

文章编号:2095-6134(2015)04-0549-07

一种面向 iOS 系统的双缓冲改进算法及其应用^{*}

石 锐^{1†}, 杨崇俊¹, 张建兵², 王 锋¹

(1 中国科学院遥感与数字地球研究所 遥感科学国家重点实验室, 北京 100101; 2 中国石油大学(北京)信息学院, 北京 102249)
(2014 年 7 月 7 日收稿; 2015 年 1 月 5 日收修改稿)

Shi R, Yang C J, Zhang J B, et al. A modified double-buffering algorithm oriented to iOS and its application [J]. Journal of University of Chinese Academy of Sciences, 2015, 32(4): 549-555.

摘 要 为提高面向 iOS (iPhone operating system) 系统的矢量地图数据渲染效率, 研究针对小尺寸屏幕的移动设备的矢量数据简化算法, 以传统的 Douglas-Peucker 矢量简化算法和切分重组压缩法为基础, 讨论结合屏幕像素、容忍时间等因素确定简化限值的方法. 另外, 基于对 Quartz 2D 引擎的渲染流程及双缓冲技术使用的效率问题的研究, 提出改进的双缓冲渲染算法. 实验结果显示, 改进后的算法与传统算法相比, 渲染效率有所提高.

关键词 iOS; 双缓冲; 矢量数据简化; Quartz 2D
中图分类号: P208 **文献标志码**: A **doi**: 10.7523/j.issn.2095-6134.2015.04.018

A modified double-buffering algorithm oriented to iOS and its application

SHI Rui¹, YANG Chongjun¹, ZHANG Jianbing², WANG Feng¹

(1 State Key Laboratory of Remote Sensing Information Sciences, Institute of Remote Sensing and Digital Earth, Chinese Academy of Sciences, Beijing 100101, China; 2 China University of Petroleum (Beijing), Beijing 102249, China)

Abstract In order to improve the rendering efficiency of iOS-oriented (iPhone operating system) vector map, a method for determining the simplification limit for small-screen mobile devices is proposed through the thorough analysis of the simplification algorithm of vector data. On the basis of Douglas-Peucker algorithm we use this method to quantitatively calculate simplification limit, considering the factors of screen pixels and time tolerance. In addition, a modified algorithm is proposed for improving the efficiency of double buffering skill, on the basis of the study on rendering procedure and double buffering skill with Quartz 2D. The experimental results show that the modified algorithm is more efficient than the traditional one.

Key words iOS; double buffering; vector data simplification; Quartz 2D

GIS 发展的最终目的是使 GIS 应用平民化^[1]. 如今已步入移动智能时代, 截至 2013 年 12 月中国的智能手机用户已经突破 3.88 亿人^[2]. 传统的桌面 GIS 已经不能满足 LBS 空间信息的动态特点和为用户提供智能化服务的要求, 移动 GIS 就应运而生了^[3]. iOS 系统是由苹果公司开发

^{*} 国家 863 计划项目(2012AA12A401, 2013AA12A403)资助

[†] 通信作者, E-mail: shirui@irsa.ac.cn

的移动操作系统, 在中国的智能手机市场占有率今年稳定保持在 15% ~ 20%, 并预测今后几年有稳步上升趋势; 而 2014 年 2 月的 TIOBE 编程语言排行榜显示, iOS 设备的主要开发语言 objective-c 也以 11.341% 的占有率成为继 C、Java 之后的第 3 大编程语言。可见, 近年来 iOS 系统在全球都已是炙手可热。iOS 系统具有运行稳定流畅、用户体验人性化等优点, 其载体 (iPhone/iPad) 的屏幕分辨率也较高, 因此 iOS 系统成为 GIS 开发的重要平台之一^[4]。

