Fermi 架构下超声成像组织运动可视化并行算法①

何兴无

(成都师范学院 网络与信息管理中心, 成都 6111309)

摘 要: 在临床超声实时成像系统中组织运动情况是医生想要获取的重要诊断信息, 例如心脏运动. 基于线积分卷积的二维矢量场可视化技术可以同时展现运动矢量场的强度和方向. 但这一算法在处理时涉及大量的复杂计算, 尤其是流线追踪处理部分, 使其成为临床实时成像系统中的一大性能提升瓶颈. 为此研究并提出了一种基于新兴的高性能并行计算平台 Fermi 架构 GPU(graphics processing unit 图形处理单元)的并行运动可视化算法. 数据测试结果显示, 与基于 CPU 的实现相比, 采用 Fermi 架构的 GPU 处理不仅可以得到一致的运动可视化和信息分析效果, 而且可以取得较大的加速效果. 对于 260×260 的图像数据在使用线积分卷积滤波器长度为 7 的情况下, 速度提高了大约 237 倍.

关键词: 高性能并行计算; 运行可视化; 线积分卷积; 并行处理; 图形处理单元

Parallel Processing Algorithm of Tissue Motion Visualization for Ultrasound Imaging on Fermi

HE Xing-Wu

(Network & Information Management Center, Chengdu Normal University, Chengdu 611130, China)

Abstract: On the clinical ultrasound real-time imaging system, tissue motion will be the important diagnostic information for doctors, such as cardiac motion analysis. In this paper, one motion visualization method based on line integral convolution (LIC) is studied. This method could show the vector field information about both direction and intensity. However because of the massive computation involved in this filter technique, especially in streamline tracking, it has been the bottleneck for the clinical real-time imaging system. In this paper, a new parallel algorithm of motion visualization based on Fermi GPU (graphics processing unit) is presented. Test results not only show the output of graphics processing unit (GPU) is definitely the same as the one of CPU, but also demonstrate the obvious speedup using GPU, that is, it can be 237 times faster than the CPU implementation with the long LIC filter (L=7) for the image size (260×260).

Key words: high performance parallel processing; motion visualization; LIC; parallel processing; GPU

1 概述

医学数字彩超系统作为一种疾病损伤临床检测的重要诊断工具, 在医学诊断中发挥难以替代的作用[1]. 而科学可视化作为计算机科学的重要领域其应用领域也随着技术的发展而更加广泛. 组织运动的可视化是彩超系统中非常重要的应用之一. 可视化技术主要是能够提取组织运动场信息并为医生进行临床诊断提供更为直接的组织运动分析依据[2]. 因此设计好的超声图像组织运动可视化算法具有非常重要的临床意义.

Cabral 和 Leedom 引入线积分卷积用于成像和刻画向量场[3]. 对于非稳定向量场的可视化, 研究者又提出了非稳定场线积分卷积算法[4], 它可以用于实现超声心脏运动的可视化[5]. 但这种可视化算法在应用于实际临床彩超检测系统时, 由于其复杂的计算过程,

① 收稿时间:2012-09-27;收到修改稿时间:2012-12-08

大大降低了系统的帧速率. 限制了其在高实时性的医学超声检测系统中的应用.

如何加速运动可视化, 也已经提出了很多方法. 文献[6]给出了一种去掉冗余计算的快速线积分卷积算法. 尽管这种优化技巧减少了计算量加快了处理速度, 但要达到较好的处理帧率仍然需要定制硬件芯片, 如 DSP. 这会使系统的制造成本变得大大增加. 因此, 本文通过研究新近出现的具有通用计算能力的 Fermi 架构 GPU 并行处理平台, 提出了一种利用 GPU 通用计算能力的超声系统组织运动可视化并行处理算法.

与传统的 CPU 处理器不同, CUDA 架构下的 GPU 统一了芯片上原本分离的顶点和片段着色器, 使 GPU 上的计算资源可以更好地完成通用计算功能. 特别是随着 Fermi 架构的 CUDA2.0 以上的 GPU 设备的出现, GPU 的处理能力大幅度提高, 同时在缓存访问机制, 计算精度等方面的硬件优化也使其更加适合于数据的并行计算[7]. 因此 GPU 技术在各种应用领域中都取得了成功的应用[8,9].

