清 华 大 学

计算机科学与技术系

计算机专业实践

课题名称： 图像修补

姓　　名 肖桐　学号 2009011258　班号 计 92

指导老师 徐昆　辅导老师 徐昆

成绩_____

二零零_年_月

目　　录

1. 图像修补.....3
1.1 引言.....3
1.2 研究现状.....3
2. 算法理论描述.....4
2.1 无用户交互部分.....4
2.2 用户交互部分.....5
3. 程序实现.....7
4. 优化及改进.....9
4.1 距离因子.....9
4.2 动态像素块大小.....9
5. 总结与展望.....10
参考文献.....10

1. 图像修补

1.1 引言

图像修补是数字图像处理领域的一个经典而有趣的问题，是指在一张图片内，将用户指定的区域去除掉，并填充以合理的像素点，使得看起来仍然很自然。图 1 很直观地给出了期望得到的效果。

图 1 图像修补算法所希望得到的效果

1.2 研究现状

在图像修补领域，当前的研究主要是针对单张静态图片的。在我接触到的论文中，大部分的学者采用的方法是将标注的区域以像素块为单位逐步替换为非标注区域的像素块。这对于单张图片来说是相对合理的一种方法。

最近的一些论文针对这个基本思想从不同方面进行了一些改进。可以归纳为以下几点：

1. 更合适的替换像素块
2. 更合适的被替换像素块
3. 保留显著的线性特征（直线、颜色分界线……）
4. 保留图像的结构特征（被遮挡的物体形状）

其中[1]提出了一个选取被替换像素块的顺序的方法，改善了 1 和 3 的算法。[2]则引入了用户的交互，使得图像的结构特征可以得到较好的保留，改善了 4 的算法。我的实验主要是基于这两篇论文完成的，并针对存在的问题，做了一定程度上的改进。

2. 算法理论描述

图像修补问题的输入是一张原图像，和一张标记图像（标记出要被去除的区域，以及用户指定的结构信息）。输出是一张目标图像，将标记的区域去除并填充上。

2.1 无用户交互部分

这部分的目标是自动地将标记的区域去除，算法的思想大致如下：

1. 找出标记区域的一个最“容易”被去除的像素块。
2. 在原图像的其他区域中找出和它最匹配的像素块来替换。
3. 重复 1 和 2 直到所有标记区域都被填充完。

可以看出，如何定义一个像素块是否“容易”被去除呢？很容易想到的一点是从边界慢慢“腐蚀”到内部是一个合理的策略，因此，如何找到好的一个边界像素块成为了我们现在面临的问题。

我们首先需要定义一些基本的标识。如图 2 所示，I 是原图像，Φ是不用被去除的区域，Ω是要被去除的区域，δΩ是标记区域的边界，p 是其上的一个像素点，$\mathbf{n_p}$是边界在 p 处的法向量，$\nabla\mathbf{I_p^{\perp}}$是原图像在 p 点的光线方向（即梯度的垂直方向），Ψ_p是在 p 点的像素块。

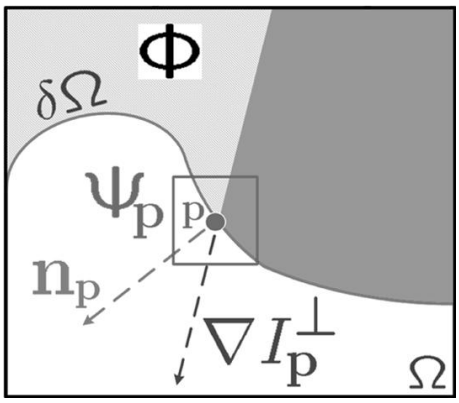


图 2 一些基本的标识

接下来，对于每个边界上的像素 p，引入它被去除的优先级

$$f(p) = c(p) \times d(p)$$

其中

$$c(p) = \frac{1}{|\Psi_p|} \sum_{q \in \Psi_p \cap (I-\Omega)} c(q)$$

$$d(p) = \frac{\left|\nabla\mathbf{I_p^{\perp}} \cdot \mathbf{n_p}\right|}{255}$$

直观上说，c(p)是像素块内包含要被去除的像素点的多少，越多c(p)的值越小；d(p)描述了点 p 是否是一个延续了原图像线性特征的点，$\mathbf{n_p}$和$\nabla\mathbf{I_p^{\perp}}$夹角越小，d(p)的值越大。因此，我们每次只需要找出 f 值最大的像素点 p 即可。

下一步我们将要找出一个和Ψ_p最接近的像素块Ψ_q，这里的度量标准采取的是对于Ψ_p中未被标记的像素，在Ψ_q中找到对应的像素，计算二者的 RGB 三个通道的差的平方和的均值，然后再对这样的像素做累加。即所谓“sum of the squared differences(SSD)”。