前人对于移动端矢量地图快速显示做了大量研究。LOD (levels of detail) 是地图快速显示的常用技术之一; 陈涛等^[5]设计了一种适合嵌入式 GIS 环境基于 LOD 的地图数据组织模型, 对于地图快速显示有很好的效果; Douglas-Peucker 矢量数据简化算法已经被广泛应用并进行了改进^[6-7], 可用于地图数据简化, 但没有提出明确的简化限值定量方法, 无法最合理地简化数据; 黄雁等^[8]通过双缓冲技术来减小系统响应时间, 但张云贵^[9]在试验中发现基于位图的双缓冲技术不完全适合 iOS 系统。

在本文中, 我们对于数据简化算法进行研究并针对简化限值定量问题提出新方法以合理地简化数据。另外, 我们还对基于 Core Graphics 框架下的渲染流程及双缓冲技术使用的效率问题进行了研究, 并提出改进的双缓冲渲染算法。

1 矢量数据的定量简化

移动设备通常显示屏较小, 内存有限^[10], 高精度数据在显示时会因数据量大而占用过多内存空间, 且很可能出现多个点重合显示在一个像元的现象, 出现数据冗余。对数据进行简化压缩能够很好地减小数据冗余, 并提高系统效率。另外, 移动软件开发非常重要的一点在于人机交互环节的设计, 过长的交互时间会导致用户急躁、厌烦^[11]。基于以上考虑, 本节探讨矢量数据的定量简化方法, 即在 Douglas-Peucker 简化算法基础上, 结合移动设备的像素大小、显示比例尺和用户容忍时间 (用户使用时感觉明显卡顿的临界点所需的等待时间) 等因素确定简化限值, 在满足显示速度和精度要求的情况下尽量压缩数据。

1.1 矢量数据分级

简化前, 首先要将数据进行分级, 即使用

LOD 技术, 在大比例尺时使用较为简略的数据, 在小比例尺时使用精细的数据。这样可以避免图幅拥挤, 提高地图辨识度, 提高渲染效率^[12]。

具体应用时, 应分别给每个图层指定显示级别区间以确定在某一比例尺时是否显示该图层。对于基础地理要素 (如行政边界、各级道路、铁路、河流等) 在国标《基础地理信息要素分类与代码》中根据重要性特征进行了要素编码, 可以以此设定显示级别区间; 对于非基础地理要素, 如学校、酒店、商业街区、街区居民地等, 其重要性特征没有统一的标准, 则可以根据这类地理要素自身包含的等级属性和用户需求 (如酒店星级、重点中学与否) 等来确定显示级别区间^[13]。

1.2 数据简化及限值确定

在确定好各图层数据显示的级别和比例尺大小后, 方可进行数据简化。点状数据简化较容易, 只需要根据点属性, 将重要的保留, 将非重点剔除即可。多段线和多边形的简化, 即用尽可能少的采样点来描述原始地物, 并保证在容许的误差限度内, 再现地物的形态特征^[14]。

Douglas-Peucker 法是多段线和多边形简化的常用算法, 其执行效率较高^[5], 禹铭月和王卫安^[15]通过验证表明该算法的简化效果最好, 但对于相邻多边形有公共边的情况 (如地图数据) 并不完全适用, 会出现拓扑错误^[16]。张胜等^[7]在此基础上进行改进, 提出切分重组压缩法, 解决了该拓扑问题, 但对于如何定量简化限值没有做明确说明。我们使用切分重组压缩法作为基础算法, 借鉴卢岚等^[17]提出的定量法采样法思想并加以改变, 结合等待时间和屏幕像素等因素, 探讨简化限值的定量方法如下。

1.2.1 空间信息量与等待时间

用户在使用地图时每次操作 (缩放、平移、旋转等) 的最长等待时间称为容忍时间。等待时间主要由以下 3 部分组成:

等待时间 = 重绘时间 + 数据组织时间 + 手势响应时间。

通过测试表明等待时间中图形绘制语句执行占用的时间占总时间的 80% 以上, 重绘操作占用的时间很大程度上决定了操作等待时间。为寻找图形重绘所需的时间与空间信息量 (SDA, spatial data amount, 可以由地图数据的某个特征变量来表示, 本文使用总点数表示) 之间的函数关系, 我

们分别在 iPhone 5S 设备和 iPad mini 设备上绘制 SDA 为 1 万个点至 20 万个点的数据,并拟合成图 1 所示的 SDA-Time 曲线.从图中可以看出,在 2 种设备中都呈现出一条通过原点的直线.由此可以得出结论,重绘时间与空间信息量之间存在函数 $y = k \times x$ 的关系,其中 k 与运行环境的性能相关,且在同一台设备中是基本不变.

