数字图像处理大作业实验报告

1. 实验原理

本实验完成的内容如下：

设计一个算法，输入一张原始图片，自行指定图片长宽比后，考虑像素的差异化处理，输出改变长宽比的图像。

算法基本思想：

算法基本思想参照了论文 Avidan,Shai,andArielShamir."Seamcarvingforcontent-awareimage resizing."中所述的，通过考虑某条缝的“像素能量”来进行图像缩放。

图像的能量函数是基于梯度的。梯度大的地方图像内容变化大、纹理丰富，一般是重要内容，删除梯度大的缝隙对图片影响大；梯度小的地方图像内容变化小，比如草地、蓝天、海水，一般是重复内容，删除梯度小的缝隙对图片影响较小。可以看出，梯度大小与能量大小应该成正比，定义能量函数如下：

$$e(I) = \left|\frac{\partial}{\partial x}I\right| + \left|\frac{\partial}{\partial y}I\right|$$

一条seam定义为像素从上至下或从左至右的连接路径。以从上至下的seam为例，从每一行选取一个像素，相邻两行的连接线只能是当前行像素位置的八连通域子集。一条seam的能量为这条seam上每一个像素的能量之和。

对于缩小图像，我们需要删除多条能量最小的seam，对于放大图像，我们需要复制多条能量最小的seam。本实验采用动态规划的方法来寻找能量最小的seam。以从上至下的seam为例，若一个像素的横 坐标为x，那么上一行组成其seam的像素的横坐标只能是x-1，x或x+1，因此有动态规划方程：

$$dp(i,j) = energy(i,j) + min(dp(i-1,j-1), dp(i-1,j), dp(i-1,j+1))$$

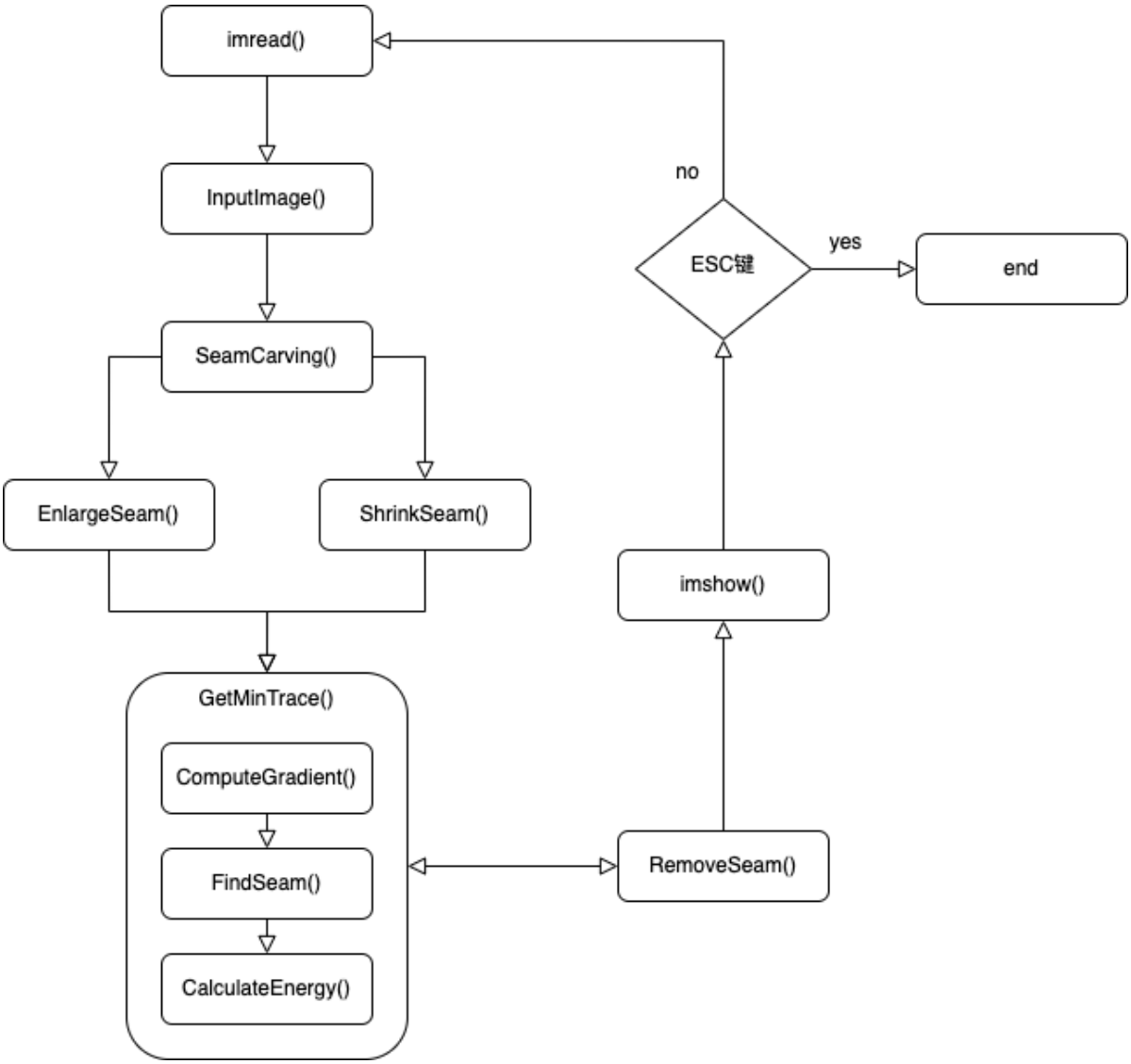
缩小图像时，算法不断重复计算能量图——动态规划寻找最小能量seam——移除最小能量seam这个过 程；放大图像时，算法先仿照缩小图像时的过程，找到能量最小的n条seam，然后复制它们，以避免持 续复制原图中同一条能量最小的seam。

