

# **数值分析第一次大作业**

## **图像扭曲变形**

**张蔚桐 2015011493 自 55**

### **目录**

1	方案设计与程序框图 . . . . .	2
2	变形函数的选取与设计 . . . . .	2
2.1	图像旋转变换 . . . . .	3
2.2	水波纹变换 . . . . .	3
2.2.1	B 样条变换 . . . . .	5
3	方案原理及误差分析 . . . . .	7
3.1	选取点, 半径时的误差 . . . . .	7
3.2	逆变换误差 . . . . .	7
3.3	插值过程 . . . . .	7
3.3.1	最近邻插值 . . . . .	8
3.3.2	双线性插值 . . . . .	8
3.3.3	双三次插值 . . . . .	9
3.3.4	三种插值方法的比较 . . . . .	9
3.4	图像输出的误差 . . . . .	10
4	实验结果分析 . . . . .	10

## 1 方案设计与程序框图

整体的设计思路是，对于新图像上的每一个点，通过所选取变换的逆变换寻求原图像上的点。通过原图像的点进行等间隔插值，插值出要求点的颜色，最终返回渲染新图像。为了方便用户进行操作，相关变换，插值方式的选择和相关参数的选定应当在用户界面上友好的展示，整个程序的设计流程图如图 1 所示，程序的开始截图如图 2 所示。

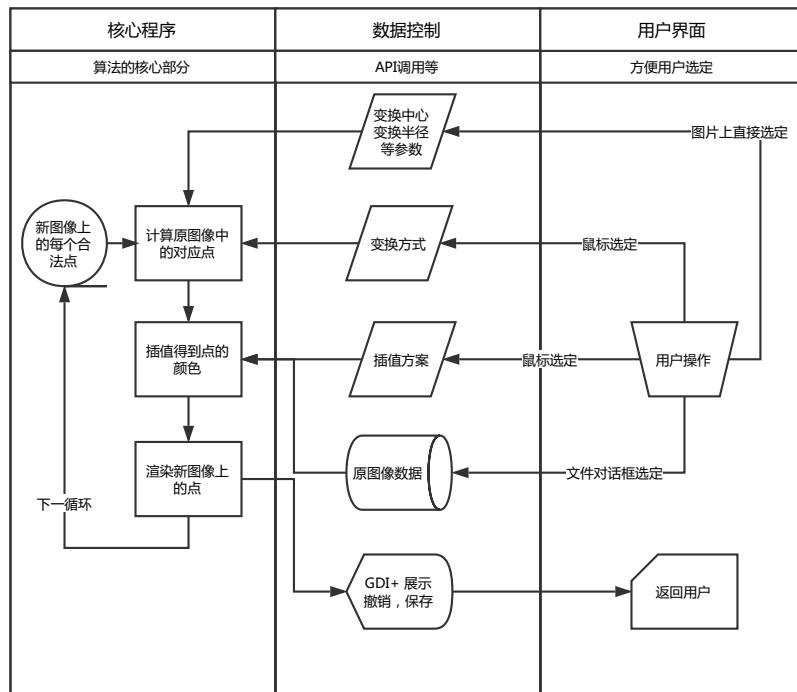


Fig. 1: 程序设计图

## 2 变形函数的选取与设计

项目的需求是实现一个图像旋转，水波纹变化，以及通过 B 样条插值得到的图像修改的功能，因此要设计适当的变形函数来将图片进行变形。即通过某变换  $f$  使得原图像  $I$  被映射为新图像  $I'$ 。通过后文的分析我们可以看出，具体实现的整体思路是将新图像中的每一个位置  $\vec{p}'$  通过寻找  $f$  的逆变换  $f'$  找到其在原图像中的位置  $\vec{p}$ ，并通过插值确定颜色。因此工作的重心实际上

