[19] 中华人民共和国国家知识产权局

[51] Int. Cl.
G06F 9/44 (2006.01)

[12] 发 明 专 利 说 明 书

专利号 ZL 03808082.6

[45] 授权公告日 2007 年 10 月 10 日

[11] 授权公告号 CN 100342331C

[22] 申请日 2003.3.10 [21] 申请号 03808082.6
[30] 优先权
[32] 2002.3.8 [33] US [31] 60/363,049
[32] 2003.3.7 [33] US [31] 10/384,371
[86] 国际申请 PCT/US2003/007420 2003.3.10
[87] 国际公布 WO2003/077125 英 2003.9.18
[85] 进入国家阶段日期 2004.10.10
[73] 专利权人 电子技术公司
地址 美国加利福尼亚州
[72] 发明人 埃里克·申克 保罗·拉隆德

[56] 参考文献
US 5555201A 1996.9.10
EP 1061477A2 2000.12.20
US5781184A 1998.7.14

审查员 王 丹

[74] 专利代理机构 北京康信知识产权代理有限责任公司
代理人 余 刚

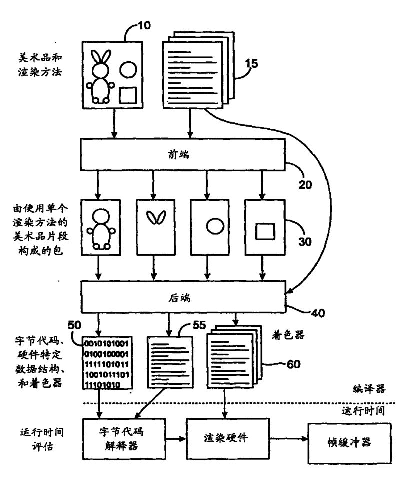
权利要求书 3 页 说明书 26 页 附图 8 页

[54] 发明名称

用于执行渲染美术品的着色器驱动编译的方法

[57] 摘要

本发明提供了预处理编译进程中执行的美术品渲染方法(图 1，#10)。利用相关的着色器程序的知识，几何数据在编译进程中进行处理。该数据转变为直接面向目标硬件平台的数据结构，且对描述渲染这些数据结构所需要的操作的代码流进行汇编。该编译器包括前端(图 1，#20)，配置为从呈独立形式的 3D 建模包和着色器读取几何数据和属性(美术品)输出，并执行独立于平台的优化；以及后端(图 1，#40)，配置为执行平台特有的优化，和执行目标平台的数据结构和代码流。



1. 一种在计算机中产生用于在运行时渲染美术品的优化代码的方法，包括:

提供美术品和一个或多个与所述美术品有关的渲染方法给编译器的前端;

在所述编译器的前端中产生一个或多个数据包，每个数据包都表示所述美术品的一个片段和对所述一个或多个渲染方法的其中之一的单独调用; 以及

使用所述一个或多个数据包在所述编译器的后端中产生优化代码和数据流,其中所述优化代码和数据流用于在运行时渲染显示器上的美术品。

2. 根据权利要求 1 所述的方法,其中所述渲染方法包括着色器输入变量的声明。

3. 根据权利要求 1 所述的方法,其中所述美术品包括多边形几何数据。

4. 根据权利要求 1 所述的方法,其中在所述后端中产生优化代码包括:

例示每个数据包，每个经过例示的数据包都包括硬变量数据和软变量数据的其中一个或全部,所述软变量数据用于识别外部数据; 以及

为所述美术品产生渲染数据结构，其中所述渲染数据结构保持与所述一个或多个经过例示的数据包的硬变量数据和软变量数据有关的变量数据。

5. 根据权利要求 4 所述的方法，其中所述渲染数据结构实质上是硬件平台指定的。

6. 根据权利要求 4 所述的方法，还包括为每个经过例示的数据包产生字节代码程序。

7. 根据权利要求 6 所述的方法，还包括:

连接每个字节代码程序，以便产生合成程序；以及

在所述合成程序上执行全局优化。

8. 根据权利要求 7 所述的方法，其中执行全局优化包括执行数据传递操作和冗余代码去除操作的其中之一或全部。

9. 根据权利要求 4 所述的方法，还包括，在例示所述数据包之前，基于与每个数据包相关的所述几何体、所述渲染方法、所述纹理、和所述矩阵调色板的一个或多个对所述数据包重排序。

10. 根据权利要求 4 所述的方法，其中例示所述数据包包括为每个数据包执行由所述被调用的单独的渲染方法指定的一个或多个转换器功能。

11. 根据权利要求 4 所述的方法，还包括，在产生所述数据结构之前，基于所述一个或多个数据包的硬变量数据和软变量数据产生词典数据结构，所述词典数据结构包括保持与所述硬变量数据和所述软变量数据相关的变量数据的存储单元的指针。

12. 根据权利要求1所述的方法，其中在所述后端中产生优化代码包括:

基于与每个数据包相关的所述几何体、所述渲染方法、所述纹理、和所述矩阵调色板的一个或多个对所述数据包进行重排序;

例示每个数据包，每个经过例示的数据包都包括硬变量数据和软变量数据的其中一个或全部，所述软数据识别外部数据;

基于所述一个或多个数据包的硬变量数据和软变量数据产生词典数据结构，所述词典数据结构包括保持与所述硬和软变量数据相关的变量数据的存储单元的指针;

为所述美术品产生渲染数据结构，其中所述渲染数据结构保持与所述一个或多个经过例示的数据包的硬变量数据和软变量数据有关的变量数据;

为每个经过例示的数据包产生字节代码程序;

连接每个字节代码程序以便产生合成程序；以及

在所述合成程序上执行全局优化。

用于执行渲染美术品的着色器驱动编译的方法

相关申请的交叉参考

本申请要求于 2002 年 3 月 8 日提交的美国临时申请第60/363,049 号的优先权，其全部内容结合于此作为参考。

背景技术

本发明总的来说涉及渲染系统，更为具体地说，本发明涉及基于着色器驱动编译方法的美术品渲染。

与诸如 Alias|Wavefront Maya™、SoftImage XSI™、和 3D StudioMax™ 等的建模应用程序情形不同，在诸如电子游戏等的用户应用程序中，大多数图形元素的拓扑都是确定的。游戏控制台和图形加速器芯片组的硬件设计员已经充分利用到了这一点，并已将它们的硬件设计为在对大的固定几何形状组渲染时比对单个多边形渲染更为有效。这一点在所使用的典型 API 中得到反映：例如，微软公司的 DirectX 8 和 OpenGL 1.1 及最新版本（例如，OpenGL:1999）都支持用于建立输入数据阵列（顶点、颜色、和其它各个顶点的属性，以及索引表）的调用，其比单个多边形提交（submission）有效得多。而且，已将多组多边形和其它渲染属性集中到显示表中，以用于以后的原子提交，其性能同样比单个多边形提交高得多。

在用户应用程序中，美术品创作是开发周期的一部分。使用某组工具对美术品进行预处理，使其成为适于应用程序的硬件和软件架构的形式。数据的预处理通常仅对几何元素进行操作。对例如光照、顶点和像素着色器选择、光栅化控制、变换矩阵等的其它渲染状态元素的设置，以及顶点缓冲器和顶点布局的选择在运行时间引擎中进行。这要求很多关于美术品使用的知识存在于代码中，以将美术品与程序员紧密联系在一起。程序员常常以牺牲效率为代价，试图使该代码得到普及对多个美术品进行处理。虽然已经将着色编译器作为这个问题的部分解决方案进行了研究，但是仍没有人利用着色器的知识系统地优化渲染技术。

工作的两个主体部分与美术品编译器的论述有关。第一个是最近针对编辑着色语言完成的工作。第二个与显示表有关。

着色语言

着色语言是 Cook 的着色树（shade tree）（Cook，R.L. 1984."Shade Trees."，In *Computer Graphics(Proceedings of SIGGRAPH 84)*，vol.18，223-231）和 Perlin 的像素流语言（Perlin, K.1985. "An Image Synthesizer.", In *Computer Graphics(Proceedings of SIGGRAPH 85)*, vol.19, 287-296）的发展。它们现在通常以 RenderMan 着色语言形式使用（Hanrahan, P. and Lawson, J. 1990. "A Language for Shading and Lighting Calculations.", In *Computer Graphics(Proceedings of SIGGRAPH 90)*，vol.24，289-298. ISBN0-201-50933-4；Apodaca, A.A. and Mantle,M.W. 1990. "Renderman: Pursuing the Future of Graphics." *IEEE Computer Graphics & Applications 10*，4（July），44-49）。着色语言近来已经适合于实时渲染图形硬件的应用。