2　基于Fermi的组织运动可视化并行处理算法

2.1 使用非稳定场线积分卷积可视化处理算法概述

本文所研究的超声系统组织运动可视化并行算法主要处理过程包括基于图像配准运动检测, 空间域和时间域的非稳定场线积分卷积计算, 彩色增强处理和局部运动的径向和切线方向速度分量计算等处理环节, 算法处理流程见图 1.

算法的第一步是要获取速度场, 本文使用的是块匹配的方法. 即从前一帧图像中选取多个 n×n 的像素块, 然后计算各个块的运动速度, 这里的匹配计算采用的是绝对误差和, 计算公式如下:

$$SAD(i,j)=\sum_{m=1}^{M}\sum_{n=1}^{N}|H^{(k)}(m,n)-H^{(k-1)}(m+i,n+j)| \tag{1}$$

其中 H 为像素块中的像素值. 具有最小的绝对误差和的块就被选择作为匹配最好的, 速度就可以由公式(2)得到:

$$v(V_x,V_y)=(j_c,i_c)-(j_0,i_0) \tag{2}$$

其中(j_0, i_0)表示图像块的中心位置, (j_c, i_c)表示的是最匹配快的中心坐标.

矢量场可视化的方法有很多, 其中基于纹理的方法不仅可以提供颜色信息, 而且当纹理中点的颜色按照一定的规则进行排列时, 纹理就有了形状, 这种形状就可以用来表示矢量场. 另外, 由于纹理是连续的图像, 基于纹理的方法可以克服箭标图和矢量线疏密造成图像杂乱的问题. 基于纹理的方法中最常用的就是线积分卷积的方法, 本文研究实现的是非稳定场线积分. 非稳定场线积分卷积是以白噪声纹理和连续的速度场作为输入沿着流线方向对每个像素点进行线积分卷积得到输出像素. 线积分卷积的基本思想如图 2 所示.

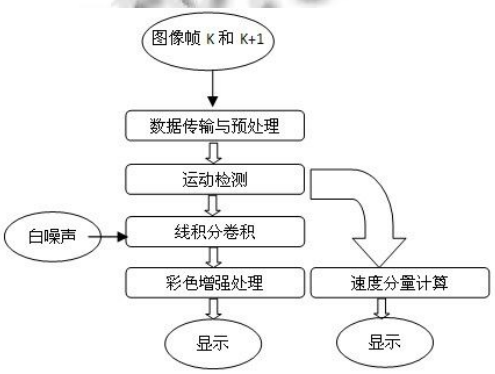


图 1　超声成像系统组织运动可视化算法的处理流程

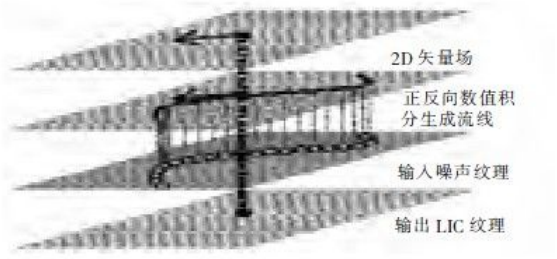


图 2　线积分卷积的示意图

为了计算图 2 中第一层上点对应的输出像素, 需要根据矢量场找出该点对应的流线, 如图 2 中第二层的线条即是从二维矢量场得到的流线, 再根据第二层的流线映射到白噪声纹理, 就找出了需要参与卷积的点, 如图 2 中第三层的点; 最后利用式(3)计算得到线积分的输出像素值.

$$I(x_0)=\frac{1}{2n+1}\left(\sum_{i=-n}^{n}T(x_i)\right) \tag{3}$$

其中 n 是流线的半长, T(xi)是流线上噪声图像的像素值.

在得到线积分卷积输出的灰度图后, 我们要对其进行彩色纹理增强处理, 使运动图可以较好地显示运

动方向和强度. 将用灰度表示的矢量场强度进行彩色化显示较好地增强了图像可视化中运动强度信息. 最后计算局部运动的径向和切线方向的速度分量, 并用时间函数描述.

2.2 数据准备和预处理

采用 GPU 进行数据处理, 首先需要将数据由内存传输到显存空间, 这里首先把图像序列第一、二帧传入 GPU, 下一次处理时用后继帧替换前面已完成处理的帧, 也就是用第三帧替换第一帧. 帧替换机制由主机端使用缓存后继帧的控制机制完成. 同时, 因为写结合模式可以使数据在通过 PCI-e 总线传输时不会被监视, 这能够获得高达 40%的传输加速, 是最适合 CPU 是只写的情况, 所以主机端采用这种工作模式提高传输效率, 即使用 cudaHostAlloc 分配主机端页锁定内存, 设置写结合模式 cudaHostAllocWriteCombined.

