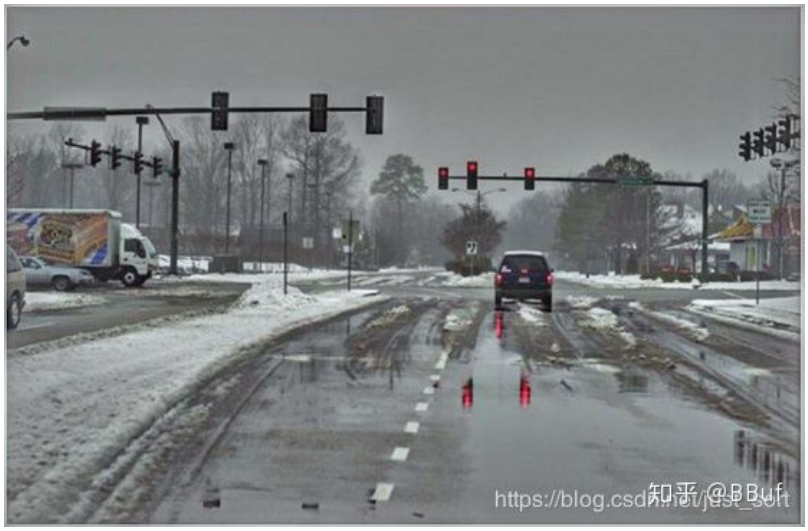
```
#include <stdio.h>
#include <iostream>
#include <immintrin.h>
#include <opencv2/opencv.hpp>
#include <opencv2/core/core.hpp>
#include <opencv2/ml/ml.hpp>
#include "opencv2/highgui/highgui.hpp"
using namespace cv;
using namespace cv::ml;
using namespace std;

namespace ACE {
	//Gray
	Mat stretchImage(Mat src) {
		int row = src.rows;
		int col = src.cols;
		Mat dst(row, col, CV_64FC1);
		double MaxValue = 0;
		double MinValue = 256.0;
		for (int i = 0; i < row; i++) {
			for (int j = 0; j < col; j++) {
				MaxValue = max(MaxValue, src.at<double>(i, j));
				MinValue = min(MinValue, src.at<double>(i, j));
			}
		}
		for (int i = 0; i < row; i++) {
			for (int j = 0; j < col; j++) {
				dst.at<double>(i, j) = (1.0 * src.at<double>(i, j) - M
				if (dst.at<double>(i, j) > 1.0) {
					dst.at<double>(i, j) = 1.0;
				}
				else if (dst.at<double>(i, j) < 0) {
					dst.at<double>(i, j) = 0;
				}
			}
		}
		return dst;
	}

	Mat getPara(int radius) {
		int size = radius * 2 + 1;
		Mat dst(size, size, CV_64FC1);
		for (int i = -radius; i <= radius; i++) {
			for (int j = -radius; j <= radius; j++) {
				if (i == 0 && j == 0) {
					dst.at<double>(i + radius, j + radius) = 0;
				}
				else {
					dst.at<double>(i + radius, j + radius) = 1.0 /
				}
			}
		}
		double sum = 0;
		for (int i = 0; i < size; i++) {
			for (int j = 0; j < size; j++) {
				sum += dst.at<double>(i, j);
			}
		}
		for (int i = 0; i < size; i++) {
```

```
        }
        return dst;
}

Mat NormalACE(Mat src, int ratio, int radius) {
        Mat para = getPara(radius);
        int row = src.rows;
        int col = src.cols;
        int size = 2 * radius + 1;
        Mat Z(row + 2 * radius, col + 2 * radius, CV_64FC1);
        for (int i = 0; i < Z.rows; i++) {
                for (int j = 0; j < Z.cols; j++) {
                        if((i - radius >= 0) && (i - radius < row) && (j - rad
                                Z.at<double>(i, j) = src.at<double>(i - radius
                        }
                        else {
                                Z.at<double>(i, j) = 0;
                        }
                }
        }

        Mat dst(row, col, CV_64FC1);
        for (int i = 0; i < row; i++) {
                for (int j = 0; j < col; j++) {
                        dst.at<double>(i, j) = 0.f;
                }
        }
        for (int i = 0; i < size; i++) {
                for (int j = 0; j < size; j++) {
                        if (para.at<double>(i, j) == 0) continue;
                        for (int x = 0; x < row; x++) {
                                for (int y = 0; y < col; y++) {
                                        double sub = src.at<double>(x, y) - Z.
                                        double tmp = sub * ratio;
                                        if (tmp > 1.0) tmp = 1.0;
                                        if (tmp < -1.0) tmp = -1.0;
                                        dst.at<double>(x, y) += tmp * para.at<
                                }
                        }
                }
        }
        return dst;
}

Mat FastACE(Mat src, int ratio, int radius) {
        int row = src.rows;
        int col = src.cols;
        if (min(row, col) <= 2) {
                Mat dst(row, col, CV_64FC1);
                for (int i = 0; i < row; i++) {
                        for (int j = 0; j < col; j++) {
                                dst.at<double>(i, j) = 0.5;
                        }
                }
                return dst;
        }

        Mat Rs((row + 1) / 2, (col + 1) / 2, CV_64FC1);
```

```
        Mat dst(row, col, CV_64FC1);
        Mat dst1 = NormalACE(src, ratio, radius);
        Mat dst2 = NormalACE(Rs, ratio, radius);
        for (int i = 0; i < row; i++) {
                for (int j = 0; j < col; j++) {
                        dst.at<double>(i, j) = Rf.at<double>(i, j) + dst1.at<d
                }
        }
        return dst;
}

Mat getACE(Mat src, int ratio, int radius) {
        int row = src.rows;
        int col = src.cols;
        vector <Mat> v;
        split(src, v);
        v[0].convertTo(v[0], CV_64FC1);
        v[1].convertTo(v[1], CV_64FC1);
        v[2].convertTo(v[2], CV_64FC1);
        Mat src1(row, col, CV_64FC1);
        Mat src2(row, col, CV_64FC1);
        Mat src3(row, col, CV_64FC1);

        for (int i = 0; i < row; i++) {
                for (int j = 0; j < col; j++) {
                        src1.at<double>(i, j) = 1.0 * src.at<Vec3b>(i, j)[0] /
                        src2.at<double>(i, j) = 1.0 * src.at<Vec3b>(i, j)[1] /
                        src3.at<double>(i, j) = 1.0 * src.at<Vec3b>(i, j)[2] /
                }
        }
        src1 = stretchImage(FastACE(src1, ratio, radius));
        src2 = stretchImage(FastACE(src2, ratio, radius));
        src3 = stretchImage(FastACE(src3, ratio, radius));

        Mat dst1(row, col, CV_8UC1);
        Mat dst2(row, col, CV_8UC1);
        Mat dst3(row, col, CV_8UC1);
        for (int i = 0; i < row; i++) {
                for (int j = 0; j < col; j++) {
                        dst1.at<uchar>(i, j) = (int)(src1.at<double>(i, j) * 2
                        if (dst1.at<uchar>(i, j) > 255) dst1.at<uchar>(i, j) =
                        else if (dst1.at<uchar>(i, j) < 0) dst1.at<uchar>(i, j
                        dst2.at<uchar>(i, j) = (int)(src2.at<double>(i, j) * 2
                        if (dst2.at<uchar>(i, j) > 255) dst2.at<uchar>(i, j) =
                        else if (dst2.at<uchar>(i, j) < 0) dst2.at<uchar>(i, j
                        dst3.at<uchar>(i, j) = (int)(src3.at<double>(i, j) * 2
                        if (dst3.at<uchar>(i, j) > 255) dst3.at<uchar>(i, j) =
                        else if (dst3.at<uchar>(i, j) < 0) dst3.at<uchar>(i, j
                }
        }
        vector <Mat> out;
        out.push_back(dst1);
        out.push_back(dst2);
        out.push_back(dst3);
        Mat dst;
        merge(out, dst);
        return dst;
}
}
```

```
    Mat src = imread("F:\\sky.jpg");
    Mat dst = getACE(src, 4, 7);
    imshow("origin", src);
    imshow("result", dst);
    waitKey(0);
}
```

