四　川　大　学

本科专业选修课课程考核

姓　　名　黄智忠　　学　　号

学院（所、中心）　软件学院　　专　　业　软件工程

课程名称　并行处理

课序号　311168020

任课教师　尹皓

2018 年春

评语：

一. 简介

1.1 主要负责的工作

我主要负责的工作有：

1. B超图像扫描转换的GPU实现，邻近插值、双线性插值、双三次插值三种插值算法GPU版本实现。
2. 对优化算法的结果进行评估，图像品质和运行时间测量，优化效率通过加速比(加速比=CPU运行时间/GPU 运行时间)来衡量，图像质量通过图像的峰值信噪比(Peak Signal to Noise Ratio, PSNR)、均方误差(Mean Squared Error, MSE)和结构相似性 (Structural Similarity Index, SSIM)三个指标进行衡量。

1.2 开发环境

1. 平台：MacBook Pro 2015 Early 13’
2. 显卡类型：Intel(R) Iris(TM) Graphics 6100
3. 标准：C++11
4. 操作系统：macOS High Sierra
5. API：OpenCL 1.2

其他平台包括NVIDIA GeForce 750Ti，NVIDIA Tesla M40；操作系统为Ubuntu，OpenCL 1.2。

二. 扫描转换的简单介绍

本实验需要对超声图像进行扫描转换，超声成像的基本原理就是:向人体发射一组超声波,按一定的方向进行扫描，每次扫描得到一条一定长度的数据，将所有得到的数据按行储存就得到了原图像。因此扇形扫描转换是对超声前端采集的等角度的波束进行从极坐标(θ, r)到笛卡儿坐标(x, y)的转变。但信号样本点与显示像素点并不是一一对应，相邻扇形扫描 线之间还有很多空缺的显示像素，这就需要对位于图像的横轴线与扫描线交点的像素点，沿着扫描线方向进行补充，以及沿横轴线对像素进行填充。

我们需要根据原图像和扫描的深度、角度、半径等数据恢复出原探头扫到的扇形空间。

扇形扫描转换主要有4个步骤：参考坐标系的转换，显示像素点坐标映射到原始采样点坐 标，沿扫描线方向对图像的横轴线与扫描线交点像素的插值，沿横轴线对像素的填充。其中，填充方法可使用不同的插值算法来实现。

2.1 参考坐标系的转换

超声系统前端信号采样得到信号样本点，建立如图1所示的坐标系，而扫描转换后的像素点分布如图2所示，为了便于计算，首先需要对图2的坐标进行平移，平移到如图3的参考坐标系中。如图3中Y_{shift}表示将图2的坐标平移到图3的坐标系中需要沿Y轴方向移动的距离，X_{shift}是将图2的坐标平移到图3的坐标系需要沿X轴方向移动的距离。

$$X_{shift} = W_{Image}/2$$

$$Y_{shift} = SectorRadius - H_{Image}$$

其中: W_{Image}和H_{Image}分别是扫描转换结果图像的宽和高。原始采样点坐标$(X_{original}, Y_{original})$ 在图3的参考坐标系下的坐标定义为(X_{target}, Y_{target})：