由此,根据设备的 SDA-Time 曲线,可以由容忍时间来确定 SDA 的容忍值.为确定容忍时间,通过快速连续触碰屏幕实验,得到连续 2 次触摸屏幕的最小间隔为 112 ms;而重绘时间占总响应时间的 80% 以上,因此每次重绘应控制在约 90 ms 以内不会感到卡顿.图 1 中,根据拟合成的曲线可以得到,在 iPhone 5S 上 SDA 容忍值约为 11.85 万个点,在 iPad mini 上 SDA 容忍值约为 3.13 万个点.

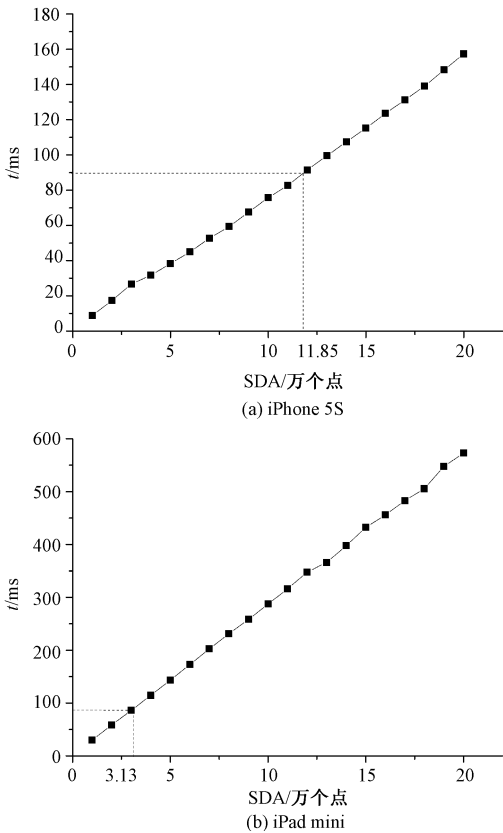


图 1 SDA-Time 曲线
Fig. 1 SDA-Time curve

1. 2. 2 简化限值的确定

根据上述测试以及讨论,可以通过以下流程确定简化限值.

1) 通过实验测试及用户体验确定用户容忍

时间,即将 SDA 由小到大加载进行重绘测试,根据用户判断,卡顿明显时即为容忍时间;

2) 根据用户容忍时间确定 SDA 容忍值,即容忍时间所对应的总点数;

3) 根据 LOD 分级的具体情况对各图层数据进行压缩.在某级别比例尺时,若需显示的总点数大于 SDA 容忍值,则利用切分重组压缩法将数据压缩至总点数小于 SDA 容忍值;若小于 SDA 容忍值,为避免出现多点重合显示在某一像元的情况,使用平均每像元显示的实际距离作为切分重组压缩法的限值,即

$$T = D_0 = \frac{\text{span}}{\text{pixels}} = \frac{1}{\text{scale}}, \tag{1}$$

其中, T 为简化限值, D_0 为每像元显示的实际距离, span 为屏幕中图幅跨度, pixels 为横向(纵向)像素个数(如 iPhone 5s 取值为 640), scale 为当前比例尺.

2 双缓冲绘图技术分析及改进

2.1 双缓冲渲染算法分析

如果重绘时间大于屏幕的刷新时间,在视觉上就会造成一种不连贯的效果^[18].双缓冲绘图技术是传统的绘图优化技术,其主要思路是在原有缓存基础上增加一个提前组织待显示数据的缓存,目的是将数据处理过程隐藏起来,在处理器空闲时进行,避免数据处理过程导致漫游卡顿现象.

