计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2018,27(10):196−201 [doi: 10.15888/j.cnki.csa.006567]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于 Unity 的数字化车间改进资源动态调度算法①

朴美燕[1,2], 胡　毅[2,3], 叶迎萍[1,2]

[1](中国科学院大学, 北京 100049)
[2](中国科学院 沈阳计算技术研究所 高档数控国家工程研究中心, 沈阳 110168)
[3](沈阳高精数控技术有限公司, 沈阳 110168)
通讯作者: 朴美燕, E-mail: 2252135602@qq.com

摘　要: 针对基于 Unity 引擎的车间仿真系统加载过程中内存占用大, 导致系统在运行时存在卡顿、不流畅的现象, 从模型资源加载角度设计一种资源动态调度算法. 首先通过四叉树算法将场景递归的分割成多个叶子节点进行存储, 然后结合资源动态调度算法以摄像机位置为中心对周边节点的资源进行预设实例化和预设销毁完成内存的管理. 最后根据 Unity 的内存管理机制, 设计并实现了对场景资源的动态调度和内存优化, 实验结果表明资源动态管理算法有效控制某时刻内存中加载的数据量相对稳定降低了 IO 总量, 避免了漫游移动时造成的内存颠簸, 使系统的运行更加流畅.

关键词: Unity; 数字化车间系统; 性能优化; 资源动态调度算法; 四叉树算法

引用格式: 朴美燕,胡毅,叶迎萍.基于 Unity 的数字化车间改进资源动态调度算法.计算机系统应用,2018,27(10):196–201. http://www.c-s-a.org.cn/1003-3254/6567.html

Improved Dynamic Resource Scheduling Algorithm on Unity-Based Digital Workshop

PIAO Mei-Yan[1,2], HU Yi[2,3], YE Ying-Ping[1,2]

[1](University of Chinese Academy of Sciences, Beijing 100049, China)
[2](National Engineering Research Center for High-End CNC, Shenyang Institute of Computing Technology, Chinese Academy of Sciences, Shenyang 110168, China)
[3](Shenyang Golding NC Technology Corporation Ltd., Shenyang 110168, China)

Abstract: Unity engine-based manufacturing simulation system occupies large memory when loading, resulting that the system is not fluent at run time and cannot run smoothly. In response to this phenomenon, we design a resource dynamic scheduling algorithm from the perspective of model resource loading. Firstly, the scene is recursively divided into a plurality of leaf nodes for storing through a quad-tree algorithm. Then, combined with the resource dynamic scheduling algorithm, taking the camera position as the center, the resources of surrounding nodes are defaulted instance and destruction of the prefab to complete the memory management. Finally, according to Unity's memory management mechanism, the dynamic scheduling and memory optimization of scene resources are designed and implemented.

Key words: Unity; digital workshop system; performance optimization; dynamic resource scheduling algorithm; quad-tree algorithm

① 基金项目: "高档数控机床与基础制造装备"国家科技重大专项 (2017ZX04011004)
Foundation item: National Science and Technology Major Project of China "High-grade NC Machine Tools and Basic Manufacturing Equipment" (2017ZX04011004)
收稿时间: 2018-02-25; 修改时间: 2018-03-19; 采用时间: 2018-03-21; csa 在线出版时间: 2018-09-28

1 引言

数字化车间[1]是智能工厂的关键组成部分, 如德国“工业 4.0”、美国“工业互联网”以及“中国制造 2025”, 均提出通过信息网络与工业生产系统的充分融合, 打造面向 CPMS 的数字化车间, 以实现价值链上企业间的横向集成, 网络化制造系统的纵向集成, 以及端对端的工程数字化集成, 改变当前的工业生产与服务模式, 全面提升制造业的整体效率.

随着数字化车间仿真系统的规模越来越庞大, 在 Unity 场景中进行漫游时需要加载大量的数控机床模型, 加载过程中内存占用大, 导致系统在运行时存在卡顿、不流畅的现象, 这种用户体验差的系统是不能满足用户的基本需求的. 目前的解决方法主要是通过升级硬件设备来改善效果, 或采用内置技术来实现对大场景的优化等方法, 通过研究发现相关的改进方法有如下几种, 第一种是使用遮挡剔除算法[2]来降低场景中需要渲染模型的数量, 从而提高实时渲染的速度, 优点是简单高效确实可以提高渲染速度, 但缺点在于增加了内存的占用空间, 在复杂的大型场景中无法真正提升性能, 第二种是通过数据库动态存储分块数据并采用动态 LOD 算法将数据分块调用[3,4], 降低了内存占用, 从而提高渲染速度, 但该算法复杂度较高需要进行大量的 CPU 计算, 上述两种方法虽然在某种程度上提高了渲染的速度, 但仍需要消耗较多的 CPU 时间和内存空间且没有充分利用 Unity 引擎的特性.

