数值分析大作业1 实验报告

2015011506 金帆 自56班

目录

[1 需求分析 2](#_Toc498283809)

[1-1 符号约定 2](#_Toc498283810)

[1-2 概述 2](#_Toc498283811)

[1-3 旋转扭曲变换 2](#_Toc498283812)

[1-4 水波纹扭曲变换 3](#_Toc498283813)

[1-5 B样条变换 4](#_Toc498283814)

[1-5-1 一次B样条变换 4](#_Toc498283815)

[1-5-2 三次B样条变换 5](#_Toc498283816)

[1-6 最近邻插值 5](#_Toc498283817)

[1-7 双线性插值 5](#_Toc498283818)

[1-8 双三次插值 6](#_Toc498283819)

[2 方案设计 7](#_Toc498283820)

[2-1 编程语言及环境 7](#_Toc498283821)

[2-2 系统模块图、类图 8](#_Toc498283822)

[2-3 必做任务的反变换求解（解析法） 9](#_Toc498283823)

[2-4 选做任务的反变换求解（迭代法） 9](#_Toc498283824)

[2-4-1 B样条正变换的性质 9](#_Toc498283825)

[2-4-2 B样条逆变换的存在性 10](#_Toc498283826)

[2-4-3 迭代法求解逆变换 10](#_Toc498283827)

[2-5 插值 11](#_Toc498283828)

[3 误差分析 11](#_Toc498283829)

[3-1 观测误差 11](#_Toc498283830)

[3-2 舍入误差 11](#_Toc498283831)

[3-3 截断误差（迭代法） 11](#_Toc498283832)

[3-4 方法误差（插值） 12](#_Toc498283833)

[4 结果展示与参数讨论 16](#_Toc498283834)

[4-1 旋转扭曲变换 16](#_Toc498283835)

[4-2 水波纹扭曲变换 19](#_Toc498283836)

[4-3 B样条变换 19](#_Toc498283837)

[参考文献 19](#_Toc498283838)

1 需求分析

1-1 符号约定

首先约定，以下对于像素点坐标的描述，其中表示像素点所在的行数，表示像素点所在的列数。假设图像有横行、纵列，则图像左上角、右上角、左下角、右下角的坐标分别为、、、。

1-2 概述

总体目标是，编写图像扭曲变形程序，可以对图像进行扭曲变形。

图像扭曲的本质是一个坐标到坐标的变换，即一个的映射。给定原图，在求取扭曲后的图像时，我们的目标是，对于扭曲后图像的每一个像素点，寻找原图中对应点的坐标，使得，即求解映射的逆映射。

一般地，通过解析法或者迭代法求得的不是整数点，而原图是离散的，只有整数点处才有值，因而必须对原始图像进行插值，用插值结果作为原图处的像素值，也就是扭曲后图像处的值。本次作业，我们尝试使用三种插值方式：最近邻、双线性、双三次。

1-3 旋转扭曲变换

对于行数与列数分别为与的图像，其图像中心点的坐标是，旋转半径。点相对于图像中心的极坐标表示记为，则

假设变换后的极坐标表示为，则变换方程为：

其中是示性函数，是参数，控制旋转的方向和角度。

其反变换为

设变换后的点的直角坐标是，则有

这样给定，先计算，再计算，得到变换前的直角坐标。这样得到的不是整数点，因此需要使用插值方法，将插值结果作为变换后图像的处的值。

这一过程对RGB三个通道遍历，并对新图像的每一个像素点遍历。

1-4 水波纹扭曲变换

直角坐标和极坐标转换的部分与1-3节的旋转扭曲相同。只是，我们将正向变换公式变形为

其中、、是参数，控制水波纹的幅度，控制波长，控制相位。

反向变换公式相应变为

其余步骤与1-3节旋转扭曲相同，此处不再赘述。需要注意的是，由于此处不再有的示性函数，因此变换后的可能超出了图像范围。这时我们的处理是，令，。

1-5 B样条变换

在原图等距选取一些控制点，组成阵列。在方向相邻控制点间隔像素，在方向相邻控制点间隔像素。我们假设图像边长是上述间隔的整数倍加1，这样保证图像的四个角上的像素一定是控制点。

