深度报文检测中基于 GPU 的正则表达式匹配引擎*

王 磊, 陈曙晖, 苏金树, 许孟晋
(国防科学技术大学 计算机学院, 长沙 410073)

摘 要: 提出了一种基于 GPU 的正则表达式匹配引擎来加速深度报文检测中的模式匹配过程。该引擎基于 DFA 模型,在匹配时每一个 GPU 线程处理一个报文,通过大量的并行线程来提高引擎的吞吐量。基于 NVIDIA GeForce 9800GT GPU 的实验表明,该引擎处理实际网络报文时的吞吐量达到了 7.91 Gbps。

关键词: 深度报文检测; 模式匹配; 正则表达式; 图形处理单元
中图分类号: TP393.08 **文献标志码**: A **文章编号**: 1001-3695(2010)11-4324-04
doi:10.3969/j.issn.1001-3695.2010.11.090

GPU-based regular expression match engine for deep packet inspection

WANG Lei, CHEN Shu-hui, SU Jin-shu, XU Meng-jin
(*School of Computer Science, National University of Defense Technology, Changsha* 410073, *China*)

Abstract: This paper proposed a GPU-based regular expression match engine for deep packet inspection, which modeled as deterministic finite automaton (DFA) and assigned a single packet to each GPU match thread. Finally, experiments on NVIDIA GeForce 9800GT GPU achieved a maximum traffic processing throughput of 7.91 Gbps using real network traces.
Key words: deep packet inspection(DPI); pattern matching; regular expression; GPU

深度报文检测(DPI)是网络安全系统(如入侵检测系统、防火墙等)中的关键技术。安全应用捕获网络上所传输的报文,并使用预先定义的模式对报文头以及报文有效载荷进行匹配。基于正则表达式的模式匹配技术是 DPI 的核心技术,它的性能是整个 DPI 系统性能的瓶颈。

正则表达式的匹配一般根据有限自动机理论,先将正则表达式转换成等价的非确定性有限自动机(nondeterministic finite automaton,NFA),直接基于 NFA 进行匹配或将 NFA 转换为等价的确定性有限自动机(deterministic finite automaton,DFA),然后基于 DFA 进行匹配。基于 NFA 的匹配中,由于状态转移的不确定性,算法效率低,但 NFA 的状态表比较小;基于 DFA 的匹配中,一次处理一个字节,匹配速度相对较快,但状态表很大。在 DPI 中,一般采用速度更快的 DFA 方法进行模式匹配。

基于 DFA 进行模式匹配时每次处理一个字节,匹配速度由系统的访存速度决定。由于网络带宽的迅速增长,传统基于软件的 DFA 方法速度上已经很难满足应用的需求。因此,研究人员转向在特定硬件结构平台上实现 DPI 中的模式匹配系统,如 ASIC[1,2]、FPGA[3,4]和 TCAM[5]以及 NP (network processor,网络处理器)[6,7]等。利用特殊的硬件系统结构,虽然系统的性能和效率得到了提升,但是整个系统的软硬件协同设计代价太高,并且适用性和可升级性较差。

随着实时、高分辨率 3D 图形渲染需求的增长,GPU (graphics processing unit, 图形处理单元)已经发展成为具有巨大计算能力的高并行度、多线程、高通信带宽的多核处理器[8]。由于在模式匹配的过程中主要是访存操作,如果能很好地利用 GPU 多线程并行处理隐藏访存延迟,系统的性能将且有很大的提升。在 GPU 上实现正则表达式匹配引擎,与基于特定硬件系统架构实现相比,具有性能高、可升级性好以及代价小等特点,因而成为近年来研究的热点。

1 相关研究

Smith 等人[9]在 NVIDIA G80 GPU 上设计实现了模式匹配原型系统。作者首先说明了 GPU 应用在模式匹配领域的机遇与挑战,定量分析了原型系统实现中关于 GPU 存储器模型、控制流,以及并发性的设计考虑,研究了 G80 的硬件组织,以及原型系统的软件实现,并给出了优化方案。在实验中,作者分别针对 XFA[10]以及传统的 DFA 进行实验,性能与 Pentium 4 CPU 相比分别得到了 6 倍和 9 倍的提升。

Jacob 等人[11]基于 Cg[12]编程平台将入侵检测系统中的模式匹配工作移植到 NVIDIA 6800 GT GPU,设计实现了入侵检测系统 PixelSnort。通过实验对 Snort 和 PixelSnort 进行了比较,结果表明随着 CPU 负载的增加,PixelSnort 相对更加健壮并且比 Snort 的效率提高了 40%。