Olano 和 Lastra（Olano，M. and Lastra,A.1998 "A Shading Language on Graphics Hardware: The Pixelflow Shading Symtem." In

Proceedings of SIGGRAPH 98, ACM SIGGRAPH/Addision Wesley, Orlando, Florida, Computer Graphics Proceedings, Annual Conference Series, 159-168. ISBN 0-89791-999-8）在其像素流系统（Molnar, S., Byles, J. and Poulton, J.1992."Pixelfow: High-Speed Rendering Using Image Composition." In *Computer Graphics(Processing of SIGGRAPH 92)*,vol.26,231-240.ISBN 0-201-51585-7）的实例中首次对面向特定图形硬件编译的 RenderMan 式语言进行描述。像素流在本质上很适合于可编程着色，但是与现今的用户级硬件有着很大的不同。

id 软件公司的产品 Quake III 整合了 Quake 着色器语言。这里，使用着色器规范（specification）来对 OpenGL 状态机进行控制。该着色器语言的目标在于确定涉及纹理单元的多过程（pass）渲染效果，并允许应用程序变量与各个过程的参数相结合。

Peercy 观察到，将 OpenGL 状态机视作 SIMID 处理器可产生用于 RenderMan 着色语言编译的帧。Peercy 等人将 RenderMan 着色器分解成结合在帧缓冲器中的一系列渲染过程（Peercy，M.S.，Olano, M., Airey, J. and Ungar, P.J.2000. "Interactive Multi-Pass Programmable Shading." *Proceedings of SIGGRAPH* 2000（July）,425-432. ISBN 1-58113-208-5）。

近来，Proudfoot（Proudfoot, K., Mark, W.R., Tzvetkov, S. and Hanrahan, P. 2001. "A Real-Time Procedural Shading System for Programmable Graphics Hardware." In *Proceedings of SIGGRAPH 2001*, ACM Press/ACM SIGGRAPH, Computer Graphics Proceedings, Annual Conference Series, 159-170. ISBN 1-58113-292-1）已经开发出一种着色器语言编译器，该编译器使用在 DirectX 8 和 NVIDIA 的 NV 顶点程序 OpenGL 扩展中可用的可编程顶点着色器（Lindholm, E., Kilgard, M.J. and Moreton, H.2001. "A User-Programmable Vertex Engine." In *Proceedings of SIGGRAPH 2001*, ACM Press/ACM SIGGRAPH, Computer Graphics Proceedings, Annual Conference

Series, 149-158. ISBN 1-58113-292-1)，以及由现代纹理合并器硬件所提供的总片段操作。通过考虑规范出现的多个等级（对象级、顶点级、或像素级），他们成功地利用了这些等级的硬件特性。

按照 RenderMan 模型，在所有上述的着色编译器中，几何数据都通过基础图形 API 与着色器通信。在 RenderMan 中几何体及其与着色器的捆绑使用 RenderMan 接口规范在程序上指定。同样，Olano 和 Lastra 的系统以及 Proudfoot 的系统通过 OpenGL API 将着色器捆绑到几何体。这要求外部系统管理着色器与几何体的捆绑，或要求显式应用代码根据美术品来对捆绑进行管理。由于这些程序既要求运行时间代码管理捆绑，又要求同步工具代码为运行时间产生适合的数据，所以这些程序比起它们乍看起来更为复杂。

显示表

美术品通常在 3D 建模和动画包中产生。这些包常常指向几何数据的交互操作以及合成对象的离线渲染。它们通常具有用于操纵几何体、拓扑结构、着色、和动画的丰富特性集。然而，原始输出模型很少适合用户级硬件。必须灵敏地创作美术品，以使其最终应用于实时用户级应用中。该美术品不仅必须从动画包存储的丰富描述进行变换，而且必须被优化和面向应用程序的硬件和软件架构。这些预处理操作包括从简单的数据变换到复杂的重新排序和优化任务。

Hoppe 指出，通过开发硬件顶点高速缓存，如何在三角形条中进行顶点的重新排序能产生更有效的渲染（Hoppe，H. 1999. “Optimization of Mesh Locality for Transparent Vertex Caching.” *Proceedings of SIGGRAPH 99*（August）269-276. ISBN 0-20148-5600-5. Held in Los Angeles, California）。Bogomjakov 和 Gotsman 指出，在事先不知道告诉缓存的大小的情况下，如何利用

顶点网格而不是三角形条开发顶点高速缓存（Bogomjakov, A. and Gotsman, C. 2001. "Universal Rendering Sequences for Transparent Vertex Caching of Progressive Meshes." In *Proceedings of Graphics Interface 2001*,81-90）。通过使用原始输入数据，这两种方法都能在渲染性能上取得双重改进。

然而，无论几何优化等级如何，都要求一定的图形硬件设置优化等级和渲染提交以获得最佳性能。早期的图形 API 通常针对绘制单独的多边形(Barrell, K. F. 1983. "The Graphical Kernel Symtem—A Replacement for Core." *First Australasian Conference on Computer Graphics*, 22-26）。OPENGL 的第一版本受到同样的限制，导致在多边形提交时产生较高的功能调用开销。出现在 OpenGL 1.1 中的 GLArrays 机制通过允许大量的多边形规范而去除这些开销的大部分。（例如，参看，OpenGL Architecture Review Board, Woo,M., Neider,J.,Davis, T. and Shreiner,D. 1999. *OpenGL Programming Guide: The Official Guide to Learning OpenGL , Version 1.2*, Addison-Wesley）。DirectX 8 的顶点流以相同原理操作（例如参看，Microsoft, 2000, *DirectX 8 Programmer's Reference*. Microsoft Press）。

尽管顶点阵列加速了几何数据的提交（submission），但是 OpenGL 和 DirectX 8 中的各种状态设置功能仍需要相当大的开销。二者都支持用来收集用于后来的原子重新提交的几何和状态设置调用的显示列表。虽然这些显示列表具有相当大的驱动器级优化潜力，但由于随后的性能限制，它们在运行期间的构造限制了显示列表优化可能达到的程度。具体而言，参数化的显示列表是有问题的。虽然单个显示列表无法进行参数化，但是显示列表可以调用自原始显示列表开始已经重建的一个或多个显示列表，从而可进行简单的参数化。然而，由于新定义的列表可能影响已经在上层显示列表中设置的任何状态，因而这种架构不允许驱动器通过这种嵌套的显示列表调用来优化状态变化。

因此，理想的是提供一种用于优化美术品渲染操作而不存在与上述方法学有关的缺陷的新系统和方法。

发明内容

本发明提供了利用着色器驱动编译技术来优化美术品渲染操作的系统和方法。

用户图形硬件的渲染性能得益于将几何数据预处理成面向（target）基础 API 和硬件的形式。然后将几何数据的各种元素与运行期间的着色程序相结合来绘制美术品。

根据本发明，提供了这样的系统和方法，其中，在编译过程中进行预处理，在该编译过程中，几何数据利用相关的着色程序知识进行处理。该数据被转换成直接面向目标硬件平台的数据结构，且对代码流进行汇编（assemble），从而描述渲染这些数据结构所要求的操作。本发明的编译器包括：前端，配置为以独立于平台的形式对从 3D 建模包和着色器输出的几何数据和属性（以下称之为美术品）进行读取，并执行独立于平台的优化；以及后端，配置为执行特定平台的优化和产生面向平台的数据结构和代码流。

有利地，编译器后端可面向各种平台，并已面向四个平台，其中三个彼此完全不同。在所有平台上，用于真实情形下的编译的美术品的渲染性能优于人工编码的（hand-coded）美术品的渲染性能。

在诸如 Sony PlayStation2™、Microsoft Xbox™、和 Nintendo GameCube™ 等的用户设备以及个人计算机中，最新一代用户级图形硬件已经普遍为用户带来高端实时图形。这种硬件主要用于电子游戏中，且在硬件架构中得到反映。

本发明的系统致力于以下要求：

—以固定的着色效果有效地渲染固定拓扑对象；

—支持对绘制的对象进行运行时间修改，但是不必修改其拓扑结构；

—开发例如顶点程序和像素组合器等的硬件性能；以及

—允许用户代码和美术品在不同的硬件平台上便携。

本发明提供了实时渲染系统，包括带有便携式 API 的小而有效的运行时间引擎，和模块化可重定目标的美术品编译器。该运行时引擎在现有图形 API 的顶部上和驱动器级上得以实施。美术品在诸如 Alias|Wavefront Maya、3D Studio Max、或 SoftImage XSI 等的某个几何建模包中进行创作。该美术品最好包括几何体、每个顶点的属性、材质集合（表面性质、颜色、纹理图使用等）和用于美术品的纹理。美术品的描述非常丰富，从而无需程序员介入，就可将美术品渲染成其最终使用的样子。

