# 一种基于三维纹理硬件加速的海量数据体绘制方法

申请号：200710065336.6
申请日：2007-04-11

申请(专利权)人　中国科学院自动化研究所
地址　100080北京市海淀区中关村东路95号
发明(设计)人　田捷　薛健

主分类号　G06T1/60(2006.01)I
分类号　G06T1/60(2006.01)I　G06T15/00(2006.01)I
公开(公告)号　101286225
公开(公告)日　2008-10-15
专利代理机构　中科专利商标代理有限责任公司
代理人　周国城

(19)中华人民共和国国家知识产权局

(12)发明专利

(10)授权公告号 CN 101286225 B
(45)授权公告日 2010.10.06

(21)申请号 200710065336.6

(22)申请日 2007.04.11

(73)专利权人 中国科学院自动化研究所
地址 100080 北京市海淀区中关村东路 95 号

(72)发明人 田捷 薛健

(74)专利代理机构 中科专利商标代理有限责任公司 11021
代理人 周国城

(51)Int.Cl.
*G06T 1/60*(2006.01)
*G06T 15/00*(2006.01)

(56)对比文件
WO 2005/104042 A1,2005.11.03, 全文.
US 2002/0130865 A1,2002.09.19, 全文.
CN 1932883 A,2007.03.21, 全文.
EP 1376472 A1, 全文.
CN 1277698 A,2000.12.20, 全文.
CN 1763786 A,2006.04.26, 全文.

审查员 董立波

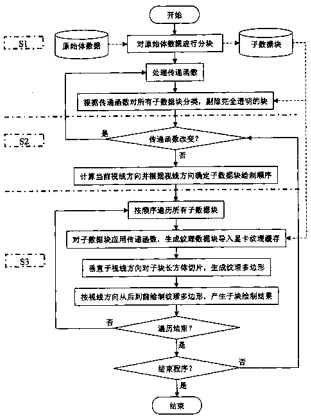
权利要求书 1 页 说明书 6 页 附图 4 页

(54)发明名称

一种基于三维纹理硬件加速的海量数据体绘制方法

(57)摘要

公开一种基于三维纹理硬件加速的海量数据体绘制方法,其中海量数据是指:数据量超过计算机物理内存所能容纳的极限,无法在处理时完全转载入物理内存。主要步骤包括:数据预处理,传递函数处理和原始数据分块;计算当前视线方向并根据视线方向确定子数据块绘制顺序;用三维纹理的方法对每个子数据块数据分别绘制。本发明通过对海量原始数据进行合理分块,使每块数据能够被装入内存单独处理,从而能够对超出内存容量的海量数据进行体绘制。同时,通过使用三维纹理硬件等措施优化分块绘制流程,大大加速整个绘制过程,实现了海量数据的快速可视化,在大规模医学影像数据处理及可视化领域有着重要的应用价值。




CN 101286225 B

1. 一种基于三维纹理硬件加速的海量数据体绘制方法，其特征在于：

步骤 S1：对原始体数据合理的分块处理，将海量原始体数据分解成可装入内存及显卡显存的独立子数据块，并将独立子数据块存入磁盘备用，同时对体绘制传递函数进行处理，生成快速查找表；将传递函数的值域映射到 RGBA，每个分量的值离散化为 [0,255] 之内的整数，根据原始数据灰度范围生成一个列表，将体素灰度值作为索引得到的列表元素值即为该体素经过传递函数作用后的最终绘制所用颜色值；并根据传递函数对子数据块进行分类，找出完全透明的子数据块，从待绘制的子数据块中剔除；

步骤 S2：检查是否更改原始传递函数，如果是，则返回步骤 S1 重新处理传递函数以及对子数据块进行分类；如果否，则计算当前视线方向，并按照总是从在视线方向上离视点最远的顶点处的子数据块开始，从后向前绘制；其中，只要计算八个顶点在视线上的投影，找出其中最大的即可确定子数据块的绘制顺序，将按该顺序排列的待绘制子数据块索引序列以及从步骤 S1 得到的处理后的传递函数传给步骤 S3；