Vasiliadis 等人[13]设计的高性能网络入侵检测系统 Gnort,基于 CUDA[8,14]编程平台将 Aho-Corasick[15]算法移植到了 NVIDIA G80 GPU 上。实验结果表明,使用生成报文进行测试时 Gnort 在最好的情况下得到了 2.3 Gbps 的吞吐量,而在实际的网络环境中运行时其吞吐量也要比 Snort 高两倍。

Onsjö 等人[16]对传统 NFA 算法进行了并行扩展并将其移植到 GPU 上。分别在 AMD Opteron 2.4 GHz CPU 和 NVIDIA T1 GPU 平台对果蝇 DNA 序列搜索算法进行实验,最终 GPU 得到了 50 多倍于 CPU 的加速比。

收稿日期: 2010-05-08; **修回日期**: 2010-06-24 **基金项目**: 国家"863"计划资助项目(2009AA01A346)

作者简介:王磊(1985-),男,陕西西安人,硕士研究生,主要研究方向为网络安全(wangleinuts@gmail.com);陈曙晖(1974-),男,湖南益阳人,副研究员,博士,主要研究方向为网络安全;苏金树(1962-),男,福建莆田人,教授,博导,主要研究方向为计算机网络、网络安全;许孟晋(1978-),男,陕西西安人,硕士研究生,主要研究方向为网络安全.

Wu Cheng-kun 等人[17]将入侵检测系统中的模式匹配工作移植到多核 CPU 以及多核 GPU,提出了一种混合并行模式匹配方法 HPSMM,并提出了针对模式集以及针对文本的负载均衡策略。GPU 被用做 CPU 的协处理器,负责文本匹配工作,在 NVIDIA GeForce 9800GTX + GPU 和 Intel Core(2) 2.4 GHz Dual CPU 上进行实验,得到了 7 倍的加速比。

上述工作中除文献[9,16]外,其他都是针对字符串匹配,不适合基于正则表达式的深度报文检测。文献[9]是针对正则表达式且给出了基于 XFA 和 DFA 的实现,但并没有给出具体的系统设计细节。文献[13]中虽然给出了存储器使用方案,但其工作针对字符串匹配且基于 NFA 模型,速度相对较慢,其设计不适合 DFA 实现。

在本文中,提出了一种基于 GPU 的 DFA 正则表达式匹配引擎结构,并根据 GPU 体系结构以及编程模型特点对并行匹配算法进行了优化。最后,实验基于 NVIDIA GeForce 9800 GT GPU 平台,并对实验结果进行了对比分析。

2　基于 GPU 的正则表达式引擎

2.1　NVIDIA GPU 简介

目前 GPU 的生产商有 AMD/ATI、NVIDIA 和 Intel 等,各品牌 GPU 的体系结构大同小异,其中,NVIDIA GPU 的开发模型 CUDA[8,14]最容易充分发挥 GPU 强大的计算能力,在实现时本文采用了 NVIDIA 公司的 GPU。下面以 NVIDIA G80 GPU 为例,简单介绍 GPU 的体系结构。

G80 GPU 包含 16 个 SM (stream multi-processor,多核流处理单元),每个 MP 包含 8 个 SP(stream processor,流处理单元),2 个 SFU (special function unit,特殊功能单元)以及 16 KB 片上共享存储器。CUDA 是 NVIDIA GPU 的编程平台,它是支持通用 GPU 计算的并行编程模型,可以使程序员更高效地利用 GPU 资源。在 CUDA 中,线程被组织成 block 的形式, block 可以是一维、二维或三维,目前每个 block 含有的线程总数不能超过 512 个。Block 又被组织成 grid,grid 也可以组织成多维。运行时,一个或多个 block 被分配到一个 MP 上执行。同一个 block 中的线程可以通过共享存储器进行通信,也可以进行栅栏同步。

G80 中包含多种存储器,包括寄存器、全局存储器(global memory)、常数存储器(constant memory)、纹理存储器(texture memory)、共享存储器(shared memory)等。这些存储器大小速度各异,是影响 GPU 应用性能的重要因素。其中,寄存器是 GPU 的片上高速缓存器,访问延迟为 1 个时钟周期,每个 SM 包含 8 192 个寄存器文件,每个寄存器文件的大小是 32 位。各种存储器的具体介绍[15]如表 1 所示,其中存储器的访问特性是指 GPU 线程访问存储器时的约束。

