

## 并行绘制系统 Chromium 中的 3D 模型数据压缩

王欢, 秦开怀

(清华大学 计算机科学与技术系, 北京 100084)

(whustb168@gmail.com; qkh-dcs@tsinghua.edu.cn)

**摘要:**网络带宽不足严重限制了 Chromium 等并行图形绘制系统渲染巨型几何场景的速度。通过对网络传输中的几何数据进行无损压缩,提出了一种能有效缓解网络负荷的方法。该方法可以很容易地实现不同算法对特定几何数据的压缩。实现了 ZLib 和哈夫曼算法对 Chromium 系统的压缩,测试了系统对 10 类 OpenGL 应用程序的加速比和压缩比,以及在 4 种配置环境下的并行运行效果。使用 ZLib 算法时,测试程序的运行速度都有不同程度的提高,最高提升 3 倍;数据压缩比平均在 5.0 以上,最高为 30;并行绘制加速比在单服务器数目下最高。ZLib 算法整体表现良好,能有效减少网络通信量。

**关键词:**Chromium 系统; ZLib 算法; 哈夫曼算法; 几何数据压缩; 分布式并行绘制系统

**中图分类号:** TP391 **文献标志码:** A

## 3D model data compression in parallel rendering system Chromium

WANG Huan, QIN Kai-huai

(Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China)

**Abstract:** The deficient network bandwidth has severely curtailed the rendering speed of large geometric scenes in parallel graphics rendering systems such as Chromium. Through lossless compression of geometric data in the network transmission, an efficient method which alleviated the network burden was proposed. This method can easily be used to implement different algorithms for specific geometry data compression. The authors applied ZLib and Huffman compression algorithms in Chromium, and tested the speedup and compression ratios of 10 kinds of OpenGL applications, and the parallel performance in the four configurations. The speed of the tested programs that uses ZLib has been improved to different degrees and the maximum raising is 3 times; the average ratio of data compression is above 5.0, up to the maximum 30; the speedup ratio of parallel rendering on a single server is the highest. The overall performance of the ZLib algorithm is good, which can effectively reduce the network traffic.

**Key words:** Chromium system; ZLib algorithm; Huffman algorithm; geometric data compression; distributed parallel rendering system

### 0 引言

并行图形的绘制主要用于高端图形的计算领域。图形硬件技术的飞速发展和大规模数据显示的迫切需求,使得研究基于集群的并行图形绘制系统变得尤为重要。相比于 SGI 等专业的图形处理机,PC 集群具有性价比高、扩展性好、使用灵活和升级方便的优点。并行图形绘制系统是 PC 集群的软件支撑平台,它的研制及优化一直是图形领域的重要研究课题。

现有的并行绘制系统按照绘制流水线的并行方式可分为前分布拼接合成(sort-first)、中分布拼接合成(sort-middle)、全图像深度合成(sort-last)三类<sup>[1]</sup>。sort-first 在几何变换之前执行,采用对屏幕划分的方法分别绘制,主要优点是通信量少,但容易使部分图元分配过于集中,造成负载不平衡,严重影响绘制性能;sort-last 发生在光栅化之后,采用每个处理器处理整幅图像数据,最终根据深度信息对像素进行合成的方法,但对传输带宽要求较高,可扩展性差,像素数据传输和深度合成是其主要问题;sort-middle 则介于两者之间,硬件产品的功能限制了该算法在集群上的应用<sup>[2]</sup>。并行绘制系统按照场景数据的控制方式可分为立即模式和保留模式两类。立

即模式每绘制一帧都要通过网络传输大量的重发数据,有限的网络带宽严重限制了绘制速度;保留模式虽然利用帧间连贯性等方式减少了传输数据量,但使用不够灵活。

文献[3]指出:网络带宽不足是并行图形绘制系统面临的主要问题。虽然可以通过增加网络带宽、增强图形处理能力等硬件措施来提高运行速度,但这样的设备或者适用范围有限,或者价格昂贵,难以满足大部分用户的需求。而且这种通过提高硬件设备速度的方法,并没有在本质上解决网络带宽不足的问题,当应用程序处理的几何数据很大时,仍然表现为带宽不足(如测试程序中的 rotateG)。目前提升并行绘制系统性能的方式可分为负载均衡、数据压缩传输和优化的图形合成三种。本文研究了应用数据压缩方法减少数据的传输量,缓解网络带宽这一课题。实验结果表明:使用 ZLib 算法对实时三维几何数据在网络环境下的压缩,有效缓解了网络带宽不足的问题。