步骤 S3：遍历待绘制子数据块索引序列，按索引从磁盘取得步骤 S1 得到的子数据块，并利用纹理图形硬件加速每一子数据分块的绘制，首先应用步骤 1 生成的传递函数查找表快速生成最终绘制所用的三维纹理块，并将此三维纹理块导入到显卡的纹理缓存中；然后，用垂直于视线的平面按照视线方向从后向前切割三维纹理块长方体，生成纹理多边形；其中，在实际生成纹理多边形之前，先通过建立一个快速的查找表，给出所述纹理长方体八个顶点的状态和要生成的纹理多边形之间的映射关系，之后，在实际生成纹理多边形时，用查表的方法来加速生成纹理多边形的过程；所述纹理块长方体八个顶点的状态指：当平面与长方体相交时，其各顶点共有两种状态：在平面正向空间内及平面上，即，代入平面方程求得的值大于等于 0；或者在平面负向空间内，即，代入平面方程求得的值小于 0；最后，仍然按从后向前的顺序绘制所有生成的纹理多边形，得到子数据块的绘制结果，遍历完成即代表一遍绘制结束，实现对海量数据的快速体绘制，如果需要重复绘制，程序不结束，则转步骤 S2 检查是否更改传递函数并继续上面的流程，否则整个程序结束。

# 一种基于三维纹理硬件加速的海量数据体绘制方法

## 技术领域

[0001] 本发明涉及计算机图形学中三维可视化领域，特别涉及海量数据的直接体绘制方法和基于三维纹理硬件的加速方法。

## 现有技术

[0002] 在大数据量的应用程序中，速度较快的内部存储器与速度较慢的外部存储器之间的数据交换通常是主要的效率瓶颈所在。专门为消除这种瓶颈而设计的算法被称为外存算法 (external-memory algorithms)。对外存算法的研究很早就开始了，最初是用于解决小内存计算机无法同时装载大量数据的问题，同时通过仔细设计内存和外存之间数据交换的策略，可以克服直接采用操作系统提供的虚拟内存管理功能所带来的低效问题。这是一类典型的以空间换时间的算法，通常通过增加计算机内存容量可以避免采用此类算法并且得到更高的运行效率。

[0003] 随着各种影像获取设备硬件技术的不断进步，所获取影像的空间分辨率越来越高，随之而来的是影像数据量急速上升，给已有的图像处理及可视化算法带来了严峻的挑战。特别是自从美国的可视人体 (Visible Human) 项目实施以来，虚拟人体采集的图像数据量已经达到 43GB，而国内研究机构于 2002 至 2003 年完成的中国男性和女性数字化可视人体的数据量已分别达到 90.65GB 和 131.04GB。如此海量的数据使得三维实时处理和显示变得更为困难。需要特别注意的是这里所说的“海量数据”是专指这样一类数据 :其原始数据量已经超过了计算机内存所能容纳的数据量 ( 通常也超过了可寻址空间的大小 )，因而不可能将其完全装入内存再进行处理。大多数文献将这样一类处理“海量数据”的外存算法称为 Out-of-Core 算法。

[0004] 体绘制算法是可视化算法中最为重要的一种，以能够产生高质量且逼真的绘制结果而闻名。对体绘制的研究出现于上个世纪 80 年代末，传统的体绘制算法可以分为三个大类 :图像空间 (Image Space) 的绘制算法、物体空间 (Object Space) 的绘制算法以及图像和物体空间混合 (Hybrid) 绘制算法。其中，常见的经典体绘制算法有体数据光线投射法 (Volume Ray Casting)、Splatting 算法和 Shear Warp 算法等。这些经典算法大多需要对原始数据进行随机访问，这就意味着需要将全部数据装入内存再进行计算，否则将带来严重的效率问题。此外，即使在小数据量的环境下，这些算法也无法达到实时的要求。

[0005] 针对海量数据，Farias 和 Silva 在 2001 年中提出了一种在有限内存的情况下对海量数据进行直接体绘制的方法。该方法遍历整个数据集的所有体元，基于穿过该体元的一段光线进行计算，为该体元在投影平面上所覆盖的每个像素保存两个片段 (fragment)( 光线与体元相交两次 )。第一次遍历完成后对所有产生的片段做外排序，排序以片段所在像素 ( 位置 ) 为第一关键字，以片段的深度为第二关键字，在排序产生的结果上就可以进行有效的光线投射计算。该方法虽然可以处理海量数据，但是由于使用外排序而严重影响了绘制速度。

