基于三向 Dexel 的 GPU 加速数控仿真方法

李一鹏 贾世宇 潘振宽 王吉强 李 冰
(青岛大学计算机科学技术学院 山东 青岛 266000)

摘 要 为了提升数控仿真的精度和速度,提出并行加速的基于三向 Dexel 的数控仿真方法。采用三向 Dexel 模型来表示工件与刀具,获得比使用单向 Dexel 模型的方法更真实的仿真效果。在表面重构方面,使用时间开销较大但效果更好的双轮廓法,通过使用并行化方法来解决仿真的实时性问题。仿真使用多线程加速和 GPU 加速技术,前者用于加速仿真中的布尔运算和三向 Dexel 数据转换过程,后者用于提高表面重构的速度。仿真实验结果表明,与仅适用 CPU 多线程的实现相比,应用 GPU 加速后表面重构的速度提高了 5 ~ 15 倍,CPU 多线程的实现相比单线程的速度提高了 232% ~289%,能够满足数控仿真在精度和速度上的要求。

关键词 数控加工仿真 三向深度元素 双轮廓 多线程加速 GPU 加速

中图分类号 TP391.9 **文献标志码** A DOI:10.3969/j.issn.1000-386x.2020.11.011

A GPU ACCELERATED NC SIMULATION METHOD BASED ON TRI-DEXEL

Li Yipeng Jia Shiyu Pan Zhenkuan Wang Jiqiang Li Bing
(*College of Computer Science and Technology, Qingdao University, Qingdao 266000, Shandong, China*)

Abstract To improve the accuracy and speed of NC-simulation, this paper presents a parallel accelerated NC simulation method based on Tri-dexel. It represented workpiece and tool instance with Tri-dexel models to obtain more authentic simulated results than traditional dexel. In the aspect of surface reconstruction, the double contouring, which costs more time but has better effect, was used, and the parallel method was used to solve the real-time problem of simulation. Multi-threading acceleration and GPU acceleration were used in the simulation. The former was used to accelerate Boolean operation and Tri-dexel data conversion process, while the latter was used to improve the speed of surface reconstruction. The simulation results show that compared with the implementation of CPU multi-threading only, the speed of surface reconstruction accelerated by GPU is 5 ~ 15 times, and the speed of CPU multi-threading is 232% ~ 289% higher than that of single thread, which can meet the requirements of accuracy and speed of NC simulation.

Keywords NC simulation Tri-dexel Dual-contouring Multi-threading GPU acceleration

0 引 言

数控仿真对数控加工有着重要的意义。数控仿真通过在虚拟环境中使用数控加工程序模拟加工过程来验证其正确性,避免因数控加工程序错误而导致的工件质量下降和资源浪费。

现代数控仿真一般使用基于空间分割的方法来表示工件模型实体,其中由 Hook[1] 提出的 Dexel 方法广泛地应用于国内外商业软件和仿真系统中。基于 Dexel 的仿真方法的主要问题在于表示几乎平行于 Dexel 方向的部分实体时精度较差,存在难以避免的阶梯化现象与信息丢失的问题。其改进版本三向 Dexel 方法则可以在保留其简单布尔操作以及较小空间占用优点的同时增加建模的精度。三向 Dexel 方法在数控仿真中应用的主要问题在于复杂的表面重构过程。目前主流的做法是利用三向 Dexel 模型与体素(Voxel)模型的相似性,将三向 Dexel 模型转换成体素模型,再

收稿日期:2019 - 07 - 05。国家十二五科技支撑计划项目(2013BAI01B03)。**李一鹏**,硕士生,主研领域:虚拟仿真技术。**贾世宇**,副教授。**潘振宽**,教授。**王吉强**,硕士。**李冰**,硕士生。

使用成熟的体素模型表面重构算法进行表面重构[2]。于珊[3]通过使用 GPU 加速的移动立方体(Marching Cube)[4-5]方法进行表面重构提高了数控仿真的速度。Peng 等[6]提出的基于轮廓的表面重构方法及 Ren 等[7]提出的基于网格的表面重构方法相比于将三向 Dexel 模型转换为体素模型,其算法复杂度较高,时间消耗较大,且难以使用并行技术进行加速。Jachym 等[8]通过使用 OptiX 光线追踪引擎来获取更具真实感的重构表面,但难以实现流畅的实时仿真。Inui 等[9]提出的四面柱(Quad Pillars)算法拥有较高的并行度和执行效率,但是仅适用于单向 Dexel 模型的表面重构。