### 1 相关工作

Chromium 系统<sup>[4]</sup>是斯坦福大学开发的大规模并行图形绘制系统,支持立即模式渲染,支持 sort-first、sort-last 及 sort-

收稿日期:2010-07-12;修回日期:2010-08-17。

**作者简介:**王欢(1988-),男,河北衡水人,博士研究生,主要研究方向:真实感图形绘制、3D 立体显示;秦开怀(1958-),男,湖北潜江人,教授,博士,主要研究方向:计算机图形学及动画、3D 立体显示、小波变换及多分辨率几何造型、虚拟现实、图像处理及可视化。

first 和 sort-last 混合配置的渲染算法。由于使用低端显卡组建成大型的集群系统,Chromium 系统使用价格低廉、可扩展性好,能够支持复杂场景的 3D 渲染,获得了广泛的应用。但对于超大规模复杂场景的渲染速度不够理想。Chromium 系统的主要特点在于流处理单元(Stream Processing Unit, SPU)组建技术和独特的状态跟踪机制。Quake III 游戏运行于 Chromium 时的典型配置环境如图 1 所示(虚线框内的 Compress 和 Uncompress 表示是否执行压缩和解压缩操作)<sup>[5]</sup>。当数据包不压缩时,Chromium 的客户端节点 Crappfaker 通过加载系统自己的 OpenGL 动态链接库,捕获 Quake III 游戏中的 OpenGL 函数和参数值,使用 Pack SPU 对其进行简单的打包后,通过网络传输到服务器端节点 Cserver,最后使用 Render SPU 执行,并把画面显示在服务器端。

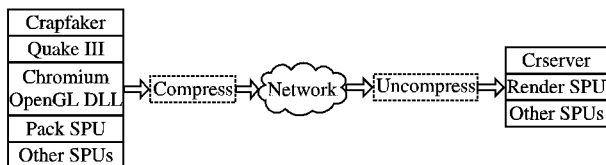


图 1 Chromium 上运行 Quake III 游戏时的配置

Chromium 建立的流处理模型 SPU,可以自由构建并行绘制结构。SPU 通过捕获 OpenGL 函数并进行一定修改,实现了把标准的 OpenGL(目前支持 OpenGL 1.5 版本)应用程序以无需修改任何代码的方式运行在分布式集群上。流跟踪机制可以捕获 OpenGL 在任何时刻的各种函数变量,并提供追踪、通信、更改功能。用户可以定义自己的 SPU,更改某些特定的 OpenGL 函数及参数值,来实现特定的显示功能。

Chromium 的状态跟踪机制可以通过网络渲染跟踪图形状态,包括进行状态跟踪的主要 OpenGL 指令、状态控制方法、显示列表技术、纹理管理和性能测试<sup>[6]</sup>。应用程序的状态参数(如照明参数、雾设置等)全部被 Chromium 系统存储,并被逐个分发到各个渲染服务器上。Chromium 可以跟踪应用程序的整个 OpenGL 状态和每台服务器的当前状态,并几乎可以在任何时间更改。由于应用程序状态参数的改变量相比于几何数据的传输量是很小的,所以状态参数的压缩与否不会明显影响系统性能。Chromium 同时还实现了直接利用现有的高性能图形应用程序 API(如 OpenGL、X11)的并行接口模型,可以实现集群的并行绘制<sup>[7]</sup>。

事实上,Chromium 系统使用 Zpix SPU 仅对 glDrawPixels 函数数据进行了初步压缩,仍有大量函数数据没有压缩,对大型几何场景还不能达到理想的性能要求(绘制速率在 20 fps 以上,而真实感图形显示一般要求在 25 fps 以上)。文献[8]针对 AnyGL 系统提出了一种几何指令流压缩方法,使用了 4 类预测器和自适应量化算法组合在一起分别压缩,仍有较大改进空间。在低速网络以及运行有大规模几何数据的大型 OpenGL 应用程序时,Chromium 的运行速度有时甚至达不到 10 fps,运行界面非常不流畅。其主要原因就是 Chromium 在通过网络发送数据包时,其几何数据占了很大流量。因此,把基于软件的数据压缩方法实现于并行图形绘制系统具有很好的实际应用价值。本文试图通过数据压缩的方法在数据包发送之前先进行压缩,把压缩后的数据包通过网络传输出去,数据包到达服务器端后,再通过解压缩算法进行解压。

