Direct3D Player 性能优化

周宽久[1],侯 刚[2],张熙菡[2]
ZHOU Kuan-jiu[1],HOU Gang[2],ZHANG Xi-han[2]

1.大连理工大学 管理学院,辽宁 大连 116024
2.大连理工大学 软件学院,辽宁 大连 116024
1.School of Management,Dalian University of Technology,Dalian,Liaoning 116024,China
2.School of Software,Dalian University of Technology,Dalian,Liaoning 116024,China
E-mail:zhoukj@dlut.edu.cn

ZHOU Kuan-jiu,HOU Gang,ZHANG Xi-han.Performance optimization of D3D Player.Computer Engineering and Applications,2007,43(23):97-99.

Abstract: The D3D Player is an indispensable tool software for video card driver development.The developers of the driver don't take the execution efficiency into account during D3D Player design.Hence,the previous D3D Player usually takes much more time to render one frame,which slows development of a video card driver.Some solutions to optimize the source codes to improve D3D Player's execution performance are issued.The new D3D Player is composed of a Logger Script compiling system and an executing system.Finally,some performance comparisons of results of the two versions are made with the famous tools 3D MARK 2001-2005.
Key words: Direct3D Player;Direct3D Logger;video card driver;performance optimization

摘 要:Direct3D(以下简称 D3D)Player 是测试显卡驱动所必须的工具软件,其设计并没有考虑到性能,渲染一帧画面通常需要很长时间,给显卡驱动程序的开发带来不便。提出针对 D3D Player 的性能优化方法,从其运行的机制上研究如何提高其性能,将其分成脚本编译子系统和编译后目标代码执行子系统,最后实现一个优化的 D3D Player,并以 3D MARK 2001-2005 的若干个测试用例来测试 D3D Player 的优化效果,并比较优化前后的性能变化。
关键词:Direct3D Player;Direct3D Logger;显卡示驱动;性能优化
文章编号:1002-8331(2007)23-0097-03 **文献标识码**:A **中图分类号**:TP31

1 引言

D3D Logger 和 D3D Player 是为 Windows 系统平台下的 DirectX(由微软开发的一种图形应用程序接口,用以提高系统性能的加速软件)而开发的两个工具软件,Direct3D Logger(D3D Logger)作为截取所有 DirectX runtime[1]和驱动程序传送的指令和缓存内容的工具,存在于 DirectX runtime 和驱动程序之间,当播放要测试的文件时,D3D Logger 将每一帧指令生成一段 D3D Player 可以识别的脚本程序以及保存所有相关顶点数据、贴图内容,并以物理文件的形式保存下来,最后再将这些指令传送给真正的驱动程序。D3D Player 通过运行 D3D Logger 生成的脚本程序来重放 DirectX 渲染的整个过程,并且可以通过单步执行来观察 DirectX 渲染的每个细节。如果驱动程序存在缺陷,驱动程序开发人员可以通过修改这些脚本程序来改正驱动程序存在的错误。但是 D3D Player 的前期设计与实现并没有考虑到性能的问题,导致 D3D Player 在渲染一帧画面的时候通常需要很长时间,渲染 3D MARK(3D 图像处理系统的专用测试软件)2005 测试用例的时候,平均每帧在 300 s 以上,给显卡驱动程序的开发造成负面影响。如果运行更大帧的话,渲染时间将会占用几个小时,从而导致严重的后果,因为帧渲染时间间隔过长,将会导致一些 DirectX 测试工具产生超时异常,后续的帧将无法被渲染,因此对 D3D Player 性能的提高有着现实的意义。

2 D3D Player 系统瓶颈分析及优化方案提出

2.1 D3D Player 系统瓶颈分析

通过对 D3D Player 工作原理的研究,发现以下问题。

(1)D3D Logger 在记录的时候直接将 DirectX runtime 送下来的指令以纯文本文件的形式保存在磁盘上,书写规范遵循 Logger Script,D3D Player 以解释运行的方式处理 Logger Script 文件时,需要将指令字符串转化成对应的指令标识 ID。这种把指令转化过程放在运行时的机制会大大降低 D3D Player 的效率。同时在执行指令时,需要一个很大的 switch 结构来匹配指令标识 ID 与相应的处理函数,当有 n 条指令时,平均需要比较 $n/2$ 次,算法复杂度为 $O(n)$,这种比较耗去了大量的系统时间。

(2)D3D Logger 在记录的时候将所有 VB、IB[2]数据都以文本十六进制的形式存放浮点数据,而 D3D Player 在用到这些数据的时候,会将这些十六进制的浮点数据转化成真正的浮点

