Computer Science and Application 计算机科学与应用, 2016, 6(1), 9-20
Published Online January 2016 in Hans. http://www.hanspub.org/journal/csa
http://dx.doi.org/10.12677/csa.2016.61002

Hans汉斯

Research on the Performance Optimization of Drawing Front-End Vector Features Based on WebGIS of HTML5

Cui Li, Zhihong Li*, Lijie Zhou

Key Laboratory of Geographic Information Science, Ministry of Education, College of Geographical Sciences, East China Normal University, Shanghai
Email: *861775491@qq.com

Received: Jan. 6th, 2016; accepted: Jan. 22nd, 2016; published: Jan. 27th, 2016

Abstract

Traditionally, the client vector graphics technology is implemented via plug-ins, but at the risk of the browser disabling the plug-ins. To solve this problem, this paper puts forward the HTML5 Canvas drawing technology based on modern browsers, achieves visualization and interaction of geographical features in WebGIS based on HTML5 Canvas, gives the realization process of drawing vector features and complex island or hole features, and further analyzes the influence factors of Canvas graphics performance mainly from four aspects: the amount of features, features' nodes, drawing style and browser difference. And then the performance analysis tool is used to find the bottleneck and make the performance optimizations of drawing features. Finally, this technology is applied to two cases successfully with good drawing effects.

Keywords

HTML5 Canvas, WebGIS, Island or Hole Features, Drawing Performance, Performance Optimization

基于HTML5的WebGIS前端要素绘制性能优化研究

李　翠，李治洪*，周力杰

*通讯作者。

文章引用: 李翠, 李治洪, 周力杰. 基于 HTML5 的 WebGIS 前端要素绘制性能优化研究[J]. 计算机科学与应用, 2016, 6(1): 9-20. http://dx.doi.org/10.12677/csa.2016.61002

华东师范大学地理科学学院，地理信息科学教育部重点实验室，上海
Email: *861775491@qq.com

收稿日期：2016年1月6日；录用日期：2016年1月22日；发布日期：2016年1月27日

摘　要

传统地，客户端矢量绘图技术采取插件机制的形式实现，但需要承担浏览器禁用插件的风险，针对这种弊端，文中提出现代浏览器中的绘图技术即HTML5 Canvas绘图技术，利用HTML5 Canvas实现WebGIS中地理要素的展示与交互，给出绘制矢量要素以及复杂岛洞要素的实现过程，并进一步对Canvas绘图性能的影响因素进行分析，主要是从不同要素数量、要素结点数、要素绘制样式、不同浏览器四个方面对比绘图性能，并利用性能分析工具找出绘制性能的瓶颈，对绘图性能进行优化。最后，将此绘图技术成功运用在两个案例中，具有良好的绘图效果。

关键词

HTML5 Canvas，WebGIS，岛洞要素，绘图性能，性能优化

1. 引言

随着互联网技术的快速发展，在客户端进行数据的图形化展示与交互也受到越来越多的关注。其中，WebGIS [1]中的地理要素的可视化显示，也需要在客户端对点、线、面、以及复杂地理要素进行绘制与编辑等交互操作。现有 WebGIS 系统中在客户端实现矢量绘图技术根据其底层实现机制，主要分为两类[2]：一种是外部代码载入方式如 Flash、Applet、SilverLight 等，一种是浏览器原生支持的方式如 Canvas、SVG、VML 等。第一种方式因为要导入外部执行环境，通常要安装插件，而在浏览器禁用插件后，相关的功能也会受到影响，有一定风险。相比较而言，后一种方式则直接由浏览器原生支持，但在浏览器支持性以及绘图效率等方面也有较大差异。

VML (Vector Markup Language，矢量可标记语言)是一种基于 XML 描述的矢量图形，支持高质量的图形显示。在 WebGIS 客户端引入 VML，可以弥补 WebGIS 中绘图的诸多不足，它作为 IE 浏览器内置的绘制工具，无需任何额外的组件，不仅提高了开发进度，而且增强了系统的适用性[3]。但它仅被 IE 支持，存在很大的兼容性问题，并且在高版本的 IE 浏览器(IE10)中也逐渐淡化 VML，开始使用 SVG 替代 VML 进行矢量绘图。