每个控制点的位置都可以在图像范围内拖动。记表示方向第个、方向第个控制点在方向的位移，表示方向第个、方向第个控制点在方向的位移。该控制点在原图中的原始坐标应为，拖动后的坐标变为

1-5-1 一次B样条变换

对于原图坐标为的像素点，一次B样条变换后，其位移如下：

其中，基函数定义由作业要求给出，

，，，

可以看到，一次B样条使用了该点“周围”最近邻的4个控制点的信息，是这4个控制点的位移的线性组合。这里的“周围”是在控制点位置没有移动时定义的。

1-5-2 三次B样条变换

对于原图坐标为的像素点，三次B样条变换后，其位移如下：

其中，基函数定义由作业要求给出，

，，，

可以看到，三次B样条使用了该点“周围”最近邻的16个控制点的信息，是这16个控制点的位移的线性组合。

由于使用了取整运算，B样条变换的逆变换没有解析解。之后我们采用迭代法，以求取其逆变换的数值解。

1-6 最近邻插值

对于非整数坐标，使用与它距离最近的整点处的值作为插值结果。

1-7 双线性插值

对于非整数坐标，定义

，，，

则插值结果

直观来说，就是先在方向做两次线性插值，再对所得结果在方向做线性插值。

1-8 双三次插值

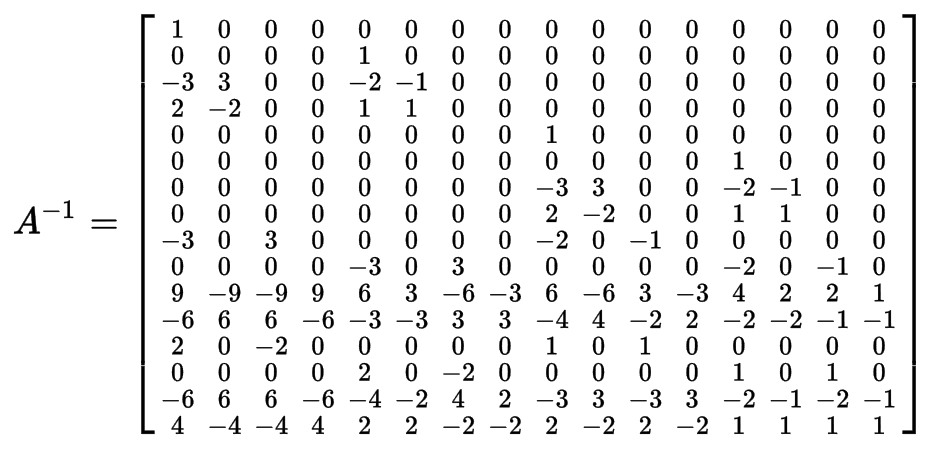
对于非整数坐标，定义

，，，

则插值结果

其中系数通过以下矩阵运算得到：

其中系数矩阵为（图片来自 <https://en.wikipedia.org/wiki/Bicubic_interpolation> ）



微分算子的定义为：（误差分析参见3-4-3节）

2 方案设计

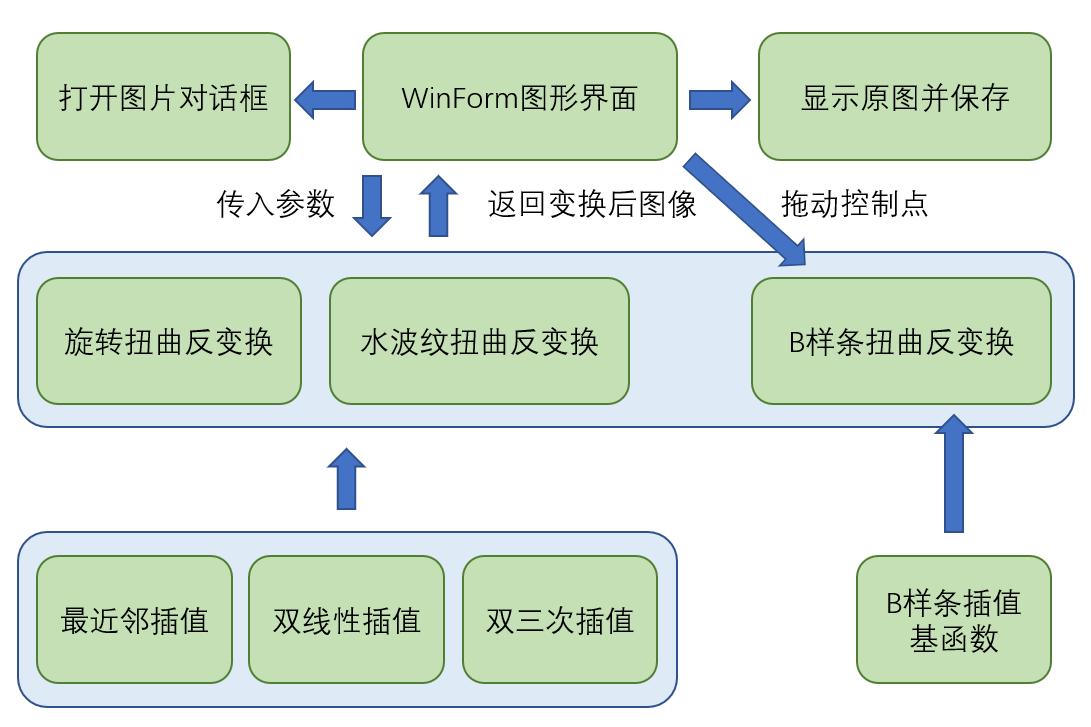
2-1 编程语言及环境

采用C#（.NET Framework 4.5.2），在Visual Studio 2015下开发，可执行文件可在Windows 10环境下直接运行，无需安装第三方库。

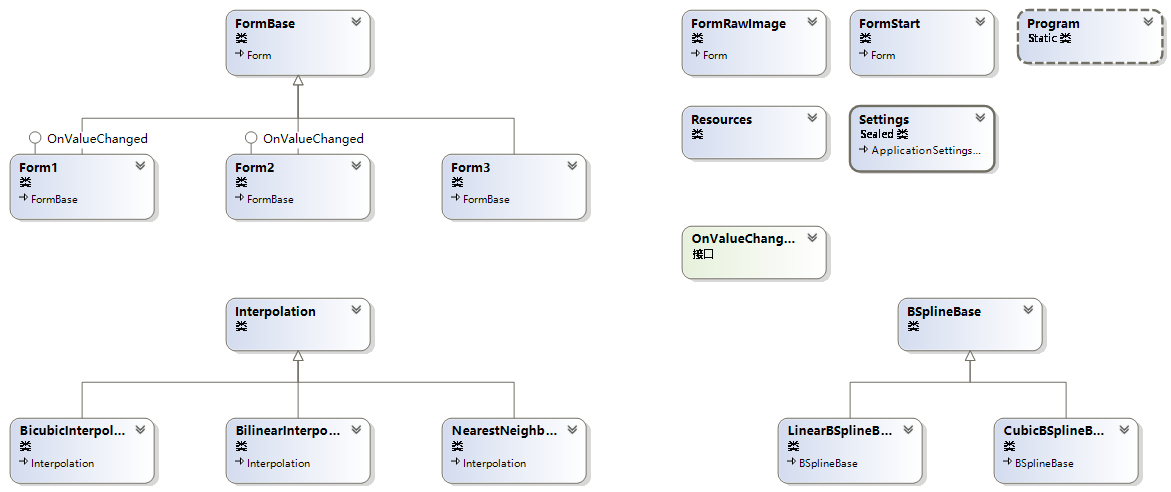
图像读写使用了.NET框架内置的System.Drawing.Bitmap类，为提高性能，绕过了Bitmap类提供的SetPixel和GetPixel方法，使用不安全的指针直接操作内存。已在编译选项中开启了“允许不安全代码”，以支持unsafe语句。