[0006] 此外，随着图形硬件的快速发展，基于图形硬件加速的体绘制算法正逐渐成为主

流。1994 年，Brian Cabral 等人提出使用纹理映射来加速体绘制的速度，但由于此算法只能运行在昂贵的图形工作站的显卡上，因此直至 1998 年，研究人员一直没有认识到这个算法的重要性。随着三维游戏等娱乐市场的不断需求，普通 PC 机上装配的显卡的三维加速能力越来越强，同时纹理映射所需要的二维光栅处理能力也越来越强，使用图形硬件来作体绘制也变得更为可行。Westermann 等人在 1998 年结合当前的显卡所提供的功能，实现了一个比较实用的基于硬件的体绘制算法，并且于后来的工作中被完善，加入了基于硬件加速的分类和光照计算。另外在 2000 年 C. Rezk-Salama 等人总结了前人所作过的工作，并且使用显卡中刚提供的多纹理功能，在普通的 PC 机上实现了实时的体绘制。在此后几年中仍不断有新的基于图形硬件加速的体绘制方法提出来，但到目前为止，大多硬件加速的体绘制算法仍然只针对小数据量 (in-core) 的体数据，还未用来加速海量数据的体绘制。

## 发明内容

[0007] 现有技术体绘制算法只针对小数据量的体数据进行处理，不能对超出内存容量的海量数据进行体绘制，本发明的目的是提供一种利用三维纹理图形硬件来加速计算的海量数据体绘制方法，用于对数据量超过计算机物理内存容量极限的数据集进行快速体绘制。

[0008] 为了实现所述的目的，本发明基于三维纹理硬件加速的海量数据体绘制方法，包括如下步骤：

[0009] 步骤 S1：对原始体数据合理的分块处理，将无法为物理内存所容纳的海量原始体数据分解成可装入内存及显卡显存的独立子数据块，并将独立子数据块存入磁盘备用，从而能够处理传统基于纹理加速的体绘制算法所无法处理的海量体数据；同时处理传递函数并根据其对子数据块进行分类，将处理结果传给步骤 S2；

[0010] 步骤 S2：检查是否更改原始传递函数，如果是，则返回步骤 S1 重新处理传递函数以及对子数据块进行分类；如果否，则计算当前视线方向，并根据视线方向确定子数据块绘制顺序，将按顺序排列的待绘制子数据块索引序列以及从步骤 S1 得到的处理后的传递函数传给步骤 S3；

[0011] 步骤 S3：遍历步待绘制子数据块索引序列，按索引从磁盘取得步骤 S1 得到的子数据块，并利用纹理图形硬件加速每一子数据分块的绘制，遍历完成即代表一遍绘制结束，实现对海量数据的快速体绘制，如果需要重复绘制，程序不结束，则转步骤 S2 检查是否更改传递函数并继续上面的流程，否则整个程序结束。

[0012] 优选地，数据预处理步骤 S1 还包括：

[0013] 步骤 S11：将存储在磁盘上的原始体数据分割成大小相同的子数据块，使得每块数据能够被完全载入内存以及显卡的显存，并将每个子数据块以独立文件的形式存储在磁盘上以备下面的步骤使用；

[0014] 步骤 S12：处理体绘制传递函数，将其转换为快速查找表，以加速传递函数映射的过程；

[0015] 步骤 S13：根据传递函数对所有子数据块进行分类，剔除完全透明的块，获得需要绘制的独立子数据块集合，以索引的形式存放在内存中。

[0016] 优选地，分块绘制步骤 S3 还包括：

[0017] 步骤 S31：按绘制顺序依次从磁盘取得待绘制的子数据块，对每个子数据块应用

传递函数，生成三维纹理块加载到显卡显存；

[0018] 步骤 S32：根据视线方向对三维纹理块进行切片，其中，通过建立一个快速查找表，给出纹理块长方体八个顶点状态和生成多边形之间的映射关系，用查表的方法来加速生成纹理多边形的过程；

[0019] 步骤 S33：按视线方向从后向前绘制生成的纹理多边形得到绘制结果。

[0020] 本发明有益效果是通过对海量原始数据进行合理分块，使每个子数据块能够被装入内存单独处理，从而能够对超出内存容量的海量数据进行体绘制。同时，通过使用三维纹理硬件等措施优化分块绘制流程，大大加速整个绘制过程，实现了海量数据的快速可视化，在大规模医学影像数据处理及可视化领域有着重要的应用价值。

