基于 GPU 的快速 Level Set 图像分割

吴仲乐 王遵亮 罗立民
(东南大学生物科学与医学工程系影像科学与技术实验室,南京 210096)

摘 要 水平集(level set)图像分割方法是图像分割中的一个重要方法,但是该算法的计算量大,往往不能达到实时处理的要求。给出了利用新一代的可编程图形处理器(GPU)实现 level set 的加速算法。首先介绍了如何在 GPU 上利用片元渲染程序进行网格化的线性运算和有限差分 PDE 计算,把 level set 方法的离散化算子映射到 GPU 上。由于以数据流处理方式的 GPU 的存储访问快,具有并行运算能力,同时 level set 算法演化的显示不再需要把数据从 CPU 传到 GPU,因此较大地提高了算法速度与交互显示。文中实现并测试了一个与初始化状态独立的二维 level set 的算子用于图像分割,并对其运算结果和性能进行了比较,结果表明该方法具有更快的速度。

关键词 水平集方法 图像分割 可编程渲染器 图形处理器
中图法分类号: TP391 **文献标识码**: A **文章编号**: 1006-8961(2004)06-0679-05

Fast Level Set Image Segmentation on Graphics Processing Unit

WU Zhong-le, WANG Zun-liang, LUO Li-min
(*Laboratory of Image Science and Technology, Dept. of Biomedical Engineering, Southeast University, Nanjing* 210096)

Abstract Level set methods are powerful tool to segment images, but these algorithms have a large computational burden thus are not suitable for real time processing requirement. In this paper, an new accelerating algorithm of level set method is presented which is implemented on the new generation of graphics processing unit (GPU) instead of on CPU. It first introduced how to implement grid computation for algebraic linear operation and finite difference solution of PDE on GPU by fragment program, and then map the level set solver to GPU. Since GPU is a parallel vector processor for streamed data with big bandwidth for data access, and the result data don't need to be transferred from CPU to GPU for data rendering, so the accelerating algorithm is suitable for real time processing and rendering. In this paper a 2D level set solver for image segmentation was tested with comparison of the performance result between fast marching method and the GPU accelerated method. It shows this method can achieve 60 percent quicker.

Keywords level set method, image segmentation, programmable shader, GPU

1 引 言

图像分割是医学图像处理中的重要研究内容之一。从医学图像中准确地提取目标物体是三维重建的基础,也是医学图像处理系统在临床上得到实用的基础,同时又是一个经典难题。由于问题的重要性和困难性,从 20 世纪 70 年代起,图像分割问题就吸引了很多研究人员为之付出巨大的努力。曲线演化(curve evolution)是处理图像分割问题[1]的一种有效方法。它通过对闭合的曲线遵循特定能量函数定义下的形变过程,寻求能量函数的最小值,从而使得闭合曲线逐渐逼近图像中目标边缘。由 Sethian 等人[1~3]提出的水平集(level set)方法是将二维(三维)的曲线(曲面)演化问题转化为三维(四维)空间中 level set 函数曲面演化的 PDE 隐含方式来求解,这种方法解决了以往算法不能解决的拓扑结构变化问题,在图像处理中得到了广泛的应用。但 level set 方法有一个缺点,即计算量太大,因为它将二维的问题扩展到三维,三维的问题扩展到四维,维数的扩展

收稿日期:2003-08-08;改回日期:2004-03-08

增加了计算复杂度，平面曲线演化算法的计算复杂度为$O(N^2)$，三维曲面演化的计算复杂度为$O(N^3)$，其中N是将平面(或空间)均匀离散成网格点后，水平方向的网格点数。因此，人们提出了如窄带算法和fast marching算法[5]等快速算法。但另一种可能的加速level set计算方法是利用新一代的图形显示核心处理器(GPU)进行。与CPU不同，GPU是一个并行的向量处理器，以一个程序多次数据模式(SIMD)工作，而它对流式数据的处理能力强于CPU。

