基于 Cache 和层次 Z 缓存算法的 3 维图形深度消隐硬件设计和实现

钟 伟 郭 立 杨 毅
(中国科学技术大学电子科学与技术系,合肥 230027)

摘 要 为了在 3 维图形渲染硬件系统中节省带宽和提高消隐效率,基于 Cache 和层次 Z 缓存算法(hierarchical Z-buffer,HZB),设计了一个深度消隐硬件模块。该硬件模块主要面向带宽有限的片上 3 维图形渲染系统,其在节省带宽的同时,还可加快消隐速度和提高消隐效率。该模块通过设计优化 Z Cache 结构来获得较高命中率,并采用了 1 级层次 Z 缓存算法,以提高消隐效果,同时加入了快速 Z 清除(Fast Z Clear)结构,以节省带宽。该设计已通过 RTL 级建模和仿真验证。实验结果表明,该新的硬件可节省大概 30% 的带宽,消隐速度和效率最多可提高 20% 。

关键词 消隐 层次 Z 缓存 Z Cache
中图法分类号: TP391 **文献标识码**: A **文章编号**: 1006-8961(2009)07-1392-07

Design and Implementation of Hidden Surface Removal Hardware Based on Cache and Hierarchical Z-Buffer Algorithm for 3D Graphics

ZHONG Wei, GUO Li, YANG Yi
(*Department of Electronic Science and Technology, University of Science and Technology of China, Hefei* 230027)

Abstract This paper presents a design of the hidden surface removal hardware module based on the Cache and Hierarchical Z-Buffer algorithm. The hardware module can save bandwidth while increasing speed and improving efficiency of hidden surface removing, which is suitable for bandwidth-limited on-chip 3D graphics rendering system. The design optimizes the Z Cache structure to acquire high hit rate, and uses one-level Hierarchical Z-Buffer algorithm to enhance the effect, meanwhile affiliates the Fast Z Clear structure to save bandwidth. The design has been described the RTL models and has passed the simulation. Experimental results show savings of about 30% of the bandwidth, speed and efficiency of removal up to 20% at best.

Keywords hidden surface removal, Hierarchical Z-Buffer, Z Cache

1 引 言

随着消费电子类市场的发展,3 维图形显示技术在移动终端和嵌入设备中的应用受到了高度关注,因此,片上 3 维图形渲染系统将获得广泛应用。

由于 3 维图形渲染流程涉及到纹理映射、模板测试、深度测试和颜色混合等操作,因此需要大量的存储器读写[1]。以渲染上万个三角面的场景为例,光栅化后将有数百万 pixels。考虑到至少 30 帧图像的渲染速度要求,以及每个像素 10 多个 Byte 数据的存取操作,因此需要超过片上系统 400 M 的总线带宽,更不用说片上系统为了节省功耗而对总线带宽提出的制约。这就成为限制片上 3 维图形渲染系统性能的一个瓶颈。本文主要考虑解决深度测试带来的带宽问题。深度测试是用来解决隐藏面消除的

收稿日期:2007-08-30;**改回日期**:2008-03-14

第一作者简介:钟 伟(1966 ~),男。2008 年于中国科学技术大学获电路与系统博士学位。主要研究方向为计算机图形学、图像处理。E-mail:hotboy@mail.ustc.cn

技术,可用于把 3 维世界的不可见像素剔除掉。

目前绝大部分图形硬件都采用深度缓存算法来解决可见性问题。该算法简单且硬件易于实现,它可让靠近视点的像素点的深度和颜色取代旧值,只需要一个 Z 缓冲器(Z-Buffer,用于存储像素深度)以及一个帧缓冲器(Frame Buffer,用于存储像素颜色)。但该算法效率不高,不仅消隐速度不够快,且带宽资源消耗大。为了减少带宽和加快消隐速度,Greene 等人提出了 Z 值金字塔的思想[2]。由于该算法需维护一个完整的 Z 值金字塔,因此尚无法实时使用,且目前没有硬件实现的相关报道。ATI 的 Hyper-Z 技术吸收并简化了这一算法,并且已在其成熟的商业 PC 显卡上应用,不过具体的实现原理尚未完全公布[3]。Cheng 等人提出了一个面向 PC 的二级层次 Z 缓存的算法,其中第 2 级是通过标记 4 个相邻层次 Z 的最大值实现[4]的。该算法虽然比较简单有效,但减少带宽消耗方面仍可改进。当然还有其他的深度缓存消隐的改进算法[5-6],但由于其电路结构复杂,实现也较为困难。

