低端嵌入式设备 3D 图形库的设计

胡 锐，马维华，解书钢
（南京航空航天大学，江苏 南京 210016）

摘 要：针对低端嵌入式设备运算速度较低，存储空间较小且不支持通用 3D 图形库的局限性，通过分析常规 3D 渲染管线和算法，对次要功能实施裁减，设计出一个简化的 3D 图形库。经在低性能手机上测试，此方案渲染效果良好，切实可行。

关键词：3D 图形库；渲染管线；嵌入式设备
中图分类号：TP334.2+1 **文献标识码**：A

Design of 3D graphics library for embedded device of lower performance

HU Rui, MA Wei Hua, XIE Shu Gang
(Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China)

Abstract: Earlier embedded devices of lower performance and smaller memory don't support general 3D graphics library. This paper analyzes traditional 3D rendering pipeline and algorithms, removes unimportant functions and designs a reduced 3D graphics library. Then tests on mobiles of lower performance prove the good rendering result and feasibility of this mechanism.

Key words: 3D graphics library; rendering pipeline; embedded device

随着计算机硬件技术的迅速发展与图形渲染技术的不断革新，工业控制与消费电子领域的嵌入式设备都已经引入了 3D 显示的功能。嵌入式设备在具有体积小、功耗低、适应性强的优势的同时，也存在处理器运算速度低，存储空间小等诸多局限性，而且部分低端设备并不支持通用的嵌入式 3D 图形库（如通用的 OpenGL ES 与手机平台的 Mobile 3D Graphics）。为了在此类设备上实现 3D 渲染与显示，需要深入分析常规 3D 渲染管线的实现过程，在此基础上去掉一些次要的效果或功能，最后设计简化后的 3D 图形库。

与 2D 图形显示相比，3D 渲染更为形象更接近现实，因为后者加入了透视效果与光影效果。在有限的条件下，设备用户往往对透视效果(如物体的纵深与遮挡)的需求远远大于光影效果(如光照与雾化处理)。因此，简化后的 3D 图形库需要保留最基本的透视效果，而去除表现光影效果的相关功能。

1 3D 渲染管线的裁减

尽管 3D 渲染的具体流程随图形处理器与用户需求的不同存在相当的差异，但大体上都要经过坐标变换、顶点与片段着色、光照计算、Alpha 混合以及相关的剔除裁剪处理。OpenGL 是目前应用最为广泛的底层 3D 图形库，并已经作为一个工业级的标准，它采用的渲染管线如图 1 所示[1]。

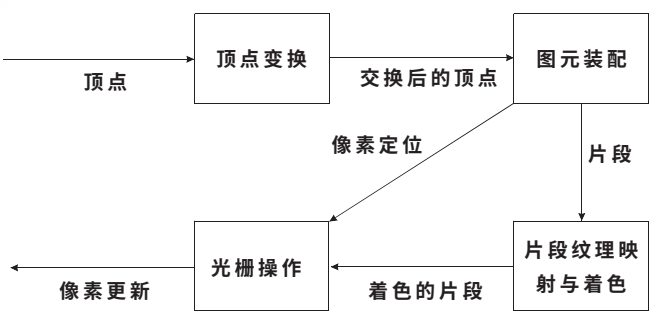


图 1 OpenGL 的渲染管线

1.1 顶点变换

顶点变换的功能有：(1)顶点位置的坐标变换；(2)顶点光照的计算；(3)顶点纹理坐标的生成与变换。

OpenGL 有 2 种坐标变换矩阵：透视变换矩阵与模型视图变换矩阵，分别由 2 个独立的堆栈维护[2]。由于

3D 坐标变换是用矩阵相乘完成的，其时间复杂度为 $O(n^3)$ 且参与运算的为浮点数，开销较大，所以将透视变换矩阵去掉，保留模型视图变换矩阵。在模型视图变换以后，对每个顶点 $P(x,y,z)$ 进行以下处理，以模拟透视变换的纵深效果(距离越远的物体显示越小)：

$$x=K_x\times y/z \quad y=K_y\times y/z$$

式中，K_x、K_y 是可调整的缩放参数，可以根据显示屏幕的长宽进行调整。

根据前面的分析，直接去掉光照计算；保留纹理坐标的生成，去掉纹理坐标的变换(因为后者主要用于纹理动画)。

1.2 图元装配

这个阶段需要计算片段在帧缓冲区的坐标以及对片段各点进行属性插值。片段各顶点的原始坐标一般是浮点数表示的，而在帧缓冲区内，像素坐标是离散的整数，这是由显示设备光栅的物理构造决定的。对此可以简单地通过将浮点坐标取整的方式获得帧缓冲区坐标，下面主要讨论插值算法。

