图形处理器中光照和纹理映射的设计与仿真实现

董 梁[1] 刘 海[2] 韩俊刚[2]

(西安电子科技大学计算机学院 西安 710071)[1] (西安邮电学院计算机学院 西安 710121)[2]

摘 要 图形处理器(GPU)通常采用流水线体系结构,遵循通用图形接口规范。在分析图形处理器的工作原理和体系结构的基础上,提出了改进的实用型流水线设计结构,并对每个功能模块进行了分析。对光照和纹理映射部分进行了深入研究,提出了具体的设计方法。通过软件仿真的结果验证了设计的正确性。最后针对光照和纹理映射的特点,提出了可编程处理器及其硬件结构。

关键词 光照,纹理映射,体系结构,流水线,图形处理器

中图法分类号 TP391.41 **文献标识码** A

Design and Simulation of Lighting and Texture Mapping in GPU

DONG Liang[1] LIU Hai[2] HAN Jun-gang[2]

(Department of Computer Science, Xidian University, Xi'an 710071, China)[1] (Department of Computer Science, XUPT, Xi'an 710121, China)[2]

Abstract Graph Process Unit(GPU) usually adopts a pipelined architecture, and implements some general computer graphics API. This paper analysed the working principle and architecture of GPU, and proposed an improved practical pipeline structure. Every module of the pipeline was analyzed. This paper focused on the parts of lighting and texture mapping which demand intensive calculations, and presented a design method. The design method has been validated with software simulation and applied in the design of a hardware prototype. In addition, this paper also proposed a programmable architecture for lighting and texture calculations based on the results of simulation study.

Keywords Lighting, Texture mapping, Architecture, Pipeline, GPU

1 图形处理器介绍

随着计算机技术的迅速发展,图形处理器广泛应用于图形图像、视觉计算等方面。图形处理器中并非每条指令都必须运算出最终结果(可能只是作为中间状态),由指令和指令之间构成数据流。只有绘图的显示指令到来时,才会从显存中将数据取出显示在屏幕上。这样客观地决定了图形处理器采用功能流水线结构。目前图形接口标准有 OpenGL 规范和 DirectX 接口标准。本文设计的图形处理器以 OpenGL 规范为主,同时也支持 DirectX 接口标准。

2 图形处理器流水线

典型的图形处理流水线在 OpenGL 规范中已有详细说明,例如顶点的一系列坐标变换表示。该流水线依次是原始顶点坐标数据,即建模坐标;然后经过模型视图变换,得到观察坐标;再经过投影变换,得到剪裁坐标;再除以 w,得到规范化的设备坐标;再经过视口变换,得到最终的屏幕坐标。另外,还有 OpenGL 的渲染流水线,可以对几何数据和像素数据进行处理[8]。

在对传统图形处理流水线分析的基础上,我们提出一种实际的流水线结构。该图形处理流水线包括了命令处理、几何变换、法向量标准化、节点染色、钳位处理、图元装配、平面剪裁、投影变换、三维剪裁、齐次坐标处理、视窗变换、消隐处理、扫描转换、像素染色、雾处理、帧存储和其他处理等模块。

如图1所示,命令处理模块负责对命令解析,使其变为 GPU 内部可以识别的命令和参数数据流,再按一定的数据结构重新组织,向下一级传送;几何变换模块就是将物体的建模坐标通过平移、缩放、旋转等变化使其成为观察坐标系中的坐标;法向量标准化是对顶点的法向量进行处理,方法是计算模型视图矩阵的逆转置矩阵$(M^{-1})^T$,然后将顶点的法向量与新矩阵相乘并进行规范化处理。