针对目前片上 3 维图形渲染系统存在的带宽资源有限问题,本文设计了一个一级层次 Z 缓存(Hierarchical Z-Buffer)消隐硬件模块。其主要思想是先把 Z-Buffer 的分辨率降低一次,然后取对应区域的最大的 Z 值来构造层次 Z 缓存。该硬件模块给光栅处理模块提供了一个读层次 Z 缓存的接口。若待处理三角面或像素的最小深度大于对应缓存的值,则表明不可见,此时光栅单元可直接抛弃。同时吸收了 Hyper-Z 技术的 Fast Z Clear(快速 Z 清除,以下统称为 Fast Z Clear)思想,把对 Z-Buffer 的部分操作转为对 Tile 标志位的清零和判断,以加快消隐速度和提高效率,以及减少不必要的像素数据总线读写请求。

2　深度消隐的实现方法

由于本文实现了层次 Z 缓存、Z Cache 和 Fast Z Clear,从而减少了带宽资源和加快消隐速度。因为其中层次 Z 缓存可提前剔除不可见像素,所以可加快消隐的速度,还可减少对不可见像素进行深度测试所带来的带宽消耗。而 Z Cache 和 Fast Z Clear 则利用数据关联性和标志位操作来直接减少带宽的消耗。

2.1　层次 Z 缓存的大小和维护

层次 Z 缓存的大小视硬件消耗、存储容量和算法效率的要求确定。若对应的 Tile 过大,则算法的效率变低,达不到剔除不可见像素的目的;若对应的 Tile 偏小,则存储容量会成为问题,效率也不一定高。综合考虑后,本文选定与层次 Z 缓存对应的 Tile 大小为 8×8。

层次 Z 缓存的数据要及时更新,这可以保证剔除掉尽可能多的不可见三角面或像素,以便确保算法效率,但层次 Z 缓存的维护比较困难。目前维护有以下两种常用方法:一是定期对 Z-Buffer 进行扫描,找出每个 Tile 中的最大 Z 值,用来更新层次 Z 缓存;二是采用位掩码来标记像素的更新情况,当某个 Tile 内的标记满足条件时,即触发对层次 Z 缓存的维护。由于定期更新策略不能保证层次 Z 缓存的及时更新,因此会对算法效率造成影响。位掩码标记能以较小的存储代价,保证算法的效率,已成为层次 Z 缓存维护的首选。

图 1 给出了位掩码的维护原理,当正方形内的所有像素的标记都置 1 时,则表明像素深度均改变,此时应该用正方形内像素的最大深度来及时更新层次 Z 缓存。

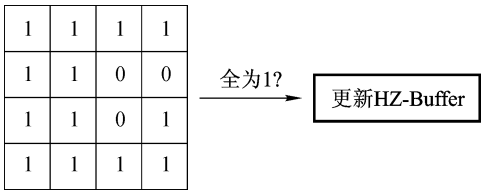


图 1　层次 Z 缓存位掩码维护示意
(以 4×4 的 Tile 为例)

Fig. 1　Bit mask cover of Hierarchical Z-Buffer

2.2　Z Cache 结构和容量

由于整个深度缓存占用的存储容量较大,不可能直接放置在消隐模块内部,因此对深度的访问是通过总线来实现的。另外,由于目前片上系统的总线资源有限,可是图形渲染却有大量的总线访问请求,为此本文采用 Z Cache 来解决该问题。

目前 Cache 已出现了各种结构,如直相连、组相连、全相连等,容量也有大有小,但都是为了尽量提高命中率。当然结构越复杂的,其容量越大,命中率相对越高,但其代价也高。

本文 Cache 的设计方案,在命中率足够高的条件下才易于硬件实现和节省资源[7]。

这里选用图 2 的 4 个场景,在存储容量从 1 K 到 4 K(1 K = 1 024 Byte,以下皆是)、结构为 2 路组相连和全相连的不同 Cache 下进行命中率比较,conf 1 是存储容量为 1 K,采用 2 路组相连结构的情况; conf 2 是存储容量为 1 K,采用 8 路组相连结构的情况;conf 3 是存储容量为 2 K,采用 2 路组相连结构的情况; conf 4 是存储容量为 4 K,采用 2 路组相连结构的情况 a,b,c,d 对应图 2 中的 4 个场景命中率比较结果见图 3。可根据图 2 和图 3 的实验数据来选定 Cache 的结构和容量。

