

**毕业论文（设计）正文**

**[](http://bysj.zjgsu.edu.cn/)**

# 基于移动最小二乘的图像Warping

**Image Warping base on Moving Least Squares**

**学 院：计算机与信息工程学院**

**专 业：计算机科学与技术**

**班 级：计科1201**

**学 　 号：1212300232**

**学生姓名：杨呈鸿**

**指导教师：杨文武**

**二○一六 年 五 月**

# 基于移动最小二乘的图像Warping

**摘 要：**在本文中，我们提供了基于使用不同类型的移动最小二乘法模型的变形函数，例如仿射变换、相似变换和刚性变换。这些变形十分逼真，给予用户操纵真实物体的感觉。同时，我们也提供了基于点集或者基于特征线段的变形方法，后者对于图像中的曲线和轮廓十分有用。对于以上的每个方法，我们都给出了封闭形式的解以快速的得到变形结果。

**关键词：图像变形，移动最小二乘，刚性变换**

**Image Warping base on Moving Least Squares**

**Abstract:** In this paper, we provide the different type use of Moving Least Squares deformation function, such as affine transformation, similarity and rigid transformations. These deformations very realistic, giving the user the feeling of manipulation of real objects. We also offer deformations base on both a set of points and a set of line segments, the latter is useful for controlling curves and profiles present in the image. For each of the above methods, we provide simple closeed-form solutions that create fast deformations.

**Keywords:** Deformations, Moving Least Squares, rigid transformations

**目录**

[基于移动最小二乘的图像Warping 1](#_Toc451461414)

[基于移动最小二乘的图像Warping 2](#_Toc451461415)

[第1章 绪 论 5](#_Toc451461416)

[1.1 课题背景 5](#_Toc451461417)

[1.2 国内外研究现状 5](#_Toc451461418)

[1.3 论文组织结构 7](#_Toc451461419)

[第2章 基于点的移动最小二乘图像变形方法 9](#_Toc451461420)

[2.1 基于点的移动最小二乘图像变形方法的数学模型 9](#_Toc451461421)

[2.2 仿射变换 10](#_Toc451461422)

[2.3 相似变换 11](#_Toc451461423)

[2.4 刚性变换 12](#_Toc451461424)

[2.5 本章小结 12](#_Toc451461425)

[第3章 基于特征线段的移动最小二乘图像变形方法 14](#_Toc451461426)

[3.1 基于曲线的移动最小二乘法图像变形方法的数学模型 14](#_Toc451461427)

[3.2 仿射变换 15](#_Toc451461428)

[3.3 相似变换 16](#_Toc451461430)

[3.4 刚性变换 17](#_Toc451461431)

[3.5 本章小结 18](#_Toc451461432)

[第4章 图像变形系统实现 19](#_Toc451461433)

[4.1 系统环境 19](#_Toc451461434)

[4.1.1 开发环境Visual Studio 2008 19](#_Toc451461435)

[4.1.2 C++ 19](#_Toc451461436)

[4.1.3 应用程序开发框架Qt 20](#_Toc451461437)

[4.1.4 Git 20](#_Toc451461438)

[4.2 系统主要算法实现 20](#_Toc451461439)

[4.2.1 初始化变形前图像和变形前的特征线 20](#_Toc451461440)

[4.2.2 输入变形后的特征线，进行变形 27](#_Toc451461441)

[4.2.3 获得变形后图像 30](#_Toc451461442)

[4.3 本章总结 31](#_Toc451461443)

[总结 32](#_Toc451461444)

[参考文献 33](#_Toc451461445)

[致 谢 35](#_Toc451461446)

# 绪 论

## 课题背景

随着信息技术的发展，人们开始越来越多的经过计算机来获取、利用、处理各种信息，而图像正是获取信息的重要方法之一。计算机图形学是一门研究如何使用计算机表示、生成、处理和显示图形的一门学科。随着计算机图形学算法的发展，硬件的进步，计算机图形学已经成为了计算机科学与技术中十分重要的学科分支，并在诸如医疗成像、计算机辅助设计、虚拟现实等领域得到了广泛的应用。

作为计算机图形学的一方面的应用，图像变形是一种在两幅图像之间产生一系列连续的期间过渡图像的技术,使得产生起始图像平滑的过渡到变形图像的效果。作为一种图像处理技术,图像变形技术已被广泛应用在虚拟现实、医疗成像、地理信息、气象预报、计算机辅助设计、计算机动画、人脸识别以及影视娱乐等各个领域。在医学成像领域，图像变形是一种十分重要的应用，对于很多的手术，通过图像变形技术对手术图像做出平滑变形可以对其进行模拟以及假设，对于医学研究和治疗有着很大的帮助。而在影视业中，虚拟角色、虚拟现实、人脸记忆合成以及常见的从某一个脸到另一个脸的变化，使它产生喜、怒、哀、乐等复杂表情的变化，从而使得图像更加逼真、更有生气。在动画领域，通过控制人物或者物体的变化，从而实现动画效果。在刑侦系统中，变形技术可以对于描述人脸，构造人脸模型，结合数字图像处理可以从监控录像中进行人脸识别和寻找。气象预报中，对若干幅气象云图进行图像变形处理以生成连续的气象云图，从而可以描述云图的运动状态和连续性，预测未来趋势。在地理信息系统的数字地形模型中，使用图像变形技术将二维图像叠加到三维地形图上，以生成更加准确的数字地形模型。

图像变形技术是计算机图形学的重要内容之一，是当今最为活跃的学科，是一个重要而有研究意义的领域，图像变形技术的研究成果被广泛应用于气象预测、计算机动画、影视娱乐、医学成像、地理信息系统、刑侦系统等各种领域。因此，对图像变形技术的研究既有重要的理论意义，也有很高的实际应用价值。

