基于GPU的地震剖面图形快速绘制算法

邓博文[1] 刘春松[1] 吴凡贤[1] 姚兴苗[2]

(电子科技大学通信与信息工程学院 成都 611731)[1] (电子科技大学资源与环境学院 成都 611731)[2]

摘 要 地震剖面图的绘制是二维地震数据可视化的基础。目前基于通用绘制引擎的地震剖面图绘制是在CPU上实现的,随着地震数据规模越来越大,传统绘制方法的绘制效率已经不能达到交互效率的要求,因此提出了一种地震剖面图快速绘制算法。该算法将地震数据的绘制和GPGPU技术相结合,利用GPU强大的并行计算能力实现图形光栅化处理。实验表明,在保证绘制效果的前提下,该方法极大地提高了绘制效率。

关键词 地震数据,可视化,图形绘制,光栅化,GPU

中图法分类号 TP391 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2016.4.060

Fast Graphical Rendering of Seismic Profile Algorithm Based on GPU

DENG Bo-wen[1] LIU Chun-song[1] WU Fan-xian[1] YAO Xing-miao[2]

(School of Communication and Information Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China)[1]

(School of Resources and Environment, University of Electronic Science and Technology of China, Chengdu 611731, China)[2]

Abstract The drawing of seismic profile is the foundation of the visualization of 2D seismic data. Current method based on common rendering engine is complied on CPU, but with the increasing scale of seismic data, the conventional visualization method has been unable to meet the requirement of efficiency of interaction. This paper proposed a method to accelerate rendering seismic data graphics, which integrates graphical rendering and GPGPU technology. The graphical rasterization is implemented based on the powerful parallel computing capability of GPU. Experimental results indicate that, on the premise of ensuring imaging quality, large seismic data can be perfectly imaged in real time on a general PC platform.

Keywords Seismic data, Visualization, Graphical rendering, Rasterization, GPU

数据可视化技术已经成为科学研究中的一种流行的技术,这种技术能够提高工程的效率和准确性。数据可视化利用计算机图形学等技术对数据进行处理,将其转换为可显示在屏幕上的图形或者图像,通过交互处理来挖掘数据相互间的规律。数据可视化技术已经被广泛用于多个领域[1-2],特别是在地质勘探工程上的应用[3-5],使得地质勘探减少了风险和成本,同时也提高了对地震数据处理的效率和质量。

地震数据可视化在国内外的地质解释系统中都有实现,是地质解释软件系统中基础而又重要的模块。地震数据可视化分为二维和三维地震数据的可视化。二维地震数据图形绘制是二维地震数据可视化的基础,也是三维地震数据分剖面可视化的基础,因此二维地震数据图形绘制是地质解释软件的基础功能。随着数据可视化的不断完善,人们针对二维地震数据的图形绘制提出了比较成熟的绘制方法,肖汉在地震数据可视化技术研究中提出了基于OpenGL绘制引擎的二维地震剖面B+W图的绘制方法[6],黄丹在基于QT的地震数据可视化的研究及应用中提出了基于Qt绘制引擎的二维地震剖面的绘制方法[7]。

随着测量技术的发展,地震数据规模越来越大,目前的地震数据图形绘制方法中地震数据处理的时间比较长,图形光栅化处理[8]效率低,导致基于Qt和OpenGL等通用绘制引擎的地震数据图形绘制难以支撑应用的需求。

随着高性能计算、图像处理和计算几何等技术的不断发展,尤其随着GPU硬件的发展和GPU可编程性的不断提高,利用GPU完成通用计算的研究渐渐活跃起来。为了提高地震数据可视化的效率,GPGPU技术也逐渐被应用到地质勘探软件中。徐赛花等提出了基于CUDA的三维数据并行可视化[9],基于CUDA编程技术利用GPU的多核并行实现了三维体绘制。K. Xie等提出了GPU和CPU协同并行实现海量地震数据可视化[10]。Daniel Patel等提出了基于GPU的层位提取的层位生长算法[11]。目前在地质勘探领域GPGPU技术的应用都利用了地震数据的独立性,借鉴目前GPGPU技术在地质勘探软件中的应用,针对二维地震剖面绘制中的地震数据之间不具有相干性,通过分析传统的绘制方法

