一种光线与三角形求交算法的硬件架构设计①

赵男男

(广东海洋大学 寸金学院，湛江 524094)

摘 要：为了实现实时性的光线追踪平台，提出了一种光线与三角形求交算法的硬件架构设计。首先介绍了一种光线与三角形求交的简洁算法。该算法与其它算法相比使用存储空间最少，却具有相似的性能，便于硬件实现。根据此算法，提出了相应的硬件架构，在架构设计过程中，通过折叠、资源共享、以及多线程等硬件架构设计方法来提高硬件使用率，使有限的硬件资源达到最高的性能。实验结果表明，与现有方案相比，本文硬件架构平台在速度和芯片面积两个方面都存在着较大的提高。为实时光线追踪的实现提供了依据。

关键词：光线跟踪；光线与三角形求交；硬件架构

Hardware Architecture Design of Ray-Triangle Intersection

ZHAO Nan-Nan

(Cunjin College, Guangdong Ocean University, Zhanjiang 524094, China)

Abstract: For interactive ray tracing, this paper proposes a hardware architecture design of ray-triangle intersection. First of all, a minimum storage ray triangle intersection was introduced. The algorithm uses minimum storage space compared to other algorithms, but with similar performance and convenient for hardware implementation. According to this algorithm, this paper presents the hardware architecture, several hardware design techniques such as multi-threading, folding and resource sharing are used to increase the hardware utilization. Experimental results show that this hardware architecture platform has a wider improvement in terms of speed and chip area, which was compared with other platforms, which provides an effective solution for interactive ray tracing.

Keywords: ray tracing; ray-triangle intersection; hardware architecture

1 前言

利用算法的并行运算，在超大型计算机上，交互式光线追踪早已实现。然而，基于 CPU 或者是网络连接的交互式光线追踪仍然不理想。许多方法都是在单个芯片上[1]将光线追踪应用映射到一个多处理器结构。虽然都没有实现，但最后还是给出了一个对它实时性的模拟仿真。时至今日，对它的完全实现变得越来越有可能，当前的图形学硬件是一个很好例子。因为它强大的可编程性使得它能够实现很多效果，同时还能够提供非常大的带宽和浮点计算，可以将其应用到光线追踪系统中。实际上，已经有一些文献中都提到了这个研究[2,3]，但结果并不比 CPU 实现效果好，主要是因为编程模块是有限制的，另外缺少类似于堆栈的一些特殊数据结构。后来，IBM 开发的名为 CELL 的功能更为强大的多核处理器被成功应用于实现光线追踪算法，性能非常接近于实时。然而，上述工程仍然属于软件实现，并不能非常有效率地应用于所选的硬件，而光线追踪系统的定制硬件架构在过去就有文献提出，不过大多数以体模型为研究目标[4]。虽然现存很多种硬件架构设计，但大多数都没有全面考量面积和时间问题。本文则设计了一种硬件架构达到了预期的目的。

2 求交算法简介

光线和三角形的求交计算是光线跟踪的核心部分。以 O 为原点的光线可以表示为：

$$R(t) = O + tD \quad (1)$$

其中 D 为标准化的方向向量。三角形的顶点分别

① 收稿时间:2010-05-31;收到修改稿时间:2010-07-25

为V_0,V_1,V_2。在光线三角形求交问题中，本文采用文献[5]中算法，该算法具有最小存储特性(只需要存储三角形的顶点)，并且不需要任何处理。对于三角形网格，使用最小存储可以节省大量的空间，并由于相邻三角形的顶点可以共享，其实际存储顶点大约在 25%至50%之间。下面介绍该算法的具体过程：

给定三角形上一个点$T(u,v)$，且

$$T(u,v)=(1-u-v)V_0+uV_1+vV_2 \tag{2}$$

其中，(u,v)是三角形的重心坐标，满足$u\geq 0,v\geq 0$且$u+v\geq 1$。在光线$R(t)$和三角形$T(u,v)$的交点处$R(t)=T(u,v)$，因此有：

$$O+tD=(1-u-v)V_0+uV_1+vV_2 \tag{3}$$

整理可得：

$$\begin{bmatrix}-D & V_1-V_0 & V_2-V_0\end{bmatrix}\begin{bmatrix}t\\u\\v\end{bmatrix}=O-V_0 \tag{4}$$