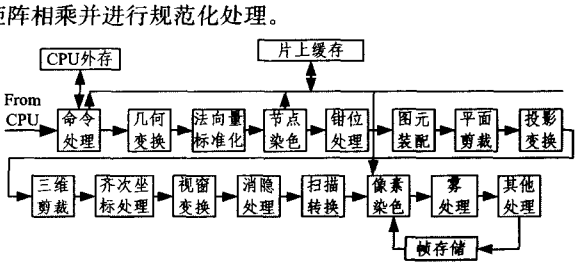


图1 图形处理器流水线

节点染色模块是根据 GL 命令对每个顶点重新组织数据结构。该数据结构中包括节点的齐次坐标、颜色值、光照、纹理坐标、法向量等信息,并将数据量较大的光照信息等存放到

到稿日期:2010-03-05 返修日期:2010-06-11

董 梁(1977-),男,博士生,讲师,主要研究方向为计算机图形学、计算机体系结构,E-mail:dodoliang@xupt.edu.cn;刘 海(1973-),男,博士,讲师,主要研究方向为计算机图形学、计算机软件与理论;韩俊刚(1943-),男,硕士,教授,CCF 高级会员,主要研究方向为 VLSI 设计与验证、计算机图形学。

片上缓存中；钳位处理模块主要是对顶点的一些参数值进行检查，以保证向下传送的数据都是合理值。

图元装配模块主要是将接收到的顶点数据进行组合，组成内部处理的图元，使得在 GPU 内部仅处理点、线和三角形；平面剪裁模块主要是用平面切割实体，从而决定实体的哪些部分处于平面之上，即可见的部分。

投影变换包括透视投影和正投影。投影变换后将视景体变换为一个边长为 $2w$，中心为圆点的正方形盒子。这个新的坐标系称为剪裁坐标系；三维剪裁模块将落在方盒子外的物体的部分截去，只剩下留在方盒子内的部分。三维剪裁的算法有很多，像 Cohen-Sutherland 区域检测算法、梁友栋-Barsky 参数化算法、Sutherland-Hodgman 多边形剪裁算法等。

齐次坐标处理是对 w 变量进行归一化的操作，即齐次坐标除以 w 值，得到规范化的坐标值；视窗变换模块是将规范化的坐标变化为显示窗口上的坐标，包括给出窗口的原点以及窗口的宽度和高度。也可以对 z 值进行变换，默认的 z 值为[0.0,1.0]，远离屏幕为正方向；消隐处理模块是考虑图元的法向量与视点方向的关系，是属于可见平面，还是不可见平面。

扫描转换模块也称为光栅化。该模块完成的功能是根据装配来的图元顶点，逐行扫描，得到每一行该图元分配的像素端点。扫描转换的算法很多，例如 Bresenham 画线算法等；像素染色模块是根据每行扫描的两个端点，计算每行中间点的坐标、光照、纹理坐标等信息，并计算出该像素点的颜色值。

帧存储模块是将像素数据保存起来，并将相同二维坐标的点进行比较，用透明度和深度值计算新的颜色值。当输出命令到来时，将该模块中一帧像素数据输出显示；雾处理模块是对场景信息的处理，雾气效果常用指数衰减函数来模拟。

3 光照设计

3.1 光照原理

在 OpenGL 光照模型中，物体表面的颜色是由照射到物体表面的光的特性和物体表面的材料特性共同决定的。OpenGL 的光照模型把光分成 4 种独立的成分：环境光、散射光、镜面反射光和发射光，这 4 种成分都可以单独进行计算，然后叠加在一起。

环境光就是那些在环境中进行了充分的散射，无法分辨其方向的光。散射光来自于某个方向，当它照射到物体表面上时，会均匀地向所有方向发散；镜面反射光也来自于某个方向，并且它会从物体表面向某个特定方向反射；发射光是物体本身发出的光。

在一个三维场景中可以有多个光源(OpenGL 最多支持 8 个光源)，每个光源发出的光都包含环境光、散射光和镜面反射光成分。此外还可能存在一个不来自以上光源的全局环境光。OpenGL 在模拟光照时，将以上每种光都分解成红、绿、蓝 3 种成分，因此光的特征就是由它的红、绿、蓝 3 种成分的量决定的。

