武汉大学计算机学院
本科生课程设计报告

基于 SIFT 算法的图像无缝拼接程序

专 业 名 称　：软件工程

课 程 名 称　：Windows 编程实验

指 导 教 师　：刘浩文　讲师

学 生 学 号　：2016302580055

学 生 姓 名　：了然

二〇一八年十一月

摘　要

本次试验的主要目的是实现一个能够无缝拼接有重叠区域图像的应用程序。在实现过程中，我利用了 SIFT 算法寻找图像之间的匹配点，进而根据多组匹配点的坐标计算出两张图片间的仿射变换模式，并用矩阵的形式表示出来。最后利用放射矩阵对原图像进行映射，将两张图片映射到一起，进而达到拼合图像的效果。在实验过程中，我使用了 WPF 技术构建可视化界面，并且利用动态链接库的知识，从托管类型的 C#代码中调用非托管的 OpenCV（C++）代码，从而极大的简化了代码。同时，我还实现了在 GUI 界面中对所选择图片进行裁剪、放缩的功能，并支持将修改过后的图片进行保存，使得应用程序在使用时有很高的流畅度，做到了对用户友好。

关键词：SIFT 算法；WPF 窗体；仿射变换；图像拼合；OpenCV

目　录

1　实验目的和意义

1.1 实验目的……4

1.2 实验意义……4

2　算法原理

2.1 概述……5

2.2 暴力枚举算法……5

2.3 SIFT 算法……7

3　图像拼合

3.1 概述……8

3.2 变换矩阵……8

3.3 坐标变换……8

3.4 优化方案……9

4　图像修饰

4.1 概述……9

4.2 图像剪裁……10

4.3 图像缩放……10

4.4 图像剪裁……11

5　成果展示

5.1 概述……12

5.2 支持多种重叠……12

5.3 支持广义重叠图像……13

参考文献 …… 14

1　实验目的和意义

1.1　实验目的

通过一个实验项目的形式,综合运用本学期所学关于 Windows 平台上的特性，用编程的方式解决实际问题，与所学知识相互印证，以求知识与技能的融会贯通以及更深层次的领悟。

1.2　实验意义

在实践中了解托管与非托管代码的区别，学习使用 OpenCV 以及相关的数字图像知识，并与 WPF 窗体技术相结合，制作出算法高效、用户友好、健壮性强的 GUI 应用程序。

2 算法原理

2.1 概述

本项目中需要处理的是两张有着部分重叠区域的图片。我们希望将这两张图片完美无缝的拼合起来，形成一张更加全面的图片。

在这里首先我们需要定义“部分重叠”的概念。

从狭义上讲，这种重叠是一种在理论上完美的重叠，即重叠区域的每一个对应坐标点的颜色数据都完全相同，如果所处理的是彩色图像，则 RGB 三个通道相对应的数值都应该一致。这种狭义上的重叠限制很严格，能够完美符合要求的图片组不易寻找。最简单的例子就是截取同一张图片的不同区域，只要这些区域有重合的部分，那么这组图片就能够满足这种狭义上严格的定义。

从广义上讲，这种重叠可以是一种视觉上的观感，表现为宏观层面的一致性，而在微观图像的数字表征上并不要求完全一致，也就是说是一种带有模糊性质的概念。生活中，用摄影设备对同一事物连续拍摄的若干张相似度很高的照片，或者是以摄影设备为轴心拍摄的一组照片都有这样的性质。将这样的图片拼接起来，就能够形成所谓的全景照片，能够从更加广阔的视角展现事物，给观者带来更加震撼的视觉体验。显然这种广义上的重叠在现实中拥有更高的应用价值。

在本次实验中，我从狭义上的概念出发，逐步泛华、扩展。力图找到更加通用的方法解决广义上的问题。

2.2 暴力枚举算法

第一步考虑如何寻找两张图片匹配点的算法。

首先只考虑狭义上的重叠概念，即重叠意味着对应区域所有位置的数值应该完全一致。在这样严格的限制下，我们需要做的其实已经变简单了很多，只需要在两张图之间不断的对比，直至寻找到这个完全一致的匹配点即可。