一方面，该架构的基础在于，利用不仅描述着色器程序而且描述输入到着色器的输入数据的语义学的着色器语言，将渲染的原始描述与运行时见分开。这些扩充的着色器称为渲染方法，且单个美术片可引用很多这样的渲染方法。相反地，一种渲染方法可被不止一个美术品使用。

在本发明的一个实施例中，将几何体捆绑到着色器在美术品编译器中进行显式管理。这提供了很多优点。例如，如果存在显式暴露的运行时间着色器参数，则仅需要特定美术品专用的用户代码。另外，作为上述的推论，美术家能迅速在目标平台上重复美术品，而无需程序员介入。而且，由于所有几何规范和捆绑最好离线执行，所以运行时间 API 得到了极大的简化。

在一个实施例中，对渲染对象进行了离线构造，且通过使外部变量与渲染对象进行运行时间链接和通过将对象的各个部分输出到运行时间，而进行修改来寻址（address）这些对象的参数化。这些方法允许围绕本发明的模型的参数化元素进行积极优化。

所形成的系统是快速而灵活的。其自身已经体现出与现有的定制渲染引擎一样快，或比现有的定制渲染引擎更快。使本发明的系统载入新平台比将定制游戏引擎和工具组载入新硬件快得多。

根据本发明的一方面，提供了一种在计算机中为在运行时渲染美术品产生优化代码的方法。该方法通常包括提供一件美术品和一个或多个与该美术品有关的渲染方法给编译器前端，并在编译器的前端中产生一个或多个数据包，每个数据包都表示该美术品的一个片段和对该一个或多个渲染方法中的一种的单独调用。该方法通常还包括利用该一个或多个数据包在编译器的后端中产生优化代码和数据流，其中该优化代码和数据流用于在运行时在显示器上对美术品进行渲染。

对说明书的剩余部分（包括附图和权利要求）的参考将实现本发明的其它特性和优点。下面将就所附附图详细描述本发明的进一步的特性和优点以及本发明的各个实施例的结构和操作。在附图中，相似的参考标号表示相同或功能相似的元件。

附图说明

图 1 示出了系统的一个实施例，该系统包括根据本发明的实施例的编译器。

图 2 示出了在 PS2 上执行高洛德着色的简单实例。

图 3a、3b、和 3c 示出了每秒几百万多边形和每秒提交几百万顶点索引的根据本发明的系统的性能图；图 3a 概括由根据本发明的系统所获得的一些典型的性能数量。

图 4 示出了使用根据本发明的系统的 PS2 应用的屏幕捕捉，展示了对人物、照亮的体育场、和用户群渲染器的蒙皮（skining，皮肤），所有这些都作为渲染方法执行。

图 5 示出了使用根据本发明的系统的另一场景，示出了由可选编译器前端和粒子系统产生的都市风景。

图 6 示出了本发明使用的典型硬件平台。

具体实施方式

图 1 示出了本发明的系统的一个实施例。如图所示，美术品 **10** 和着色器规范的扩展 **15**（称为渲染方法，包括着色器输入变量声明和离线计算，以从该美术品产生这些变量）被提供给美术品编译器的前端 **20**。编译器前端 **20** 将美术品 **10** 和例如多边形几何数据和模型属性等渲染方法 **15** 作为输入，并产生分段的几何优化的美术品片段，称为数据包（packet，包）**30**，其表示对渲染方法的单独调用。应当理解，利用本发明的教导可构造可选前端来寻址非多边形数据，例如样条曲面或粒子系统等。美术品编译器的后端 **40** 获得由前端产生的包 **30**，并产生用于在运行时渲染美术品的优化代码和数据流 **50**、以及硬件特定数据结构 **55** 和着色器 **60**。

在本发明的一个实施例中，在美术片编译器中对几何体捆绑到着色器进行显式管理。这提供了很多优点。例如，如果存在显式暴露的运行时间着色器参数，则仅需要美术品专用的用户代码（例如，参看下面的变量部分）。另外，作为上述的推论，美术家能在目标平台上迅速重复美术品，而无需程序员介入。而且，由于所有的几

何规范和捆绑最好离线执行（为在运行时产生捆绑而提供工具，但是由于这种动态模型远不如编译的模型效率高，所以通常不鼓励这种用法（例如，参看以下的 API 部分）），所以运行时（runtime）API 得到显著简化。

在一个实施例中，通过离线构造来渲染对象，且通过使外部变量运行时链接到渲染对象和通过将对象的各个部分输出到运行时进行修改来寻址这些对象的参数化（例如，参看下面的变量部分）。这些方法允许围绕本发明的模型的参数化元素进行积极优化。

运行时环境（Runtime Environment）

为了使美术品编译器的讨论具体化，以下给出目标运行时间环境的简要描述。图 6 示出本发明使用的硬件平台 **100** 的典型部件。如图所示，平台 **100** 包括通过系统总线 **130** 与 GPU **120** 相连的 CPU **110**。CPU **110** 和 GPU **120** 中每一个通常都包括本地芯片存储器，例如 CPU RAM 高速缓存 **115** 和 GPU RAM 高速缓存 **125**。平台 **100** 通常还包括例如硬盘驱动器或其它存储器等的存储单元 **140** 和被配置为接收各种介质 **155** 的一个或多个（例如众所周知的 CD、DVD、和软盘介质）的介质设备 **150**。一个或多个用户输入装置 **160** 可与平台 **100** 相连。用户接口装置的实例包括键盘、操纵杆、鼠标等。平台 **100** 及其各个部件通常用于产生或渲染用于在例如监控器、电视屏幕、LCD 显示器等显示装置 **170** 上显示的图像。显示驱动器 **165** 通常设置为与显示器 **170** 通信。

硬件环境的影响

编译器的目标环境包括特定硬件渲染平台以及在该平台上的运行时间程序库。本发明的运行时间程序库和编译器可在很多平台上执行，包括，例如，Sony PlayStation2™（PS2）、Microsoft Xbox™、

Nintendo GameCube™（NGC）、和 DirectX 8 PC 等平台。虽然这些架构实质上不同（例如，参看 Suzuoki, M., Kutaragi, K.Hiroi, T., Magoshi, H., Okamoto, S., Oka, M., Ohba, A., Yamamoto, Y., Furuhashi, M., Tanaka, M., Yutaka, T., Okada, T., Nagamatsu, M., Urakawa, Y., Funyu, M., Kunimatsu, A., Goto, H., Hashimoto, K., Ide, N., Murakami, H., Ohtaguro, Y.和 Aono, A. 1999. "A Microprocessor with a 128-Bit CPU, Ten Floating-point Mac's, Four Floating-point Dividers, and a MPEG-2 Decoder." In *IEEE Journal of Solid-state Circuits: Special Issue on the 1999 ISSCC: Digital, Memory and Signal Processing*. IEEE Solid-state Circuits Society, 1608; Lindholm, E., Kilgard, M.J. and Moreton, H.2001. "A User-programmable Vertex Engine." In *Proceedings of SIGGRAPH 2001*, ACM Press/ACM SIGGTAPH, Computer Graphics Proceedings, Annual Conference Series, 149-158.ISBN 1-58113-292-1），但它们共有一些基本特征，包括：

—CPU 和 GPU 是通过较窄的总线连接的独立的处理器。（注意，即使其为一体化内存架构，XBOX 相对于 GPU 仍具有有限的总线带宽。）

—GPU 是用户可编程的。（这在 Nintendo GameCube 的情形下不是严格正确的，但是丰富的预定计算元素的集合是可用的。）

—平台具有不同等级的纹理合并支持，但是所有纹理合并作为后 GPU 阶段出现，而不反馈给 GPU 或 CPU。（这样的反馈是可能的，且可在某些平台上被本发明的着色器使用，但是应当从美术品编译程序外部管理。）

记住这些特征，理想的是避免形式读/计算/写/提交给 GPU 的 CPU 操作，这些操作需要静态顶点向 GPU 的提交的总线流通量的至少 3 倍。因此，在一个实施例中，本发明将静态顶点数据和拓扑