## 国内外研究现状

对图像变形方法的研究最早认为是从20世纪60年代开始，当时生成一幅数字图像到另一幅图像的方法一般使用一种交叉溶解的方法实现[1]，这种方法通过线性插值的方法从而实现从起始图像到目标图像的转化，严格的来说，这种方法只是图像转变的方法，远远达不到图像变形的要求，效果很差，难以令人满意，不过当时对于图像的变形方法也并没有足够的研究和重视。

20世纪80年代末，Smythe[2]在电影制作中使用了网格扭曲的变形方法。这种方法使用了基于网格的技术，利用网格来控制变形，采用溶解技术以得到变形后的图像。

图像变形中的一个重要的问题是控制句柄。改变这些控制句柄，从而对图像进行操纵，使图像产生变形[3]。根据控制句柄的不同，可以将图像变形分为基于网格、基于域、基于点、基于控制曲线或线段的图像变形。

基于网格的技术有很多，例如自由形式变形[5-7]，采用了二元三次样条函数来创建C^2的变形函数。通常这些方法需要设置对应图像特征的控制点，这些控制点的设置对于用户来说非常麻烦，不利于操纵控制点进行变形。

在此基础上，Kho和Garland[8]等人提出了一种基于非规则多边形网格的变形技术。使用这种技术，用户可以在平面上操纵曲线来创建变形。这种技术操作简单、变形效果直观。

Beier等人[9]的技术在则更进一步，允许用户使用控制线段来控制变形。用户只需要在两幅图像中标定特征线，就能产生相当不错的从起始图像到目标图像的图像变形效果。但是它的效率不高，运算复杂度非常高，也就是说随着特征的复杂度提升，线段数量的增多，运算时间急剧上升。

Igarashi等人[10]提出的图像变形技术是一种基于点的图像变形技术，通过在计算机动画领域的研究，使得图像变形极可能刚性可以提升图像变形的效果。它们的做法是将输入图像先进行三角剖分，得到未知变量个数等于三角顶点数之和的线性方程组。求解这个线性方程组，得到变形函数。应用变形函数就可以得到从源图像到目标图像的变形结果。

在基于域的图像变形技术中，值得一提的是Han等人[11]提出了一种基于样条曲线的变形技术。该算法能够提取出样条曲线上的点，并对每个点加入了影响因子。这样的做法将限制局限在了某一个域中，从而该使得该技术可以做到精确控制图像变形。

变形前和变形后的图像中选出一些对应的点作为特征点，已知它们之间的对应关系，利用径向基函数对这种对应关系进行处理从而创建变形函数，得到起始图像到目标图像的变形结果，这种图像变形方法被称为基于径向基函数的图像变形方法。

在基于对称径向基函数的图像变形算法的基础上，李旭东等人[12]提出了一种基于非对称径向基函数的图像变形算法，它克服了原来的算法由于对称径向基函数的全局性，导致出现图像边界变形过大、变形容易在边界处产生问题的困难。

注意到网格变形方法往往无法保证网格一一对应，那么如果当网格变形到一幅图像时，可能会发生图像又重新折叠回来的情况。针对这个问题，Lee等人[13-14]提出了能量最小化的方法，从而保证了这种情况不会发生。在这个变形方法中，变形受到能量的限制。当能量最小时，能够产生非常平滑的变形。但是，这种方法的计算量太大，对于实际应用来说效果并不好。

Lee等人又提出了多层次自由变形(MFFD),这种方法比能量最小化方法更简单，更快。

1998年，Levin等人[15]提出了最小二乘法的理论模型。基于该模型，Weng等人[16]提出了一种被称为非线性最小二乘规划的算法（Nonlinear Least Squares Optimization）。该算法输入的值为一个用图形向量或者位图表示的边缘为多边形的二维图形。该算法有两个性质：变形中保持外轮廓的拉普拉斯坐标不变，变形中保持内部的局部面积不变。第一个性质可以表现图形边缘的局部细节，所以三维网格变形中，这种方法被广泛的应用。但是，对于二维变形来说，仅仅表现出了图形边缘局部细节是远远不够的，还有很多的工作需要完成。

Schaefer等人[17]在此基础上提出了一种基于移动最小二乘的变形方法，通过控制点或者控制线段对图像进行变形。在方法介绍中，它们分别对基于点、基于线段，仿射变换、相似变换和刚性变换进行了分析，对于每一种对应的情况，都进行了数学推导，并在给出了每一种情况下的易于实现的封闭形式的解。

## 论文组织结构

本文各章节的安排如下：

第一章：绪论。阐述了图像变形技术的研究背景和研究意义，并介绍了该技术的研究现状。

第二章：首先阐述基于点的移动最小二乘图像变形方法的数学模型，接着对于仿射变换、相似变换和刚性变换三种情况下的移动最小二乘图像变形方法作出推导，给出封闭形式的解。

第三章：首先建立了基于控制线段的移动最小二乘图像变形的数学模型，与第二章类似，接下推导仿射变换、相似变换和刚性变换三种情况下的变形公式，并对其优化得到封闭形式的解。

第四章：介绍图像变形系统的系统环境、系统功能和主要的算法实现。

总结：总结本文的工作。

# 基于点的移动最小二乘图像变形方法

## 基于点的移动最小二乘图像变形方法的数学模型

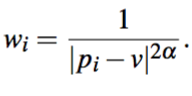
图像变形可以被认为是在变形函数作用下由未变形图像到变形图像的过程。设变形前的控制点集为*p*，变形后的控制点集为*q*，对于变形函数一定有以下性质：

1. 插值性：在f作用下*p*点直接到*q*点。即
2. 光滑性：函数应该产生光滑的变形。
3. 确定性：如果控制点*p*和控制点*q*是同一点，变形函数应该是确定的方程。即