在用Ψ_q替换Ψ_p之后，我们还需要更新Ψ_p中被标记像素的c值为c(p)。这样就完成了一次块替换的过程。

2.2 用户交互部分

用户交互如图 3 所示，用户在标记出要去除区域（蓝色）之后，再标记两条维持结构（绿色）的线。算法将首先把基本结构的区域去除，然后再利用 2.1 的方法将其他部分去除。

图 3 用户交互示意图

用户交互的目的是保留一些不易被计算机分析得到的图像结构，如图 4 所示，该问题可以形式化地描述为：

给出一系列需要被去除的结构点p_i，以及可以用作替换的像素块的集合P，要在 P 中找到最合适p_i的像素块$\mathrm{P}(x_i)$。即将问题转化以为标记类问题（labeling problem）。

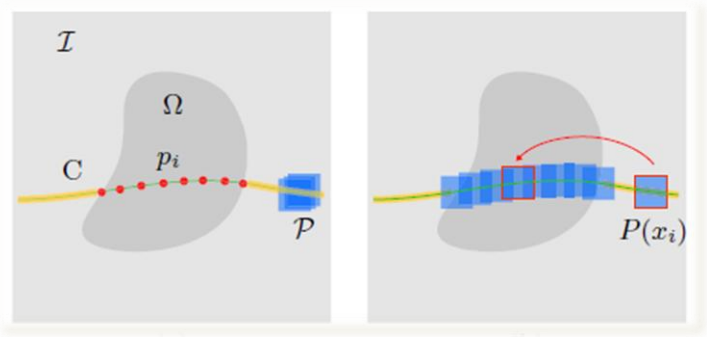


图 4 用户交互的问题描述

针对标记类问题，通常的方法是定义能量函数，然后使其最小化。这里能量函数定义为

$$E = \sum_i E_1(x_i) + \sum_{(i,j)} E_2(x_i, x_j)$$

其中

$$E_1 = k_s \cdot E_s(x_i) + k_b \cdot E_b(x_i)$$

这里，$E_s(x_i)$是将p_i替换为$\mathrm{P}(x_i)$的结构上的代价，如图 5 左所示，其值是黄线的每个点到红线的最短距离的和，以及红线的每个点到黄线的最短距离的和，二者分别除以各自的像素点数之后再相加。直观上说，两条线的形状、位置越相近，代价越低。

$E_b(x_i)$是针对边界上的匹配而设的，如图 5 右绿色框所示，其值是黄色区域的 SSD。

$E_2(x_i, x_j)$是相邻两个像素块的匹配程度，如图 5 右红色框所示，其值是黄色区域的 SSD。

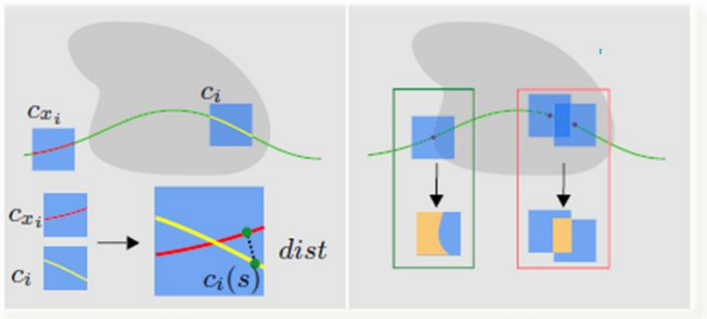


图 5 能量函数每项的直观含义

在定义好了能量函数之后，就可以利用动态规划（Dynamic Programming）或者置信传播（Belief Propagation）来求其最小值及对应的标号。

动态规划方法是针对单独一条线的方法，即定义$\mathrm{f}(\mathrm{i},x_i)$表示前 i 个点已被去除，且其中第 i 个点是用$\mathrm{P}(x_i)$这个像素块来替代时$E$的最小值。那么容易得到如下的递推关系

$$\mathrm{f}(\mathrm{i},x_i)=E_1(x_i)+\min_{x_{i-1}}\{E_2(x_i,x_{i-1})+\mathrm{f}(\mathrm{i}-1,x_{i-1})\}$$

初始值 $\mathrm{f}(0,x_i)=E_1(x_i),\ \forall x_i$，依次递推即可。

置信传播算法定义$\mathrm{M}(\mathrm{i},\mathrm{j},x_j)$表示节点 i 相信节点 j 被标号$x_j$的程度，它按照如下公式一次次更新

$$\mathrm{M}(\mathrm{i},\mathrm{j},x_j)'=\min_{x_i}\left\{E_1(x_i)+E_2(x_i,x_j)+\sum_{k\neq j,k\in N(i)}\mathrm{M}(\mathrm{k},\mathrm{i},x_i)\right\}$$