另一方面, 由于超声图像上存在噪声信息, 因此在进行运动检测之前需要对图像序列施以一个均值滤波处理将噪声平滑. 本文所设计使用的并行均值滤波算法是采用粗粒度的方式减少冗余计算. 对于每一帧图像的预处理在 GPU 上实现的具体做法是使用两个核函数:

第一个核函数采用的并行方式是让一个线程扫描一行的方式沿 X 方向计算滤波窗口尺寸的均值, 算法实现伪代码为:

__global__ void Ave_X(float *data, float *out, int w, int h, int r, float scale){//r 为窗口半径, scale 为窗口长度的倒数

//将线程标识号与所需要处理的行索引位置建立映射

int x = blockIdx.x * blockDim.x + threadIdx.x;

//进行横向的求和计算方式是采用加上下一个位置数据的同时减去上一个的方式来完成的, 输出为每一点的窗口均值. 伪代码如下:

for 线程对应的这一行每个像素点.

{

①加上后面一个像素点, 减去最先一个像素点

②表示的是当前像素点与左右相邻两个点的和, 再乘以 scale

③将结果写入 out

}

第二个核函数以同样的方式沿 Y 方向计算各种窗口尺寸的和, 并求对应的均值.

这里特别需要说明的是在沿 X 方向处理时, 线程对数据的访问不满足合并访问规则, 在 Fermi 架构的 GPU 下可以设置优先使用 L1 缓存, 来增加 L1 的空间大小以得到较好的性能.

2.3 运动检测

经过平滑处理后, 就可以使用块匹配的方法来获取速度场. 在寻找到最匹配的块时, 除了通过公式(1)得到块匹配的绝对误差和之外, 本文还采用了一种更为有效的搜索算法寻找到最匹配的块, 即梯度下降的搜索方法. 梯度下降算法是利用负梯度方向来决定每次迭代的新的搜索方向, 使得每次迭代能使待优化的目标函数逐步减小[10]. 通过前面对本处理环节的分析, 可以看到这里需要解决两个问题: 一个是进行像素块的绝对误差和计算, 另一个是搜索最小的绝对误差和.

对于第一个问题的并行解决方法是改进了的并行归约算法, 即让每一个线程首先计算像素块在行方向的绝对误差和, 然后再使用传统的并行归约算法得到整个像素块的绝对误差和. 对于 n×n 的像素块, 一般的直接计算方法需要 n2-1 次加法运算, 而本文设计的改进了的并行归约算法仅仅需要 n-1+logn 次加法. 值得注意的是这里虽然完全采用传统的并行归约算法只需要 2logn 次加法, 但它会增加大量的同步指令, 所以本文结合实际应用中的数据规模, 选择的是这种加法指令和同步指令总量更少的方式.

解决第二个问题的方式为: 对于点(i, j), 计算它左右邻域点的绝对误差和; 如果右边小, 就一直向右搜索直到寻找到一个非递减的点; 同理, 如果左边小, 就一直向左搜索直到寻找到一个非递减的点. 通过这样的搜索, 我们就可以找到 X 方向的具有最小绝对误差和的像素块, 然后再沿 Y 方向寻找, 最终得到具有最小绝对误差和的像素块.

2.4 非稳定场线积分卷积

非稳定场线积分卷积有两个需要考虑的问题, 一个是卷积核的选择, 一个是如何追踪流线. 对于卷积核的选择, 从运算速度和算法效果的平衡的角度考虑, 这里选择的是常数卷积核, 见公式(3). 为了避免流线追踪过程中的冗余计算, 这里使用一种基于查找表的方法, 即预先计算并存放在一个三维查找表中, 线积分卷积计算是沿着查找表里存储的流线完成的, 过程如下:

首先创建一张存放流线中每个像素上一点和下一点的坐标索引. 在这个方法里, 查找表一共有 N×M 个元素, 其中 N 表示矢量场的宽度, M 表示矢量场的高度, 每一个元素存储了该点在流线上的上一点的坐标和下一点的坐标. 也就是说, 对于矢量场的每个点, 查找表都有一个元素与之对应. 设计这样的存储方式是为了减少冗余存储. 例如, 如果按流线来存储, 即如果一共有 N 条流线, 那么就会有 N 个数组分别存储每条流线, 数组内部每个元素依次存储流线上点的坐标, 这样就会造成非常多的冗余. 由于点的横纵坐标都是无符号的整数, 按照这样的存储方式可以节省约 n 倍空间, n 是指流线长度的一半. 表的创建是基于块的方式, 也就是把矢量场分块.