对于图像中的任一点*v*，将*f*作用到点*v*，就可以得到变形后的图像。根据最小二乘法的模型，必然存在变形函数，使下式取得最小值：

（1）

其中和用行向量表示，为权重，表达式为：

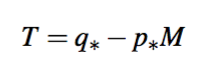


为调节参数，变形中一般取为1。由于形式和最小二乘法相同，且有着权重随着*v*的取值不同而不同的特性，因此这种方法被称为移动最小二乘法。

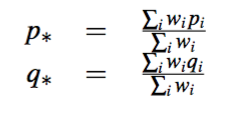
如果变形函数是仿射变换，那它可以被分为两部分：线性变换矩阵*M*和平移变换项*T*。



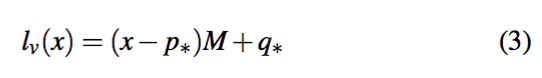
将公式（2）代入公式（1），并求最小值，也就是对的变量求导且导数等于零，那么我们得到：



其中和是加权重心：



在此基础上，我们得到了变形函数(x)的一般表达形式：



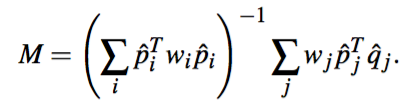
那么，等式（1）也可以被重写为：



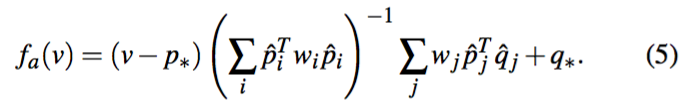
其中：，。注意到在移动最小二乘法中，矩阵*M*是一个一般化的仿射变换，包含了仿射变换的缩放、剪切、旋转等成分。通过对矩阵*M*的限制，我们可以得到仿射变换、相似变换、刚性变换的变形函数(x)。

## 仿射变换

对公式（4）求偏导令结果等于零，我们可以得到线性变换矩阵*M*的表达式：

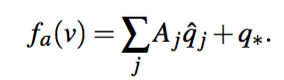


尽管线性变换矩阵*M*中包含矩阵转置的计算，但是*M*是一个2\*2的矩阵，所以它的转置非常易于计算。那么，将上式代回等式（3），我们可以得到仿射变换函数：

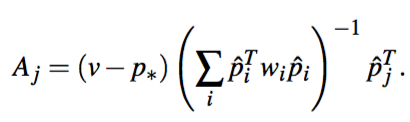


等式（5）代入图像中的每个点*v*,我们就得到变形后的新图像。

注意到，当用户控制*q*移动以得到新的变形时，点*p*是保持不变的。那么等式（5）是可以通过预计算来得到快速变形结果的。特别地，我们可以重写等式（5） ：



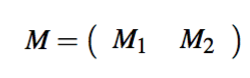
其中：



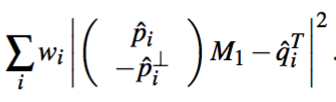
也就是说，对于给定的点*v*，中的所有值都可以通过预计算得到。

## 相似变换

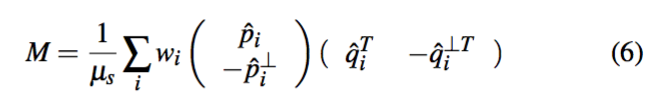
仿射变换由于存在不均匀的缩放和不均匀的剪切，使得图像的变形的真实感有所降低。相似变换是一种特殊的仿射变换，它只包括平移、旋转和同比例缩放。如果我们限定了图像只能做相似变换，那么变形矩阵*M*有着这样的性质：对于某些值有。设矩阵*M*可以被表示为块矩阵的形式：



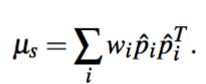
那么我们可以得到以及。我们可以得到。求解等式（4）的最小化问题可以被转化为求使得下式取得最小值：



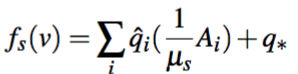
上式有唯一解，那么我们可以得到矩阵*M*的表达形式：



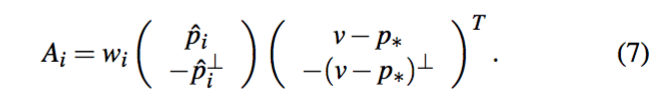
其中：



将等式（6）代回等式（4），我们得到相似变换函数：



其中的、只依赖于和*v*，可以预计算得到，其中：

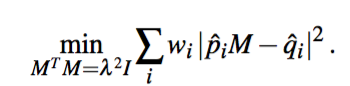


## 刚性变换

近来，Alexa等人的研究显示，为了使得变形更加的真实，对于物体的变形应该尽可能的刚性，也就是说，为了真实感的要求，我们需要物体做出刚性变换，即变形的比例往往不应该相同。传统的研究者在这个问题上一般采用非线性约束的条件去处理，但是Horn1987年的最近重复点集这篇文章中指出：在包含和的协方差矩阵的特征值和特征向量中能够找到刚性变换封闭形式的解。由以上，我们推导基于点的刚性变换的变形公式。

定理2.1：

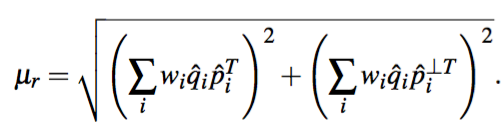
设*C*是使得相似变形函数取得最小值的矩阵，即：



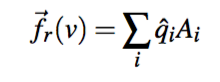
其中，*R*是旋转矩阵，是一个标量，则我们得到了旋转矩阵*R*使得刚体变换的变形函数取得最小值：



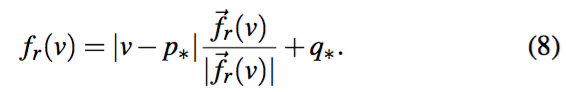
利用这个定理，我们发现刚性变换中的矩阵*M*的表达式和相似变换的基本相同，只是一个常数不同：