到稿日期:2015-04-08 返修日期:2015-08-19 本文受国家自然科学基金(41104067)资助。

邓博文(1990-),男,硕士,主要研究方向为计算机图形学、计算机网络,E-mail:dengbowen1st@163.com;刘春松(1990-),男,硕士,主要研究方向为计算机图形学;吴凡贤(1986-),男,硕士,主要研究方向为信息可视化;姚兴苗(1976-),男,博士,副教授,主要研究方向为计算机图形学、复杂多维信息处理与显示。

可以看出对每个地震数据点的处理完全一致，满足 GPGPU 设计准则，因此针对目前图形显示光栅化处理效率低的问题，本文提出了基于 GPU 的地震数据图形绘制，利用 GPU 的并行计算能力实现图形光栅化处理，提高光栅化的效率。

本文首先介绍了基于通用绘制引擎实现二维地震数据可视化的方法，即 B+W 图绘制方法，分析了基于通用绘制引擎的绘制特点和绘制效率；其次，通过结合地震数据特点和图形光栅化的原理，并结合 GPGPU 技术提出了基于 GPU 的并行绘制算法；最后，从绘制时间和绘制效率方面对基于通用绘制引擎的绘制方法和基于 GPU 的并行绘制方法进行了对比分析。

1 基于通用绘制引擎的地震数据图形绘制方法

地震剖面图(B+W 图)采用曲线的方式来表示检波点的振动情况，即检波点的波形图。曲线由很多的小线段构成，线段的顶点坐标与数据值是一一对应的，把它们首尾相连就可以实现曲线的绘制。另外，相邻地震道之间由于波形相似，反射时间比较接近，波峰比较靠近，在地震剖面上相互叠套成串，形成同相轴，为了更好地表现同相轴，需要将波形图的地震数据正值区域或者负值区域填充，最终构成 B+W 图。算法实现的流程如图 1 所示。

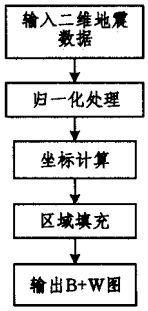

图 1 基于通用绘制引擎的地震数据图形绘制流程

1.1 地震数据的归一化处理

地震数据值的大小代表振幅的强弱，正负代表示振动的方向。这些特征不会随着图形的变换而改变，所以波形变化前后是相似的。利用这种相似性，可以将波形转换到某个范围内，并保持整个波形的特征。于是将地震数据变换到[−1,1]范围内，这样保留了数据值的相对大小，而且值的正负表示了振动方向。

地震数据归一化处理的具体实现方法为：找出地震数据中绝对值最大的值，并以该绝对值作为标准值。所有的数据值都通过这个标准值进行映射。映射公式如式(1)所示。

$$mappedValue=\frac{sampleValue}{vstad} \tag{1}$$

其中，$vstad$ 为映射的标准值，$sampleValue$ 和 $mappedValue$ 分别是原始数据值和映射后的数据值。

1.2 地震波形曲线绘制

地震波形图是由很多的小线段构成的，线段的顶点坐标与地震数据是一一对应的。因此，地震波形绘制最关键的就是计算地震数据对应的顶点坐标。

地震数据属于规格化数据，绘制点的坐标计算与横纵向的间距有关。假设每道所占的宽度是 $traceDistance$，每个数据点所占的高度是 $sampleHeight$，横向是振动方向，因此 X 坐标是由道间距与归一化的数据共同决定的，计算公式如式(2)所示。式(2)中的道间距决定了图像的横向缩放比例，而归一化后的数据保持着波形的基本特征。横向变换如图 2 所示。

$$X=mappedValue\times\frac{traceDistrance}{2} \tag{2}$$

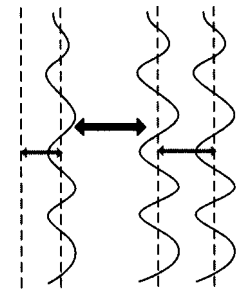

图 2 道的横向变换

Y 坐标的计算与 X 坐标的不同，它仅与数据点的间距和地震数据点在该道中的位置有关。假如地震数据点在该道的起始索引号为 siy，结束索引为 eiy，当前的绘制点索引为 j，则该点的 Y 坐标计算公式如式(3)所示。图 3 表示了纵向数据点间距对图形的影响。