分块后, 对于点(x_0, y_0), 它所在的块的中心点(x_c, y_c)可通过下式计算得到:

$$(x_c, y_c)=\left(\left\lfloor\frac{x_0}{L_b}\right\rfloor\times L_b+\frac{L_b}{2},\left\lfloor\frac{y_0}{L_b}\right\rfloor\times L_b+\frac{L_b}{2}\right) \tag{4}$$

其中 Lb 是块的长度. 对于点(x_0, y_0)在流线上的下一个点(x_n, y_n)可以由该点的斜率来得到. 首先得到斜率 k, 定义为:

$$k=\frac{V_y}{V_x} \tag{5}$$

其中(V_x, V_y)为(x, y)处的矢量. 当 k>1 时, 交换 x 和 y, 要计算(x_0, y_0)的上一个点, 只需要将(V_x, V_y)改为(-V_x, -V_y); 否则, 按下式计算:

$$\begin{cases}(x_1,y_1)=(x_0+1,y_0) & offset\le 0.5\\(x_1,y_1)=(x_0+1,y_0+1) & offset>0.5\end{cases} \tag{6}$$

其中

$$offset=(x_0-x+1)\times k-\lfloor(x_0-x+1)\times k\rfloor \tag{7}$$

构建完查找表就可以对流线进行卷积. 计算方法是首先找出一个点 P 在流线上的上一点 P_{prev} 和下一点 P_{next}; 接下来找出 P_{prev} 在流线上的上一点和在下一点 P_{next}, 判断得到的两个点的坐标是否是在白噪声之内以及流线长度是否达到了预先设定值, 如果在白噪声之内并且流线长度没有达到预定值, 则继续向前向后搜索; 否则结束搜索. 对得到的流线进行卷积, 前面的搜索过程会得到一系列的坐标, 根据这些坐标在白噪声中找到相应的像素值, 带入卷积式进行计算, 得到该点的输出像素值.

这一步骤使用两个 GPU 核函数来完成, 一个进行表的创建, 另一个进行线积分卷积运算. 表的创建过程中, 各个点之间的运算完全独立, 可以直接并行化. 与此同时, 正如前面所述, 为了减少流线追踪过程的大量冗余计算, 这里采用了查表方式. 在 GPU 上, 我们使用三维纹理空间来存放这个表, 存放方式与数据序列保持一致, 只是每个点存放上一点和在下一点的坐标. 使用 cudaMalloc3DArray 分配 CUDA Array 存储空间存放图像数据. 数据生成后传入 CUDA Array, 利用 cudaBindTexture 绑定到物理存储器. 注意在绑定前需要设置纹理通道参数来决定纹理拾取的工作条件, 在后续使用时由 tex3D 函数进行拾取.

第二个核函数就根据创建好的查找表沿着正向和逆向同时进行流线追踪, 由公式(3)知, 各个像素点之间的处理完全独立, 可以直接并行化. 在线程网格设计时, 为了避免在地址计算时出现的整数求商求模这样的低吞吐率的运算指令, 本文设计使用的是一种二维网格和二维线程块的线程结构, 定义如下:

$$B_{xDim}=\left\lceil\frac{width}{T_{xDim}}\right\rceil\quad B_{yDim}=\left\lceil\frac{width}{T_{yDim}}\right\rceil \tag{8}$$

其中 B_{xDim},B_{yDim} 分别表示线程网格的 x 方向和 y 方向上的线程块数目. T_{xDim} 和 T_{yDim} 分别表示同一个线程块内部 x 方向和 y 方向上线程数目, 对于 Fermi 架构的 GPU 线程块内 T_{xDim} 采用 32 的倍数最好, 本文设置为 32. width 和 height 分别为目标输出图像的宽度和高度.