## 2 压缩算法的实现

### 2.1 ZLib 和哈夫曼压缩算法

数据压缩是利用各种算法将数据冗余压缩到最小,并尽可能地减少失真,从而提高传输效率和节约存储空间<sup>[9]</sup>。数据压缩技术一般分为有损压缩和无损压缩。无损压缩是指重构出的压缩数据与原来数据完全相同的压缩算法。与本文相关的反映无损压缩算法好坏的标准有压缩比和加速比两个指标。压缩比  $\alpha$  的定义如下:

$$\alpha = l_{in}/l_{out}$$

其中  $l_{in}$  表示输入流大小,  $l_{out}$  表示输出流大小。压缩比的大小反映了数据包中冗余数据的多少。压缩比越大,意味着数据包的冗余数据越多,加速效果会越明显;但压缩比过大,也会消耗更多的压缩和解压时间,导致程序运行速率减慢。加速比  $\beta$  的定义如下:

$$\beta = v_2/v_1$$

其中  $v_1$  表示压缩前程序执行速度,  $v_2$  表示压缩后执行速度。一般情况下,数据包一次性传送的数据量越大,加速比会越大,相应的加速效果也越明显;但数据包容量过大,也会导致网络延迟,致使加速比减小。

ZLib 压缩算法是一种对 GZIP 算法改进的无损压缩算法,由 Jean-loup Gailly 与 Mark Adler 开发。ZLib 是一个跨平台的压缩函数库,提供了一套 in-memory 压缩和解压函数,并能检测解压出来的数据的完整性。ZLib 使用抽象化的 DEFLATE 算法,把 LZ77 算法和哈夫曼树结合起来实现,占用很少的系统资源,并对各种数据提供良好的压缩效果。ZLib 算法已被广泛用于如 Linux 内核、Zip 文件格式及嵌入式设备等各种产品中,已经成为一种事实上的业界标准。ZLib 提供了可以直接对字节流操作的压缩函数 compress 和解压缩 uncompress,函数的具体使用方法可以参见文献[10]。

哈夫曼算法先对原始字符串的频率排序,再建立哈夫曼树重新编码,使出现频率高的字符使用较短的码长,从而达到压缩存储的目的。由于哈夫曼编码符号的出现频率难以预知,需要执行统计和编码两次操作,所以压缩速度相对慢些。此外,由于该算法必须记录字符串频率等固定信息,所以当压缩文件较小时,压缩比并不高。因此,可以认为使用哈夫曼算法对网络环境下数据的实时压缩效果不会理想。

### 2.2 数据打包机制

Chromium 对 OpenGL 命令和参数把整个缓冲区划分为操作码缓冲区和数据缓冲区分别对数据进行打包。第一部分为 OpenGL 操作码缓冲区,使用哈希表法存储 OpenGL 函数名,每个函数名占用 1 B;第二部分为 OpenGL 数据缓冲区,存储函数参数。OpenGL 操作码缓冲区的指针按地址递减方式存储,OpenGL 命令参数缓冲区的指针按地址递增方式存储。Chromium 实现网络命令打包的数据结构如图 2 所示(图中阴影部分的每个小框代表 1 B)<sup>[11]</sup>。

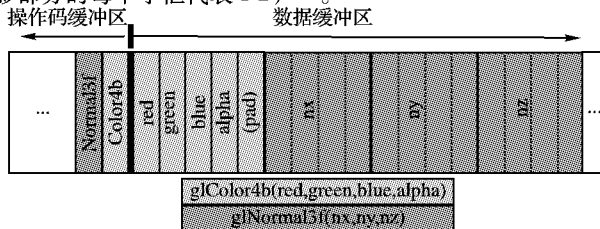


图 2 Chromium 系统几何数据打包原理

应用程序的 OpenGL 函数被客户端截获后,通过 Pack SPU 打包至网络端,以消息形式等待发送。图 2 显示了即将发送的数据的结构,所有打包数据都以字节流的形式存在。此时,运行算法 1(压缩算法),通过网络发送到服务器端,然后运行算法 2(解压缩算法),再继续执行。。

注意到 Chromium 系统传输的数据量大小不一,并且类型各异,大致可分为几何数据、操作码数据和控制数据三类。我们不能对所有的传输数据进行压缩。事实上,压缩比并不是我们要关心的主要问题,虽然压缩比的提高确实能在一定程度上提高加速比。一味地提高数据压缩比是不明智的,毕竟它只是提升程序加速比的中间过程。对于数据类型比较复杂且长度比较小的数据包,执行压缩反而会降低程序运行速率。因此,我们只针对数据量较大的几何数据进行压缩,这样压缩比和执行效果才会比较好。这样程序中需要加入是否压缩标志,以便区分。由于压缩后的数据包长度大小不一以及压缩时间难以控制,所以需要建立一个压缩缓冲区,控制发送长度和发送时间间隔,避免程序运行速度时快时慢。