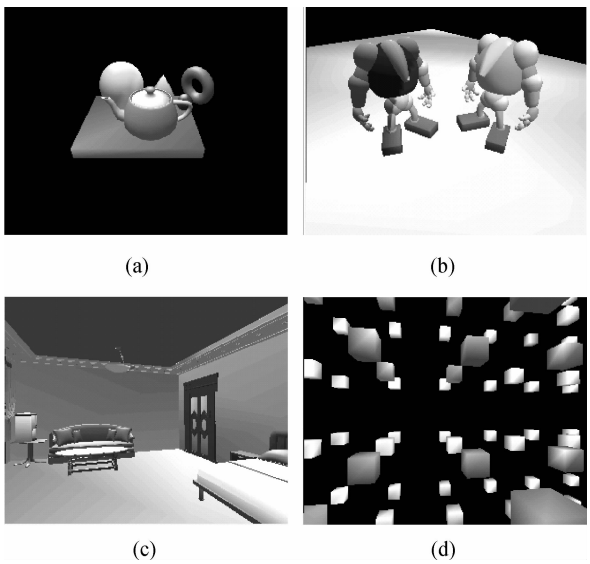


图 2 测试场景(屏幕大小为 320 × 240,光栅 Tile 大小为 8 × 8)

Fig. 2 Test scenes

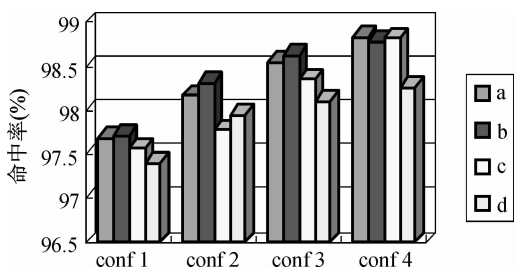


图 3 Cache 命中率比较(Tile 大小为 8 × 8,替换策略为 LRU, Cache Line 线容量为 128 Byte,Z 精度为 16 bits)

Fig. 3 Comparison of the hit rate of Cache under 4 conditions

由图 3 可见,存储容量为 1 K 到 4 K 时,Cache 的命中率都达到 97% 以上。相同容量下,全相连结构的命中率比 2 路组相连结构约高 0.5 个百分点;但当 Cache 的存储容量增大到 4 K 时,2 路组相连结构的命中率已经比较理想。从设计难度和存储容量考虑,本文主要考虑 Cache 存储容量为 2 K 的 2 路组相连方案,每个 Cache 行包含 8 × 8 Tile 内的像素深度数据(深度精度为 16 bits),以充分利用数据的时空相关特性。

由于加入了对层次 Z 缓存的维护,Z Cache 的结构还要加以改进方能实现。而具体的改进则是对每个 Cache 行增加一个用于标记像素更新的位掩码信息行(如图 4 所示),以便在更新深度数据的同时,也更新对应的比特位,并随时监测是否满足对层次 Z 缓存进行更新的触发条件。

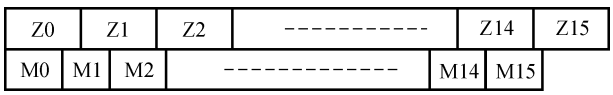


图 4 Cache 行的改进(以 16 个数据为例)

Fig. 4 Improvement of Cache row

图 4 中,当 M0-M15 都置 1 时,即可以将 Z0-Z15 中的最大值用来更新层次 Z 缓存了。

2.3 Fast Z Clear

Fast Z Clear 是 Hyper-Z 技术提出的可以减少带宽使用的思想[3,8]。每帧开始时,Z-Buffer 的数值要重写为背景深度,这需要对每个像素执行写操作一次;同时 Cache 第 1 次请求的某个 Tile 数据是已知的背景深度。考虑到这些特性,硬件模块里面可以设置一组标志位,对应于每个 Tile。写操作可以转换成对标志位的清零操作。在 Cache 请求 Z-Buffer 数据前也可以检查对应 Tile 的标志位,若为 0,则可不用向 Z-Buffer 请求,而改为直接把背景深度写入 Cache 行。本文采用了这些措施,从而减少了带宽的消耗。

