基于 WebGL 的 3D 动画实时播放系统①

杨润斌[1], 梁文章[2], 陈 姝[1]

[1](湘潭大学 信息工程学院, 湘潭 411105)
[2](广西卡斯特动漫有限公司, 南宁 530003)

摘 要: 由于受到网络带宽及三维动画数据量过大等因素的影响, 三维动画如果要在网络上一次性传输往往需要花费比较长的时间. 这不仅影响了动画播放的实时性, 同时, 传输过程中数据的不完整也会造成动画播放不流畅. 通过对基于 WebGL 的三维动画实时播放关键技术的研究, 提出模型文件预加载、文件压缩传输等方法, 在有限带宽的条件下将三维动画内容实时传输到用户端, 利用 WebGL 技术实现三维模型的创建和渲染, 并在此基础上实现三维动画实时播放, 使用户能够直接通过浏览器进行观看, 或参与到场景交互中.

关键词: 三维动画; WebGL; 实时渲染; 文件预加载; 文件压缩传输

3D Animation Real-Time Playing Based on WebGL

YANG Run-Bin[1], LIANG Wen-Zhang[2], CHEN Shu[1]

[1](College of Information Engineering, Xiangtan University, Xiangtan 411105, China)
[2](Guangxi Cast Animation Co. Ltd., Nanning 530003, China)

Abstract: Due to the limited network bandwidth and the fact that the 3D animation data are very huge, it will take a long time to transmit the data of 3D animation through network on one time. As a result, it can't be smoothly displayed in real-time as the transmitted 3D animation data are incomplete. The key technology of WebGL based 3D animation is presented, such as model files preload and transmission of compressed files, et al. Under the condition of limited bandwidth, the proposed approach has the capability to transmit 3D animation content to the client in real time. The 3D model is created and rendered by WebGL, and users can view and interact with the 3D digital animation in high-speed through the browser.

Key words: 3D animation; WebGL; real-time rendering; files preload; compressed files transmission

互联网以其快捷、资源丰富、信息共享等特性, 正成为人们获取信息最重要的途径. 人们对信息获取的要求不再停留在以前的 2D 页面, 而是越来越趋向于 3D 页面, 随着信息数据呈指数增长, 如何有效且具体地把信息展现出来成为 Web 领域研究的重点. 作为网络技术与三维图形技术[1]相结合的产物, Web3D 技术[2]得到越来越广泛的发展与应用, 它通过在网页上建立一个虚拟空间, 使得用户能够通过浏览器来实现三维场景的漫游和互动[3].

WebGL[4]可以为浏览器提供硬件图像加速渲染,借助系统显卡, 开发人员可以在浏览器里更流畅地展示 3D 场景和模型, 还能创建复杂的导航和数据视觉化. 显然, WebGL 技术标准免去了开发网页专用渲染插件的麻烦, 可被用于创建具有复杂 3D 结构的网站页面, 提高渲染性能与效果. 目前, 国内 WebGL 相关产品比较稀少, 相关资料与资源也比较匮乏, 因此, 本文提出的基于 WebGL 的 3D 动画在线实时播放是虚拟现实技术急需解决的关键技术问题之一, 具有很好的应用前景.

① 基金项目:国家自然科学基金(61100139,61040009);2013 年南宁市人才小高地专项资金(2013022 号)
收稿时间:2015-03-10;收到修改稿时间:2015-04-29

1 系统简介

本系统采用 WebGL 中的主流框架 Three.js[5]进行系统开发. 对于 WebGL 及 Three.js 的具体介绍请看参考文献[6,7], 这里不再重复说明.

1.1 系统设计方案

用户在浏览器上访问本系统网站, 请求并加载 3D 模型文件到用户本地实时渲染, 生成 3D 动画, 并加入用户交互操作. 本系统主要功能有:

(1) 3D 场景自动漫游动画在线实时播放;

(2) 用户在线实时控制 3D 场景自由漫游;

(3)用户在线实时控制汽车模型在 3D 场景中自由行走, 包括碰撞效果;

(4) JS 模型文件传输中的数据压缩与解压缩.