$$X_{target} = X_{original} - X_{shift}$$

$$Y_{target} = Y_{original} + Y_{shift}$$

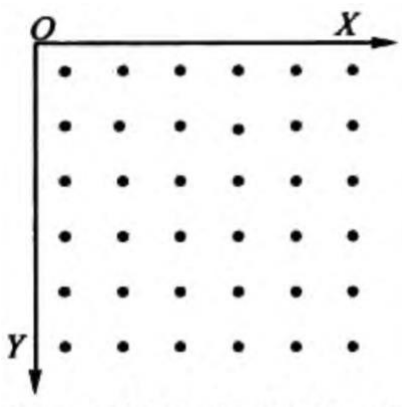


图1 原始信号采样点示例

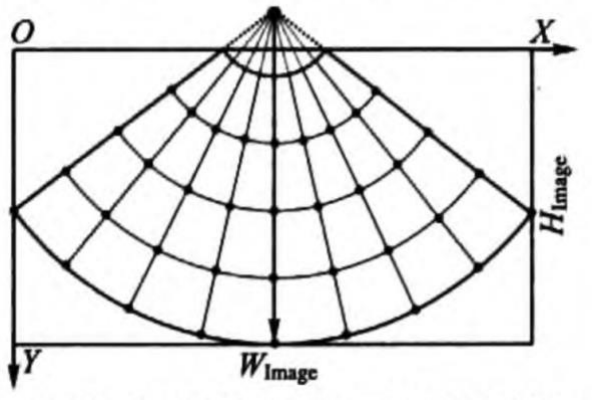


图2 扇形扫描转换的显示像素点与回波信号采样点的位置关系

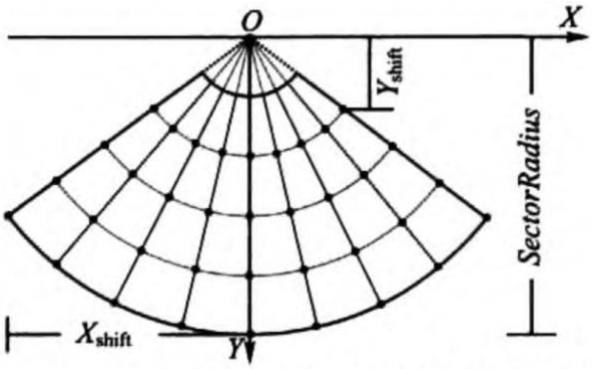


图3 坐标平移后显示像素点与回波信号采样点的位置关系

2.2 计算显示像素点坐标对应原始采样点坐标

经过第一步坐标转换后.需要将显示像素点坐标映射到 采样点的极坐标系中，利用以下公式求显示像素点所在的扫描线与中心扫描线的夹角:

$$\theta = \arctan(X/Y)$$

显示像素点在该扫描线上的深度Depth是:

$$Depth=\sqrt{x^2+y^2}$$

然后，由夹角0，深度Depth可以得到显示像素点在原始

采样坐标系中对应的采样点坐标(X_{org},Y_{org})，但是这个采样点并不真实存在，所以需要求与该点最邻近的原始采样点的坐标$(X，Y)$，以及与原始采样点$(X，Y)$的偏移量。然后通过适当的插值方式求得显示像素点值。

$$X_{org} = (\theta - \frac{\varphi}{2})/\theta_0$$

$$X = IDn = \left\lfloor (\theta - \frac{\varphi}{2}) / \theta_0 \right.$$

$$X_{offset} = \frac{\theta - \frac{\varphi}{2}}{\theta_0} - X$$

$$Y_{org} = Depth - \mathrm{ProbeRadius}$$

$$Y = \lfloor (Depth - \mathrm{ProbeRadius})$$

其中：ProbeRadius是探头半径，X_{offset}是显示像素点相对原始采样点沿X轴方向的偏移量，Y_{offset}是沿Y轴方向的偏移量。

三．使用到的插值方法和插值策略

3.1 邻近插值

最简单的插值算法是最邻近插值，也称为零阶插值。它输出的像素灰度值就等于距离它映射到的位置最近的输入像素的灰度值，最邻近插值算法简单，在许多情况下都能得到令人满意的结果，但是当图像中包含像素之间灰度级有变化的细微结构时，最邻近算法会在图像中产生人为加工的痕迹。双线性插值算法计算量比零阶插值大，但缩放后图像质量高，不会出现像素值不连续的的情况，这样就可以获得一个令人满意的结果。

最邻近点插值取插值点的4个邻点中距离最近的邻点灰度值作为该点的灰度值。设插值点(i, j)到周边4个邻点f_k(i, j), (k = 1, 2, 3, 4)的距离为d_k (k= 1, 2,3,4)，则：

$$g(i,j) = f_k(i,j),\ d_l = min\{d1, d2, d3, d4\}, \qquad l = 1,2,3,4$$

3.2 双线性插值

双线性插值是利用了需要处理的原始图像像素点周围的四个像素点的相关陛，通过双线眭算法计算得出的。对于一个目的坐标，通过向后映射法得到其在原始图像的对应的浮点坐标($i+u$，$j+v$)，其中i，j均为非负整数，u，v为[0，l]区间的浮点数，则这个像素的值f(i + u，j + v)可由原图像中坐标为(i，j)、(i + l，j)、(i，j + 1)、(i + 1，j + 1)所对应的周围四个像素的值决定，即：

f(i + u，j + v) = (1 − u) × (1 − v) × f(i，j) + (1 − u) × V × f(i，j + 1) + u × (1 − v) × f(i + l，j)
+ u × v × f(i + l，j + 1)

其中f(i，j)表示源图像(i，j)处的的像素值，以此类推，这就是双线性内插值法。

如图1所示，已知(0，0)、(0，1)、(1，0)、(1，1)四点的的灰度，可以由相邻像素的灰度值f(0，0)和f(1，0)在X方向上线性插值求出(x，0)的灰度f(x，0)，由另外两个相邻像素f(0，1)和f(1，1)在X方向上线性插值可求出(x，1)的灰度f(x，1)，最后由f(x，0)，f(x，1)在Y方向上进行线性插值就可以得到(x，y)的灰度f(x，y)。

