[19]中华人民共和国专利局

[11] 公开号 CN 1116009A

[12] 发明专利申请公开说明书

[21]申请号 94190814.3

[51]Int.Cl[6]

[43]公开日 1996 年 1 月 31 日

G06T 15/10

[22]申请日 94.1.3

[30]优先权

[32]93.7.27 [33]US[31]08/097,602

[86]国际申请 PCT/US94/00007 94.1.3

[87]国际公布 WO95/04330 英 95.2.9

[85]进入国家阶段日期 95.6.21

[71]申请人 塔里根特公司

地址 美国加里福尼亚州

[72]发明人 约翰·彼特逊

拉杰夫·杰恩

[74]专利代理机构 北京市中原信达知识产权代理公司

代理人 余 朦 余 刚

权利要求书 3 页 说明书 25 页 附图页数 7 页

[54]发明名称 面向目标的再现系统

[57]摘要

本发明涉及以灵活方式在具有存储器并与显示器相连的计算机上处理图形目标的方法和系统。通过把图形目标装入存储器并根据与边缘和三角数据（碎片）分离的模型方式对顶点数据进行的处理以提高再现处理效率的方式来再现图形目标以实现本发明的目的。该方法和系统包括在具有存储器和相连的显示器的计算机上处理图形目标的能力。处理始于接收和存储多个顶点，然后根据这些顶点生成一个表面的三角形。最后用三角形再现表面以便在显示器上形成该表面。

权利要求书

1. 在具有存储器和显示器的计算机上进行图形目标处理的方法，其特征在于包括以下步骤：

(a)接收多个顶点并将其存入存储器以使每个顶点的Z值具有蕴含的X值和Y值；

(b)从多个顶点中生成一个表面的三角形；以及

(C)用三角形再现表面以便在显示器上形成该表面。

2. 权利要求1所述方法，还包括为每个顶点计算法线和浓淡值并用法线和浓淡值改进表面再现的步骤。

3. 权利要求2所述方法，还包括为每个顶点生成一个标帜以控制由于限制而导致的可见性的步骤。

4. 权利要求2所述方法，还包括为每个顶点生成一个标帜以控制由于后面导致的可见性的步骤。

5. 权利要求2所述方法，还包括为每个顶点生成一个标帜以控制一个特定三角形边的可见性的步骤。

6. 权利要求1所述方法，还包括存储每个顶点的色彩信息的步骤。

7. 权利要求1所述方法，还包括存储每个顶点的结构信息的步骤。

8. 权利要求1所述方法，还包括存储每个顶点的浓淡信息的步骤。

9. 一种处理图形目标的装置，其特征在于包括：

(a)一个计算机；

(b)一个位于计算机中的存储器；

(c)一个与计算机相连的显示器；

(d)一个结构装置，用于生成一个图形目标并将图形信息存在存储器中；

(e)该结构装置包括从图形信息中生成一个表面的三角形的装置；以及

(f)该结构装置包括用三角形再现图形目标的表面从而在显示器上形成该表面的装置。

10. 权利要求 9 所述装置，还包括接收多个顶点并将其存入存储器以便每个顶点的 Z 值具有蕴含的 X 值和 Y 值的装置。

11. 权利要求 10 所述装置，还包括计算每个顶点的法线和浓淡值并据其改进表面再现的装置。

12. 权利要求 11 所述装置，还包括为每个顶点生成一个标帜以控制由于限制导致的可见性的装置。

13. 权利要求 11 所述装置，还包括为每个顶点生成一个标帜以控制由于后面导致的可见性的装置。

14. 权利要求 11 所述装置，还包括为每个顶点生成一个标帜以控制一个特定三角形的边的可见性的装置。

15. 权利要求 9 所述装置，还包括存储每个顶点的色彩信息的装置。

16. 权利要求 9 所述装置，还包括存储每个顶点的结构信息的装置。

17. 权利要求 9 所述装置，还包括存储每个顶点的浓淡信息的装置。

18. 一种在具有存储器并与显示器相连的计算机上处理图形目标的方法，其特征在于包括以下步骤：

(a)生成一个图形目标并将与该目标相关的图形信息存在存储器中；

(b)从图形信息中生成一个表面的三角形，以及

(c)用三角形再现图形目标的表面以便在显示器上形成该表面。

19. 权利要求 8 所述方法，还包括把多个顶点接收到存储器中的步骤。

20. 权利要求 19 所述方法，还包括存储每个顶点的系数的步骤。

21. 权利要求 20 所述方法，还包括根据每个顶点的系数把色彩分配给再现的表面的步骤。

22. 权利要求 21 所述方法，还包括根据每个顶点的系数把浓淡分配给再现的表面的步骤。

说　明　书

面向目标的再现系统

本发明涉及计算机系统改进，尤其涉及在一个图形显示上再现表面的系统和方法。

本申请的部分内容受版权保护，版权持有者允许用在专利商标局的专利文档或记录中出现的专利文件或专利公开说明书来复制和传真，但要保留所有版权权利。

表面(surface，是指在三维图形中构成一个三维目标最外层边界的层)的图形处理是以美学和人机工程方式表达信息的重要问题。因此，为了以人机工程方式进行信息处理，采用设计来处理不同信息(包括文本和图形信息)的信息处理系统的现代图形系统变得越来越复杂。

先有技术的软件操作系统结构受到其表面再现(rendering，是指在三维图形中把一个模型从一给定摄象位置转换成基于屏幕的视图的处理)能力的限制。这种限制是指该操作系统结构不能给定外设，如果结构未被设计或不能被修改以支持该外设的话。进而，先有技术结构只能以单一的预定方式处理表面再现信息。

因此，本发明的一个主要目的是在具有存储器以及显示器的计算机中以灵活方式对图形目标进行处理。通过把图形

目标装入存储器，并根据顶点数据以模块方式（与边缘和三角数据（碎片）相分离）再现图形目标以提高再现处理的效率的方法以实现本发明的目的。

本发明的方法和系统包括在具有存储器和显示器的计算机上处理图形目标的能力。该处理始于接收多个顶点并将其存储起来。然后，根据这些顶点生成多个用于表面的三角形。最后，用三角形再现表面以便在显示器上形成表面。