结构的特权给予几何体。本发明也支持动画，例如可将很多变形应用在 GPU 和像素组合器中，而不要求 CPU 修改输入数据。具体而言，本发明完全支持具有任一数量的坐标帧（包括加权蒙皮（skinning）(例如，参看 Terzopoulos, D., Platt, J., Barr, A. and Fleischer, K.1987. “Elastically Deformable Models.” In *Computer Graphics（Proceedings of SIGGRAPH 87）*, vol.21, 205-214; Lander, J.1998. “Skin Them Bones: Game Programming for the Web Generation.” *Game Developer Magazine*, 11-16））的分级坐标帧动画（例如，参看 Stern，G.1983. “Bbop—A System for 3D Keyframe Figure Animation.” In *Introduction to Computer Animation, Course Notes 7 for SIGGRAPH 83*,240-243.)。

运行时 API

为了绘制作为原始对象的模型，本发明的运行时环境向用户展示了 API（例如，小的 C++API）给用户。模型是与需要以预定效果绘制它们的着色器捆绑的几何图元（例如，三角形、b-样条、点等）和状态对象（渲染状态、纹理等）的集合。模型被分成多个几何体，每个几何体独立地打开或关闭。几何体和模型直接映射到输入到编译器的美术品。美术品编译器的每次运行都产生包含一个或多个几何体的单个模型。模型可暴露一组控制变量给用户（参看下面的变量部分）。通过设置所需要的控制变量和调用模型的 \verb+Draw+方法完成模型的绘制。当系统保持控制变量的值时，仅需要在变量的值改变时设置控制变量。这是用于控制动画以及其它动态渲染效果的机制。

API 最好不暴露多边形渲染调用。用户可使用类似于 OpenGL GLArray 结构的 API 在程序上构造模型。这些称为动态模型，且具有比本发明的编译模型更差的渲染性能。

除了模型渲染外，本发明还对例如设置用于渲染的硬件的辅助工作（house keeping）进行操作。本发明还提供用来操纵硬件状态对象和纹理的实用功能。注意，这些对象位于进行编译的模型之中，但可暴露给用户。对用户来说方便的是，本发明还提供了可选视口和照相机提取（abstraction）。而且，本发明提供了经由其所暴露的控制变量而与模型连接的动画引擎。

渲染方法

渲染方法包括一组变量和使用这些变量的着色程序的规范（specification）。可对渲染方法进行充分抽象（abstract），从而同样的渲染方法可用在范围广泛的美术品上。与此相反，撰写了一些渲染方法以解决特定美术品中出现的问题。本发明的编译器架构允许人们将渲染方法应用于任何具有满足着色器输入变量所必需的数据的美术品。

图 2 示出了在 PS2 平台上执行高洛德着色渲染法的简单实例。

类型

由于使用渲染方法将任意的几何数据直接捆绑到着色器，所以对输入量和变量部分中的变量进行分类（type）。这确保了呈现给着色器的数据为要求的形式。进行分类得到的类型是用户可扩展的，并在渲染方法规范中给出。类型定义是平台指定的，且包括例如 CPU 存储器中的对象大小、GPU 存储器中的对象大小、将该类型从 CPU 传递到 GPU 所需要的硬件状态等特性。与类型有关的信息使得编译器能操纵美术品的元素，而不用对它们做出设想。在一个实施例中，本发明提供了用于所有平台的一组相同的基本类型。

输入

输入部分声明了本渲染方法所需要的数据元的类型。这些称为输入变量。在此实例中，坐标输入变量用类型坐标4来进行声明。接下来，此变量可举例来说在转换器或其它用来构造用于变量部分的数据图像的数据操纵程序中被引用。该输入声明可伴随有确切值，以分配给在美术品不提供这种输入时所使用的变量。来自美术品的输入数据通过名称捆绑到每个这样的输入变量。由编译器提供的变量包括来自美术品的顶点和材质数据，以及用户可附加到顶点和材质上的任意数据。这使得易于通过将用户数据捆绑到渲染方法输入的简单命名转换来扩展编译器的功能性。

变量

变量部分声明了计算可用的数据。变量附加有特定类型和多个元素以作为阵列规范。在缺少阵列规范时，对阵列的长度进行假定。特定的阵列长度nElem用于简单的约束系统，以使硬件约束内的元素数目达到最大。

所有变量（除了被noupload关键词标记为临时变量的以外）具有从在输入部分中声明的输入所导出的值。当使用=转换器（参数，...）语法指定显式转换器时，利用给定的参数来执行转换器功能（从用户可扩展的库中动态加载）。参数本身或者是转换功能的结果，或者是输入变量。在缺少显式转换器规范时，对身份转换器进行假定。转换器机制允许对数据元进行简单的编译时间操作。典型使用举例来说包括数据压缩操作以及各种重索引任务，和各种诸如副法线向量产生或色平衡的预处理操作。由于着色器通过本发明的编译器与美术品紧密相连，所以最好将尽可能多的着色器指定预

处理移动到这些转换器中，使得美术品转换过程尽可能均匀，以使美术品和着色器不同。

然而，不是着色器使用的所有变量都可通过编译器预处理使用。例如，例如变换矩阵等数据在编译时不可用。它们可作为隐式变量被通信。这将限制用户扩展这种变量的集合。根据一方面，渲染方法规范通过外部链接进行扩展。通过将外部关键词以及分配形式＝变量_名添加到变量声明，而引用命名为变量_名的外部变量。在图 2 的一个实例中，运行时间负责用命名为 Viewport::XFormProject 的变量的实际运行时间地址的指针，替换此变量的未解决的引用。当加载美术品时，该外部引用才得以解决。美术品最好以 ELF 格式（例如，工具接口标准（Tool Interface Standards）, 1998, ELF: 可执行及可链接格式(Executable and linkable format) ftp://ftp.intel.com/pub/tis）储存，并通过 ELF 动态装入程序提供外部链接。该库使用动态装入程序寄存多个变量，举例来说使得变换矩阵可用于渲染方法。用户也可在运行时寄存其自己的变量。

由于 GPU **120** 和 CPU **110** 可并行操作，所以对 CPU 上的外部链接的变量的改变可产生竞态条件。为了消除竞态条件且不导入不适当的锁定限制，理想的是复制可产生这种竞态的数据元。然而，由于所需要的开销，复制所有的这种变量数据是不理想的。作为替换，在本发明的一方面，对可能利用易失关键词诱导这种竞态的元素进行标记。用户可忽略该关键词，使得通过引用使用而非复制使用，以及获得更有效的执行。忽略外部链接的变量上的易失关键词通常是危险的，因为所允许的对这种变量的修改没有限制。

变量的另一重要的方面在于，它们一般而言对运行时间是不透明（opaque）的。由于编译器可能已经对数据元进行重排序、简化或将其隐藏，所以不可能检查或设置所编译的美术品内的值。虽然

这种限制使得渲染效率更高，但是可能不是充分灵活的。经常需要在 CPU 上而不是在渲染方法中检查存储在模型中的数据。在很多情形下，输出与外部变量不同的存在于美术品中的控制变量是有用的。在此情形下，用户不需要在运行时构造、寄存、和管理这些变量。

通过为声明加前缀关键词输出，将变量声明为在渲染方法中输出。由于编译器发出标记为已输出的数据，所以其添加一引用（例如，ELF 符号引用）给变量，并将其名称给与每个模型相关的词典。在运行时用户可询问模型词典以找到特定变量的地址。一般而言，这是用相同名称在多个变量上的迭代。如果变量共用同一名称和二进制映象，则仅将该变量的一个版本添加到该词典中。如果二进制映象不同但是名称相同，则二者以同一名称都添加进去。由于这些名称出现在用相同的渲染方法编译的每个包中，所以需要某一系统来区分它们。

为了允许用户区分这些变量，提供了名称扩展机制。串变量最好以变量名被引用，以在编译时扩展变量名。例如，在图 2 中，状态变量用串变量 geometry_name 的内容扩展。这些串变量使用与任何其它输入相同的机制进行通信，从而可通过编译器前端产生，或可以是与材质或输入模型相连的用户数据串。这种名称扩展机制可用于执行变量的范围界定（scoping）系统。编译器前端在模型、几何体、和包等级上提供了范围界定。通过在原始创作包中标记用户的美术品，用户可很容易地对另外的范围界定等级进行管理。

虽然输出的变量词典将指针返回到输出变量，但成本与所允许的对编译数据的修改有关。例如，在 PC 级架构上，顶点数据被复制到不能从 CPU 进行有效存取、但是对于 GPU 来说非常快的存储器中。因此，通过 CPU 对此数据所作的修改要求对数据重新复制。为了避免或减少这样的性能成本，本发明在一个实施例中要求对可