物体表面材料也具有不同的环境、散射和镜面颜色，分别表示材料对环境光、散射光和镜面反射光中的红、绿、蓝光成分的反射率。因此，物体表面的颜色就由材料所反射的红、绿和蓝光的比例来决定。

综上所述，在启用光照的情况下，顶点的颜色是按照下面的公式计算的：

$$C = emission + ambient_{global} \times ambient_{material} + \sum_{i=0}^{n} \left(\frac{1}{k_c + k_l d + k_q d^2}\right)_i \times spotEffect_i \times [ambient_{light} \times ambient_{material} + \max\{L \cdot v, 0\} \times diffuse_{light} \times diffuse_{material} + \max\{s \cdot v, 0\}^{shininess} \times specular_{light} \times specular_{material}]_i$$

公式中各变量的含义如下：

$emission$ 是顶点处材料的发射光颜色；$ambient_{global}$ 是全局环境光颜色；$ambient_{material}$ 是顶点处材料的环境光颜色属性；n 是场景中光源的数量(需将每个光源产生的颜色分量叠加在一起)；k_c，k_l，k_q 分别是光源的常量衰减因子、线性衰减因子和二次衰减因子；d 是从顶点到光源的距离；$spotEffect$ 是光源的聚光灯效果；$ambient_{light}$，$diffuse_{light}$ 和 $specular_{light}$ 分别是光源的环境光颜色、散射光颜色和镜面反射光颜色；$ambient_{material}$，$diffuse_{material}$，$specular_{material}$ 分别是顶点处材料的环境光、散射光和镜面反射光的颜色属性；L 是从顶点指向光源位置的单位向量，v 是顶点处的单位法向量，$L \cdot v$ 表示这两个向量的点积；s 是一个单位向量，其计算方法是先将从顶点指向光源位置的向量与从顶点指向观察点的向量相加，再把相加结果单位化得到 s；$shininess$ 是镜面指数。

光源的聚光灯效果 $spotEffect$ 的取值分为以下 3 种情况：

(1)如果光源不是聚光灯，则取值为 1；

(2)如果光源是聚光灯，但顶点位于聚光灯光锥之外，则取值为 0；

(3)其它情况取值为 $\max\{L \cdot d, 0\}^{spotExponent}$，其中 L 是从顶点指向聚光灯的单位向量，d 是聚光灯的单位方向向量，$spotExponent$ 是聚光指数。

3.2 光照仿真实现

光照设计在整个图形流水线的第二级，位于几何变换之后。经过光照后，还需要图元装配、裁减、背面剔除、投影、光栅化以及片断处理才能得到最后的屏幕输出结果。本文的光照设计支持多边形的 Gouraud 着色和 Phong 着色方法。为了实现 Gouraud 着色，在图 1 所示的图形处理流水线的“节点染色”一级计算三角形各顶点的光照颜色，然后在“扫描转换”和“像素染色”一级对三角形各顶点的颜色进行插值，计算出三角形内各点的颜色。插值计算三角形内各点颜色的方法如图 2 所示。

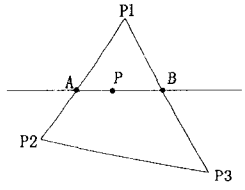


图 2 三角形内点插值示意图

假定待绘制的三角形的 3 个顶点为 P_1，P_2 和 P_3，一条扫描线与三角形的两条边分别交于 $A(x_A, y_A)$，$B(x_B, y_B)$ 两点，$P(x, y)$ 是 AB 上的一点($y = y_A = y_B$)。A 点的颜色 C_A 由 P_1，P_2 点的颜色 C_1，C_2 线性插值得到