新一代图形处理器的可编程渲染器改变了以往GPU固定管道线的绘制方式(如图1所示)，使得应用能够定制独特的渲染程序得到特殊的图像效果。同时，它的可编程性已经使得GPU转变为具有一般计算能力的流式向量处理器，一些物理现象的模拟已经在GPU上进行计算和快速显示，而且，GPU还能够进行除图形绘制以外的科学和通用计算。GPU片元渲染器或像素渲染器进行图像像素计算，也可当作是在求解空间中离散网格点上的计算，即grid computation。在GPU上进行运算的优点是对于栅格数据能够快速地存储访问和并行处理，而且数据处理的中间结果和最后结果的显示不再需要数据从CPU传送到GPU，尤其对三维体数据算法，三维level set的水平集显示可以利用等值面的非多边形直接显示，而无需在中间结果中重建网格面。

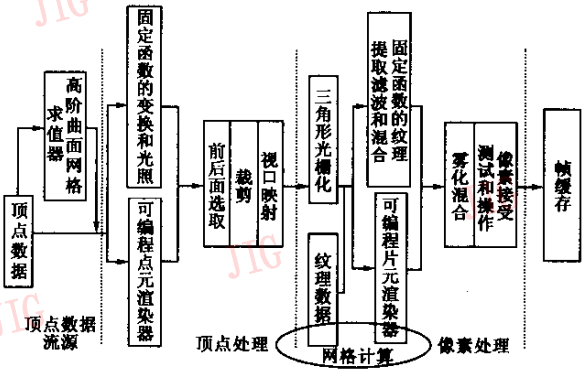


图1 GPU图形管道模型

(纹理映射和片元渲染器的结合构成了基于网格的计算模型)

2 level set模型

Osher-Sethian[1]提出的水平集方法所要解决的是遵循如下偏微分方程形变的二维闭合曲线$C(t)$演化的问题，

$$\partial C(t)/\partial t = FN, C(p,0) = C_0 \tag{1}$$

式中，F为根据特定应用所定义的曲线在其法线方向的速度；N为曲线的单位法矢量。水平集方法是将上述二维平面的闭合曲线视为三维空间连续函数曲面$\varphi(x,y,t)=c$(水平集函数)的特定水平集，通常取0水平集$c=0$。事实上level set方法是一种跟踪曲线或曲面演化的数学表达方法。它不直接对曲线或曲面进行形变，而是通过把曲线或曲面嵌入到更高一维的标量函数φ(称为水平集函数)。于是水平集函数由PDE方程控制演化，而在任意时刻，演化中的曲线都可以从水平集函数中提取。水平集的最大优势是可以表示任意复杂的形状和支持拓扑形状的变化，比如曲线的合并和分裂等。

level set方法适用于图像分割，一般是利用图像的特征或信息比如图像中值强度、梯度、边界等来控制支配PDE方程的演化。典型的应用是用户初始化一个曲线，然后利用PDE方程进行演化，直到曲线到达解剖结构的边界。

二维level set函数$\varphi(x,y,t)$随时间演化的公式遵循Hamilton-Jacobi偏微分方程

$$\begin{cases}\partial\varphi(x,y,t)/\partial t + f\|\nabla\varphi(x,y,t)\| = 0\\ \varphi(x,t,0) = \varphi_0(x,y)\end{cases} \tag{3}$$

其中，$\varphi_0(x,y)$是初始水平集函数，f是从原始图像得到的水平集扩散的外部力。采用如下的有限差分法：设h是离散水平集的二维网格步长，

$$(x_i,y_j) = (ih,jh), 1\leqslant i\leqslant M, 1\leqslant j\leqslant N \tag{4}$$