程序自带一个WinForm界面，不提供命令行接口。在Visual Studio中已对工程的输出做了重定向，生成的exe文件就在根目录下的exe文件夹内。

2-2 系统模块图、类图



我们使用面向对象（OOP）的思想编写程序。3个任务分别对应3个窗体，由于有较多的共有部分，共有部分写为FormBase类中，其被3个任务的窗体分别继承，并加入各自的参数控件，并重写反变换算法。插值被单独做成一个类，三种插值分别是其一个子类，接口统一，这样上层就无需关心具体是哪种插值的实现。同理，一阶和三阶的B样条变换同样也采用父类-子类的继承关系。



2-3 必做任务的反变换求解（解析法）

在1-3节和1-4节，我们已经给出了必做任务的反变换的解析公式。程序从参数控件上获取到当前的参数，代入解析公式，对每个像素点，反变换得出对应的坐标。由于这里的坐标不是整数点，需要调用插值模块，将插值结果作为变换后图像的位置的值。对于RGB图像，每个通道都如此操作。

2-4 选做任务的反变换求解（迭代法）

B样条正变换的公式（参见1-5节）比较复杂，由于取整运算，其逆变换的解析公式不易求得。

2-4-1 B样条正变换的性质

显然，B样条正变换公式在和均是非整数时是连续的。

注意到一阶B样条基函数具有以下性质：

因此，展开和式后，发现在当或跨越整数时，仍然连续的。

综上，一阶B样条正变换是一个的连续映射。对于三阶B样条变换，基函数的一阶、二阶导数也有类似性质，使得三阶B样条变换是二次连续可微的。

又由于基函数是有界的，假设其绝对值的界是，则对1-5节的公式进行放缩：

也是有界的，因此B样条正变换满足有界。

2-4-2 B样条逆变换的存在性

此部分不做要求，此处略去，以下假定逆变换存在且唯一。

2-4-3 迭代法求解逆变换

在本小节中，以符号表示一个点的坐标的二元数组，经过B样条正变换后坐标变为。逆变换的本质是，给定，求取使得。算法如下：