3 硬件实现

3.1 硬件框图与工作流程

硬件按照功能分为 Cache 部分、深度测试部分和层次 Z 缓存部分。其中 Cache 部分又可以分为地址产生模块、Z Cache 模块和位掩模 Cache 模块,而层次 Z 缓存部分还融合了 Fast Z Clear 模块。硬件框图如图 5 所示。

整个硬件模块的工作流程描述如下:

首先是地址产生。地址产生时,地址产生模块根据像素的屏幕坐标 X 和 Y,按照一定的规则计算其 Z-Buffer 地址,以便读写相应的深度值。

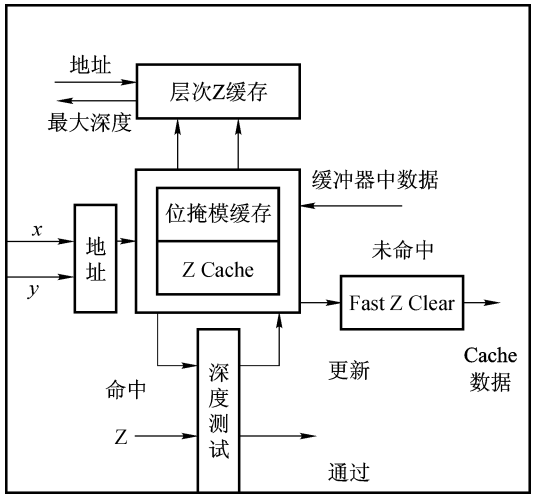


图 5　深度消隐硬件框图

Fig. 5　Structure of hidden mask removal hardware

接着是 Cache 请求,即把相应的 Z-Buffer 地址传给 Z Cache。若数据在 Cache 中,则返回一个命中信号,随后把数据读到测试模块进行测试。否则返回不命中的信号,让测试模块等待 Cache 的相应处理。

一旦得到深度数据,便可以进行相应测试。考虑到读写使能信号,要根据最终测试结果决定是否回写以及是否抛弃像素数据。一旦需要回写,则要对 Cache 中的相应数据进行更新。

更新数据时,要根据地址找到相应的行和字,将深度修改成新值。此时应同时修改所对应的位掩模缓存中的数据,并标记该像素的深度值已被修改过。

位掩模缓存数据的修改会导致层次 Z 缓存的更新。若一个 Cache 行的深度数据都更新过,则表明可以更新层次 Z 缓存了。

若 Cache 请求不命中,则会导致 Cache 向 Z-Buffer提出数据请求。

3.2　硬件描述

3.2.1　地址产生模块

这个模块是将像素的屏幕坐标转换为其深度在 Z-Buffer 中的对应地址,供进行 Cache 命中判断和向 Z Buffer 请求数据时使用。这里充分考虑了基于 Tile 光栅算法的特性,将同一个 Tile 内像素的深度组织到一起,以提高数据的关联度和访问命中率。整个 Z-Buffer 数据的访问顺序如图 6 所示。

图 6 是本文对 Z Buffer 进行访问顺序的设计,首先是 Tile 间的行优先,接着是 Tile 内的列优先。

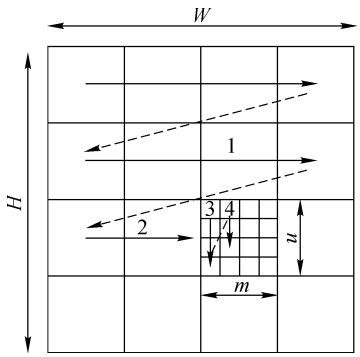


图 6　Z-Buffer 的访问顺序

Fig. 6　Accessing sequence of Z-Buffer

本文采用一个新颖的地址计算方法[9]。假定像素坐标为(x,y),则对应的 Z-Buffer 地址 $\mathrm{Addr}_{\mathrm{re}}(x,y)$ 为

$$\mathrm{Addr}_{\mathrm{re}}(x,y)=\underbrace{(y-y\bmod n)W}_{1}+\underbrace{(x-x\bmod m)n}_{2}+\underbrace{(x\bmod m)n}_{3}+\underbrace{y\bmod n}_{4}$$