相比于移动立方体方法,双轮廓(Dual Contouring, DC)[10]方法通过使用厄米特数据(Hermite data,物体表面点及其法线数据)计算特征点,可以还原由于采样问题丢失的特征(如尖锐棱角特征)。双轮廓法在数控仿真中应用的主要限制在于其高昂的计算开销,在 CPU 端进行的双轮廓法往往难以满足数控仿真对于实时性的要求,因此目前该方法还未在数控仿真中有所应用。

针对上述问题,本文提出一种并行化的基于三向 Dexel 的数控仿真方法,通过使用三向 Dexel 建模方法提高仿真精度,采用双轮廓方法进行表面重构,并通过运用并行加速技术提高计算速度。相较于现存的数控仿真方法,在仿真的精度相同时,本文方法使用的 GPU 加速的双轮廓算法能够更准确地还原出数控加工工件的尖锐棱、角特征,为实际加工提供更加准确的仿真结果。同时通过应用多线程加速和 GPU 加速技术,使得中等配置(参考配置见仿真结果部分)的计算机也能实现流畅、准确的仿真,降低了对计算机硬件的要求。

1 仿真方法

数控仿真方法的结构如图 1 所示。数控仿真程序在 CPU 端、GPU 端有不同的分工。仿真程序从存储工程文件中读取信息后,在 CPU 端根据胚料信息进行实体建模,根据刀具参数计算每行 NC 代码控制下刀具所扫过的空间体(下简称为扫掠体);将工件实体与扫掠体进行相减操作;将所得三向 Dexel 模型转换成适合进行表面重构的结构后发送到 GPU 端;由 GPU 对表面进行重构,并对重构所得多边形表面进行图形渲染。如此重复上述过程,直至 NC 代码读取完毕。

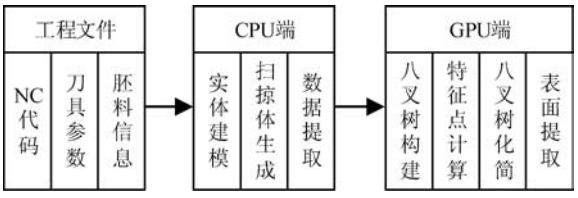


图 1 数控仿真方法示意图

2 仿真算法实现

2.1 三向 Dexel 实体建模方法

数控仿真使用实体间的布尔减运算来描述数控加工中刀具切削工件的过程,为此需要对工件和刀具进行实体建模。单向与三向 Dexel 模型示意图如图 2 所示。单向 Dexel 模型通过记录一组平行且等距的光线在实体内部的部分来表示实体,这种方式会造成如图 2(a)所示的信息丢失。三向 Dexel 模型则通过如图 2(b)所示的方式,在三个相互垂直的方向进行取样来减少信息丢失,从而提高仿真精度。三向 Dexel 模型的结构由三组单向 Dexel 模型组成。

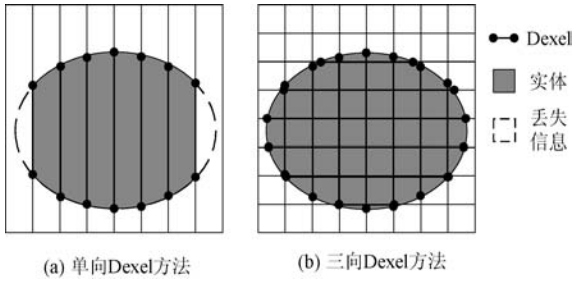


图 2 单向 Dexel 与三向 Dexel 模型示意图

2.1.1 单向 Dexel 模型的建模

单向 Dexel 模型在其 XY 平面上,沿 X 轴、Y 轴正方向以仿真精度 P 为间隔分布着向 Z 轴正向发射的光线。每条光线与实体相交的交点都被保存下来,每两个交点之间的线段代表着光线在实体内部的部分,称为一个 Dexel。构成 Dexel 的两个交点中距离 XY 平面距离较远的为 Dexel 的上端点,较近的为下端点。同一条光线上的 Dexel 端点依据距离发射点的距离按升序排列形成的链状结构称为 Dexel 链。这些 Dexel 链可以构成一个二维的网格(Grid)结构。假定网格结构沿 X 轴、Y 轴方向分为 n_X、n_Y 份,则该 Dexel 模型的分辨率为 $n_X \times n_Y$,空间复杂度为 $O(n_X \times n_Y)$。