一个很自然的思路就是采用暴力穷举的方式来寻找这个匹配区域。在实际操作过程中，我们可以设定一个边长为 m 的正方形核，m 可以取比较小的值，比如说 3 或者 5。我们认为如果在这个核的范围内两张图片上的数值完全相同，则就可以认为找到了一个匹配点。所以就在两张图片上分别遍历，以试图找到所有可能的匹配点。

然而这样的算法无疑是非常低效的，设图片的边长为 n，那么如果不做任何优化，单纯的只是遍历核在两张图中所有可能的位置的话，算法的复杂度将是 $O(n^4m^2)$ 级别的，这显然是无法容忍的灾难型算法。

实际上可以考虑借鉴用于字符串快速匹配的 KMP 算法，这个算法用于解决在一个字符串中匹配另一个字符串的问题，设两个字符串的长度分别为 m 和 n，KMP 将算法的复杂度从暴力穷举算法的 O(nm)降低到 O(m+n)。按照这个思路，把核的对比类比为字符串的快速匹配，那么至少我们能把这个算法优化到 $O((m+n)n^3m)$。

经过优化，这样的算法效率有所提升，但仍然过于低效。且 m 的选取是一个很难把握的问题，如果 m 过小，可能会匹配到大量看似成功的匹配点，但其实这些只是因为所选取的核过小，没有暴露出两幅图片之间的差别而已。而如果 m 的数值过大，则有可能让运算更加缓慢，并且很可能错过真正的匹配点。

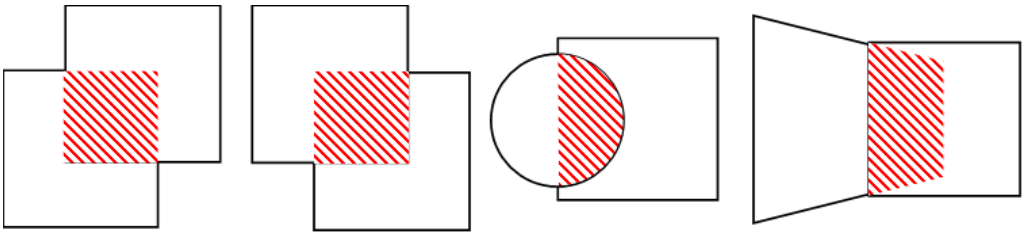

图 2.1 多种多样的重叠形式

而另一个优化的思路是基于图片的形状特点的。因为我们一般而言所处理的图片都是矩形的，所以完全可以采用更加简单的遍历方式。我们只需要枚举第一张图片的左上角在第二张图片中对应的位置即可。这样就完全抛弃了上文所提出的核的概念，算法的复杂度则是 $O(n^4)$。这种做法的复杂度虽然有所降低，但依然难以接受。并且这种算法只考虑到了如图 2.1(a)和(b)中，重叠区域为矩形的情况。对于更加一般的重叠形式没有考虑，对如图 2.1(c)和(d)这样的圆形、梯形重叠区无能为力。

上述讨论的三种算法各有特色，但有一共同的弱点，即只能应用于狭义上的重叠区域判定，不能处理广义重叠区域的例子，无法实现全景照片等应用。

2.3 SIFT 算法

SIFT 是尺度不变特征转换(Scale-invariant feature transform)的简称。这是一种在计算机视觉领域，用来侦测与描述影像中的局部性特征的算法。简要来说，它在空间尺度中寻找极值点，并提取出其位置、尺度、旋转不变量等关键特征并加以分析。此算法由 David Lowe 在 1999 年所发表，2004 年完善总结。其应用范围包含物体辨识、机器人地图感知与导航、影像缝合、3D 模型建立、手势辨识、影像追踪等。

