改进的自适应特征细分方法及其对Catmull-Clark曲面的实时绘制

林金花[1],王延杰[2],孙宏海[3]

(1.长春工业大学 应用技术学院,长春 130012; 2.中国科学院长春光学精密机械与物理研究所 机械电子工程系,长春 130033; 3.中国科学院大学 大衍学院,长春 130033)

摘 要:传统的自适应特征细分(FAS)算法对曲面上的全部特征点进行统一深度的细分,影响算法的执行效率,针对这一问题,提出了一种自适应特征细分方法。首先,设计了一种模块,即特征块处理单元(FPU),用于计算不规则区域的细分因子,根据Catmull-Clark细分模式来处理特征区域的不规则块,同时减少了GPU渲染块的数目;然后,对FAS的数据结构进行扩充,将关键点数据存放在细分表和渲染表中,GPU对表中数据进行全局细分与绘制,提高了细分和绘制的速度。实验结果表明,改进后的细分表和渲染表结构能够保证细分的动态特性和渲染的实时性。与传统FAS方法相比,本文算法能够保证三维曲面的绘制精度的同时,提高了28%的绘制速度,在实时性方面优于传统FAS方法。

关键词:信息处理技术;自适应特征细分;图形处理器;细分;绘制

中图分类号:TP391 **文献标志码**:A **文章编号**:1671-5497(2018)02-0625-08

DOI:10.13229/j.cnki.jdxbgxb20161210

Improved feature-adaptive subdivision for Catmull-Clark surface model

LIN Jin-hua[1], WANG Yan-jie[2], SUN Hong-hai[3]

(1.*School of Applied Technology, Changchun University of Technology, Changchun* 130012, *China*; 2.*Department of Mechanical and Electronic Engineering, Changchun Institute of Optics, Fine Mechanics and Physics, Chinese Academy of Sciences, Changchun* 130033, *China*; 3.*Dayan Institute, University of Chinese Academy of Sciences, Changchun* 130033, *China*)

Abstract: In order to solve the problem of over-subdivision of Fast Adaptive Segmentation (FAS) algorithm, a dynamic adaptive feature subdivision method is proposed. First, the Feature Processing Unit (FPU) is constructed to generate the dynamic subdivision factor for the irregular block. The subdividing depth of the feature block is calculated and the irregular block of the feature region is refined according to the Catmull-Clark subdivision rule. Then, the number of blocks is controlled for Graphics Processing Unit (GPU), the block buffer and subdivision table are established to deal with each level of the same type of blocks in parallel, which improves the speed of subdivision and rendering. Finally, the data structure of the traditional FAS is expanded to generate new subdivision tables and drawing tables, which support the dynamic subdivision and real-time rendering of blocks.

收稿日期:2016-11-10.

基金项目:"863"国家高技术研究发展计划项目(2014AA7031010B);吉林省教育厅"十三五"科学技术研究项目([2016]345).

作者简介:林金花(1980-),女,讲师,博士.研究方向:数字图像处理.E-mail:ljh3832@163.com

Experiment results show that the structures of the improved subdivision table and rendering table can guarantee the dynamic feature of the subdivision and the real-time rendering. Compared with the traditional FAS, the proposed method can ensure the drawing accuracy of 3D surface model, and increase the drawing speed by 28%, which is better in real-time performance than traditional FAS.

Key words: information processing technology; feature-adaptive subdivision; graphic process unit (GPU); subdivision; render

0 引 言

细分曲面自诞生以来已被广泛应用于三维虚拟现实领域[1-4]。由于细分规则的计算方法较简单,使得细分曲面表示方法的兼容性大大提高。尽管如此,细分表面点的计算需要耗费大量的时间和空间性能,限制了细分曲面技术的使用范围。通用编程图形处理芯片(GPGPU)的出现拓宽了细分曲面技术的使用范围,这种参数化方法提高了细分绘制效率,但是当曲面结构复杂且曲面片规模较大的情况下,GPGPU 的工作性能也随之受到影响。Niessner 等[5]于 2012 年提出了一种新的网格细分技术,从软件方面克服了 GPGPU 的缺陷,直接 GPU 芯片上进行计算分析操作。然而这种方法以参数曲面的表示精度作为代价,绘制速度提高的同时,精度也随之下降。自适应细分方法(FAS)的出现改变了现状,仅对特征区域进行细分操作,既提高了细分性能,又保证了复杂曲面的绘制精度。FAS 广泛应用在三维动画软件和游戏制作领域,例如:OpenSubdiv,一种由皮克斯动画工作室采纳的开源新方案。