为了压缩 Dexel 模型所需的储存空间,在记录端点位置信息时仅记录端点到 XY 平面的距离。需要获取端点的确切坐标时,依据当前 Dexel 链在二维网格中的位置进行计算即可。坐标为(i,j)的 Dexel 链上端点的坐标(x,y,z)为:

$$x = O_X + P \times i \quad y = O_Y + P \times j \quad z = O_Z + d_Z \tag{1}$$

式中:$\{O_X, O_Y, O_Z\}$ 为单向 Dexel 坐标系中的原点;P 为仿真精度;d_Z 为端点到 XY 平面的距离。

2.1.2　三向 Dexel 模型的建模

三向 Dexel 模型的结构由三个相互垂直方向的单向 Dexel 模型叠加得到。由于数控加工的仿真过程中仅包含布尔减运算,仿真中可能发生布尔运算的范围可以限定在工件胚料的包围盒中。建模时选择在胚料包围盒的 X、Y、Z 方向分别建立编号为 0、1、2 的三组单向 Dexel 模型。第 i 组单向 Dexel 模型的局部坐标系 $\{\boldsymbol{O}, \boldsymbol{e}_X, \boldsymbol{e}_Y, \boldsymbol{e}_Z\}$ 为:

$$\boldsymbol{O} = \boldsymbol{B}_{\min} \quad \boldsymbol{e}_X = \boldsymbol{e}_{(i+1) \bmod 3} \quad \boldsymbol{e}_Y = \boldsymbol{e}_{(i+2) \bmod 3} \quad \boldsymbol{e}_Z = \boldsymbol{e}_i \tag{2}$$

式中:$\boldsymbol{O}$ 为单向 Dexel 模型局部坐标系的原点;$\boldsymbol{e}_X$、$\boldsymbol{e}_Y$、$\boldsymbol{e}_Z$ 为单向 Dexel 模型局部坐标系 X、Y、Z 轴的单位向量;$\boldsymbol{B}_{\min}$ 为胚料包围盒的最小点;$\boldsymbol{e}_0$、$\boldsymbol{e}_1$、$\boldsymbol{e}_2$ 为胚料包围盒坐标系的 X、Y、Z 轴单位向量。

三向 Dexel 模型的分辨率 $N_X \times N_Y \times N_Z$ 为:

$$\begin{aligned} N_X &= \mathrm{ceil}[(\boldsymbol{B}_{\max}.x - \boldsymbol{B}_{\min}.x)/P] \\ N_Y &= \mathrm{ceil}[(\boldsymbol{B}_{\max}.y - \boldsymbol{B}_{\min}.y)/P] \\ N_Z &= \mathrm{ceil}[(\boldsymbol{B}_{\max}.z - \boldsymbol{B}_{\min}.z)/P] \end{aligned} \tag{3}$$

式中:$\boldsymbol{B}_{\max}$ 为胚料包围盒的最大点;P 为仿真精度。

2.1.3　三向 Dexel 模型的布尔运算

文献[11]详细描述了三向 Dexel 的并、交、差布尔运算,数控仿真中主要用到的为三向 Dexel 的减运算。三向 Dexel 模型之间进行布尔运算时,对双方 X、Y、Z 方向的单向 Dexel 模型分别进行布尔运算,即为相对应光线上 Dexel 的减运算。数控仿真中运算根据刀具 Dexel 和工件 Dexel 的相对位置关系,共有图 3 所示的 6 种情况。