图 1 是根据本发明最佳实施例的个人计算机系统的框图；

图 2 展示了根据本发明最佳实施例的碎片；

图 3 示出了根据本发明最佳实施例的各种表面再现；

图 4 是根据本发明最佳实施例的详细逻辑流程图；

图 5 是根据本发明最佳实施例的表面再现系列。

图 6 是根据本发明最佳实施例从一个三维数字化仪获取数据的系统；

图 7 示出了根据本发明最佳实施例测定的 Z 坐标以及蕴含的 X 和 Y 坐标；

图 8 示出了根据本发明最佳实施例从一个有规律的间隔格栅得到的碎片的布局；

图 9 示出了根据本发明最佳实施例的支持物，它作为一有限元素模型而被分析。

图 10 示出了根据本发明最佳实施例的一个网状模型；以及

图 11 示出了根据本发明最佳实施例利用应变能的彩色

显示得到的表面再现。

本发明最好由个人计算机操作系统的上下文实现。这类计算机包括 IBM PS/2 或 Apple Macintosh 计算机。示意性的硬件环境如图 1 所示，它示出了一个根据发明主题的典型硬件的计算机结构。该计算机包括一个具有一个内部非易失性存储器 11 的中央处理器 10(例如常规的微处理器)以及多个与系统总线 12 相连的其他部件。图 1 所示的计算机工作站包括 RAM 14，ROM 16，用于连接外围设备(例如磁盘 20 和软盘 21)到总线的 I/O 适配器 18，用于连接键盘 24，鼠标器 26，扬声器 28，麦克风 32 和/或其他用户接口设备(例如触摸屏幕图中没显示)到总线的用户接口适配器 22，用于把计算机连接到数据处理网络 23 的通讯适配器 34，以及把总线连到显示设备 38 的显示适配器 36。该计算机装有象 Apple System/7 之类的操作系统。

在一个最佳实施例中，本发明由面向目标的编程技术以 C^{++} 语言实现。正如本领域的技术人员所知，面向目标的编程(OOP)的目标包括数据结构和在数据上的操作的软件实体。这些元素一起使目标分别成型为任意实际的现实实体。该现实实体是根据由数据元素表示的特性以及由其数据操纵功能表示的行为所成型的。以这种方式，目标可成型为具体事物，例如人和计算机；它们也可以成型为数字或几何概念之类的抽象概念。目标技术受益于三个基本原理：封装(encapsulation)，多形性(polymorphism)以及继承(inheritance)。

目标隐藏，或封装，它们的数据的内部结构以及使其功

能工作的算法，代之于展现这些实施细节，目标表达表示其抽象而无其他信息的界面。多形性较之封装又进了一步，该概念是许多形状而一个界面。一个软件部件可以生成另一个部件的请求而无需准确知道那个部件是什么。接收该请求的部件对其进行翻译并根据其变量和数据判定如何执行该请求。第三个原理是继承性，它使开发者可以再次使用以前所有的设计和编码。该特性使开发者不必由擦除来生成软件。反之，通过继承，开发者得到具有继承行为的子分类，然后由开发者定制以满足其特定需求。

先有技术的途径是将目标和分类库在一个过程环境中分层。在市场上的许多应用程序结构采用这种设计方式。在这种设计中，在一个单一操作系统的顶部有一个或多个目标层。虽然该方式利用目标分层中的所有封装，多形和继承原理，且实际上是对过程编程技术的实质改进，但其仍具有局限。这些困难来自于用户易于再使用其自己的目标，而难于使用来自于其他系统的目标，而且开发者仍然需要由过程操作系统调用进入较低的非目标层。

面向目标编程的另一方面是应用程序开发的结构(framework)。对结构最合理的定义之一是由易里露易斯大学(University of Illinois)的强森(Ralph E. Johnson)和普渡大学的卢梭(Vincent F. Russo)作出的。在题为“重新使用面向目标的设计(Reusing Object—Oriented Design)”一文中，(该文章发表于 1991 年的 University of Illinois tech report (UIUCD-CS91—1696)，它们提出以下定义：“一个抽象分类是设计一

组目标，这些目标共同执行一组任务。因此，一个结构是一组共同执行定义的计算任务组的目标分类”。从编程观点出发，结构实际上是多组内连的目标分类，这些目标分类提供了工作应用程序的预先制作结构。例如，一个用户界面结构可以提供图形视窗，卷动条，项目表等支持和“系统预置”行为。由于结构基于目标技术，该行为可被继承和置换以使开发者扩展该结构在特定的知识领域生成定制的方案。由于程序员不改变原始码而即可扩展该软件，这便成为一个超越传统编程的主要优点。此外，开发者不会盲目地通过编码层次工作，因为该结构提供了体系结构指导和模型，但同时又将它们与提供与问题范畴相关的特定行为分离开来。

从商业角度来看，结构可视为封装或实施一个特定的知识领域中专门知识的方式。企业开发机构，独立的软件销售商以及系统集成商已经获取了特定领域的专门知识，例如前述例子中的制造、财会或现金交易。这一专门知识实现于其编码之中。结构使企业机构捕获和组装该专门知识的共同特性，其作法是将专门知识实现在机构的编码中。首先，这使得开发者得以生成或扩展一个利用该专门知识的应用程序，因而问题一旦被解决，商业定律和设计就可以一直地实施和使用。另外，结构和在结构后面实现的知识对已经获取了统管和销售全过程的市场的知识(例如制造、财会)的那些组织有策略的有益含义，或为了包装，再销售，以及推广应用其知识，并进而进行技术开发及普及，生物技术需要有一个分配机制。

历史上，结构只是在最近才成为个人计算机的主导概念。这一进展得利于面向目标语言的出现，例如C++传统上，C++是最近通常在UNIX系统上以及研究人员的工作站上发现的，而不是在商业个人计算机上。正是象C++及其他面向目标的语言（例如Smalltalk和其他）使得许多大学和研究项目能产生今天商用结构和类别库的先驱。其中的例子包括斯坦福大学（Stanford University）的InterViews，卡内基—麦伦大学（Carnegie—Mellon University）的Andrew toolkit以及苏黎世大学（University of Zurich）的ET++结构。

有许多结构，其取决于所关心的系统级以及问题的性质。结构的类型范围包括从辅助开发用户界面的应用程序结构到提供通讯、打印，文档系统支持，以及图形之类的基本系统软件服务的低级结构。应用程序结构的商用例子是MacApp（Apple），Bedrock（Symantec），OWL（Borland），NeXTStep App Kit（NeXT）以及Smalltalk—80 MVC（ParcPlace）等等。

具有结构的编码需要惯于其他系统的开发者有新的思维方式。事实上，它与常规的“编程”不同。在以前的操作系统（例如DOS或UNIX）中，开发者自己的程序提供了所有构造。操作系统通过系统调用提供服务—开发者的程序在需要服务时进行调用并在服务提供后控制返回。程序构造基于控制流程，它以开发者编写的码所实现。

而当采用结构时，该过程成为逆向，开发者不再负责流程控制。开发者必须提前根据执行的流程来理解编程任务。

反之，必须根据目标的响应进行思考。这必须根据结构来决定何时执行任务。开发者编写的例行程序由不是开发者写的编码所驱动，而且开发者甚至从未见过这些编码。控制流程的转换对只是在过程编程有经验的开发者而言可能是较大的精神障碍。然而，一旦这一技术被理解，结构编程较之其他编程所需的工作将大为减少。