插值算法有很多种，常用的有线性插值、Hermite 插值、最小曲率插值等，其中线性插值执行效率最高，在精度要求较低的场合使用较为广泛。

在较高性能的系统中(如 PC 机)，插值运算是由硬件(GPU)完成的，插值运算后还要经过各种平滑滤波处理，速度快且效果好。而对于嵌入式设备，可以采用 2 次线性插值的方法来达到较为理想的效果：先由构成片段的顶点属性(位置坐标、纹理坐标、颜色等)计算片段边界线上各点的属性值，同时用行扫描线算法得到片段覆盖的有效区域内每条扫描线的起止点坐标；再对每条扫描线起止点之间的各点进行属性插值，最终计算出片段内部所有点的属性值。

例如，由顶点 $P1(1,2,3)$、$P2(6,3,2)$、$P3(3,8,1)$ 构成的片段，先计算片段边界线上各点的坐标。以 $P1$、$P2$ 为例，由于 $|x1-x2|>|y1-y2|$，所以利用 x 坐标作参考量对 y 坐标与 z 坐标(即深度缓冲)进行插值，即：

$$y=y1+(x-x1)\times(y2-y1)/(x2-x1)$$

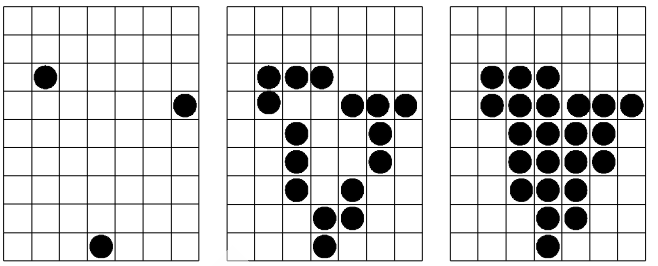
$$z=z1+(x-x1)\times(z2-z1)/(x2-x1)$$

其中 $y1<y<y2$。

对 y 坐标进行舍入取整处理后，插值结果为 $(2,2,2.8)$、$(3,2,2.6)$、$(4,3,2.4)$、$(5,3,2.2)$。同理可以得到 $P1$ 与 $P3$，$P2$ 与 $P3$ 各点的结果，同时得到了 7 条行扫描线起止点的 x 坐标对：$(2,3)$、$(1,4)$、$(2,5)$、$(2,5)$、$(2,4)$、$(3,4)$、$(3,3)$。

然后对每条扫描线 2 个边界点之间的各点即片段内部各点进行属性插值，以第 3 条扫描线为例，它的起止点坐标为 $(2,4,2.333)$ 与 $(5,4,1.8)$，计算得到其间 2 点坐标为 $(3,4,1.977)$、$(4,4,2.155)$。整个运算过程如图2 所示。

对片段内部各像素点的纹理坐标和颜色进行插值



构成片段的 3 个顶点　对片段边界点进行 1 次插值　对片段内部点进行 2 次插值

图 2　2 次线性插值算法对片段各点插值

的算法与此类似，其中颜色的插值又可以分为 R、G、B 3 个通道进行。

1.3 片段纹理映射与着色

如果纹理映射未开启或当前片段未绑定有效的纹理，那么上阶段插值得到的片段各点颜色值就是片段着色的结果；否则还需要利用上阶段插值得到的各点纹理坐标(通常称为 UV 坐标)计算其颜色值，其计算公式可以简单表示如下：

$$Color=tex[int(u\times texWidth)][int(v\times texHeight)]$$

式中，tex 是存储纹理各像素颜色值的二维数组，因为 $0<u$、$v<1$，所以将 u、v 分别与纹理宽度 $texWidth$ 与纹理高度 $texHeight$ 相乘再取整即可得到顶点映射的纹理位置，再通过查找就能得到顶点的颜色。

这样简单处理可能会带来一些问题，例如一个面积很大的片段在映射一片面积较小的纹理区域时，会造成纹理贴图因放大明显分辨率不足而出现格状，而面积很小的片段在映射一片面积较大的纹理区域时，又会造成片段内顶点纹理坐标不稳定而出现画面躁动。这些现象都会影响视觉效果。为此，OpenGL 与 Direct3D 等通用 3D 图形库会根据实际情况采取一些修正措施。