与传统细分算法相比,FAS 仅对不规则区域进行细分,不仅保留了不规则区域的拓扑特征,还提高了细分速度。然而,当不规则区域的数量大规模增加时,FAS 算法的效率会随指数降低,限制了算法的时间性能。

为了改善 FAS 方法的实时绘制速度,本文对 FAS 方法进行了改进,提出了一种动态的自适应特征细分算法,通过构造特征块处理单元来动态地提取和细分不规则块,实现不规则区域的不同细分深度的划分,得到较少数量的细分块,减少渲染块的数量,提高绘制速度。此外,本文使用块缓冲区来并行地处理所有细分层次产生的块,提高了细分速度。

1 相关概念和理论

1.1 可编程图形处理器

GPU 是一种单指令流多数据流模式的并行图形处理器。本文细分管线使用 GPU 绘制流程,对 GPU 进行编程,增加了两个功能模块:可编程顶点处理器和可编程块处理器,如图 1 所示。

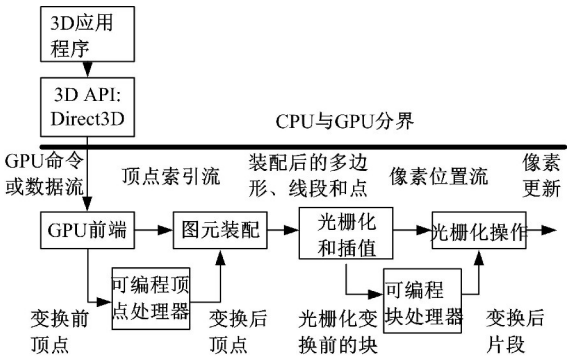


图 1 可编程图形流水线

Fig.1 GPU rendering pipeline

GPU 处理芯片仅对表面点操作,对于上级细分顶点的控制能力较弱。块处理器弥补这点不足,直接在块级上对几何曲面进行计算和光栅化处理,从而可以充分地表现多边形内部细节。

块处理器负责执行处理块参数值的程序,对通过流水线的每一个块都执行一遍。顶点处理器的输出缓冲区中的顶点参数经过插值后,生成块参数传递给块处理器的输入缓冲区,作为其输入参数。程序执行后,改变后的块参数被写入到块处理器的输出缓冲区中。本文算法的细分核心部分就是在块处理器上实现的。

1.2 Catmull-Clark 细分规则

Catmull-Clark 细分模式[6]是一种传统的细分曲面生成方法,通过网格计算规则来重复递增地生成双三次 B 样条参数曲面。本文使用 Catmull-Clark 细分来初始化特征曲面片。Catmull-Clark 细分模式如下所示:

(1)面点:每个面的 4 个顶点记为 V_1、V_2、V_3、V_4,面点计算规则(见图 2(a))如下:

$$V_F=(V_1+V_2+V_3+V_4)/4 \tag{1}$$

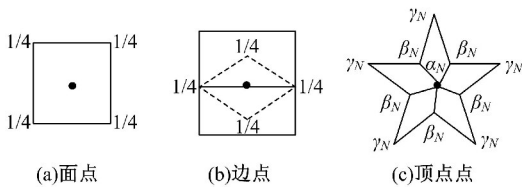
(2)边点:首先将内部顶点记为 V_i、V_j,此顶点的邻接面分别记为 F_1 和 F_2,边点计算规则(见图 2(b))如下:

$$V_E=1/4(V_i+V_j+F_1+F_2) \tag{2}$$

(3)顶点点：点 V 的邻接点分为记为 V_1，V_2，…，V_{2N}，顶点点的计算规则(如图 2(c))如下：

$$V_V = \alpha_N V + \beta_N \sum_{i=1}^{N} V_{2i} + \gamma_N \sum_{i=1}^{N} V_{2i-1} \quad (3)$$

式中：$\alpha_N = 1 - N(\beta_N + \gamma_N)$；$\beta_N = \frac{3}{2N^2}$；$\gamma_N = \frac{1}{4N^2}$。



(a)面点　(b)边点　(c)顶点点

图 2　Catmull-Clark 细分规则

Fig.2　Catmull-Clark subdivision rule

1.3　自适应特征细分(FAS)算法

FAS 算法是 Niessner 等[7,8]提出的一种基于 GPU 的快速曲面绘制方法。该算法包括初始化阶段、细分核心部分和分层绘制 3 个部分。各部分功能详述如下：