服务器端:当收到用户访问请求时, 服务器首先向浏览器端传输场景初始化所需的文件, 之后根据浏览器请求实时把已经预处理好的相应的模型文件发送到浏览器端.

浏览器端:浏览器收到服务器的响应应答后, 解析收到的场景和模型文件, 进行初始模型与场景渲染, 之后根据用户在三维场景中漫游时使用鼠标、键盘等输入设备的操作, 实现场景的实时渲染. 在漫游过程中系统将进行预加载判断, 使浏览器端能够向服务器发送请求, 在后台进行场景资源文件传输, 确保场景资源的实时获取与渲染.

系统的数据流图如图 1 所示.

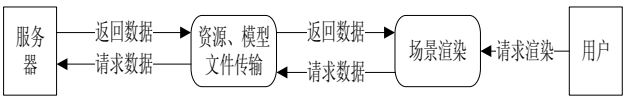

图 1 系统的数据流图

1.2 系统模块划分

本系统包括七大模块: 场景管理模块(Scene Manage)负责场景设计、资源加载; 模型管理模块(Model Manage)负责模型设计、模型加载; 用户事件响应[8]模块(User Event Respon)负责鼠标、键盘、界面按钮等事件的响应; 预加载模块(PreLoad)负责文件预加载及加载优先级判断; 渲染模块(Render)负责场景渲染; 压缩处理[9]模块(Compression)负责模型文件的压缩与解压缩处理; 异常处理模块(Exception Manage): 负责处理系统异常.

各模块有各自的功能, 而他们之间又有紧密的关系, 如图 2 所示.

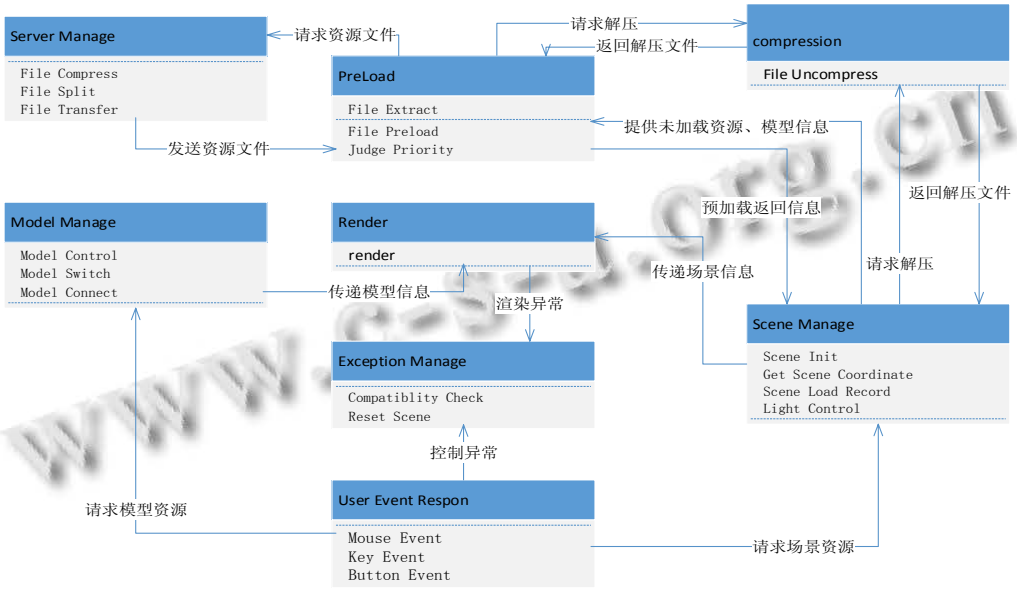

图 2 系统各模块间的关系图

2 实现的关键技术

2.1 模型处理

2.1.1 模型文件格式

Three.js 支持多种格式的 3D 模型文件的导入与渲染, 包括.js、.obj、.mtl、.dae、.ctm、.stl 等, 各种格式的文件有各自的特点, 其存储的数据信息与渲染效果也有明显的差异. 本系统采用的 3D 模型文件格式为.js, 即 JSON 格式. 系统的模型文件采用 3ds max 软件设计

而成, 可导出.3ds 或.obj 等格式, 并通过 Blender 软件[10]转换成 JSON 格式, 这就是我们要导入到场景中所需的最终文件格式.