故本文根据 Unity 引擎中内存管理机制[5]的特性, 从数控机床模型载入的角度设计了一种资源动态调度算法, 通过内存优化和动态调度降低内存占用和 IO 总量的同时降低 CPU 的占用, 从而实现整体系统的性能优化. 在进行资源动态调度算法前, 首先通过四叉树算法将大规模的场景进行分层和分割成场景块, 并对其行、列值进行标记区分. 然后在漫游过程中, 实时计算摄像机投影到的场景地面所在的行、列值更新需要调入的模型资源列表, 动态的调入模型资源, 并卸载需要调出的模型来更新资源列表, 最后通过实验对比应用资源动态调度算法的前后的计算机性能的情况, 即计算机内存占用情况和载入用时及宏观上系统的流畅度来验证算法的高效性.

2 Unity 引擎中的内存管理机制

2.1 Unity 资源管理方式

Unity 引擎中的资源文件和场景文件有所区别, 资源文件因为是程序要依赖的文件主要存放在硬盘里, 而场景文件是在程序运行时动态的加载到内存. 其中预设是联系两者的重要概念, 预设就是场景中模型对象及组件的集合, 具有继承和重载等属性, 从而可以做到资源的重复利用. 预设作为一种资源集合的引用, 可以在场景中通过创建或者销毁预设来实现对资源的动态管理, 而车间内的机床也是资源文件的一种, 在场景需要时加载指定的机床模型到至场景, 同时卸载不必要的机床模型.

2.2 Unity 内存及管理方式

在 Unity 中提供了多种创建和销毁物体对象 (含网格、贴图、材质等信息) 的方式, 不同操作方式导致内存的占用过程也不一致. Unity 中内存结构如图 1 所示.

Unity 给开发者提供了两个加载预的方式, 第一种是通过调用 Resources.Load 方法, 另一种是调用 AssetBundle.Load 方法. 这两种方法并没有实质区别, Resources.Load 是从缺省包 Load 之后进行实例化操作, 它只能加载存放在 Resources 文件夹目录下的本地资源, 此文件夹下的资源不管会不会被加载到场景中, 并被打包到本地, 增大了场景文件的大小. AssetBundle.Load 方法进行实例化操作时, 需要自己创建文件、指定路径和来源后运行时动态加载再进行实例化, 开发者可以将任何物体封装成 AssetBundle 文件后放到硬盘或者网络, 可以随时下载使用. 场景资源读取到本地内存时是以内存镜像数据块的形式存在, 这时还没有 Assets 的概念, 只有调用 AssetBundle.Load方法, 才能让内存镜像数据块中的数据资源通过复制并反序列化操作后成为场景内可用的 Asset 对象, Load 过程中就会直接加载预设全部依赖资源, 实例化过程只是进行了克隆操作. 由图 1 可以看出 Destroy()函数销毁的只是实例化过程中对资源的引用或复制, 并没有释放内存中的文理和材质等资源. UnloadAsset(obj)释放区域中指定的资源. UnloadUnusedAssets() 会卸载当前所有没有被占用的资源.

通过上述分析可知在实现大规模场景的漫游时采用 AssetBundle.Load 加载方式能同时加载预设的纹理、材质信息, 此方法能够避免卡顿现象. 卸载时使用 Destroy() 函数销毁物体对象后, 在场景块卸载完成后调用 UnloadUnusedAssets() 函数将未被占用的纹理、贴图资源彻底卸载以实现内存回收. 这样在场景漫游过程中, 机床模型资源的计算机占用率始终维持在较低水平.