表 1　GPU 中的存储器

类型	大小	位置	缓存	访问延迟	访问特性	备注
全局存储器	768 MB	片外	无	400 ~ 600 个周期	可读写	进程在访问时应满足合并访问要求
共享存储器	16KB/SM	片内	无	1 个周期	可读写	同一个 block 中的线程共享,被组织成 16 个 bank
常数存储器	64 KB	片外	有	缓存命中则需要 1 个周期,否则 400 ~ 600 个周期	只读	每 SM 拥有 8KB 缓存,用于存放查找表等只读信息
纹理存储器	128 MB	片外	有	同上	只读	每两个 SM 有 16 KB 缓存

2.2　基于 GPU 的并行匹配引擎

为了提高匹配速度,必须最大限度地使匹配任务并行化。在匹配时,多个 GPU 线程并行执行,每个线程独立完成相应的任务,并将匹配结果上传至 GPU 全局存储器中。当所有线程匹配结束后,CPU 从 GPU 全局存储器中下载匹配结果。

引擎首先将正则表达式集编译成一个 DFA,然后将 DFA 的状态表(二维数组的形式)绑定到 GPU 纹理存储器而将报文加载至全局存储器。在匹配时虽然访存密集但相对集中,而纹理存储器有加速缓存,所以访问速度很快。在匹配过程中需要对报文逐字节处理,全局存储器的访问开销太大,如果不能满足合并访问的条件,性能会更低。可行的方法是利用线程共享存储器对报文进行缓冲进而使线程对全局存储器的访问满足合并访问约束。匹配引擎的整体结构如图 1 所示。

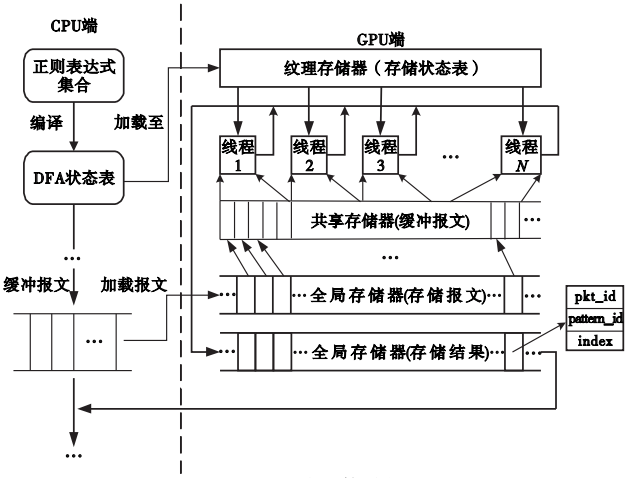


图1　引擎整体结构

基于 GPU 的引擎中,GPU 被用做 CPU 的协处理器只负责匹配工作,而 CPU 负责正则表达式的编译、被匹配文本的收集、DFA 状态表和报文从内存向 GPU 存储器的加载,以及从 GPU 全局存储器下载匹配结果。匹配结果由报文 ID、命中的正则表达式 ID 以及报文中命中位置索引组成。

GPU 线程的负载分配涉及到 GPU 的负载均衡[17]。显然,将任务均分给每一个线程是最合理的均衡策略。因为 GPU 中同一时间在每一个 MP 上运行的所有线程执行相同的指令,当任务均等时,如果不考虑分支因素,所有线程在执行结束时不会有等待。在基于 GPU 的方案中,由于现实网络中的报文长度不一,在匹配之前需要对报文进行预处理,使所有报文的有效过载长度相同,使所有线程负载相同。

基于 GPU 的匹配引擎中存在两个层次的并行:block 之间的并行和 block 内线程之间的并行。为了使应用能获得更高的效率,block 的数目应是 GPU 中 SM 数目的 2 倍以上,使 GPU 各 SM 满负荷运行。同时,为了隐藏存储器延迟,提高并行性,block 内线程的数目在满足硬件资源约束(如寄存器、共享存储器等)的前提下也应尽可能地大。

2.3　并行匹配算法

DFA 的状态表是一个 $m \times 256$ 阶矩阵(二维数组,m 代表 DFA 的状态个数),矩阵的第 i 行第 j 列的值表示在状态 i 输入字符 j 时的状态机将转移到的下一个状态。在 CPU 上单线程匹配时,对于要匹配文本的每一个字符,通过“当前状态基地址 + 输入字符”一次访存就可得到下一个状态的地址,复杂度为 $O(n)$,n 为被处理文本的长度。基于 GPU 并行匹配,匹配任务被分配给大量的运行线程,算法有很大的不同。算法 1 是