Quartz 2D 是在 iPhone OS 环境和 Mac OS 环境下常用的二维绘图引擎^[19].使用该引擎实现双缓冲技术包括 2 个阶段:第 1 阶段为在首次渲染或在地图平移、缩放、旋转等漫游操作之后停止触摸 (`touchesEnded`) 重绘图形时,不直接在默认视图上下文中渲染图形,而是在内存中另外创建一个位图上下文,在绘制图形时先将各图元绘制到位图上下文中,再创建位图上下文的视图快照,将视图快照一次性显示到系统默认的视图上下文中;第 2 阶段即需要进行地图漫游操作时,此时指触摸屏幕事件触发 (`touchesMoved/handlePinchFrom`),一般使用已经生成好的缓存位图直接显示在视图上下文中来代替在视图上下文中使用 Quartz 2D 的绘图句柄重绘矢量图元.

何宇和林晓煊^[20]运用双缓冲技术成功在 iOS 系统中解决了重绘时屏幕闪烁和波形不连续的问题.但张云贵^[9]在测试中发现基于位图的双缓冲

技术不适合 iOS 设备. 为分析该技术在 iOS 系统中的适用性,我们在 iPhone 5s 设备(屏幕尺寸:4 英寸,像素 1 136 × 640,CPU:苹果 A7/M7 协处理器 1 331 MHz,GPU:4 核 PowerVR G6430,内存:1 G)中做了如下试验.

首先将 SDA 为 5 万个点的矢量地图数据分别使用无缓冲和有缓冲 2 种方式进行重绘,对每一环节进行多次测试后得到如表 1 的平均结果.

表 1 重绘效率测试						
Table1 Test of reprint efficiency						
	创建	清除	在上下文中	重建缓存	显示缓存	总计
	上下文	背景	绘制图形	位图	位图	
无缓冲	—	—	38	3	—	41
有缓冲	2	10	38	1	15	66

从结果可以看出,无缓冲方式重绘直接在视图上下文中绘制,在绘制完成后建立一张缓存位图即重绘完成,用时较短;而使用双缓冲之后需要先创建位图上下文,绘制之前先做清屏处理,绘制之后同样需要创建缓存位图,最后将缓存位图一次性显示到视图上下文中,相比前一种方式用时较长. 但仔细观察发现,使用双缓冲比不使用双缓冲耗时的主要原因是多出了创建位图上下文、清除背景,以及显示缓存位图 3 个环节,而在上下文中绘制的耗时是一样的. 为了对比数据量的不同是否会影响在视图上下文和位图上下文中的耗时关系,我们将 SDA 从 2 千个点至 4 万个点的矢量地图数据分别用 2 种方式绘制,耗时情况如图 2.

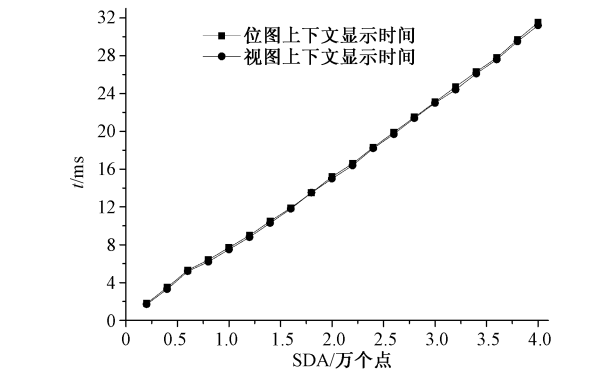


图 2 在 2 种绘图环境中绘制的效率对比

Fig. 2 Comparison between the efficiencies in the two contexts

从图 2 中可以看出,在 2 种绘图上下文中渲染的耗时几乎一样,并且没有随数据量的变化表现出差异. 研究其原理发现,在位图上下文中渲染是在 CPU 中完成的,将渲染完成的位图再传递给

GPU 进行绘制,速度较慢并且不能使用硬件加速特性^[21].

