

基于 GPU 的实时球面化算法

黄建彪 陈国华* 张爱军 周厉颖
(北京化工大学 机电工程学院, 北京 100029)

摘要: 分析了球面映射算法速度过慢的原因, 针对传统插值计算中普遍存在的速度与精度相互制约的问题, 改进了现有的基于立体投影的半球面纹理映射模型, 提出了基于 GPU 的球面化算法, 使用 CUDA 并行编程实现双线性插值的并行计算。球面化实验表明该算法在保证输出精度的前提下, 可获得 10 倍左右的加速比, 显著提高了计算速度, 可用于实时性较高的应用场合。

关键词: 实时球面化; 半球面纹理映射; 立体投影; 并行计算

中图分类号: TP391.4

引言

纹理映射是真实感图形绘制的一种重要技术, 用于快速生成物体表面纹理细节, 可显著增强真实感。球面化就是将一幅图形处理成有球面立体感的图形, 广泛应用于视频软件及图像处理软件中。

Bier 等^[1]首次提出了两步纹理映射, 将纹理空间到景物空间的映射分解为两个简单映射的复合。由于球面是不可展曲面, 整球面映射算法会在两极点处出现纹理汇聚现象, 从而产生较大的纹理变形。为减少纹理变形, Bier 等提出了球面映射中纹理不变形的 3 个判断准则, 同时提出了基于立体投影的半球面纹理映射算法, 但这种算法只满足点的相邻性准则。江巨浪等^[2-4]基于球冠映射法按面积比相等构造映射函数得到的基于等面积比的映射算法, 满足了点的相邻性和面积比不变性, 改善了纹理映射的质量。唐永等^[5]提出了基于等长宽比的映射算法, 应用等长宽比约束使纹理平面上的圆曲四边形映射到球面对应的球曲四边形的过程中保持一定比例, 该算法显著地提高了纹理映射的质量。

复杂度较低的基于立体投影的球面化算法在主流 CPU 上处理耗时量大, 单纯通过提升 CPU 主频来提升算法的执行速度已不能满足实时性需求。随着 GPU (graphics processing unit) 技术的迅速发展,

GPU 性能的提升速度大大超过了摩尔定律, 主流 GPU 单精度浮点处理能力已达到同时期 CPU 的 10 倍左右^[6]。本文首先分析了现有的半球面映射算法处理速度过慢的主要原因, 并针对如何提高运算速度, 提出了基于 GPU 的球面化算法, 从两个方面对现有算法进行了改进。

1 现有的半球面纹理映射分析

1.1 映射模型

设球面上点 $P(x, y, z)$, 纹理平面上点为 $Q(u, v)$, 球面映射就是建立 P, Q 两点之间的映射关系。基于立体投影的半球面映射^[1]模型如图 1 所示。

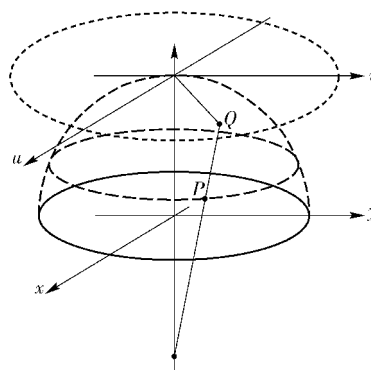


图 1 基于立体投影的半球面纹理映射

Fig. 1 Stereographic projection-based hemisphere texture mapping

基于立体投影的半球面映射关系为

$$\begin{bmatrix} u \\ v \end{bmatrix} = r \begin{bmatrix} \cos \theta \\ \sin \theta \end{bmatrix} = \frac{d+1}{d+z} \begin{bmatrix} x \\ y \end{bmatrix} \quad (1)$$