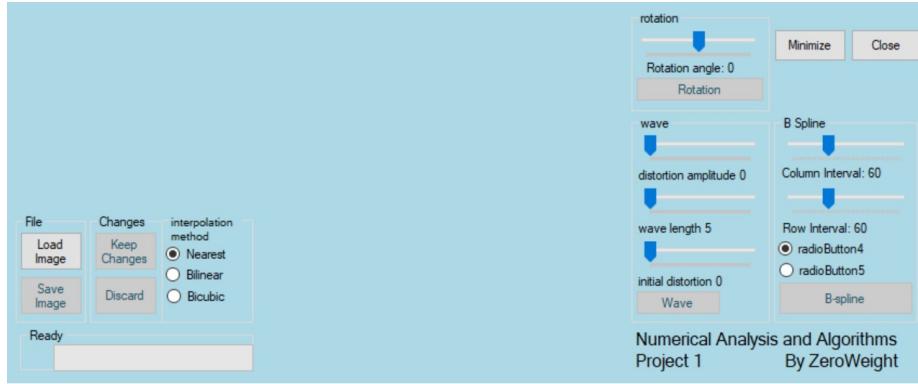


Fig. 2: 程序运行图

在于对于逆变换  $f'$  的求取。下面将针对三种变形进行讨论。

## 2.1 图像旋转变换

作业要求中已经给了我们这个变换的正变换公式<sup>1</sup>为

$$\begin{cases} x = r \cos(\alpha + \theta \frac{row - r}{row}) \\ y = r \sin(\alpha + \theta \frac{row - r}{row}) \end{cases} \quad (1)$$

因此我们的任务是寻找其逆变换，式 1 的各个点坐标均是在以图片为中心的极坐标下取得的，因此我们可以看出这是一个保幅变换。因此我们可以迅速得到方程 2

$$\tan(\alpha + \theta \frac{row - r}{row}) = \frac{x}{y} \quad (2)$$

通过  $x, y$  的具体符号，我们可以很容易的得到  $(\alpha + \theta \frac{row - r}{row})$  的表达式，并进一步得到  $\alpha$  的值。具体的算法可以由算法 1 表示

## 2.2 水波纹变换

经过仔细的考察，发现作业中给出的公式 3 存在着一定的问题，这里简述如下

$$\begin{cases} x = r \sin(\alpha + \sin(\frac{r}{R}\rho + \phi) + R \\ y = r \cos(\alpha + \sin(\frac{r}{R}\rho + \phi) + R \end{cases} \quad (3)$$

不论对公式进行如何修正<sup>2</sup>，当变换点取在“最大变化半径”  $R$  之外时，变换后点位置不变，暂且记为极坐标形式  $r, \alpha$ ，当变换点在内式，变换后的

<sup>1</sup> 这里修正了 PPT 中除 2 的错误

<sup>2</sup> 包括将  $\sin, \cos$  互换位置，将后面的  $+R$  项忽略等

---

**Algorithm 1** 求新图位置  $\vec{p}' = \{x', y'\}$  在原图像中的位置

---

**Require:**新图像中的绝对坐标位置  $x', y'$ 变换原点的位置  $x_c, y_c$ 参数  $row, \theta$ **Ensure:** 原图像中的绝对坐标位置  $x, y$ 

```

1: function 旋转扭曲逆变换  $(x', y', x_c, y_c, row, \theta)$ 
2:   计算新图像中相对  $x_c, y_c$  的相对位置  $x'_r = x' - x_c, y'_r = y' - y_c$ 
3:   计算相对原点的距离  $r = \sqrt{x'^2_r + y'^2_r}$ 
4:   if  $r < 1$  then
5:     //说明点在变换原点附近, 为防止模型病态
6:     return  $x = x', y = y'$ 
7:   end if
8:   计算  $\phi = \arctan \frac{y'_r}{x'_r}$ 
9:   if  $y'_r < 0$  then
10:     $\phi = 2\pi - \phi$ 
11:   end if
12:   计算  $\alpha = \phi - \theta \frac{row - r}{row}$ 
13:   计算  $x = x_c + r \cos \alpha, y = y_c + r \sin \alpha$ 
14:   检查  $x, y$  是否在合法范围内
15:   return  $x, y$ 
16: end function

```

---

点位置发生改变, 主要表现为角度发生了  $\sin(\frac{r}{R}\rho + \phi)$  的变化。因此, 当变化点的幅值从两侧接近  $R$  的时候, 变换后的点的幅值变化可以被式 4 表征

$$\begin{cases} \lim_{r \rightarrow R^-} \Delta \angle \vec{p} = \sin(\rho + \phi) \neq 0 \\ \lim_{r \rightarrow R^+} \Delta \angle \vec{p} = 0 \end{cases} \quad (4)$$