最终的标号为

$$x_i^*=arg\min_{x_i}\left\{E_1(x_i)+\sum_{k\in N(i)}\mathrm{M}(\mathrm{k},\mathrm{i},x_i)\right\}$$

其方法的本质想法是逐次迭代，通过相邻节点标号的确定一步步算出每个节点标号。

3. 程序实现

以上是算法所依据的理论基础，在程序的实现中，其流程可以表示如下：

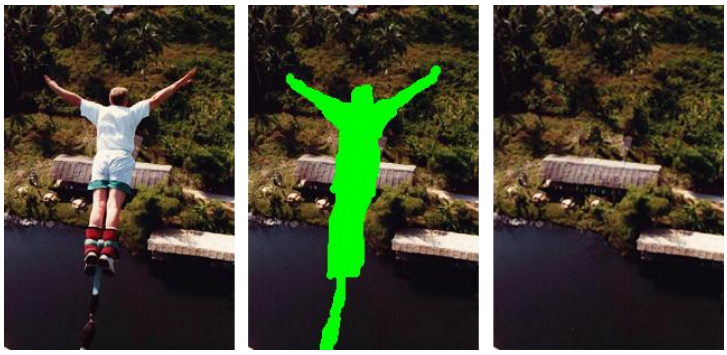
1. 用户标记出要去除的部分，以及要保留的结构特征。
2. 根据结构特征建图，得到要去除的结构点集、用来替换的像素块集。
3. 去除结构区域部分，保留结构特征。
    a) 对于没有与其他线条相交的线，用动态规划方法。
    b) 否则，使用置信传播方法。
4. 去除其他部分
    a) 找出边界上最“容易”被去除的像素块。
    b) 找出原图像中和它最相似的像素块替换。
    c) 重复 a)和 b)直到全部区域都被去除为止。

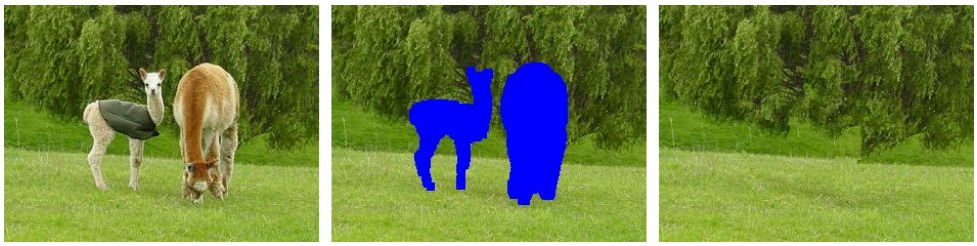
我用 Matlab 实现了不带用户交互的部分，用 Qt 实现了带交互的完整版本。Matlab 版本的使用方法是在 inpainting.m 中设好原图像、标记图像和生成图像和视频的文件名，然后直接 F5 运行即可。需要注意的一点是，为了使其效率提高，我采用了 matlab 调用 c 程序的方法，因此可能需要提前用 mex 编译 c_getfitpatch.c，附带的是 64 位 win7 版本的二进制文件。

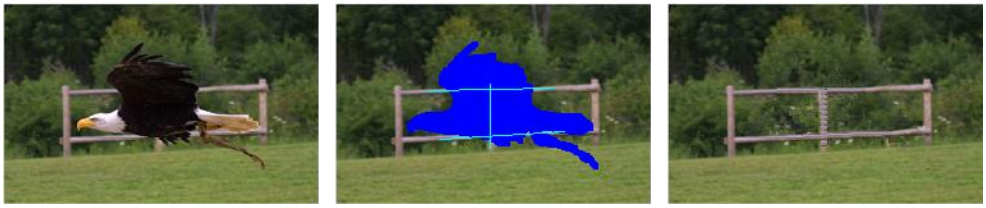
Qt 版本，直接运行 inpainting.exe 即可。分为以下步骤：

1. Ctrl+O 打开图像
2. Ctrl+I 打开图像修补面板
3. 标记要去除的区域
    a) 按“Markup”
    b) 在图像上按住并拖动鼠标
    c) 按“End Markup”
4. 标记要保留的结构特征
    a) 按“Start Structure”
    b) 在图像上按住并拖动鼠标，按住 shift 键将画直线
    c) 按“End Structure”
5. 按“Inpainting”开始自动去除并填充
6. 完成后，可以在图像区域单击鼠标观察变化，按“Verify”确认
7. 如果效果不理想，可以在当前基础上重复 3~6 步骤以达到比较好的效果