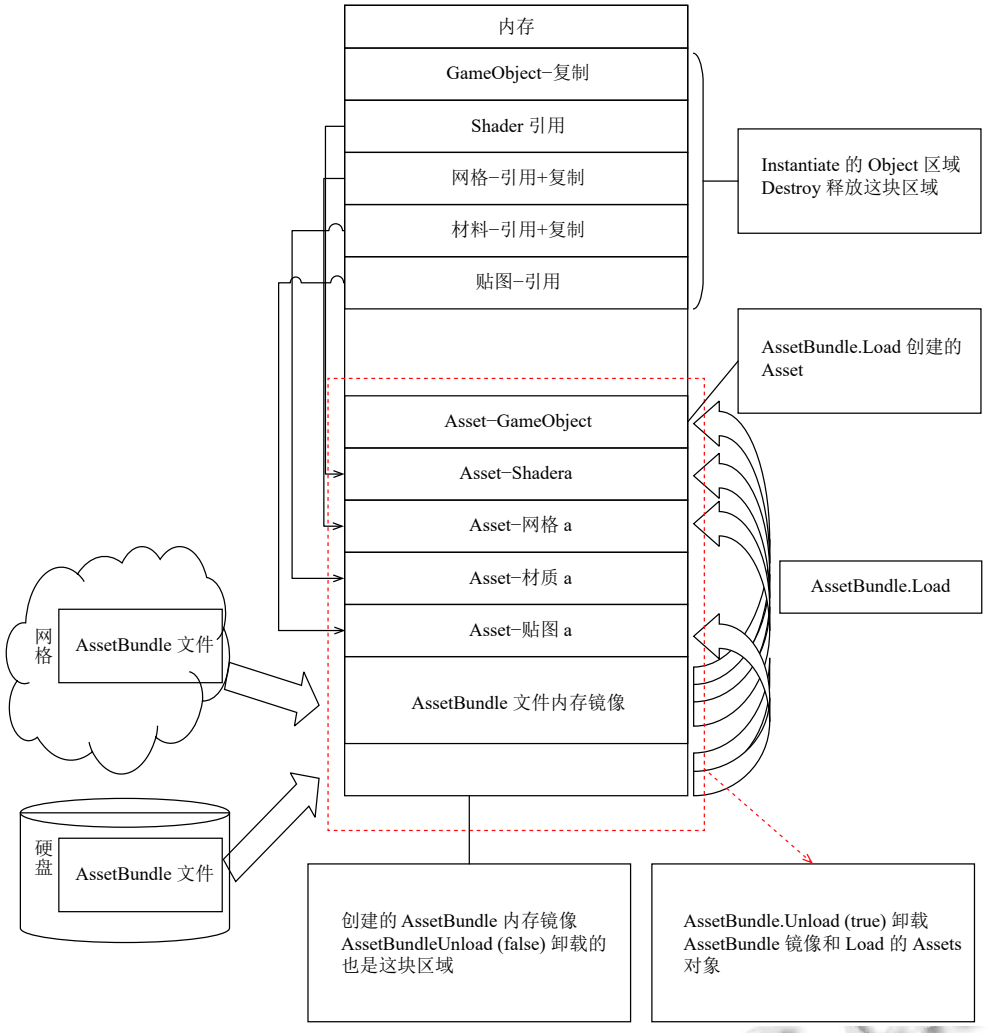

图 1 Unity 引擎内存结构

3 资源动态调度算法

3.1 四叉树算法原理及应用

在进行资源动态调度算法前, 首先需要将大规模的场景进行分层和分割成场景块, 目前的三维绘制中分层、分块算法有几何多分辨率 (GeoMipmapping) 算法、实时优化自适应网格 (Real-time Optimally Adaptive Meshes, ROAM) 算法、四叉树算法等. 其中几何多分辨率算法在快速读取大规模三维场景数据时有很好的优势, 但是处理场景时容易出现裂缝, 而实时优化自适应网格算法虽然可以解决出现的裂缝问题, 但是需要复杂的编程过程. 而使用四叉树算法不仅对场景可以准确、快速地进行分层、分割, 通过控制层次间的精度差不超过 1, 四叉树算法可以很好地解决场景内出现的裂缝. 故本文选择四叉树算法来对场景进行分层、分块处理.