SVG 指可缩放矢量图形(Scalable Vector Graphics)，也是基于 XML 的，支持 DOM 事件模型，具有良好的交互性。但在绘制对象数量较多时，DOM 操作会严重影响性能。并且它不支持绘制 png、jpg 等格式的图片，绘制的图形也无法导出成图片。

HTML5 Canvas [4]是 W3C 推出的新一代浏览器端绘图 API，它是基于像素级的绘图技术，支持图像像素操作，能够绘制路径和栅格图像，绘制后的对象可以转成 base64 编码，在前端保存为 png、jpg 图片。目前，主流浏览器都支持 Canvas，IE8 之前的版本也可通过 ecxcanvas.js 插件实现。许多客户端图表也是基于 Canvas 实现的[5] [6]，如一些插件 echarts、CanvasXPress、AwesomeChartJS、RGraph 等都是基于 Canvas 实现的。而且随着 HTML5 标准的规范化，HTML5 技术日趋成熟，使用 Canvas 进行前端矢量绘图也成为一种趋势。

Canvas 除了用于基本的图形展示外，还可用于 WebGIS 中空间要素的显示与表达[7]，实现地图数据的显示和交互，尤其是在要素量较大的情况，绘制效率远高于 SVG。对基本的点、线、面的表达，Canvas API 中提供了相关的接口，但对复杂地理要素诸如多线、多面、岛洞多边形等并没有提供绘图接口。而

实际的地理对象中并不只是单一的几何对象，有时会出现一些复杂的地理对形的嵌套，形成岛、洞等复杂对象。文中针对这类复杂地理要素的绘制给出解决方案，并对其性能进行分析。

2. HTML5 Canvas API 简介

Canvas 是 HTML5 中新增加的标签元素，用于客户端绘图，其绘图能力依赖于 JavaScript 脚本。它定义了一个矩形画布，通过画布元素的 context 对象，调用一系列的绘图方法，主要包括：moveTo、lineTo、stroke、fill、arc、quadraticCurveTo 等绘制路径的方法，以及 drawImage 绘制图像的方法，该方法可接受 img、canvas，video 对象等作为参数。除了绘制方法外，还包括一些坐标变换、样式设置、像素级别的操作方法。

Canvas 不仅支持绘制简单图形如线、矩形、圆、扇形等，还支持绘制复杂的曲线如二次贝塞尔曲线、三次贝塞尔曲线。除了支持矢量要素的绘制，还可用于栅格图像的绘制。目前，现代浏览器如 Internet Explorer 9、Firefox、Opera、Chrome 以及 Safari 都已支持<canvas>及其属性和方法。

3. 要素绘制的基本方法

3.1. 点要素的绘制

点要素的绘制实际上就是圆的绘制，调用 Canvas API [8]中的 arc 方法，以鼠标点击位置为圆心，设定半径以及填充样式，绘制圆即可，代码如下：

```
ctx=document.getElementById(“myCan”).getContext(“2d”);
ctx.arc(pt.x, pt.y, 4, 0, Math.PI * 2, true);
ctx.fill(); //绘制半径为 4 的圆
```

3.2. 线要素的绘制

绘制线要素就是不断地调用 lineTo 方法，首先，在鼠标单击下第一点时，通过 moveTo 绘制线段起点，之后每次单击的位置点作为线段的下一拐点，通过 lineTo 绘制两点间的线段，直至双击结束，完成该线要素的绘制。

```
ctx.moveTo(sxy0.x, sxy0.y);
for (var i = 1; i < ptArr.length; i++) {
   ctx.lineTo(ptArr[i].x, ptArr[i].y); }
   ctx.stroke();
//ctx.closePath();       //ctx.fill(); }
```

3.3. 面要素的绘制

Canvas API 中没有提供绘制多边形的方法，只有绘制矩形的 drawRect 方法，而多边形的绘制和绘制线要素基本一致，不同的是，线要素绘制结束后使用 stroke 描边，而面要素需使用 closePath 闭合和 fill 方法进行填充。

3.4. 岛洞要素的绘制

要素绘制过程中，采用的是双层画布，在临时画布上绘制的都是简单多边形，双击结束后，将临时画布上的要素绘制到要素层画布 featureCanvas 上，在该画布上的绘制会涉及到复杂的岛洞多边形。因此，在将临时画布上的要素向要素层上绘制时，还需要进一步判断当前绘制的多边形与要素层上已存在的多边形的位置关系。如果当前多边形与已有的多边形不存在包含关系，则作为简单多边形处理；否则的话，则会形成岛或者洞多边形如图 1，具体地，需要根据两个多边形的方向进行判断。如果两者的方向一致，