在运行时使用可修改的关键词进行修改的元素进行正式声明，防止对未这样标记的数据进行修改。该词典保存有表示变量的可修改状态的标记，并举例来说使用 C++常量机制加强了对修改的限制。可修改的标记的使用与易失标记正交。由于它允许在适当时（对较大的数据结构的不频繁的改变）通过对输出的可修改数据的引用而使用，同时仍允许在必要时（在具有不同参数设置的相同帧中的模型的重复使用、频繁改变）通过复制而使用，因此这一点是非常有用的。这种修改机制有利地提供了等价于参数化的显示表、在编译器中离线构造、和适当优化的功能性。

根据一方面，通过输出的变量词典成为可能的简单扩展是用来远程观看和修改与模型相关的输出变量的运行时间 GUI 工具。这使得美术家或程序员能调整（tweak）模型的外观和行为，而不需要对美术品进行重新编译。反过来，该编译器能将来自该工具的保存数据作为输入，用于相同美术品的后来的编译，或具有类似布局的其它美术品的编译。

计算

除了导入新的着色器语言，本发明的一方面在每个目标平台上使用本机着色器语言。为了更容易地写渲染方法，将复杂的着色器分成可再用的参数化宏。该宏在外部文件中定义，并被连接和汇编成机器特有的着色器程序。传递约定的参数将变量和相关类型信息捆绑到宏参数。这使得用户能使用现有程序段迅速建立新效果的原型。然而，高度优化的着色器要求手工制作的着色器程序。着色器编译器，例如 Proudfoot 等人或 Peercy 等人的着色器编译器，在此等级上可在此架构中采用。

编译器前端

重新参看图 1，美术品编译器的前端 **20** 获得美术品 **10** 和渲染方法 **15**，并构造渲染该美术品所需要的一系列包 **30**。在一方面，尽管其它前端可利用本发明的技术产生以处理其它类型的数据，例如样条表面、粒子系统、传统地形网格、以及传统城市渲染系统等，但本实例的前端 **20** 专门对由多边形组成的美术品进行处理。

编译器前端 **20** 将美术品 **10** 分成已经由美术家定义的几何体。在每个几何体内，多边形最好被收集并根据材质和顶点属性进行分类。

材质对于所使用的美术品包是内在的，且可包括任意的用户定义的数据以及标准的预定义材质属性。在一方面，前端还负责识别实际上相同的材料并将它们合并。

顶点包括例如坐标、纹理坐标、法线（normal）等属性及其它的公知属性，但是也可包括任意的用户定义的数据。

在将多边形分类后，接下来，编译器选择一种渲染方法以与每种类型相关联。每种类型包括一组分别具有名称的材质属性、包括一组命名的数据元的顶点的集合、以及由顶点组成的多边形的集合。在一个实施例中，通过在一列可用的渲染方法上进行重复，直到找到一种未定义输入可由类（class）中的可用数据满足的方法，而选择渲染方法。为了提供更好的控制，材质也可附加有请求的渲染方法。如果所请求的渲染方法的未定义输入不能被可用数据满足，则应用缺省机制，且发出警告。

一旦一种类型已经与渲染方法相关，则编译器最好从该类中的数据建立一个或多个包。在某些方面，基于可限制可传递给 GPU

的数据元数目的硬件限制，将数据切成较小的包。例如，在某些平台上这是基于 GPU 存储器的大小的硬限制。在另外的平台上，它是用实验方法确定的最佳效率点。某些平台将数据流与常数元区分开。这反映在分段计算中。一旦被构造，则该包被传递到包编译器层。

将类（class）中的数据分成包的过程包括执行用于有效地渲染基础多边形的三角形剥离或网格重排（例如 Hoppe, H. 1999, “Optimization of Mesh Locality for Transparent Vertex Caching.” *Proceeding of SIGGRAPH 99*,（August）269-276）.ISBN 0-20148-5600-5. Held in Los Angeles, California), 并且可能需要对必须出现在不止一个包中的顶点数据进行克隆。

本发明的分包过程中的一个步骤是使多个坐标帧与顶点相连。特征动画系统允许每个顶点与一大组坐标帧的其中之一相连。这些坐标帧反过来由一较小组的表示动画轮廓的坐标帧构成。由于 GPU 上的存储限制，出现在每个包中的独特的坐标帧的数量优选受限。这组坐标帧称为矩阵调色板。这将网格优化步骤转换成多维优化问题：同时使硬件上所要求的顶点变换的数据和诱导的矩阵调色板的数据最小。一个条通常不能扩展为特定三角形，因为这将导致包所要求的矩阵调色板超过最大调色板尺寸。

包编译器

发送在运行时解释的一列包可造成运行时间性能较差。存在两种优化这种方案的方法。一种方法是优化运行时间环境，执行例如最小化硬件状态的修改、纹理使用的重排渲染、对再用的计算结果缓存等策略。然而，由于待渲染的数据是提前知道的，所以可离线执行这些优化策略中的多个策略。

包编译器负责将由前端产生的包转换成数据和可被执行以渲染美术品的相关代码，而无需程序员的外部介入。由包编译器产生的代码是定制为准确渲染输入美术品的优化程序。注意，包编译器不使用关于拓扑结构的信息。因此，它可用于编译任意的数据集，而不仅仅是多边形数据集。

输出形式可根据不同的平台而完全不同。除了这些差别外，在编译器后端中存在共同的结构。例如，后端总是产生模型对象，该模型对象包括以下各项：必须被执行以渲染模型的字节代码流的指针；指向从在表示模型的包中使用的渲染方法输出的数据的词典；以及将在加载时间解析为（resolve）指针的输入数据的外部引用。这些可出现在字节代码中，或其它硬件特定数据结构中。另外，字节代码包括对包含渲染所需要的信息的硬件特定数据结构的引用。

对于每个平台，用于渲染速度的硬件特定优化在产生的字节代码和数据结构上执行。这些优化很大程度上依赖于模型的渲染可作为原子操作，因此，硬件状态在模型渲染过程中，在提交给硬件的每个包之间被完全控制。

在一个实施例中，后端被配置为在如下几种过程（pass）:

过程 1：包排序。

过程 2：变量数据构造。

过程 3：输出数据聚集。

过程 4：数据结构产生。

过程 5：代码生成。

过程 6：数据结构和代码的全局优化。

过程 7：代码和数据发射。

过程 1—数据包排序

为了有效地使用基础硬件，对包进行重新排序以使硬件状态中的昂贵改变最小化。对重排序进行限制，以允许用户保持对渲染顺序的控制。具体而言，应保证模型的几何体将以其出现在美术品中的顺序被渲染。在一个实施例中，执行简单的试探以使昂贵的状态改变最小化。数据包首先以几何体、然后以渲染方法、然后以纹理、最后以矩阵调色板进行分组。一方面，更复杂的编译器举例来说可检查所产生的代码流，并对其操作成本进行建模，以确定最可能的数据排序。

过程 2—变量数据构造

前端提供一列包给后端，其中每个包都具有相关的渲染方法和输入数据集。输入数据不是最终输送给着色器的数据，因此其必须被转换成在与包相关的渲染方法的变量部分中所定义的数据。这通过执行由渲染方法特有的转换器功能而实现。其结果是经过例示的数据包（instantiated packet）。在经过例示的数据包中，或者用于每个变量的数据是已知的，或者将在运行时解析到该数据的存储单元的外部符号引用是已知的。对将完全已知的数据内容作为硬数据的变量进行引用。仅被外部声明（输入数据）定义的变量称为软数据。在此阶段，编译器还将符号名分配给每个变量数据。这些符号名用于指示包含数据的存储单元，并在必须产生对数据的直接引用时用于剩余的过程。在软变量数据的情况下，符号名是由外部声明在渲染方法中定义的名称。

一方面，尽管在这里符号名被分配给每个数据块，但是数据本身既不被发送也不放入特定数据结构中。这最好在后面的过程中执行。

过程3—输出数据聚集

这个过程聚集了与可在运行时用于找到输出变量的模型相关联的词典数据结构。分配到先前过程中的数据的符号名此处用于填充所形成的词典中的指针。

过程4—数据结构产生

在此过程中，产生了保持由例示包引用的硬软数据的渲染数据结构。在很多目标平台上，理想的是尽可能直接地供给基础渲染硬件。这有利地允许避免设备驱动器系统开销和数据的不必要的操作。这可通过以尽可能近的本地形式构建数据结构而实现。