2.1.2 模型导入

在 Three.js 中, 针对不同格式的 3D 模型文件提供了不同的导入方法, JSON 格式的模型采用 THREE.JSONLoader()方法, 其中的回调函数 callback 是对模型进行初始化处理, 并利用 THREE. MeshFace Material()进行模型材质渲染.

2.2 模型文件预加载

由于三维模型数据量巨大[11], 浏览器端加载完毕需要较长时间, 也有可能无法一次性全部加载, 这样就会造成等待时间浪费、产品用户体验差甚至不可用. 为解决这种大数据量的三维模型渲染问题, 需要设计一种动态载入数据的渲染方法, 如何有效的动态调度海量数据, 提高渲染效率, 保证人机交互的实时性, 是其中的关键问题, 而有效的预加载技术通常会改善客户的体验.

本系统的预加载原理: 把整个场景在(X,Z)平面上分为多个同等大小的网格, 这里称之为子区域, 依据 camera 在场景的(X,Z)平面中的位置, 采用八邻域法判断其邻近子区域内的模型文件是否已经加载, 若未加载则在后台对其进行加载. 若同时加载的文件过多, 优先选择距离 camera 近的文件进行加载. 算法描述如以下两点.

2.2.1 文件预加载(File Preload)

输入: 未加载的子区域 ID

处理: 根据输入的子区域 ID 加载子区域资源

算法描述:

```
Function LoadAreas(unloadareaslist){
    Var loadedAreas = new Array();
    For (var id in unloadareaslist){
        VERIFY(LOADEVENT);
        loadSubArea(id);
        loadedAreas.push(id);
      }
      Return loadedAreas;
}
```

VERIFY(LOADEVENT),判断是否可以继续加载当前未加载的子区域列表;

loadSubArea(id),返回指定 ID 的子区域资源;

输出: 本次调用加载完毕的子区域列表

2.2.2 优先级判断(Judge Priority)

输入: 用户控制对象的坐标

处理: 根据输入的坐标, 为所有分块的子区域按可能被渲染的先后顺序分配优先级,并按优先级从高到底将所有子区域排序

算法描述:

```
Function judgePriority(x,y,z){
    Var areaPriority = new Array();
    Var currentAreaid = getCurrentAreaID(x,y,z);
Var currentDirection = getCurrentDirection(x,y,z);
// 根据用户当前的坐标和漫游方向等信息//为所有子区域动态划分优先等级
     Return areaPriority;
}
```

getCurrentAreaID(x,y,z), 获取当前所在子区域;

getCurrentDirection(x,y,z), 获取当前漫游方向;

输出: 返回子区域优先级信息

2.3 交互控制技术

2.3.1 平移、旋转与缩放

在三维空间坐标中, 平移变换是最简单的变换, 变换后点坐标等于初始位置点坐标加上一个平移向量; 旋转变换较为复杂, 原理是变换后点坐标等于初始位置点坐标乘以一个变换矩阵, 绕指定的任意轴旋转变换就是由几个绕坐标轴旋转变换和平移变换效果叠加而成的. 缩放变换的原理同样也是变换后点坐标等于初始位置点坐标乘以一个变换矩阵.

在 Three.js 中, 控制对象的平移、旋转及缩放是比较容易实现的, 分别对应 position, rotation, scale 这三个属性. 函数 updateMatrix() 和 updateMatrixWorld (force)将根据 position, rotation, scale 参数更新对象的 matrix 和 matrixWorld, 其中 matrix 表示本地的模型矩阵, 仅仅表示该对象的运动, 而 matrixWorld 则需要依次向父亲节点迭代, 每一次迭代都左乘父亲对象的本地模型矩阵, 实际上就是左乘父亲对象的全局模型矩阵. 另外, 值得强调的一点是, 系统中加入了 Physi.js[12]这个物理框架, 当修改对象的 position, rotation 参数, 如: obj.position.x + = 5; obj.rotation.y = 0.1;我们会发现, 修改后该对象的模型矩阵不会自动更新, 依旧保持原来的状态, 这时需要加入 obj.__dirtyPosition() 及 obj.__dirtyRotation()来更新模型矩阵, 达到修改效果.