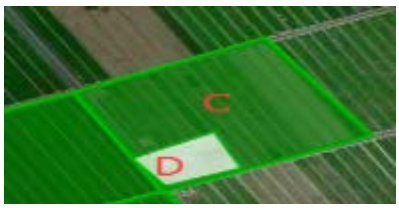

Figure 1. Hole polygon
图 1. 洞多边形

即同为逆时针或顺时针，则形成岛；否则形成洞多边形。

具体过程：在完成一个简单多边形的绘制后，将当前多边形与要素层上的已有多边形要素逐一进行多边形在多边形内的判断(代码见下)。如果存在包含关系，即为岛洞多边形，需要将两个多边形合并成一个新的要素，主要是将其中的坐标信息_lineRings 合并，重组成一个新的多边形要素，并从内存数组和要素层中移除原来要素，流程如图 2。

```
//判断多边形在多边形内
function isPolyInPoly(poly,outerPoly){
 for(var i=0;i<poly.length-1;i++ ){
    var pt1 = poly[i];
    var pt2 = poly[i+1];
    var line = [pt1,pt2];
    var isInpoly1 = pointInPoly(pt1,outerPoly);
    var isInpoly2 = pointInPoly(pt2,outerPoly);
     //两端点不都在多边形内
    if(!isInpoly1 || !isInpoly2)    return false;
    //判断每条边是否在多边形内
   var flag = isLineInPoly(line,outerPoly);
   if(!flag) return false;}
     return true;       }
```

要素的空间信息都存储在 lineRings 二维数组中，具体的组织方式如下：

1) 简单多边形的空间信息 lineRings：[ptArr],其中 ptArr 是多边形顶点坐标组成的数组。

2) 复杂多边形的空间信息 lineRings：[ptArr1,ptArr2,....ptArr i....]，ptArr1 是外部多边形的顶点坐标数组，ptArr2...ptArri 是内部的岛或洞的顶点坐标数组。

在将临时画布上的简单要素或合并后的复杂要素向要素层上绘制要素时，由于存在岛洞的情况，需要根据每类要素的特点采取不同的绘制策略，具体如表 1。

绘制岛洞多边形时，首先，需要判断这个多边形要素的各部分的包含关系，即多边形在多边形内的判断，确定其包含层级关系，然后通过 isClockWise 方法判断多边形的方向，是顺时针还是逆时针方向；比较存在包含关系的两多边形的方向是否一致，如果方向不一致，则形成一个洞，绘制时将这个内部多边形的填充色设为白色，形成一个空白区域。否则的话，这个内部多边形就是一个岛，绘制时只绘制边框。具体绘制过程如下：

```
// 岛洞多边形的绘制
var dir_outer = isClockWise(lineRings[0].points);
for(var i=1;i<lineRings.length;i++){
```

```
            var pointArr=lineRings[i].points;
            var dir_inner=isClockWise(pointArr);
if((dir_outer&&!dir_inner)||(!dir_outer&&dir_inner)){
holeOrIsland = 'hole'; //方向不一致
}else{     holeOrIsland = 'island';}
            drawPoly_inner(ctx,pointArr,holeOrIsland );
}
```

4. 要素交互式绘制与内存模型

4.1. 交互式绘制技术

Canvas 绘图是基于位图的，按照像素单位在画布上进行绘制，绘制的内容保存在 Canvas 对象中。Canvas 整体作为一个对象，支持 DOM 事件模型，但是 Canvas 中每个图形则无法直接通过 DOM 事件进行交互。因此，在对画布上的图形进行交互时，需要通过鼠标相关事件模拟交互过程。图 3 是以绘制多边形为例，作以说明绘制过程中的鼠标事件。

首先，声明两个相同宽高的画布图层，使用绝对定位叠放在一起，一个用于显示绘制好的图形(featureCanvas)，一个用于显示绘制过程中的图形的变化(overlayCanvas)。使用两个画布的目的在于提高绘制效率，在绘制过程中，鼠标每移动一次就需要重绘一次，如果在同一个画布上，则需要反复清空画