1.4 光栅操作

以上步骤已经完成了像素的定位、顶点与片段的着色。在将这些信息发送到显示屏幕之前，还要进行必要的测试，以剔除不需要渲染的像素，这些测试主要包含[3]：(1)坐标测试；(2)Alpha 测试；(3)模板测试；(4)深度测试。

坐标测试是将定位在显示屏幕之外的点去掉，这一步骤是必须保留的；Alpha 测试主要用于混色(blending)以达到一些特殊的半透明效果(如火焰)，所以这一步骤应该去掉；模板测试主要用于制作镂空、镜面、阴影等效果，也应该去掉；深度测试用于决定各个顶点或物体之间的遮挡关系(x、y 坐标相同的条件下，z 坐标即深度值较小的顶点覆盖 z 坐标较大的顶点)，予以保留[4]。

经过裁减后的 3D 渲染流程如图 3 所示。

1.5 硬件配置需求

以显示屏幕分辨率为 200×200、像素颜色深度与深度缓存深度均为 16 位(2 字节)的手持设备为例，根据裁减以后的渲染管线，帧缓冲区由像素颜色缓存与深度缓

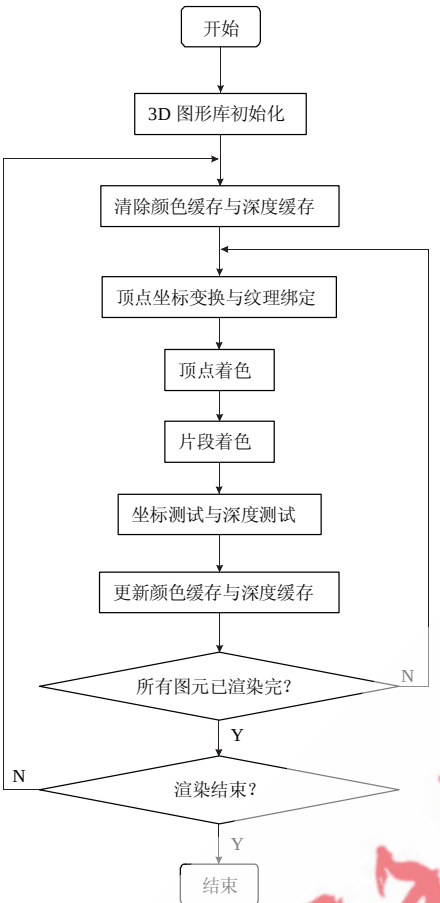


图 3 裁减以后的 3D 渲染流程图

存组成,需求为 200×200×2×2=160 KB,加上正弦、余弦等函数表以及各类常数的存储,设备的存储空间至少要在 200 KB 以上。如果要渲染较为复杂的模型,由于纹理贴图与骨骼动画关键帧信息还需要更多的空间,则存储空间须在 1 MB 以上,处理器频率至少要在 50 MHz 以上。

2 软件解决方案

2.1 3D 模型数据结构的简化

当前 3D 模型在计算机中大多是用网格(mesh)表示与存储的,而每个网格又由若干个三角形组成,同时每个三角形又由 3 个顶点(vertex)组成,顶点数据包含位置坐标、纹理坐标、顶点颜色等信息,这样就构成了网格–三角形–顶点的三级结构。由于 1 个顶点会同时被几个三角形包含,为了避免重复存储,三角形并不存储所含 3 个顶点的详细信息,而仅仅存储 3 个顶点的索引;而每个网格除了它所包含的三角形信息外还要存储纹理贴图,材质等多种信息,为了防止结构体过于庞大,一般不直接存放每个三角形的详细信息而只存放三角形的索引[5]。

使用这样的结构,每渲染 1 个顶点实际上只要经过 2 层索引查找操作,因为网格含有的材质信息与三角形含有的法线向量数据(均用于光照计算)并不需要,所以将上述三级结构简化为网格–顶点二级结构,如图 4 所示。这样既简化了数据结构节省了存储空间,又提高了编程效率。