为格点坐标，则

$$\varphi_{i,j}^{(k)} = \varphi(x_i,y_j,k\Delta t) \tag{5}$$

是关于h的对$\varphi(x,y,k\Delta t)$的网格近似。其中，$k\geqslant 0$，$\varphi^{(0)}=\varphi_0$，并有x,y方向的前后差分为

$$\begin{cases}D_{i,j}^{-x} = \varphi_{i,j}^{(k)} - \varphi_{i-1,j}^{(k)}\\ D_{i,j}^{x} = \varphi_{i+1,j}^{(k)} - \varphi_{i,j}^{(k)}\\ D_{i,j}^{-y} = \varphi_{i,j}^{(k)} - \varphi_{i,j-1}^{(k)}\\ D_{i,j}^{y} = \varphi_{i,j+1}^{(k)} - \varphi_{i,j}^{(k)}\end{cases} \tag{6}$$

设$u:=\nabla\varphi, H(u):=f\|u\|$，为简化计算采用通量函数$g$来近似$H$，则有迭代公式

$$\varphi_{i,j}^{(k+1)} = \varphi_{i,j}^{(k)} - \Delta t\cdot g(D_{i,j}^{-},D_{i,j}^{+}) \tag{7}$$

其中，

$$\begin{aligned}D_{i,j}^{-} &= (D_{i,j}^{-x},D_{i,j}^{-y})\\ D_{i,j}^{+} &= (D_{i,j}^{x},D_{i,j}^{y})\end{aligned} \tag{8}$$

根据Enguist-Osher的“通量守恒”差分方法[4]，可选择一个简单的g的数值解表达式如下

$$g(U,V) := H(0)+\int_0^U H'(s)^+ \,\mathrm{d}s+\int_0^V H'(s)^- \,\mathrm{d}s$$
$$= f^+ \sqrt{\|U^+\|^2+\|V^-\|^2}+f^- \sqrt{\|U^-\|^2+\|V^+\|^2} \tag{9}$$

其中,

$$X^+ := \max(X,0)$$
$$X^- := \min(X,0) \tag{10}$$

这样,最后的离散公式为

$$\varphi_{i,j}^{(k+1)} = \varphi_{i,j}^{(k)} - \Delta t \cdot (f^+ \sqrt{\|U^+\|^2+\|V^-\|^2}+$$
$$f^- \sqrt{\|U^-\|^2+\|V^+\|^2}) \tag{11}$$

其中,为了满足稳定性条件,必须使得

$$\Delta t/h < 1/\max(f)$$

3　GPU 计算实现

文献[3]、[6]提出了利用图形硬件实现扩散方程算子和 level set 的图像分割方法的实现。但其利用的是图形硬件的前后缓存的混合机制,只能进行非常简单的运算,程序设计只能满足最简化的 level set 模型。新一代的图形处理器 GPU 的可编程能力使得能够在其上实现一些 PDE 方程的解。基于 GPU 的计算与 CPU 的计算不同,GPU 处理的数据以纹理(数据块)的形式存储,并以数据流的方式在各个网格点上进行同一种运算。GPU 对显存的访问具有更高的带宽,渲染程序有 2 到 4 个并行管道运算,从而其在处理速度上较 CPU 有优势。

3.1　GPU 上的网格计算

为了映射 level set 演化算子到片元渲染器上,首先介绍利用片元渲染器进行计算。

比如 2 幅图像的相加程序,片元渲染器只需要一个相加的指令,如图 2 所示。

片元渲染器程序是一小段的程序。它的输入主要是纹理数据块和用于提取纹理数据光栅化的三角面片。受图形硬件的限制,一个渲染程序只能同时处理 4～8个纹理数据,而输出是一个二维帧缓存,缓存上每个网格点上的值就是片元渲染器程序对相应网格位置上输入值的计算结果。通常复杂计算要经过多遍渲染处理,渲染程序的输出结果往往又是中间数据并且要作为下一个渲染程序的纹理输入。这种中间数据重用的最好方式就是利用 RenderToTarget 扩展,RenderToTarget 的像素缓存在绘制后又可作为渲染程序输入的纹理缓存,不需要拷贝中间数据。因为这些