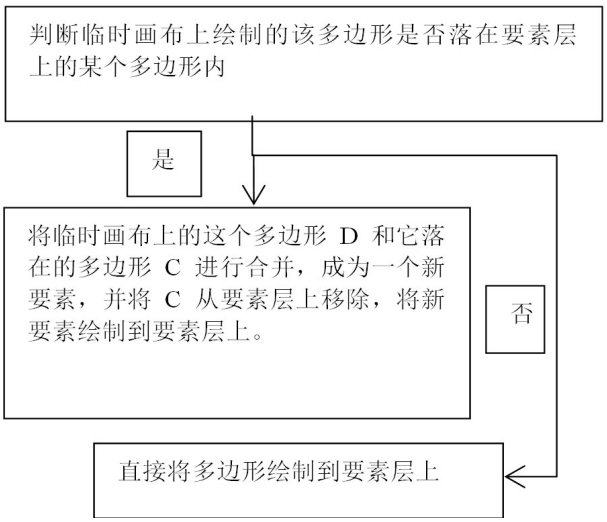

Figure 2. Workflow of drawing complex features
图 2. 绘制复杂要素的流程

Table 1. Strategies of drawing diverse features
表 1. 各种类型的要素绘制策略

类型	特点	绘制策略
简单多边形	单一	绘制各顶点之间的连线
岛多边形	在多边形内部嵌套多边形，特殊的多面	绘制外部的多边形(设置边框、填充样式等)，再绘制内部嵌套的多边形的边框
洞多边形	在多边形内部存在一块不属于该多边形对象的区域	先绘制外部的多边形(设置边框、填充样式等)，再绘制内部嵌套的多边形，填充样式设为白色，形成一个空白区域

布再重绘，图形数量较多时，重绘会降低性能。使用双层画布，将当前绘制的图形放在临时画布上，已经绘制好的放在另一画布上，在绘制过程中，只需要重绘临时画布上的内容即可，能够大大提升性能。

```
<div id="drawArea">
<canvas id="featureCanvas"></canvas>
<canvas id="overlayCanvas"></canvas>
</div>
```

4.2. 要素内存模型

在前后台的通讯过程中，主要通过实验室自主研发的 Geoserver2013 提供的要素服务接口进行要素的增删改查操作，除了要素服务外，同时它还提供了切片地图服务、动态图服务、文件服务、数据库服务等接口。在前端绘制的要素，使用 gFeature 对象存储，通过 Geoserver 提供的要素服务接口，将其提交至服务器进行保存；前端也可以通过属性查询和几何查询向服务器端请求要素集，以 Geojson 格式返回，在前端处理成 gFeature 格式，并绘制显示。

WebGIS 中的矢量要素通常包含空间和属性两大部分，空间数据主要描述地理对象的位置，属性描述该地理对象的特征信息。在前端存储要素数据时，将要素的两部分信息分别存储在 gFeature 对象中，空间信息主要存储在 shape 中，属性信息存储在 fields 中，具体如图 4。