和相似变换函数不同的是，对于刚性变换我们难以预计算出如此多的信息。令：



其中来自于等式（7），通过上式我们可以得到刚性变换函数的表达式：



由于归一化计算，刚性变换函数的计算比相似变换更慢一些；然而，依旧是非常的快。

## 本章小结

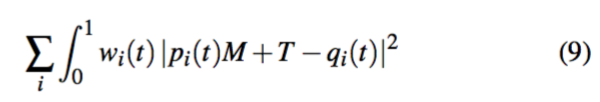
在2.1节中，整体的我们对于移动最小二乘模型进行了推导。在接下来的2.2，2.3和2.4节中，我们分别对于仿射变换，相似变换和刚性变换的情景给出了封闭形式的解，并对最后的表达式做出了利于预计算的表达。由推导过程和解的形式表达中，我们可以看到仿射变换，相似变换，刚性变换之间的关系。

# 基于特征线段的移动最小二乘图像变形方法

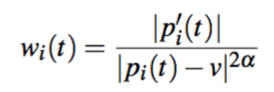
## 基于曲线的移动最小二乘法图像变形方法的数学模型

在第二节我们已经考虑了基于控制点来创建图像变形的方法。在需要诸如图像轮廓曲线精确控制的程序中，控制点不足以指定变形。一个方法是将用户控制的曲线转化了密集的点集，然后应用基于控制点的变形。然而，这种方法的计算时间和创建的控制点的数量呈正比。

我们同时希望第二节中的移动最小二乘图像变形方法能够推广到平面任意曲线。设为变形前第*i*段控制曲线，为对应的变形后的第*i*端控制曲线，其中。由第二节的结论可知，变形函数使得下式取得最小值：



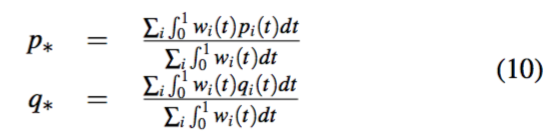
其中：



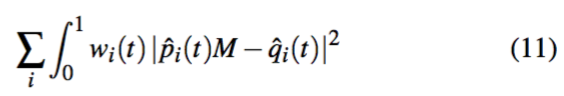
其中是的导数。求等式（9）的最小值，即对其变量求导并等于零，有：



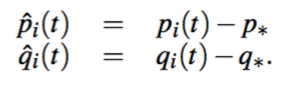
其中：



等式（9）可以改写为：



其中：



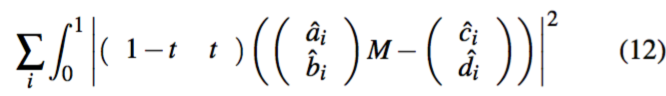
就此，我们得到和均为任意曲线的情况的解。然而等式（11）中的积分依旧十分复杂。所以，我们限制了这些函数为线段并从这些线段的端点推导出封闭形式的解。类似第二节的做法，我们先讨论仿射变换的情景，然后推导到相似变换，最后我们使用相似变换来解决刚体变换的问题。

## 仿射变换

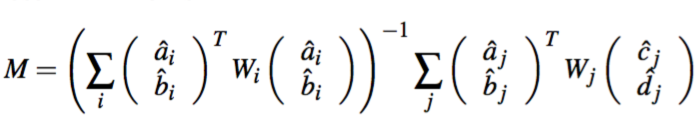
假设和均为线段，那么我们可以有下列的表示：

## 

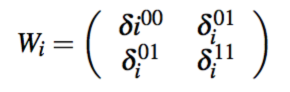
那么，等式（11）可以被重写为：



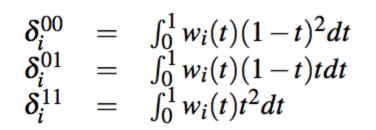
对上式最小化可以得到矩阵*M*的表达式：



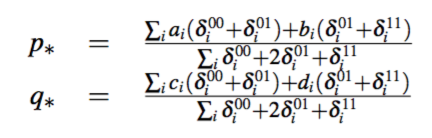
其中：



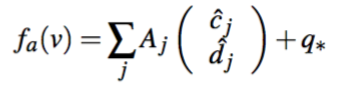
其中的表达式由给出：



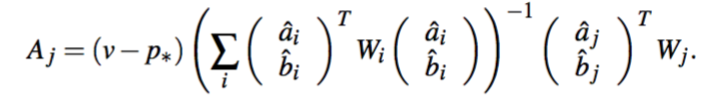
上式这些积分的封闭形式的解由的取值决定。假设上述这些积分可以用于计算等式（10）中的和，我们可以得到：



由此，我们可以得到基于特征线段的仿射变换函数：



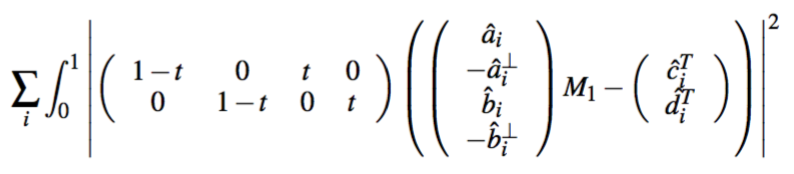
其中：



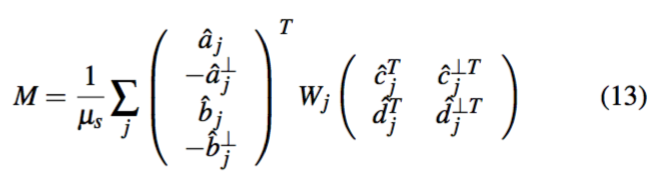
与2.2节中相似，用户控制特征线段进行变形时，变形前的线段保持固定不动。那么，的计算独立于变形后的特征线段，可以通过预计算得到。

## 相似变换

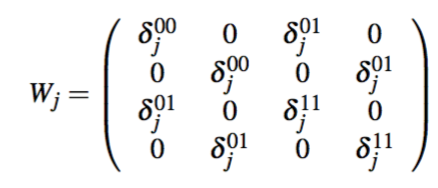
和2.3节中相似，我们限制矩阵*M*具有以下性质：对于特定值有。那么矩阵*M*可以被*M1*替代，得到下式：