并行匹配的流程。

算法 1　并行匹配算法

```
#define M 64
#define N 128
#define T 16
procedure MatchKernel(unsigned char * d_text, result_t * d_result)
begin
1    __shared__ unsigned char s_text[M][N+4];
2    state_t current_state, next_state;
3    current_state = 0;
4    for integer times = 1 to T do
5      load N bytes payload form global memory to shared memory
6      for integer index = 0 to N do
7        char input = s_text[threadIdx.x][index];
8        next_state = tex2D ( texref, current_state, input );
9        current_state = next_state;
10         check if current _state is an acceptable state
11       end for
12   end for
end
```

算法中一个 block 共有 M 个线程,一个线程分配的共享存储器的大小是($N+4$) Byte,共申请($M\times(N+4)$) Byte 共享存储器空间,而目前一个 SM 共有 16 KB 的共享存储器,因此 $M\times(N+4)$应小于 16 KB。从第 4 行开始循环 T 次,一次循环处理 N 个字节,一个线程共处理 $T\times N$ 个字节,所以每个报文的长度都应不大于 $T\times N$。很显然,在每个线程任务一定的情况下,T 越大则 N 越小,进而 M 可以增大即可以使 block 内的线程数目增多。

共享存储器被组织成 16 个 bank,每个 bank 的带宽是 4Byte,因此 N 必须是 64 的倍数。之所以多申请 4Byte,是为了使线程在访问共享存储器时避免 bank 冲突。

6~11 行是 DFA 的匹配过程。第 7 行得到 DFA 下一个输入,然后根据当前状态和输入字符得到下一个状态;第 10 行判断该状态是否为接受状态,如果是接受状态则记录当前的报文 ID 以及被匹配上的正则表达式在规则集中的 ID。

GPU 中没有分支预测单元,因此,分支指令将降低算法的性能。第 4 行和 6 行循环中,T 和 N 是常数,编译前已经确定并且在编译时 CUDA 会进行循环展开,因此不存在分支的情况。第 10 行会产生分支但这是可以接受的,因为实际中匹配的命中率比较低,产生分支的概率比较低。如果只记录转移后的状态,由 CPU 来判断是否为接受状态,这样虽然可以避免分支,但对每个字节,GPU 都会有一次写操作,系统的性能将受到很大影响。

3　性能测试

3.1　实验设置

实验中正则表达式测试集选择入侵检测系统 Bro[18] 中的规则集,其中包含 227 个正则表达式,编译成 DFA 后共有 6 533 个状态,每个状态共有表项 256 个占 1 KB 空间,总的状态表的大小约为 6.4 MB。

NVIDIA GeForce 9800GT GPU 包含 14 个 SM,每个 SM 上有 8 个 SP,主频 1.51 GHz。它的显存大小为 512 MB,纹理存储器为 128 MB,每 SM 的共享存储器大小为 16 KB,它的体系结构与 G80 GPU 相似。

每个线程处理一个报文,共处理 32 768 个报文。引擎一次同时运行 4 096 个线程,处理完所有报文需要运行 8 次。同时,性能的计算只针对匹配工作,不包括报文的加载以及必要的预处理时间,也不包含结果的处理时间。

3.2　结果分析

测试随机生成的报文。试验中每个报文的负载是随机生成的字符串,且长度相同。文献[19]中给出了 AC[15] 算法在 Intel Core(2) 2.4 GHz Dual CPU 平台上匹配的吞吐量约为 0.501 Gbps,以及该算法在 NVIDIA GeForce 9800GTX + GPU 上的吞吐量约为 3.79 Gbps。因为 AC 算法是基于 DFA 的,可以用它作为对比来说明 CPU 上的 DFA 算法效率。同时,文献[19]中被匹配文本也是随机产生的,因此与本方法的吞吐量具有很强的可比性。

图 2 给出了本文方法与文献[19]中方法的性能比较。可以看出,本文方法相对于 CPU 得到的加速比大约是 18,而相对于文献[19]中的 GPU 方法的加速比约为 2.4。