四叉树算法的基本思想是把地理空间递归的划分为不同层次的树结构, 它将已知范围的空间分成四个相等的子空间, 如此递归下去直至树的层次达到一定深度或者满足某种要求后停止分割. 在应用资源动态调度算法前, 需要将场景进行分层、分割成场景块, 通过四叉树算法[6]可以做到对场景空间的分层及分割并标记场景块的行、列值. 目前四叉树算法大部分都应用在了对地形漫游的处理上, 该算法是采用了分层分块的思想, 对地形块内数据按照分辨率的大小分层存储. 本文将四叉树算法应用在车间仿真系统中进行分层和分块处理, 并对分割成块的场景块标记其坐标值即行、列值.

将四叉树算法应用于大规模车间仿真系统时, 首

先将车间仿真系统场景进行分层、分割后, 可以对场景块进行行、列值的标记, 而 Unity 引擎使用行、列值信息的二维数组的索引来存储机床模型资源名称, 通过摄像机位置在 X、Z 平面的投影判断当前场景地面块的范围, 然后通过坐标值的比较后对设备模型资源进行动态调度. 以图 2 为例, 其中 (2,2) 场景块为摄像机初始所在位置, 其周边的 8 块浅色区域在初始化时已经加载到内存, 在进行漫游时加载摄像机移动方向前方的场景块, 卸载超出摄像机周边范围的场景块.

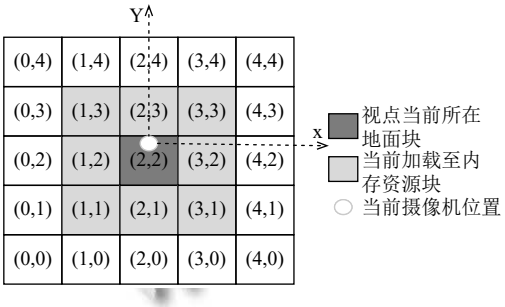

图 2 摄像机位置在地面坐标系映射关系

若摄像机向右移动一块进入 (3,2) 场景块, (4,1)、(4,2)、(4,3) 场景块将进入摄像机视角范围内, 即加载这三个走近的场景块, 而卸载远离视角的 (1,1)、(1,2)、(1,3) 场景块. 加载和卸载出去的场景块数保持一致, 从而维持计算机的内存也处于稳定状态.

若以地面中心为为世界坐标系[7]中心, 则图 2 中的摄像机位置映射关系可以如下式 (1) 表示.

$$\begin{cases} CamNum\_X = (CamPos.x + PlaneSize/2) \\ \quad /PlaneSize + N/2; (0 CamPos.x) \\ CamNum\_X = (CamPos.x - TerrainSize/2) \\ \quad /PlaneSize + N/2; (CamPos.x < 0) \end{cases} \tag{1}$$

其中, *CamNum_X* 为摄像机映射至平面的行值; *CamPos.x* 为摄像机在世界坐标系中 X 轴分量; *PlaneSize* 为平面大小; *N* 为平面块数组维度. 以此类推摄像机在平面的 Z 方向映射计算方法也可以算出来. 通过上述介绍的四叉树算法可以得知, 四叉树算法在漫游场景时可以有效计算出摄像机映射在平面上的坐标值, 然后通过计算出的坐标动态的加载或卸载资源, 资源动态调度算法的实现是在四叉树算法处理的前提下进行. 在大规模车间仿真系统的模型资源进行动态加载, 只加载摄像机周围的模型资源, 能有效降低场景的 CPU 和 GPU 消耗.

3.2 资源动态调度算法的设计与实现

通过前面对 Unity 引擎内存管理机制的叙述和分析确定了适合本系统的预设的加载方式[8,9], 用四叉树算法分割成场景块并标记行、列值后, 用存有行、列值信息的二维数组的索引来存储机床模型资源名称, 通过摄像机在平面的投影确定当前场景块的范围, 然后通过坐标值的比较后对设备模型资源进行动态调度.

结合上述思想设计了资源动态调度算法, 该算法是以摄像机在地面的投影位置为中心, 通过对中心节点周围资源进行预设实例化和预设销毁完成内存的管理.

算法的具体执行流程如下:

1) 程序初始化时根据摄像机初始位置加载其周围九个场景块, 并将场景块需要载入的机床模型资源名称存入当前资源列表 CurrentModelList(保存当前已加载模型名称).

2) 程序进入帧刷新, 获取当前摄像机行、列值计算其映射场景块, 若摄像机位置没有变化则不处理, 否则进行资源动态调度.