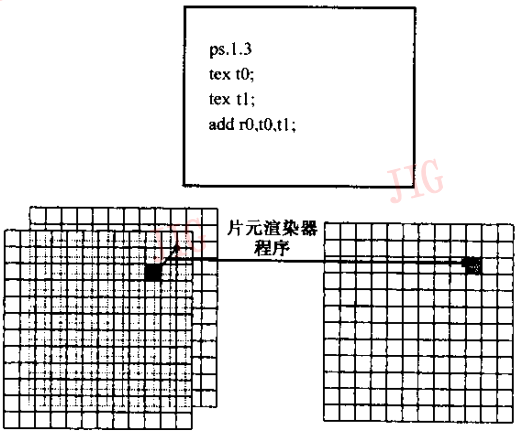


图 2　片元渲染器的 2 幅图像加法指令和示意图

数据都储存在显存空间,他们的显示不再需要把数据传送到显存,因而提高了效率,这在体数据处理时更为明显。渲染程序计算指令支持加减法、乘法等一些简单的运算指令和逻辑比较运算指令。但是 ps. 1. 3 (Direct3D 的 Pixelshader1. 3 版)中没有开平方根的指令和差分指令。对于开平方根指令,一种方法是采用如下近似公式,用一小段片元渲染程序近似计算得到。

$$g(U,V) := f^+ \cdot N(U^+,V^-)+$$
$$f^- \cdot N(U^-,V^+) \tag{12}$$
$$N(A,B) := c(|A|_1+|B|_1)+$$
$$(1-c)\max(|A|_\infty,|B|_\infty) \tag{13}$$

另一种方法是利用纹理依赖提取(dependent texture fetch),即把开平方根表以纹理存储作为一个查找表的方式在渲染程序中使用依赖纹理提取得到,这样可以节约运算。差分运算事实上就是 2 幅具有一个相对偏移的相同图像的差。因此,差分的渲染程序仍然只是简单的相减,输入是同样的纹理图像,而相对偏移量可以使用纹理矩阵使另一幅图像得到相对平移。执行一个渲染程序即为一遍绘制。而一个渲染程序因为硬件上有纹理数据输入个数 N_t,程序长度 N_l,寄存器个数 N_r 的限制,level set 演化算子需要进行多遍绘制 N_p。但为了最大限度地发挥图形硬件的性能,加快算法效率,要使得算法的 N_p 最小。因此要尽量使得一遍绘制中使用最多的输入,执行最多的指令和减少中间数据。

3.2　GPU 上计算的数据格式

GPU 是向量处理器,输入和输出的数据格式为具有 4 个通道的 8 位 RGBA 颜色模式,数据值为[0,255]区间。而在渲染程序内部具有更高的浮点数据精度表示,且把输入数据值映射为[0,1]的区间后进行

运算，而输出的数据也是把[0，1]映射为[0，255]的 8 位数据。因此算法必须考虑数据的截断误差问题。对于 8 位图像数据，初始水平集函数 $\varphi_0(x_i,y_i)$ 要映射到[0，255]区间，把水平集值 0 映射为 127，初始轮廓内部为[0，127]，外部为[127，255]，极端情形即可取轮廓内部为 0，外部为 255。水平集值存储在一个二维纹理 RGBA 所有通道，而其横向和纵向的差分分别存储在一个纹理的 RGB 通道和 A 通道中。为了使得 $\varphi(x_i,y_j)$ 的差分值仍在[0，255]区间内表示，实际计算的是 $\varphi(x_i,y_j)/2$ 的差分，其范围是[−127，127]，并且把该差分值偏移 127 使得满足[0，255]区间表示的要求。在最后更新 φ 值时再把相应的更新值乘以 2 即可。在对 8 位图像进行分割时，上述 GPU 数据表示方法已经满足了数据精度的要求。

3.3 level set 算法步骤

开发了一个基于 pixel shader 的运算库，实现了各种运算。它包括如下的片元渲染程序集：图像求和、差、差分，2～4 幅图像的最值、图像灰度中值截断、开平方查找表等。基于 GPU 硬件加速的 level set 算法步骤如下：