创新点：

1. 经实验中实践发现，该算法在处理风景图等前景内容较为简单的图片时效果较好，但是在放缩边缘形状较为特殊的图片时（例如本实验测试例中使用了一副带有牛的图片，其中牛的身体曲线较为复杂），图片放缩后部分区域可能出现锯齿现象。因此，在处理图像后，使用双边滤波方法平滑图像，保留边缘的同时去除噪声，使放缩后的图像在视觉上更为美观。具体编码实现在函数main中。
2. 在算法编码实现中合并了能量图计算步骤和动态规划步骤。在原本的算法设计中，每次寻找最小能量的seam时，先使用O(n)复杂度的循环遍历全图，计算出每个像素的能量，然后再次使用O(n)复 杂度的循环遍历图像进行动态规划，其中n为图像像素点的个数。本实验将它们放至同一个循环中 进行，降低了算法时间复杂度。具体编码实现在函数CalculateEnergy中。

2. 实验步骤

本实验采用c++语言编码实现，调用了opencv库的部分方法与数据结构，如imread()，waitkey()，Mat.at()等。




```
int main()
{
    读图片，保存原图以便后续重新键入尺寸;
    while(1)
    {
        //初始化工作，展示图片，提示用户键入图像新尺寸
        InputImage();
        //比较用户输入的尺寸与原始图片尺寸，判断是放大还是缩小，先处理列，再处理行
        SeamCarving();
        //使用双边滤波方法平滑图像
        bilateralFilter();
        //算法运行结束，输出运行时间和运行结果。如果按下ESC键，退出，按其他任何键，可再次输入
尺寸
    }
}

/*放大图像，增加count这么多条seam，参数flag = C代表增加竖直方向的seam，flag = R代表增加水平
方向的seam*/
void EnlargeSeam()
{
```

```
    Mat trace[MAXNUM];
    //如果增加水平方向的seam，将图像旋转90度处理即可
    for(int i = 0;i < count;i++)
    {
        //找出count条能量最小的seam，记录在trace中
        Mat min_Trace = GetMinTrace();
        min_Trace.copyTo(trace[i]);
        //删掉图中的这条seam，接着寻找下一条能量最小的seam
        RemoveSeam();
    }
    利用trace扩大图像，注意调整trace中seam的相对位置
    //如果增加水平方向的seam，将图像旋转90度
}

/*删除一条seam，参数flag = C代表删除一条竖直方向的seam，flag = R代表删除一条水平方向的
seam*/
void ShrinkSeam()
{
    //如果删除一条水平方向的seam，将图像旋转90度处理即可
    Mat min_Trace = GetMinTrace();
    RemoveSeam();
    //如果删除一条水平方向的seam，将图像旋转90度
}

/*返回图中能量最小的seam，以矩阵形式记录*/
Mat GetMinTrace()
{
    //彩色图像转换为灰度图像
    cvtColor(src, dst, COLOR_BGR2GRAY);
    //计算图像梯度
    ComputeGradient();
    //通过动态规划用能量图找到能量最小的seam
    FindSeam();
    return min_Trace;
}

/*计算一副灰度图每个像素的能量函数，生成能量图*/
void ComputeGradient()
{
    //求水平梯度所使用的卷积核
    Mat kernel_H = (Mat_<float>(3, 3) <<0, 0, 0, 0, 1, -1, 0, 0, 0);
    //求垂直梯度所使用的卷积核
    Mat kernel_V = (Mat_<float>(3, 3) <<0, 0, 0, 0, 1, 0, 0, -1, 0);
    //水平与垂直滤波结果的绝对值相加，可以得到梯度大小
    filter2D();
    add(abs(gradiant_H), abs(gradiant_V), dst);
}
```

3. 实验结果

实验环境：

CPU: Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz 1.80 GHz RAM 8.00GB

GPU: GeForce MX250

测试例一（前景较为复杂）：

原图尺寸：213 × 320

改变尺寸为：180 × 380 处理时间：7.638s

改变尺寸为：100 × 200 处理时间：10.892s

测试例二（前景较为简单）：

原图尺寸：384 × 512

改变尺寸为：300 × 470 处理时间：25.673s

改变尺寸为：200 × 400 处理时间：51.182s

改变尺寸为：500 × 500 处理时间：32.057s