3) 根据变化后的摄像机所映射的场景块行、列值, 计算当前摄像机周边模型资源名称, 将其存入更新后资源列表 RefreshModelList(保存帧更新后模型名称).

4) 由帧更新前后的资源列表的差集, 计算需要加载和卸载的机床模型.

5) 采用 AssetBundle.Load() 方法加载更新后的模型; 采用 Destroy() 函数销毁模型, 并在完成后调用 UnloadUnusedAssets() 回收内存.

6) 完成调度后, 将 RefreshModelList 更新列表赋值给当前资源列表 CurrentModelList.

7) 判断程序是否结束, 若结束则退出程序, 否则进入下一帧循环.

程序流程如图 3 所示.

结合图 2 知, 列表 RefreshModelList 与 CurrentModelList 的差集代表需要加载的新的模型资源, 而 CurrentModelList 与 RefreshModelList 的差集则代表需要卸载的模型. 完成模型资源动态加载卸载后, 执行 UnloadUnusedAssets() 函数释放模型资源的纹理、材质等内存资源, 并更新当前资源列表, 由此完成整个场景资源的调度过程, 程序进入下一帧循环. 在整个帧循环过程中, 内存中的数据量维持不变, 降低了 IO 总量.

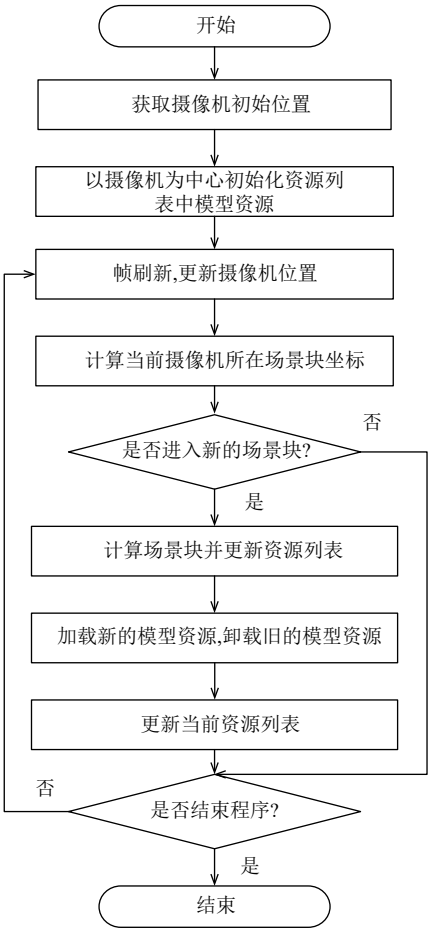

图 3 资源动态调度算法流程图

4 实验验证及性能分析

4.1 资源动态调度算法的应用验证

如图 4 场景中摆放着 1000 个不同形状的模型来模拟车间系统中的设备模型, 通过四叉树算法分割成均匀的场景块, 并模拟出摄像机移动过程中应用资源动态调度算法的过程如图 5 所示.

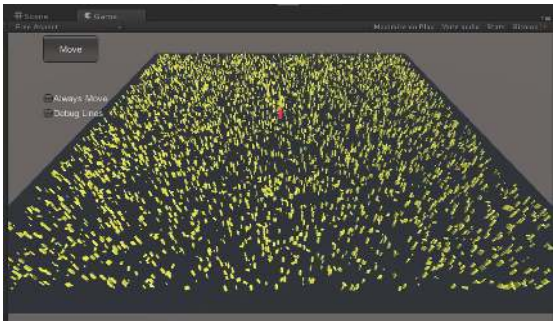

图 4 摆放 1000 个模型的场景

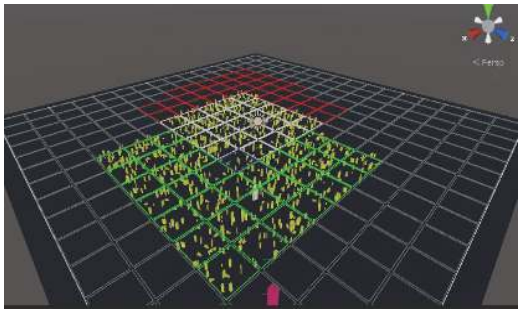

图 5 资源动态调度算法的应用效果

可以看出应用资源动态调度算法后有效的避免了在漫游过程中摄像机的来回移动时对象的反复加载导致的内存颠簸[10].