应用结构为开发者以同样方式提供预定功能性，系统结构(例如包括在最佳实施例中的那些)相应地有同样概念，通过提供系统级服务，储如系统程序这一类的开发者用子类别置换以生成定制的方案。例如，考虑一个可以提供基础以支持新的及多用途的设备(例如声频、视频、MIDI、动画电等等)的多媒体结构，需要支持新型设备的开发者必须写一个设备驱动程序，采用结构来完成此任务的话，开发者只需提供该新设备的具体特征以及行为标准。

在这种情况下，开发者将为多媒体结构调用的特定成员功能提供实施，开发者所获得的直接好处是用于每类设备的通用码已由多媒体结构提供了。这意味着开发者所要编定、测试及调试的设备驱动程序码要少许多，使用系统结构的另一例子是对小计算机系统接口(SCSI)设备，NuBus 卡以及图形设备有不同的 I/O 结构。由于具有继承的功能性，每个结构为设备类别中发现的通用功能性提供支持。其他开发者则可以依据这些对所有设备保持一致的界面。

一个最佳实施例采用了结构的概念并将其用于整个系统之中。对商用或公司开发者，系统集成商，或 OEM 而言，这

意味着诸如 MacApp 之类的结构所展示的优点不仅可影响文本和用户界面一类的应用层级，亦可影响例如图形，多媒体，文档系统，I/O，测试等服务的系统级。

在最佳实施例的结构中生成应用程序的过程很类似于依附于结构协议的特定谜块编写。这种方式中，编程的整个概念改变了，取代调用多个 API 层级的逐行编写，通过从环境中的预先存在的结构中衍生类别并加入新行为和/或置换继承的行为的方式而进行软件开发。

如此，开发者的应用程序成为由所有其他结构应用程序编写和共用的代码集。这是一个非常强有力的概念，因为开发者将能够根据相互的工作进行工作。这还为开发者提供了按照需要进行客户定制的灵活性。一些结构将按原样使用。在某些情况下，客户化的量将是最小，所以开发者插入的谜块最小。而在另些情况下，开发者可进行很大改动并生成全新的东西。在图 1 所示的最佳实施例中，位于 RAM 14 中并由 CPU 10 控制的程序负责管理利用面向目标的图形结构多种任务。

一个最佳实施例采用新体系结构来存储三维数据(这种三维数据用于定义例如部分表面图形目标)，由现实源扫描或数字化的数据，例如主体 CAT 扫描之类的图象再现，以及从其他计算(即液流量)近似而来的形状。在最佳实施例中，这些数据库类型称为：离散化数据”，因为要再现的形状是由离散点而非连续函数所确定的(注意，数据可根据某些连续操作进行初始定义。但是，应用程序用离散点，即液流量，参

数表面，或内含表面，来指定要再现的表面）。

TSampledSunface3D 目标通过提供允许客户管理数据的结构而提供给客户。该目标是一个 C++ 目标，正如上述说明书引言中所述。为了再现客户数据的图象，所需要的是客户生成碎片（shard 是指在三维图形中一个曲成三维表面的小三角形部分。在再现处理过程中，三维曲线表面被分为许多小的线性片它们被称为碎片，因为它们是由一些其他几何形定义的片组成的。碎片称为具有任选浓淡信息的三维三角形。“碎片”一词用于代之三角形一词，是为了叙述再现基单元，因为一个碎片的顶点包括除几何信息之外的其他信息（色彩，构造映射坐标，法线等）。采用与三角形相连的顶点作为确定要再现的形状的基本媒介为离散点定义的三维表面数据提供了一个简单的公分图。

由客户生成的离散数据不仅限于几何信息，还有法线，色彩，以及结构映射标志，可在一个碎片的每个顶点处提供。这使得应用程序对数据的外观具有详尽的控制，并将除目标的基本几何信息之外的其他变量结合到合成的再现图象中。用于在顶点处提供色彩信息的两个已知应用程序包括采用色彩显示应变能的有限元件模型以及记录目标的色彩及其形状的三维扫描仪。

为提高效率，碎片的拓扑结构与其顶点分离。当再现离散数据时，将每个顶点信息与碎片分开处理。这是由于几个碎片通常共用一个顶点。例如在图 2 中，在点 5 处的取样栅格由六个碎片共用。把碎片与顶点分开处理可使再现设备对

每个顶点只进行一次操作，例如把点转换成屏幕空间。

为了实施 TSampled Surface3D 子类别，客户必须提供一个迭代程序以便成批返回顶点。每个顶点在空间中定义一个三维点，但也可以包括其他信息，例如色彩，结构映射坐标等。子类别必须提供的下一例行程序生成迭代程序，它产生三个进入碎片的顶点的顶点陈列中心的参数。所以，就图 2 所示网格而言，第一个调用返回一个有 16 个顶点的陈列。迭代程序目标依次必须提供另一迭代程序，它生成若干组三参数，例如(0,5,1)，(0,4,5)，(1,6,2)，(1,5,6)等等，用于上述图中的 18 个碎片。注意顺序很重要，当在模型空间中从表面的外部看时，顶点应该是顺时钟方向的。

与该组参数一起，一组标帜被返回，它们指示碎片的边缘是否可见。这些标帜在碎片构成时被检查。例如在图 3 中，在 300 处的再现使所有碎片边缘成形。在 310 处的再现中，已经没有了对角边，而在 320 处，则只绘制了数据边缘。这些标帜也管理目标以避免重复构成同样的边。该子类别无需一次便生成所有顶点。这使得目标在再现处理过程中在生成顶点和碎片之间交替变化。这对在浮区生成点而且不想一次便存储整个目标的顶点的数据组是有用的。该再现程序一直请求成批顶点和碎片迭代程序，直到不再可用之时(即，Copy-VertexInfo 方法返回 FALSE，以指示不再有碎片可用)。

类别结构

以 C++ 实现的 TSampledSurface3D 目标看起来如下：

```
class TSampledSurface3D {
  virtual TShardChunkIterator* CreateShardChunkIterator() = 0;
  // See section below on streaming
  virtual StreamOutAsShards( TStream& towhere ) const;
}

class TShardChunkIterator {
  // TShadingInfo has point, normal, color, texture coords, etc
  // Returns TRUE if there is more data to fetch; copies data into the data parameter
   virtual Boolean CopyVertexInfo(TShadingInfoArray& data) = 0;

   virtual TShardIterator * GetShardIterator() const = 0;

   // Used for optimization, see below
   virtual SetCamera(const TCamera& cam, Boolean cullBackFaces) = 0;
}
```