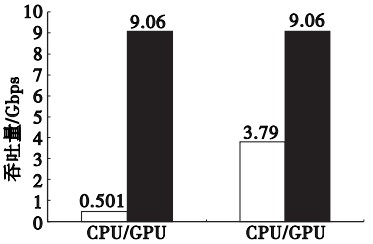


图2　与其他方案的性能比较

测试实际网络中的报文。使用 Wireshark[20] 软件在 Internet 环境下捕捉 65 536 个报文,再通过使用 WinPcap[21] 开发包编程来读取报文的数据。由于现实网络中的报文长度不同,为了提高匹配性能,需要在报文末尾填充空字符。通过实验,该引擎的吞吐量达到了 7.91 Gbps,较随机产生的报文性能有所下降的原因是,匹配命中率的提高使匹配时分支的概率提高的同时,往全局存储器写结果也将影响性能。

与基于 CPU 的方法相比,GPU 的实现方法使正则表达式匹配速度提高了约 16 倍。基于 GPU 的应用中,关键的步骤是对程序的优化。在 CUDA 环境下,对应用的优化除了对算法的优化外,主要体现在对存储器访问的优化、设备与主机通信优化、运行配置优化、指令优化以及线程的负载均衡等[14]。

4　结束语

本文基于 GPU 高并行多线程体系结构提出了一种新的高速正则表达式引擎。该引擎具有速度快、可扩展性好、成本低等特点,其不足是支持的正则表达式的数目受到存储器硬件的限制。随着 GPU 的发展,当 GPU 的存储器容量增大、访问速度提高时,正则表达式匹配引擎将会有新的设计思路,如可以将每一个表达式独立编译成 DFA 并使用单一独立的线程并行匹配所有报文,等等。DFA 状态表爆炸问题直接制约着匹配引擎支持的正则表达式的数目,同时状态表的增大会导致访存分散,缓存的命中率也就越低。下一步的工作主要是研究基于 GPU 的简化状态表的匹配引擎,使其支持的正则表达式数目更多,同时提高性能。

参考文献:

[1]　Regular expression processor[EB/OL]. http://www.titanicsystems.com/pdf/products/1.pdf.

[2] BRODIE B C, TAYLOR D E, CYTRON R K. A scalable architecture for high-throughput regular-expression pattern matching[J]. SIGARCH Comput Archit News,2006,34(2):191-202.

[3] SIDHU R, PRASANNA V K. Fast regular expression matching using FPGAs[C]//Proc of the 9th Annual IEEE Symposium on FCCM. Washington DC:IEEE Computer Society,2001:227-238.

[4] 孙志刚,张子文. 正则表达式匹配的高效硬件实现[J]. 计算机工程与科学,2009,31(10):5-8.

[5] YU Fang, KATZ R H, LAKSHMAN T V. Gigabit rate packet pattern-matching using TCAM[C]//Proc of the 12th IEEE International Conference on Network Protocols. Washington DC:IEEE Computer Society,2004:174-183.

[6] YU Jian-ming, LI Jun. A parallel NIDS pattern matching engine and its implementation on network processor[C]//Proc of International Conference on Security and Management. Las Vegas:CSREA Press, 2005:375-381.

[7] LIU R Tong-tai, HUANG Nen-fu, KAO C N, *et al*. A fast pattern-match engine for network processor-based network intrusion detection system[C]//Proc of International Conference on Information Technology: Coding and Computing. Washington DC:IEEE Computer Society,2004:97-101.

[8] CUDA programming guide[EB/OL]. http://www.serc.iisc.ernet.in/ComputingFacilities/systems/Tesla_Doc/NVIDIA_CUDA_Programming_Guide_2.3.pdf.

[9] SMITH R, GOYAL N, ORMONT J, *et al*. Evaluating GPUs for network packet signature matching[C]//Proc of International Symposium on Performance Analysis of Systems and Software. 2009.

[10] SMITH R, ESTAN C, JHA S. XFA: faster signature matching with extended automata[C]//Proc of IEEE Symposium on Security and Privacy. Los Alamitos:IEEE Computer Society,2008:187-201.

[11] JACOB N, CARLA C. Offloading IDS computation to the GPU[C]//Proc of the 22nd Annual Computer Security Applications Conference. Washington DC:IEEE Computer Society,2006:371-380.

[12] MARK W R, GLANVILLE R S, AKELEY K, *et al*. Cg:a system for programming graphics hardware in a C-like language[J]. SIGGRAPH,2003,22(3):896-907.