SIFT 利用了物体上的一些局部特征与影像的大小和旋转无关，并且对于光线、噪声、微视角改变的容忍度也相当高。基于这些特性，它们是高度显著而且相对容易撷取。在基数庞大的特征数据库中，很容易利用这些特征辨识物体，而且鲜有误认。并且 SIFT 算法非常高效，在现今的电脑硬件速度下和小型的特征数据库条件下，辨识速度可接近即时运算。

Lowe 将 SIFT 算法分解为如下四步：

1. 尺度空间极值检测：搜索所有尺度上的图像位置。通过高斯微分函数来识别潜在的对于尺度和旋转不变的兴趣点。

2. 关键点定位：在每个候选的位置上，通过一个拟合精细的模型来确定位置和尺度。关键点的选择依据于它们的稳定程度。

3. 方向确定：基于图像局部的梯度方向，分配给每个关键点位置一个或多个方向。所有后面的对图像数据的操作都相对于关键点的方向、尺度和位置进行变换，从而提供对于这些变换的不变性。

4. 关键点描述：在每个关键点周围的邻域内，在选定的尺度上测量图像局部的梯度。这些梯度被变换成一种表示，这种表示对局部变形和光照变化容忍度较高。

SIFT 算法可以很高效的检测出两幅图片之间的匹配点。并且不过对于狭义上的匹配拥有近乎 100%正确率，还能应用于广义上的匹配，即允许两幅图片重叠区域的形变以及光照变化、角度变化等等。我按照算法给出的一组匹配点的可靠性进行排序，择优选取其中最好的一部分用于进一步的计算。OpenCV 中已经实现了成熟的 SIFT 算法，直接调用即可。

3　图像拼合

3.1　概述

在这一章节，我利用由上述算法得到的匹配点进行图像的映射，最终将两张图片映射到统一参考系中，实现图像的拼接。

3.2　变换矩阵

通过上述的 SIFT 算法，我已经取得了一组比较可靠的匹配点。现在就是要利用这一组匹配点，寻找到两张图片之间的映射关系。

一般我们选取一张图像的左上角为原点建立坐标轴，以宽度方向建立 x 轴，以高度方向建立 y 轴。这样一对匹配点其实就是代表着一对坐标，这两个坐标是这两个点在各自图像参考系下的坐标。而只要能够把他们映射到同一个参考系中，我们就可以做到图像拼合的功能了。实际上就是找到两个参考系之间的映射关系。

这里我们用到了线性代数与数字图像中的知识。从二维参考系到另一个二维参考系中的映射可以用一个 3x3 的矩阵表示。比如说下面所示的矩阵就代表了平移变换——沿 x 轴平移 a 个单位，沿 y 轴平移 b 个单位。类似的变换矩阵还有很多种，包括旋转、缩放、偏移等等。并且多个变换矩阵可以通过相乘的形式叠加，形成更加复杂的变换矩阵。

$$\begin{bmatrix}1 & 0 & a\\ 0 & 1 & b\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}x\\ y\\ 1\end{bmatrix}=\begin{bmatrix}x+a\\ y+b\\ 1\end{bmatrix}$$

而反过来，已知若干匹配点对的坐标，就可以建立方程组，在数学上求出这个变换矩阵。这可以通过最小二乘法的方式快速求出近似解。这一步在 OpenCV 中已经实现了相应的函数，直接调用即可。

3.3　坐标变换

在实际中，我选择将第二幅图片映射到第一幅图片中的坐标系中。然后将两张图片自然的重叠在一起，重叠区域全部取自第一幅图片中的。

这里有一点需要注意的。由于映射关系多种多样，很有可能在把第二幅图片映射过去后，部分坐标值出现了负数。而在我们缩设立的参考系中，坐标原点代表一张图片的左上角，显然负数坐标是无意义的，无法在最终图像中呈现出来。因此这种情况需要特殊处理。