通过解上述线性方程组可以得到三角形的重心坐标和光线原点到交点的距离t。上述过程也可看作是将三角形的一个顶点移动到原点，并且将它变换成单位三角形的几何变换，如图 1 所示。其中，$M=\begin{bmatrix}-D & V_1-V_0 & V_2-V_0\end{bmatrix}$。这一阶段称之为预处理阶段。

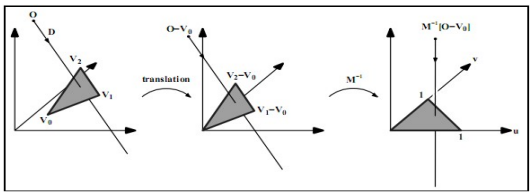

图 1 三角形的几何变换

在式 (4) 中，令$E_1=V_1-V_0$，$E_2=V_2-V_0$，$T=O-V_0$。利用克莱姆法则，有：

$$\begin{pmatrix}t\\u\\v\end{pmatrix}=\frac{1}{P\cdot E_1}\begin{pmatrix}Q\cdot E_2\\P\cdot T\\Q\cdot D\end{pmatrix} \tag{5}$$

其中，$P=D\times E_2$，$Q=T\times E_1$。可以看出，在式（5）中存在有 4 个内积和 2 个叉积，这个阶段被称作是核心计算阶段。而把用除法器得到标准化的t、u和v的阶段称作是标准化阶段。最后利用条件$u\geq 0,v\geq 0$且$u+v\geq 1$来检验光线与三角形是否相交。

应用这种方法不仅能降低外部带宽，而且还能增加超高速缓存命中率，因而选用此方法进行硬件架构设计。

3 硬件架构设计

在硬件架构过程中，首先最需要关注的问题是带宽和片上存储器。本文求交算法的硬件架构流程如图 2 所示。

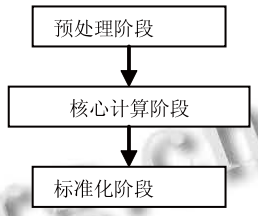

图 2 求交算法的硬件架构流程图

其中，每个阶段的硬件设计图分别如图 3、图 4 和图 5 所示。

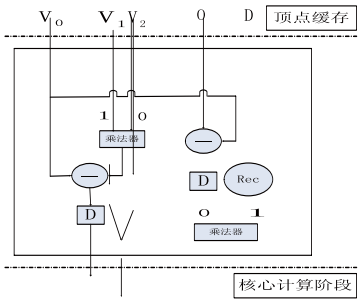

图 3 求交单元预处理阶段的硬件设计图

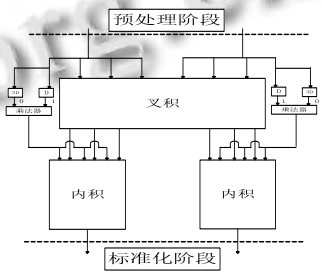

图 4 求交单元核心计算阶段的硬件设计图

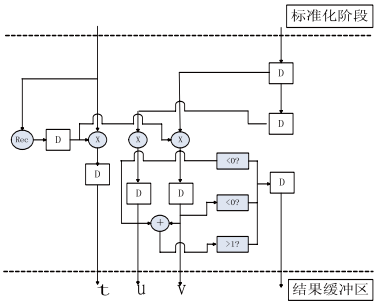

图 5 求交单元标准化阶段的硬件设计图

硬件架构过程中，用到了 24 个乘法器、23 个加法或减法器，以及 6 个除法器。

通常采用直管线以达到高性能的目的，而在许多方法中则是利用对硬件资源的有效分配来达到目的。研究发现，在这些方法中，其遍历单元和求交单元的硬件数量是不一样的，而且这两个单元的吞吐量也都是为了保证硬件资源的利用率尽可能高而有所差异。根据保守估计，选择两单元中的关键一个的吞吐比来实现避免性能退化和过载的不平衡。也就是说，遍历单元的吞吐量是一周一次，而求交单元则是两周一次。因而采用了如图 3、4、5 所示的架构设计技术。