例如，在PS2平台上，连锁式(chained)直接存储器存取(DMA)单元将数据供给与CPU平行操作的GPU。该DMA单元支持嵌套的CALL结构，该结构与过程调用非常类似。这允许用嵌在DMA链中的渲染数据预先构建较大的DMA链片段。其一个优点在于，CPU不需要总是接触这些预汇编的DMA片段中的数据，仅在渲染时将CALL操作与DMA片段链接在一起。另一优点是由于不需要额外的数据副本，所以降低了模型提交所需要的存储器开销。

在Gamecube平台上，构建直接输送给硬件的类似数据结构。在XBOX和PC平台上，对顶点缓冲器和硬件命令流进行预汇编。

过程5—代码产生

在代码产生过程中，针对执行渲染包含在包中的数据所需要的一组 CPU 操作的各个例示包产生了字节代码程序。在下一过程中，连接字节代码程序，且通过所生成程序执行全局优化。

字节代码最好用于表达这些程序，而不是本地汇编指令。解释该字节代码中的开销是最小的，且被字节代码解释程序与 CPU 上的指令高速缓存相符合这一事实所抵销（offset）。在某些硬件上，字节代码解释比执行在线机器指令流要快。这是由于指令高速缓存失误（miss）和过程调用的减少而造成。而且，字节代码具有两个主要的优点。首先，字节代码中的程序非常紧凑。其次，由于本发明的字节代码指令集非常小（例如 10 到 20 条指令，这取决于平台），所以很容易将该字节代码写入优化程序。

用于平台的指令集取决于基础硬件。例如，在 PS2 平台上，单个 DMA 链被提交给为整个场景进行渲染编码的硬件。此硬件上的字节代码指令执行适合（gear toward）汇编该 DMA 链的简单操作，以及产生 DMA 链中需要的数据的更多的 CPU 加强操作。前者的实例包括将对固定的 DMA 数据块放入链中，并将易失数据直接复制到链中。后者的实例包括将新的顶点着色器程序上载到硬件，并根据动画数据计算用于动画对象的矩阵调色板集。在编译器和硬件之间具有多个 API 的平台上，字节代码与对基础渲染 API 的调用严密对应，例如，顶点流的设置、状态设置调用、和渲染提交指令。

过程 6—全局优化

在此过程中执行的具体优化取决于目标平台的性质。这些优化分成两类：数据传递优化和冗余代码去除。

在执行数据传递优化中，理想的是去除传递给 GPU 的冗余数据。事实上，这是冗余代码去除的一个具体例子。在一个实施例中，

这通过在执行对模型进行渲染时模拟 GPU 存储器的内容和指出不改变存储器图像的上载而完成。举例来说，这种优化步骤去除变换矩阵从一个包到下一个包的冗余设置。

由于 GPU 执行没有详细模拟，因此仅提供了将数据上载到 GPU、对 GPU 何时将修改存储单元的内容作出提示，以强迫上载入该位置。将渲染方法变量标记为对优化程序的提示的两个关键词包括：noupload 和暂时。关键词"noupload"表示变量是根据需要被 GPU 使用的临时变量。关键词"暂时"表示必须在运行着色器程序之前设置，但是通过执行着色器程序被修改的变量。

与数据上载优化一样，机器寄存设置指令和字节代码指令的类似优化也是理想的。例如，在 PS2 平台上记录的是仅与其字节代码中的 CALL 指令一起出现的连续包，并将用于包的 DMA 链合并在一起。这使得仅用几何字节代码指令就可提交非常大的模型。

作为另一实例，在 PS2 平台上，数据传递机制自身是必须被适当设定，以在数据被输送给 GPU 时对该数据的状态机进行编码。在此情形下，对数据传递硬件进行模拟，以找到将数据传递硬件设定为理想状态所需要的最小寄存改变集合。这可使在 PS2 上渲染时间减少多达 20%。

如本领域中的技术人员将理解的，许多其它的具体优化均在所支持的各种平台上使用。

图 3a、3b、和 3c 示出在每秒几百万多边形和每秒提交几百万顶点索引下本发明的系统的性能图。参与者（player）分别是没有用纹理或光照绘制的 3998 多边形模型、用纹理和光照绘制的 3998 多边形模型、以及用纹理和光照蒙皮的 3998 多边形模型。兔子模型包括 69451 个多边形。（经斯坦福计算机图形实验室允许的兔子

模型。PC是带有ATI Radeon 8500图形加速器的1.4Ghz AMD Athlon计算机。)

图4示出了使用根据本发明的系统的PS2应用的屏幕捕捉，展示了对人物、照亮的体育场、和用户群渲染器的蒙皮（skinning），所有这些都作为渲染方法执行。

图5示出了使用根据本发明的系统的另一场景，示出了由用户开发的可选编译器前端和粒子系统产生的都市风景。

图3a总结了通过根据本发明的系统获得的一些典型性能数目。对于产生美术品来说，这些图是可持续的吞吐率（throughput rate）。为了进行比较，用于斯坦福兔子模型的图也包含在内。CPU在这些实例中基本空闲，这些实例对于CPU的交互应用使用是必需的。这些系统上的瓶颈通常在GPU的总线、变换和rasterization引擎上。在某些情形下，通过CPU的增强使用，可获得更好的性能数目，但是这不代表所需要的使用情形。本发明的系统所获得的性能通常自始至终都较好或很好，在CPU使用上比试图替换的定制渲染引擎好的多。

所示出的系统自身易于进入（port）不同的新架构。将使用本发明的产品从一个平台移到另一平台是容易的。利用完全的产品已经获得了仅为一周的周转时间。通常来说，产品进入需要约一个月，包括考虑平台性能差异美术品所需要的版本修改。

由于用户通过写下其自己的渲染方法极大地扩展了系统的功能性，渲染方法范例也已经证明是成功的。这包括例如群众渲染系统（参看图4）、多粒子系统（实例参看图5）、以及专门的多纹理多过程效果等特性。

此外，在编译器前端语义学不足以支持用户的需要时，可利用本发明的教导开发新的前端。例如，新前端已经被开发并成功地配置。举例来说，图5所示的都市风景包含共享其几何数据的大部分的多个对象。使用提供给外部链接的变量的异常分支，写下允许该数据在模型之间共享的用户前端。

包含在本文中的所有参考全部结合于此。

应当理解，用于执行本发明的全部或部分的代码可在例如CD、DVD、或软磁盘等计算机可读介质上存储和提供，以进行分发。这样的代码包括用于控制平台智能（例如，CPU **110** 和/或 GPU **120**）的指令，以执行本发明的各种性能和方面的一些或全部，如本文中所教导的，包括编译方面和渲染方面。

尽管通过实例和具体实施例描述了本发明，应当理解，本发明不限于所披露的实施例。相反，本领域的技术人员容易知道，除了上述的那些之外，其旨在覆盖各种修改和相似的设置。因此，应对所附权利要求书的范围进行最广泛的解释，以包括所有的这种修改和相似设置。