图 4 是根据本发明的最佳实施例再现一个 SampledSurface 3D 目标的详细逻辑流程图。处理始于功能框 400，它提出了再现操作的革新。然后，为了再现一个离散表面，再现程序在功能框 402 中生成一个 TShardChunkIterator。在框 404 中提出一任选步骤，以确定从一个摄象目标定义的角度来看，目标的哪一部分是可见的。然后它调用迭代程序的 CopyVertexInfo()，如功能框 406 所示，以获取该表面的每个顶点信息的复制。在功能框 410 处进行测试以确定是否剩下任何顶点。若否，则处理完成于 412 所示。若还有顶点要处理，则在功能框 414 处由再现程序将特定三维景象的数据初始化（即将点转换成屏幕空间），然后在实际绘制目标时调用 TShardIterator。该处理以“块”的形式继续，如功能框 420 所示，每一块始于调用 CopyVertexInfo，并结束于 CopyVertexInfo 返回 FALSE，其指示所有数据已从目标中检出，如判别框 430 所检测的。不必让再现程序共用每块之间的顶点，因为在两个块之间衔接处的顶点数量相对于每块中的顶点数（0($\sqrt{n}$)）而言很小。再现处理的细节在图 5 的实例中进行叙述。

图 5 示出了被分成四块的表面 500。TShardChunkIterator 用于再现表面 510，而 TShardIterator 再现表面 520。TShadingInfoArray 是一个目标阵列，它包括在 TShadingInfo 中的信息（点，法线，色彩，结构坐标，切线等）。TShardIterator 与再现程序所用的 TSampledSurface3D.（如图 4 中的功能框 414 所述）有关。TShardIterator 的类别结构如下所示：

```
class TShardIterator {
 public:
        enum { kFrontFace, kBackFace, kUnknown } EFrontFaceFlag;
        enum { kClipped, kUnclipped, kUnknown } EClipFlag;

        virtual Boolean GetNextShard( long& i0, long& i1, long& i2,
                                      Boolean& edge01vis,
                                      Boolean& edge12vis,
                                      Boolean& edge20vis);

// Second version with optional flags for render optimization, see below
        virtual Boolean GetNextShard( long& i0, long& i1, long& i2,
                                      Boolean& edge01vis,
                                      Boolean& edge12vis,
                                      Boolean& edge20vis,
                                      EFrontFaceFlag frontFlag,
                                      EClipFlag clipFlag );
};
```

TSampledSurface 目标再现码如下所示：

```
TGrafDevice::RenderSampledSurface3D( TSampledSurface3D& dd3d )
{

  TShadingInfoArray vertices;
  TShardChunkIterator * dd3Diter = dd3d.CreateShardChunkIterator();

 // The outer loop processes "chunks" of data from the SS3D
  while (dd3Diter->CopyVertexInfo( vertices ) )
  {
      // Take the generic TShadingInfo data, and add device level information
      TDeviceVertices devVerts = PrepareData( vertices );

      TShardIterator * shardIter = dd3Diter->GetShardIterator();

      while (shardIter->GetNextShard( i0, i1, i2, f0, f1, f2 ))
      {
        RenderShard( devVerts[i0], devVerts[i1], devVerts[i2] );
      }
      delete shardIter;
  }
  delete dd3Diter;
}
```

值i0,i1 和 i2 是整数，它们被编入由 CopyVertexInfo()返回的阵列中。注意 TSampledSurface3D 目标是可成流的(streamable)。可成流涉及采用使目标在以后以可被重新构成的方式把目标的内容编写成数据流或文档。若该目标在同样机器中正在成流，则可采用基本子类别，因为抽取再现数据的库代码是可用的。若一个 SampledSurface3D 目标要假脱机或通过一个网络流向一个没有定义该子类别的库的其他机器，则可采用 TSampledSurface3D 的 StreamOutAsShards 方法。实施这一过程的例子示于下面：

```
TSampledSurface3D::StreamOutAsShards(TStream& towhere )
{
    TShadingInfoArray vertices;
    TShardChunkIterator * ss3Diter = CreateShardChunkIterator();

    while (ss3Diter.CopyVertexInfo( vertices ))
    {
```

```
        TRUE >>= towhere;
// Flag indicating more data available
        vertices >>= towhere;
// Stream out the vertices

TShardIterator * shardIter = ss3Diter.GetShardIterator();

        while (shardIter->GetNextShard( i0, i1, i2, f0, f1, f2 ))
        {
                i0 >>= towhere;
                i1 >>= towhere;
// Stream out indices into the block of vertices
                i2 >>= towhere;
                f0 >>= towhere;
                f1 >>= towhere;
                f2 >>= towhere;
        }
        -1L >>= towhere;
// Sentinel indicating no more shards.
        delete shardIter;
    }
    FALSE >>= towhere;
// Flag no more data available.
    delete ss3Diter;
}
```

一个最佳实施例任选两个最佳方案来大力改善性能，如图 4 中的功能框 440 和 450 所示。第一个方案是忽略表面的后面部分(若后面是处于打开状态的话)，如功能框 440 所示；第二个方案则是忽略超出荧光屏幕的表面部分，如功能框 450 所示。TSampledSurface3D 通过为再现程序提供其他两个方法来利用这些优化方案。这两个方法是 SetCamera 和返回限制标帜的 TShardIterator 的版本。SetCamera 采用了一个参数，它指向指定要显示目标的三维视图的摄象目标。再现程序在启动 CopyVertexInfo 之前调用 SetCamera。当再现碎片时(如功能框 460 所示)，再现程序检查每个碎片的返回限制标帜以确定是否有必要进行限制或后面挑选测试。对每个碎片，有两个限制标帜值返回。这些标帜指示：

- 该碎片已知不被限制(再现程序可以越过限制测试)
- 该碎片可以或不可以被限制(再现程序必须测试它)
- 该碎片是后面(若后面挑选操作被切断)
- 该碎片是前面
- 该碎片的取向不知且必须被测试

为说明这些情况，考虑实现一个三维高度场的子类别。THeightField3D 该高度场实际上是一个平面矩形，其在每个格栅点处从表示高度的平面的 Z 方向的扰动很小。在 SetCamera 调用中 THeightField3D 目标首先检查高度场的基平面。若该平面是后面(由 Camera 的 IsBackfacing 调用确定)，且该后面(backface)是挑选的，则目标知道无需关心生成任何数据(即 CopyVertexInfo 立即返回 FALSE)。类似地，

THeightField3D 可调用在高度场平面四个角上的摄象机的 GetClipCode 方法。若该返回的限制码指示总高度场超出屏幕，则可跳过再现。类似地，若限制和后面测试指示整个目标是前面且/或无限制的，则再现程序可以跳过对单独碎片的这些测试。

这些优化亦可在再现的离散数据组上完成。例如，THeightField3D 可在九个块中再现数据(即在返回 FALSE 前有九个对 CopyVertexInfo 的调用)。这些块中的每一个可以进行后面和限制测试。因此，若一个角要被完全限制，只要跳过它即可。