效果




知乎@BBuf
https://blog.csdn.net/just_sort


知乎@BBuf
https://blog.csdn.net/just_sort


知乎@BBuf
https://blog.csdn.net/just_sort


https://blog.csdn.net/just_sort
知乎 @BBuf


https://blog.csdn.net/just_sort
知乎 @BBuf


@BBuf

- 论文原文：ipol.im/pub/art/2012/g-...
- 作者开源代码：ipol.im/pub/art/2012/g-...
- 参考1：blog.csdn.net/piaoxuezh...
- 参考2：cnblogs.com/whw19818/p/...

同期文章

- OpenCV图像处理专栏一 | 盘点常见颜色空间互转
- OpenCV图像处理专栏二 |《Local Color Correction 》论文阅读及C++复现
- OpenCV图像处理专栏三 | 灰度世界算法原理和实现
- OpenCV图像处理专栏四 | 自动白平衡之完美反射算法原理及C++实现
- OpenCV图像处理专栏五 | ACE算法论文解读及实现
- OpenCV图像处理专栏六 | 来自何凯明博士的暗通道去雾算法(CVPR 2009最佳论文)
- OpenCV图像处理专栏七 | 直方图均衡化算法及代码实现
- OpenCV图像处理专栏八 |《Contrast image correction method》论文阅读及代码实现
- OpenCV图像处理专栏九 | 基于直方图的快速中值滤波算法
- OpenCV图像处理专栏十 | 利用中值滤波进行去雾
- OpenCV图像处理专栏十一 | IEEE Xplore 2015的图像白平衡处理之动态阈值法
- OpenCV图像处理专栏十二 |《基于二维伽马函数的光照不均匀图像自适应校正算法》
- OpenCV图像处理专栏十三 | 利用多尺度融合提升图像细节

欢迎关注GiantPandaCV, 在这里你将看到独家的深度学习分享，坚持原创，每天分享我们学习到的新鲜知识。(• ω •)✧

有对文章相关的问题，或者想要加入交流群，欢迎添加BBuf微信：

u.wechat.com/MPWFDnmCPu... (二维码自动识别)

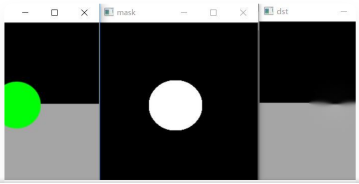
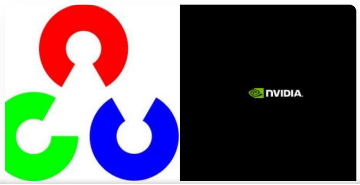
发布于 2020-02-08 23:19

图像处理 算法 OpenCV

文章被以下专栏收录

PandaCV
欢迎关注同名公众号PandaCV

推荐阅读

图像处理- 特征点检

OpenCV图像

2 条评论 切换为时间排序

写下你的评论...

愿爱你如初 2021-05-28

作者能出一个注释版吗

赞

虚御安玄 2020-11-06

代码很有用

赞