## 附图说明

[0021] 图 1 是基于三维纹理硬件加速的海量数据体绘制方法的流程图；

[0022] 图 2 是体数据在模型空间分布示意图；

[0023] 图 3 是生成的纹理多边形示意图；

[0024] 图 4 是一个纹理多边形的例子；

[0025] 图 5 是本发明应用于医学影像数据集的绘制结果。

## 具体实施方式

[0026] 下面将结合附图对本发明加以详细说明，应指出的是，所描述的实施例仅旨在便于对本发明的理解，而对其不起任何限定作用。

[0027] 本发明的基本思想是通过合理的分块将无法为物理内存所容纳的海量体数据分解成可装入内存的独立子数据块，同时利用纹理图形硬件加速每一子数据块的绘制，从而实现对海量数据的快速体绘制，其中“海量数据”特指数据量超过计算机物理内存所能容纳的极限的数据集，无法在处理时完全转载入物理内存。

[0028] 下面结合附图详细描述本发明的方法。本发明的一种具体实现方案如图 1 基于三维纹理硬件加速的海量数据体绘制方法的流程图所示，其中，圆柱体表示外存储器（一般为硬盘），实线箭头表示算法流程，虚线箭头表示数据流动的方向；

[0029] 主要包括三个步骤：数据预处理、视线方向的计算和子数据块绘制顺序的确定、分块绘制。下面逐一进行介绍。

[0030] 步骤 1：数据预处理

[0031] 数据预处理包括三个主要部分。

[0032] 首先是对原始数据进行分块。为便于后面的绘制，将所有子数据块的大小选择一致，但子数据块尺寸作为算法的参数可以调节。每个子数据块以单个文件的形式存放在磁盘上，文件名放入一个列表中，以便随时取用。

[0033] 其次是对体绘制传递函数进行处理，生成快速查找表。由于后面绘制所用的纹理块将数据类型统一成 RGBA×8bit 的格式，所以将传递函数的值域映射到 RGBA，每个分量的值离散化为 [0,255] 之内的整数，根据原始数据灰度范围生成一个列表（通常为数组），将体素灰度值作为索引（数组下标）得到的列表元素值即为该体素经过传递函数作用后的最终绘制所用颜色值。这样，后面在绘制子数据块时就能够快速生成最终用于绘制的纹理块。

[0034] 由于原始数据中有相当大的一部分是背景，这部分数据经过体绘制传递函数的处理一般变为透明，不绘制的话也不会影响到最终结果。而分块后将产生一些完全透明的子数据块，这些子数据块无需绘制，可以从待绘制的数据块中剔除。

[0035] 因此，骤 1 的最后是根据传递函数对子数据块进行分类，找出完全透明的子数据块，从待绘制的子数据块中剔除，在实际的实现中是打上空块的标记，以便在绘制时略过。

[0036] 在步骤 1 中，需要特别注意的是子数据块尺寸的选择，其上限是显卡所能接受的最大三维纹理尺寸。如果尺寸选的过小，则会在磁盘上产生大量小文件，导致打开、关闭文件的次数过多而影响最终的绘制速度；如果尺寸选的过大，可以忽略的完全透明的子数据块数目就会减少，最终也会影响绘制速度。因此，子数据块的尺寸需谨慎选择，一般容量在 1GB 以上的体数据，子数据块应大于 64 体素 ×64 体素 ×64 体素，并以尽可能多的产生全透明子数据块为宜。

[0037] 步骤 2：视线方向的计算和子数据块绘制顺序的确定

[0038] 视线方向在屏幕空间始终是垂直于屏幕从外向内，即屏幕空间向量 (0.0,0.0,1.0)。设模型变换矩阵为 M，视图变换矩阵为 V，投影变换矩阵为 P；在屏幕空间取起始点和终止点分别为 (0.0,0.0,-1.0) 和 (0.0,0.0,1.0)，设其在模型空间中对应视线起始点为 s，其坐标计为 $(x_s, y_s, z_s, w_s)^T$ 终止点为 e，其坐标计为 $(x_e, y_e, z_e, w_e)^T$，这里均采用齐次坐标，则：

[0039]
$$\mathbf{s}=\begin{bmatrix} x_s \\ y_s \\ z_s \\ w_s \end{bmatrix}=\mathbf{M}^{-1}\mathbf{V}^{-1}\mathbf{P}^{-1}\begin{bmatrix} 0.0 \\ 0.0 \\ -1.0 \\ 1.0 \end{bmatrix},\quad \mathbf{e}=\begin{bmatrix} x_e \\ y_e \\ z_e \\ w_e \end{bmatrix}=\mathbf{M}^{-1}\mathbf{V}^{-1}\mathbf{P}^{-1}\begin{bmatrix} 0.0 \\ 0.0 \\ 1.0 \\ 1.0 \end{bmatrix} \qquad (1)$$