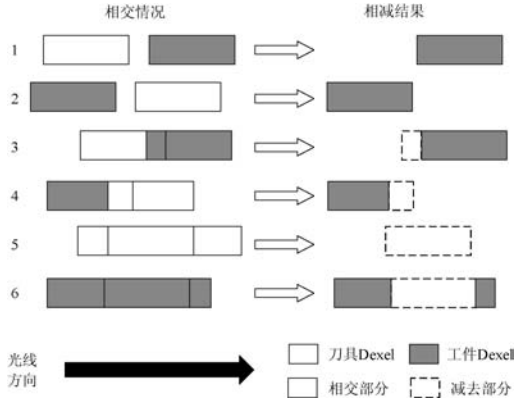


图 3　Dexel 间的相减运算

(1) 刀具 Dexel 与工件 Dexel 没有任何相交,不需要修改工件 Dexel。

(2) 与情况 1 相同,不需要修改工件 Dexel。

(3) 工件 Dexel 的下端点到刀具 Dexel 上端点之间的部分被切除。

(4) 刀具 Dexel 的下端点到工件 Dexel 的上端点之间的部分被切除。

(5) 工件 Dexel 被完全删除。

(6) 刀具 Dexel 的下端点到上端点之间的部分被切除,工件 Dexel 分裂成两个 Dexel。

可以看出,相减产生的新 Dexel 端点的位置为相应刀具 Dexel 端点的位置,法线与该刀具 Dexel 端点法线方向相反。

2.2　刀具扫掠体生成

刀具扫掠体的生成有两种方法:第一种为近似的方法[12],根据操作的轨迹位置、刀轴角度信息以一定的间距进行插值计算,将扫掠体离散为多个刀具实体;第二种方法为精确的方法[13-14],通过刀具轮廓线和操作起止点信息精确地计算刀具的扫掠体。精确计算的主要优势在于精度高于前者,劣势在于计算需要更多时间。因此本文选择近似的构造方法,以仿真精度 P 的一半为间隔插值生成刀具实例,将这些实例的集合作为当前操作所生成扫掠体的近似。

工件与刀具的减运算首先需要生成刀具实例的三向 Dexel 模型。为了减少建模的运算量,在三向 Dexel 模型的局部坐标系中生成该实例的包围盒,仅在该范围内进行模型生成。之后从工件三向 Dexel 模型中减去所生成的模型。其多线程加速运作机制如图 4 所示。假定共有 i 个辅助线程,将其分别编号为 1,2,…,n,创建辅助线程的主线程也作为 0 号线程参与运算,则第 i 号线程负责每个刀具实例包围盒内第 $k(n+1)+i$ 条 Dexel 链的构建与对应工件 Dexel 链的相减运算。

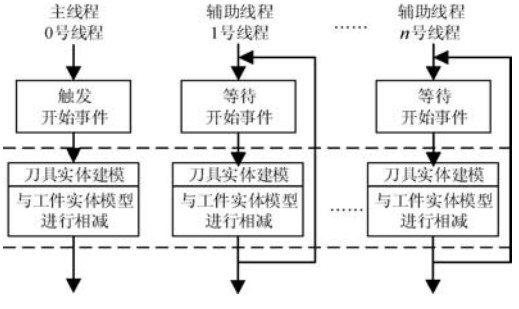


图 4　布尔减运算的并行运作机制

2.3　工件的表面重构

三向 Dexel 模型不适用于可视化,为了获取适合图形设备渲染的多边形表面,需要对实体进行表面重构。本文方法在表面重构时,在 CPU 端将三向 Dexel 模型转化为厄米特数据,并在 GPU 上进行八叉树的构建、特征点的计算、八叉树的化简,以及重构表面的提取。

如图 5 所示,由于三向 Dexel 模型的采样精度有限,直接使用表面重构算法(如移动立方体算法,以下

简称 MC 算法)进行表面重构会产生如图 5(c)所示的特征丢失。双轮廓(简称 DC)算法则可以通过使用由三向 Dexel 模型提取而来的厄米特数据,通过计算每个体素内部的特征点(图 5(d)中的白色点)来还原其丢失特征。表面重构算法的步骤如下:

(1) 在 CPU 中将三向 Dexel 数据转换为适合构建八叉树的数据,并将数据送入 GPU;

(2) 在 GPU 上生成能够完全容纳网格数据的最小八叉树;

