媒体计算作业一

计71 张程远 2017011429

0 实验要求

实验一的目的是图像生成，通过拷贝源图像的一些像素到背景的某些位置上，拷贝时尽可能让新加入的像素与已有图像较好地融合，减少视觉上的不协调。

1 算法流程

算法要解决两个问题：第一是在哪放patch，第二是放上去的patch有哪些像素被使用。对于第一个问题，算法提供了3种方式选择patch的位置：第一种（Random）是随机选择；第二种（Best_Pos）是定义一个Cost值，每次patch的摆放位置由计算获得；第三种（Best_Patch）是事先确定patch的大小和摆放位置，通过之前定义的Cost，在源图像中选择最合适的patch。

算法的总体流程如下：

（1）在生成图像中放入第一个patch

（2）进入循环：对于Random和Best_Pos模式，patch的形状和像素值已经确定，根据对应的模式计算得到这个patch要放在生成图像的哪个位置，然后重新计算该patch的高和宽；对于Best_Patch模式，patch的形状、要放在哪个位置已经确定，计算得到最佳的像素值（相比于源图像的偏移量）

（3）借助mask检查与已有图像的重合情况，每个重合的像素都看做一个节点，建图并用dinic算法求最小割

（4）对求出来的所有来自T的像素点，以及patch新覆盖到的像素点，更新该点的像素值和mask

（5）如果目标区域全部被填满，终止循环

这里要说明几个细节。

（1）第一个patch的放入，前两种模式的patch就是一整块源图像，我将其直接放在生成图的左上角。后一种模式的patch是部分源图像，我设置为0.6乘长和宽，偏移量为（0，0），并放在生成图的左上角。

（2）连边时所用的权值，正无穷我设置为2^{50}，其余边的权值设置为下式：

$$M(s,t,A,B) = |A(s) - B(s)| + |A(t) - B(t)|$$

其中像素相减即4通道的差的绝对值的平均值。

（3）Best_Patch需要预先确定被放在生成图的哪个位置，我是从左到右从上到下依次选择位置的，每次patch向右或向下扩展的像素随机占整个patch的40% - 80%；对于Best_Pos和Best_Patch，我的初始K值设置为0.01，Var则是输入图像矩阵的var值。每隔5-10轮迭代，K值会被乘一个系数（1.1 - 1.25），表示选择策略需要更加随机化一些。

2 Bonus部分

2.1 old cuts

用which_patch_mask数组来记录当前生成图像中每个像素来自于哪个patch，并将每次处理后的patch存入patch_list。在建图时，对于每个重合像素，检查该像素A和左边邻居B来自于哪个patch（上方的邻居同理）。如果来自于不同patch（设A来自于patch a，B来自于patch b），再检查a是否包含B点且B是否包含a，如果是则说明这是一条旧的seam，那么插入新的节点S，并计算3条arc的权值；如果不是

则说明这里是某一个patch的边界，稍后考虑。如果A和B来自于同一个旧patch，则按原算法所述连边。最后考虑边界，如果边界点有邻居来自于旧patch，则给该边界点连接到S的arc，权值为正无穷；否则如果该边界点有邻居还未填充像素，则给该边界点连接到T的arc，权值也为正无穷。在node_list中存放了上述的节点，其中前node_cnt个节点代表重叠像素，接下来是S和T，最后在arc中插入的节点都在node_listd的后半段，因此跑完算法后只要检查前node_cnt个节点属于S还是T即可。

2.2 surrounded regions

在cut_patch函数中实现了对surrounded regions的特判。如果一个patch覆盖的所有像素的mask值都大于0，就说明该patch一个新的像素都没覆盖，也就是特殊情况。我在patch的内部（距离各边缘2个像素）随机选取了一个点，并令这个点跟T连接一条权值为正无穷的边，这样其内部就会产生一个最小割。