[0040] 当使用透视投影时，还需对视线起始点 s 和终止点 e 做进一步处理：

[0041]
$$\mathbf{s}=\frac{\mathbf{s}}{w_s}, \qquad (2)$$

[0042]
$$\mathbf{e}=\frac{\mathbf{e}}{w_e} \qquad (3)$$

[0043] 则最终视线方向为：d = e-s。注意，在做进一步计算前先将 d 归一化。

[0044] 如图 2 体数据在模型空间分布示意图所示，在模型空间中，其中顶点和边的序号用于建立快速生成纹理多边形所需的查找表；

[0045] 原始体数据按图 2 所示分布一个大长方体中，则按其中⓪①②③④⑤⑥⑦八个顶点在视线方向上哪个离视点最远，共有八种不同的子数据块绘制顺序，即总是从离视点最远的顶点处的子数据块开始，从后向前绘制。因此这里只要计算八个顶点在视线 d 上的投影，找出其中最大的即可确定绘制子数据块的顺序。

[0046] 步骤 3：分块绘制

[0047] 按步骤 2 所确定的子数据块绘制顺序遍历所有子数据块，对每个子数据块：

[0048] 首先应用步骤 1 生成的传递函数查找表快速生成最终绘制所用的三维纹理块，并将此三维纹理块导入到显卡的纹理缓存中。

[0049] 然后，用垂直于视线 d 的平面从后向前切割三维纹理块长方体，生成纹理多边形，如图 3 生成的纹理多边形示意图所示。长方体边与平面求交可得到多边形顶点，为避免复杂的点集凸壳计算，我们用另一种方法来快速地生成这些多边形。

[0050] 当平面与长方体相交时，其八个顶点共有两种状态：在平面正向空间内及平面上（代入平面方程求得的值大于等于 0）或者在平面负向空间内（代入平面方程求得的值小于 0）。所以平面与长方体相交最多有 $2^8 = 256$ 种情况，实际上其中有一部分是不可能出现的，但是为了简化查找表结构以加快查找，所以仍采用 256 个元素的列表记录这些情况，八个顶点的状态码排在一起恰好作为索引，而每个列表元素记录长方体与平面相交的边的序号如图 2 中所示为 0、1、2、3、4、5、6、7、8、9、10、11，边序号顺序就是生成多边形的顶点顺序。一个平面最多同时与长方体的 6 条边相交，所以每个列表元素固定为 7 个元素的整数数组，其中第一个元素为多边形顶点个数，后面依次排列顶点所在长方形边的序号。以图 4 中一个纹理多边形的例子所示，纹理多边形切割平面的顶点依次相交于长方体边 1、边 9、边 8、边 3 上，处于平面向外空间内的顶点为 ⓪①，处于平面向内空间内的顶点为②③④⑤⑥⑦，在查找表中得索引值为 3，即 8 位二进制数 00000011，而该列表元素则为 {4,1,9,8,3,-1,-1}。这样，生成多边形时，只需计算八个顶点的状态码、得到索引、查表得到多边形顶点序列、依次计算各顶点坐标即可，避免了先求顶点坐标再生成多边形所需的复杂运算，加快了绘制流程。

[0051] 最后，仍然按从后向前的顺序绘制所有生成的纹理多边形，得到子数据块的绘制结果。混合 (Blend) 运算采用如下方式：

[0052] $$C_{dst} = (1-\alpha_{src})C_{dst} + \alpha_{src}C_{src} \quad (4)$$

[0053] 其中，$C_{src}$ 为当前绘制颜色，$C_{dst}$ 为帧缓存中的颜色，$\alpha_{src}$ 为当前绘制颜色的 alpha 分量（相当于不透明度）。

[0054] 由于子数据块的绘制也是按照从后往前的顺序，所以当所有子数据块绘制完成后即得最终的绘制结果。运行结果

[0055] 我们在一台计算机上，用 C++ 编制程序实现了上述算法，以验证本发明所提算法的有效性和实用性。图 5 是本发明的方法用于实际医学影像的例子，该体数据大小为 1040×1280×1125，总容量为 1.39GB，为更好地验证算法的有效性，我们也实现了 in-core 和 out-of-core 的光线投射算法，并在同一组数据集上做了对比，结果如表格 1 所示。其中数据 1、2、3 的数据量分别为 11.43MB、308MB 和 1.39GB，从结果可以看出在小数据量下本发明的算法甚至比 in-core 的算法还要快，同时数据所占用的内存要小得多。而在大数据量的条件下，in-core 算法已无能为力，而本发明的算法仍然比直接的 out-of-core 算法要快得多。所有测试的硬件和软件环境为：Pentium 42.8GHz 处理器，1GB 物理内存，GeForce 6800 显卡，Windows 2000 操作系统。