$$Y=(j-siy)\times sampleHeight \tag{3}$$

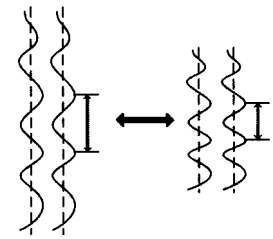

图 3 道的纵向变换

1.3 区域填充

地震波形区域填充时需要对振幅的正值区域和负值区域做出准确判断，结合道的平衡位置，将正值或者负值与平衡位置所围成的区域进行填充。由于地震波形和平衡位置围成的多边形不一定是凸多边形，因此首先需要对多边形进行凸多边形化，再对凸多边形进行填充。传统的绘制方法是基于通用绘制引擎实现的，因此在对正值区域或者负值区域填充时是直接调用绘制引擎的 API 函数填充的。

基于通用绘制引擎的 B+W 图绘制方法实现主要是基于 Qt 或者 OpenGL 等通用绘图引擎。基于 Qt 绘图引擎实现绘制时，整个绘制和显示都是调用 Qt 绘图引擎的 API 实现的，基本上所有事情都在 CPU 上完成，因此交互的效率会受到地震数据的绘制效率的直接影响。通过分析通用绘制方法可知，基于 Qt 绘制引擎实现的时间复杂度为 $O(M*N*N)$，基于 OpenGL 绘制引擎实现的时间复杂度为 $O(M*N)$。因此，随着地震数据的规模越来越大，传统的绘制算法的绘制时间呈线性增长，严重影响到整个图形显示系统的效率，导致交互的效率不能满足现在的需求。因此，在常规的地震数据图形绘制方法的基础上提出了基于 GPU 的地震数据图形绘制方法。

2 基于 GPU 的地震数据图形快速绘制方法

2.1 地震数据特点

地震数据按道组织，一个地震剖面对应的地震数据可以看成是二维的数据点阵。地震剖面图(B+W 图)是由按道的地震波形图和地震波形的正值区域或者负值区域填充表示的。由常规的绘制方法可以看出地震剖面图绘制的过程中地震数据每道之间没有任何关系，同时每道的地震数据点的坐标值和同一道上的其他地震数据点的坐标值没有任何的关系，因此满足 GPU 要求的数据之间没有相干性，即数据的计算不依赖于另外一些数据的计算结果。

2.2 图形光栅化

图形光栅化作为沟通几何-像素的桥梁，是现代渲染系统的核心部分，光栅化的效率直接影响着图形显示系统的效率。图形光栅化处理主要包括线的光栅化处理和填充光栅化处理。Qt 的线光栅化处理采用的是中点画线算法，整个处理的过程都是在 CPU 上实现的。光栅化过程中的每个像素点的颜色值的计算和其他的像素点的颜色值没有任何关系，满足 GPU 要求的数据之间没有相干性。因此，光栅化处理满足 GPGPU 设计准则，可以利用 GPU 强大的并行计算能力并行计算，提高光栅化的效率。

2.3 基于 GPU 的地震数据图形快速绘制方法

结合地震数据的特点和图像显示光栅化处理的原理可知，地震剖面图的绘制可以利用 GPU 的并行计算能力实现地震数据绘制的图形光栅化。因此本文提出了一种基于 CUDA 并行实现 B+W 图的绘制算法。

2.3.1 地震波形曲线绘制

由基于通用绘制引擎的地震剖面绘制算法的分析可知，地震波形曲线根据地震数据值、X 方向的显示间距和 Y 方向的间距分布，因此每道的地震波形曲线图对应的像素点集固定在特定的区域内。因为波形曲线是由很多的小线段构成的，根据图形显示光栅化的原理可知任何一条线段最终都对应一系列连续的像素点集。因此，相对传统的绘制方法中首先计算地震数据对应的坐标位置，然后再首尾相连，最后由绘制引擎实现线的光栅化处理，B+W 图并行算法直接实现线段的光栅化处理，采用纵向插值的方法[12]计算出每道地震波形曲线图对应的像素点的个数和位置。