来自三维数字化仪的示例数据。

假设来自一个三维测距仪型的数字化仪的数据作为一个 1000×1000 高度值存储起来。这些数据点将被用作再现表面的 Z 坐标。图 6 示出了一个测距仪系统的示例图。测距仪 600 的测距范围是已知的，其被分为 1000×1000 的相等间隔。610 处示出了被扫描的目标，测距仪摄像机示于 620 处，而存储再现数据的计算机示于 630。计算机 630 对应于图 1 的计算机。图 7 示出了从摄像机接收的测距仪数据。从测距仪而来的数据只有 Z 坐标 720，X 坐标 700 和 Y 坐标 710 作为系统蕴含于 1000×1000 的数据点陈列。

由于 1,000,000 是很大的数据点量，所以放在 10 个较小单元（称为块)中实现这些数据。使用上述展示为最佳实施例的再现系统来再现这一表面要求实施下述例行程序。第

一个便是 TShardChunkIterator::CopyVertexInfo()。这一程序将从扫描仪获取头 10,000 个数据,并提供 X 和 Y 坐标。用于处理由摄象机接收的数据的代码展示如下:

```
Boolean
TShardChunkIterator::CopyVertexInfo( TShadingInfoArray& data )
{
        // fRawData is a 1,000,000 element array with the 1Kx1K
        // collection of data from the scanner.  fChunkCount keeps track of
        // how many times CopyVertexInfo has been called.

        if (fChunkCount == 10)
                return FALSE;                   // All data has been copied out

        for (i = 0; i < 100; i++)               // 100 rows in this chunk
           for (j = 0; j < 1000; j++) // 1000 items per row
           {
                data[i * 1000 + j] = kDefault;          // Set default color, etc.

                TGPoint3D * p = &data[i * 1000 + j].fPoint;

                p->fX = j;                      // Generate implied X and Y
                p->fY = fChunkCount * 1000 + i

                p->fZ = fRawData[(fChunkCount * 1000 + i)*1000 + j];
           }
        fChunkCount++;
        return TRUE;
}
```

注意根据顶点的原始数据，再现系统将为每个顶点计算一次法线和浓淡值(从而免除了为每个再现的三角形进行冗余计算的负担)。图 8 示出了根据本发明一个最佳实施例来自定距格栅的碎片的布局。在 800 处，T＝三角形数，在 810 处，R＝每一行取样的数目。

下一步是生成一个碎片迭代程序，以生成用于从扫描数据再现表面的三角形，如图 4 的功能框 414 所示。这一处理生成编入到产生 CopyVertexInfo 的顶点信息中的参数；即只有如何把三角形挂钩的拓扑是需要的。这是通过图 8 的 800 和 810 所示的标号方案实现的。实施 GetNextShard 的方法如下：

```
TShardIterator::GetNextShard(  long& i0, long& i1, long& i2,
                               Boolean& edge01vis,
                               Boolean& edge12vis,
                               Boolean& edge20vis )
{
    // fShardNumber is the number of the shard we've generated.

    if (fShardNumber == 1000 * 100)
        return FALSE;              // All shard for this chunk generated

    if (even( fShardNumber ))
    {
        i0 = fShardNumber / 2;
        i1 = fShardNumber / 2 + 1000;
        i2 = fShardNumber / 2 + 1;
        // Show edge visibility, diagonal edges are not visible.
        edge01vis = TRUE; edge12vis = FALSE; edge20vis = TRUE;
    }
    else // odd # shard
    {
        i0 = fShardNumber / 2 + 1;
        i1 = fShardNumber / 2 + 1000;
        i2 = fShardNumber / 2 + 1000 +1;
        // Show edge visibility, diagonal edges are not visible.
        edge01vis = FALSE; edge12vis = TRUE; edge20vis = TRUE;
    }
    fShardNumber++;
    return TRUE;
}
```

来自有限之网格再现的示例数据

三维模型软件生成一个有限之模型(FEM)。通常是设计一个部分来满足特定的形状要求，然后再进行“衔接(mesh)”，或者由FEM软件将其分成碎片。在碎片的顶点处，FEM软件计算顶点的应变能值。应变能估算通常被转换成色彩梯度以便给出该部分张力的图形显示。图9示出了由FEM方法分析的支撑物。图10示出了由FEM软件生成的网格模型。除了X'Y'Z几何位置信息之外，每个编号的顶点都有一个代表该点应变能的值。由FEM软件生成的模型可作为顶点表存起来，亦可作为由顶点即(0,5,10)(1,5,6)(5,10,11),(1,6,2)等形成的三角形碎片表存起来，其中，括号里的数涉及顶点的参数。

为了用上述的SampledSurface3D结构再现这一模型，将CopyVertexInfo例行程序定义如下(假设FEM数据已装在其他地方)：

```
Boolean
TShardChunkIterator::CopyVertexInfo( TShadingInfoArray& data )
{
        for (i = 0; i < NumberFEMVertices; i++)
        {
                data[i].fPoint.fX = FEMdataXCoordinate[i];
                data[i].fPoint.fY = FEMdataYCoordinate[i];
                data[i].fPoint.fZ = FEMdataZCoordinate[i];
                // Set the color according to the strain energy at the
                // vertex.  Energy to Color performs some user-defined
                // mapping, perhaps via a lookup table.

                data.fBaseColor[i] = EnergyToColor( FEMdataStrain[i] );
        }

        // This assumes all data is processed in one step.  For large data sets,
        // this can be broken into multiple calls, like example #1.
        return FALSE;
}
```

ShardIterator 也很简单，因为它只返回由 FEM 程序生成的数据。

```
TShardIterator::GetNextShard(  long& i0, long& i1, long& i2,
                               Boolean& edge01vis,
                               Boolean& edge12vis,
                               Boolean& edge20vis )
{
    if (fNumberOfShards > FEMdataNumberOfShards)
        return FALSE;       // All shards rendered
    i0 = FEMdataV0index[ fNumberOfShards ];
    i1 = FEMdataV1index[ fNumberOfShards ];
    i2 = FEMdataV2index[ fNumberOfShards ];
    // Edge visibility may or may not be useful
    edge01vis = FALSE; edge12vis = FALSE; edge20vis = FALSE;
    fNumberOfShards++;
    return TRUE;
}
```

随着碎片再现，再现程序用应变能生成的色彩来产生该部分张力分布的平滑浓淡(shade是指在三维图形中改变通过目标的色彩和亮度以模拟其真实形状)图，如图11所示。应变能不是可以该方式显示的唯一变量，象温度分布等亦可以类似方式显示。该技术通常用于“科学显象”应用所生成的数据库。