式(1)中, z 为 P 点到赤道平面的距离, d 为投影中心对应 Z 坐标的相反数, x 与 y 表示输出图像的坐

收稿日期: 2012-06-20

第一作者: 男, 1986 年生, 硕士生

* 通讯联系人

E-mail: cghboy@163.com

标,即球面上点的坐标, u 与 v 表示输入图像坐标。

基于等面积比的半球面映射^[2-3]计算公式为

$$\begin{bmatrix} u \\ v \end{bmatrix} = \frac{1}{\sqrt{1 - \sin\varphi}} \frac{1}{\sqrt{1 + z}} \begin{bmatrix} x \\ y \end{bmatrix} \quad (2)$$

式(2)中, φ 为球面上点 P 的纬度。

基于等长宽比的半球纹理映射^[5]的关系为

$$\begin{bmatrix} u \\ v \end{bmatrix} = \frac{\cos\beta}{1 - \sin\beta} \frac{1}{1 + z} \begin{bmatrix} x \\ y \end{bmatrix} \quad (3)$$

式(3)中, β 表示球冠的最大仰角。

以上 3 种模型的共同点是纹理使用极坐标表示,半径相同的点映射到球面上构成一条纬线,极角相同的点映射到球面上构成一条经线。

1.2 现有算法的不足

无论使用哪一种映射模型,在应用向后映射将输出图像像素逐个映射到输入图像中时,会不可避免地映射到纹理图像中的非整数点,为了提高输出精度需要对图像进行插值处理。

双线性插值算法^[7]具有比最近邻点法更好的连续性,但计算速度较慢。为提高双线性插值的计算速度,可以使用离散化双线性插值算法^[8]。离散化双线性插值算法的实质是将一个像素划分 9 个子像素,用这 9 个子像素替代落在此像素区域内的点。使用立体投影模型对两种插值法进行测试,结果如图 2~4 所示。

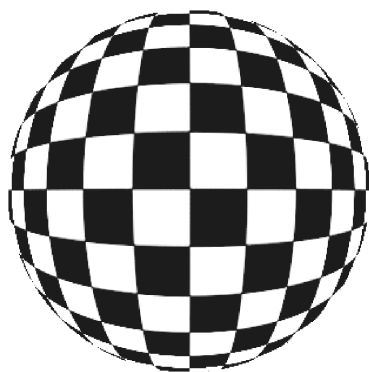


图 2 双线性插值结果

Fig. 2 Result of bilinear interpolation

使用离散化算法的球面化,其输出图像质量欠佳,出现明显的锯齿形(如图 3 所示)。由图 4 可知其执行速度约两倍于使用双线性插值的球面化,表明插值算法对球面化速度起决定作用,即在整个球面化处理过程中插值计算占用了大部分时间,直接影响球面化速度,插值速度过慢是球面化速过慢的主要原因。

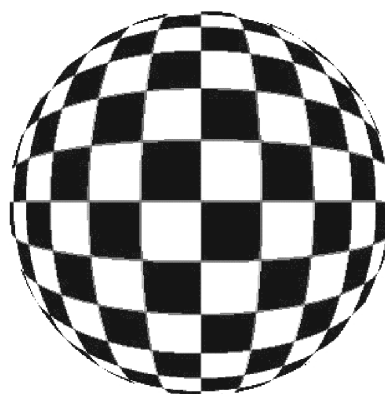
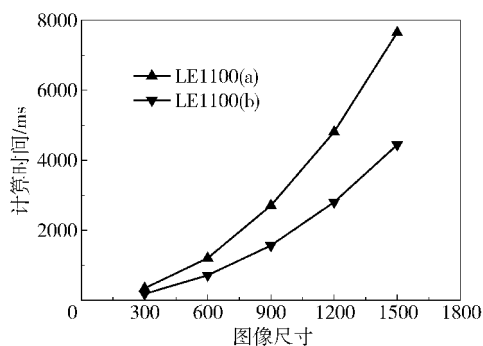