(3) 并行处理每个叶子节点,对其特征点进行计算;

(4) 对八叉树进行化简,并提取重构表面。

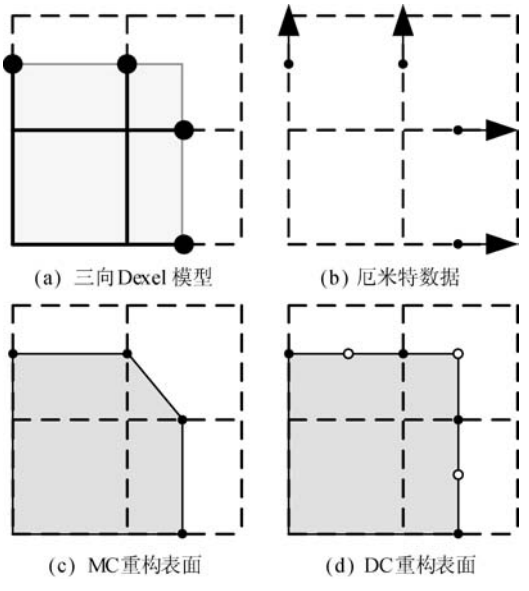


图 5 移动立方体与双轮廓表面重构

2.3.1 网格信息与赫米特数据的提取

八叉树的建立需要包含完整三向 Dexel 模型的网格信息,因此需要建立与三向 Dexel 模型同等分辨率的网格,网格格点及厄米特点的数据结构如图 6 所示。为了节省显存,格点数据中不保存完整的厄米特点数据,只保存索引值。索引数组的 0、1、2 号元素分别代表格点 X、Y、Z 方向的厄米特点。厄米特点的数据单独封装,以一维数组的形式传递给 GPU。格点信息的提取通过遍历三向 Dexel 模型中的 Dexel 完成,如图 7 所示。首先为每个 Dexel 的端点分配索引值,并将其所经过的格点均设置为内部格点,在最靠近 Dexel 上下端点的内部格点处记录对应 Dexel 端点的索引值。

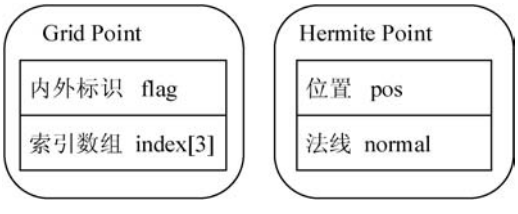


图 6 格点与厄米特点的数据结构示意图

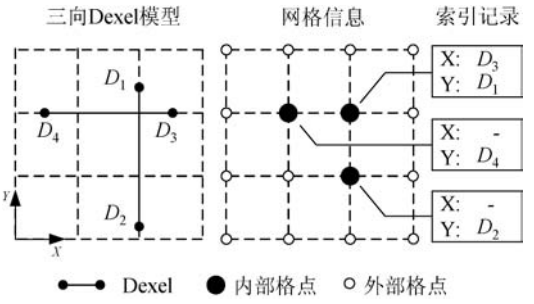


图 7 从三向 Dexel 模型中提取网格信息

2.3.2 八叉树的建立

双轮廓法结合自适应精度八叉树可以简化生成的表面,有效减少表面包含三角形的数量,提高渲染速度。本文使用的在 GPU 端运行的并行八叉树生成算法[15]能够在节约显存的同时提升算法的运行效率,可以极大缩短八叉树构建时间。所构建八叉树的最大深度由传入网格数据的尺寸决定,大小为能够完全容纳网格的最小八叉树。由于 GPU 并不支持动态内存分配,每层节点所需要的存储空间需要提前分配。该方法首先构造叶子节点,并从底层向上依次进行八叉树的构造。若一个叶子节点为非空叶节点(8 个角点中既包含内部格点又包含外部格点的叶节点),则构建一个新的节点,使用原子计数器对其进行编号并使用查询表来存储子节点与父节点之间的关系。在建立八叉树的过程中需要统计构建叶节点数量,用于为后续步骤会用到的二次误差方程数据分配存储空间。八叉树构建完成后,使用构建的八叉树来进行特征点的计算和八叉树的简化。