5. 要素绘制性能分析

在要素绘制过程中不可避免地会涉及到绘制性能问题，影响其性能的因素众多，如要素的数量、要

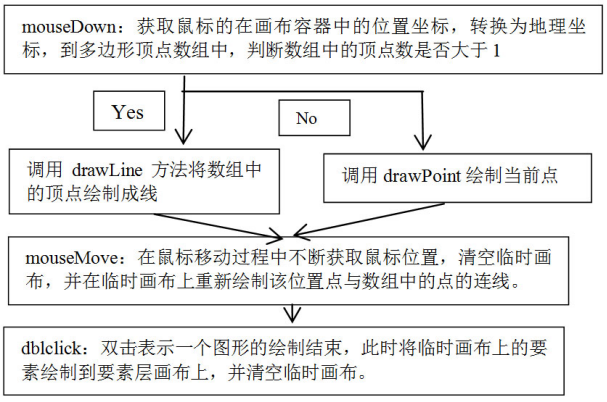

Figure 3. Mouse interaction event
图 3. 鼠标交互事件

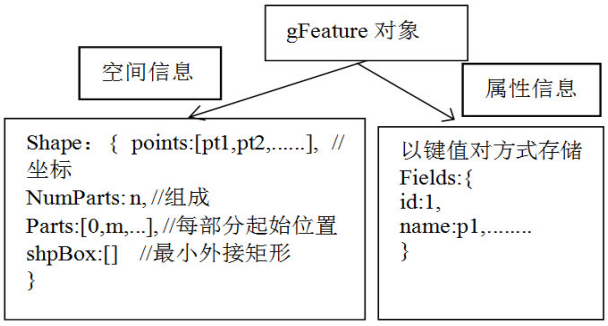

Figure 4. Memory model of features
图 4. 要素内存模型

素类型、绘制样式、浏览器类型以及硬件配置等。本文主从以下三个方面：不同数量、不同绘制样式、以及不同浏览器来对比 Canvas 绘制性能。

5.1. 基本性能分析

以绘制最简单的多边形为例，测试在同一浏览器中，以相同的绘制样式绘制不同数量的多边形耗费的时间。先声明一个 1000 × 600 的画布，利用随机函数在画布内生成含有三个结点的多边形，对比绘制不同数量的相同结点数的多边形的耗时。在测试过程中，多边形的绘制样式统一设置了填充色、边框色、线宽、边角样式等，测试使用的浏览器是 FireFox33。由于单次测试存在一定偶然性，因此，绘制耗时的计算采用的是取 100 次测试结果的平均值，最后，得出不同绘制规模下的平均绘制时间(如表 2)，绘制时间和绘制数量成线性关系。

要素绘制性能不仅与要素的数目有关，还与每个要素的复杂程度有关，相同数目的要素，要素结构越复杂，即构成每个要素的结点数目越多，绘制时就越耗时。在测试实验中，分别对比同等数量下，三结点和五结点构成的多边形的绘制耗时。结果发现，两者的绘制时间存在一个倍数关系，同等数量下，绘制五边形的时间是三角形的 2.26 倍左右。

5.2. 绘制样式对性能的影响

进行要素绘制时，不仅可以使用颜色填充，也可以使用图像填充。但是二者的填充效率如何，需要进一步测试。分别比较使用纯色填充和图片填充的绘制效率，其中，图片大小为 3 KB，图片填充样式的设置如下：

```
var fillImg = new Image();
fillImg .src = 'apple.png';
var pstyle=ctx.createPattern(fillImg ,'repeat');
ctx.fillStyle=pstyle;
```

图 5 是对比绘制不同规模数量的三角形，分别使用两种填充方式所需要的时间，从表中可看出，图像填充比较耗时，约是纯色填充耗时的 4 倍左右。

5.3. 不同浏览器中绘制效率对比

不同浏览器中绘制效率对比 Canvas 是由浏览器解析并处理的，因此，其绘图的性能与浏览器平台有

Table 2. Time consuming of drawing different numbers of features
表 2. 绘制不同数量要素的耗时对比

数量(个)	平均耗时(毫秒)
100	5.87
500	28.74
1000	58.64
2000	117.82
5000	286.4
8000	461.8
10,000	574.8
15,000	887.9
20,000	1158.6

直接关系，与客户端的硬件设备性能也有间接关系。不同浏览器对 JavaScript 的解析与执行速度会有所差异，这也会影响 Canvas 的绘制性能。部分浏览器开发商实现了 Canvas GPU 硬件加速，会大大加快了绘制速度。

下面的测试实验是在启用了浏览器的 GPU 加速功能后，对比三类浏览器中的绘制效率。分别在 Chrome43、FireFox33、IE11 三个浏览器中，绘制同等数量的三角形，都采用纯色填充的方式，对比要素的绘制效率，测试结果如下所示，从图 6 中可看出，Chrome 中 JavaScript 的执行效率是最高的，其次是 FireFox，IE 最慢。各浏览器 JS 的执行速度还存在一定的数量关系，Chrome 中 JS 的执行速度是 Firefox 的 4~5 倍，是 IE 的 6~8 倍。

6. 要素绘制性能分析

6.1. 性能分析

在绘制过程中，要素数量、结点数、填充样式等都会影响绘制性能，而具体地这些因素是如何影响性能的，还需要进一步探讨。要找出绘制过程中时间消耗在哪些操作中，可以借助 Chrome 提供的 Timeline 工具，它会将 web 应用过程中各部分的耗时概况显示出来，主要包括 DOM 事件的处理、页面布局渲染、向屏幕绘制元素三个层面上的耗时数据，帮助找出耗时操作。