算法 1 中压缩算法首先判断数据包类型是否为规定的 MSGCOMPRESSTYPE 以及是否大于最小长度,如果满足条件则执行压缩,否则标记该包不用压缩。压缩前先去除数据包的头部信息,之后选择要使用的压缩算法,默认使用 ZLib 算法。所有压缩函数至少包含 4 个参数,分别表示压缩前后字符串及其长度。数据包压缩完后,需执行恢复数据包头部信息、标记数据包已被压缩、记录原始数据长度、重新设置需发送的字节数等操作。算法 2 中解压缩算法的执行过程与算法 1 正好相反,只是需要先判断数据包是否已被压缩过,并获得原始数据包长度。

压缩过程有很好的可扩展性。压缩和解压缩代码都在数据发送前和接收后立即执行,避免了 Chromium 系统大规模的代码修改。如果使用如 LZW 等其他压缩算法,只需选择相应的压缩类型,并在代码中插入压缩函数即可实现。而要实现多重压缩,只需定义一个压缩数组,多次调用压缩函数,并在服务器端调用解压缩数组相应次数即可。

#### 算法 1 压缩算法。

```
//选择要压缩的数据类型,且大于最小压缩长度
if msgBefore.msgType == MSGCOMPRESSTYPE &&
    lenBeforeCompress > minCompressLen
msgBefore = msgBefore + lenMsgHead; //选择数据包的压缩范围
switch COMPRESSTYPE //选择压缩算法,默认为 ZLib
case 1:
//用哈夫曼算法压缩
compressHuff(msgAfter, &lenAfterCompress, msgBefore,
    lenBeforeCompress);
default:
//用 ZLib 算法压缩, alpha 压缩参数,使用默认值
compressZLib(msgAfter, &lenAfterCompress, msgBefore,
    lenBeforeCompress, alpha);
end
msgAfter -= lenMsgHead; //恢复数据包的头部数据
msgAfter -> ifCompress = TRUE; //标记数据包已压缩
msgAfter -> lenBefore = lenBeforeCompress;
//记录压缩前的数据包长度
lenTrans = lenAfterCompress; //重新设置需传输的数据包长度
else
msgBefore -> ifCompress = FALSE; //标记数据包没有压缩
```

end

#### 算法 2 解压缩算法。

```
//选择要解压的数据类型,且数据确实被压缩过
if msgReceive.msgType == MSGCOMPRESSTYPE && msg_opcodes
-> ifCompress
msgReceive = msgReceive + lenMsgHead;
//选择数据包的解压缩范围
lenUnCompress = msgReceive -> lenBefore;
//获取数据包压缩前的长度
switch COMPRESSTYPE //选择解压缩算法,默认为 ZLib
case 1:
//用哈夫曼算法解压缩
uncompressHuff(msgUnCompress, &lenUnCompress,
    msgReceive, lenReceive);
default:
//用 ZLib 算法解压缩, alpha 解压缩参数
uncompressZLib(msgUnCompress, &lenUnCompress,
    msgReceive, lenReceive);
end
msgUnCompress -= lenMsgHead; //恢复数据包的头部数据
msgUnCompress -> ifCompress = FALSE;
end
```

注意到使用上述方法,不一定会使 Chromium 的运行速度更快。这是因为几何数据在压缩和解压缩之时都要花费时间。只有选择具有高压缩比和快执行速率双重优点的数据压缩算法,Chromium 的运行速度才有可能得到提升。

### 3 实验结果

本文初步实现了 ZLib 和哈夫曼压缩算法在 Chromium 上的应用,并进行了大量测试确保程序的有效性。测试机器的客户端 CPU 使用 Intel 酷睿 2 双核 E8200 主频 2.66 GHz 型号,服务器端 CPU 使用 Intel 酷睿 i7 920 主频 2.66 GHz 型号,交换机使用华为公司生产的 S1016 型号,其传输速率为 10 Mbps/100 Mbps。所有测试程序的帧绘制速率每隔 5 s 记录一次平均值,共记录 50 次。使用哈夫曼算法压缩时,所测试的 10 个程序中只有 city 程序加速最明显,但加速比也只有 1.698,不足 ZLib 算法的 80%;同类程序中,哈夫曼算法的数据压缩比一般不超过 1.5,也明显落后于 ZLib 算法。这表明哈夫曼算法的加速效果明显落后于 ZLib 算法,因此下面仅讨论 ZLib 算法的压缩效果。