所以这个变换在平面上当  $\sin \rho + \phi \neq 0$  的情况下是不连续的, 即不论采用怎样的变换方式, 势必在  $r = R$  的圆上产生割痕。并且经过实验, 取  $\sin \rho + \phi = 0$  加强条件也很难有较好的效果。因此放弃了采用推荐形式的方式。

我们采用了开源图像处理库 JHLabs<sup>3</sup>中的处理方式并加以改进。原算法的处理方式为, 对于每一个在变换区域内的点, 在他的幅(距离原点的长度)上加一个正弦项的扰动, 即若原位置距离变换中心为  $r$ , 新位置距离

<sup>3</sup> <http://www.jhlabs.com/ip/filters/index.html>

变换中心的距离为  $(1 + \omega)r$ , 其中  $w$  是一个随着  $r$  变换的幅值很小的扰动项, 可以认为  $\|w\| < 1$  因此我们可以将变换  $r' = (1 + \omega)r, r = \frac{r'}{1+\omega}$  近似为  $r = (1 + \omega')r'$ , 其中  $\omega'$  是另一个波动项, 在一阶近似的情况下,  $\omega' = -\omega$ 。不妨取  $\omega' = A \sin(2\pi \frac{r}{\lambda} + \phi)$ , 其中  $A$  可以表征水波的振幅,  $\lambda$  可以表征水波的波长,  $\phi$  可以表征水波的初相位。采用这种方式, 不仅可以克服在  $r = R$  圆上角度不连续的问题, 而且引入的参数也更有物理意义。

在 JHLabs 提供的实现算法思路的基础上, 我们在每一个点的波动基础上加入衰减项, 表现出波动振幅的衰减特性, 具体表述将上式中的  $r = (1 + \omega')r'$  修正成为  $r = (1 + \exp(-\frac{r}{\Lambda})A \sin(2\pi \frac{r}{\lambda} + \phi))r'$ , 式中  $\Lambda \equiv 0.2R$  使得波动项在边界  $R = r$  处基本达到振幅为 0, 使得图像更平滑。

和算法 1 类似的, 我们也可以得到逆向计算波纹的算法如算法 2 所示。

---

**Algorithm 2** 求新图位置  $\vec{p}' = \{x', y'\}$  在原图像中的位置

---

**Require:**

新图像中的绝对坐标位置  $x', y'$

变换原点的位置  $x_c, y_c$

参数  $\lambda, \phi, R, A$

**Ensure:** 原图像中的绝对坐标位置  $x, y$

- 1: **function** 图像水波纹变换  $(x', y', x_c, y_c, \lambda, \phi, R, A)$
  - 2:     计算新图像中相对  $x_c, y_c$  的相对位置  $x'_r = x' - x_c, y'_r = y' - y_c$
  - 3:     计算相对原点的距离  $r' = \sqrt{x'^2_r + y'^2_r}$
  - 4:     计算衰减波动因子  $\Omega = A \exp(-\frac{5r'}{R}) \sin(2\pi \frac{r'}{\lambda} + \phi)$
  - 5:     计算  $x = x_r + x_c = x'_r(1 + \Omega) + x_c, y = y_r + y_c = y'_r(1 + \Omega) + y_c$
  - 6:     检查  $x, y$  是否在合法范围内
  - 7:     **return**  $x, y$
  - 8: **end function**
- 

### 2.2.1 B 样条变换

从作业中给出的定义中我们可以知道, 当控制点为  $\vec{P}$  的时候, 我们可以得到二位图像中的每一个位置的位移如式 5

$$\begin{cases} \Delta_x(x, y) = \sum_{l=0}^n \sum_{m=0}^n G_{l,n}(u) G_{m,n}(v) \Delta \vec{P}_{x(i+l, j+m)} \\ \Delta_y(x, y) = \sum_{l=0}^n \sum_{m=0}^n G_{l,n}(u) G_{m,n}(v) \Delta \vec{P}_{y(i+l, j+m)} \end{cases} \quad (5)$$

其中  $n$  为 B 样条的次数,  $\Delta_x, \Delta_y$  分别是原位置在  $x, y$  点的位移。 $G_{\cdot,\cdot}(\cdot)$  为 B 样条插值基函数。具体的表述形式作业中已经给的很全面了, 这里略去。