$$C_A = \frac{|AP_2|}{|P_1P_2|}C_1 + \frac{|P_1A|}{|P_1P_2|}C_2$$

$$C_B=\frac{|BP_3|}{|P_1P_2|}C_1+\frac{|P_1B|}{|P_1P_2|}C_3$$

$$C_P=\frac{x_B-x}{x_B-x_A}C_A+\frac{x-x_A}{x_B-x_A}C_B$$

以上计算 A,B 两点颜色的过程是在“扫描转换”部分完成，在“像素染色”部分对 A,B 两点的颜色进行线性插值，可得到 P 点的颜色 C_p。

为了实现 Phong 着色，需要在“扫描转换”和“像素染色”一级对三角形每个顶点的法向量进行插值，得到三角形内每个点的法向量，然后利用光照明模型计算出每个点的颜色。由于对三角形内的每个点都要进行光照计算，因此 Phong 着色的计算量较大，但能得到更好的光照着色效果。

Phong 着色中插值计算法向量的方法与上述计算颜色的线性插值方法相同。设图 2 中 P_1，P_2 和 P_3 顶点处的法向量分别为 N_1，N_2 和 N_3，则点 A,B,P 处的插值分别用以下公式计算：

$$N_A=\frac{|AP_2|}{|P_1P_2|}N_1+\frac{|P_1A|}{|P_1P_2|}N_2$$

$$N_B=\frac{|BP_3|}{|P_1P_2|}N_1+\frac{|P_1B|}{|P_1P_2|}N_3$$

$$N_P=\frac{x_B-x}{x_B-x_A}N_A+\frac{x-x_A}{x_B-x_A}N_B$$

需要注意的是，使用光照模型计算三角形内某点的颜色时，需使用该点到光源的距离、光源到该点的向量以及该点到观察点的向量。但在“像素染色”一级，由于图形已经过投影和视口变换，点的空间坐标已不存在，无法直接计算上述几个值，只能通过对三角形顶点的值进行线性插值（计算方法与上述颜色和法向量的计算方法相同）而得到，而顶点的值是在“节点染色”一级计算出来并保存在顶点数据结构中的。以下是我们使用的顶点数据结构：

```
typedef struct {
    short light_index;//光照参数索引
    short back_face_index;//平面背面材料参数索引
    short front_face_index;//平面前面材料参数索引
    FACETYPES face;//表明仅对前面或后面计算光照，还是双面都计算
    EnumVector4 position;//顶点坐标
    EnumVector4 color; //顶点颜色
    EnumVector4 normal;//顶点处法向量
    float distance[8];//顶点到光源的距离
    EnumVector4 vNormal[8];//光源到顶点的向量
    EnumVector4 eNormal;//顶点到观察点的向量
    EnumVector4 TexCoord;//顶点对应的纹理坐标
} nVertex;
```

“节点染色”部分将光照和材料参数存入片上缓存，然后产生光照和材料参数的索引，并保存在上述 nVertex 数据结构中。“像素染色”部分需根据这些索引从片上缓存读取光照和材料参数，完成光照计算。

以下是每个光源的光照参数数据结构：

```
typedef struct {
    bool enabled;// 光源是否被启用
    EnumVector4 ambient;// 光源的环境光成分
    EnumVector4 diffuse;// 光源的散射光成分
    EnumVector4 specular;// 光源的镜面反射光成分
    EnumVector4 position;// 光源的空间位置
    EnumVector4 spot_direction;// 聚光灯方向向量
    Scalar1 spot_exponent; // 聚光指数
    Scalar1 spot_cutoff;// 聚光灯切角
    Scalar1 constant_attenuation;// 常量衰减因子
    Scalar1 linear_attenuation; // 线性衰减因子
    Scalar1 quadratic_attenuation;// 二次衰减因子
} LightType;
```