2.3.3 特征点的计算

双轮廓法中特征点的计算是通过求解叶节点的二次误差方程(Quadratic Error Function,QEF) 得到的。QEF 的定义如下:

$$E[\boldsymbol{x}] = \sum_i (\boldsymbol{n}_i \cdot (\boldsymbol{x} - \boldsymbol{p}_i))^2 \tag{4}$$

式中:$\boldsymbol{p}_i$、$\boldsymbol{n}_i$ 代表着该叶子节点所对应体素边上厄米特点的位置与法线,可以通过查询该体素 8 个角点对应格点的索引数组得到。将式(4)转化为矩阵形式并展开为:

$$E[\boldsymbol{x}] = \boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x} - 2\boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{b} + \boldsymbol{b}^{\mathrm{T}}\boldsymbol{b} \tag{5}$$

式中:矩阵 $\boldsymbol{A}$ 代表着体素边上厄米特点的法线 $\boldsymbol{n}_i$;向量 $\boldsymbol{b}$ 的第 i 个分量为 $\boldsymbol{n}_i\boldsymbol{p}_i$,求解方程最小值所得的 $\boldsymbol{x}$ 即是该体素的特征点。方程的具体解法参照文献[7]。为了方便后续八叉树的简化及多边形表面的提取,由体素构造出的 QEF 及其计算结果被单独封装在 QEF 类中,存储于事先分配好的存储空间中并由原子计数器为其分配索引值。构建完成后,在八叉树节点中记录其 QEF 的索引值以建立对应关系。所有叶节点的特

征点计算完成后,对八叉树进行简化。

2.3.4　八叉树的化简与重构表面的提取

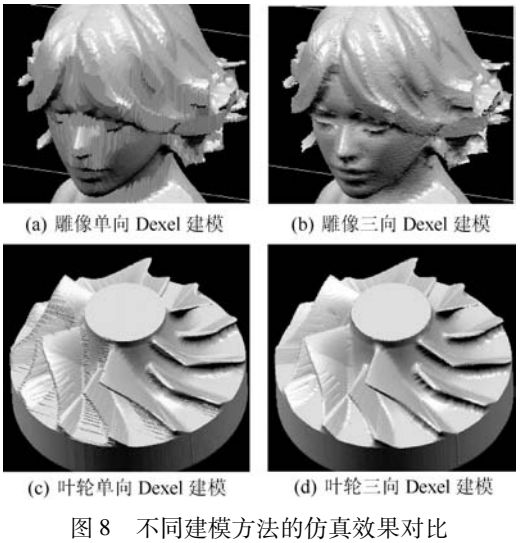
对八叉树进行化简的目标是使用更少的三角形来表示重构表面的平缓区域,在保持图形特征不改变的前提下减少渲染三角形的总量,以降低图形设备的负担。首先将同属于一父节点的 8 个叶节点的 QEF 中各项相加,若计算相加所得 QEF 解得的误差值小于一给定阈值,则可以对这些叶节点进行化简。化简后将叶节点的特征点修改为父节点的特征点,并将父节点标记为伪叶子节点,如此往复由底向上逐层进行化简。

化简完成后遍历八叉树的每个叶节点,对其进行表面提取。将每个叶节点的特征点提取到数组中,使用原子计数器为其分配索引值。最后根据叶节点之间的拓扑关系设置索引,完成多边形表面的提取。

3　仿真结果

数控仿真系统软件使用 C ++ 编写 CPU 端程序,使用 OpenCL 编写 GPU 端程序,使用 OpenGL 进行图形渲染,运行硬件环境为 Intel Core i7-3770 四核 CPU,12 GB 内存,2 GB 显存的 NVIDIA GeForce GTX 660 Ti 显卡。

仿真实验 1 测试不同建模方法的仿真效果,如图 8 所示,其中:(a)、(c)为使用单向 Dexel 建模方法,(b)、(d)为使用三向 Dexel 建模方法。可以看出,应用三向 Dexel 建模方法后消除了使用单向 Dexel 建模方法时的阶梯化问题。