4.2 资源动态调度算法的性能分析

针对车间系统 (图 6) 进行资源动态调度算法后在性能上有没有改变, 需要通过量化的属性去比较数据, 本次实验主要从内存占用情况、帧率方面进行性能的分析, 通过比较当系统一次性全部加载与采用资源动态调度算法加载系统资源的方式后计算机资源的使用情况来分析性能[11]. 如表 1 是两种加载方式的性能参数.

图 6 车间仿真系统中进行漫游

表 1 两种加载方式性能指标对比

指标	一次性加载	动态调度
FPS(帧/秒)	31.3	72.6
渲染耗时 (ms)	30	14
总内存占用 (MB)	323	130
纹理内存占用 (MB)	79.6	16.6
网格内存占用	4.9 MB	316.5 KB

由表 1 和图 7 可见, 采用资源动态调度算法后程序中的纹理内存占用、网格内存占用以及总内存占用均明显降低, 帧率也提升了一倍. 从 Unity 引擎自带的性能分析器[12]也可以看出应用算法后内存也始终维持在了稳定的水平.

图 7 CPU 和 GPU 占用情况

5 结论

本文针对当一次性加载大型车间仿真系统时内存占用量大、运行卡顿的问题, 详细分析了 Unity 的内存管理机制, 分析并设计出适合本系统的资源动态调度算法. 对场景用四叉树算法划分成具有行、列值的均匀的场景块后, 根据摄像机运动的轨迹不断更新当前视点周围的资源列表, 从而实现了对模型资源的动态调度. 在大场景漫游过程能够保持流畅浏览同时降低计算机性能消耗, 对于控制仿真对象的资源占用和优化有重要作用. 实验证明该算法能维持系统运行时内存消耗在较低的水平, 同时显著提升帧率验证了算法的可行性.

参考文献

1 燕继明. 数字化车间生产现场实时监控系统的设计与实现[硕士学位论文]. 成都: 西南交通大学, 2014.
2 张典华, 陈一民. 基于 Unity3D 的多平台虚拟校园设计与实现. 计算机技术与发展, 2014, 24(2): 127–130, 135.
3 耿肖, 韩志伟, 廖峰. 基于 Unity 引擎的地形块动态调度算法研究. 计算机与数字工程, 2016, (9): 1629–1634. [doi: 10.3969/j.issn.1672-9722.2016.09.001]
4 姜昱明, 胡曼丽. 地形漫游中数据块调度算法研究. 计算机工程与设计, 2007, 28(15): 3743–3745, 3786. [doi: 10.3969/j.issn.1000-7024.2007.15.063]
5 刘克, 李建方. 基于 Unity3D 的内存优化的研究. 中国传媒大学学报 (自然科学版), 2016, 23(5): 56–61. [doi: 10.3969/j.issn.1673-4793.2016.05.010]
6 方沁. 基于 Unity 和 3dmax 的虚拟实验室三维建模设计与实现[硕士学位论文]. 北京: 北京邮电大学, 2015.
7 姜康, 柯榕, 赵小勇, 等. 数字化车间虚拟监控系统研究. 航空制造技术, 2016, (20): 97–100, 104.
8 王履华, 孙在宏, 吴长彬, 等. 基于虚拟现实和物联网的水闸自动化监控系统. 地理信息世界, 2013, (4): 51–55. [doi: 10.3969/j.issn.1672-1586.2013.04.010]
9 冯宁, 沈雪微, 何建, 等. 基于物联网和虚拟现实的灭火救援及训练系统. 物联网技术, 2015, 5(2): 56–59. [doi: 10.3969/j.issn.2095-1302.2015.02.025]
10 侯志霞, 邹方, 吕瑞强, 等. 信息物理融合系统及其在航空制造业应用展望. 航空制造技术, 2014, (21): 47–49, 53. [doi: 10.3969/j.issn.1671-833X.2014.21.007]
11 任状. 数字化车间远程监控关键技术及实现[硕士学位论文]. 西安: 西安电子科技大学, 2011.
12 李卉. 数据可视化技术在物联网监控系统中的研究与应用[硕士学位论文]. 北京: 北京邮电大学, 2013.