(1)初始化阶段。对初始曲面进行分析，定义自适应特征部分，抽取特征块，生成块内控制顶点集合。同时进入索引缓冲单元，用于存放数据，索引缓冲单元主要由信息块和索引表构成。依据 Catmull-Clark 细分模式，得出每个层次的细分点的信息，同时存储到细分表中，即：面点表、边点表和顶点点表。

不同细分层次下的表面拓扑呈现出不同的网格连接结构，为了实现不同细分层次间曲面片的链接，FAS 使用转换块来实现，转换块由 5 种模板组成，分别对应不同细分层级间的表面结构，连接后的表面拓扑结构完整且无缝，如图 3 所示。

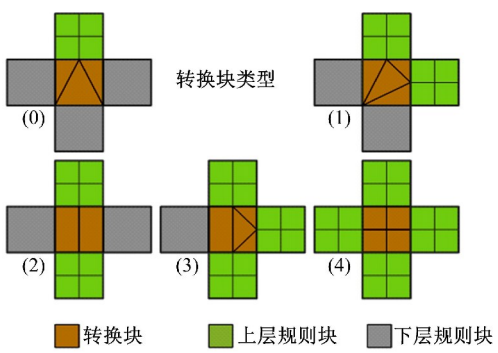


图 3　FAS 转换块模板

Fig.3　FAS join_patch model

(2)细分核心部分。依据细分模板和细分初始网格模型，对不规则区域细分到各自不同的细分深度，得到曲面的关键控制顶点。再根据细分表中的内容，计算得出下级细分层次上的顶点数据，将其存储到 GPU 缓存。每个不规则区域的细分层级是固定不变的。

(3)分层绘制。将前两个部分得到的细分结果和表面模型数据作为本阶段的输入，经过细分后的表面模型，将其存放到对应的 GPU 存储单元中。GPU 处理器将存储单元中的待细分数据完成细分操作，得到顶点和表面拓扑数据，并将其送入渲染器完成整体模型的绘制过程。

由此可见，FAS 对网格模型的全部细分块使用统一的细分深度，通常会出现 GPU 缓冲不够或运行缓慢的现象。本文算法对 FAS 算法中的不规则区域设置了不同的细分深度，对于距离视点较远且分辨率较低的区域使用较低的细分层级，简化了细分计算，保证了模型绘制分辨率，同时提高了 GPU 绘制速度。

2　改进的算法架构

2.1　改进的数据结构

本文对传统 FAS 算法的数据结构进行了扩充，如下所示：

(1)细分表结构

本文对 FAS 的细分表结构进行了改进，如图 4 所示，不同的颜色线圈代表不同细分层级下的网络点，这些网格点的位置信息存放在细分表中，同时为了保证 GPU 快速读取这些顶点信息，本文将生成的二维表存放在 GPU 缓冲中，以便随时读入 GPU 绘制区域完成实时绘制。

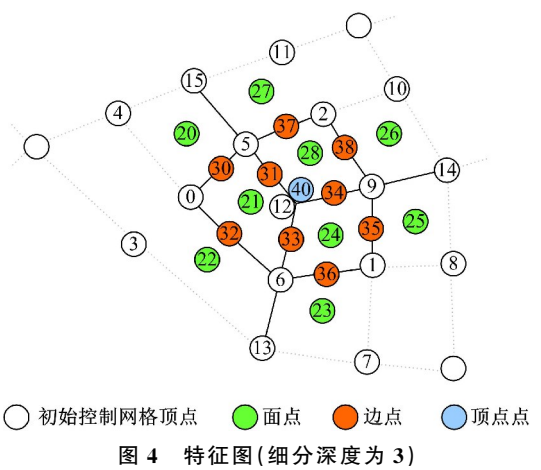


图 4　特征图(细分深度为 3)

Fig.4　Feature mapping(subdivision depth 3)

首先，使用 Catmull-Clark 模板生成曲面模型，存储为面点表、边点表和顶点点表，主要存放顶点和前一级别的关键点。其次，为所有的坐标

点分配各自的编码,构成细分表中的二元信息数据。最后,对GPU全局缓存进行实时更新操作,关键点被存入GPU的细分表。

改进的细分表结构如图5所示[9]。与FAS的区别在于本文仅仅对GPU缓存中存放的少数关键点进行更新操作,并将其存储在细分表中,节省了GPU的计算时间,同时减少了存储单元的占用数量。