在同一行内根据待插值像素点与其前后的原图像像素点的位置距离进行加权线性插值，即离原图像像素点越近的待插值像素点，原图像像素的加权系数就越大；行间根据待插值行与其上下的原图像行间的距离进行加权线性插值，即离原图像行越近的待插值行，原图像行的加权系数就越大。

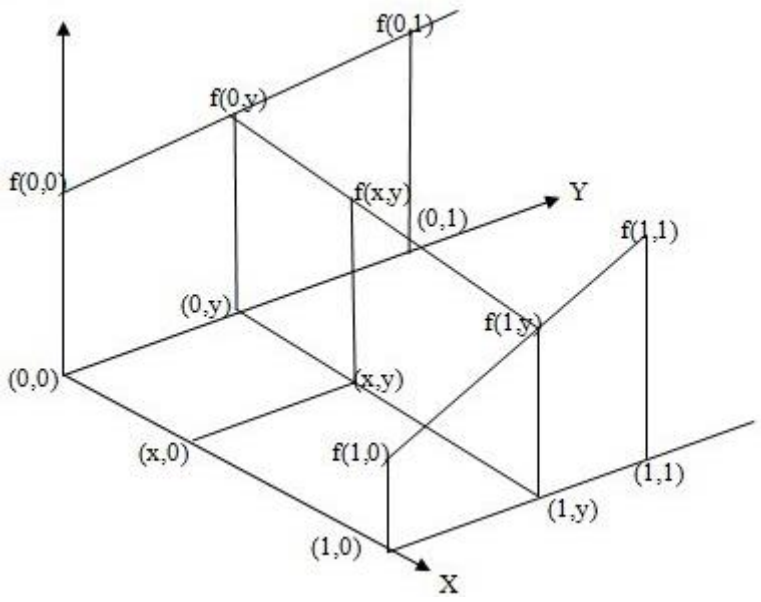


图3 双线性插值原理

基于双线性插值的程序流程图如下图4所示：

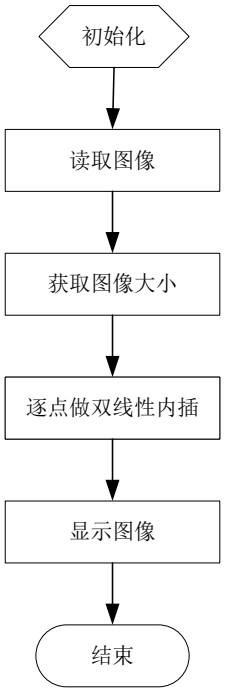


图4 程序流程图

3.3 双三次插值

双三次插值又称立方卷积插值。三次卷积插值是一种更加复杂的插值方式。该算法利用待采样点周围16个点的灰度值作三次插值，不仅考虑到4个直接相邻点的灰度影响,

而且考虑到各邻点间灰度值变化率的影响。三次运算可以得到更接近高分辨率图像的放大效果，但也导致了运算量的急剧增加。这种算法需要选取插值基函数来拟合数据,其最常用的插值基函数如图5所示，本次实验采用如图所示函数作为基函数。

$$W(x)=\begin{cases}(a+2)|x|^3-(a+3)|x|^2+1, & |x|\le 1\\ a|x|^3-5a|x|^2+8a|x|-4a, & 1<|x|<2\\ 0, & otherwise\end{cases}$$