(1) 加载原始图像和初始水平集函数 $\varphi_0(x_i,y_j)$；

(2) 计算并加载扩散力 $f^+(x_i,y_j)$，$f^-(x_i,y_j)$；

(3) 设定 Δt；

(4) 计算 $D^-\varphi$ 和 $D^+\varphi$；

(5) 计算 $\|(D^-\varphi)^+\|^2$，$\|(D^-\varphi)^-\|^2$，$\|(D^+\varphi)^-\|^2$，$\|(D^+\varphi)^+\|^2$；

(6) 计算 $f^+\sqrt{\|U^+\|^2+\|V^-\|^2}$

$f^-\sqrt{\|U^-\|^2+\|V^+\|^2}$；

(7) 计算 $\varphi^{(k)}-\Delta t\cdot H(u)$；

(8) 更新为 $\varphi^{(k+1)}$。

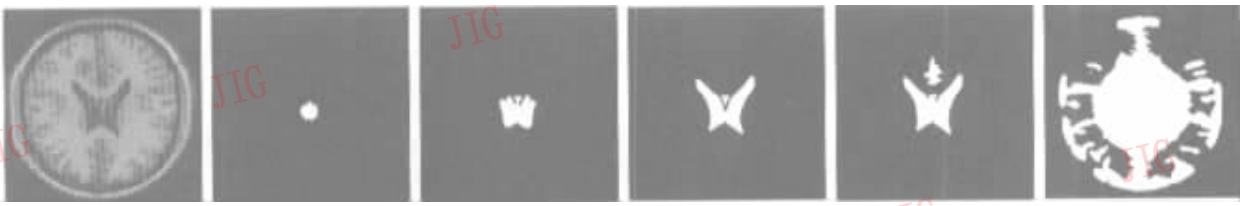
第 1～3 步是水平集的初始化过程，第 4～8 步是 φ 进行一次迭代的步骤，上述水平集方程每进行一次迭代时，需要 GPU 进行 10 遍绘制。其中第 4 步需要 2 遍，第 5 步需要 4 遍，第 6 步需要 2 遍，第 7 步和第 8 步共需要 2 遍。

总的说来，基于 GPU 的计算在程序设计上编程工作比较大，而且需要为特定的图形硬件制定相应的渲染程序和多遍绘制，尽量减少多遍绘制的遍数，一遍绘制中必须包含尽可能多的纹理数据和计算来减少中间结果，以最大发挥图形硬件的性能。

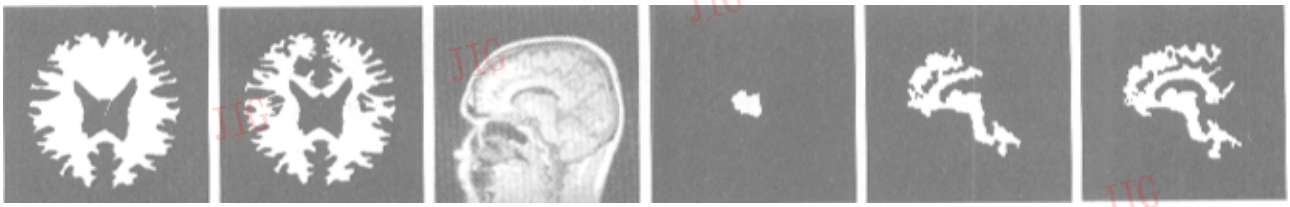
4 试验结果

实验中所用的数据是一幅 256×256 的脑部水平面 MRI 图像和矢状面 MRI 图像，扩散力是基于图像灰度的模型。实验所用的显卡分别是 Geforce4 Ti42000 (PixelShader 1.3) 和 GeforceFX5900 (PixelShader 2.0+)。显存空间均是 128M。图 3 中(b)～(e)给出了分割侧脑室的过程，其初始轮廓是一直径为 8 的中心圆轮廓；(f)～(h)给出了分割白质的过程，其初始轮廓是一直径为 100 的中心圆轮廓。