表 1 使用 ZLib 压缩算法的测试程序运行结果

程序名称	压缩前平均帧率/fps	压缩后平均帧率/fps	加速比	数据压缩比
rotateG	5.38	17.43	3.23	6.32
spheres	3.31	6.27	1.90	3.64
magic room	30.11	122.72	4.08	31.50
hlsaver	24.27	40.55	1.67	1.52 ~ 6.51
atlantis	206.97	462.09	2.23	6.04
city	85.70	186.67	2.18	9.14
v6 engine	565.43	953.61	1.69	4.32
space scene	584.04	1435.21	2.46	1.80 ~ 16.23
Quake III	27.78	39.56	1.42	6.78 ~ 50.32
space invaders	10.06	11.91	1.18	4.43

表 1 为 10 个典型的 OpenGL 程序使用 ZLib 压缩算法的平均加速比和平均压缩比。从表中可以看出,测试程序的运行速度都有不同程度的提高,大部分程序提高超过了 0.5 倍,

magic room 程序速度提升最快,平均加速了 3 倍;数据压缩比最高为 magic room 程序,压缩了近 30 倍,数据压缩比最低为 hlsaver 程序,但也在 1.5 倍左右,平均数据压缩比在 5.0 以上,结果比较理想。

然下降得很厉害,这是下一阶段将要解决的问题。

## 4 结语

网络带宽不足是并行图形绘制系统的主要问题。本文在

Chromium 系统上,针对特定数据包的特定数据,使用 ZLib 和哈夫曼压缩算法实现了数据的压缩传输,有效缓解了网络带宽不足问题。ZLib 算法表现最好,大部分测试程序的运行速度提高明显,显示了很好的压缩效果。

从测试结果可以看出,有些交互性较强的程序(频繁的键盘、鼠标事件)压缩效果不很明显。因此,下一步需要分析键盘、鼠标等控制数据的传输机制。测试 space invaders 程序时,虽然程序运行速度提升了,但少数帧画面出现短暂停顿,可能的原因是压缩时间控制不均匀,导致数据包接收时快时慢,这也是需要进一步改进的地方。

### 参考文献:

- [1] MOLNAR S, COX M, ELLSWORTH D, *et al.* A sorting classification of parallel rendering[J]. IEEE Computer Graphics and Applications, 1994, 14(4): 23-32.
- [2] 彭浩宇,金哲凡,石教英. 基于保留模式的 in-the-core 并行超大数据量图形绘制[J]. 软件学报, 2004, 15(zk): 223-231.
- [3] 杨建. AnyGL: 一个大规模混合分布图形系统[D]. 杭州: 浙江大学, 2002.
- [4] HUMPHREYS G, HOUSTON M, NG R, *et al.* Chromium: A stream processing framework

for interactive rendering on clusters[J]. ACM Transactions on Graphics, 2002, 21(3): 693-702.

- [5] Chromium Documentation: Version 1.9 [EB/OL]. [2010-02-15]. <http://chromium.sourceforge.net/>.
- [6] BUCK I, HUMPHREYS G, HANRAHAN P. Tracking graphics state for networked rendering[C]// Proceedings of SIGGRAPH/Eurographics Workshop on Graphics Hardware. New York: ACM, 2000: 87-95.
- [7] IGEHY H, STOLL G, HANRAHAN P. The design of a parallel graphics interface[C]// Proceedings of SIGGRAPH 98. New York: ACM, 1998: 141-150.
- [8] 金哲凡, 杨建, 石教英. 分布式并行绘制系统中几何指令流压缩的研究与实现[J]. 计算机辅助设计与图形学学报, 2002, 14(9): 824-828.
- [9] SALOMON D. Data compression: The complete reference[M]. 2nd edition. 北京: 电子工业出版社, 2003.
- [10] JEAN-LOUP GAILLY, MARK ADLER. ZLib manual[EB/OL]. [2010-02-15]. <http://www.zlib.net/manual.html>.
- [11] HUMPHREYS G, BUCK I, ELDRIDGE M, *et al.* Distributed rendering for scalable displays[C]// Proceedings of the 2000 ACM/IEEE Conference on Supercomputing. Washington, DC: IEEE Computer Society, 2000: 30.