再现系统的类似应用亦包括部分表面生成，缠含表面取样，以及参数表面再现。

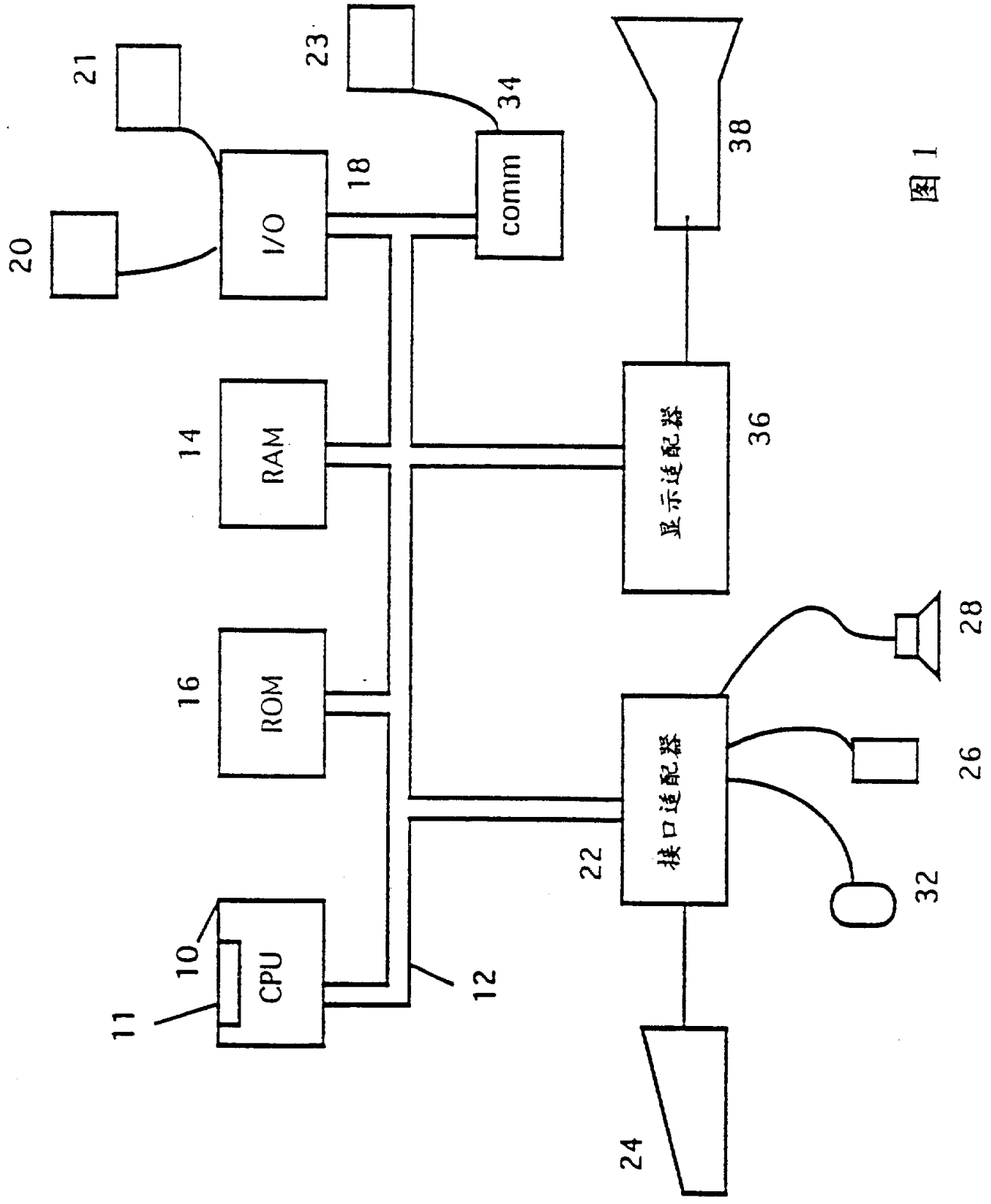

CPU
ROM
RAM
I/O
comm
显示适配器
接口适配器
10
11
12
14
16
18
20
21
22
23
24
26
28
32
34
36
38

图 1

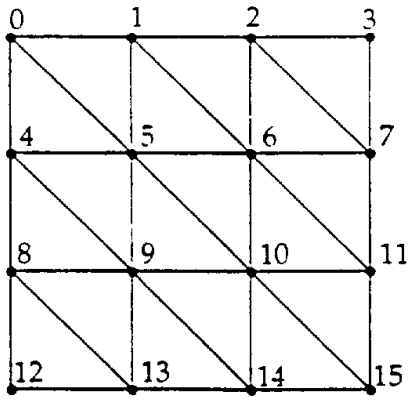
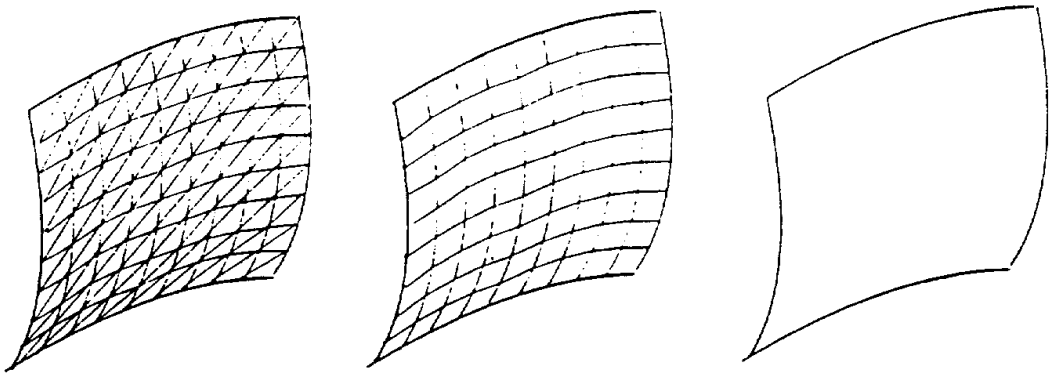