$$=W(y-y\bmod n)+xn+y\bmod n \quad (1)$$

如果屏幕和 Tile 的大小都假定为 2 的指数,则式(1)可简化成

$$\mathrm{Addr}_{\mathrm{re}}(x,y)=W\cdot y[k:i]+\{x,y[i-1:0]\} \quad (2)$$

其中 $2^i=n$

本文在设计时选择 Tile 为 8×8 大小,Z-Buffer 大小为 320×240。

3.2.2　Z Cache 模块

在算法部分已经介绍了 Cache 结构和容量对命中率的影响,并在此基础上确定了 Cache 参数。在整个模块中,Z Cache 的设计属于比较复杂的,其不但要跟测试单元打交道,还要控制位掩模缓存和更新层次 Z 缓存(如图 7 所示),不过结构相对固定,这里主要介绍设计时的时序要求。

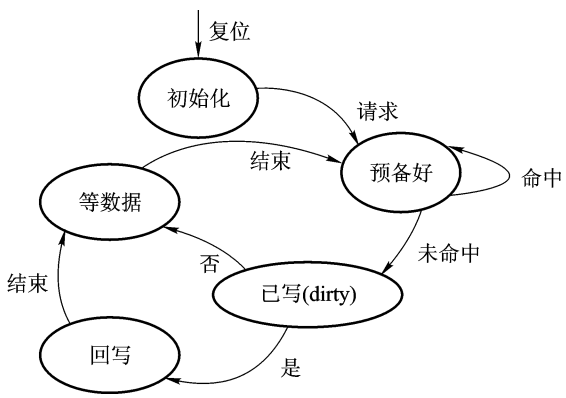


图 7　Z Cache 的时序控制状态转换

Fig. 7　Timing control states of Z Cache

由于对每个像素均有 Cache 请求,深度测试,更新 Z 值,更新层次 Z 缓存等些步骤,假如不采用流水线技术的话难以保证像素处理的速度和流畅。设计重点是保证采用流水线技术后,即使出现同一像素相邻时访问 Cache 仍能正确。图 8 给出了像素 A、B、C 相继访问 Z Cache 的时序。

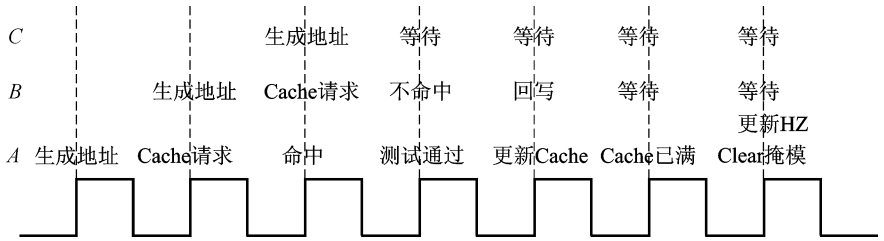


图 8 流水线下像素相继访问 Cache 的时序

Fig. 8 Time sequence of pixels accessing to Cache under pipeline structure

3.2.3 位掩模模块

位掩模在设计时,其实际上是融合在 Z Cache 中去的,因此可采用跟 Z Cache 相同的结构。但当被替换时其数据可采用不同的方法解决。若开辟一块位掩模缓存,其中的数据便同 Z 值一样,回写到 Z-Buffer 中。若不想用额外的空间来存储位掩模信息时,则可以在 Cache 行被替换时,扫描行内像素的深度,并更新层次 Z 缓存。本文采用开辟新的缓冲器(Buffer)的方法来解决位掩模信息的存储。

为了节省层次 Z 缓存更新被触发时扫描整个 Tile 内深度的时间,可在对每行进行掩模的同时,增加一个 Temp Z 变量,用来随时保存进入该 Tile 的最大深度,即当新像素的深度大于 Temp Z 时,则把 Temp Z 替换成该深度值。但 Temp Z 同样面临和位掩模一样的问题,即 Cache 行被替换时,其如何存储的问题。这里将掩模和 Temp Z 同等对待,一并在硬件模块外开辟新缓冲器解决。

3.2.4 层次 Z 缓存模块

这个模块保存着 Z-Buffer 中各个不相重叠的 8×8 大小的 Tile 的最大深度。为了使光栅能有效剔除不可见的三角面和 Tile,本文将其设计在硬件模块内。由于其存储容量是 Z-Buffer 的 1/64,因此其耗费的存储容量不大。进一步考虑到进行光栅阶段的比较不需要太高的精度,可以让层次 Z 缓存的数据只保留 8 bits 精度即可。以 320×240 分辨率为例,由于层次 Z 缓存只有 1 200 Byte 大小,因此完全可以放进模块里。