分别对比绘制 5000 个要素时，启动 Chrome 的 Timeline 模块，记录下绘制过程中的具体耗时概况，结果如图 7。其中，黄色部分代表分析和执行 JavaScript 所用的时间，紫色表示计算元素样式和布局所用

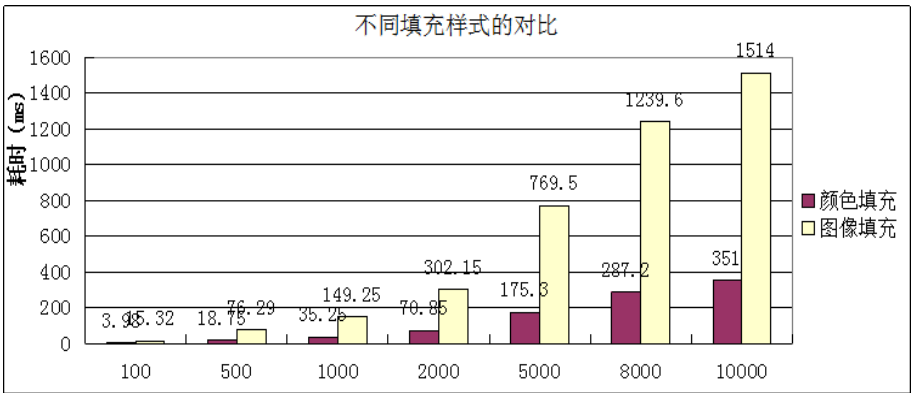

Figure 5. Time consuming comparison in different styles
图 5. 不同绘制样式的耗时对比

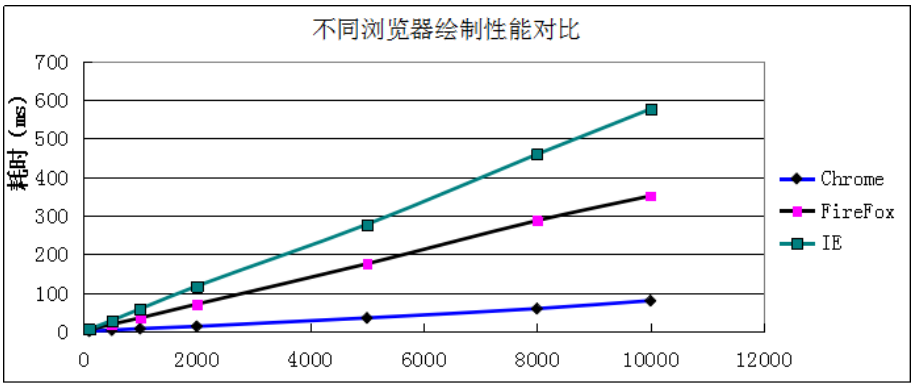

Figure 6. Canvas rendering time consuming in different browsers
图 6. 不同浏览器中 Canvas 绘制耗时对比

时间，绿色是绘制屏幕所用时间，在三者的耗时中，脚本运算耗时最多，占比最大，而且随着要素数量的增加，脚本运算耗时几乎成倍增加。由此可见，绘制过程中，脚本运算对性能影响最大。

Profile 模块提供了对脚本中各函数执行时间的统计，可以帮助网站找出哪些函数导致的脚本阻塞。利用该模块下的 JavaScript CPU Profile 来记录测试页面中的 JavaScript 各函数的执行时间，从中找出最耗时的操作。在性能优化时，考虑最大程度的减少这些耗时操作。表 3 是分别绘制 1000 个和 5000 个要素时，脚本各函数运行时间的记录。其中，当绘制 5000 个要素时，描边操作耗费了 10.9 ms，填充、画线、移动节点分别消耗 4 ms 左右。在绘制过程中，绘制时间主要消耗在画线、描边、填充等操作上。

通过上述分析可以看出，stroke、fill、lineTo、moveTo 等函数的执行耗时占比较大，这也说明了要素数量、结点数以及填充样式会对要素的绘制时间产生影响。在随着要素数量增多时，这几个耗时函数的调用次数都会增多，导致最终的耗时增加；同等数量的要素，要素结点数越多，会使 moveTo、lineTo 函数的调用次数增加，从而增加绘制时间；填充样式的复杂程度会影响 fill 函数的执行时间，最终影响绘制时间。因此，绘制性能的提升主要体现在对这些操作的优化，如何最大程度地较少耗时操作。