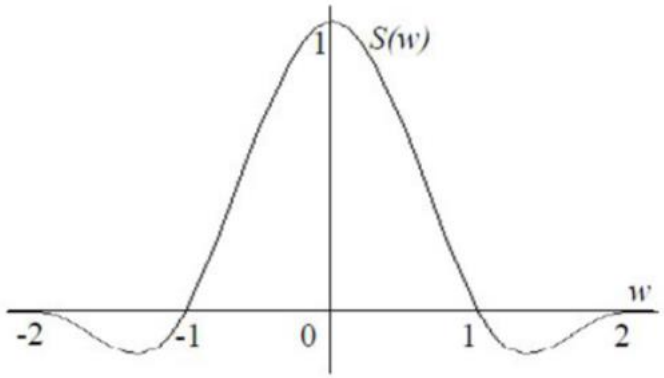
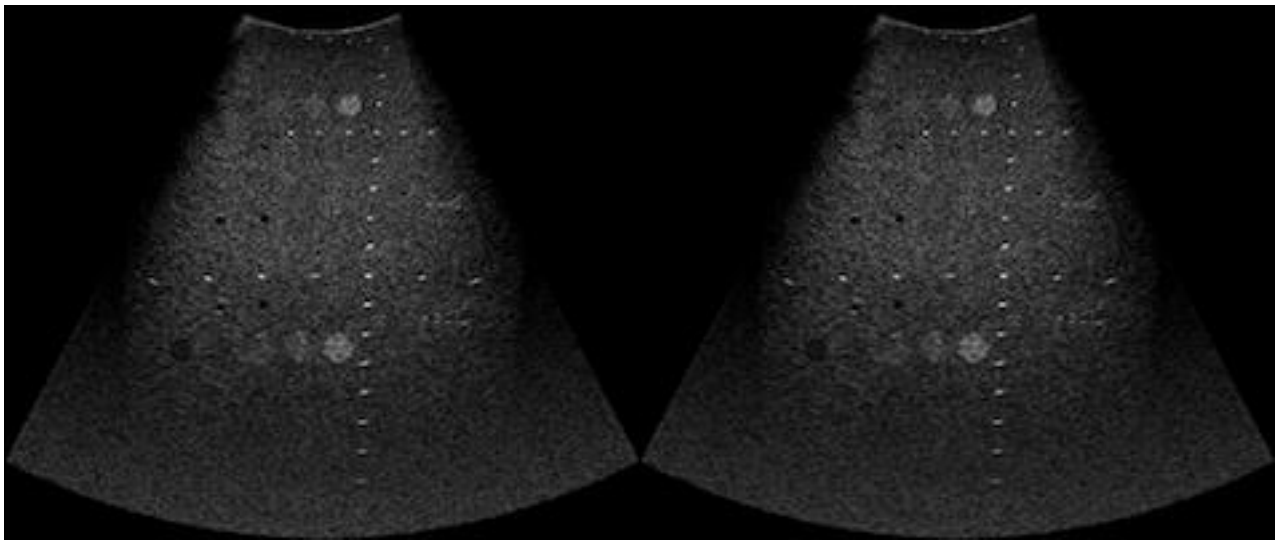


图5　插值基函数曲线

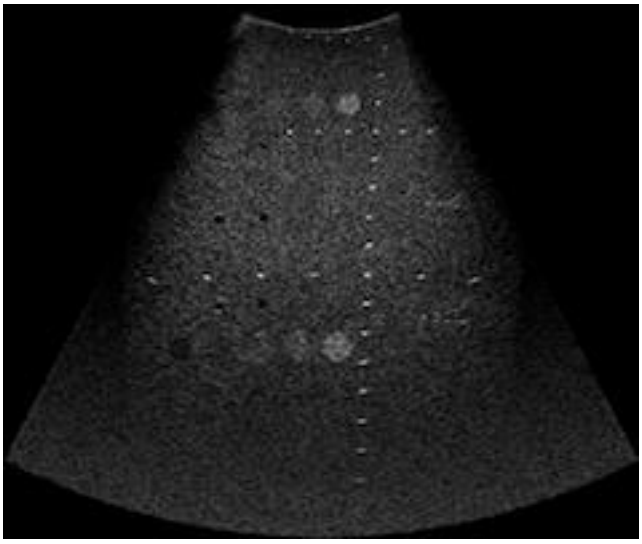
四．运行结果

图像品质：

插值方式	PSNR	MSSIM	MSE
邻近插值	31.03	0.31	51.32
双线性插值	39.51	0.33	7.27
双三次插值			



邻近插值　　双线性插值



双三次线性插值

图六 效果图

数据分析

以下数据均为运行counts = 5 次，取平均值。
未调优

一、不同 groupSize

1. $\text{groupSizeX} = 1, \text{groupSizeY} = 1$

插值方式	CPU 运行时间（ms）	GPU 运行时间（ms）	加速比
邻近插值	70.57	18.82	3.751
双线性插值	66.30	18.59	3.567
双三次插值	1956.69	129.18	15.146

2. $\text{groupSizeX} = 2, \text{groupSizeY} = 2$

插值方式	CPU 运行时间（ms）	GPU 运行时间（ms）	加速比
邻近插值	68.87	5.64	12.202
双线性插值	69.75	5.91	11.809
双三次插值	1957.58	36.15	54.156