以一道地震数据的波形曲线图绘制为例，假设 X 方向的间距为 *traceDistance*，Y 方向的间距为 *sampleHeight*，输入的地震数据为 *data*[*sampleCount*]，输入地震数据的绝对值最大值为 max*Data*，纵向插值的起始点的索引值为 *sIndex*，终止点的索引值为 *eIndex*，则采用纵向插值计算像素点对应的地震数据值的公式如式(4)所示。每个像素点在特定区域的位置计算公式如式(5)所示。一道地震数据的波形曲线绘制示意图如图 4 所示。

$$\alpha=(index-sIndex)/(eIndex-sIndex)$$

$$interpData[index]=data[sIndex]+\alpha*(data[eIndex]-data[sIndex]) \quad (4)$$

其中，*interpData*[*index*]是待插值点插值后的值，*data*[*sIndex*]和 *data*[*eIndex*]分别是线性插值起始值和终止值。

$$location[index]=\frac{pixelData[index]}{\max Data}*\frac{traceDistance}{2} \quad (5)$$

其中，*pixelData*[*index*]是曲线经过的所有像素点对应的地震数据值，*traceDistance* 是 X 方向的间距，*location*[*index*]是像素点偏离中心位置的像素宽度，该值的正负表示该像素点位于中心位置的左侧或右侧。

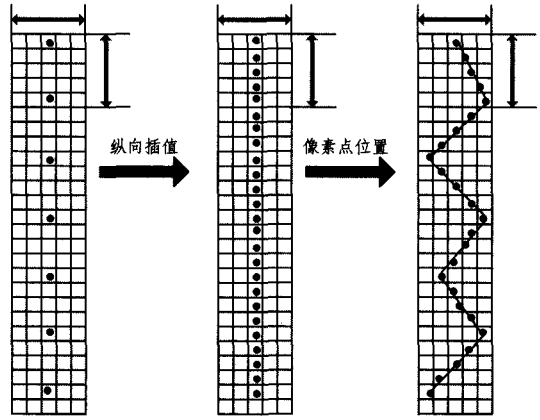

图 4 一道地震数据波形曲线绘制示意图

2.3.2 区域填充

区域填充是在波形曲线的基础上，将地震数据的正值区域或者负值区域填充成填充色，因此区域填充光栅化处理就是将每道所占的固定像素区域的满足要求的像素点设置成填充色。

以一道地震波形曲线的正值区域填充为例，具体实现方法为：将每道地震数据纵向插值生成的像素点和原始地震数据对应的像素点标记为边界点，记为 BoundaryPoint；判断在固定区域内的其他非边界点的像素点(JudgePoint)与边界点的相对位置关系，满足填充条件的像素点设置为填充色，其他像素点设置为背景色。填充判断伪代码如下所示，填充示意图如图 5 所示。

填充判断伪代码：

```
if(FillDirection==Right)
  if(BoundaryLoction >=0)
    if(JugeLocation >=StartLoction && JugeLocation <=Boun-
    daryLoction)
      Flag=1;fill filling color on the pix
    else
      Flag=0;fill background color on the pix
  else
    Flag=0;fill background color on the pix
else if (FillDirection==Left)
    if(BoundaryLoction <=0)
      if(JugeLocation<=StartLoction && JugeLocation > Boun-
      daryLoction)
        Flag=1;fill filling color on the pix
      else
        Flag=0;fill background color on the pix
    else
      Flag=0;fill background color on the pix
```

其中，FillDirection 表示填充的方向，Right 表示填充正值区域，Left 表示填充负值区域。BoundaryLoction 表示纵向插值计算的边界像素点，StartLoction 表示平衡位置，即计算道中地震数据值为 0 对应的像素点位置。JugeLocation 表示需要判断的像素点的位置，Flag 表示该像素点设置的颜色标识，