6.2. 优化方法

6.2.1. 使用双层画布

在绘制过程中，每进行一次鼠标操作，就需要执行一次重绘，以保证最新的绘图状态。当画布比较大，且画布上的要素比较多时，重绘操作会阻塞线程，出现卡的现象，影响用户体验。采取的解决办法是使用双层画布，将已绘制好的要素放置在一个目标画布上，而正在绘制或修改的要素放在一个覆盖层画布上，当绘制完成或修改完成后，一般是在鼠标触发 MouseUp 时，再将覆盖层画布上的要素绘制到目标画布上，更新目标画布的视图状态。

6.2.2. 预渲染

使用离屏 Canvas，声明一个不可见的 Canvas 画布，同目标画布同等大小，预先将要素绘制到离屏画

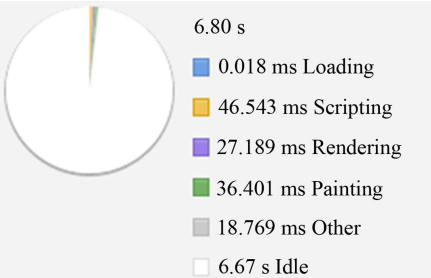

Figure 7. Time consuming of drawing 5000 features
图 7. 绘制 5000 个要素的耗时

Table 3. Executing time of drawing functions
表 3. 绘制函数的执行时间

函数	1000 个要素	5000 个要素
beginPath	-	0.2 ms
moveTo	-	4.4 ms
lineTo	1.1 ms	3.8 ms
stroke	2.1 ms	10.9 ms
fill	1.1 ms	4.0 ms

布上，在需要的时候，通过 drawImage 方法把离屏画布中的内容绘制到目标画布中。

```
function preRender(){     //将离屏 mCanvas 中的内容绘制到目标画布
 ctx.drawImage(mCavans,0,0,mWid,mHei);
 }
```

6.2.3. 请求视窗范围内的地理要素

在绘制地理要素时，地理范围可能会很大，可以先根据地理坐标与屏幕坐标的换算公式，确定当前画布视窗范围内能够显示的地理要素。在执行修改要素、绘制要素时，只涉及到当前视窗范围内的要素状态变化，减少重绘操作的性能损耗。

6.2.4. 函数节流

当画布尺寸发生变化时，视图范围随之变化，视窗范围内的要素需要重新计算，并执行要素的重绘操作。但如果画布尺寸变化事件不断触发，如在改变浏览器窗口大小时，则会频繁执行重绘操作，带来严重的性能损耗。为了避免这种情况的发生，可以采用函数节流的方式，让重绘操作只在超过指定的时间间隔后才执行[9]，如指定在 500 毫秒后再调用画布尺寸变化的处理函数，避免画布尺寸频繁变化时，连续触发响应函数。

```
function throttle(method,context){
clearTimeout(method.timerId);
method.timerId=setTimeout(function(){
        method.call(context);},500);     }
```

7. 要素绘制性能分析

在《上海市农业布局规划动态管理系统》中(图 8)，依托上海市高清遥感影像，对上海全市农用地数据进行在线采集[10]。其中，农用地地块数据主要采用了 Canvas 进行绘制，并通过 fillText 方法绘制地块类型的文字。在性能方面，采用了双层画布绘制、控制视窗范围内的要素显示以及函数节流等进行优化，缩放平移操作中，要素响应基本在 1 秒内；在并发访问上，可以满足不少于 20 个用户的同时在线数据编辑。