基金项目:国家自然科学基金(the National Natural Science Foundation of China under Grant No.70431001)。
作者简介:周宽久,副教授,博士,从事系统工程和嵌入式系统的研究。

数,这样也就产生了最大的系统性能瓶颈。另外每一帧的 VB、IB 也有大量数据是重复的,每一帧的前导帧和后继帧的三角形变化频率是比较低的,但是 D3D Player 对这些重复数据的辨别是无能为力的,当运行下一帧的时候,会重新读取 VB、IB,这会造成很大的资源浪费。采用 Intel VTune[3]性能测试工具进行实际测试,测试结果表明:Str2Int()函数占用 34%的 CPU 处理时间,memset()函数占用 14%的 CPU 处理时间。经过代码跟踪可以得出:Str2Int()用于将文本转化成整数,memset()用于一次性读取 VB、IB 数据。

通过以上分析发现,传统的 D3D Player 属于解释运行方式,当 D3D Player 进入运行时,即将 Logger Script 装入内存,随即执行脚本语言的翻译工作,其中包括词法分析、语法分析、数据转化等编译器所进行的一系列操作,这样势必导致执行时间过长。

2.2 D3D Player 优化方案

将原来的单一解释执行模式划分为编译子系统和执行子系统[4,5],对 Logger Script 进行编译,这样,指令字符串到指令标识 ID 的转换、Str2Int()所处理的数据的转换等工作只在编译期有,而执行期没有。在优化后的执行期,指令处理函数被单独地提取出来,指令处理函数的入口地址和指令标识 ID 做哈希映射。最后,将必须在执行时包含的文件,直接做内存文件映射[6],来减少对 memset()的调用。

3 优化的 D3D Player 的设计

3.1 编译子系统的设计

编译子系统首先打开每个 txt 文件,按照原来的 D3D Player 逐行处理方式,首先找到指令,并将指令转化为相应的标识符数值(ID),然后将随后的文本数据按照指令格式直接转化成相应的整型或者浮点型数值,由于整型或者浮点型数据长度在 32 位微机中都是 4 Byte,这样可以采用统一格式存放。如果遇到加载 VB、IB 指令,则编译子系统读取这些数据,随即将其从文本行格式转化为二进制流格式,紧排列在指令标识符之后。如果期间参数必须要用字符串类型,为了使字符串的 ASCII 码和指令标示符做出区分,按照字符串出现的先后顺序,将字符串统一按 ASCII 编码成字符串流,将其放在整个目标文件的前端,同时将它们按照出现的先后顺序进行全局编码,将此编码在指令标识符后的参数位置占位。字符串的处理类似于某些高级语言的编译,将初始化数据统一放到可执行镜像文件的前端。

连接的过程,首先对每一帧文件产生的数据结构进行长度累加,同时对所有字符串进行长度累加,将这些信息写入二进制文件的最前端的文件描述符,然后循环所有字符串结构,将其写入二进制文件,之后循环所有指令结构,也将其在字符串之后写入二进制文件。每一帧都有一个对应的编译过的二进制文件,执行子系统只需执行每个文件即可完成一套 D3D Script 的播放。编译子系统的流程如图 1 所示。

3.2 执行子系统的设计

执行子系统需要将每个指令所对应的函数入口地址在编译 Logger Script 时就写入一个哈希表,即以指令标识符与其对应函数入口地址做映射。执行子系统首先打开编译好的目标文件,读该文件最前端的头结构,从而定位字符串和指令以及数据的偏移量,将这些数据保存在执行子系统函数的局部变量

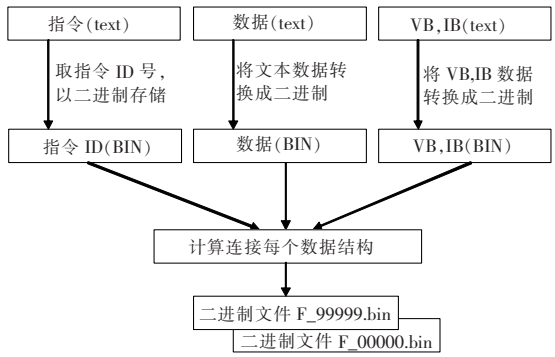

图 1 编译子系统

中,每一帧进行一次初始化变量的工作。在相关信息设置好后,执行子系统按照顺序读取指令标识符,并且按照指令要求的格式进行参数的填充,如果遇到字符串,则按照出现的先后顺序,从字符串开始偏移处依次读取所需要的字符串。对于 texture(用于保存图像背景及纹理的文件)文件,D3D Logger 在记录的时候原封不动地将它们存储成二进制文件位图形式,所以没有必要再被编译。texture 文件在执行子系统中首先被直接映射到虚拟内存中,当遇到加载 texture 文件的指令时,所做的操作和读取二进制文件的操作相同。当一条指令的所有相关工作完成之后,就调用相应的指令处理函数,指令处理函数地址将直接从哈希表中取出。然后应用指令处理函数对 SDK 函数进行参数填充,最后执行指令所对应的 SDK 函数。执行子系统的流程如图 2 所示。