3. $\text{groupSizeX} = 4, \text{groupSizeY} = 4$

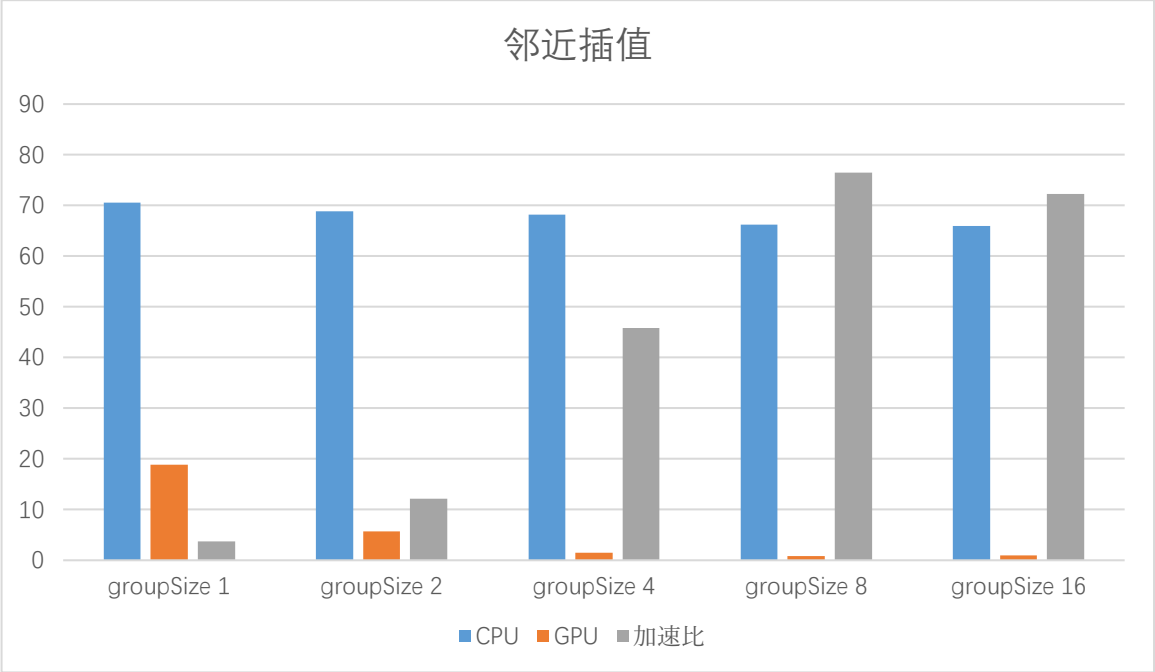
插值方式	CPU 运行时间（ms）	GPU 运行时间（ms）	加速比
邻近插值	68.14	1.49	45.872
双线性插值	65.32	1.63	40.039
双三次插值	1933.93	20.20	95.734