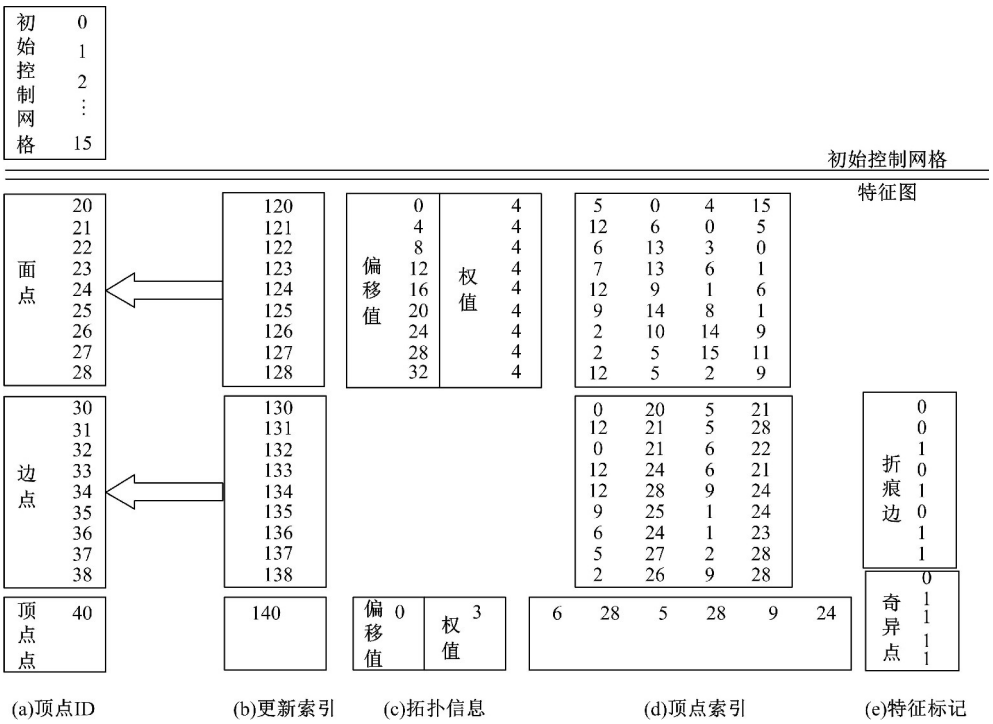


图5 扩充的细分表

Fig.5 Extended subdivision mapping

(2)渲染表

本文将待渲染的数据存放在渲染表中,主要包括奇异点周围块的细分层次d_v、邻接块的拓扑信息和邻接顶点的属性值,FPU接收渲染表中数据,再送入GPU处理单元进行渲染处理,如图6所示。

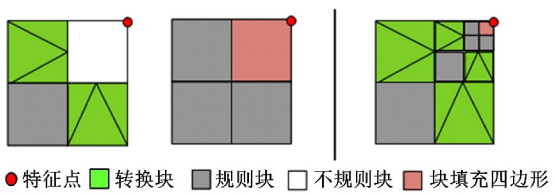


图6 渲染块(d_v=3)

Fig.6 Rendering patch(depth=3)

渲染表中的块属性包括特征块、规则块和不规则块。特征块通常可用于连接周围块的拓扑信息,即作为转换块出现;规则块是无需细分的块;而不规则块是本文自适应细分的对象,即要经过不同层级的细分,实现对奇异点区域的细分绘制。如图6所示,红色顶点表示奇异点,第3幅图表示奇异点周围块已被细分3次,而不同细分层次间使用绿色的转换块实现块间的无缝拼接,每个转换块被分成3个三角形区域,不同的细分深、转换块具有不同的拓扑缩放,当达到细分因子规定的深度后,不规则区域的自适应细分结束,表示奇异点周围的不规则区的拓扑结构已经到达最逼近真实渲染效果的程度,此时停止细分,将渲染表送入GPU,GPU开始对渲染表中的数据进行渲染绘制。图7为渲染块表缓冲区。

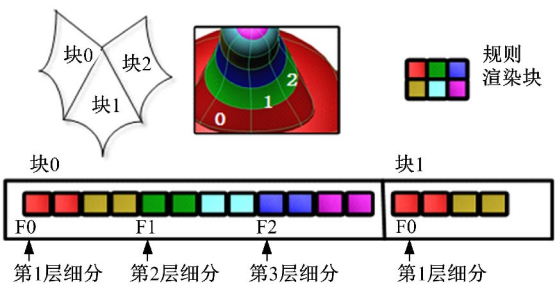


图7 渲染块表缓冲区(d_v=3)