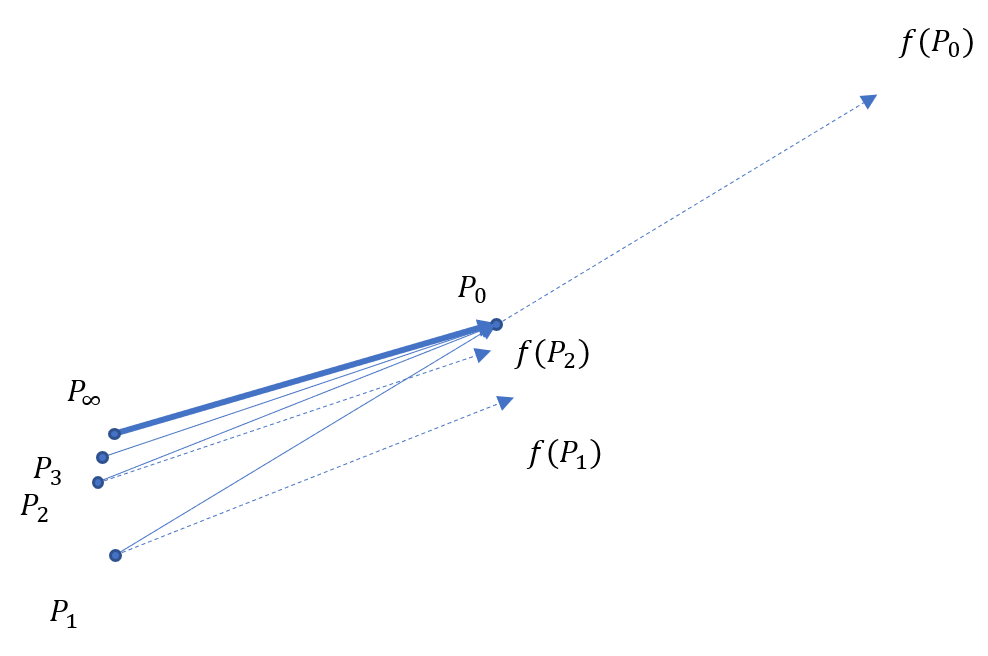
输入：（二元数组），映射

输出：使得

步骤：

for i = 1, 2, …, MaxIteration

求解满足如下条件的：向量



假设得到的一系列点收敛至。注意到

因此“二维点列收敛到”与“二维点列收敛到”等价。

2-5 插值

插值算法见1-6、1-7、1-8节。

在插值时，图像的边缘处理比较麻烦。我们采取的办法是，累加的时候，对于下标超出了原图范围的项一律置零，相当于补上0。

为了加快双三次插值的速度，在代码中部分运算被展开，使其差分的意义不容易看出，这是以牺牲可读性为代价的。

3 误差分析

3-1 观测误差

不属于本文讨论范围。

3-2 舍入误差

C#中的double类型，精度有15-16位有效数字，因此舍入误差可以忽略。在计算的最后一步，将double类型的数转成0~255范围的uint8，舍入误差上界为0.5。

3-3 截断误差（迭代法）

在2-4-3节中，迭代算法在给定的最大迭代次数后停止。为了便于控制精度，同时也为加快运算速度，我们在每次迭代后，比较当前的与目标点的距离（以无穷范数度量，即）；如果距离小于给定的误差界，则提前停止迭代。

经过试验，在每一个像素点上，我们都提前停止了迭代。因此，我们下式满足：

其中是我们算法的输出结果，进而，其与的误差满足：

其中，在的某个邻域内。

这样在无穷范数意义下，截断误差的上界是。

3-4 方法误差（插值）

在本节中，约定符号表示一个的映射。当与是整数时，表示原图在该位置的某个通道的取值（0~255）。就是未知的被插值函数。

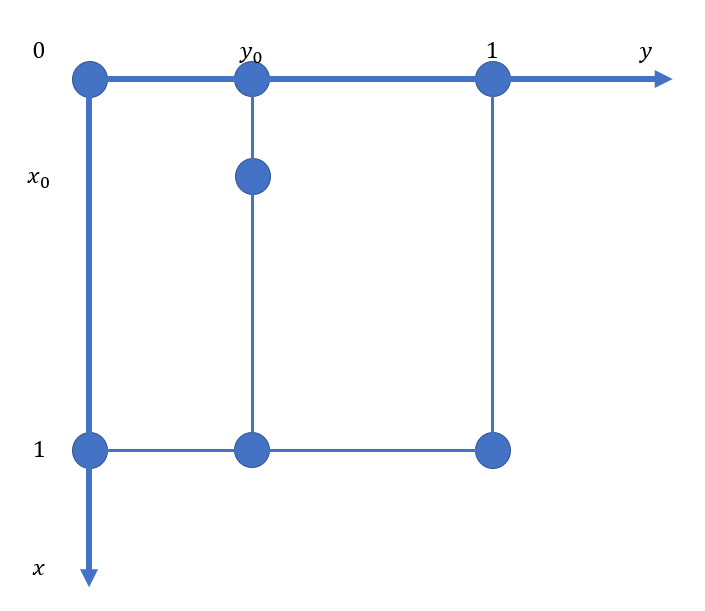
3-4-1 最近邻插值

最近邻插值的方向与方向是解耦的。我们假定，在方向和方向上的一阶导数存在且有界。除去物体边缘等梯度大的区域外，都很小；在图像中一些梯度大的地方（例如物体轮廓线），较大。