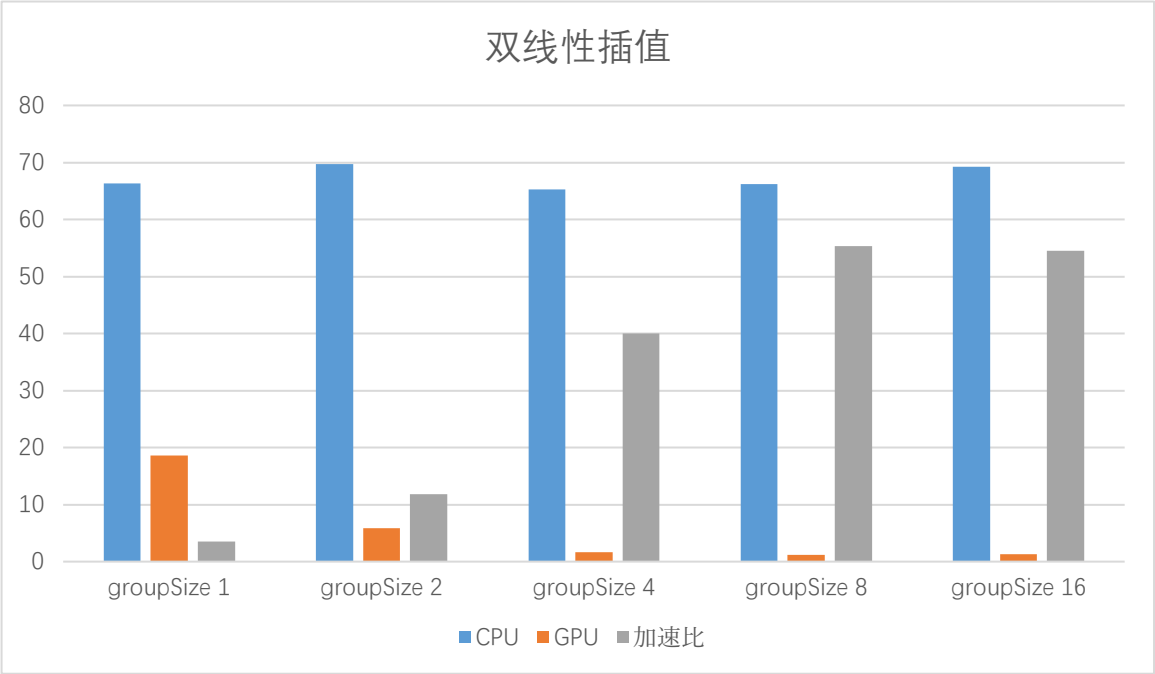
4. $groupSizeX = 8, groupSizeY = 8$

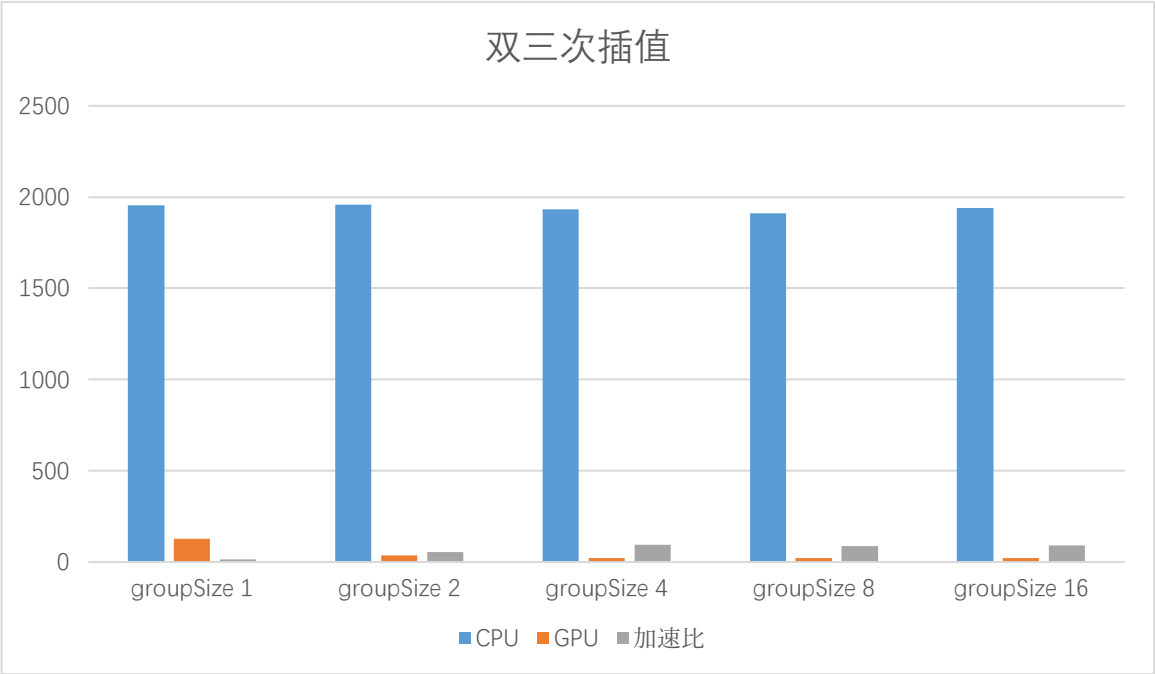
插值方式	CPU 运行时间（ms）	GPU 运行时间（ms）	加速比
邻近插值	66.20	0.86	76.537
双线性插值	66.16	1.20	55.328
双三次插值	1911.14	21.91	87.244

5. $groupSizeX = 16, groupSizeY = 16$

插值方式	CPU 运行时间（ms）	GPU 运行时间（ms）	加速比
邻近插值	65.93	0.91	72.262
双线性插值	69.25	1.27	54.469
双三次插值	1939.50	20.91	92.743