2.3 梯度信息

梯度分为横向和纵向两种。patch在（x，y）处的梯度值如下：

$$G_x = |G(x-1,y) - G(x+1,y)|$$

$$G_y = |G(x,y-1) - G(x,y+1)|$$

然后按照论文中的公式计算分母的值。

2.4 快速傅里叶变换

在计算Cost的时候，能量公式是输入图像像素和对应输出图像像素差的平方，其展开式的负项是一个卷积的形式，可用FFT加速。首先提取出mask，input和output，其中mask代表输出图像中哪些像素已被填充。然后借助numpy的fft2库把数值转换为虚数形式，再输入ifft2进行计算，得到i2，o2，mio，其中i2 o2分别代表公式中的前两项$I(p)^2$ 和 $O(p)^2$ ，mio则是卷积项$I(p)O(p+t)$。

2.5 像素填充策略

填充策略主要是对于Random和Best_Pos而言的。最开始的想法是每次只要有新像素被填充就可以，但这样并不够好，因此我设置了一个判断：考虑全局剩余的空白像素数blank_pixel，如果blank_pixel > 0.4 * patch的像素数，那么说明空白的像素还有很多，此时选择的patch的位置应能够使得新覆盖的像素占到patch的40%-80%；如果blank_pixel >= 0.05 * patch像素数 且 blank_pixel <= 0.4 *patch像素数，就让patch新覆盖的像素比例下降到1% - 40%；在小于1%时即有新像素被覆盖就合法。当然，blank_pixel有多少并不代表一个patch最多能覆盖多少，因此有可能会出现一个位置都不满足上面的标准。这个时候我会选择能填充最多空白像素的点作为这个patch的位置。这样做以后，首先所需要的迭代轮数减少了大概一半，其次生成图像中少了很多不必要的缝隙和边缘，效果也变好了。

另外生成图片相比于目标图片一般要大一些，防止填充边缘时对图片本身的影响。至于生成图像的数组具体大小，我一般开到目标图片宽和高1.5倍到2倍，然后取其左上角为最后结果。

3 实验结果

由于考虑old cuts相比于原算法有大量的修改，因此我直接实现了old cuts的版本，故没有对比图。以下是引入feature前后的比较，以及3种偏移算法得到的结果。