对于第 2 阶段的测试,同样使用 SDA 从 2 千个点至 4 万个点的数据用 2 种方法显示,以对比缓存位图直接显示与在视图上下文中重绘图元的效率,得到如图 3 所示结果.

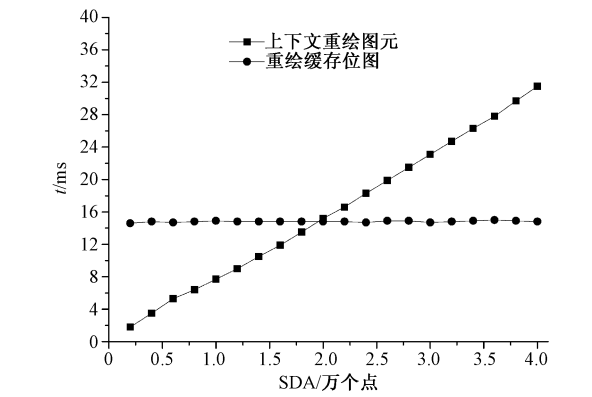


图 3 2 种方式效率对比

Fig. 3 Efficiency comparison between the two methods

从图 3 中可以看出,在绘图上下文中重绘图元的耗时随着 SDA 值的增加呈线性增长,而重绘缓存位图的耗时处于非常稳定的水平. 当 SDA 值小于约 1.95 万个点时,在上下文重绘图元耗时更少,当 SDA 大于 1.95 万个点时,重绘缓存位图效率更高.

在图形上下文中重绘图元的过程,每一个图元都要调用语句逐一绘制,因此时间复杂度与图元个数成线性相关. 对于大数据量的矢量图形绘制时,该方法就显得效率偏低,使用重绘缓存位图的方式无疑更加高效.

显示缓存位图的耗时稳定是由于 CGContextCreateImage 函数创建的缓存位图是栅格数据^[22],其绘制速度与像素大小有关,与图元多少无关;但在数据量较小时,该方法绘制效率反而不及矢量绘图速度,原因是 iOS 系统中推荐使用的栅格数据格式为 PNG,是压缩数据格式,在绘图前都要先对数据进行解压缩,因此耗费一些时间.

综上可以得出初步结论,在 iOS 系统中,使用双缓冲技术有其优势之处,如第 2 阶段在数据量较大的时候能明显提高渲染效率;但在小数据量时,双缓冲技术并没有提高渲染效率,甚至在重绘阶段运用该技术反而因增加了渲染步骤而降低效率. 因此,考虑到数据量的影响,将算法进行改进.

2.2 双缓冲渲染算法改进

由于双缓冲技术的适用性与数据量有关,因此可以在首次渲染时计算数据量界值,以此界定在渲染进行中动态判断是否如何执行渲染.在视图上下文中重绘图元的时间与数据量大小呈正线性关系,因此可以假设该关系为

$$T_1 = k \times SDA + b.$$
 (2)

当数据量为 0 时,渲染时间也为 0,故 b 值为 0. 此系数在数据首次渲染时就可以通过渲染时间与数据量大小求出. 再观察缓存位图显示时间,为非常稳定的一条平行于横轴的直线. 即得到缓存位图显示效率函数为

$$T_2 = t.$$
 (3)

在首次渲染时,若运用双缓冲技术,则在显示缓存位图时就可以得出 t 值. 联立二式得

$$SDA_0 = t/k,$$
 (4)

SDA_0 就是数据量界值,当数据量大于 SDA_0 时显示缓存位图可以提高效率,小于 SDA_0 时直接在视图上下文中重绘图元效率更高.