2.3.2 视角变换

系统中有三种视角变换控制方式: 自动漫游动画、自由漫游及控制汽车行走.

自动漫游动画使用的是 PathControl.js, 此方法可以为场景中的 camera 设定行走路线, 让 camera 沿着路线行走, 达到漫游效果, 最终渲染成 3D 动画. 漫游动画的路线设定灵活, 可以用函数表示, 或者定义一连串点组成路径数组, 加入到漫游控制器中, 然后就可以在场景中生成漫游动画了. 算法如下所示.

```
//生成控制器
var pathControls = new
                    THREE.PathControls(camera);
// 把路线加入控制器里
controls.points.forEach(function(e)
    pathControls.waypoints.push([e.x, e.y, e.z])});
//开始漫游
var delta = clock.getDelta();
THREE.AnimationHandler.update(delta);
pathControls.update(delta);
```

自由漫游采用的是 FirstPersonControls.js, 该方法是以第一人称视角, 通过控制鼠标和键盘完成漫游. 方法中有多个属性可以修改漫游的效果, 灵活自由, 如 lookSpeed、movementSpeed 等. 值得注意的是, 其中两个属性, lon 和 lat 是用于设置开始漫游时 camera 所看的方向.

```
Var camControls = new
                THREE.FirstPersonControls(camera);
camControls.lon = -150;
camControls.lat = 120;
```

2.3.3 控制汽车模型运动

汽车的运动原理, 是把轮胎跟车身绑定在一起, 产生运动约束的效果, 即通过轮胎的转动及转向带动车身前进. 我们引入另外一个框架——Physi.js, 此框架基于 Three.js 建立了产生物理运动效果的功能函数, 使 3D 场景更加生动形象, 贴近现实. 在 Physi.js 中, 有多种对象间的运动约束效果, 包括 PointConstraint(点约束)、HingeConstraint(铰链约束)、SliderConstraint(滑动约束)、ConeTwistConstraint(关节约束)、DOFConstraint(万能约束, 或称自由约束).

对于汽车的运动, 需要用到 DOFConstraint 方法. 如 var frConstraint = new Physijs.DOFConstraint (fr,body,new THREE.Vector3(0,4,8)), 第一个参数 fr 是约束主体, 第二个参数 body 是绑定对象, 第三个参数是第一个参数所要绑定到的坐标位置, 即 fr 的运动受到了 body 的约束, 也就是说 fr 运动带动了 body 的运动. 这里不再详细描述 DOFConstraint, 因为 Physi.js 中有专门针对控制汽车运动而建立的功能函数 Physijs.Vehicle(), 实现算法如下所示:

```
var vehicle = new Physijs.Vehicle(mesh, new
Physijs.VehicleTuning(
            100.88, 10.83, 0.2, 500, 10.5, 30000));
vehicle.addWheel(
    for ( var i = 0; i < 4; i++ ) {
     //汽车主体与 4 个轮胎绑定, 各参数设置
    ``````)}
//设置控制汽车行驶的各种参数
input[0] = { power: null, velocity:0, vsize:100,
            vmax:320, brake: null, direction: null,
            steering: 0, front:2, back:0, fd:1,
            pos:vehicle.mesh.position };
```

### 2.4 文件压缩与解压缩技术

由于三维模型文件巨大, 对数据存储和数据传输提出了挑战, 采用压缩编码技术可以节省数据的存储空间, 同时也使数据易于在网络上进行传输, 故本文设计出一种有效的模型文件无损压缩与解压缩方法.

本系统所用的方法基于 zip.js 压缩算法, 先在服务器端对模型文件(.js 格式)进行压缩, 得到无损压缩文件(.zip 格式), 然后结合 Three.js 中的 THREE.JSONLoader()方法, 写出 loadZip()方法, 用以在场景中加载经过 zip 压缩的模型文件. 当压缩文件加载完成, 浏览器对其进行解压, 其解压原理也是根据 zip.js 中的解压算法, 得到压缩前的模型文件(.js 格式), 进而调用 THREE.JSONLoader()方法解析模型, 并渲染到场景中. 整个过程的流程图如图 3 所示, 具体压缩效果在实验结果中体现.