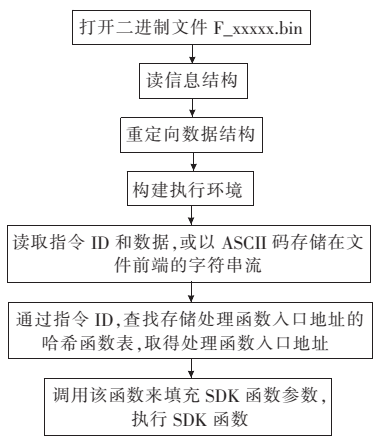

图 2 执行子系统

4 优化的 D3D Player 性能分析

4.1 空间复杂度分析

优化前 D3D Player 系统读取的是遵循 Logger Script 语法格式.txt,优化后 D3D Player 读取的是以整型或者浮点型数值存储.bin 文件。

(1)VB、IB 文件

优化前 VB、IB 是以十六进制文本形式存放的。如:

0x3f800000 0x3f800000 0x3f800000 0x3f800000

0x3f800000 0x3f800000 0x3f800000 0x00000000

每一行字符数为(length("0x3f800000")*4)+' '*3+2=45 Byte,其中每行包含 3 个空格和 1 个换行字符'0D0A'。优化后,进行编译每行 0x3f800000->INT=4 Byte*4+1=17 Byte,因为每 4 个参数占用 1 Byte 的参数位图。这样,一行数据可以平均节省 62.2%的磁盘存储空间,节省空间的同时也可以相应地节

省最少 62.2%的读盘时间。

(2)指令字符串

各个指令出现的几率各不相同,现列举出现几率高的几个指令来做计算。指令出现率:TRANSFORM-20%,VIEWPORT-20%,TRIANGLE-15%,INDEXBUFFER-10%,VERTEXBUFFER-10%,TEXTURE_LOAD-15%,其他 10%。当然每个 Logger Script 中,各种指令出现的几率有所不同,现做出大致计算:

"VERTEXSHADER_DECLARATION_CREATE"最长指令字符串;

"FOG"最短指令字符串;

Other 是平均指令字符串长度;

Other =(length (VERTEXSHADER_DECLARATION_CREATE)+length(FOG))/2=(31+3)/2=17

AVG 是平均指令字符串缩减,计算方法衡量了出现频率较高的几个指令以及其他剩余指令所占用的权重。

AVG = 100% - 2/(Length (TRANSFORM) * % 20 +
Length (VIEWPORT) * % 20+
Length (TRIANGLE) * % 15+
Length (INDEXBUFFER) * % 10+
Length (VERTEXBUFFER) * % 10+
Length (TEXTURE_LOAD) * % 15+Other*10%)=
1-2/(1.8+1.6+1.6+1.1+1.2+3+1.7)=83.3%

每个指令字符串都会被编译子系统转换成一个 2 Byte 的指令标识 ID。从计算结果可以得出平均节省 83.3%的空间。

通过上述分析发现优化的 D3D Player 使用编译子系统对操作文件进行编译,生成.bin 文件,占用更小的磁盘空间。同时,这也使 D3D Player 执行子系统在运行时节省了更多的时间。因为未优化的 D3D Player 运行时,消耗了大量的时间进行 VB、IB 文件的读盘和转化操作。

4.2　时间复杂度分析

优化前的 D3D Player 中影响性能的主要因素为 Str2Int()函数,而模式匹配,指令字符串到指令标识 ID 的转化,以及字符数字到整数的转换都要调用 Str2Int()函数实现,Str2Int()函数的算法复杂度达到了 $O(n^2)$。前面指出的从指令到指令处理函数的寻找,采用了很大的 switch 结构,其算法复杂度也达到了 $O(n)$。总体来说,整个回放算法的复杂度为 $O(n^2)$。

本文对 D3D Player 的优化采取了先编译、后执行的方法。这样,运行时节省了 Str2Int()函数执行的时间。同时,将指令处理函数的入口地址和指令 ID 做哈希映射,使得整个回放算法的算法复杂度降低至 O(1)。