我采取了一种简单的方式，来判断这种情况并加以处理。首先我把第二幅图片的左上角、左下角、右上角、右下角作为判断的主要依据，首先加以变换到第一幅图片中的坐标系中。如果这四个变换后的点的 x 坐标或者 y 坐标出现任何一个是负值的话。对上一小节求出的变换矩阵做微调，具体来说就是给这个变换矩阵叠加一个平移变换，将坐标值为负数部分的图像平移到正数范围内。这样就可以保证两幅图片拼合完成后可以完美的全部显示出来。

3.4 优化方案

目前我所描述的这种最直接的拼接方式有一个小小的不足。可能会在两种图片的拼接处形成一根明显的分界线，有明显的拼接痕迹。

图 3.1 明显的拼接痕迹

一个可行的解决方案是，提前计算分界线的位置，然后在叠加分界线附近图像的时候，设定一个可变的权重，让两张图片加权叠加，实现平稳过度。具体而言，假设分界线的左侧是图一，右侧是图二，则可以将分界线上的权重设置为图一图二各 50%，然后向左图一的权重逐渐增加，向右则把图二的权重不断增加。

4 图片修饰

4.1 概述

为了给用户更加流畅的使用体验，我提供了在界面中修剪所选取的图片的功能。用户可以在程序界面中自由的对所选图片进行剪裁、缩放和保存三种操作

4.2 图片剪裁

在点击“Crop Image”按钮后，会弹出一个新的窗体，窗口内是用户希望剪裁的图片。用户只需要按住鼠标左键，然后拖动鼠标，一个红色的长方形框会随着鼠标的拖动而出现，代表着用户选择的裁剪区域。当用户松开鼠标时，这个红色矩形框所代表的图像就会被剪裁下来，其他图像被抛弃。用户还可以连续进行多次裁剪。最后用户只需要关闭这个窗体，代表完成裁剪操作。

图 4.1 剪裁图像

4.3 图片缩放

图片的缩放功能与剪裁功能很相似，在点击“Crop Image”按钮后，会弹出一个新的窗体。在窗体的上次有一个可以拖动的滑动条，用户只要拖动它就可以自由的完成对图片的放大、缩小效果。滑动条上显示的数字代表百分比，初始的时候为 100%，即原图像大小，最大可以拖动到 200%，即讲原图像放大一倍。最后用户只需要关闭这个窗体，代表完成缩放操作。并且支持与缩放操作同时进行，允许混合使用两种操作。