(a) 雕像单向 Dexel 建模　(b) 雕像三向 Dexel 建模

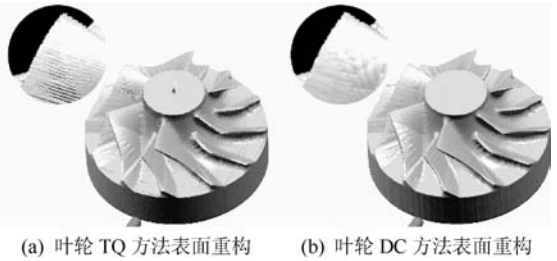
(c) 叶轮单向 Dexel 建模　(d) 叶轮三向 Dexel 建模

图 8　不同建模方法的仿真效果对比

仿真实验 2 为双轮廓法对特征修复的验证,图 9、图 10 分别为该方法对雕像模型和叶轮模型丢失特征的修复效果。图 9(a)、图 10(a)为使用三向四面柱(Tri-Quadpillars,简称 TQ,将文献[9]中的算法扩展到三维得到)算法进行表面重构得到的结果图;图 9(b)、图 10(b)为使用双轮廓法进行表面重构得到的结果图。可以看出,使用双轮廓法后,由特征丢失导致的毛刺、锯齿问题得到了修复,同时能够更好地保存尖锐棱角等信息。

(a) 雕像 TQ 方法表面重构　(b) 雕像 DC 方法表面重构

图 9　双轮廓法对雕像模型丢失特征的修复效果



(a) 叶轮 TQ 方法表面重构　(b) 叶轮 DC 方法表面重构

图 10　双轮廓法对叶轮模型丢失特征的修复效果

仿真实验 3 测试仿真方法加速前后的运行效率。表 1 为仿真实验 3 中所使用的实验对象,实验共使用两组数控加工程序:第 1 组为雕像的加工,第 2 组为叶轮的加工。

表 1　仿真所用实验对象

实验对象	NC 程序	NC 代码(行)	模型分辨率
雕像 1	第 1 组	80 411	167 × 167 × 180
雕像 2	第 1 组	80 411	250 × 250 × 250
叶轮 1	第 2 组	26 255	167 × 167 × 84
叶轮 2	第 2 组	26 255	250 × 250 × 125

表 2 所示为多线程加速前后表面重构的平均时间。可以看出,多线程加速比单线程双轮廓法表面重构快 232% ~289%。由于双轮廓算法中八叉树构建所耗费的时间在算法总耗时中占比最大,且该部分并行化程度较高,多线程加速能够获得较高的加速比,但由于双轮廓算法本身时间开销较大,仅在 CPU 端进行加速仍然难以满足实时仿真的要求。

表 2　多线程加速前后表面重构平均时间

实验对象	单线程/ms	多线程加速/ms	速度比
雕像 1	3 594.54	974.09	3.69
雕像 2	5 255.78	1 579.44	3.32
叶轮 1	3 165.93	813.92	3.89
叶轮 2	3 879.32	1 078.25	3.60

表3 展示了 GPU 加速前后表面重构平均时间。可以看出 GPU 加速比 CPU 下多线程加速双轮廓法表面重构快 5 ~ 15 倍。完全在 GPU 端进行的双轮廓算法在重构对象分辨率升高时,因网格数据与厄米特数据的增大而增加了 GPU 的读写压力,造成了加速比的下降,但仍然能满足实时仿真的要求。

表 3 GPU 加速前后表面重构平均时间

实验对象	多线程加速/ms	GPU 加速/ms	速度比
雕像 1	974.09	93.89	10.37
雕像 2	1 579.44	246.89	6.40
叶轮 1	813.92	49.52	16.44
叶轮 2	1 078.25	126.89	8.50

4 结 语

本文实现了一种并行化的基于三向 Dexel 的数控仿真方法。采用三向 Dexel 建模方法提升仿真的精度,使用双轮廓法来获取更加真实的重构表面。该方法的创新之处在于其通过双轮廓算法修复因表面特征丢失造成的毛刺、锯齿问题,并通过将多线程加速、GPU 加速与双轮廓算法结合起来,使得表面重构的速度相对于多线程的实现提升了 5 ~ 15 倍,能够在满足仿真实时性要求的前提下,修复由于三向 Dexel 模型采样问题造成的特征丢失,从而提高仿真的真实度。

参 考 文 献

[1] Hook T V. Real time shaded NC milling display[J]. Computer Graphics,1986,20(4):15 - 20.