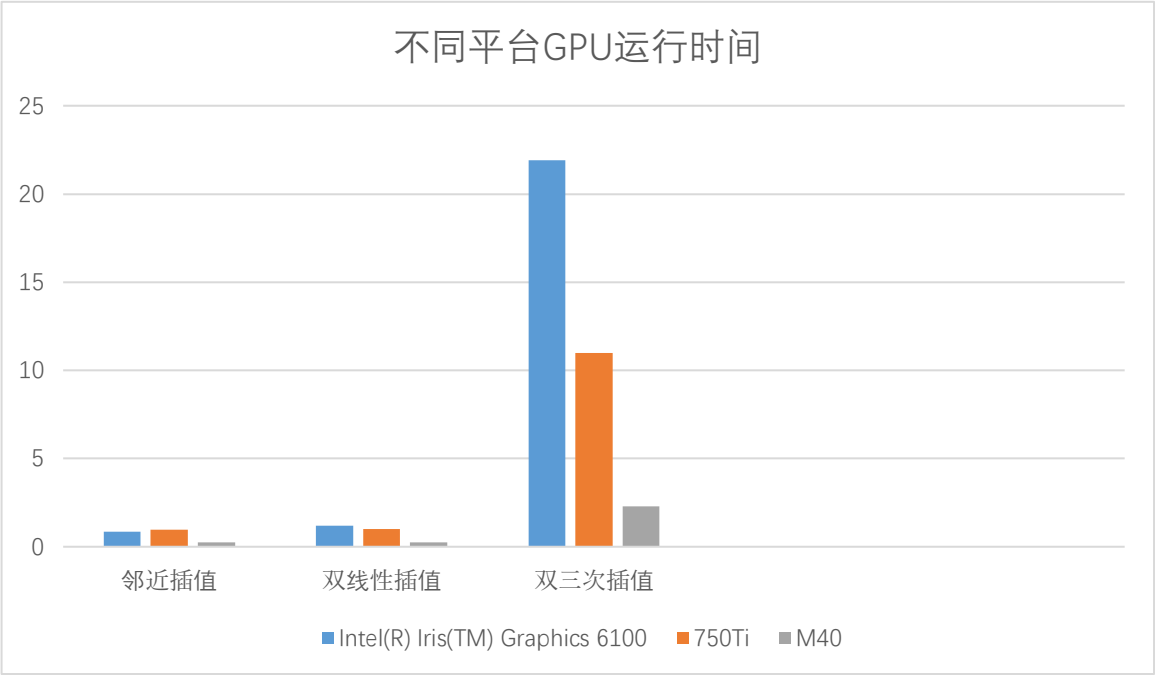
因此可以总结出，当$groupSizeX = 8, groupSizeY = 8$时，取得最好的性能。

二、不同显卡

当$groupSizeX = 8, groupSizeY = 8$时，分别是MacBook Pro集显、NVIDIA GeForce 750Ti，NVIDIA Tesla M40。

算法	显卡类型	CPU 运行时间（ms）	GPU 运行时间（ms）	加速比

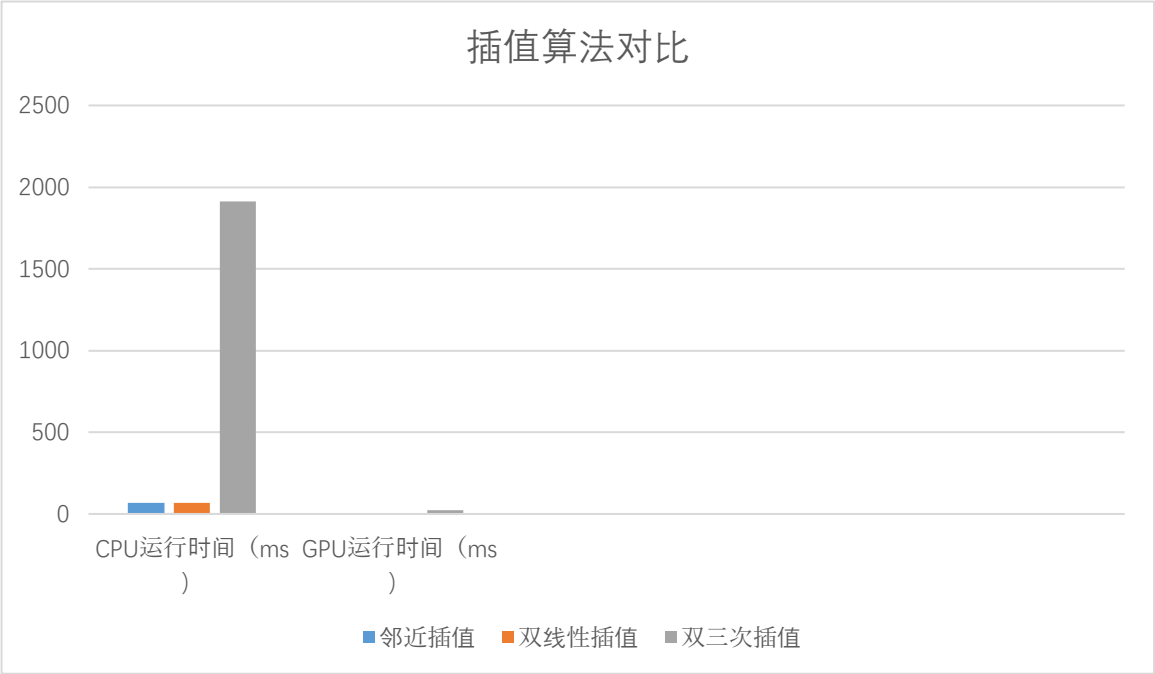
邻近插值	Intel(R) Iris(TM) Graphics 6100	66.20	0.86	76.537
	750Ti	104.07	0.98	105.785
	M40	79.13	0.26	310.250
双线性插值	Intel(R) Iris(TM) Graphics 6100	66.16	1.20	55.328
	750Ti	103.51	0.99	104.456
	M40	80.61	0.26	312.685
双三次插值	Intel(R) Iris(TM) Graphics 6100	1911.14	21.91	87.244
	750Ti	3074.60	10.99	279.745
	M40	2331.63	2.30	1013.285