方向与方向的误差上界各为，总的方法误差上界为。

3-4-2 双线性插值

不失一般性，假设求值点位于单位正方形内，4个顶点是插值点。



假设在方向和方向的二阶导数存在且有界。

首先考察与处的插值结果的误差界。这相当于在直线和上做两次方向的线性插值，因此误差界是

然后给定与，考察处的插值结果的误差界。这相当于在直线做一次方向的线性插值，因此相对于给定的与，误差界是

再加上与的误差界，最坏情形下两种误差叠加，总误差为

3-4-3 双三次插值

在使用双三次插值时，误差来自两方面：一是使用差分近似代替导数的误差，二是双三次插值本身的误差。在最坏情形下，两者叠加，

首先，假设插值函数的二阶导数存在且有界，即

同理假设的三、四阶导数存在，且有界、。

首先分析差分代替导数的误差界。对于导数的估计，公式是（参加1-8节）：

构造辅助函数。

由微分中值定理，存在使得

故有

将这个结论应用到1-8节的3个公式，得到误差界

同理，根据辅助函数

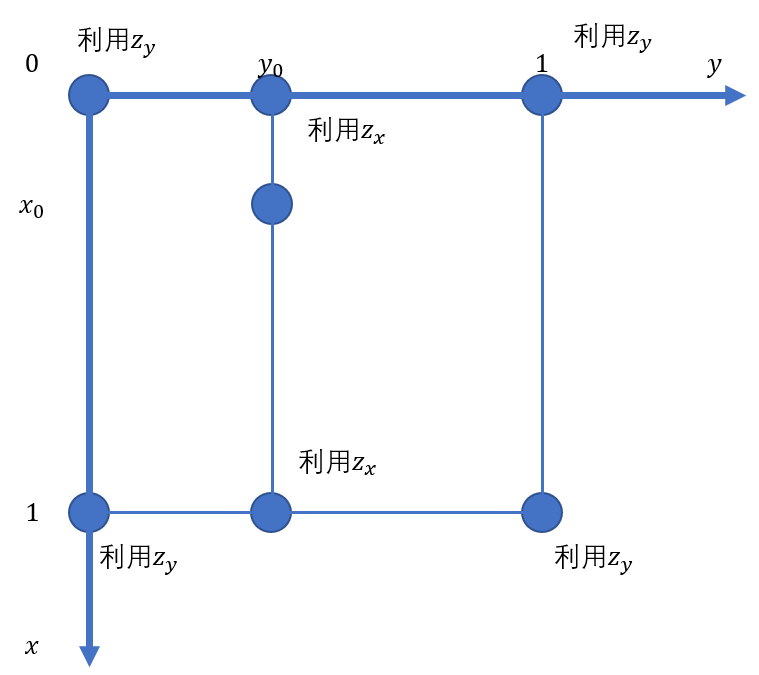
可以得到误差界

第二部分的误差来自双三次插值本身。根据Wikipedia提供的公式，双三次插值的结果可以使用以下的二次型表示：

其中矩阵

如果记，，则。

由于矩阵乘法的结合律，可以将双三次插值视为先在方向利用和信息进行2次三次Hermite插值，然后在方向利用和对结果做1次三次Hermite插值。

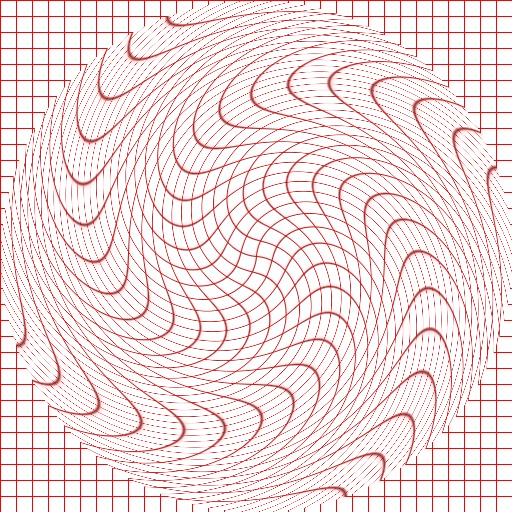


与3-4-2节同理，最坏情形下，插值的误差界是两方向的误差界之和，即

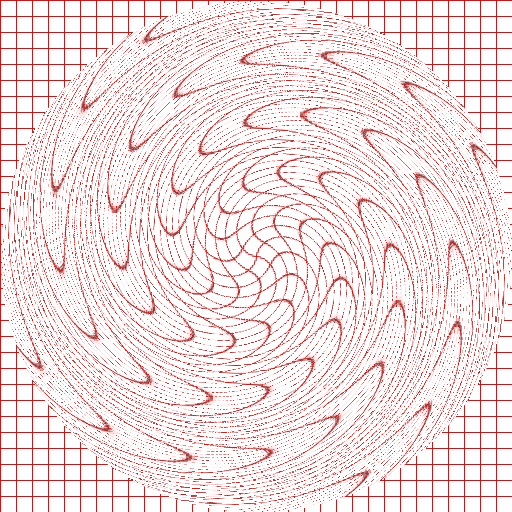
综上，最坏情形下，双三次插值方法（包含导数的差分估计）误差界是两方面误差之和，即