层次 Z 缓存中还融入了 Fast Z Clear 模块。Cache 每次请求 Z-Buffer 前要先判断该 Tile 的标志位。该模块只需很小的存储容量。

3.2.5 深度测试模块

该模块在图形渲染流水线中的功能很明确,就是根据预先设定的测试选项对像素进行测试。当选择 GL_LESS 或 GL_LEQUAL 时,则是常规意义上的消隐,而其他选项则是用来产生特殊效果的。

4 综合、仿真和分析

本文选用 Xilinx 公司的 Vertex 2 pro 系列 XC2VP30 板子为目标来进行综合和仿真。

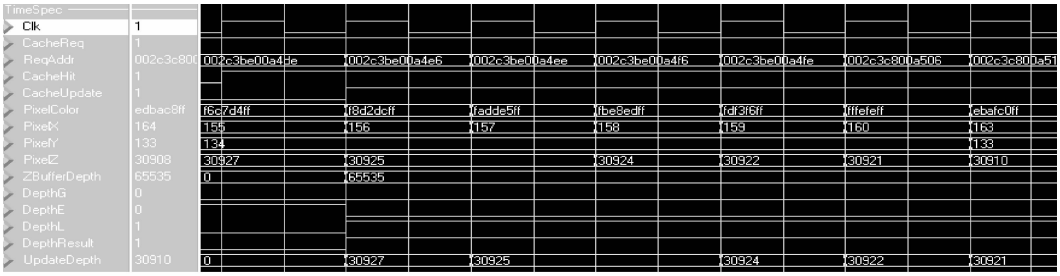
4.1 资源消耗和时序仿真

表 1 给出了主要模块的资源消耗情况,由表 1 可知,资源消耗较多的是触发器和查找表,因为为了保证模块的工作频率,以确保像素处理的能力和速度,需要大量加入流水线技术,从而使时序设计方面变得复杂,且时序逻辑和组合逻辑也相应增多。两个模块在现场可编程门阵列(FPGA)上的频率能综合超过 100 MHz,便是得益于流水线技术的加入。不难知,在专用集成电路(ASIC)工艺下,将可进一步提高最高的工作频率。图 9 给出了核心模块的 RTL(register transfer level)时序仿真结果。图 9(a)是几个像素通过连续请求 Cache 来完成深度测试的

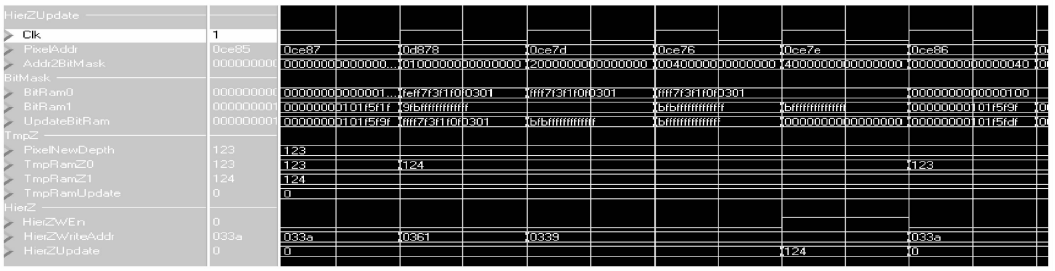
表 1 模块综合资源消耗

Tab. 1 The consuming of resourses

资源分类	资源消耗	
	Z Cache	层次 Z 缓存
Number of Slices	988	600
触发器(Slice Flip Flops)的数目	1 151	363
4 输入查找表(LUT)的数目	1 343	1 101
最大频率	220 MHz	133 MHz



(a) Z Cache仿真结果



(b) 层次Z缓存的仿真结果

图 9　Z Cache 和层次 Z 缓存的行为级时序仿真结果

Fig. 9　Timing simulation on RTL level of Z Cache and Hierarchical Z-Buffer

时序,而图 9(b)则是位掩模满足更新层次 Z 缓存条件下的操作。这两个时序组合起来便是一个像素通过深度测试来改变掩模信息,并导致层次 Z 缓存更新的完整的流水线操作。

4.2　算法性能和带宽节省

仍以前述测试场景为例来说明得到一级层次 Z 缓存的剔除效率(如表 2 所示):