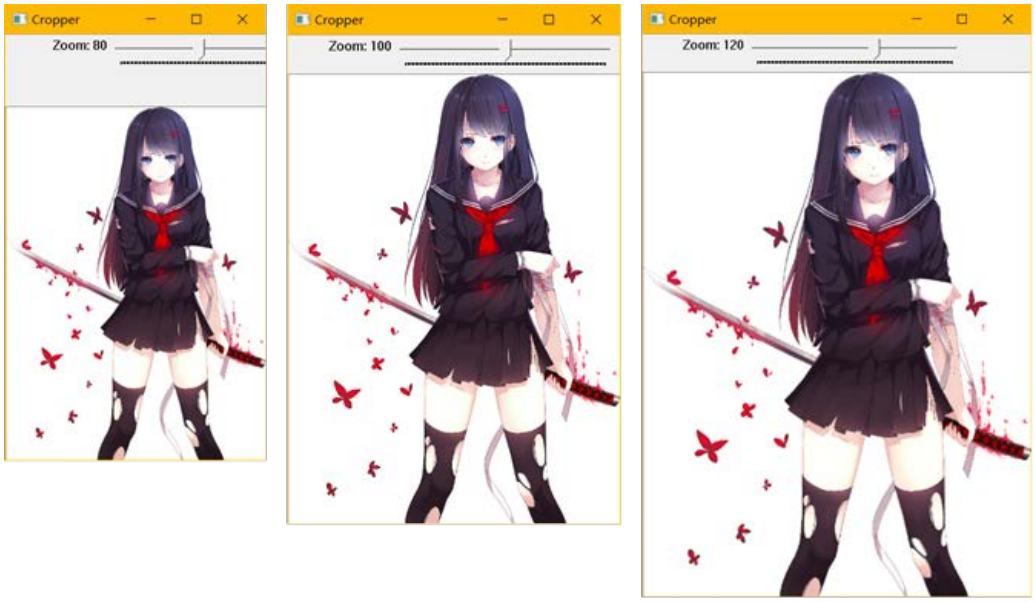

图 4.2 放缩图像

4.4 图片保存

用户只需要点击在图像预览框旁边的“Save As”按钮就可以完成保存图像的操作。被保存的图片可以使原图片，即相当于将原图片复制，也可以是讲过上述两种操作修饰过后的图片。目前支持 JPG 和 PNG 两种格式的保存操作，用户可以在选择保存路径的时候通过修改扩展名的方式选择保存格式。