在核心计算阶段，由最初的 4 个内积、2 个叉积减少到 2 个内积和 1 个叉积。叉积和内积模块的设计图分别如图 6、图 7 所示。图中所示乘法器是根据奇偶时钟周期来选择数据的。另外，求交算法中的 6 个浮点分配器并没有出现在图中。实际上，浮点分配器具有面积无效性和所需延迟时间长等特点。即使是在现在的高端 GPU 设计都不容易完成这样的模块设计。相反，根据泰勒展开式，利用查询表和一些数学运算来估算上述相互操作。利用这种方法不仅能够保证浮点的有效性，而且能保证它的精确性。

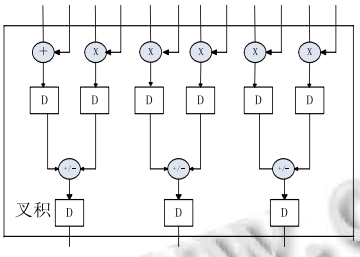

图 6　核心计算阶段的叉积模块设计图

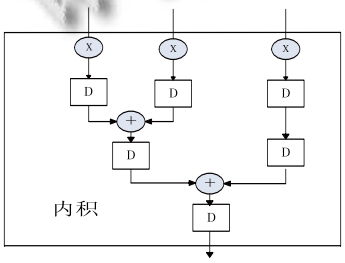

图 7　核心计算阶段的内积模块设计图

在图 3 和图 5 中出现的 REC 标志是倒数运算.最后在得到重心坐标，和距离后，将最终结果移动到结果缓冲单元中去。

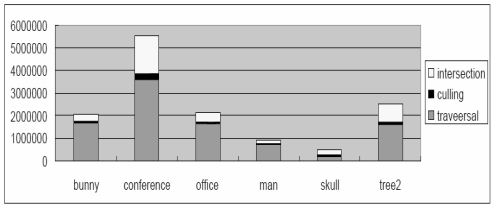

图 8　遍历、求交和剔除的计算比

至此，对本文算法有利的剔除体的硬件设计还没有被提及。因为它的实现无法独立完成，只有借助于求交单元才能实现。如图 8 所示，可以看到遍历和求交单元的比率是不一样的，而且剔除体的量与其他两个操作相比是非常少的。因此在这种情况下，在一个独立的模块上来实现它则会导致硬件资源的浪费。而且利用平面方程的剔除体非常适合共享求交单元。这是因为在剔除体的核心计算阶段，其平面方程和端点之间有 4 次内积运算，而且需要 12 次乘法运算。反过来说，一次叉积运算、两次内积运算相当于 12 次乘法运算。改进的求交单元的架构如图 9、10 所示。在这里仅仅对内积和叉积模块进行了改进。为了共享这两个模块，加入了一些控制信号和浮点加法器。与图 7、8 相比，多了的部分则就是为剔除体加入的部分。

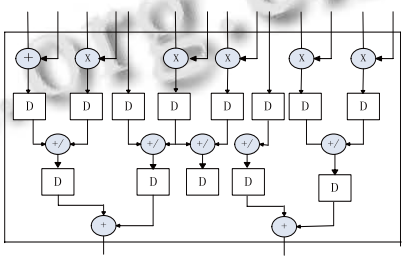

图 9　改进的叉积模块

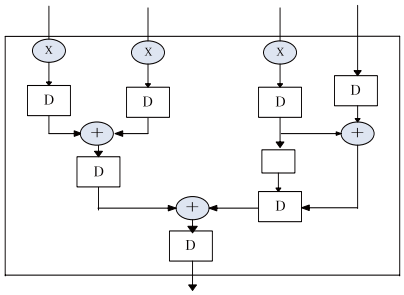

图 10　改进的内积模块

4 实验及分析

如表 1 所示，会议场景下的帧数是六种场景中最差的。虽然会议场景的多边形计数器是最大的，但他的帧数降低速率比大些的多变形计数器的要小一些。因此，光线追踪的对数复杂性是真实有效的。另外，尽管兔子场景有比办公室场景有更小一些的三角形计数器，但它的帧数要比办公室场景的大一些。也就是说，从表 1 中可以得出，由于兔子场景更趋向于均匀分布，所以多边形的分布比多边形计数器显得更为重要一些。而对于带宽，尽管是用了小的高速缓存，但带宽仍然很小，其范围为 70~450MB/s。而现如今像 CELL 和 GPU 一样的高端 VLSI 设计都有着高于 40GB/S 的带宽容量。因此，几十个光线追踪单元可连接到固定的系统总线上并行运行。