另外,在使用双缓冲技术的同时,联合 grand central dispatch (GCD) 多线程技术,将数据准备过程放在逻辑线程中,在后台执行,主线程中只执行手势函数及 drawRect 函数 (drawRect 只能在主线程中执行). 主线程接到漫游 (平移、缩放、旋转) 命令时立刻传递指令给渲染线程,此时渲染线程负责查询索引、在位图上下文中渲染以及重建缓存位图的一系列任务,在所有任务执行完毕后通知主线程调用 drawRect 方法显示缓存位图并刷新 UI. 通过此方法,在长时间的重绘大数据量过程中不会阻塞 UI,明显降低了卡顿现象,提升了用户体验.

2.3 改进后的渲染流程

改进后的渲染流程如图 4 所示. 其中,

- 1) 查询索引是指按照屏幕显示范围在索引中查询范围内的对象;
- 2) 视图上下文指系统默认的在 drawRect 函数中所有描画命令的工作环境;
- 3) 位图上下文指在内存中用 CGContextCreate 创建的绘图上下文;
- 4) 重建缓存位图指用 CGContextCreateImage 函数创建位图上下文的视图快照;
- 5) 显示缓存位图指将 CGContextCreateImage 创建好的视图快照在视图上下文中显示

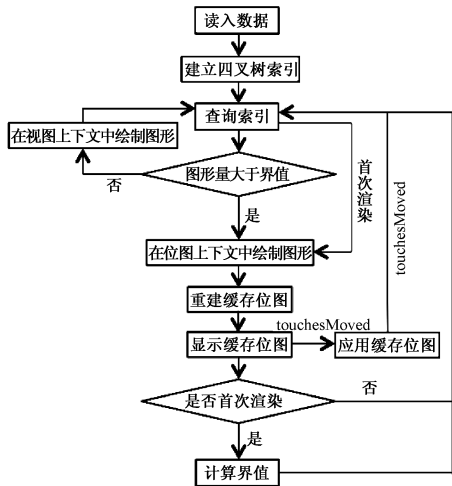


图 4 改进后的渲染流程

Fig. 4 The modified rendering flow

出来;

6) 应用缓存位图指在漫游操作过程中,用显示缓存位图的方式代替图形重绘以加快显示.

3 实验结果和分析

3.1 实验平台和数据描述

基于上述研究,依托全球海量空间信息集成分析与主动服务技术平台中的移动信息系统,图 5 为系统功能展示图. 该系统有导航定位、空间数据管理、空间数据可视化、空间数据查询、空间数据分析、无线数据通信等功能. 支持常规 GIS 数据格式,支持超大量影像数据,包括 TIF 等格式数据,能够进行图层管理,包括图层可见,可编辑、设置显示比例等,以及显示风格管理,包括点线面文本图层的显示风格设定等. 同时还具有专题图功能包括单值专题图、分段专题图等,还可进行点线面数据 GPS 采集,手动勾绘等功能.

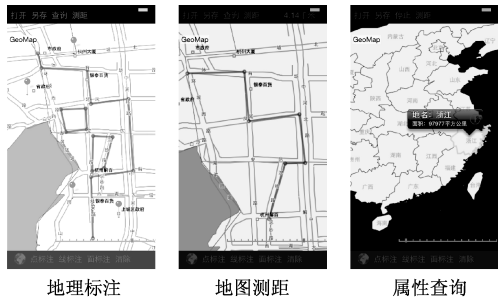


图 5 系统功能展示

Fig. 5 Function of system

将本文中的技术和算法应用到移动采集系统中进行测试. 实验使用的数据格式为 ESRI

Shapefile 矢量数据,数据内容为全国、各省、地级市、区县以及上海市乡镇的行政界线,坐标采用

WGS84 坐标系.实验数据在显示前均根据文中方法进行简化,得到最终实验数据(如表 2).