以下是材料参数数据结构：

```
typedef struct {
    FACETYPES face; // 指明物体的哪些面接受光照
    Scalar1 shininess; // 镜面指数
    EnumVector4 ambient; // 材料的环境颜色
    EnumVector4 diffuse; // 材料的散射颜色
    EnumVector4 specular; // 材料的镜面颜色
    EnumVector4 emission; // 材料的发射颜色
    EnumVector4 color_indexes;// 环境、散射和镜面颜色索引
} MaterialType;
```

图形处理流水线中的各模块都通过实现一个状态机来完成与其它模块的握手或进行内部操作状态的转换，图 3 是“像素染色”模块的光照模型状态机。

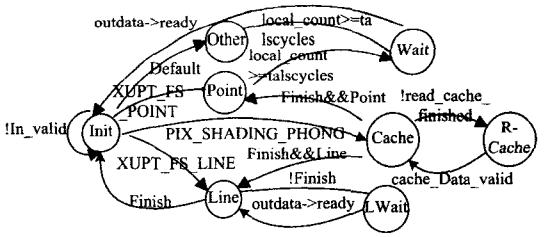


图 3　像素染色的光照模型状态机

如图 3 所示，“像素染色”模块分别在 Point 和 Line 状态下对从“扫描转换”模块传入的点和平行线进行处理。如果启用了光照并使用 Phong 着色，则需进入“Cache”状态，读取片上缓存中的光照和材料参数。

3.3　结果验证

为了验证光照效果，本文选取三维世界地图作为研究目标。最终显示出白光照射在三维地图上的明暗效果，如图 4 所示，实验结果证明了光照设计部分的正确性。

图 4　带光照的世界地图

4　纹理映射设计

4.1　纹理映射原理

纹理映射可以有一维、二维和三维纹理映射。一维纹理映射可以是线颜色映射，二维纹理可以是一幅二维图像的映射，三维纹理可以是一个实体表面的映射。其中二维的纹理映射是最常见和纹理映射的基础。因此，我们以二维纹理映射作为重点研究。

首先需要建立坐标对应关系，即需建立物体坐标、纹理坐

标和纹理图像的关系。物体坐标系的坐标轴分别记为 x 和 y，如图 5(a)所示。纹理坐标系的坐标轴分别记为 s 和 t，默认取值范围[0.0,1.0]，如图 5(b)所示。纹理图像的坐标轴分别记为 u 和 v，则分别对应图像的宽和高，如图 5(c)所示。

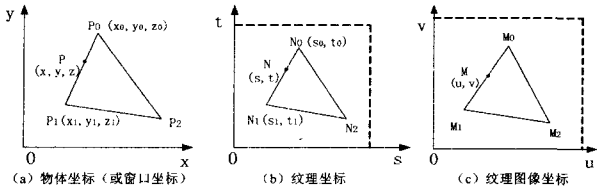


图 5　坐标对应关系

在物体坐标系中，一个三角形的 3 个顶点($P_0P_1P_2$)分别对应着纹理坐标系的 3 个顶点($N_0N_1N_2$)，同时又分别对应纹理图像坐标系的 3 个顶点($M_0M_1M_2$)，如图 5 所示。物体坐标系中的三角形会经历几何变换、投影变换、视窗变换等坐标变换。在经过上述坐标变换之后，三角形处于窗口坐标中，但 3 个顶点与对应的纹理坐标的 3 个顶点的关系不变(假定三角形的 3 个顶点都没有被三维剪裁模块剪裁掉)。