表 1 六种场景的实验结果 （帧数/秒）

场景	办公室	会议	兔子	人体	树	头骨
每秒帧数	22.69	12.76	24.65	36.46	27.48	58.43
带宽（MB/s）	121.39	254.17	249.03	188.97	425.24	78.95

表 2 不同平台的芯片面积

平台	工艺	面积	标准面积
本文算法	130nm	2.89	2.89
ASIC	130nm	13.58	13.58
CELL	90nm	221	464.4
GPU1	90nm	484	1009.2
GPU2	90nm	353	736.5
OpenRT	90nm	135	281.67

本文硬件实现与其他平台的比较如表 3 所示。但是由于不同平台具有不同的结构，所以很难进行比较。本文进行了公平、公正的比较。其中，VLSI 设计有三个主要的因素需要特别注意，这三个因素包括面积、定时和电源。标准化这些因素，显示其真正的性能才可进行公平的比较。而电源在非嵌入式平台中是一个非常重要的因素。因此，尽管本文工程的电源优于其他平台，但并不包括在比较范围之内[6]。又由于定时因素要牵涉到单元库和架构设计技术，所以它也不在比较范围之内。而且要分别出它们真的很难。因此，我们仅用面积因素来做标准化。

换而言之，本文选择的比较参数是每平方毫米祯速度。这些平台的帧大小都不尽相同。本文采用的帧大小为 512*384。而由于平台需要更大的相关性来得到更好的性能，所以这对本文所用平台并没有优势。另外，本文没有硬件实现底纹描影，这在其他工程中也没有列出数据，只列出了有底纹的帧数。本文通过乘以因数 1.5 来再现无底纹性能来进行标准化。而这种做法由于帧数相等、底纹需要并行运行而并不有利于本文平台。

表 3 每平方毫米祯速度的性能比较

平台	帧大小	底纹	办公室（原）	会议（原）	兔子（原）	办公室（标）	会议（标）	兔子（标）
本文	512*384	否	22.69	12.76	24.65	1.971	1.101	2.1388
ASIC	1024*768	是	9.92	8.23	N/A	1.109	0.909	N/A
CELL	1024*1024	否	N/A	57.2	N/A	N/A	0.165	N/A
GPU1	1024*1024	是	N/A	16.7	12.7	N/A	0.033	0.025
GPU2	1024*1024	否	N/A	15.2	N/A	N/A	0.0275	N/A
OpenRT	1024*768	是	2.6	2.0	N/A	0.009	0.007	N/A

标准化之后的数据如表 2、表 3 所示。表 2 中列举了不同平台的面积标准化到 130nm 之后的硅工艺。表 3 中“原”的意思是原始帧速率，而“标”代表标准化之后的帧速率。从表 3 中可看出，OpenRT 主要因为其普通的目的处理器的实现而劣于其他平台。两个 GPU 平台的实现有着相近的结果，但又同时由于有限的程序设计模型而不如所期望的那样好。而 CELL 平台由于其多处理器结构而具有相对较好的性能[7]。不过，两个 ASIC 设计(包括本文平台)则更优于其他平台。这也正说明了本文对于光线追踪的特殊设计能够起到对场景的渲染更为有效的作用。

而本文平台又相对更优一些。首先，本文平台采用了高电平硬件描述语言，并且不需要优化门电平。其次，本文应用了更多的硬件定向算法，更大程度的减少了芯片静态存储空间。最后，本文采用诸如折叠加工、资源共享、以及多线程等方法最大限度的利用了硬件资源。