Flag 为 1 表示设置为填充色，Flag 为 0 表示设置为背景色。

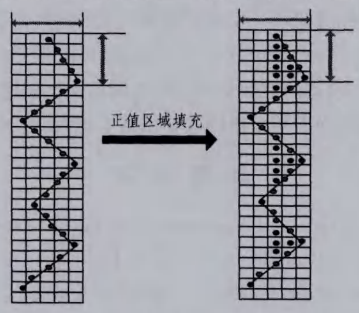

图 5　地震波形曲线图填充示意图

2.3.3　基于 CUDA 的地震数据图形绘制的实现

根据 B+W 图的并行绘制算法的思想，假如输入的地震数据记为 *Date*[*TraceCount*][*SampleCount*]，X 方向的间距记为 *TraceDistance*，Y 方向的间距记为 *SampleHeight*，则输出为(*TraceCount* * *TraceDistance*) * ((*SampleCount*−1) * *SampleHeight*)像素大小的 B+W 图。实现的伪代码如下：

```
loop i from 0 to (SampleCount−1) * SampleHeight in steps of 1
  loop j from 0 to TraceCount−1 in steps of 1
    scaleddata[i,j]=scale(Date[i,j])
  end of loop
end of loop
loop i from 0 to (SampleCount−1) * SampleHeight in steps of 1
loop j from 0 to TraceCount * TraceDistance * 4 in steps of 1
  flag=map(scaleddata[i,j])
  if(flag)
    imagedata[i,j]=0,fill filling color on the pix
  else
    imagedata[i,j]=1,fill background color on the pix
end of loop
end of loop
image(TraceCount * TraceDistance,(SampleCount−1) * Sample-
Height,imagedata)
```

其中，scale 为根据显示要求的拉伸插值函数，map 用于计算当前像素是位于标记点的左边还是右边，image 函数用于根据得到的 uchar 数据一次性转换为图片。分析上述伪代码可知每个像素点的计算是彼此独立的，同时处理的方法是完全一致的，满足 GPGPU 设计准则：处理的数据之间没有相干性并且处理数据的方法完全一致，因此可以通过 GPU 语言 CUDA 实现该并行算法，一次性完成所有像素点的计算。同时，通过上述的分析可知基于 GPU 的并行绘制算法的时间复杂度为 O(1)，与基于通用绘制引擎的绘制算法的时间复杂度相比，基于 GPU 的并行绘制算法的绘制效率有相当明显的提高。