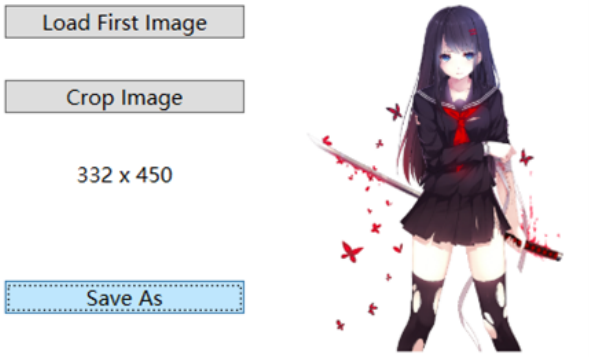

图 4.3 保存图像

5　成果展示

5.1　概述

在这一章节，展示了一组由我的程序所完成的无缝拼接效果。

5.2　支持多种重叠模式

图 2.1 所示的四种重叠模式我的程序都能完美的识别出来。

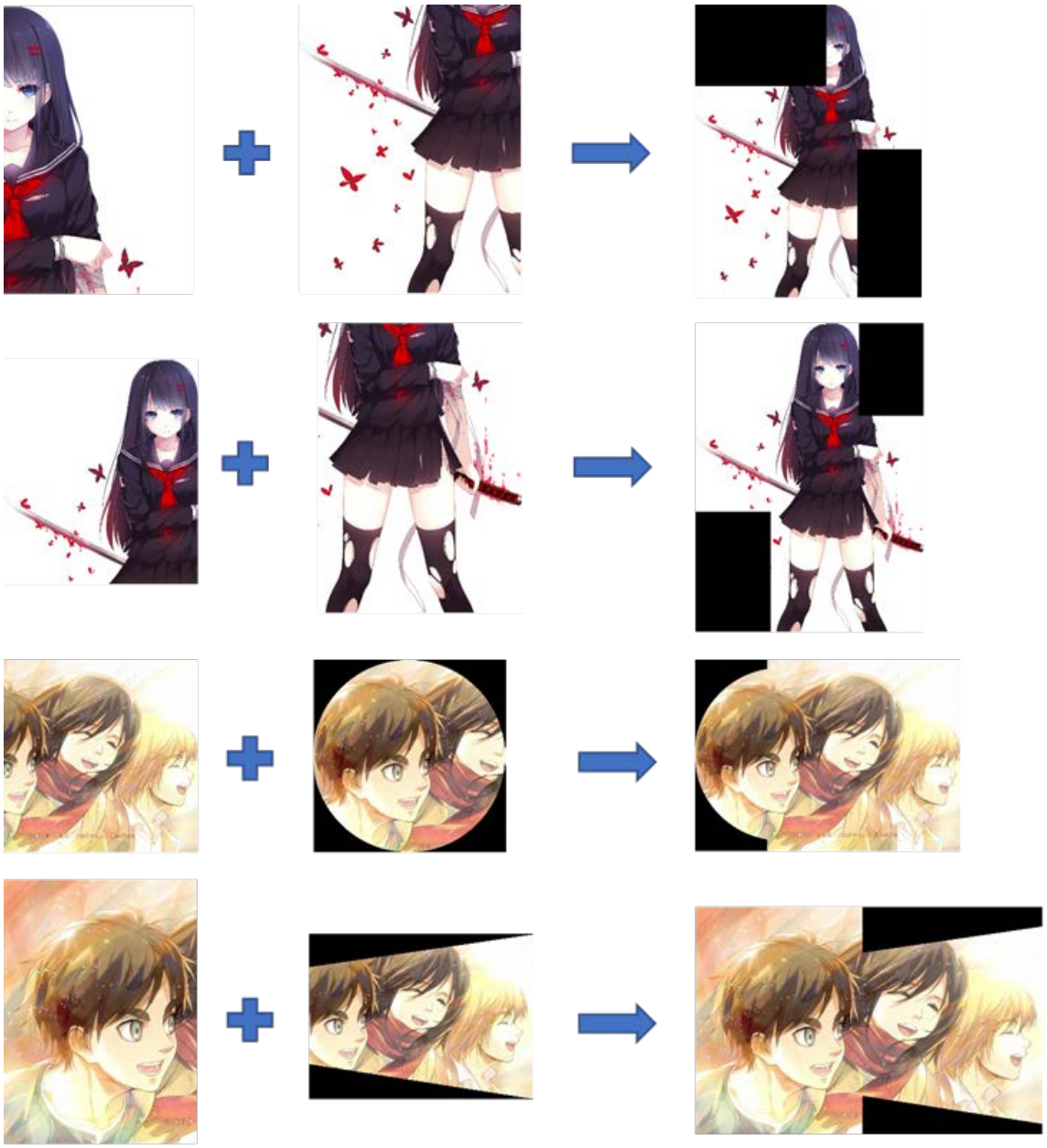

图 5.1　支持多种重叠形式

5.3 支持广义重叠图像

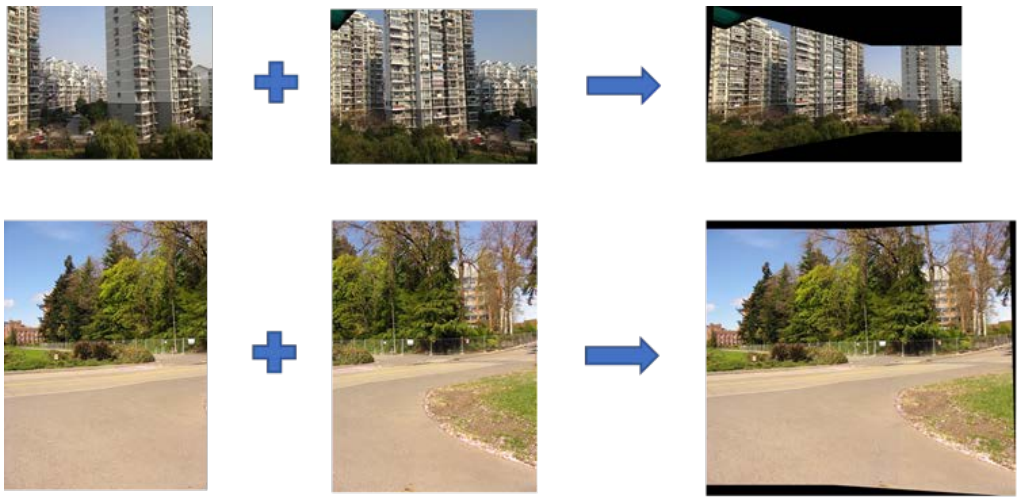

图5.2 支持拼接广义重叠图像

参考文献

[1] OpenCV

[2] OpenCvSharp - .NET Framework wrapper for OpenCV

[3] Getting Started With OpenCvSharp 3

[4] [Cpp] SIFT and SURF

[5] OpenCV GUI 基本操作，回调函数，进度条，裁剪图像等

[6] SIFT 算法