综上所述，本文硬件平台应用这些硬件资源运行具有更高的效率。

(下转第 129 页)

```
procedure Tw_main.load_right(yhbm:string);
var
    q:Tadoquery;   kjmc,kjlm:string;
begin
    q.SQL.clear;
    q.sql.add('select distinct a.kjmc,a.kjlm from xt_kjlbk
a join xt_kjqxdyk b on b.kjbm in (a.kjbm) join xt_yhqxk
c on b.qxbm in (c.qxdm) where c.yhbm= '+yhbm+ '   and
a.zt=1 ');
    //根据登录用户返回用户的控制控件
    q.Open;
while not q.Eof do
   begin
     kjmc:=trim(q.fieldbyname('kjmc').AsString);//控件
名称
     kjlm:=trim(q.fieldbyname('kjlm').AsString);// 控 件
类名
     kjztsz(kjmc,kjlm,2);//调用自定义函数将控件设置
成不可见或不可控制
     q.Next;
   end;
   q.Free;
end;
```

5 结论

本文采用角色访问控制模型实现了对界面可控制原子操作的细粒度权限控制。文中的方法对中小规模的程序比较适用，较大规模程序可以使用粗细粒度相结合的方式，对部分界面有选择地使用细粒度，避免由于粒度太细导致的设置复杂，又可以加深权限树的深度，从而实现多粒度的权限控制。本文所述的权限管理方式在本单位的高校招生体检系统中实现，编程环境为 delphi7、MS-SQL2000。

参考文献

1 Andhur C, Feinstein H. Role-Based access control models. IEEE Computer, 1996,29(2):38－47.

2 刘宏波.一种采用 RBAC 模型的权限体系设计.计算机技术与发展,2009,9:154－163.

3 来竞,郎昕培.基于角色和目录服务的细粒度访问控制方法.计算机与数字工程,2006,34(9):8－11.

4 覃章荣,王强,欧镔进.基于角色的权限管理方法的改进与应用.计算机工程与设计,2007,28(6):1282－1284.

5 Cantu M. 王辉,谭海平,等译.Delphi3 从入门到精通.1998.188.

6 萨师煊,王珊.数据库系统概论.北京:高等教育出版社,2000.128.

7 蔡昭权.基于业务无关的权限管理的设计与实现.计算机工程,2008,34(9):183－185.

(上接第69页)

5 结论

本文采用了一种非常适合硬件实现的求交算法，其比 KD 树更益于硬件实现，因为其去除了不必要的列表和存储。在硬件架构设计过程中，本文提出了光线与三角形求交的硬件架构方法，这种方法在一定程度上解决了费时的硬件计算问题，采用诸如折叠、资源共享、以及多线程等硬件架构设计方法，有效的提高了硬件资源的使用效率。

光线追踪模型的研究仍然是一个任重而道远的事情。今后的工作将会考虑在单个芯片上集成阴影计算，这需要进一步研究如何减少总线数据量。另外，为了使本文算法对动态场景具有更好的支持性，将来会致力于实现更为有效的硬件设计单元。

参考文献

1 Jayasena N, Ho R, Dally W, Mai K, Paaske T, Horowitz M. Smart memories: A modular recongurable architecture. IEEE International Symposium on Computer Architecture. 2000.

2 Horn DR, Sugerman J, Houston M, Hanrahan P. Interactive k-d tree gpu raytracing. Symposium on Interactive 3D Graphics. ACM Press, 2007.

3 Popov S, Gunther J, Seidel HP, Slusallek P. Stackless kd-tree traversal for high performance gpu ray tracing. Computer Graphics Forum, 2007,26(3):415－424.

4 Knittel J, Lauer H, Pfister H, Hardenbergh J, Seiler L. The VolumePro real-time ray-casting system. Computer Graphics, 1999,26:251－260 .

5 Trumbore B, Moller T. Fast, minimum storage ray triangle intersection. Journal of Graphics Tools, 1997,2(1):21－28.

6 Brunvand E, Woop S, Slusallek P. Estimating performance of array-tracing asic design. Proc. of IEEE Symposium on Interactive Ray Tracing 2006, September 2006. 7－14.

7 Benthin C, Wald I, Michael, Friedrich H. Raytracing on the cell processor. Proc. of IEEE Symposium on Interactive Ray Tracing 2006, September, 2006. 7－14.