式中

$$\begin{cases} u \equiv \frac{x}{N_x} - \left\lfloor \frac{x}{N_x} \right\rfloor & , i \equiv \left\lfloor \frac{x}{N_x} \right\rfloor \\ v \equiv \frac{y}{N_y} - \left\lfloor \frac{y}{N_y} \right\rfloor & , j \equiv \left\lfloor \frac{y}{N_y} \right\rfloor \end{cases} \quad (6)$$

为了方便后文表述，我们记这个正变换为  $F(x, y) \rightarrow (x', y')$

B 样条插值的主要困难是不方便像前两个问题一样顺利的解决变换的逆变换的问题。即很难找到一个  $G(x', y')$  使得  $F(G(x, y)) = x, y$  因此我们猜想利用一些  $F(\cdot, \cdot)$  函数的性质。 $F(\cdot, \cdot)$  的性质主要取决于后面的差分项，定义  $F(x, y) = 1 + H(x, y)$  来使用  $H(\cdot, \cdot)$  函数来表达这个差分项。并试图利用误差反向传播，参考非线性方程的相关解法得到方程的近似解。根据目前所学的知识，很难证明这个求逆在无限时间内必然收敛到  $x, y$ 。但是，由于 B 样条曲线相近点的位移基本相同，因此这个思路是可取的，并且由于 B 样条变形的控制点对于整个图像在表现上的作用本身不是很明显，因此近似的非准确的实现这个过程就可以很好的完成要求。

具体的算法如算法 3 所示，具体实验中  $k = 1, \text{eps} = 2$  就可以很好的完成上述任务，没有出现不合理情况。

---

### Algorithm 3 求 B 样条插值的逆变换的函数

---

**Require:** 新图像中的绝对坐标位置  $x', y'$

**Ensure:** 原图像中的绝对坐标位置  $x, y$

```

1: function B 样条变换逆变换  $(x', y')$ 
2:   定义常量  $k, \text{eps}$ 
3:   记  $x_i = x', y_i = y'$ 
4:   for 进行固定次数循环 do
5:     计算  $\{x_t, y_t\} = F(x_i, y_i)$ 
6:     考察计算误差  $\{x_e, y_e\} = \{x_t, y_t\} - \{x', y'\}$ 
7:     误差反向传播  $\{x_i, y_i\} = \{x_i, y_i\} - k\{x_e, y_e\}$ 
8:     if  $\|H(x_i, y_i)\|_\infty < \text{eps}$  then
9:       退出循环
10:    end if
11:   end for
12: end function

```

---

### 3 方案原理及误差分析

在通过合理的方式获得了逆变换的坐标之后，我们希望通过插值的方式利用逆变换点周围的若干点进行等间距插值。之所以采用逆变换之后插值，主要原因是这样变换之后可以利用等间距插值，不仅减少了算法的难度，而且提高了算法的精度。

考察这个过程中可能出现的误差，可能来源于如下的几个可能性。

#### 3.1 选取点，半径时的误差

由于选取中心点，半径的过程是用户手动选择的，在将鼠标位置映射到像素点坐标的过程中可能存在一个像素的误差，这个误差主要出现在将用户的鼠标操作离散化的过程中。同样，选取半径的过程中也受到这个离散映射的过程中，即选取的半径只能表现为  $r = \sqrt{n^2 + m^2}$ ,  $n \in \mathbb{Z}, m \in \mathbb{Z}$  这样使得变化的半径可能不是作者指定的半径。确切的说，是需要变换的点在原图像上组成的集合并不是一个完美的圆，而是圆内像素的集合。

#### 3.2 逆变换误差

这部分的误差，主要出现在对于双精度类型 double 的舍入误差上，显然，相对于后面几个计算的误差，这个误差相对较小，可以忽略。

另一方面，如果我们采用 B 样条插值，寻找一个点的原像过程是采用一个简单的数值计算方式得到的，如果迭代次数足够大，选择原像的误差应当小于 1，然而由于必须对迭代次数加以限制，这个误差的估计变得比较困难。然而，B 样条曲线插值本身就是一个交互性很强的方式，很多操作主要依靠用户根据图形的样式加以调整，这种计算误差往往会被用户通过进一步的操作进行消除，因此，讨论这个误差的意义也不大。