图 3 快速离散化双线性插值结果

Fig. 3 Result of fast discrete bilinear interpolation



LE100(a)—使用传统双线性插值; LE100(b)—使用快速离散化双线性插值

图 4 两种插值算法的对比

Fig. 4 Comparison of two interpolation algorithms

2 基于 GPU 的球面化算法的改进

2.1 算法的提出

提高计算速度通常有两种方法,一是提高 CPU 的主频,二是使用并行算法。CPU 靠提高工作频率及增加指令级并行来提高单个核心的性能,随着工艺不断提高,晶体管的尺寸小到接近原子的数量级,漏电流及发热能问题越加显著,因此处理器的频率提升速度越来越慢,成本也比较高^[6]。

图 5 为在 4 款不同 CPU 的主机上对同一图片处理 20 次得到的结果,随着图像尺寸的增大,处理时间急剧增长,由图 5 可以看出使用主频更高的 CPU 虽然获得了速度的提升,但处理速度仍然较慢不能满足实时性要求。

并行计算已经成为突破摩尔定理局限的重要研究方向,早期的 GPGPU 开发难度大、成本高。CUDA 架构的出现使得 GPU 并行编程变得简单,只要掌握 C 语言就可以实现 GPU 编程^[9]。因此本文提

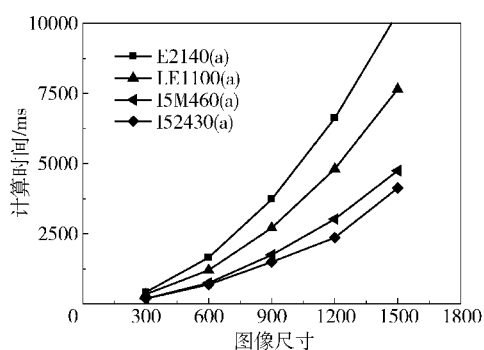


图 5 不同 CPU 下的双线性插值

Fig. 5 Bilinear interpolation with different CPU

出一种基于 GPU 的实时球面化算法,使用 CUDA 对 GPU 进行编程。理论上,它的计算速度取决于单个像素处理的最长时间及内存复制的时间。在使用 CUDA 调用 GPU 进行计算的过程中,内存复制占据了大部分时间,因此在保证输出精度的前提下应尽可能地减小图像尺寸。

2.2 基于 GPU 的球面化算法的实现

传统的基于 CPU 的球面化算法流程如图 6 所示。基于 GPU 的球面化算法的基本思想是:在球面映射过程中,将 CPU 上循环执行的插值运算转交给 GPU 并行处理,在 GPU 上为每个像素分配一个单独线程,每个线程只处理对应像素的插值运算,计算完后将结果返回给 CPU 进行其他处理。

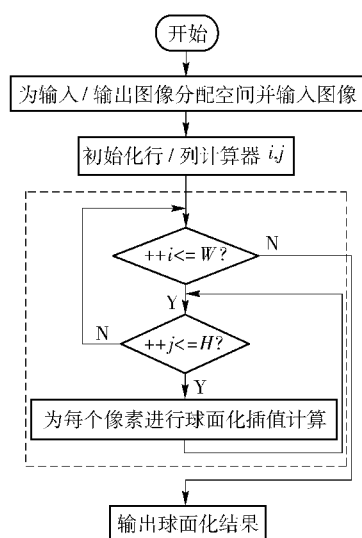


图 6 基于 CPU 的球面化算法

Fig. 6 CPU-based spherizing algorithm

将 CPU 单线程的两层嵌套循环串行部分(图 6 虚线框)交给 GPU 内核(图 7 虚线框)进行多线程并行处理,内核函数只对一个像素进行处理,减少插值计算的时间。