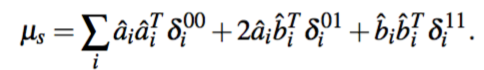
对上式求最小值，由*M1*和*M*的关系我们可以得到矩阵*M*的表达式：



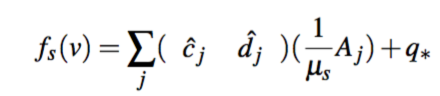
其中是权重矩阵：



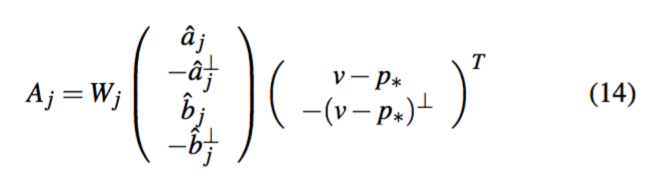
常量：



由此我们得到相似变换变形函数，它与基于控制点的相似变换变形函数的形式十分的相似：

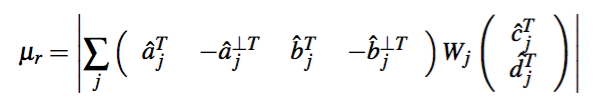


其中：

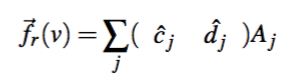


## 刚性变换

使用3.3节的结论以及定理2.1，我们得到了刚性变换变形函数的封闭形式解。矩阵*M*和等式（13）相同，当然我们要使用不同的常量满足条件。



类比于2.4节中的基于点的刚性变换，我们可以如等式（8）一样以得到刚性变换的变形函数，那么我们需要得到。



其中的值由等式（14）给出。

## 本章小结

在第三章中，由第二章基于控制点的移动最小二乘变形方法的基础，我们将这种技术扩展到了基于线段的情景。。3.1节中对于任意曲线的情景进行了推导，得到了基于曲线的移动最小二乘模型。在接下来的小节中，设曲线是由一段段是线段构成，那么在3.2，3.3，3.4节中我们分别就仿射变换、相似变换、刚性变换的情景进行了推导，并得到了它们分别的封闭形式的解。

# 图像变形系统实现

## 系统环境

图像变形系统基于C++开发，使用Visual Studio 2008作为集成IDE，搭配QT作为应用程序开发框架进行开发，Git作为版本控制工具，实现了轻量级开发的优势。系统实现了用户输入变形前图像和变形后图像，并绘制特征线后，通过移动最小二乘方法实现变形的过程。

### 开发环境Visual Studio 2008

Microsoft Visual Studio 2008，代号“Orcas”，于2007年11月19日更新到MSDN开发者平台，它是对Visual Studio 2005一次全面的升级。

Microsoft Visual Studio 2008的重点是在Windows Vista，Office 2007以及Web应用程序的开发。Microsoft Visual Studio 2008引入了很多新特性，并且重新设计调整了对象、关系型数据库、XML的访问方式，使得语言更加简洁。Visual Studio 2008的特性包括了一个基于XAML的设计器（代号”Cider”），工作流设计器，SQL设计器,XSLT调试器， JavaScript的支持，JavaScript调试支持，并发构建系统，等等。Visual Studio 2008也是最后一个支持针对Windows 2000的C ++应用程序的版本。

Visual Studio 2008 需要.NET 3.5框架和默认配置了编译程序集.NET Framework 3.5,当然它也支持让开发者选择具体哪个.NET Framework的版本运行。Visual Studio 2008也提供了各种高级开发工具，帮助开发者快速开发Windows Vista，Office 2007以及Web应用程序。

### C++

C++是在C语言的基础上开发的一种面向对象的通用编程语言， 它同时支持面向对象、泛型和过程化编程，同时还提供了底层存储器的操作支持。许多其他的编程语言都收到了C++的影响，包括C#，Java，和C的1998年以后的更新版本。

C++设计之初是偏向于系统编程、嵌入式系统和大型系统的编程，特点是高性能、高效率和灵活性。后来C++广泛使用到了其他的领域，包括桌面应用程序，服务器（如电子商务，网络搜索或SQL服务器）和高性能应用（如电话交换机或空间探测器）。

C++是一种编译性语言，很多组织都提供了它的实现，包括FSF, LLVM, Microsoft, Intel和IBM。

### 应用程序开发框架Qt

Qt是一个C++的跨平台应用程序开发框架。它被广泛用于开发底层代码库很少或基本没有变化的软件或硬件平台上运行的应用软件。Qt 使用了特殊的元对象编译器和宏，实现Qt易于扩展，组件编程的特性。

Qt拥有良好的封装机制，这使得Qt的代码模块化程度非常高，代码重用率很高，对于开发者来说非常高效。并且Qt拥有丰富的API，支持二维三维的图形渲染和OpenGL的使用。对于大量的API，Qt对此提供了详细的开发文档，提供给开发者查阅。

### Git

Git是目前世界上最先进的分布式版本控制系统，它是由Linux的创始人Linus开发的。Git迅速的称为了最流行的分布式版本控制系统，特别地，2008年Github网站上线，它为开源项目免费的提供Git存储，从而进一步推广了Git的使用，无数著名的开源项目的代码也使用Git进行管理，托管在了Github上。

CVS和SVN都是集中式的版本控制系统，版本库集中存放在中央服务器，需要先从中央服务器取得最新的版本，修改完代码后需要再将修改后的内容推送到中央服务器。集中式的版本控制系统最大的问题在于需要联网才能进行工作，一但失去网络就无法向中央服务器取得最新版本和推送修改后的内容。并且，一但中央服务器出现问题，代码数据可能出现丢失。