在实现纹理映射时，需要考虑窗口坐标(此时图 5(a)中 x 和 y 为窗口坐标)、纹理坐标和纹理图像坐标三者的映射关系。在窗口坐标中，在 P_0P_1 线段上，逐行扫描得到一个交点，记为：$P(x,y,z)$。然后根据双线性插值算法，计算对应纹理坐标的 $N(s,t)$，具体计算公式为：

$$s=(1-\alpha)s_0+\alpha s_1,\ t=(1-\alpha)t_0+\alpha t_1$$

式中，$\alpha=\dfrac{|PP_0|}{|P_0P_1|}$。

这种算法对于正投影变换公式是正确的。若采用透视投影变换，这时要采用双曲线插值：

$$s=\frac{(1-\alpha)z_1}{(1-\alpha)z_1+\alpha z_0}s_0+\frac{\alpha z_0}{(1-\alpha)z_1+\alpha z_0}s_1$$

$$t=\frac{(1-\alpha)z_1}{(1-\alpha)z_1+\alpha z_0}t_0+\frac{\alpha z_0}{(1-\alpha)z_1+\alpha z_0}t_1$$

在从纹理坐标向纹理图像坐标进行映射时，具体计算公式为：

$$u=s\times(width-1),\ v=t\times(height-1)$$

式中，$width$，$height$ 分别为纹理图像的宽和高，均为正整数。

在得到 $M(u,v)$ 后，纹理滤波方式有两种，分别是点采样和线性滤波。

点采样(point sampling)：找到距离 M 点最近的纹理图像像素点，用该像素点颜色值作为 M 点的纹理颜色值。

线性滤波(linear filtering)：对以 M 点为中心的一块 2×2 的纹理单元取加权平均值。

如图 6 所示，首先找到距离 M 最近的左下角纹理像素点 M_0。然后以 M_0，M_1，M_2，M_3 构成一个 2×2 的纹理单元，将 M 包围其中。以 M_0 为原点，对每个像素点分配相对坐标，则有：

$$w_i=(1-|x_i-localwidth|)(1-|y_i-localheight|)$$

$$C_M=\sum_{i=0}^{3}w_iC\quad localwidth=u-u_0,\ localheight=v-v_0$$

式中，w_i 为纹理像素点 M_i 对应的权值，x_i，y_i，w_i，C_i 分别为纹理像素点 M_i 对应的相对坐标、权值和颜色值。C_M 为 M 的纹理颜色值。

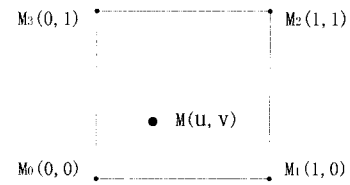


图 6　纹理滤波

4.2　纹理映射仿真实现

为了能够实现纹理映射，需要设计专门存放纹理图像的片上缓存，在顶点数据结构中设置对应的纹理坐标，即在节点染色时要将对应顶点的坐标值和纹理坐值联系起来。同时要保留纹理索引，以便于查找纹理图像。纹理图像是以二维数组存放的。纹理图像中每个像素点应包括 RGBA 数据值，其数据结构如下：

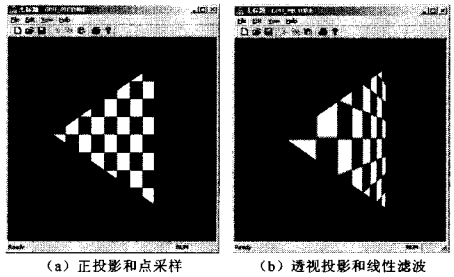
EnumVector4 TextureImage[TextureImageHeight][TextureImageWidth]

还需要定义纹理的参数，其数据结构如下：

```
typedef struct{
    unsigned int TexImageIndex;//纹理图像在 Cache 中的索引
    unsigned int TexImageHeighet;//纹理图像的高度
    unsigned int TexImageWidth;//纹理图像的宽度
    unsigned int TexNameNumber;//纹理对象号
    TextureType TexType;//纹理的类型:1D,2D,3D
    TextureParaName TexParaName;//纹理参数名字
    TextureMagFilter TexMagFilter;//纹理滤波器选择
    TextureWrapMode TexWrapMode;//纹理扩展方式
    PixelFormat PixFormat;//纹理图像像素格式
    TextureEnvTarget TexEnvTarget;//glTexEnv 函数 Target 参数
    TextureEnvParameter TexEnvParameter;//glTexEnv 函数 Paname 参数
    TextureEnvCalculatePara TexEnvCalculatePara;//glTexEnv 函数 Param 参数
    GLTYPES elementType;//纹理图像数据类型字节整型浮点型
}TextureParameter;
```