[0056] 表格 1 算法运行结果的对比

[0057]

| | 本发明的算法 | | in-core 算法 | | out-of-core 算法 | |
|---|---|---|---|---|---|---|
| | 绘制时间 | 内存占用 | 绘制时间 | 内存占用 | 绘制时间 | 内存占用 |
| 数据 1 | 0.142s | 170KB | 2.19s | 11.43MB | 95.7s | 1.63MB |
| 数据 2 | 2.52s | 1.3MB | 6.95s | 308MB | 741s | 14.6MB |
| 数据 3 | 25.5s | 1.5MB | N/A | N/A | ≥1h | 40.6MB |

[0058] 以上所述，仅为本发明中的具体实施方式，但本发明的保护范围并不局限于此，任何熟悉该技术的人在本发明所揭露的技术范围内，可理解想到的变换或替换，都应涵盖在本发明的包含范围之内，因此，本发明的保护范围应该以权利要求书的保护范围为准。

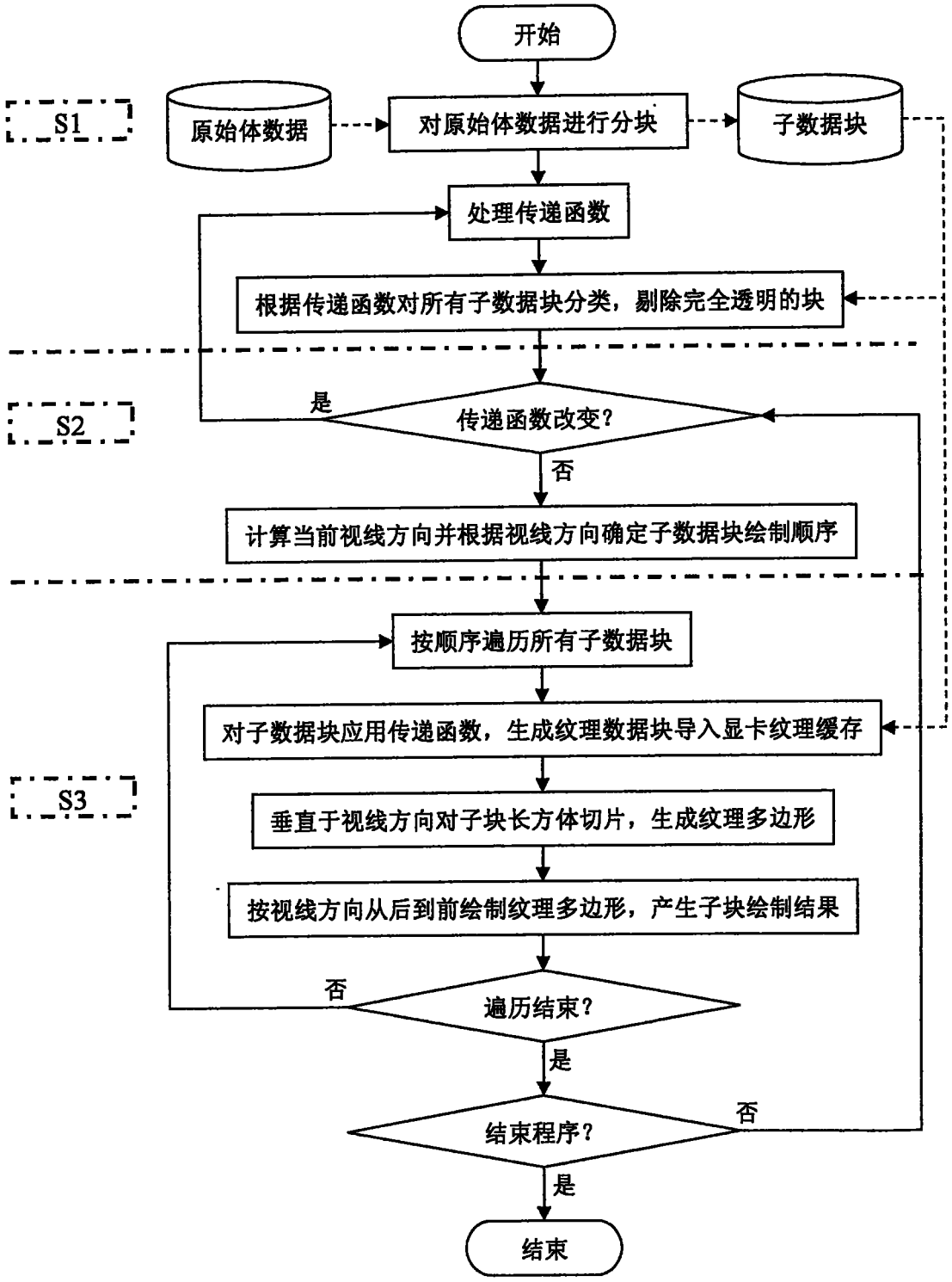

开始
S1
原始体数据
对原始体数据进行分块
子数据块
处理传递函数
根据传递函数对所有子数据块分类，剔除完全透明的块
S2
是
传递函数改变？
否
计算当前视线方向并根据视线方向确定子数据块绘制顺序
按顺序遍历所有子数据块
对子数据块应用传递函数，生成纹理数据块导入显卡纹理缓存
S3
垂直于视线方向对子块长方体切片，生成纹理多边形
按视线方向从后到前绘制纹理多边形，产生子块绘制结果
否
遍历结束？
是
否
结束程序？
是
结束