另外，由于尽可能避免极小数作为除数，使得计算过程中出现了病态，在中心点附近（往往是曼哈顿距离为 1 的那些点）直接避免了反算原始坐标点，而认为这个距离内的点变化后位置不变，显然，在保幅变换的情况下，这个变换的误差应当小于两个像素。而且这个误差仅仅存在于中心点周边，含中心点的 5 个像素中，因此这个误差在采用正常的图像，如 Lena（512 \* 512 像素）图像等问题时不足为虑。

#### 3.3 插值过程

实验过程中，实现了最近邻插值，双线性插值，双三次插值三种插值算法。首先为了方便之后的叙述，我们将整幅图片定义为将二维向量映射为四维向

量 (R,G,B,A) 的多元函数, 设连续情况下图像可导, 得到其 Jacobi 矩阵为

$$J = \begin{pmatrix} J_{11}J_{12} \\ J_{21}J_{22} \\ J_{31}J_{32} \\ J_{41}J_{42} \end{pmatrix}$$

并定义全微分形式为  $dy = Jdx$ 。对 Jacobi 矩阵每一个元素均取在二维向量定义域中的最大值得到梯度上界  $\max J$  我们认为照片的采集过程中每一个像素点都是准确的, 即没有观测误差。这里逐一将其工作原理和误差简述如下

### 3.3.1 最近邻插值

最近邻插值就是选用距离目标点最近的插值节点的颜色作为插值节点的颜色, 这是三种插值方式中最快的一种, 也是相对误差最大的一种。由于给定的 Lena 等图片的分辨率仍较好, 因此这种插值方式并没有出现明显的问题。

由于没有任何计算过程, 计算中的舍入误差就是像素点的舍入误差, 由于像素点中的每一维颜色是  $[0, 255]$  的整形, 因此我们可以知道对每一种颜色的误差均为  $\delta_1 = (0.5 \ 0.5 \ 0.5 \ 0.5)^T$ 。

对于插值误差, 我们可以很容易发现当目标点正好落到四个节点中间的时候误差最大, 此时误差可以被表示为如式 7 所示

$$\delta_2 = \max J \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix} \quad (7)$$

两种误差做合成, 得到插值的总误差为  $\delta = \sqrt{\delta_1^2 + \delta_2^2}$ , 上述操作对于  $\delta_i$  的每一个元素进行。

### 3.3.2 双线性插值

双线性插值考虑的是目标点周围的四个节点, 我们记这四个节点的对应值为  $a_{00}, a_{01}, a_{10}, a_{11}$ , 设目标点的小数部分

$$t_x = x - \lfloor x \rfloor, t_y = y - \lfloor y \rfloor$$

可以立即得到插值公式 8

$$t_y(t_x a_{11} + (1 - t_x)a_{01}) + (1 - t_y)(t_x a_{01} + (1 - t_x)a_{00}) \quad (8)$$

利用误差公式直接得到插值公式 8 的舍入误差为 9 所示。

$$\delta_1 = t_x t_y + t_y - t_x t_y + 1 + t_x t_y - t_x - t_y - t_x t_y + t_x \Delta = 0.5 \quad (9)$$

下面分析双线性插值的插值误差，为了方便起见，我们只对颜色维数的一维进行分析，其他维数同理。我们将整个二维平面强行分割成两个一维的直和。对两个方向上的二阶导数的最大值分别为  $M_1, M_2$ 。对两个维度进行先后插值。参考书上结论得到在一维上进行插值的误差限为  $\frac{M}{8}$ ，因此得到先后进行两次插值的误差为  $\frac{M_1}{8} + \frac{M_2}{8}$ ，其中第一项为第二次插值的算法直接引入的误差项，第二项为第一次插值引入并按照误差传递公式传导的误差项。我们不妨记  $M = \max\{M_1, M_2\}$ ，立即得到误差的表达式如 10 所示。

$$\delta_2 = \frac{M}{4} \quad (10)$$

两种误差做合成，得到插值的总误差为  $\delta = \sqrt{\delta_1^2 + \delta_2^2}$ 。

### 3.3.3 双三次插值

双三次插值相对于前两种插值方式，时间的损耗更大，但是插值的效果更好。具体的实现<sup>4</sup>的思路我们可以通过算法 4 表示 类比双线性插值，可以得到双三次插值，这里的实现方式仍是先对一维进行分段三次插值，在对另一位进行三次插值。我们假设某一维颜色在两个维度上的四阶导数的共同最大值为  $M$ 。同理立即得到两种误差的表达式如??所示。

$$\begin{cases} \delta_1 = 0.5; \\ \delta_2 = 2 \frac{M}{384}; \\ \delta = \sqrt{\delta_1^2 + \delta_2^2} \end{cases} \quad (11)$$