从以上数据可知，所以显卡性能越强，所需花费时间更短，插值效率更快。

三、插值对比

$groupSizeX = 8, groupSizeY = 8$



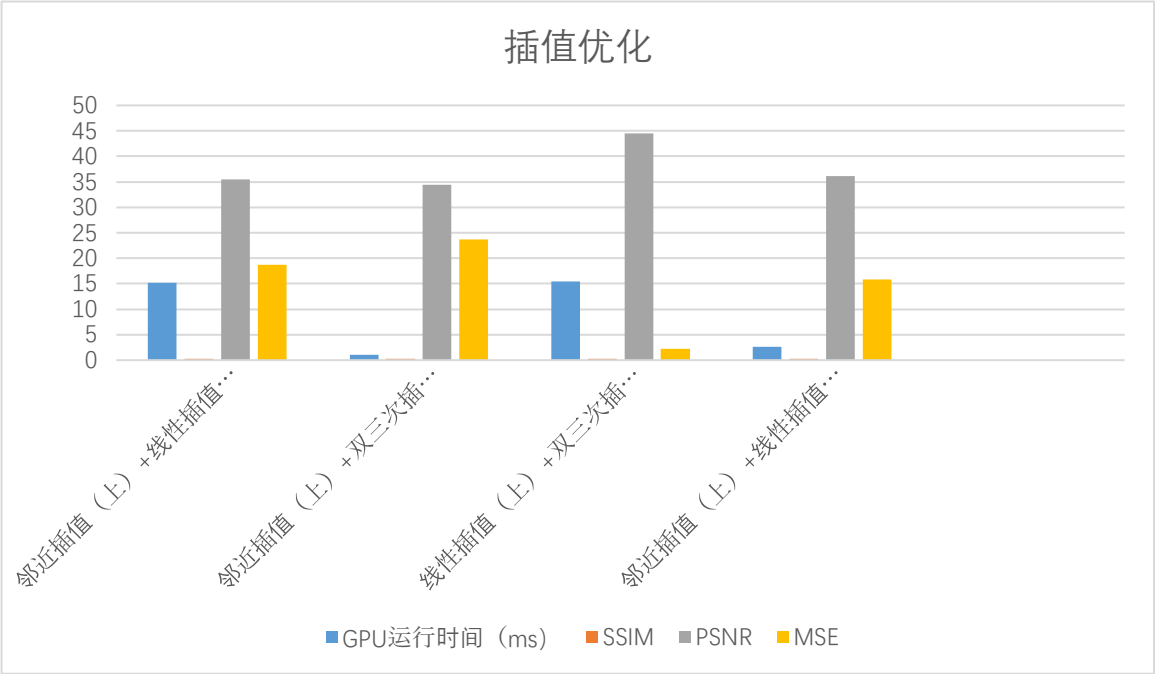
从以上数据可知，插值算法所需花费时间依次为邻近插值、双线性插值、双三次插值，其中双三次插值速度远远慢于其他俩种。

四、插值调优

在以上基础上，GPU为Intel(R) Iris(TM) Graphics 6100，$groupSizeX = 8, groupSizeY = 8$，对插值策略进行调优：

其中，主要是按照上、中、下三部分或者上、下两部分分别使用不同的算法，由于图像上半部分的数据少、拉伸幅度 较小，因此可以采用速度相对较快的简单插值进行处理，而图像下半部分的数据多、拉伸幅度较大，因此选用较复杂的插值算法可以得到一个相对较好的图像质量，两种插值算法的组合较双三次插值也减少了一定的计算量。

插值方式	GPU 运行时间（ms）	SSIM	PSNR	MSE
邻近插值（上）+线性插值（下）	15.22	0.33	35.42	18.67
邻近插值（上）+双三次插值（下）	1.02	0.32	34.39	23.66
线性插值（上）+双三次插值（下）	15.43	0.33	44.55	2.28
邻近插值（上）+线性插值（中）+双三次插值（下）	2.60	0.33	36.13	15.85



因此，可以明显看出邻近插值（上）+线性插值（中）+双三次插值（下）这种方法取得了性能和误差上的平衡。