图 2



300 310 320

图 3

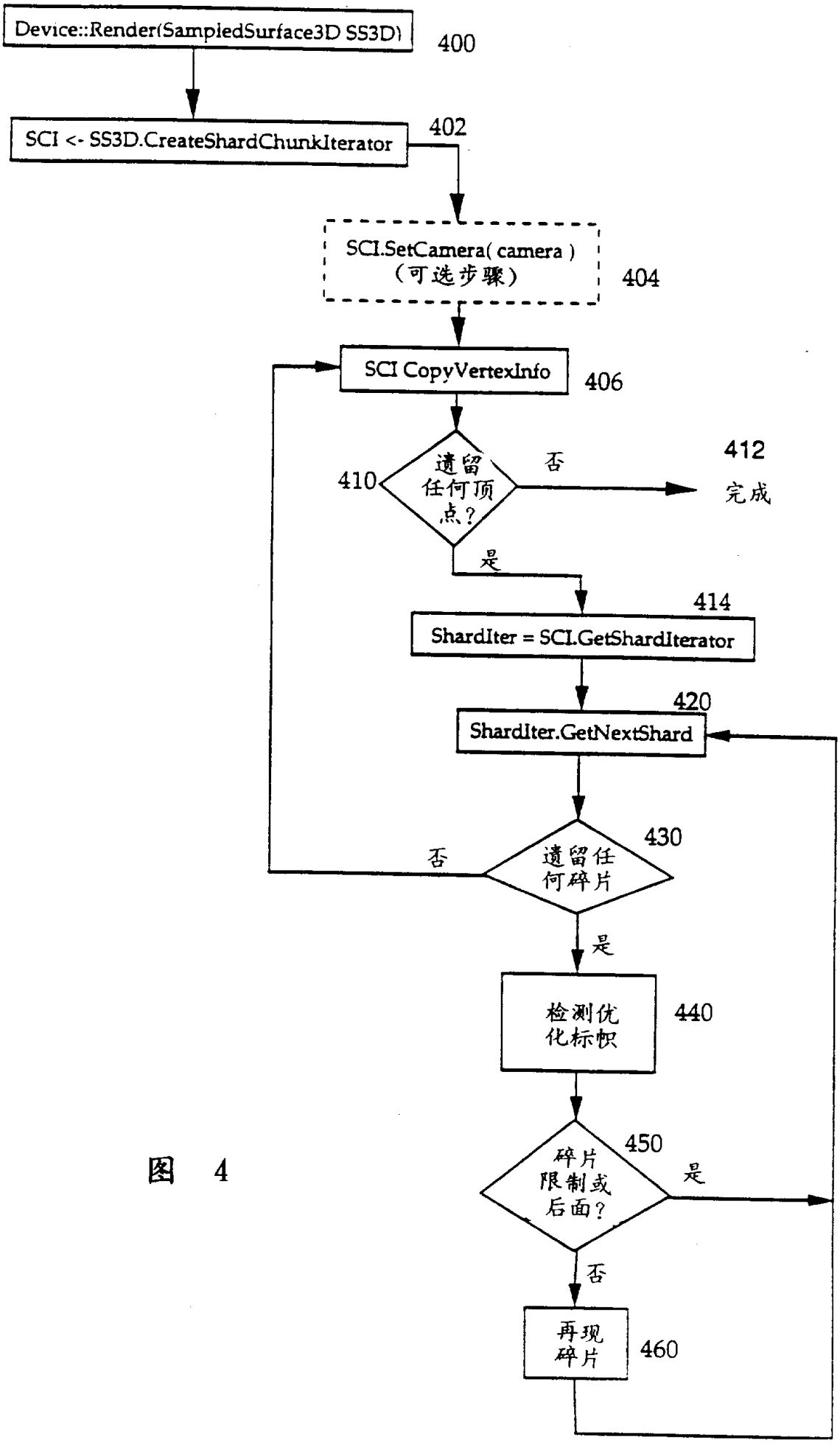

Device::Render(SampledSurface3D SS3D)
400
SCI <- SS3D.CreateShardChunkIterator
402
SCI.SetCamera(camera)
（可选步骤）
404
SCI CopyVertexInfo
406
遗留任何顶点？
410
否
412
完成
是
414
ShardIter = SCI.GetShardIterator
420
ShardIter.GetNextShard
遗留任何碎片
430
否
是
检测优化标帜
440
碎片限制或后面？
450
是
否
再现碎片
460