表 2 实验数据

Table 2 Experimental data

图层名称	LOD 级别	比例尺范围	简化前大小/KB	简化后大小/KB
中国行政界限	1	(0,1:1 344 800 00]	2 349	43
全国省级政区界限	2	(1:134 480 000,1:33 620 000]	12 212	177
全国市级行政界限	3	(1:33 620 000,1:2 101 000]	29 544	664
全国县级行政界限	4	(1:2 101 000,1:525 000]	63 492	4 650
全国乡镇级行政界限	5	(1:525 000,+∞)	246 496	21 496

3.2 实验结果

将 SDA 为 2 千个点至 4 万个点的数据量依次在系统中进行渲染,对比 2 种算法在地图漫游时的渲染效率,得到如图 6 和表 3 所示的结果.

从以上图表中可以看出,改进后的双缓冲算法在 SDA 小于 1.95 万个点时渲染时间随 SDA 值的增加而增加,但其效率明显高于传统双缓冲算法;在 SDA 大于 1.95 万个点时改进后的算法保持与传统算法相同的效率,且远远低于用户容忍时间.因此可以得出结论,改进后的双缓冲算法在继承传统算法对于大数据量可以高效渲染的优点的同时,明显提高了小数据量时的渲染效率,使算法在整个数据量范围内均表现出较优的效率.

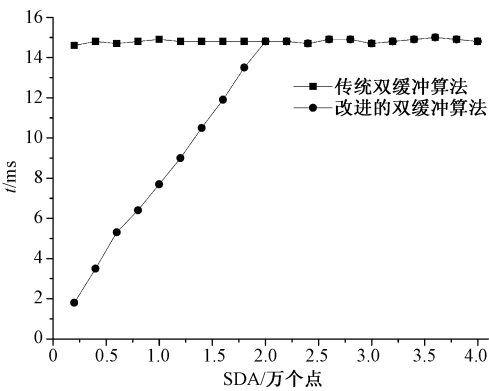


图 6 改进后的算法效率

Fig. 6 Efficiency of the modified algorithm

表 3 2 种算法效率对比

Table 3 Efficiency comparison between two algorithms

SDA/万个点	0.2	0.4	0.6	0.8	1.0	1.2	1.4	1.6	1.8	2.0
传统算法/ms	14.6	14.8	14.7	14.8	14.9	14.8	14.8	14.8	14.8	14.8
改进算法/ms	1.8	3.5	5.3	6.4	7.9	9.0	10.5	11.9	13.5	14.8
SDA/万个点	2.2	2.4	2.6	2.8	3.0	3.2	3.4	3.6	3.8	4.0
传统算法/ms	14.8	14.7	14.9	14.9	14.7	14.8	14.9	15.0	14.9	14.8
改进算法/ms	14.8	14.8	15.0	14.9	14.7	14.9	14.9	14.9	14.8	14.9

在系统使用时,通过变换比例尺,在各缩放级别测试屏幕当前图形显示量、漫游时间以及重绘

时间,得到如表 4 的统计.

表 4 实验测试平均结果

Table 4 Average results of the experimental test

比例尺/ $\times 10^3$	1:134 480	1:67 240	1:33 620	1:8 405	1:4 202	1:2 101	1:1 050	1:525	1:262
平均数据量/万个点	0.65	1.50	1.10	4.50	3.30	2.00	6.10	3.30	4.30
漫游时间/ms	6.0	12.9	10.2	14.8	14.9	14.9	14.8	14.7	14.8
重绘时间/ms	6.0	12.9	10.2	40.1	28.3	17.8	54.3	30.0	38.2

从表 4 中可以看出,经过数据简化后,显示到屏幕的数据量都小于容忍值并随比例尺变化;漫游时间都保持在 15 ms 及以下的高效状态,这个效率满足平滑的漫游效果;数据重绘时间也控制在较低的水平,通过多线程技术将此步骤放在后台执行,没有明显的卡顿.可见,文中涉及的关键

技术在实际应用中效果良好。

4 结语

为了提高面向 iOS 系统的矢量数据渲染效率,本文详细研究了矢量数据简化算法,提出基于简化限值的数据简化新方法,可根据硬件条件及效率需求定量的简化矢量数据,减小数据冗余。本文还重点研究 iOS 系统的渲染特性,并对传统双缓冲渲染算法进行改进。实验证明,改进后的双缓冲算法在继承传统算法对于大数据量可高效渲染优点的同时,在小数据量时渲染效率明显提升,使得该算法在整体数据量范围内均表现出较优的效率。改进后的双缓冲算法可用于面向 iOS 系统的地图、画板等各种需要渲染矢量数据的应用当中。