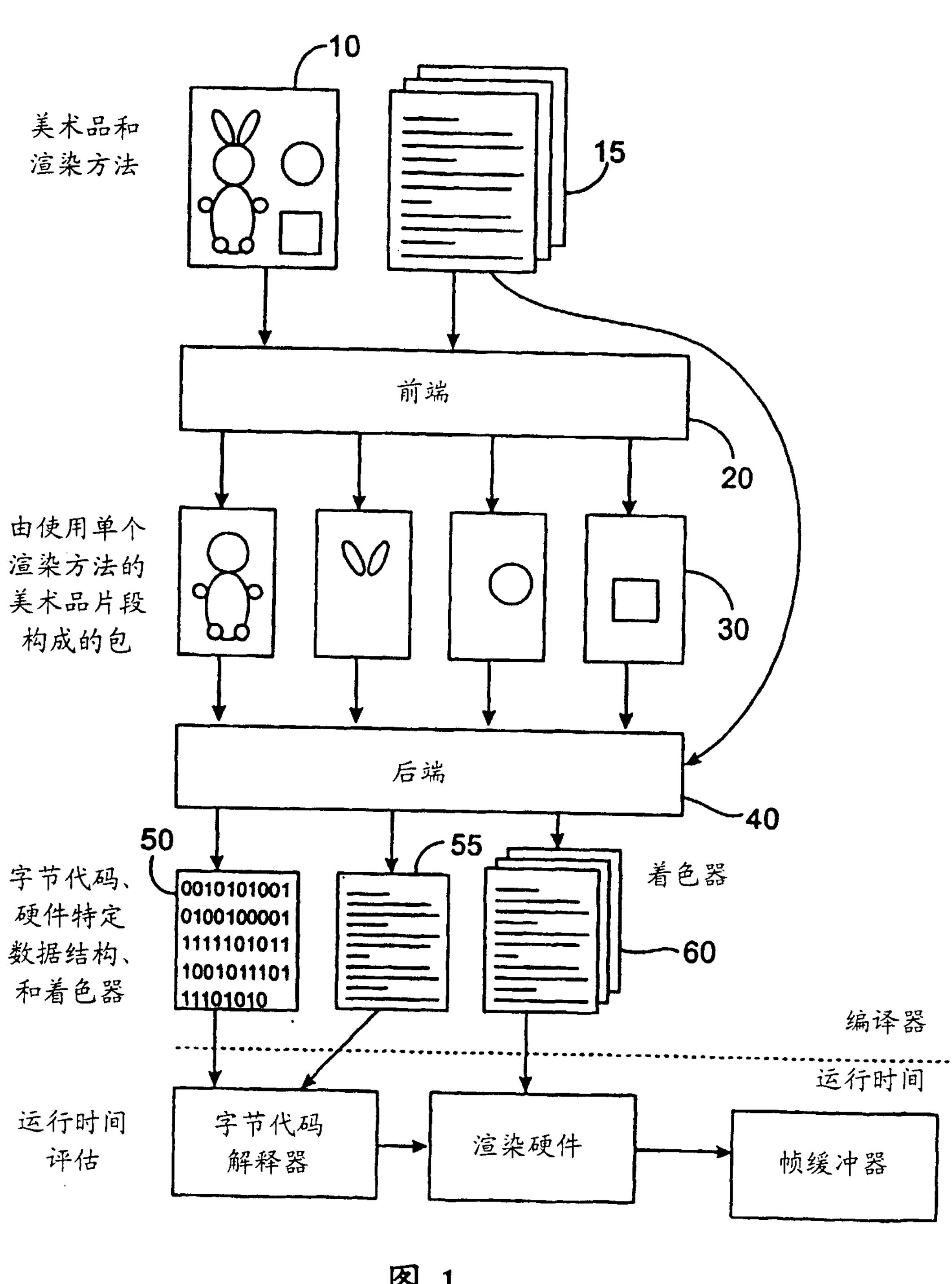


图 1

输入部分描述了来自美术品的对渲染方法可用的变量。变量部分描述了被渲染器使用的变量。计算部分描述了渲染器程序。

（The inputs section describes the variables made available to the render method from the art asset. The variables section describes the variables used by the shader. The computations section describes the shader program.）

```
#include "types.rmh"
rendermethod gouraud {
  inputs {
    Coordinate4 coordinates;
    Char geometry_name;
  }
  variables {
    int nverts;
    extern volatile Matrix xform_project
      = Viewport::XFormProject;
    PackCoord3 coords[nElem]
      = PackCoordinates(coordinates);
    ColourARGB colours[nElem];
    export modifiable
          State geometry_name::state;
    noupload RenderBuffer output[nElem];
  }
  computations {
    TransformAndColour(nverts, coords,
      colours, xform_project, output);
    XGKick(geometry_name::state);
    XGKick(output);
  }
}
```

图 2

平台	Gouraud	Lit	Skinned	Gouraud
PS2	17.0/22.6	10.9/14.7	8.5/11.5	25.2/31.8
XBox	47.2/91.4	22.4/43.4	14.2/30.3	63.9/93.8
NGC	18.7/NA	10.3/NA	7.2/NA	NA/NA
PC	24.1/46.1	15.9/20.9	5.1/10.9	26.3/36.2