2.5 彩色增强处理

图像的彩色增强处理可以看成是对线积分卷积输出 I1 和运动矢量场强度图像的一个线性插值, 计算公式如下:

$$I=(1-\beta)\times I_1+\beta\times A \tag{9}$$

其中, β 为一个取值范围在 0 到 1 之间的调节系数.

$$A=\frac{\sqrt{V_x^2+V_y^2}}{L}\times 255 \tag{10}$$

其中, L 为速度场最长向量的长度. 由公式(9)和(10)可知, 这一环节的计算对于每个像素点之间是完全独立的, 因此可以直接并行化. 这里主要设计了 4 个 CUDA 核函数来执行这一处理环节. 第一个核函数计

算个像素点的运动强度; 第二个核函数使用并行归约算法获得最大运动强度; 第三个核函数将运动强度图像映射到 0~255 之间; 第四个核函数利用公式(9)完成增强处理, 并读取预定义的一个彩色查找表, 将灰度图像映射为彩色图像. 所有核函数的线程结构设计如前面所述的一种二维网格和二维线程块的线程结构.

2.6 计算局部径向/切线方向的速度分量

最后我们要得到的是用户感兴趣区域的切线方向和径向的速度(v_r 和 v_t). 为了得到一个用户感兴趣区域内所有像素点的平均速度, 我们需要通过公式(12)来决定有哪些点属于这个区域. 然后对区域内的像素点进行坐标转换得到在速度场的坐标位置, 转换公式见(13).

$$Set_{ROI}=\left\{(x,y)\,|\,R-\frac{w}{2}\le x\le R+\frac{w}{2}\,\&\&-\frac{h}{2}\le y\le\frac{h}{2}\right\} \tag{12}$$

$$\begin{pmatrix}x'\\y'\end{pmatrix}=\begin{pmatrix}\cos\alpha & -\sin\alpha\\ \sin\alpha & \cos\alpha\end{pmatrix}\begin{pmatrix}x\\y\end{pmatrix}+\begin{pmatrix}x_0\\y_0\end{pmatrix} \tag{13}$$

其中

$$\alpha=\arccos\left(\frac{x_c-x_0}{R}\right) \tag{14}$$

在得到速度场的坐标位置后, 就可以找出各个点的速度并利用公式(15)计算区域的平均速度.

$$\bar{v}=\frac{\sum_{p\in Set_{ROI}}v(p)}{N} \tag{15}$$

其中 v(p)表示点 p 处的速度, N 为 set 中点的个数.

从而我们可以得到切线方向和径向的速度, 计算公式如下:

$$v_r=\frac{\vec{v}\cdot\vec{R}}{|\vec{R}|}\quad v_t=\frac{\vec{v}\cdot\vec{R}_\perp}{|\vec{R}_\perp|} \tag{16}$$

其中 $\vec{R}$ 为 $(x_c\text{-}x_0, y_c\text{-}y_0)$, $\vec{R}_\perp$ 与 $\vec{R}$ 垂直.

由计算过程可知, 区域内各个点的运算是相互独立的, 可以直接并行化, 这里主要设计了 3 个 GPU 核函数. 第一个是使用了如前面所述的二维网格和二维线程块的线程结构进行各个点的坐标转换运算. 第二个核函数使用并行归约求和算法, 线程结构为数据规模的一半, 并计算出平均速度. 第三个核函数使用与第一个核函数一样的线程结构同时计算切线方向和径向的速度.

3 实验结果与分析

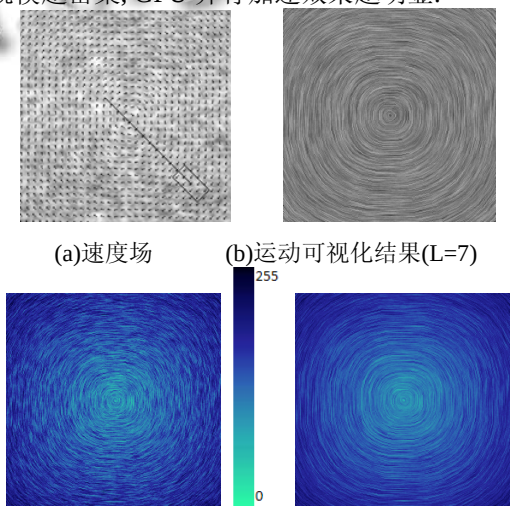
本文实验平台为 2.4GHz 的 Intel Core2 Duo P8600 CPU, 2GB DDR2 RAM,操作系统为 Windows 7. GPU 为 NVIDIA Geforce GTX 560 Ti, 显存为 1GB, 核心频率 1.645GHz,14 个多处理器, 使用 4.0 版本的 CUDA toolkit 及对应的开发包. 编程环境为 Visual Studio 2010.