表 2　算法性能(分辨率为 320×240,Tile 大小为 8×8)

Tab. 2　Performance of the arithmetic单位:%

	层次 Z 缓存算法的剔除效率	Fast Clear 的剔除效率
场景 a	22	14
场景 b	14	12
场景 c	12	14
场景 d	10	22

表 2 表明,本文采用的层次 Z 缓存算法的剔除效率在 10%~22% 之间,即是说,在光栅化过程中大约有 20% 的 Tile 被直接抛弃掉,而不需要进入渲染流程,从而减少了像素数据量和带宽消耗。表 2 还给出了采用 Fast Z Clear 技术后不需要从 Z-Buffer 中读数据的 Tile 所占的比例,这意味着直接减少了 12%~22% 的带宽资源。而 1 级层次 Z 缓存算法配合 Fast Z Clear 技术,最终能节省 26% ~36% 的总线带宽。当然,测试结果会随场景不同而改变,但最终带宽节省是很明显的。这是由于在整个模块内集成了层次 Z 缓存和 Fast Z Clear 模块,而在模块外则开辟了掩模缓冲器和 Temp Z 缓冲器所致。总线带宽的节省与硬件复杂程度的增加和资源消耗的增加两者是不可兼得的,需要在两者之中寻求平衡点。

4.3　Cache 性能分析

由测试图 2 的场景分析可见,本文设计的存储容量为 2 K 的 2 路组相连结构的 Cache 的命中率平均能达到 98%(见图 3 中 conf 3 对应的情况)。这一方面是由于光栅化是以 Tile 为单位进行的,其像素数据存在较大的时空关联性,而 Z-Buffer 中的深度值也相应采用 Tile 结构组织,数据访问的规律性得到充分的利用所致;另一方面则是由于 Cache 的结构和容量保证了 Cache 的基本性能的缘故。这两方面的综合考虑,就使 Z Cache 的命中率达到了比较理想的程度。

5　结　论

本文采用层次 Z 缓存算法和利用自顶向下的设计方法实现了一个 1 级层次 Z 缓存消隐硬件模块,并通过了仿真验证。由于模块中 Z Cache 可达到 98% 的命中率,而层次 Z 缓存还能取得 10% ~

20% 的剔除效果,因此可快速有效减少不可见的像素信息。在综合考虑 Fast Z Clear 技术后,深度测试耗费的带宽资源可节省达 30% 。

参考文献(References)

1 Akenine Möller T, Haines E. Real-Time Rendering (second edition)[M]. Wellesleg, MA, USA: A. K. Peters Ltd., 2002.

2 Greene N, Kass M. Hierarchical Z-buffer visibility[A]. In: Proceedings Annual Conference Series Computer Graphics [C], Anaheim, CA, USA, 1993: 231-238.

3 Morein S. ATI radeon hyper-Z technology[A]. In: Hot 3D Graphics Hardware Workshop [C], Interlaken, Switzerland, 2000.

4 Chen Cheng-hsien, Lee Chen-yi. Two-level Hierarchical Z-Buffer with compression technique for 3D graphics hardware [J]. The Visual Computer, 2003, **19**(7-8): 467-479.

5 Aila T, Miettinen V, Nordlund P. Delay streams for graphics hardware [J]. ACM Transactions on Graphics, 2003, **22**(3): 792-800.

6 Zhang H, Manocha D, Hudson T, *et al*. Visibility culling using hierarchical occlusion maps[A]. In: Proceedings SIGGRAPH'97 [C], Los Angeles, CA, USA, 1997: 77-88.

7 Hakura Ziyad S, Gupta Anoop. The design and analysis of a cache architecture for texture mapping[A]. In: Proceedings of the 24th International Symposium on Computer Architecture [C], Los Angeles, CA, USA, 1997: 108-120.

8 Chang H Y, Lee S K. A hierarchical depth buffer for minimizing memory bandwidth in 3D rendering engine: DEPTH FILTER[A]. In: Proceedings of IEEE International Symposium on Circuits and Systems[C], Denver, CL, USA, 2003: 108-120.

9 Sambuddhi Hettiaratchi, Peter Y K. Cheung. A novel implementation of tile-based address mapping[A]. In: Proceedings of the Design, Automation and Test in Europe Conference and Exhibition [C], Bangkok, Thailand, 2004: 724-727.