Fig.7 Rendering patch table buffer(depth=3)

2.2 细分因子

本文对FAS的细分因子进行了扩充,生成不规则块的细分层次,进而控制奇异点周围的不规则区域的自适应细分过程,下面依据不同的块属性,将细分因子生成方式进行了划分:

(1)规则块：使用 GPU 内部设置的细分层级。

(2)不规则块：当不规则区域由 n 个顶点构成时，$P=\{p_1,p_2,\cdots,t,\cdots,p_n\}$，$t$ 表示奇异点，P 对应的特征图表示为 $V(P)$。k 表示初始模型中不规则块的顶点，$P'=k\cup\{P\}$，将 P' 所对应的特征图表示为 $V(P')$。当 $V(P')$ 包含 t 所属的块，即奇异点所属特征块与 k 所属于特征块之间的最小距离值与真实值相差 α 时，P 的细分因子计算公式如下：

$$d_P=\operatorname{argmax}_i\left\lceil\log_2\left(\frac{Area(V(t)\times p_s\bigcap V(k)\times p)}{Area(V(P')\times p)}\right)\right\rceil \tag{4}$$

式中：$V(t)$ 为 t 所属的特征块的最短距离函数；$Area()$ 为特征块的面积值；$p_s=\{e_i\mid 1\leqslant i\leqslant m\}$，$e$ 表示周围邻居边，m 表示周围边的数量；$p=\sum_{i=1}^{m}u_ip_s$，且 $\sum_{i=1}^{m}u_i=1$，$\boldsymbol{U}=\{u_1,u_2,\cdots,u_m\}$ 表示特征块的三维变化矩阵，如图 8 所示[10,11]。

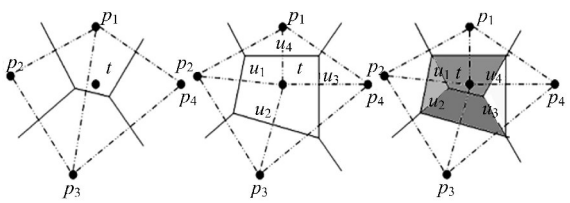


图 8　特征单元变换过程

Fig.8　Feature unit space transformation

2.3　动态自适应特征细分

本文算法通过对每个不规则块单独分配不同的细分深度来实现动态细分过程，进而减少细分产生的计算量。算法流程图如图9所示，包含3个模块：①FPU-特征块处理单元，用于计算块的细分层次，并构建细分表；②由 Catmall-Clark 细分模板，对曲面进行细分，生成关键点；③使用 GPU 绘制渲染表中的不同属性的块。

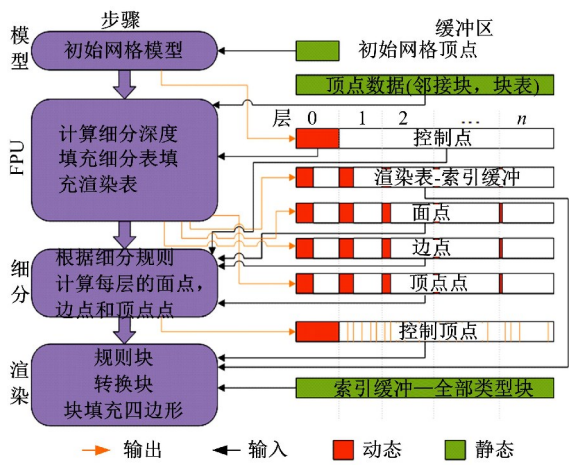


图 9　本文算法框图

Fig.9　Block diagram of proposed algorithm

第一阶段：特征块处理单元。由式(1)得到细分因子的确切值，对 GPU 的处理单元进行编码，自动生成不同特征块的各自细分层次。由初始网格模型计算得到的控制顶点数量作为输入数据的索引信息传送给 FPU，FPU 经过本文算法管线后得到 3 张表，分别为细分层次表、细分表和块渲染表，执行 FPU 的关键代码，如下所示：

```
//每个顶点对应一个数据流
  gpuk(unit Tid){   //数据流
    //顶点的邻接顶点
    VertexData& v=getVertexData(Tid);
    //确定块细分因子和细分深度
    computeSubdDepth(v);
    if(v.valence=4){//规则块
      //填加到渲染表;不需要细分
      append_regular_point(v);
      }else{
      for(unit patch=0;patch<v.valence;patch++){
        for(unit level=0;level<v.d_v;level++){
          //填加点到细分表
          append_face_points(v,patch,level);
          append_edge_points(v,patch,level);
          append_vert_points(v,patch,level);
          //填加块到渲染表
          if(level<v.d_v-1){
            //两个转换块
            append_transit_patches(v,patch,level);
            //一个规则块
            append_regular_patches(v,patch,level);
          }else{
            //最后一层细分的三个规则块
            append_regular_patches(v.patch.level);
            //最后一层细分的一个块填充四边形
            append_irregular_patches(v.patch.level);
}}}}};
```