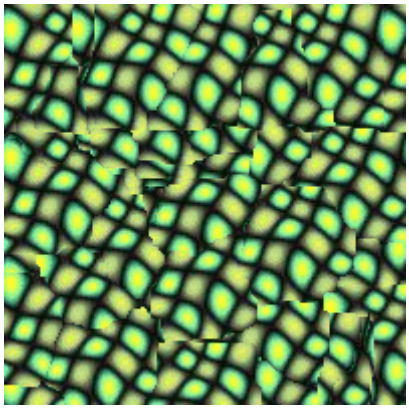
引入梯度信息的比较

引入梯度信息前的割，花费15秒完成。

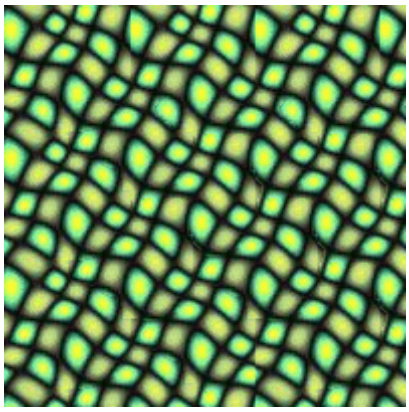
引入梯度信息后的割，花费7秒完成。

引入像素填充策略的比较

在使用像素填充策略前，填充下面的图片需要花费290秒，效果也不够好。

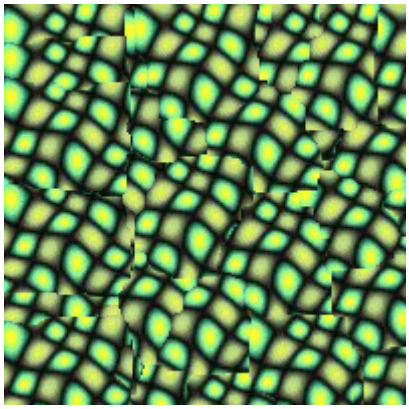


加入像素填充策略之后，效果如下。

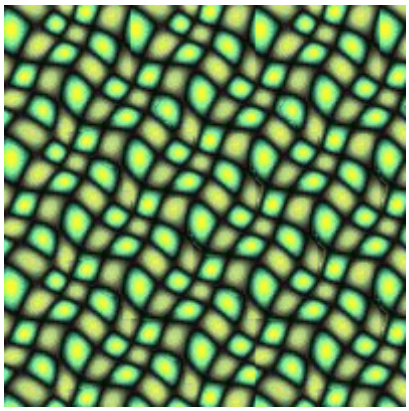


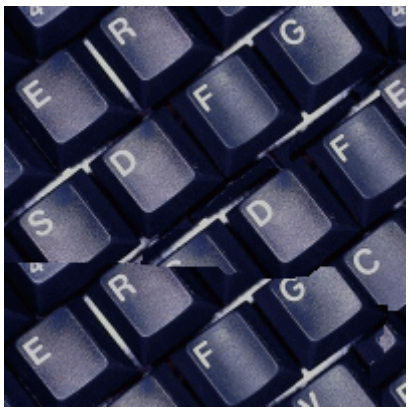
使用3种偏移算法给出的结果

Random：

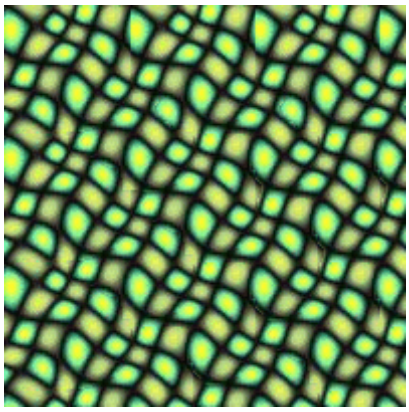


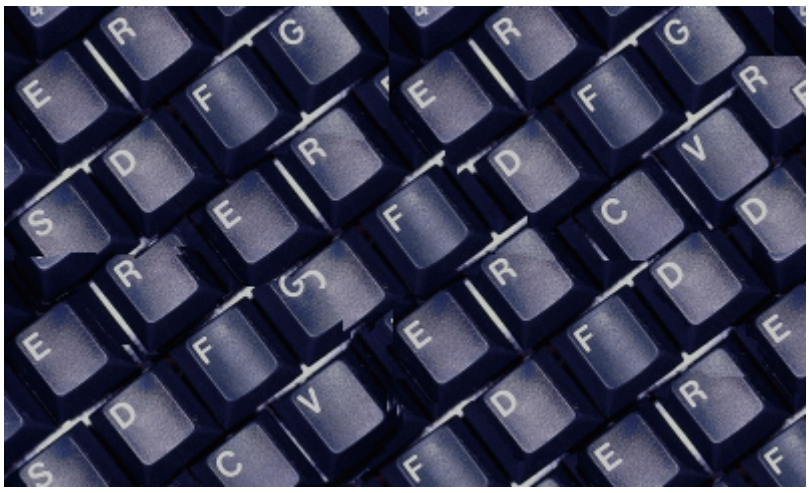
Best Patch:





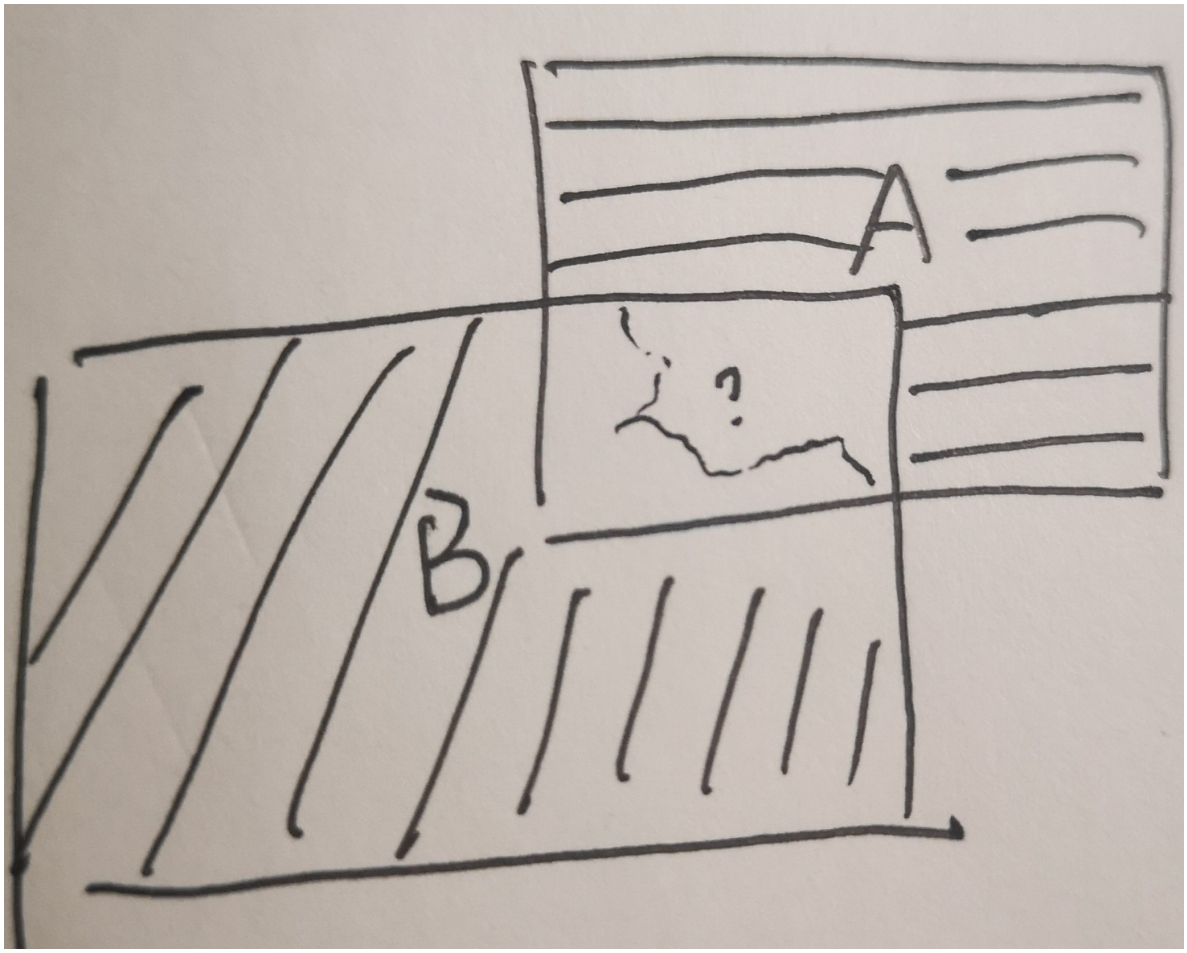
Best Pos:





从结果上来看，最好的结果都来自于Best_Pos模式。但相比于论文中的结果，实验结果还是差了一些，原因可能是我没有很好的fine-tune算法。论文中的图片patch位置摆放应该经过一定的设置，例如keyboard结果中除了左上角摆的原图，只有DFCV四个字母；草莓结果中图片patch基本是从左到右、从上到下摆放的；没有Rotation & Mirroring等等。

举一种效果不好的例子而言，如果图中有两个正方形键A和B，B来自新的patch, 其中斜线部分是新覆盖住的点，横线部分是原有的像素，像下图所示：



那么无论怎么割，A和B的结合都应该会成为奇形怪状的图形。

4 Honor Code

这篇论文与seam_carve作业不同，它只描述了算法的一个大体思路，许多细节需要我们自己脑补，比如Best_Patch模式如何选择patch的大小和位置，怎样保证在一定时间内像素能够填满矩形区域，边界怎么处理等等。为此我与计76 周子栋同学进行了很长时间的讨论，确定了这些细节的实现方式。他也教了我FFT和最小割在OI中的常规写法。在此非常感谢他！