### 3.3.4 三种插值方法的比较

我们可以明显的看出，双三次插值的精度要高于双线性插值，同时高于最近邻插值。同样道理，双三次插值的时间消耗也明显高于双线性插值，双线性插值的时间消耗也明显高过最近邻插值。在将图像扭曲——还原的实验中<sup>5</sup>，双三次插值可以清晰的还原图像，而双线性插值则相对差一点，最近邻插值则出现了明显的图像虚化。这跟我们之前的理解也是符合的。

同时，值得说明的一点是上述的插值方法是在假定了原来的连续图像存在的情况下，以双三次插值为例，我们认为原来连续图像在插值节点的导数，二阶导数连续。实际上这种情况在现实情况中不一定成立，但是这个问题是由原图像的离散化带来的，我们不在插值这里分析

<sup>4</sup> 参考了 <https://stackoverflow.com/questions/20923956/bicubic-interpolation> 并进行了修改

<sup>5</sup> 即将图像顺时针旋转扭曲若干次之后在进行逆时针的反变换的操作

---

**Algorithm 4** 双三次插值

**Require:** 周围的四个点的值  $v_0, v_1, v_2, v_3$  以及目标点的小数部分  $frac$

**Ensure:** 插值结果  $result$

```

1: function 三次插值 CUBIC( $v_0, v_1, v_2, v_3, frac$ )
2:    $A = (v_3 - v_2) - (v_0 - v_1)$ 
3:    $C = v_2 - v_0$ 
4:    $D = v_1$ 
5:   return  $D + frac * (C + frac * (B + frac * A))$ 
6: end function

```

**Require:** 周围 16 个点的值  $v_{00}, \dots, v_{33}$ , 插值位置小数部分  $frac_x, frac_y$

**Ensure:** 双三次插值结果  $result$

```

1: function 三次插值 ( $v_{00}, \dots, v_{33}, frac_x, frac_y$ )
2:   for  $i = 0; i < 4; ++i$  do
3:      $v_i = Cubic(v_{i0}, v_{i1}, v_{i2}, v_{i3}, frac_y)$ 
4:   end for
5:   return  $Cubic(v_0, v_1, v_2, v_3, frac_x)$ 
6: end function

```

---

### 3.4 图像输出的误差

在进行了插值之后，我们需要将图像色彩归一化到  $[0, 255]$  区间的整数，这中间包括取整所带来的误差，也包括上下限制所带来的误差，取整带来的误差仍为 0.5 个颜色单位，上下限制带来的误差没有意义讨论，这是因为实际图像中也不可能出现超过这个范围的数值。实际上，细致的讨论几种插值方法，均是周边几个点的线性组合，且他们的权重和为 1，由于周边几个点均在  $[0, 255]$  区间中，因此出现上限下限的点实际上不可能被插值函数获得，加入这一步仅仅是为了保证程序的运行安全，避免因为误计算导致程序崩溃。

## 4 实验结果分析

我们采用了 Lena 和网格图进行了测试，如下是三种变换和三种插值方法之后对应的效果图 可以看出几种插值算法的效果都还比较理想，对于 B 样条插值，双三次插值的平滑性要好于双线性插值和最近邻插值。同时，我们可

Tab. 1: Lena 各种方案的测试效果图

图像变换	最近邻插值	双线性插值	双三次插值
图像旋转			
水波纹变换			
一次 B 样条插值			
三次 B 样条插值			

以看出一次 B 样条插值的光滑性不如三次 B 样条插值的光滑性，当然，双三次插值，三次 B 样条插值的时间消耗也更长。实验过程中，旋转角度为 125 度，水波纹幅值为 1，波长为 5，初始相位角为 0.

Tab. 2: 网格图各种方案的测试效果图

图像 变换	最近邻插值	双线性插值	双三次插值
图像旋转			
水波纹变换			
一次 B 样条插值			
三次 B 样条插值			