参考文献

- [1] 许捍卫. 移动 GIS 发展的关键技术及应用前景[J]. 测绘工程,2003,10(1):34-36.
- [2] IDC. China's smartphone shipments to exceed 450 million by 2014 [EB/OL]. (2013-09) [2014-06-30]. <http://www.idc.com/getdoc.jsp?containerId=prCN24344613>.
- [3] 陈飞翔,杨崇俊,申胜利,等. 基于 LBS 的移动 GIS 研究[J]. 计算机工程与应用,2006(2):200-202.
- [4] 张丽娜. 基于 iOS 的智能交通信息发布系统的设计与实现[D]. 济南:山东大学,2012:11-15.
- [5] 陈涛,翟京生,陈双军,等. 嵌入式 GIS 中基于 LOD 的地图数据组织模型设计[J]. 测绘科学技术学报,2011,28(5):374-377.
- [6] 张宏,温永宁. 地理信息系统算法基础[M]. 北京:科学出版社,2006:91-92.
- [7] 张胜,朱才连,钟世明. Douglas-Peucker 算法的改进及应用[J]. 武汉理工大学学报,2005,29(5):671-674.
- [8] 黄雁,冯艳杰,孟庆祥. 嵌入式 GIS 矢量数据的快速显示方法研究[J]. 城市勘察 2012,2(1):20-23.
- [9] 张云贵. 面向移动设备的矢量绘图平台设计与实现[D]. 北京:北京理工大学,2013:17-19.
- [10] 郭峰林,胡鹏,徐小双,等. 移动 GIS 中可视化技术研究[J]. 测绘科学,2007,32(6):11-15.
- [11] 王悦,岳玮宁,王衡,等. 手持移动计算中的多通道交互[J]. 软件学报,2005,16(1):29-36.
- [12] 徐智勇,吴小芳. LOD 技术在电子地图显示中的应用研究[J]. 测绘信息与工程,2004,29(5):19-20.
- [13] 安晓博,陈蜀宇,刘小威. 嵌入式 GIS 地图快速显示方法的应用[J]. 计算机系统应用,2012,21(10):204-207.
- [14] 贾利峰. 无级比例尺 GIS 的线状要素简化算法研究[D]. 成都:西南交通大学,2005:12-13.
- [15] 禹铭月,王卫安. 多边形形状简化及其质量评价[J]. 测绘与空间地理信息,2011,34(6):152-155.
- [16] Yang L, Zhang L Q, Kang Z Z, et al. An efficient rendering method for large vector data on large terrain models [J]. Science China Information Sciences, 2010, 53(6): 1122-1129.
- [17] 卢岚,刘兴权,蔡俊. 矢量数据 LOD 模型中尺度采样方法[J]. 矿山测量,2013,4(2):24-26.
- [18] 王莹,南敬昌. 基于双缓冲机制的多尺度导航电子地图显示设计[J]. 计算机测量与控制,2012,20(9):2477-2480.
- [19] 动态几何画板的研究及其在 iPhone 平台的实现[D]. 成都:电子科技大学,2011:11-22.
- [20] 何宇,林晓焕. 基于 iOS 双缓冲绘图技术的研究[J]. 电子设计工程,2013,21(21):46-48.
- [21] Jack N, Fredrik O, Dave M, et al. Beginning iOS7 development [M]. California: Apress, 2014:489-518.
- [22] Apple Inc. Drawing with Quartz 2D [EB/OL]. (2013-12) [2014-06-30]. https://developer.apple.com/library/mac/documentation/graphicsimaging/reference/CGBitmapContext/Reference/reference.html#apple_ref/c/func/CGBitmapContextCreateImage.