图 1

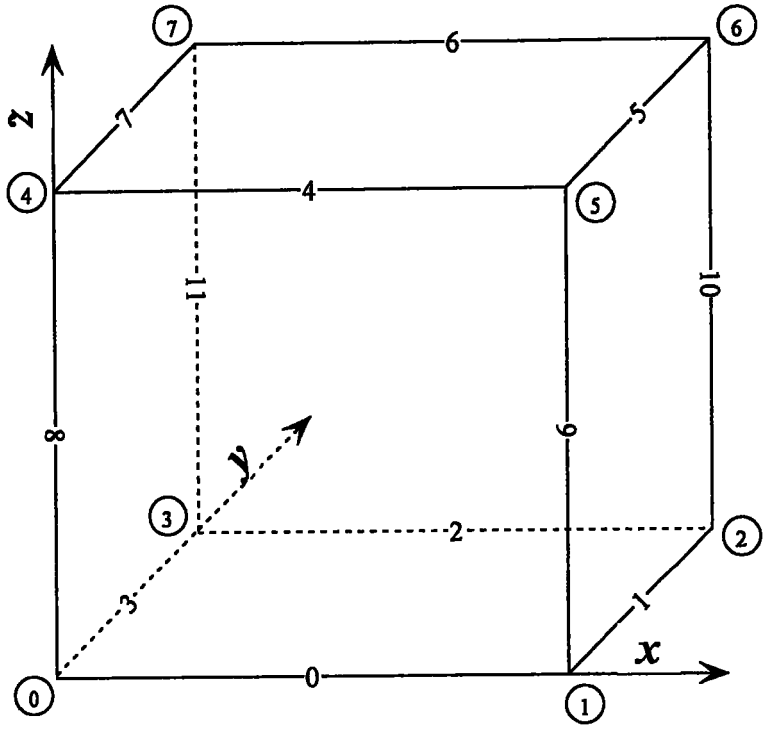



图 2

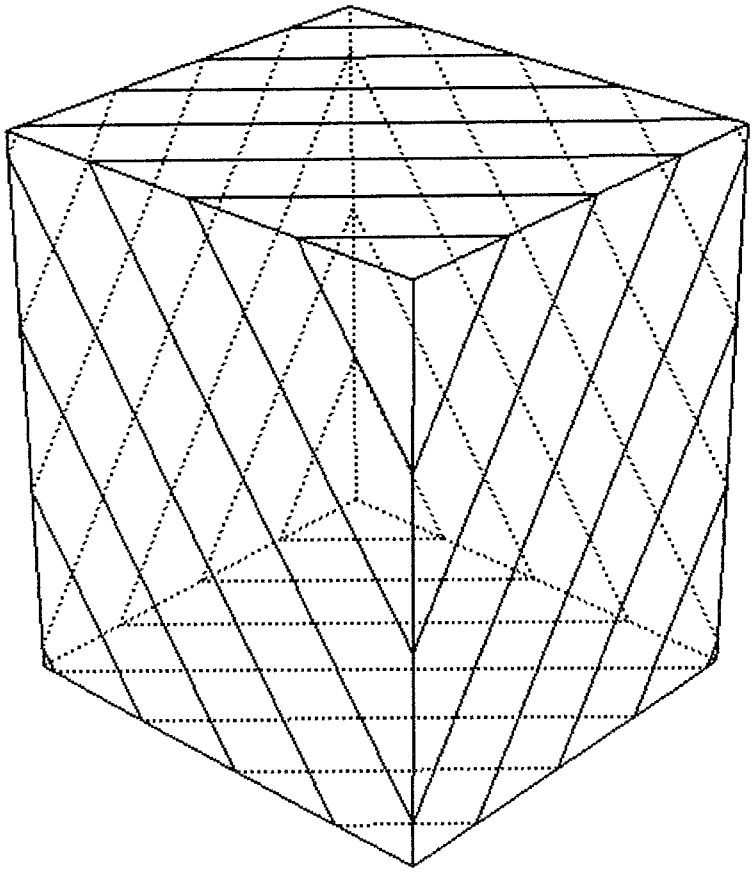