分布式版本控制系统在每个人的电脑中都有一个独立的完整的版本库，虽然也有中央服务器的概念，但是它仅仅作为方便交换修改的作用而已。

## 系统主要算法实现

根据第二章和第三章的论述，刚性变换函数的实现并不是非常困难；关键之处在于提升变形算法的性能。注意到我们在推导过程中将可预计算得到的部分作出了分离，这也为了程序实现和算法效率计算提供了便利。

### 初始化变形前图像和变形前的特征线

通过CLattice类输入变形前图像信息，CPolygon类输入变形前的特征线信息。对于每个点*v*，由于已知变形前的特征线的信息，可以预处理出所有的，并将信息保存在\_precalculatedArray里，供以后调用。

|  |
| --- |
| void CMLS::SetLatticeFeatureLine(CLattice\* srcGrid,const varray<CPolygon \*>& srcFeatureLine) {  // isNewInput  // 循环原图所有网格点  \_srcGrid = srcGrid;  varray<double>& arrayVertexPos = srcGrid->GetVertexPos();  for (int i=0; i<arrayVertexPos.size()/2; i++) {  Vec2 v(arrayVertexPos[ i+i ],arrayVertexPos[ i+i+1 ]);  bool isBreak = false;  Vec2 weightP;  double sumP = 0.0;  varray<double> deltaArray;  // 对于每个点v，循环每条特征线，预计算参数  for (int k=0; k<srcFeatureLine.size(); k++) {  CPolygon\* pPolygon = srcFeatureLine[k];  int iVNumber = pPolygon->GetVNumber();  bool isClosed = pPolygon->GetClosed();  varray<Vec2>& arrayVertices = pPolygon->GetPolygon();  int nEdge = isClosed ? iVNumber : iVNumber-1;  int iEdge = 0;  for(; iEdge < nEdge; iEdge ++) {  int iV1 = iEdge, iV2 = (iV1+1)%iVNumber;  // 得到line segments的两个端点Ai Bi  Vec2 ai = arrayVertices[iV1],  bi = arrayVertices[iV2];  double deltaI = Cross( ai-v , ai-bi );  double thltaI = atan( ( Dot( bi-v , bi-ai ) )/( Cross( bi-v , bi-ai ) ) )  - atan( ( Dot( ai-v , ai-bi ) )/( Cross( ai-v , ai-bi )) );  double belta00 = Dot( ai-v , ai-v);  double belta01 = Dot( ai-v , v-bi);  double belta11 = Dot( v-bi , v-bi);  double delta00,delta01,delta11;  if ( fabs(deltaI)<1e-8 ) {  // when deltaI ==0  if ( (ai.\_x<=v.\_x && v.\_x<=bi.\_x) || (bi.\_x<=v.\_x && v.\_x<=ai.\_x) ) {  // when v is on the line segment defined by ai and bi  isBreak = true;  \_precalculatedArray.push\_back(ai.\_x);  \_precalculatedArray.push\_back(ai.\_y);  \_precalculatedArray.push\_back(bi.\_x);  \_precalculatedArray.push\_back(bi.\_y);  }  else {  // v is on the extension of one of these line segments  delta00 = pow( (ai-bi).Magnitude(), 5.0 ) /  ( 3.0 \* pow( Dot( v-bi , bi-ai ), 1.0 ) \* pow( Dot( ai-v , bi-ai ), 3.0 ) );  delta01 = -pow( (ai-bi).Magnitude(), 5.0 ) /  ( 6.0 \* pow( Dot( v-bi , bi-ai ), 2.0 ) \* pow( Dot( ai-v , bi-ai ), 2.0 ) );  delta11 = pow( (ai-bi).Magnitude(), 5.0 ) /  ( 3.0 \* pow( Dot( v-bi , bi-ai ), 3.0 ) \* pow( Dot( ai-v , bi-ai ), 1.0 ) );  }  }  else {  delta00 = (ai-bi).Magnitude() / (2.0 \* deltaI \* deltaI) \*  (belta01/belta00 - belta11\*thltaI/deltaI );  delta01 = (ai-bi).Magnitude() / (2.0 \* deltaI \* deltaI) \*  (1.0 - belta01\*thltaI/deltaI);  delta11 = (ai-bi).Magnitude() / (2.0 \* deltaI \* deltaI) \*  (belta01/belta11 - belta00\*thltaI/deltaI );  }  weightP += ai\*( delta00 + delta01 ) + bi\*( delta01 + delta11 );  sumP += delta00 + delta01 + delta01 + delta11;  deltaArray.push\_back(isBreak==true?1.0:0.0);  deltaArray.push\_back(delta00);  deltaArray.push\_back(delta01);  deltaArray.push\_back(delta11);  }  if (isBreak) {  break;  }  }  \_sumDeltaArray.push\_back(deltaArray);  if (isBreak) {  continue;  }  int deltaPos = 0;  weightP /= sumP;  \_precalculatedArray.push\_back(weightP.\_x);  \_precalculatedArray.push\_back(weightP.\_y);  // isBreak is false  for (int k=0; k<srcFeatureLine.size(); k++) {  CPolygon\* pPolygon = srcFeatureLine[k];  int iVNumber = pPolygon->GetVNumber();  bool isClosed = pPolygon->GetClosed();  varray<Vec2>& arrayVertices = pPolygon->GetPolygon();  int nEdge = isClosed ? iVNumber : iVNumber-1;  int iEdge = 0;  for(; iEdge < nEdge; iEdge ++) {  int iV1 = iEdge, iV2 = (iV1+1)%iVNumber;  // 得到line segments的两个端点Ai Bi  Vec2 ai = arrayVertices[iV1] - weightP,  bi = arrayVertices[iV2] - weightP;  deltaPos++;  double delta00 = deltaArray[deltaPos++];  double delta01 = deltaArray[deltaPos++];  double delta11 = deltaArray[deltaPos++];  double a00 = delta00 \* ai.\_x + delta01 \* bi.\_x;  double a01 = delta00 \* ai.\_y + delta01 \* bi.\_y;  double a10 = delta00 \* ai.\_y + delta01 \* bi.\_y;  double a11 = delta00 \* -ai.\_x + delta01 \* -bi.\_x;  double a20 = delta01 \* ai.\_x + delta11 \* bi.\_x;  double a21 = delta01 \* ai.\_y + delta11 \* bi.\_y;  double a30 = delta01 \* ai.\_y + delta11 \* bi.\_y;  double a31 = delta01 \* -ai.\_x + delta11 \* -bi.\_x;  Vec2 vp = v - weightP;    \_precalculatedArray.push\_back( a00\*vp.\_x + a01\*vp.\_y );  \_precalculatedArray.push\_back( a00\*vp.\_y + a01\*-vp.\_x );  \_precalculatedArray.push\_back( a10\*vp.\_x + a11\*vp.\_y );  \_precalculatedArray.push\_back( a10\*vp.\_y + a11\*-vp.\_x );  \_precalculatedArray.push\_back( a20\*vp.\_x + a21\*vp.\_y );  \_precalculatedArray.push\_back( a20\*vp.\_y + a21\*-vp.\_x);  \_precalculatedArray.push\_back( a30\*vp.\_x + a31\*vp.\_y);  \_precalculatedArray.push\_back( a30\*vp.\_y + a31\*-vp.\_x);  }  }  }  return ;  } |