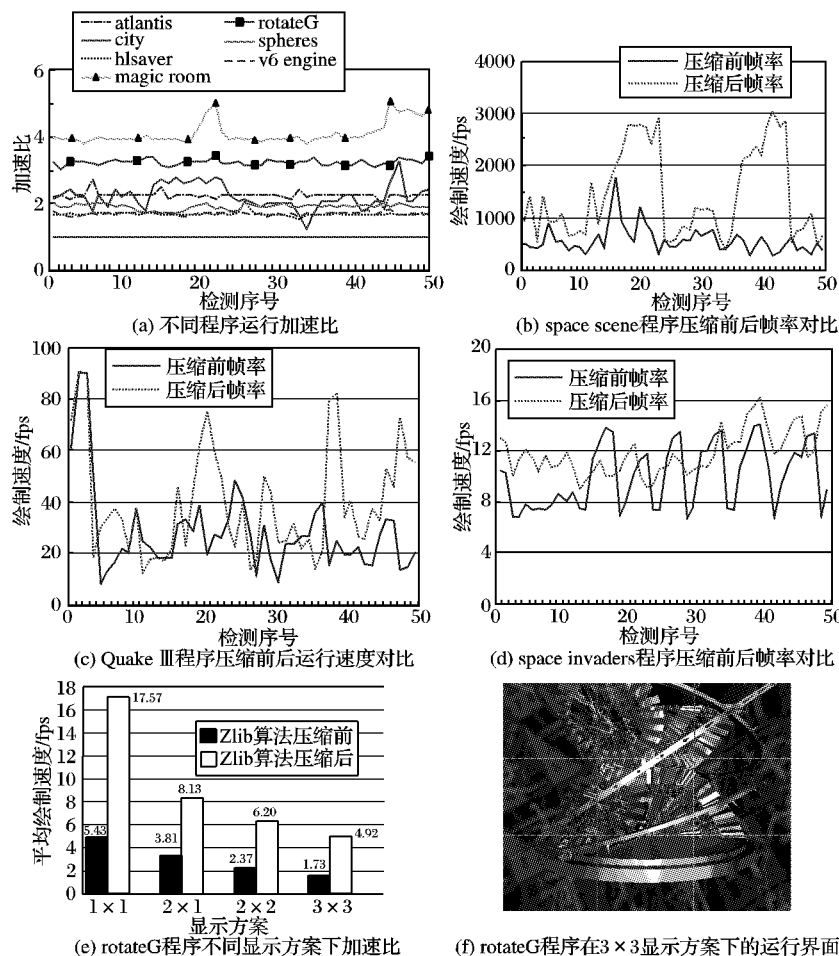


图 3 使用 ZLib 压缩算法时, 10 类 OpenGL 程序测试结果

图 3(a) 显示了加速效果较明显的 OpenGL 程序运行时加速比随运行时间变化的动态过程, 可以看出这些应用程序的加速比随运行时间的波动比较小, 说明传输数据相对一致。图 3(b) 显示了加速比高但波动剧烈的 space scene 程序压缩前后的对比, 说明传输数据变化范围比较大, 压缩消耗时间控制不平衡, 导致程序运行快慢不一致, 需进一步改善。图 3(c) 显示了 Quake III 程序压缩前后运行速率的对比, 可以看出帧绘制速率波动很大, 这是由于压缩时间控制不均匀所致。其主要原因在于该程序的交互性比较强, 发送的控制数据相对较多。测试程序中的 space invaders 交互性也很强, 它的结果也证实了这个原因, 见图 3(d)。

为了测试压缩算法对程序并行运行结果的影响, 本文分别在  $1 \times 1$ 、 $2 \times 1$ 、 $2 \times 2$ 、 $3 \times 3$  四种显示方案下对 rotateG 程序进行了测试, 测试结果见图 3(e) 和 (f)。可以看出,  $1 \times 1$  方案下加速效果最明显, 平均加速比是 3.23。当只有一台服务器时, 加速效果最明显, 加速后的帧绘制速率约在 17.57 fps 左右, 当增加 1 台服务器时, 渲染速度迅速下滑至 8.14 fps, 之后绘制速度随着服务器数目的增多依次下降。很显然, 从单服务器绘制过渡到并行绘制这一阶段速率下降最明显, 这和 Chromium 系统的设计有关。这说明随着渲染服务器数目的增多, 尽管 ZLib 算法有了较好的加速效果, 但其渲染速度仍