图 4

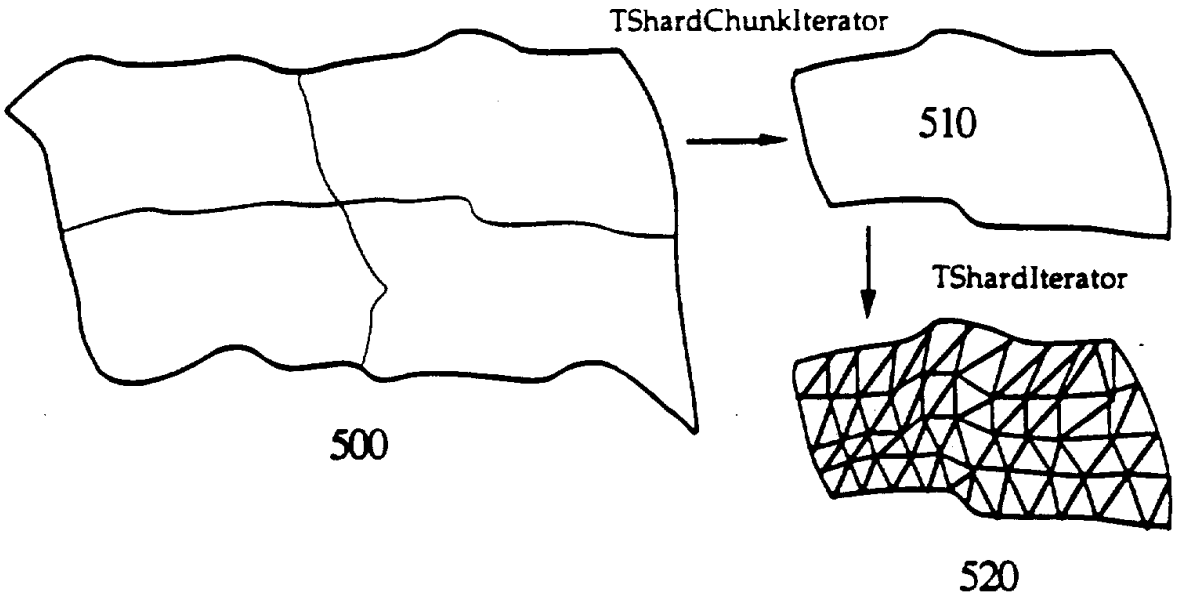

TShardChunkIterator
510
TShardIterator
500
520

图 5

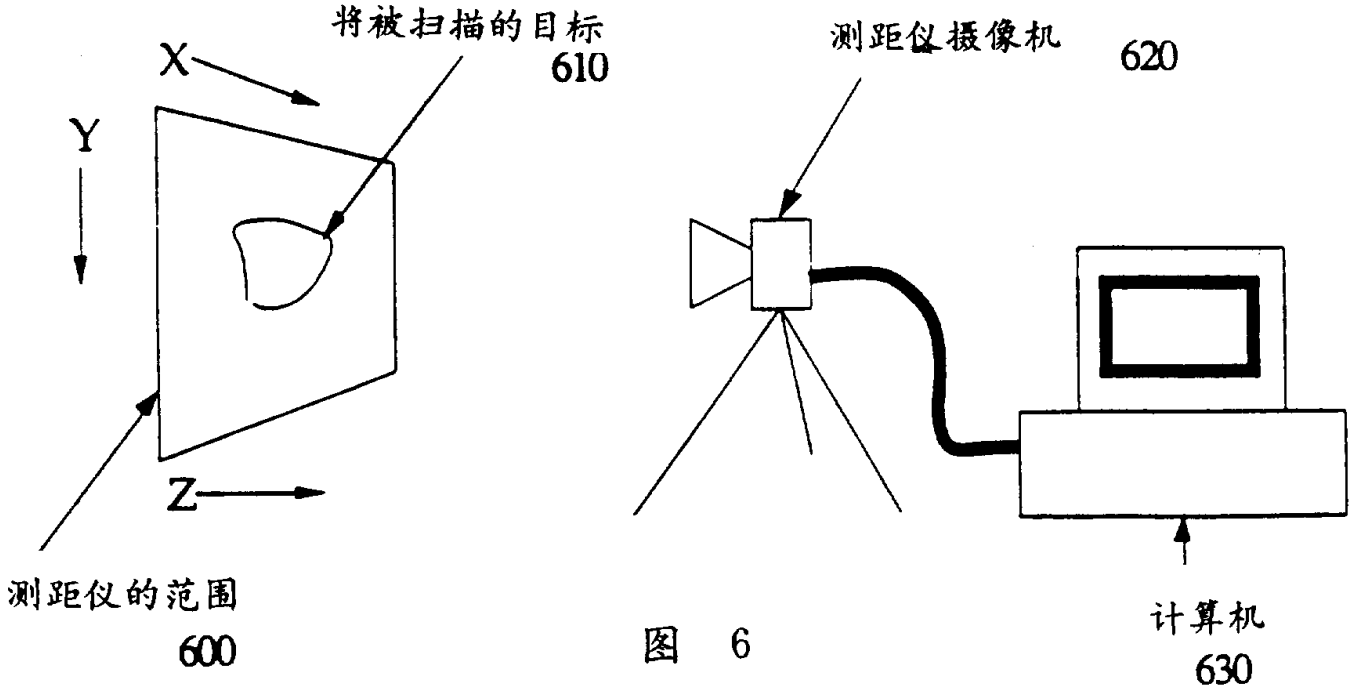

将被扫描的目标
610
X
Y
Z
测距仪的范围
600
测距仪摄像机
620
计算机
630

图 6

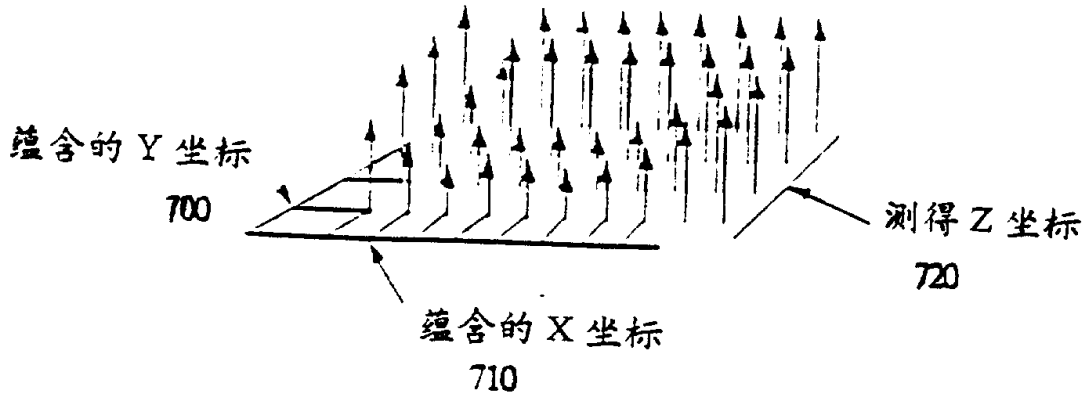


图 7

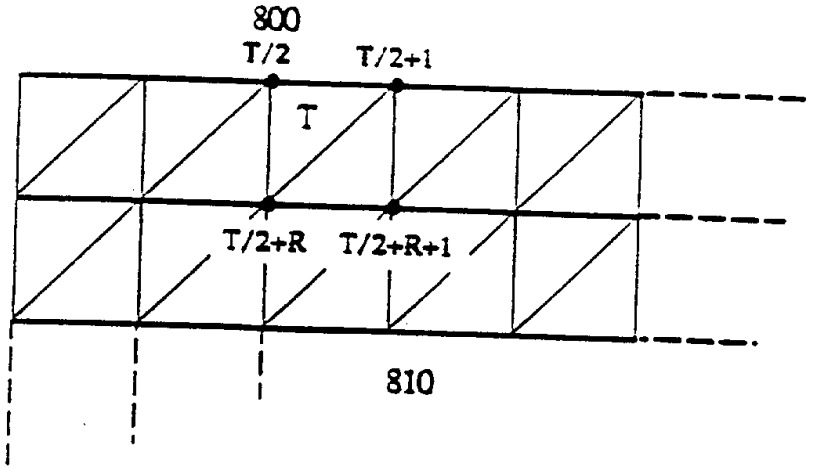


图 8

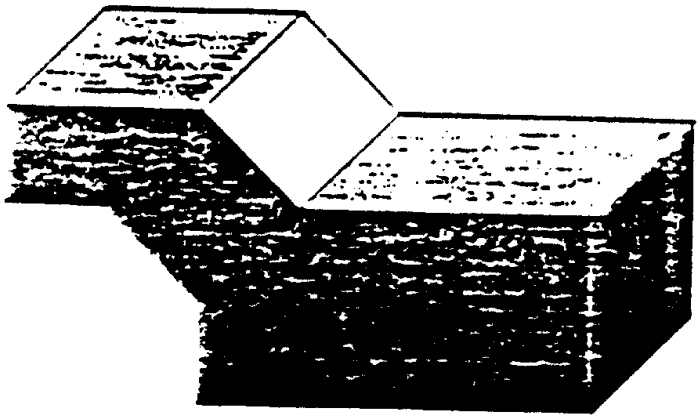


图 9

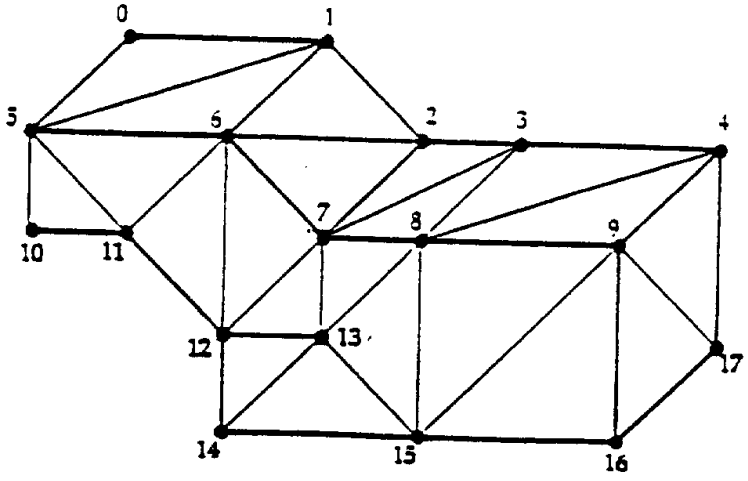


图 10



图 11