[13] VASILIADIS G, ANTONATOS S, POLYCHRONAKIS M, *et al*. Gnort: high performance network intrusion detection using graphics processors[C]//Proc of the 11th International Symposium on Recent Advances in Intrusion Detection. Berlin, Heidelberg:Springer-Verlag, 2008:116-134.

[14] NVIDIA CUDA C programming best practices guide[EB/OL]. http://www.serc.iisc.ernet.in/ComputingFacilities/systems/Tesla_Doc/NVIDIA_CUDA_Best_Practices_Guide_2.3.pdf.

[15] AHO A V, CORASICK M J. Efficient string matching: an aid to bibliographic search[J]. Communications of ACM,1975,18(6):333-340.

[16] ONSJÖ M, AONO Y. Online approximate string matching with CUDA [EB/OL]. (2009-03-23)[2009-12-18]. http://odinlake.net/wordpress/wp-content/uploads/2009/03/pattmatch-report.pdf.

[17] WU Chen-kun, YIN Jian-ping, CAI Zhi-ping, *et al*. A hybrid parallel signature matching model for network security applications using SIMD GPU[C]//Proc of the 8th International Symposium on Advanced Parallel Processing Technologies. Berlin, Heidelberg:Springer-Verlag,2009:191-204.

[18] Bro intrusion detection system[EB/OL]. http://www.bro-ids.org.

[19] 吴诚　. 面向网络预警的并行模式匹配方法研究[D]. 长沙:国防科学技术大学,2009.

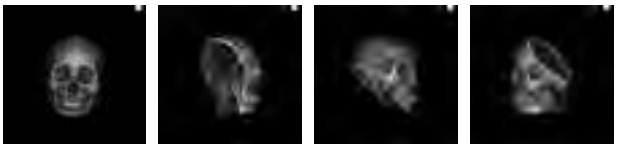
[20] Wireshark[EB/OL]. http://www.wireshark.org.

[21] WinPcap[EB/OL]. http://www.winpcap.org.

[22] KUMAR S, DHARMAPURIKAR S, YU Fang, *et al*. Algorithms to accelerate multiple regular expressions matching for deep packet inspection[C]//Proc of Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications. New York: ACM Press,2006:339-350.

(上接第 4323 页)512×132),运用 VTK 中的合成体绘制算法完成三维重建,然后进行平面切割,部分结果如图 4 所示。

图 4(a)为待切割的颅骨模型;(b)为初始平面绕 x 轴旋转 45°后切割的结果;(c)为平面绕 y 轴旋转 45°后的切割结果;(d)为将初始平面绕空间向量(1,1,1)旋转 45°后的切割结果。



(a)　(b)　(c)　(d)

图4　平面切割结果

3　结束语

本文配合重建的三维模型,提出了基于标记点的三维表面模型切割以及基于平面的三维体模型切割的方法。在基于标记点的三维表模型切割中,采用了插值的方法来模拟切割轨迹,并可以在三维空间对标记点进行交互操作;在基于平面的体模型切割中,利用四元数的旋转方法减少了切割平面任意旋转所耗费的计算时间,实现了对三维体模型的灵活切割。实验证明,通过扩展 VTK,利用标记点或者平面对三维模型进行切割高效、实时,在具体实现方式上也比基于底层库(如 OpenGL)方式更为简单,可方便地应用在虚拟外科手术中,具备较强的应用价值。

参考文献:

[1] 王满宁,韩正之,宋志坚. 三维医学图像的快速重建和任意切割[J]. 复旦学报:医学版,2002,29(2):142-144.

[2] SCHROEDER W. The VTK user's guide[M]. New York: Kitware, 2001:96-97.

[3] 王晓强,周明全,耿国华. 一种三维颅骨表面标志点平滑移动的方法[J]. 计算机工程,2004,30(17):61-62.

[4] 刘奇,周明全,刘晓宁. 一种基于标志点的三维表面模型的切割方法[J]. 计算机应用研究,2007,24(2):163-164.

[5] 刘力强,周明全,耿国华. 一种平行透视下的三维拾取方法[J]. 西北大学学报,2002,32(1):39-42.

[6] SCHNEIDER P J, EBERLY D H. 计算机图形学几何工具算法详解[M]. 周长发,译. 北京:电子工业出版社,2005:354-357.

[7] 郭圣文. 利用链表对三维物体实现多平面剪切[J]. 计算机应用,2006,26(B06):54-55.

[8] 王勇,马立元,王忠强. 四元数法在计算机图形中的应用[J]. 军械工程学院学报,2001,13(2):48-51.