根据上述的分析，基于 CUDA 的并行地震数据快速绘制算法主要由以下 4 个步骤实现。

(1) 地震数据预处理：为了适应 CUDA 编程模型的三层结构[13]，需要将地震数据分成适当大小的块。

(2) 逐道纵向插值：根据 Y 方向的间距和每道地震数据点数，纵向插值计算波形曲线经过的像素点数和像素点的位置。

(3) 计算块内像素点的颜色值：将根据式(2)计算出的像素点作为边界点，计算块内像素点和边界点的位置关系，将满足填充条件的像素点设置成填充色，其他设置成背景色。

(4) 像素点渲染：根据计算出的像素点位置和像素点的颜色直接渲染像素点，生成 B+W 图。

基于 CUDA 的并行地震数据快速绘制算法的实现流程如图 6 所示，CUDA 线程结构如图 7 所示。

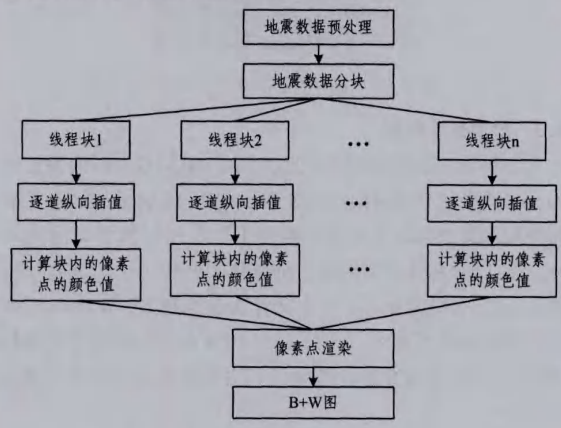

图 6　基于 CUDA 的并行算法实现流程

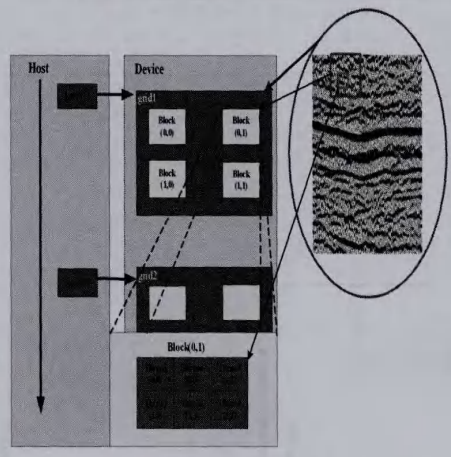

图 7　CUDA 线程结构

3　实验结果

3.1　仿真环境

本文的仿真在 Windows 7、3.06GHz 英特尔 Pentium 双核 E6600、4GB 内存、Nvidia GeForce GTX 560 下进行。编程环境为 VS2008，编程语言为 C++、CUDA。

3.2　绘制效果对比

常规地震数据图形绘制方法的效果如图 8 所示，基于 GPU 的地震数据图形快速绘制方法的效果如图 9 所示。通过图 8 和图 9 的对比可知本文提出的绘制方法在显示效果上与常规方法一样。

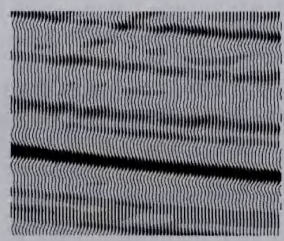

图 8　基于通用绘制引擎的常规绘制算法绘制的 B+W 图

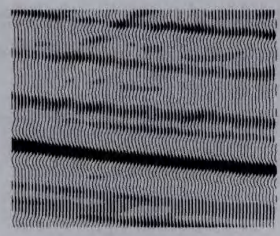

图 9　基于 GPU 的并行绘制算法绘制的 B+W 图

3.3　绘制效率对比

各种算法绘制 B+W 图的效率的对比以绘制时间为比较标准。总的绘制时间的对比如表 1 和图 10 所示，图 11 表示传统绘制算法与本文的算法绘制 B+W 图时节省时间的对比。图 12 表示几种绘制算法绘制 B+W 图的一个点的时间，图 13 表示传统绘制算法和本文算法在绘制 B+W 图的一个点时绘制时间差对比。图 11 和图 13 中的运行时间差是指利用传统 CPU 绘制方法和利用 GPU 并行绘制方法的运行时间之差。

表 1　各种算法绘制 B+W 图的时间比较

测试数据编号	测试数据（以 float 型为例）	基于 Qt 方法的绘制时间（ms）	基于 OpenGL 方法的绘制时间（ms）	基于 GPU 并行方法的绘制时间（ms）
1	250 * 375	143	548	14
2	250 * 750	260	695	23
3	250 * 1500	577	690	21
4	500 * 1500	1193	1217	35
5	1000 * 1500	2329	2143	51
6	1000 * 3000	5371	3661	79
7	1000 * 6001	13391	5763	153
8	1559 * 6001	22330	8738	224

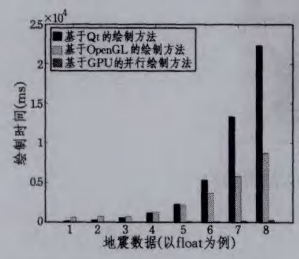

图 10　各种算法 B+W 图的时间对比直方图

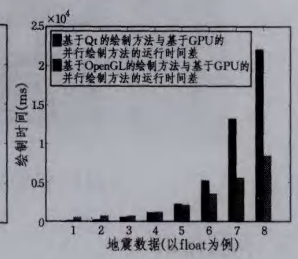

图 11　B+W 图绘制算法节省时间对比直方图

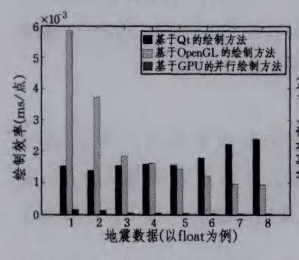

图 12　各种算法绘制 B+W 图的效率对比直方图

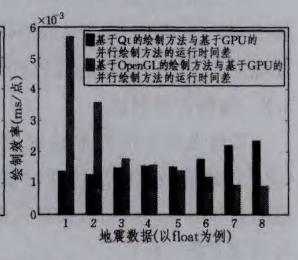

图 13　B+W 图单个点绘制节省时间对比直方图

针对不同规模的数据，对比常规绘制方法和基于 GPU 的并行绘制方法的总的绘制时间可以看出，在地震数据为 1000 * 6001 时本文提出的方法的绘制时间是常规方法绘制时间的 1/100，效率提升相当明显。