## 3 实验结果

### 3.1 实验环境

本实验采用的操作系统为 Windows 7; 硬件配置为: Intel®Core® i3 处理器, 主频 3.30GHz, 内存 4GB, Intel® HD Graphics 独立显卡 512M; 浏览器为 Google Chrome 36; 网络带宽 4MB/s, 即真实下载速度为 512KB/s. 系统
```

以 Three.js 为开发框架, 在线实时渲染 3D 动画.

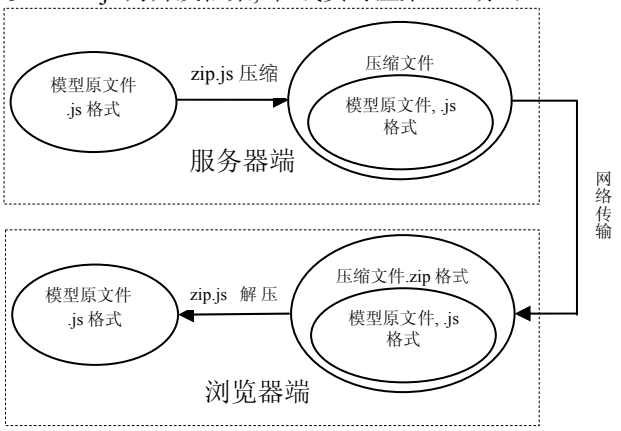

图3　模型文件压缩与解压流程图

3.2 渲染初始场景

首先浏览器访问服务器, 服务器向浏览器发送场景资源与文件, 浏览器对其进行解析之后渲染出初始场景. 在实验中, 分别测试了同一环境中, 包括硬件、软件、网络带宽等, 两个初始场景从开始加载文件到最终渲染出整个初始场景所需的时间, 其中一个初始场景是未对模型文件进行压缩, 另一个是已对模型文件进行过压缩. 共进行了五组测试, 结果如表 1 所示.

表 1　两个场景初始化所需时间

序号	未压缩场景文件	已压缩场景文件	时间比
1	183.1s	47.2s	3.87 : 1
2	204.6s	51.0s	4.01 : 1
3	171.9s	45.4s	3.78 : 1
4	205.5s	52.2s	3.93 : 1
5	190.2s	47.8s	3.98 : 1
平均值			3.91 : 1

从表中可以看出, 由于模型文件经过压缩处理后文件缩小, 传输所需时间更短, 至渲染完成所需时间约为未压缩时的 1/4, 所以, 本系统采用的压缩传输方法大大减少了渲染初始化的时间, 提高了渲染的实时性.

3.3 交互效果

初始场景渲染完成后, 接着进行 3D 场景自动漫游动画在线实时播放, 如图 4(1), 系统在未进行其他操作前会一直循环播放动画, 退出可按"1"、"2"或"3"键, 重新播放可按"4"键; 可以按"3"键切换到自由漫游视角, 在线实时控制 3D 场景自由漫游, 如图 4(2), 按"↑","↓","←","→"控制漫游行走, 移动鼠标转变漫游方向, 并且可按"q"键暂停/恢复漫游; 可以通过按"1","2"键选择想要控制的汽车模型, 在线实时控制汽车模型在 3D 场景中漫游行驶, 如图 4(3), 同样地, 按"↑","↓","←","→"控制汽车的行走, 按"空格"键刹车; 汽车行驶中加入了物理碰撞效果, 当汽车碰到建筑物或木箱时是会受到阻碍, 如图 4(4), 跟现实中的碰撞现象十分相似, 用户体验效果很好.

(1) 初始场景

(2) 自由漫游

(3) 控制汽车行走

(4) 物理碰撞效果

图 4　系统 3D 动画在线实时播放效果

3.4 渲染帧率

帧率表示图形处理器处理场景时每秒钟能够更新的次数. 高帧率表示更流畅、更逼真的动画效果. 一般来说 30FPS 就是可以接受的, 也是实时的效果. 但是将性能至 30FPS 以上则可以提升交互感和逼真感. 在实验中, 分别测试了系统在自动漫游动画、自由漫游及控制汽车行驶三种视角状态下场景渲染的帧率, 每隔 10 秒记录一次, 结果如表 2 所示.

表 2 渲染帧率记录表

序号 / 视角状态	1	2	3	4	5	6	7	8	9	10	平均值
自动漫游动画	42FPS	49FPS	46 FPS	45FPS	43FPS	50FPS	45FPS	46FPS	46FPS	50FPS	46FPS
自由漫游	35FPS	39FPS	42 FPS	40FPS	39FPS	45FPS	47FPS	46FPS	42FPS	47FPS	42FPS