图 3

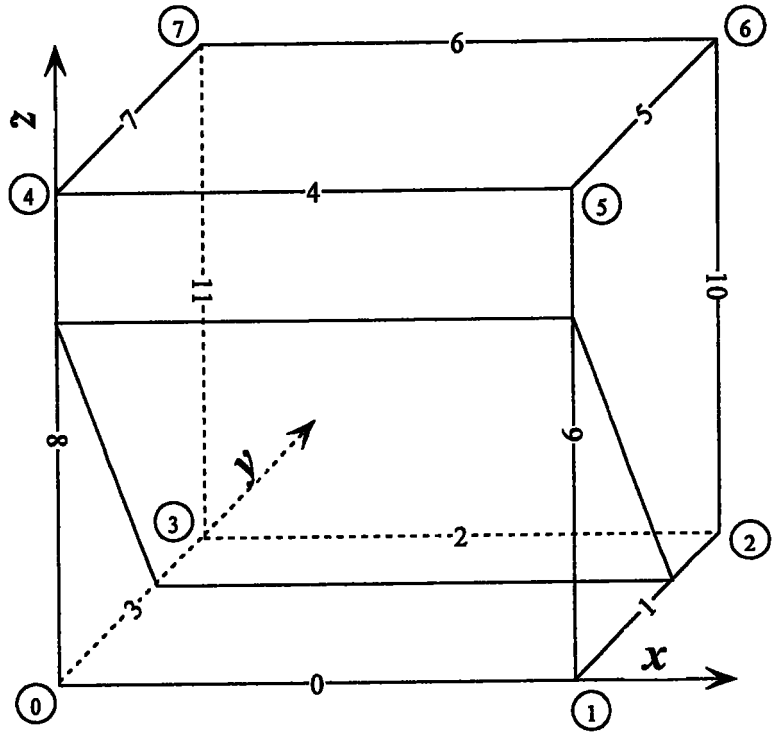



图 4

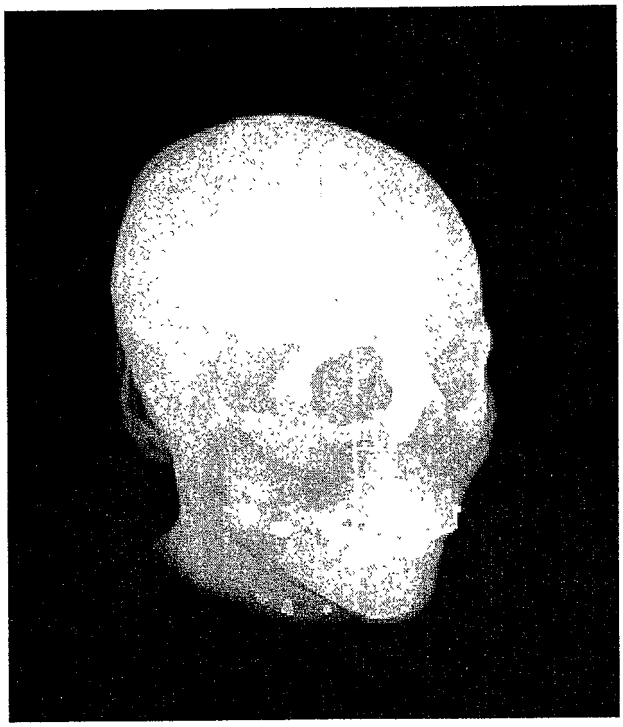


图 5