### 输入变形后的特征线，进行变形

在初始化输入图像时，我们已经预处理和预计算了。在程序输入变形后的特征线信息之后，我们就可以通过预计算得到的结果直接进行变形函数的运算，并将运算后的结果保存在\_targetGrid中。

|  |
| --- |
| void CMLS::DeformMesh(const varray<CPolygon \*>& targetFeatureLine) {  // 循环原图所有网格点  const varray<double>& arrayVertexPos = \_srcGrid->GetVertexPos();    if(\_targetGrid.size() != arrayVertexPos.size()){  \_targetGrid.clear();  \_targetGrid.resize(arrayVertexPos.size());  }  int precalucatedPos = 0;  for (int i=0; i<arrayVertexPos.size()/2; i++) {  Vec2 v(arrayVertexPos[ i+i ],arrayVertexPos[ i+i+1 ]);  varray<double>& deltaArray = \_sumDeltaArray[i];  int deltaPos = 0;  bool isBreak = false;  Vec2 weightQ;  double sumQ = 0.0;  for (int k=0; k<targetFeatureLine.size(); k++) {  CPolygon\* pPolygon = targetFeatureLine[k];  int iVNumber = pPolygon->GetVNumber();  bool isClosed = pPolygon->GetClosed();  varray<Vec2>& arrayVertices = pPolygon->GetPolygon();  int nEdge = isClosed ? iVNumber : iVNumber-1;  int iEdge = 0;  for(; iEdge < nEdge; iEdge ++) {  int iV1 = iEdge, iV2 = (iV1+1)%iVNumber;  // 得到line segments的两个端点Ci Di  Vec2 ci = arrayVertices[iV1],  di = arrayVertices[iV2];  if ( fabs(deltaArray[deltaPos++])>1e-8 ) {  // v is on the line segments defined by ai and bi  double ax = \_precalculatedArray[precalucatedPos++];  double ay = \_precalculatedArray[precalucatedPos++];  double bx = \_precalculatedArray[precalucatedPos++];  double by = \_precalculatedArray[precalucatedPos++];  Vec2 ai(ax,ay);  Vec2 bi(bx,by);  double fvx = (di.\_x-ci.\_x)\*(v.\_x-ai.\_x)/(bi.\_x-ai.\_x)+ci.\_x;  double fvy = (di.\_y-ci.\_y)\*(v.\_y-ai.\_y)/(bi.\_y-ai.\_y)+ci.\_y;  \_targetGrid[i+i] = fvx;  \_targetGrid[i+i+1] = fvy;  break;  }  double delta00 = deltaArray[deltaPos++];  double delta01 = deltaArray[deltaPos++];  double delta11 = deltaArray[deltaPos++];  weightQ += ci\*(delta00+delta01)+di\*(delta01+delta11);  sumQ += delta00+delta01+delta01+delta11;  }  if (isBreak) {  break;  }  }  if (isBreak) {  continue;  }  weightQ /= sumQ;  double px = \_precalculatedArray[precalucatedPos++];  double py = \_precalculatedArray[precalucatedPos++];  Vec2 weightP(px,py);  Vec2 frv;    for (int k=0; k<targetFeatureLine.size(); k++) {  CPolygon\* pPolygon = targetFeatureLine[k];  int iVNumber = pPolygon->GetVNumber();  bool isClosed = pPolygon->GetClosed();  varray<Vec2>& arrayVertices = pPolygon->GetPolygon();  int nEdge = isClosed ? iVNumber : iVNumber-1;  int iEdge = 0;  for(; iEdge < nEdge; iEdge ++) {  int iV1 = iEdge, iV2 = (iV1+1)%iVNumber;  // 得到line segments的两个端点Ci Di  Vec2 ci = arrayVertices[iV1]-weightQ,  di = arrayVertices[iV2]-weightQ;  double a00 = \_precalculatedArray[precalucatedPos++];  double a01 = \_precalculatedArray[precalucatedPos++];  double a10 = \_precalculatedArray[precalucatedPos++];  double a11 = \_precalculatedArray[precalucatedPos++];  double a20 = \_precalculatedArray[precalucatedPos++];  double a21 = \_precalculatedArray[precalucatedPos++];  double a30 = \_precalculatedArray[precalucatedPos++];  double a31 = \_precalculatedArray[precalucatedPos++];  frv.\_x += ci.\_x\*a00+ci.\_y\*a10+di.\_x\*a20+di.\_y\*a30;  frv.\_y += ci.\_x\*a01+ci.\_y\*a11+di.\_x\*a21+di.\_y\*a31;  }  }  Vec2 ans = frv/frv.Magnitude()\*(v-weightP).Magnitude()+weightQ;  \_targetGrid[i+i] = ans.\_x;  \_targetGrid[i+i+1] = ans.\_y;  }  return ;  } |