控制汽车行驶	33FPS	32FPS	40FPS	43FPS	38FPS	37FPS	39FPS	35FPS	35FPS	36FPS	37FPS

从表 2 可知, 系统总体平均渲染帧率约为 42 FPS. 在实验过程中, 由于不同时刻场景内容不同, 渲染帧率出现不小的波动, 但三种视角状态的渲染性能都达到了可接受范围, 动画与场景渲染流畅, 用户体验较好. 其中在控制汽车行驶时的渲染帧率最低, 这是由于在控制汽车行驶时交互响应较多, 系统解析需要花一定的时间, 且汽车模型一直出现在场景动画中, 占用了一定的渲染空间. 即便如此, 其 37 FPS 的平均渲染帧率还是能反映出系统良好的在线实时渲染效果.

另外, 为了体现 WebGL 具有硬件加速效果的特点, 以此与部分传统 Web3D 技术(不具有硬件加速功能)形成优势对比, 实验中还测试了在同一场景下, 未启用硬件加速功能时的渲染帧率, 并计算出其平均值, 结果如表 3 所示.

表 3 是否启用硬件加速时的渲染帧率比较

视角状态	启用硬件加速	未启用硬件加速
自动漫游动画	46 FPS	27 FPS
自由漫游	42 FPS	22 FPS
控制汽车行驶	37 FPS	16 FPS

从表 3 中可以看出, 在未启用硬件加速功能的条件下, 系统的渲染帧率大幅降低, 渲染性能相去甚远. 因此, WebGL 具有硬件加速的特点为 3D 渲染带来了较好的效率与实时性, 具有明显的优势.

4 总结

本系统是基于 WebGL 的三维动画实时播放系统, 采用 Three.js 框架进行系统开发, 在研究 3D 场景的创建及渲染技术的同时, 提出模型文件预加载、文件压缩传输等方法, 提高了场景文件的加载速度, 增强了动画渲染的效率与实时性. 系统平均渲染帧率约为 42FPS, 相比较其他 Web3D 渲染技术, 如 Flash3D, WebGL 免去了开发网页专用渲染插件的麻烦, 同时场景及模型渲染效果更加形象逼真, 动画播放流畅, 渲染帧率高, 交互性强, 具有很好的研究前景与应用价值.

参考文献

1 王映辉.3D 建模与编程技术.计算机应用研究,2004, 1:37–43.

2 韩义.Web3D 及 Web 三维可视化新发展——以 WebGL 和 O3D 为例.科技广场,2010:81–86.

3 张健康,杨宜康,李雪等.基于 Java3D 的地球空间可视化研究.计算机应用研究,2013,30(1).

4 方路平,李国鹏,洪文杰.基于 WebGL 的医学图像三维可视化研究.计算机系统应用,2013,22(9):25–30.

5 Jos.Dirksen. Learning Three.js: the JavaScrip 3D library for WebGL. Packt Publishing ,2013: 7–35.

6 王东.基于 Web3D 的 VRML 三维造型及动画技术研究.成都:四川大学,2005.

7 张平,杨时杰,梁斌.基于 Java3D 的空间机器人运动仿真系统.计算机应用研究,2007,24(9):19–21.

8 陈滔滔,江晓宇,温佩贤等.基于 Web3D 的人脸模型定制系统.系统仿真学报,2014,26(2):382–388.

9 李海生,刘成.三维模型网格数据压缩技术研究.系统仿真学报,2013,25(9):2150–2156.

10 Parisi T. WebGL Up and Running. O'Reilly Media, 2012: 145–155.

11 崔滨.海量数据实时三维交互式显示关键技术研究.上海:上海大学.2010.

12 https://github.com/mrdoob/three.js/.