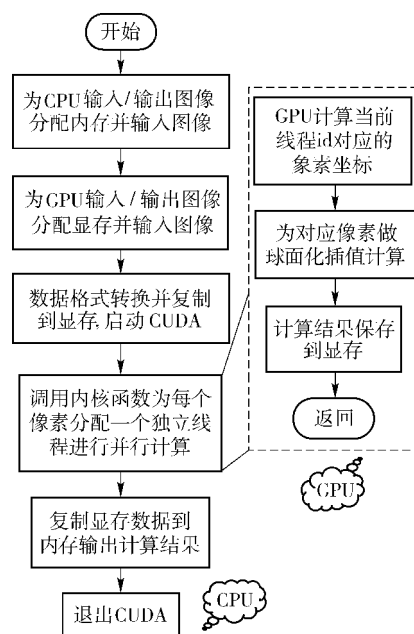


图 7 基于 GPU 的球面化算法

Fig. 7 GPU-based spherizing algorithm

基于 GPU 的球面化算法分为两部分,一部分为主机代码(图 7 的左半部分),另一部分为设备代码(图 7 的右半部分)。设备端主要完成对每个线程对应像素球面化插值计算,主机端口主要完成数据传输等串行工作(如图 7 右串行部分)。

具体实现过程如下。

步骤一 启动 CUDA;

步骤二 在 CPU 端为输入、输出图像分配空间;

步骤三 将 RGB 图像数据打包成 uchar4 类型的二维数组;

步骤四 为 GPU 分配显存,并将内存中的二维数组复制到显存中;

步骤五 为 GPU 分配显存,用于存放输出数据;

步骤六 调用 kernel 内核函数进行并行计算,并将结果写入显存指定位置;

步骤七 将显存中的计算结果读回内存;

步骤八 在 CPU 端将 uchar4 的二维数组转换成图像并显示;

步骤九 释放内存和显存空间,并退出 CUDA。

2.3 改进的半球纹理映射模型

球面化的速度主要取决于图像尺寸和插值算法速度,由于式(1)中的系数大于 1,纹理图像大于球面化输出图像,内存占用量较大,内存复制时间较

长。因此对模型进行改进,将纹理平面置于赤道平面上,以减小纹理图像尺寸,如图 8 所示。

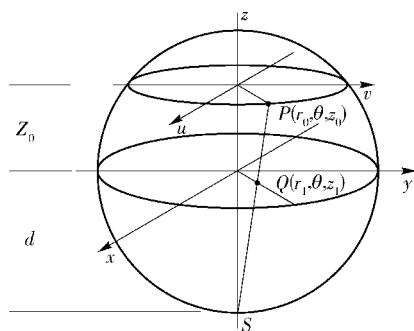


图 8 改进的立体投影模型

Fig. 8 Improved stereographic projection model

设球面半径为 R , 投影中心到原始图像的距离为 d , 球面上的点 $P(r_0, \theta, z_0)$ 对应图像上的映射点为 $Q(r_1, \theta, z_1)$, 其中 z_1 表示纹理图像在坐标系中的高度, $z_1 = 0$ 即表示将输入图像置于赤道平面。 PQ 两点在直角坐标系下的可表示为 $P(x_0, y_0, z_0)$, $Q(x_1, y_1, 0)$ 。其中,

$$z_0 = \sqrt{R^2 - x_0^2 - y_0^2} \quad (4)$$

不同坐标系之间的转换关系为

$$\begin{bmatrix} x_i \\ y_i \end{bmatrix} = r_i \begin{bmatrix} \cos \theta \\ \sin \theta \end{bmatrix}, i = 0, 1 \quad (5)$$

根据三角形的相似性可得

$$\frac{d + z_0}{d} = \frac{r_0}{r_1} \quad (6)$$

联立式(5)~(6)得到 P 、 Q 两点的映射关系

$$\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \frac{dr_0}{d + z_0} \begin{bmatrix} \cos \theta \\ \sin \theta \end{bmatrix} = \frac{d}{d + z_0} \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} \quad (7)$$

特别地,当光源位于负极点,即 $d = R$ 时,有

$$\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \frac{R}{R + z_0} \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} \quad (8)$$

一般地,令 $R = 1$, 球面坐标用 x, y, z 表示,纹理坐标用 u, v 表示,则有

$$\begin{bmatrix} u \\ v \end{bmatrix} = \frac{1}{1 + z} \begin{bmatrix} x \\ y \end{bmatrix} \quad (9)$$