[2] 侯增选,Krause F L,田荣鑫. 基于压缩 Voxel 模型的五坐标数控加工仿真新方法[J]. 计算机工程与应用,2006,42(20):25 - 28.

[3] 于珊. 高性能数控仿真显示模式的研究[D]. 贵阳:贵州大学,2015.

[4] Lorensen W E,Cline H E. Marching cubes:A high resolution 3D surface construction algorithm[C]//Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques. ACM,1987:163 - 169.

[5] Lewiner T,Lopes H,Vieira A W,et al. Efficient implementation of marching cubes' cases with topological guarantees [J]. Journal of Graphics Tools,2003,8(2):1 - 15.

[6] Peng X B,Zhang W H. A virtual sculpting system based on triple dexel models with haptics[J]. Computer-Aided Design and Applications,2009,6(5):645 - 659.

[7] Ren Y F,Zhu W H,Lee Y S. Feature conservation and conversion of tri-dexel volumetric models to polyhedral surface models for product prototyping[J]. Computer-Aided Design and Applications,2008,5(6): 932 - 941.

[8] Jachym M,Lavernhe S,Euzenat C,et al. Effective NC machining simulation with OptiX ray tracing engine[J]. The Visual Computer,2019,35(2):281 - 288.

[9] Inui M,Umezu N. Quad pillars and delta pillars:Algorithms for converting dexel models to polyhedral models[J]. Journal of Computing and Information Science in Engineering,2017,17(3):031001.

[10] Ju T,Losasso F,Schaefer S,et al. Dual contouring of hermite data[J]. ACM Transactions on Graphics,2002,21(3): 339 - 346.

[11] 高新瑞,张树生,侯增选. 多面体三向 DEXEL 模型与布尔运算[J]. 计算机工程与应用,2007,43(12):3 - 5,56.

[12] 张世民,郭锐锋,彭健钧. 五轴数控加工仿真中刀具扫掠体的计算[J]. 组合机床与自动化加工技术,2010(6): 10 - 14.

[13] 黎先才. 五轴数控加工中通用刀具扫掠体生成方法[J]. 西安工业大学学报,2017,37(8):594 - 599.

[14] 江笑龙,王广官,单岩,等. 凸包 STL 模型的运动包络体计算及其应用[J]. 机械设计与研究,2014,30(3):5 - 7,14.

[15] 王吉强,贾世宇. 基于 GPU 的并行八叉树生成算法[J]. 青岛大学学报(自然科学版),2018,31(4):69 - 75.

(上接第 40 页)

[8] 任泽林. 数据驱动的非线性过程监测方法研究[D]. 哈尔滨:哈尔滨工业大学,2017.

[9] Nguyen V H, Golinval J C. Fault detection based on kernel principal component analysis[J]. Engineering Structures, 2010, 32(11): 3683 - 3691.

[10] Choi S W, Lee C, Lee J M, et al. Fault detection and identification of nonlinear processes based on kernel PCA[J]. Chemometrics and intelligent laboratory systems, 2005, 75(1): 55 - 67.

[11] 张建明,徐磊,许仙珍,等. 基于混合 PCA 模型的多工况过程监控方法[J]. 控制工程,2010,17(4):553 - 556,560.

[12] Landman R, Kortela J, Sun Q, et al. Fault propagation analysis of oscillations in control loops using data-driven causality and plant connectivity[J]. Computers & Chemical Engineering, 2014, 71: 446 - 456.

[13] Yuan T, Qin S J. Root cause diagnosis of plant-wide oscillations using granger causality[J]. Journal of Process Control, 2014, 24(2): 450 - 459.

[14] Naghoosi E, Huang B, Domlan E, et al. Information transfer methods in causality analysis of process variables with an industrial application[J]. Journal of Process Control, 2013, 23(9): 1296 - 1305.

[15] 张琳. 面向高效率负载跟踪的 SOFC 系统优化与控制研究[D]. 武汉:华中科技大学,2015.