4 结果展示与参数讨论

4-1 旋转扭曲变换



（双线性）

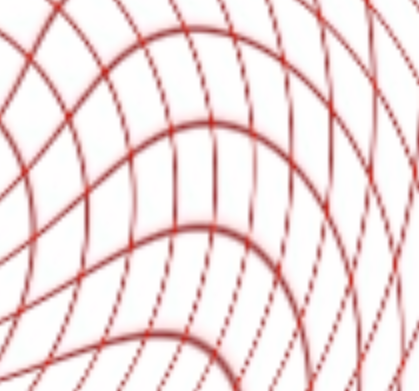


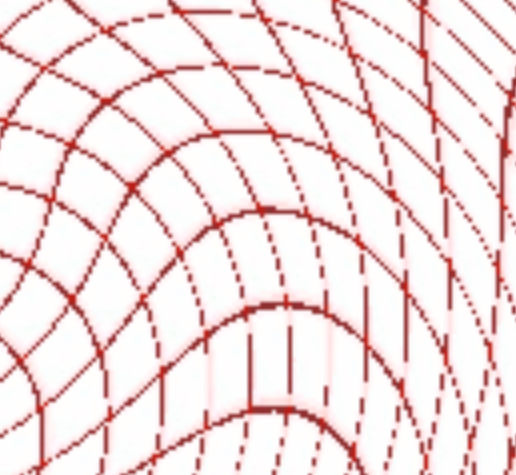
（最近邻）



（双三次）

考察一些扭曲较大的局部，观察双线性插值相比最近邻插值的改进：

（双线性）

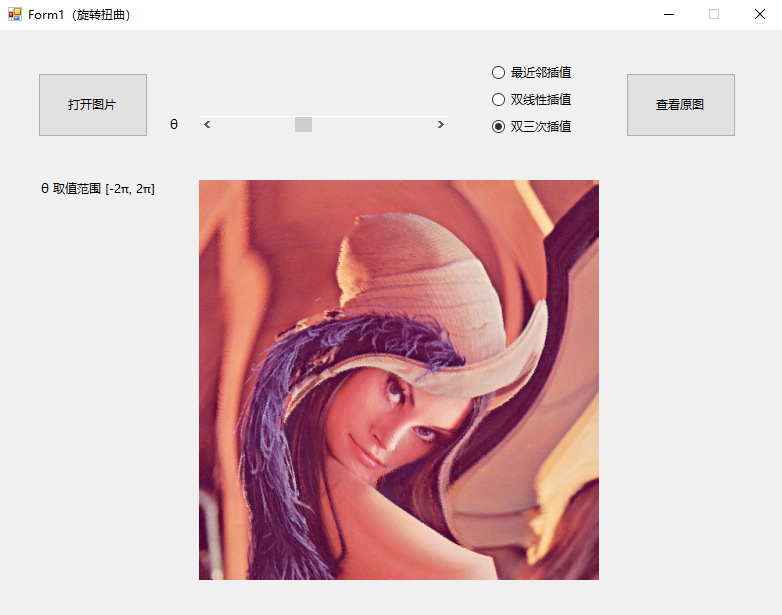
（最近邻）

最近邻插值的结果存在断续，而双线性的插值结果更平滑，在红色的线条和空白之间存在一些浅红色的渐变。这符合预期。

双三次的结果则更加平滑：



利用图形界面，可以打开更多的图片，并尝试改变旋转角度：



点击“查看原图”可以保存图片，并避免操作系统自带的DPI缩放机制。

4-2 水波纹扭曲变换

4-3 B样条变换

参考文献

李庆阳、王能超、易大义：《数值分析》第5版

<https://en.wikipedia.org/wiki/Bicubic_interpolation>

<https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/double>

<https://msdn.microsoft.com/en-us/library/ms165366.aspx>

<https://msdn.microsoft.com/en-us/library/system.drawing.bitmap(v=vs.110).aspx>

<https://stackoverflow.com/questions/13511661/create-bitmap-from-double-two-dimentional-array>