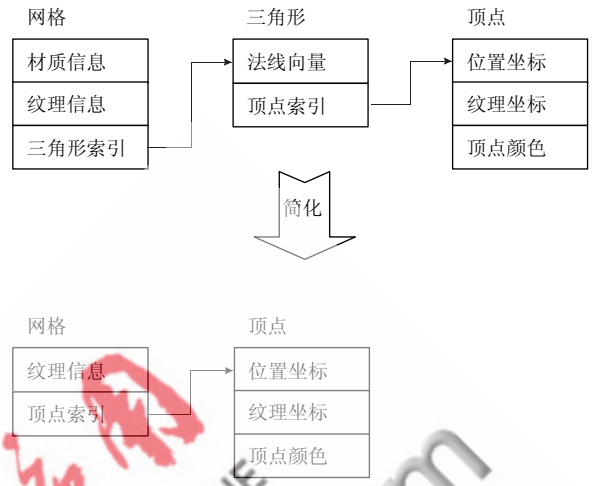


图 4 3D 模型数据结构的简化

2.2 浮点运算的模拟

一些低端嵌入式设备(如基于 ARM7 的微控制器系统与 CLDC1.0 版本的手机)不支持浮点数,而浮点运算贯穿于整个 3D 渲染过程,所以必须用定点数表示浮点数并模拟浮点运算。

以 long 型(4 个字节)为例,可以划分 8 位表示指数,其余 24 位表示尾数,每次运算以后都要进行规格化以便保存。这是 IEEE 的 float 型运算的软件实现,精度较高但需要耗费较多的时间。

对此可采用另一种精度较低但更为快速的方法——小数掩码法,即将定点数的最后若干位划为小数位,每次乘法运算后将结果右移相应位数,每次除法运算前将被除数左移相应位数,而加减运算规则不变。小数位划分越多运算结果越精确,反之可表示数的范围越大。本文取 8 位作小数位,运算精度为 $1/2^8$,可以满足一般的要求,同时可以保证整数部分在进行乘法时不会越界,这是因为嵌入式设备显示屏幕一般分辨率较小(长宽均小于 2^8=256),而相乘后整数部分有效位不会超过 16 位,小数部分有效位也不会超过 16 位,运算结果用 4 字节 32 位的 long 型即可存储。此外,小数掩码法附加的运算都是移位运算,硬件执行效率较高。

2.3 帧缓冲区的设计

帧缓冲区是 3D 渲染过程的终点,由于不需要模板缓存与像素的 Alpha 值,这段区域仅由像素颜色缓存与深度缓存组成。

像素的颜色格式有很多种,如 RGB、ARGB、YUV 等,PC 机平台常用的 32 位 ARGB 真彩色可以充分满足人眼的视觉需求。但考虑到嵌入式设备存储空间有限,

故采用 16 位 ARGB 1555 加强色格式，每个像素占用 16 位 2 个字节的空间，RGB 每个颜色通道各占用 5 位，另留 1 位作为 Alpha 标记，用于表示该像素是否透明，进而可以表现纹理贴图边缘的不规则形状。

深度缓存根据需要可以设置为 16 位、24 位、32 位等，在嵌入式设备上采用 16 位便可以满足需求。如果设备不支持浮点数，可以用小数掩码法用短整数来表示，划分低 8 位作小数位。

本文从分析常规 3D 渲染管线出发，结合低端嵌入式设备处理器运算速度较低，存储器空间较小且不支持现有通用 3D 图形库的局限性，提出了一种经过功能裁减后的渲染方案并计算所需的硬件最低配置需求，最终完成简化 3D 图形库的设计。这种图形库去除的是一些需要耗费大量运算时间与存储空间的光照、雾化、半透明、阴影、粒子等光影效果，而保存了 3D 图形基本特征与透视效果，所以仍然能够满足设备用户的一般视觉需求。此外，本文针对软件设计中的几个问题进行了讨论，并给出了解决的方案。

参考文献

[1] OpenGL Pipeline Overview [DB/OL]. http://www.lighthouse3d.com.

[2] Nehe，OpenGL Tutorials[DB/OL]. http://nehe.gamedev.net.

[3] OpenGL 编程指南(第四版)[M].邓郑祥，译.北京:人民邮电出版社，2005.

[4] 唐泽圣，周嘉玉，李新友.计算机图形学基础[M].北京：清华大学出版社，1995.

[5] MILLER J R. Vector geometry for computer graphics [J]. Computer Graphics，1999，19(3):66-73.

(收稿日期：2009-04-15)