本文的 FPU 计算得到细分表，该细分表用于存储即将进入 GPU 的细分顶点，同时存放这些顶点的特征图，依据不同的顶点输入将细分表分层 3 个表结构：面点区、边点区和顶点点区，如图 5 所示。FPU 同时构建渲染表，依据渲染块的类型将其分为 3 个表：规则表、不规则表和转换表。FPU 依据细分表生成的细分深度 l 、块的细分因

子 t 和 GPU 缓冲信息绘制索引缓冲区的内容。

第二阶段:细分。算法首先对初始网格进行全局扫描,计算得到全部需要细分的区域,然后使用 Catmull-Clark 细分模板和 GPU 块处理单元对这些细分数据进行处理,得到每个细分层级上的顶点信息。本文算法对不规则区域周围的奇异点进行自适应细分,得到不同拓扑结构的网络表面数据,并将这些数据传输给 GPU 进行二次绘制渲染。

第三阶段:块绘制。使用改进的细分因子计算方法得到不同细分层次上的深度数据,实现对不规则区域的自适应细分。而对于规则块和细分块,本文采用 FPU 渲染管线来处理。GPU 根据渲染表和细分表中的数据,对处理单元中的信息进行并行的光栅化操作,渲染输出所有细分后生成的块数据,实现对不规则区域的实时细分,提高块的绘制精度[12,13]。

3 实验分析

3.1 实验的软件与硬件环境

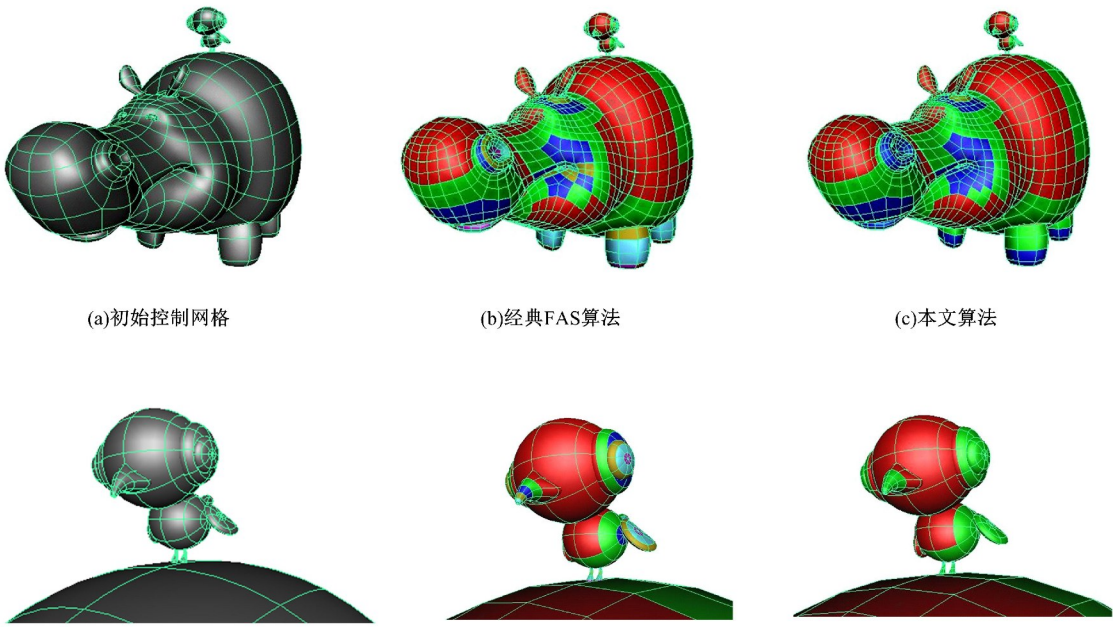
操作系统为 Windows 8,开发语言为 C++,开发工具为 Microsoft Visual C++ 6.0,采用高级绘制语言,三维图形编程接口 Direct3D。CPU 类型/主频为 INTEL Xeon E5/3.0 GHz,内存为 8 Gbit。采用图形处理器 NVIDIA GTX 980,该图形处理器有 11 条像素管线,支持 OpenGL4.4 /DirectX12。