结束语　本文结合地震数据的特点和图形光栅化的原理，利用 GPU 的并行计算能力，提出了基于 GPU 的地震剖面图形快速绘制算法。仿真实验表明：(1)基于本文算法的地震剖面的绘制效果和基于通用绘制引擎的地震剖面的绘制效果是一致的，本文算法保证了地震剖面图形绘制的效果；(2)对不同规模的地震数据的绘制时间的对比表明，基于本文算法的地震剖面图形绘制的效率得到了很大的提高。

参 考 文 献

[1] Kehrer J, Hauser H. Visualization and Visual Analysis of Multifaceted Scientific Data: A Survey[J]. IEEE Transactions on Visulization And Computer Graphics, 2012, 19(3): 495-513

[2] Lyubomir G, Zagorchev. A Curvature-Adaptive Implicit Surface Reconstruction for Irregularly Spaced Points[J]. IEEE Transaction on Visualtion & Computer Graphics, 2011, 18(9): 1460-1473

[3] Zhu Liang-feng, Pan Xin, Wu Xin-cai, et al. Construct three-dimensional visualization methods geological fault model and implementation techniques [J]. Journal of Software, 2008, 19 (8): 2004-2017(in Chinese)
朱良峰，潘信，吴信才，等. 地质断层三维可视化模型的构建方法与实现技术[J]. 软件学报，2008，19(8)：2004-2017

[4] Wan Min, Wang Yu. Variational surface reconstruction based on Delaunay triangulation and graph cut[J]. International Journal for Numerical Methods in Engineering, 2011, 85(2): 2011

[5] Chen Cheng-kai, Correa H C, Ma K-L, et al. Visualizing 3D Earthquake Simulation Data[J]. Computing in Science & Engineering, 2010, 13(6): 52-63

[6] Xiao Han. Research studies of seismic data visualization [D]. Changsha: Hunan University, 2007: 23-25(in Chinese)
肖汉. 地震数据的可视化研究技术研究[D]. 长沙：湖南大学，2007：23-25

[7] Huang Dan. QT-based seismic data visualization research and application of [D]. Chengdu: University of Electronic Science and Technology of China, 2011: 45-50(in Chinese)
黄丹. 基于 QT 的地震数据可视化的研究及应用[D]. 成都：电子科技大学，2011：45-50

[8] Shi Zhi-xin. Study basic raster graphics generation algorithm[D]. Jinan: Shandong University, 2007(in Chinese)
石志昕. 基本光栅图形生成算法研究[D]. 济南：山东大学，2007

[9] Xu Sai-hua, Zhang Er-hua. Parallel CUDA-based 3D data visualization[J]. CT Theory and Applications, 2011, 20(1): 47-54(in Chinese)
徐赛花，张二华. 基于 CUDA 的三维数据并行可视化[J]. CT 理论与应用研究，2011，20(1)：47-54

[10] Xie K, Wu P, Yang S. GPU and CPU cooperation parallel Visualisation for large seismic data[J]. Electronics Letters, 2010, 46 (17): 1196-1197

[11] Patel D, Bruckner S, Viola I, et al. Seismic Volume Visualization for Horizon Extraction[C]// IEEE Pacific Visualization Symposium. 2010: 73-80

[12] Xiao Han. The use of parallel GPU computing bilinear interpolation algorithm[J]. Journal of Chinese Computer Systems, 2010, 31(11): 2241-2245(in Chinese)
肖汉. 利用 GPU 计算的双线性插值并行算法[J]. 小型微型计算机系统，2010，31(11)：2241-2245

[13] NVIDIA CUDA. Programming Guide Version 3. 0[OL]. http://developer. download. nvida. com/compute/cuda/3_0/toolkit/docs/NVIDIA_CUDA_ProgrammingGuide. pdf