以下是一些运行效果的展示







4. 优化及改进

4.1 距离因子

在测试中，我发现了之前算法存在的一些不足之处，如图 6 所示，一些替换的像素块明显与周围的环境不符。

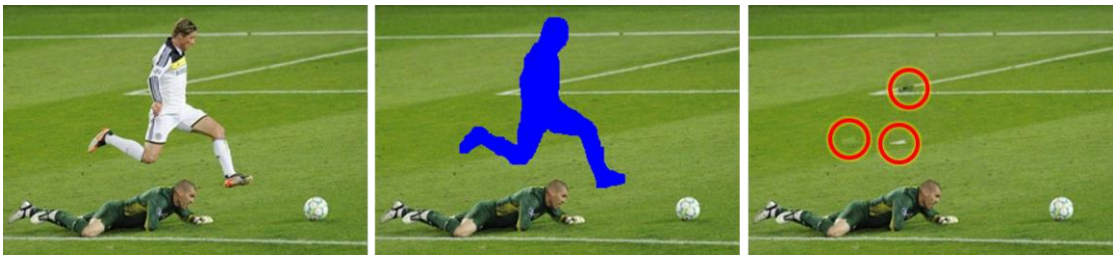


图 6 原有算法的一些缺陷

为了解决这个问题，我尝试了两种方法，其一是将色彩空间转换为 Lab 空间，但效果并不理想；其二是考虑到两个像素块之间的距离因素，将两个像素点p_1, p_2间的差异值乘以如下的系数

$$\alpha = 1 - \exp(-\frac{\|p_1 - p_2\|^2}{\sigma^2})$$

加入了这个改进之后的对比效果如所示

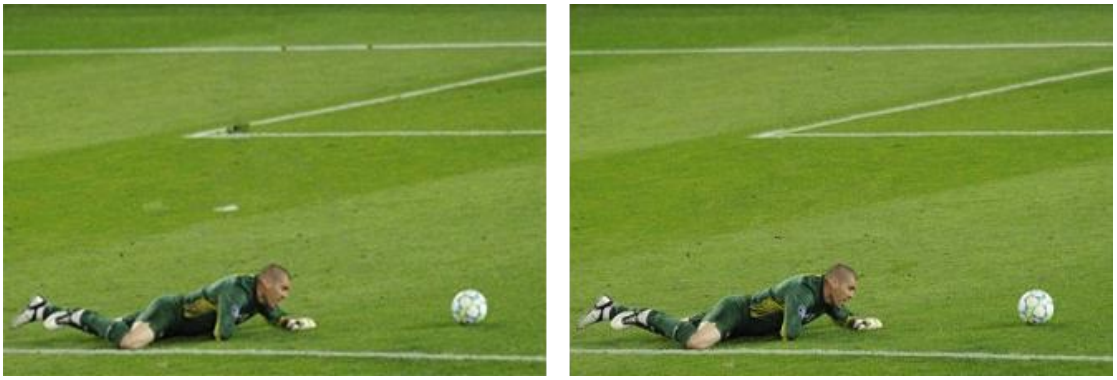


图 7 加入距离因素之后的效果

4.2 动态像素块大小

对于整个算法而言，像素块的大小是决定最终去除效果好坏的非常重要的参数。论文中的像素块大小都是固定的，这样在一些细节变化比较大的地方会使得效果不好。因此，我尝试了动态调整像素块大小，根据的是之前所述的d(p)这个值，如果值越大，那么意味着越可能是线性性比较好的像素，因此采用比较小的像素块来填充；否则，使用大一些的像素块填充以加快速度。

5. 总结与展望

通过本次专业实践，我又获得了一些关于图像处理方面的知识。而且，自己探索发现问题并尝试各种方法解决的过程也令我收获了很多研究方面的经验。

针对图像修补这个问题，我期望的效果是能投入实际使用。这就提出了两方面的要求，其一是最终效果，其二是运行速度。目前这个版本的程序在部分测试图像中表现出来的效果还可以接受，速度方面，由于我是有实时的反馈，所以从体验上来看也可以忍受。但是这两方面无疑都还存在很多的提升空间。

通过看论文和自己思考，我也有一些改进想法。

1. 加入图像融合的部分，比如利用 poisson editing 使得替换的像素块看起来有更好的效果。
2. 在选取替换的像素块时，可以利用简单的旋转变换，看看是否能得到更好的像素块。
3. 采用 cpu/gpu 多线程来更快地找到好的替换像素块。
4. 采用一些近似算法加速保留结构特征。

之后我还希望能够接着完善这个程序，可以留意 https://github.com/Cysu/CysuPik 的更新。

参考文献

1. Antonio Criminisi, Patrick Pérez, et al. *Region Filling and Object Removal by Exemplar-Based Image Inpainting*
2. Jian Sun, Lu Yuan, et al. *Image Completion with Structure Propagation*
3. Patrick Pérez, Michel Gangnet, et al. *Poisson Image Editing*