将该算法与 fast marching 方法在同一台计算机上的计算时间进行了比较。fast marching 用的测试程序是基于 ITK[7] 工具包编制的一个程序。



(a) 原始图像 (b) 第 10 次迭代 (c) 第 50 次迭代 (d) 第 150 次迭代 (e) 第 300 次迭代 (f) 第 800 次迭代



(g) 第 2 000 次迭代 (h) 第 3 000 次迭代 (i) 矢状面图像 (j) 第 15 次迭代 (k) 第 220 次迭代 (l) 第 400 次迭代

图 3 实验结果

表 1　算法计算时间的比较

迭代次数和(交互显示的间隔次数)	fast marching 计算时间(s)	GPU 计算时间(s)	GPU 相对 fast marching 的计算时间百分比(%)
150(1)	6.3	3.1	49.2
300(1)	10.8	6.4	59.3
800(10)	29.5	16.2	55.0
2 000(50)	71.4	43.4	60.7
3 000(10)	116.9	60.3	51.5
3 000(100)	104.7	59.2	56.5

从表 1 可以看出,GPU 每次迭代的平均时间约为 2ms,GPU 相对 fast marching 的计算时间提高了约 40%～60%。

5　结　论

基于 GPU 的计算实现是 level set 的一种新的加速算法。利用图形硬件的加速特性和可编程渲染器,把 level set 算子映射到 GPU 上计算。对于 8 位灰度图像,该算法得到的结果与经典的 level set 实现方法的结果一致。该算法与 fast marching 的快速算法相比较在算法效率上有了较大的提高。而随着图形处理器(GPU)的飞速发展,高级的图形处理器能够支持 32 位精度的颜色通道和更多的显存空间以及更复杂的渲染程序,这为实现更多的复杂计算带来了可能。该算法还可以扩展到体数据的 level set 分割,使用硬件三维纹理,利用渲染程序对体数据断层逐个进行处理,实现三维的 level set 图像分割。该算法在三维 level set 的交互显示性能上会有较大的提高。

参 考 文 献

1 Osher S, Sethian J A. Fronts propagating with curvature dependent speed: algorithms based on Hamilton-Jacobi formulation[J]. Journal of Computer Physics, 1988, **79**(1): 12～49.

2 Sethian J A. Numerical algorithms for propagating interfaces: Hamilton-Jacobi equations and conservation laws[J]. Journal of Differential Geometry, 1990, **31**(1): 131～161.

3 Rumpf M, Strzodka R. Nonlinear diffusion in graphics hardware [A]. In: Proceedings EG/IEEE TCVG Symposium on Visualization, 2001[C], Ascona Switzerland, 2001: 75～84.

4 Engquist B, Osher S. Stable and entropysatisfying approximations for transonic flow calculations[J], Mathematics of Computation., 1980, **34**(93): 45～57.

5 Sethian J A. Level set methods and fast marching methods[M]. Cambridge University Press, Cambridge, UK, 1999.

6 Strzodka R, Rumpf M. Level set segmentation in graphics hardware [A]. In: Proceedings International Conference on Image Processing 2001, [C], Thessaloniki, Greece, 2001: 1103～1106.

7 ITK Software Guide[EB/OL] http://www.itk.org/HTML/Documentation.htm.

吴仲乐　1977 年生,博士研究生,2000 年毕业于东南大学生物医学工程系。主要研究方向为计算机图形学与可视化、医学图像处理。

王遵亮　1975 年生,博士研究生,1998 年毕业于东南大学生物医学工程系,2002 年 2 月于东南大学获工学硕士学位。主要研究方向为医学图像处理、计算机视觉与计算智能。

罗立民　1956 年生,博士生导师,教授,1986 年于法国雷恩大学获信息处理专业博士学位。长期从事医学图像处理和生物医学工程领域的研究工作。