另外,在读盘期间的 memset()函数也占用了大量的 CPU 时钟周期,本文对 D3D Player 的优化采用了"内存文件映射"的办法直接将物理文件映射为系统虚拟内存,节省了从磁盘调入系统缓存,又从系统缓存调入相应应用程序缓冲区的过程,而直接调入应用程序缓冲区,可以节省一半的读盘时间。

5　系统性能测试

显示卡采用 S3 公司 GPU 的 Delta Chrome S18 Nitro 显示卡(支持 Microsoft DirectX 9),显示内存为 256 MB DDR,采用 TSOP 封装的 3.3 ns 三星显存芯片。支持 AGP 8x,具备 8 条像素渲染管线,支持 Pixel/Vertex Shader 2.0+,核心及显存频率分别为 325/650 MHz。此卡属于 S3 公司系列显卡中终端显卡,在全系列显卡中性能比较稳定,但是图形运算能力一般,有助于衡量 D3D Player 在 S3 的整个产品线测试中的表现。

计算机采用 Intel Pentium 4 2.0 GHz 处理器,内存 512 MB,运行环境为 Windows XP Service Pack 2,带 Microsoft DirectX 9。显示终端设备为 19 寸 View Sonic G90f+,考虑到测试都为一般应用,所以分辨率为 1 024*768。编程环境为 Visual Studio 2003,VC++。编译环境为 DirectX 9 SDK,DDK(Device Development Kit,设备开发包,用以开发设备驱动程序)。

(1)测试环境 1:3D Mark2001 testing 1-3,3D Mark2003 testing 1-3,3D Mark2005 1-3。

分别测试 3D MARK 2001,3D MARK 2003,3D MARK 2005 这三个测试软件的 testing1-testing3 以 D3D Logger 所截取到的 Logger Script,测试同一个 Logger Script 分别由未优化的 D3D Player 和优化后的 D3D Player 来运行,测试结果以 Windows 系统时间戳差值为准,测试计算的时间精确到 0.1 s。得到的结论大致相同,优化使得 D3D Player 的平均执行时间缩小至原来的 1/3,也就是执行性能提高为原来的 3 倍。

表 1　3D MARK 测试结果表

3D Mark01	time/s	3D Mark03	time/s	3D Mark05	time/s
Testing1(old)	86.2	Testing1(old)	144.5	Testing1(old)	417.3
Testing1(new)	28.5	Testing1(new)	47.8	Testing1(new)	139.1
Testing2(old)	92.1	Testing2(old)	149.5	Testing2(old)	360.0
Testing2(new)	31.3	Testing2(new)	50.2	Testing2(new)	121.7
Testing3(old)	83.2	Testing3(old)	186.2	Testing3(old)	372.1
Testing3(new)	29.0	Testing3(new)	62.9	Testing3(new)	125.3
性能提高统计≈3 times		性能提高统计≈3 times		性能提高统计≈3 times	

(2)测试环境 2:Intel Vtune 4.6,对 D3D Player 进行热点测试,得到结果如图 3、图 4:

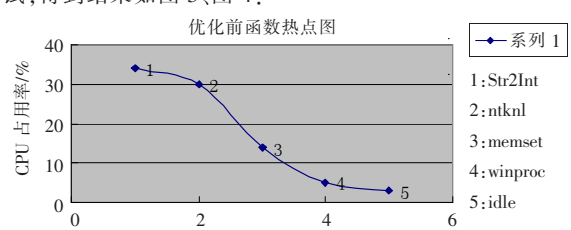

图 3　优化前函数热点图

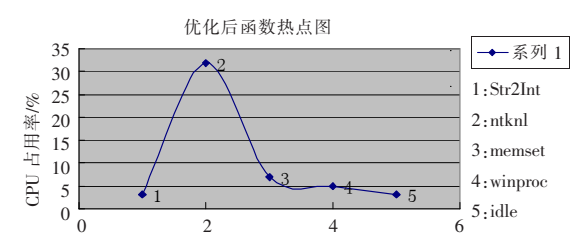

图 4　优化后函数热点图

6　结论

本文的工作在原有的 D3D Player 基础上采用编译 Logger Script 的机制来提高回放速度,将 D3D Player 分为编译子系统和执行子系统,经测试,D3D Player 优化解决了 DirectX 回放时间过长问题以及自动回放几百兆甚至几千兆字节 Logger Script 性能问题。

实验测试表明播放 3D MARK 2003-2005 的 testing1-testing3 情况下,S3 GamaChrome 系列卡平均速度达到 3 倍以上;

(下转 146 页)