3.2 性能分析

3.2.1 精确性(显示精度)比较

建立初始控制网格(见图 10(a)),控制网格由四边形网格组成,具有明显折痕和尖点等不规则特性。图 10(b)为采用经典 FAS 方法对初始控制网格细分绘制之后的效果图,图 10(c)为采用本文算法对初始控制网格绘制之后的效果图。图 10(d)、(e)、(f)分别是对前 3 个效果图取一个不规则区域(河马背上的小鸟)放大之后的效果图。由图 10(e)、(f)的局部放大效果图可以看出,本文算法显示精确性与 FAS 方法相近,无法用肉眼辨别出不同。传统 FAS 方法使用相同的细分层次对表面模型的不规则区域进行细分,渲染输出的精度较高,存在的弊端是对应每个块计算相同的细分深度会增加 GPU 的计算时间,本文算法通过对经典 FAS 算法进行扩充,对奇异点周围的不规则区域进行了不同层次的细分,能够较完整地构建曲面模型的表面拓扑结构,同时降低了块的细分和渲染时间,GPU 的渲染精度也更加逼真。

由图 10(e)、(f)的局部放大效果图可以看出,本文算法显示精确性与 FAS 方法相近,无法用肉眼辨别出不同。经典的FAS算法对初始控制网



(a)初始控制网格 (b)经典FAS算法 (c)本文算法

(d)局部放大后初始控制网格 (e)经典FAS算法 (f)本文算法

图 10 本文算法与经典 FAS 算法的比较

Fig.10 Proposed method compared with classical method

格的不规则块使用统一的细分深度,绘制效果好,但块计算量较大,本文算法通过对经典 FAS 算法进行扩充,对不同区域的不规则块使用不同的细分深度,较好地保留了曲面模型的折痕等局部特征,块计算量较小,绘制效果逼真。

3.2.2　实时性(绘制速度)比较

分别采用 FAS 算法和本文算法对图 11 中的模型进行细分绘制绘制,并根据不规则块的数量不同来比较两种算法的绘制速度和细分速度,如表 1 所示。

(a)鳄鱼模型初始网格　(b)猩猩模型初始网格　(c)考拉模型初始网格

(d)绘制后的鳄鱼模型　(e)绘制后的猩猩模型　(f)绘制后的考拉模型

图 11　复杂度不同的 3 种模型初始控制网格

Fig.11　Primitive grid of three different models

表 1　两种算法绘制曲面模型速度的对比

Table 1　Speed comparison of two algorithm with different models

曲面模型	河马模型		鳄鱼模型		猩猩模型		考拉模型	
	FAS 算法	本文算法	FAS 算法	本文算法	FAS 算法	本文算法	FAS 算法	本文算法
初始网格块数量	1882	1882	1550	1550	2260	2260	1310	1310
不规则块数量	872	872	664	664	902	902	742	742
不规则点数量	480	480	244	244	398	398	272	272
绘制时间/ms	0.620	0.340	0.498	0.301	0.650	0.389	0.499	0.220
细分时间/ms	0.128	0.112	0.272	0.155	0.162	0.159	0.152	0.099
总绘制时间/ms	0.748	0.452	0.770	0.456	0.812	0.548	0.651	0.319
时间差/ms	0.296		0.314		0.264		0.332	

分析表 1 中数据可以得出如下结论:

(1)与 FAS 方法不同,本文算法的细分阶段包含了 FPU 的执行时间,尽管如此,本文算法的细分速度仍比 FAS 算法快。

(2)选取 4 个具有不同复杂程度的初始网格模型,计算 FAS 算法与本文算法的速度差,曲面模型包含不规则块或点越多,本文算法绘制细分曲面模型的速度越快。

3.2.3　绘制性能比较

采用本文算法与 FAS 算法在大规模地形场景上进行绘制性能比较,模型的四边形面片数量为 805 293 个(见图 12)。在较密集特征块下绘制同一种模型,本文算法绘制速度较快,即当细分密度越是增大时,本文算法绘制速度增快越明显。图 13 给出了两种算法绘制该场景所需的总体绘制时间比较。本文算法能够在场景网格上产生均