Figure 8. The planning system of Shanghai agriculture
图 8. 上海市农业布局规划系统

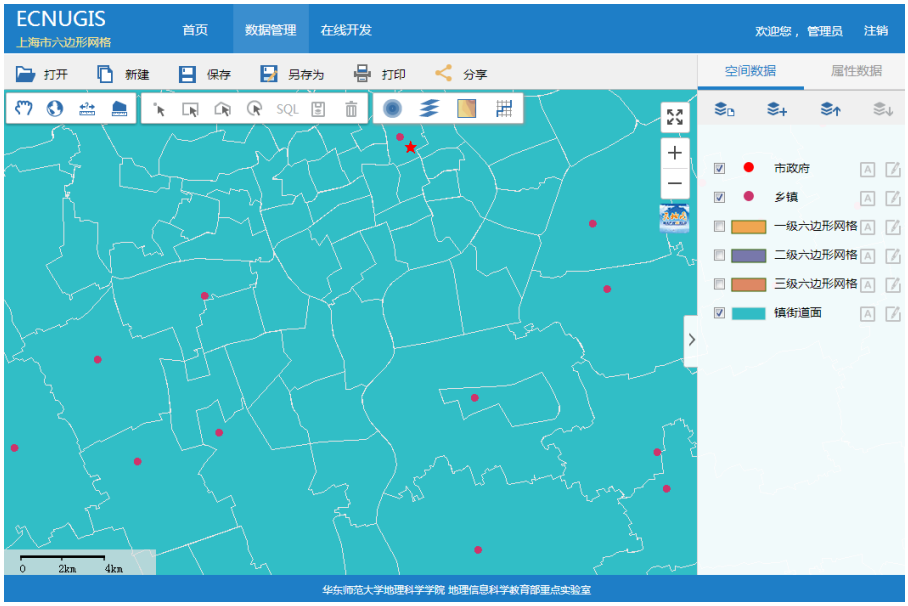

Figure 9. ECNUGIS platform
图 9. ECNUGIS 平台

实验室自主研发的 ECNUGIS [11]平台(图 9)是一个开放的数据管理与分享平台，提供了在线创建要素集，进行要素的绘制与编辑以及数据分享的功能。其中，要素的绘制与编辑使用的也是 Canvas，用户可以自主创建要素集，进行在线矢量化，并通过实验室研发的 Geoserver 服务器将要素集保存在后台，通过要素服务和动态图服务供用户后续调用。

8. 结论

文中对前端使用 Canvas 进行地理要素的绘制进行了探讨，针对 Canvas 交互做了鼠标事件模型，实现了复杂岛洞多边形要素的绘制，为 WebGIS 中复杂地理要素的可视化显示以及要素编辑提供了基础。在性能方面，主要从要素数量、要素结点数、绘制样式、浏览器类型四个方面进行了要素绘制耗时的对比试验，并进一步提供了在 WebGIS 中绘图时的几种优化方法。

HTML5 Canvas 客户端绘图技术越来越成为一种趋势，尤其是针对这种数据量大、比较零碎的地理对象的绘制与展示，而且随着 HTML5 规范的标准化，Canvas 的浏览器支持性会更好。此外，Canvas 元素除了支持 2D [12]绘图，还支持 3D 绘制，这主要赖于浏览器的支持，随着浏览器的不断迭代更新，未来直接通过 Canvas 在浏览器中实现 3D 绘图也变得十分有可能。

参考文献 (References)

[1] 李治洪. WebGIS 原理与实践[M]. 北京: 高等教育出版社, 2011: 269-292.

[2] 廖明, 潘媛芳. WebGIS 矢量地图绘制方法的性能分析与比较 WebGIS 的技术方案[C]. 《测绘通报》测绘科学前沿技术论坛摘要集, 2008.

[3] 蔡锦辉, 陆明典. 用 VML 构建基于 WebGIS 的交通气象服务系统[J]. 气象研究与应用, 2014, 35(2): 60-62.

[4] Fulton, S. HTML5 Canvas 开发详解[M]. 任旻, 王洋, 等, 译. 北京: 人民邮电出版, 2014: 17-165.

[5] 龚丽. HTML5 中的 Canvas 绘图研究[J]. 软件导刊, 2014, 13(4): 151-153.

[6] 谷伟. 基于 HTML5 Canvas 的客户端图表技术研究[J]. 信息技术, 2013(9): 107-110.

[7] 徐莎, 杨帆, 徐昌庆. 基于 HTML5 的 WebGIS 的研究与应用[J]. 信息技术, 2012(4): 149-151.

[8] 陆凌牛. HTML5 与 CSS3 权威指南[M]. 北京: 机械工业出版社, 2013: 49-205.

[9] Zakas, N.C. (2010) High Performance JavaScript. O'Reilly Media, Sebastopol, 61-80.

[10] 丁扬. 上海市农用地数据在线采集系统研究[D]: [硕士学位论文]. 上海: 华东师范大学, 2015.

[11] 地理大数据协同创新平台[DB/OL]. https://ccgis.cn/mapb/, 2015-7.

[12] 朱文. 基于 HTML5 的 2D 动画的设计与实现[D]: [硕士学位论文]. 广州: 中山大学, 2013.