本次研究所使用的仿真图像序列是利用 iMago C21, Saset Healthcare Inc 在超声体模上获取的, 速度场是已知的, 图 3(b)到(d)显示的是本文实现的在不同线积分卷积滤波器长度 L 条件下的运动可视化灰度与彩色效果, 可以看到是符合图 3(a)中定义的旋转场运动效果的. 图 4 上显示的是图 3 上感兴趣区域的理论速度曲线. 本文实现的方法结果通过图 2 可知基本与理论速度曲线一致, 存在的误差在可接受范围内, 这里 GPU 的输出与 CPU 是完全一致的.

表 1 性能比较

数据尺寸 (260×260)	CPU 运行时间(ms)	GPU 运行时间(ms)	加速比
L=3	106.88	0.65	164.43
L=5	170.17	0.78	218.17
L=7	232.03	0.98	236.77

表 1 给出了本文所研究的运动可视化算法 CPU 和 GPU 处理的性能比较. 程序的运行时间是图像序列进入显存后进行可视化处理的全部运行时间. 从表 1 可以明显看出, 本文所提出的基于 Fermi 架构 GPU 并行处理算法相比较于传统的 CPU 串行处理对于长线积分情况卷积提高了大约 237 倍. 加速比也说明了数据计算规模越密集, GPU 并行加速效果越明显.



(a)速度场 (b)运动可视化结果(L=7)

(c)可视化彩色增强结果(L=3) (d)可视化彩色增强结果(L=7)

图 3 运动可视化处理结果

4 结语

本文提出了一种基于 Fermi 架构 GPU 的组织运动可视化并行实现方法，实验结果显示了用本文所提出的并行实现方法得到的处理结果和通用 CPU 处理的结果基本一致，而在时间性能方面达到了实时系统的处理要求，得到大约 230 倍的加速效果. 在本文的并行算法中主要包括了利用二叉树归约进行并行求和计算绝对值误差和寻找最匹配块，利用查找表避免分支处理情况，结合 Fermi 架构 GPU 性能特性设计合理的线程结构和存储器使用策略等. 本文提出的方法为超声心动检测可视化实时处理提供了解决方案，使这一算法在临床实时系统中可以得到较好的应用. 下一步的研究工作主要是进行体内测试和提高算法运动检测精度等.

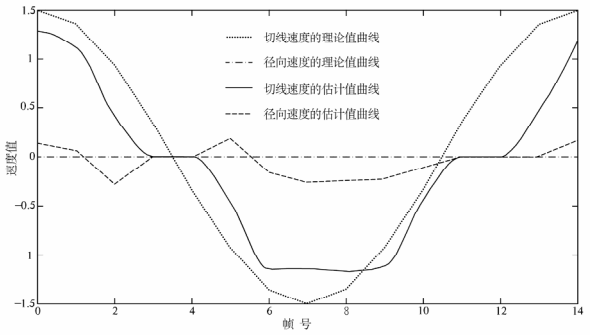


图 4　运动径向和切线方向速度的 GPU 处理结果

参考文献

1 李治安.临床超声影像学.北京:人民卫生出版社,2003:45–60.

2 Liu DC, Hou LL, Liu PS. Motion visualization of ultrasound imaging: Advances in Visual Computing. Springer, 2005: 653–658.

3 Cabral B, Leedom C. Imaging vector fields using line integral convolution. Computer Graphics Proc., 1993:263–270.

4 Shen HK, Kao DL. A line integral convolution algorithm for visualizing unsteady flows. Proc. of IEEE Visualization, 1997:317–322.

5 Cao T, Tan CW, Liu DC. Adaptive curve region based motion estimation and motion visualization of cardiac ultrasound imaging. 2009 IEEE Inter. Conference on Bioinformatics and Biomedical Engineering, 2009.

6 Stalling D, Hege HC. Fast and resolution- independent line integral convolution. Proc. of SIGGRAPH'95. 1995: 249–256.

7 NVIDIA Corporation. CUDA Programming Guide4.0. [2011-05-06]. http:// www.nvdia.com.

8 夏春兰,石丹,刘东权.基于 CUDA 的超声 B 模式成像.计算机应用研究,2011,28(6):2011–2015.

9 范正娟,石丹,刘东权.基于 CUDA 的超声彩色血流成像.计算机应用,2011,31(3):856–859.

10 Jan Snyman A. Practical Mathematical Optimization: An Introduction to Basic Optimization Theory and Classical and New Gradient-Based Algorithms. Berlin: Springer, 2005.