图 3a

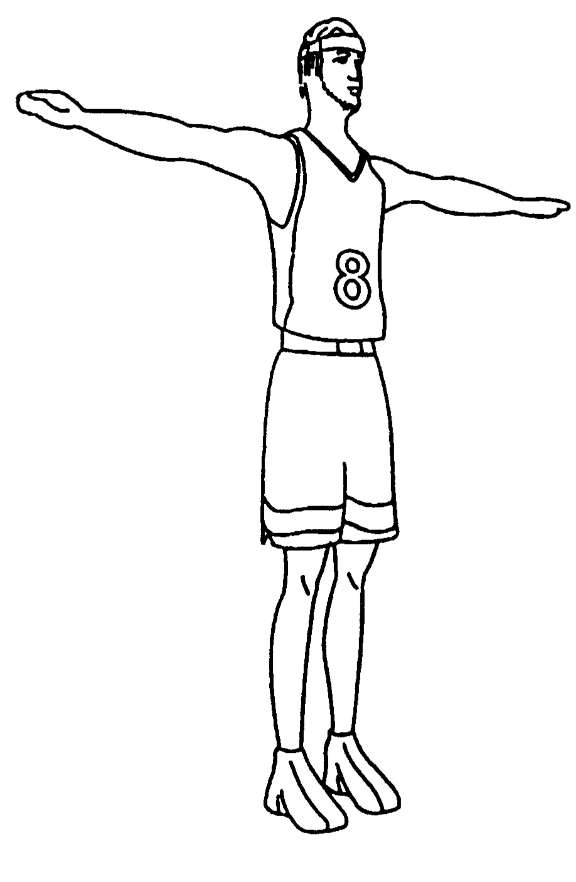


图 3b

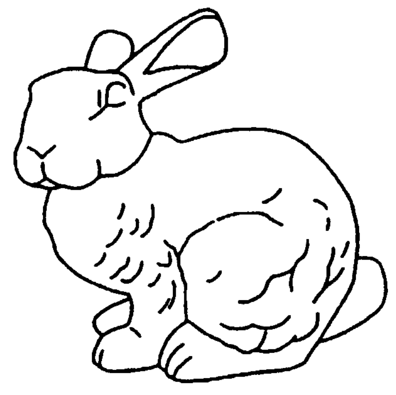


图 3c

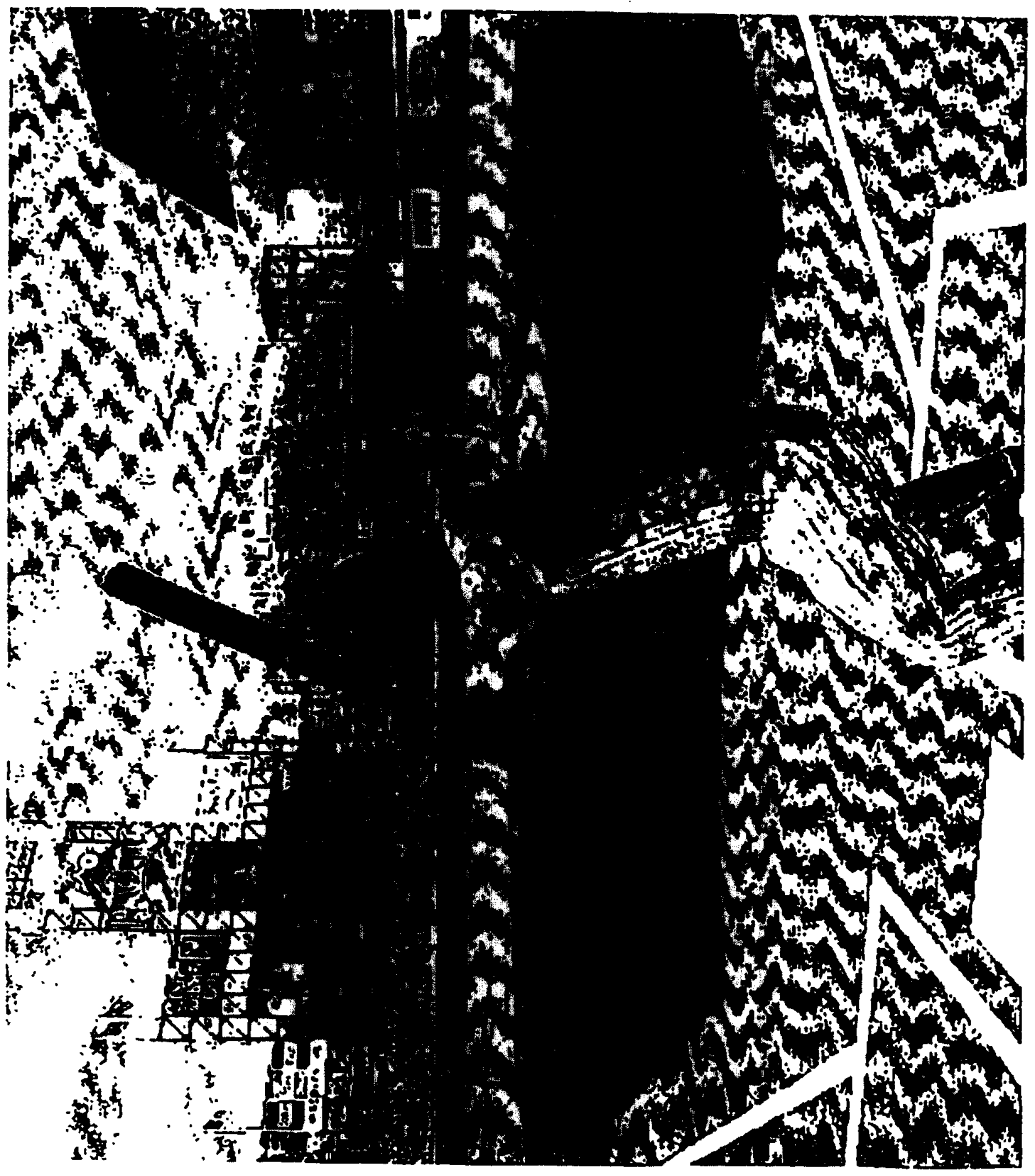


图4

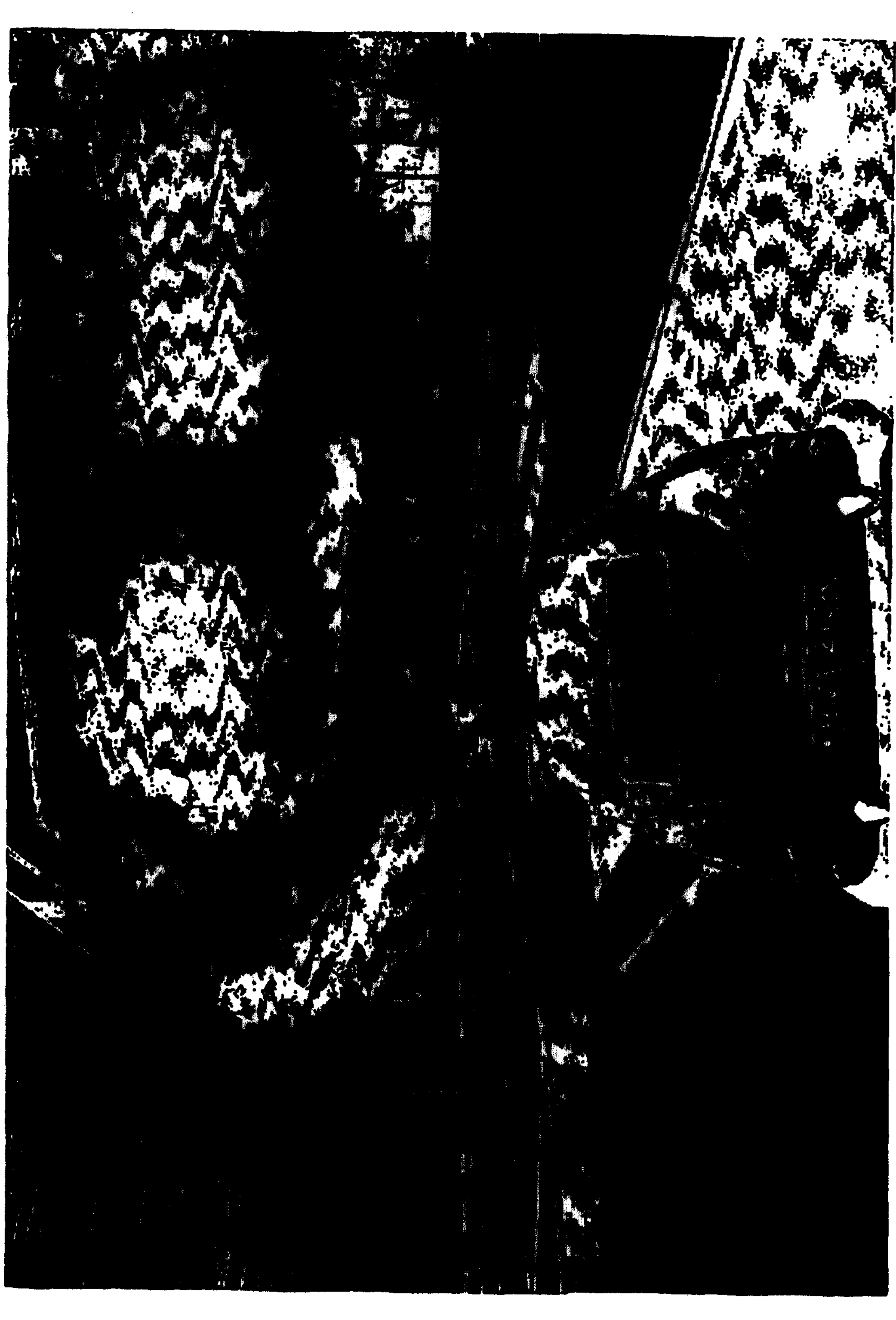


图5

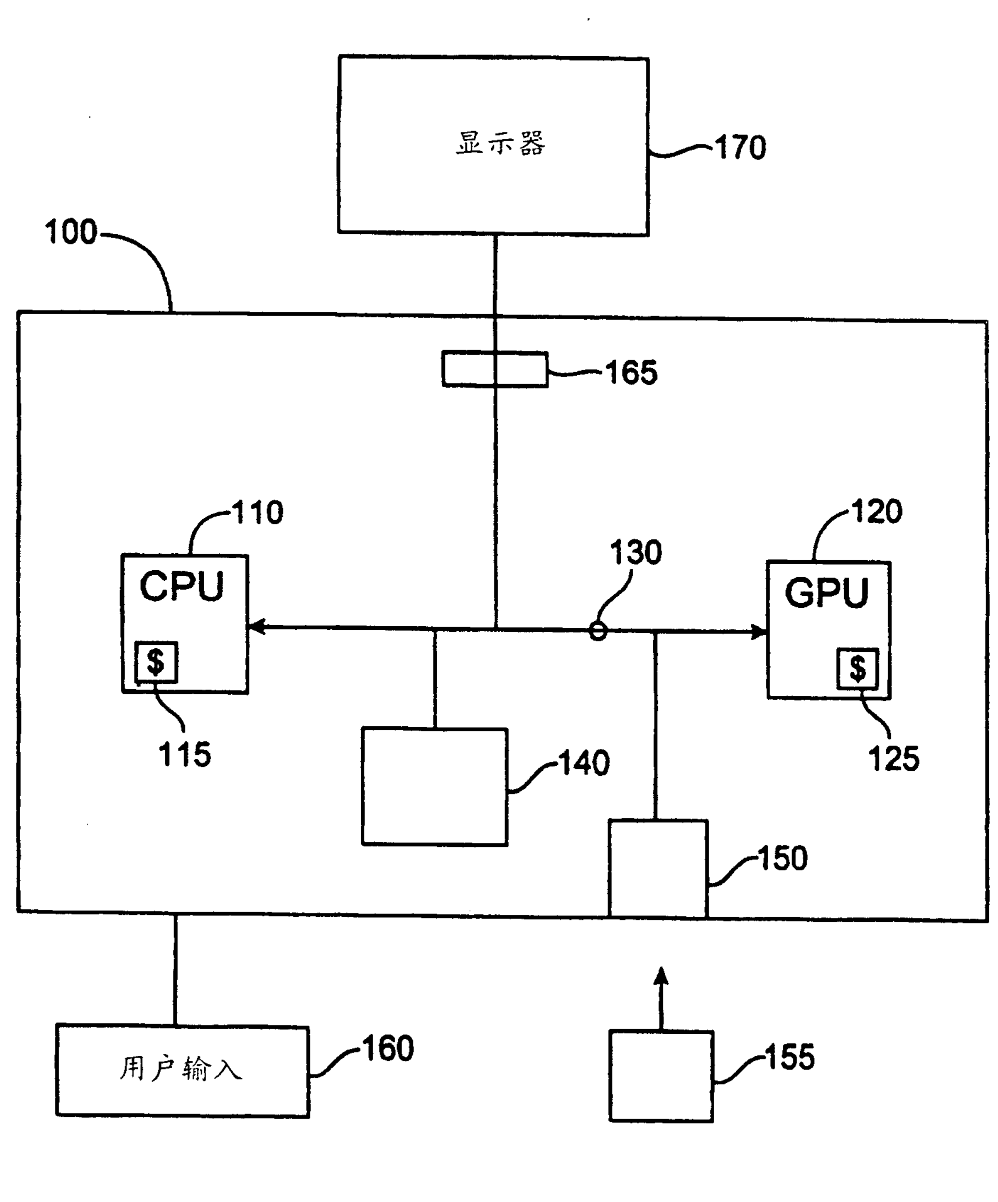


图 6