匀的细分密度,适用于大规模场景的实时绘制。

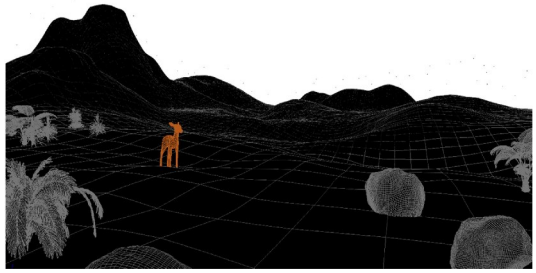


图12 本文算法对大规模地形场景的绘制

Fig.12 Proposed algorithm applied to the rendering of large-scale terrain scene

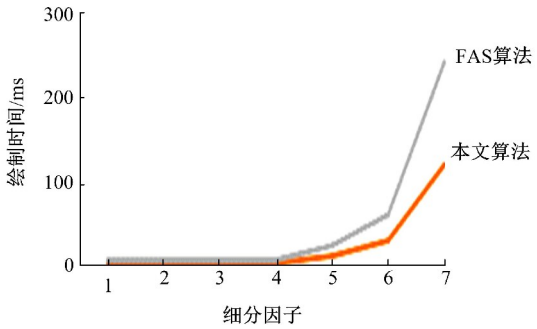


图13 两种算法绘制大规模场景模型时间对比

Fig.13 Time comparison of two algorithms for rendering large-scale scene

4 结束语

提出了一种基于GPU的动态自适应特征细分方法,能够对表面特征较明显的三维Catmull-Clark细分曲面模型进行实时绘制,并且通过引入新的细分因子计算方法,构建特征块处理单元来改进FAS方法的过度细分问题。实验生成的三维曲面模型显示分辨率良好,绘制速度与传统FAS方法相比提高了28%,能够满足三维虚拟环境的实时绘制需求。本文方法还可以应用于其他虚拟环境中的三维曲面模型的建立,例如Loop曲面模型的动态自适应绘制,对于用一般方法难于模拟的具有多不规则表面特征的曲面模型有较好的绘制实时性。

参考文献:

[1] Niessner M, Loop C, Meyer M. Feature-adaptive GPU rendering of Catmull-Clark subdivision surfaces [J]. ACM Transactions on Graphics, 2012, 31(1): 1-11.

[2] Doo D, Sabin M. Behaviour of recursive division surface near extraordinary points [J]. Computer-Aided Design, 1978, 10(6): 356-360.

[3] Loop C, Schaefer S. Approximating Catmull-Clark subdivision surfaces with bicubic patches [J]. ACM Transactions on Graphics, 2008, 27(1): 1-11.

[4] Stam J. Exact evaluation of Catmull-Clark subdivision surfaces at arbitrary parameter values [C]// Proceedings of SIGGRAPH, New York, NY, USA, 1998: 395-404.

[5] Niessner M, Loop C, Greiner G. Efficient evaluation of semi-smooth creases in Catmull-Clark subdivision surfaces [C]// Eurographics Proceedings, Cagliari, 2012: 41-44.

[6] Catmull E, Clark J. Recursively generated B-spline surfaces on arbitrary topological meshes [J]. Computer-aided Design, 1978, 10(6): 350-355.

[7] Niesser M, Loop C. Analytic displacement mapping using hardware tessellation [J]. ACM Transactions on Graphics, 2013, 32(3): 1-9.

[8] Schafer H, Niessner M, Keinert B. State of the art report on real-time rendering with hardware tessellation [C] // Eurographics, Strasbourg, France, 2014: 93-117.

[9] 黄韵岑, 冯洁青. 表驱动的细分曲面GPU绘制[J]. 计算机辅助设计与图形学学报, 2014, 26(10): 1567-1575.
Huang Yun-cen, Feng Jie-qing. Mapping driving subdivision surface upon GPU rendering [J]. Journal of Computer-Aided Design & Computer Graphics, 2014, 26(10): 1567-1575.

[10] Deng C, Ma W. A unified interpolatory subdivision scheme for quadrilateral meshes [J]. ACM Transactions on Graphics, 2013, 32(3): 1-10.

[11] 李桂清. 细分曲面造型及应用[D]. 北京: 中国科学院计算技术研究所, 2001.
Li Gui-qing. The modeling and application method to subdivision surface [D]. Beijing: Institute of Computer Technology, Chinese Academy of Sciences, 2001.

[12] Schafer H, Keinert B, Niessner M. Local painting and deformation of meshes on the GPU [J]. Computer Graphics Forum, 2015, 34(1): 26-35.

[13] Yuan Ya-zhen, Wang Rui, Huang Jin. Simplified and tessellated mesh for realtime high quality rendering [J]. Computer and Graphics, 2016, 54(10): 135-144.