4.3　结果验证

图 7 是采用软件仿真得出的空间三角形经过纹理映射所得的结果。其中图 7(a)采用了正投影和点采样，可看出方格分布均匀且边界锐利。图 7(b)采用了透视投影和线性滤波，具有明显的透视效果，而且边界平滑。



(a) 正投影和点采样　　(b) 透视投影和线性滤波

图 7　纹理映射示意图

5　光照和纹理映射的硬件设计

本文流水线结构中对不同模块采用不同的设计，比如几

(下转第 301 页)

speculative multithreading(SPSM) architecture: compiler-assisted fine-grained multithreading[C]// Proceedings of the IFIP WG10. 3 Working Conference on Parallel Architectures and Compilation Techniques. Manchester, UK: IFIP Working Group on Algol, 1995: 109-121

[5] Madriles C, Sánchez C G Q J, Marcuello P, et al. The Mitosis Speculative Multithreaded Architecture[C]// Proceedings of Parallel Computing: Current & Future Issues of High-End Computing. 2006: 27-38

[6] Quiñones C G, Madriles C, Sánchez J, et al. Mitosis compiler: an infrastructure for speculative threading based on pre-computation slices[C]// Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation. New York, NY, USA: ACM Press, 2005: 269-279

[7] 肖刚, 周兴铭, 徐明, 等. SMA: 前瞻性多线程体系结构[J]. 1999, 22(6): 582-590

[8] 邓鹏. 前瞻多线程编译优化技术的研究与实现[D]. 长沙: 国防科技大学, 2000

[9] 鲁建壮, 王志英, 张春元. 面向 SCMP 的多线程前瞻控制分析与设计[J]. 计算机工程与科学, 2006, 28(10): 128-130

[10] Franklin M, Sohi G S. ARB: A Hardware Mechanism for Dynamic Reordering of Memory References[J]. IEEE Transactions on Computers, 1996, 45(5): 552-571

[11] Renau J, Tuck J, Liu W, et al. Tasking with out-of-order spawn in TLS chip multiprocessors: microarchitecture and compilation [C]// Proceedings of the 19th Annual Iinternational Conference on Supercomputing. NY, USA: ACM Press, 2005: 179-188

[12] Pickett C J F, Verbrugge C. SableSpMT: A Software Framework for Analysing Speculative Multithreading in Java[C]// Proceedings of PASTE'05: Proceedings of the 6th ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering. New York, NY, USA: ACM Press, 2005: 59-66

[13] Chen M, Olukotun K. The Jrpm system for dynamically parallelizing java programs[C]// Proceedings. 30th Annual International Symposium on. 2003: 434-445

[14] Kazi I H, Lilja D J. JavaSpMT: A Speculative Thread Pipelining Parallelization Model for Java Programs[C]// Proceedings of the 14th International Parallel and Distributed Processing Symposium. 2000: 559-564

[15] 冯威. 基于 CMP 的推测多线程执行机制及其编译关键技术研究[Z]. 2007

[16] Vijaykumar T N, Gopal S, Smith J E, et al. Speculative Versioning Cache[J]. IEEE Trans. Parallel Distrib. Syst., 2001, 12(12): 1305-1317

[17] Olden Benchmark Suit[EB/OL]. http://www.cs.princeton.edu/~mcc/olden.html

(上接第 287 页)