### 获得变形后图像

|  |
| --- |
| const varray<double>& CMLS::GetDeformLatticeVPos() const {  return \_targetGrid;  } |

## 本章总结

在这一章中，我们主要介绍图像变形系统所使用的IDE，开发框架，语言及其它们所具有的特性。使用了Visual Studio 2008，QT以及C++，实现了轻量级开发的优势。接下来，我们梳理了核心代码的核心函数，并且介绍了它们的设计思路和实现。

# 总结

近年来，图像变形是计算机图形学和数字图像处理领域的热点问题，它被广泛应用于计算机动画、医学影像处理、虚拟现实等各个领域。在本文中，我们对于基于点和基于特征线段的移动最小二乘图像变形方法进行了研究。

首先，对于基于点的移动最小二乘图像变形方法的研究中，我们给出了基于点的移动最小二乘的模型。然后推导了在仿射变换、相似变换、刚性变换三种情况下的变形函数。尽管刚性变换中存在非线性的计算，我们展示了如何通过相似变换和刚性变换的关系从而绕过非线性最小化的计算。

基于点的方法在描述轮廓和复杂形状上存在不足，所以我们研究了基于特征线段的移动最小二乘方法。相似的，我们先建立一般化的移动最小二乘模型，然后针对仿射、相似、刚性变换三种情况分别展开推导，并分别给出了它们封闭形式的解。

介绍了一个图像变形系统。介绍了该系统的开发环境和核心功能。应用该系统获得的图像变形效果令人满意。

本文对图像变形的一部分关键技术进行了研究，但还有很多方面不够完善，需要进一步的提高，仍有大量的工作需要完成。例如在变形方面，对于图像变形空间重叠的问题，本文中的方法具有局限性，即在部分情况下变形结果和真实感相距甚远。如何消除空间重叠产生的非真实感是一个问题。

# 参考文献

1. 王城峰，陈建平.基于特征点的图像变形技术及应用。广西师范大学学报，2003，21(1):450-454
2. Smythe D B. A two-pass mesh warping algorithm for object transformation and image interpolation. ILM Computer Graphics Department. San Rafael,Calif,1990,Technical Report 1030.
3. Hui Fang, Hart J.C. Detail Preserving Shape Deformation in Image Editing. ACM Trans Graph, 2007, 26(3): 12-16.
4. Summer R W, Schmid J, Pauly M, Embedded Deformation for Shape Manipulation. ACM Trans, Graph, 2007, 26(3): 80-86.
5. MacCracken R, Joy K I. Free-form deformations with lattices of arbitrary topology. In proceedings of ACM SIGGRAPH 1996, ACM Press, 1996:181-188.
6. Sederberg T W, Parry S R. Free-form deformation of solid geometric models. In Proceedings of ACM SIGGRAPH 1986. ACM Press, 1986:151-160.
7. Milliron T, Jensen R, Barzel R et al. A Framework for Geometric Warps and Deformations. ACM Trans, Graphics, 2002, 21(1):20-51.
8. Kho Y, Garland M. Sketching Mesh Deformations. International Conference on Computer Graphics and Interactive Techniques. Los Angeles, California. 2005:934-941.
9. Beier T, Neely S. Feature-based image metamorphosis. SIGGRAPH’ 92 Conference Proceedings, Chicago, Illinois, U.S. 1992,26:34-42.
10. Igarashi T, Moscovich T, Hughes J F. As-rigid-as-possible shape manipulation. ACM Trans, Graph, 2005, 24(3):1134-1141.
11. Han L, Amicis R D, Conti G. Interactive Spline-Driven Deformation for Free-Form Surface Styling. Proceedings of the 2006 ACM symposium on Solid and physical modeling.
12. 李旭东，张振跃，非对称径向基函数与稳定边界图像变形算法. 计算机辅助设计与图形学学报, 2004, 16(6):747-752.
13. Lee S, Chwa K Y, Hahn J et al. Image morphing using deformation techniques. The Journal of Visualization and Computer Animation,1996, 7(1):3-23.
14. Terzopoulos D, Platt J, Barr A et al. Elastically deformable models. 1997,21(4):205-214.
15. Levin D. The approximation power of moving least squares. Mathematics of Computation, 1998, 67(224):1517-1531.
16. Weng Y L, Xu W W, Wu Y C et al. 2D Shape Deformation Using Nonlinear Least Squares Optimization. The Visual Computer, 2006, 22(9):653-660.
17. Schaefer S, McPhail T, Warren J. Image Defromation Using Moving Least Squares. ACM SIGGRAPH, 2006:533-540.

# 致 谢

经过几个月的论文研究方向选定、研究对象的选择、算法选择、系统框架搭建，毕业论文设计已经接近尾声。作为一个首次接触论文写作的我来说，确实是不小的挑战，刚一开始接触课题的时候，我几乎没有任何想法，也不知道论文该如何下手，但是经过和指导老师的邮件和面谈后，渐渐的对课题有了自己的认识，对这篇论文的构架有了一个系统的具有逻辑性的认识。

首先感谢指导老师杨老师，悉心指导我完成这篇论文，他对我论文的帮助超过了我自己的所有努力。在百忙之中，他一直关心着我的毕业设计，并且对于我的问题，他每次都表现的非常耐心，并给我提供了许多具有建设性的意见。其次，我要感谢我的室友，他在对我的系统环境搭建和算法具体实现上起到了关键的作用，在他的帮助下，我的系统进度能够在预期时间内完成。

本论文的顺利完成，倾注了指导老师大量的精力和心血。在此，感谢导师，感谢朋友的指导和帮助；感谢四年大学学习中曾经教育过我的各位老师，感谢我的同学和家人给我的帮助和支持，感谢大学对我四年的培养。