何变换、投影变换采用硬件加速器形式，光照变换和纹理映射采用可编程处理器结构。我们分析了不少图形处理的染色程序和 DSP 图像处理程序，并按照分析的结果设计了一个扩展的 VLIW 处理机。这个机器的基本结构如图 8 所示。该机器包括 8 个向量处理器和 3 个标量处理器。每个向量处理器的宽度是 16 个标量数字。在 8 个向量处理器中：3 个是通用处理器，用于实现大多数算数和逻辑运算；5 个是专用处理器。其中，VMove 用来在向量寄存器间移动数据；Reduction 用来做算数缩减运算；BSort 用来做整数排序；VMult 是向量乘法器；Select/Spread/Generate 用来从向量中选取任意单元，将一个标量值用某个函数分布到向量中去，或者按照规定函数产生一个向量。

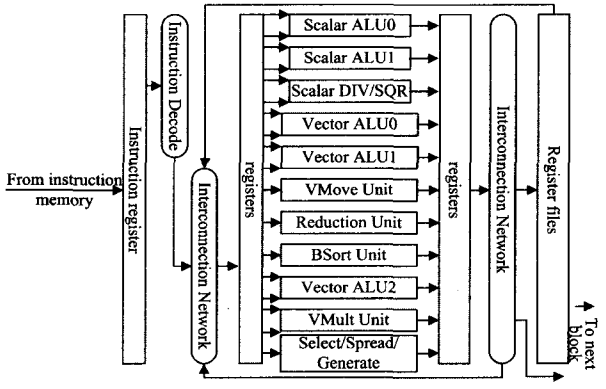


图 8 可编程处理器结构

3 个标量处理器中，一个是简单处理器，只能处理简单算数和逻辑运算；另一个的指令较多，还可以处理自动循环和转移；第三个可以执行整数除法和开平方。在这些处理器之间有数据传输网络，还有一组寄存器。处理器之间不做相关性检查，不做冒险处理，也没有 Inter-lock 机制。这些都由软件来完成。光照染色和纹理映射都用此处理器编程来实现。

结束语 本文论述了图形处理器流水线的设计思想。在提出了一条实用的图形处理流水线的基础上，研究了光照和纹理的基本原理，并进行了软件仿真和硬件设计。通过软件仿真设计结果进一步说明了我们设计的正确性。

图形处理器历经二三十年的发展，已经广泛应用于图形图像处理、视觉计算，甚至包括通用计算、大规模并行计算等等。目前图形处理器设计技术仍然掌握在国外两三家公司，可以查阅的资料较少。国内也有少量相似研究图形关键算法和硬件实现[9]，但核心技术仍受制于人。我们在这方面做了一定的研究，希望能对国内相关工作提供一些参考和借鉴。

参 考 文 献

[1] Foley J D, van Dam A, Feiner S K, et al. Computer Graphics: Principles and Practice in C(2 edition)[M]. Addison-Wesley Professional, August 1995

[2] Phong B T. Illumination for Computer Generated Images [J]. CACM, 1975, 18: 311-317

[3] Blythe D. The Direct 3 D 1 0 system [J]. ACM Transactions on Graphics(TOG), 2006, 25(3): 724-734

[4] 吴恩华, 柳有权. 基于图形处理器(GPU)的通用计算[J]. 计算机辅助设计于图形学报, 16(5): 601-612

[5] Common Shader Core(DirectX HLSL). Microsoft[OL]. http://msdn.microsoft.com/en-us/library/bb509580(VS.85).aspx

[6] Gouraud. Continuous Shading of Curved Surfaces [J]. IEEE Trans. Computers, 1971, 20: 623-629

[7] Crow. The Aliasing Problem in Computer Generated Shaded Images[J]. CACM, 1977, 20(11): 799- 805

[8] Shreiner D, 等. OpenGL 编程指南(第 6 版)[M]. 北京: 机械工业出版社, 2009

[9] 杨毅. 面向移动设备的真实感图形处理系统设计与实现[D]. 合肥: 中国科技大学, 2008

[10] 邱航, 陈雷霆. 基于点的计算机图形学研究与进展[J]. 计算机科学, 2009, 36(6): 10