对比式(3)和式(9)可以看出,当式(3)中 $\beta = \pi/2$ 时,与式(9)有相同结果,说明改进的模型与基于等长宽比的球面化模型有相同的优点,即在满足点的相邻性和长宽比不变性的情况下近似地满足了面积比不变性。在这 4 个模型的计算公式中,式(9)的计算量相对较小,表明将纹理图像置于赤道

平面并将投影中心放到负极点的投影映射能取得较好的纹理映射效果,速度也较快。

2.4 算法的特点

CUDA 编程模型将 CPU 作为主机 (HOST), GPU 作为协处理器或设备。运行在 GPU 上的 CUDA 并行计算函数称为 kernel, 即内核函数。CPU 串行代码完成 kernel 启动前的数据准备和设备初始化工作,当程序中有大量并行计算时,则将这些并行计算交给 GPU。

GPU 运算能力远远超越 CPU, GPU 提供单指令多数据类型处理,适合于数据并行计算。但其条件控制能力非常弱,若程序中大量使用条件分支则会极大影响执行效率。因此,应尽量减少条件分支,将部分条件控制语句用计算来代替。

基于 GPU 的球面化算法利用并行双线性插值算法代替传统双线性插值,提升了球面化的速度,在现有的 3 种模型中均可提高球面化速度。同时,改进的半球纹理映射模型,将纹理平面移到赤道平面,使所需的输入图像减小,缩短了 CUDA 内存复制的时间,并降低了 CPU 上的内存开销,因此取得了较好的处理效果。

3 球面化实验

为了避免将 CUDA 初始化时间计入测试时间,在开始测时之前,对数据进行一次完整的输入输出操作,令 GPU 从节能模式直接进入工作状态,使测试结果更可靠。分别在硬件配置如表 1 所示的主机上对改进后的模型进行测试,图 9 是基于 GPU 的实时球面化映射的效果图,图 10 是 CPU/GPU 球面化速度对比,加速比如表 2 所示。

表 1 测试主机的配置

Table 1 Configuration of test computers

主机	CPU	主频/GHz	GPU	显存/MB
1	IE1100	1.9	GT210	512
2	I52430	2.4	GT540	1024

分析图 9~10 和表 2 可以得知:

(1) 基于 GPU 的实时球面化算法速度明显快于使用 CPU 处理的速度,加速比接近于 10;

(2) 基于 GPU 的实现球面化算法在低主频 CPU 下也可以取得较好性能,主机 1 上使用 GPU 的计算速度((LE1100(c)))超过了主流 CPU 上使用传统插值(I52430(a))和快速插值(I52430(b));

表 2 基于 GPU 的实时球面化算法的加速比

Table 2 Speedup of the GPU-based real-time spherizing algorithm

图像尺寸	加速比	
	主机 1	主机 2
300 × 300	8.32	7.40
600 × 600	8.49	9.20
900 × 900	9.30	9.07
1200 × 1200	9.78	8.20
1500 × 1500	10.00	9.14

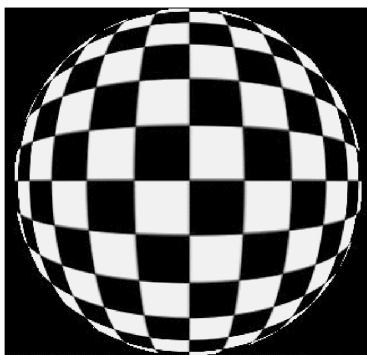
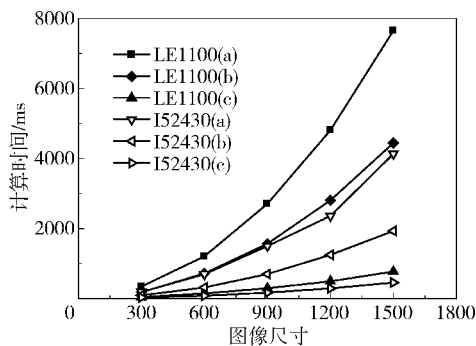


图 9 基于 GPU 的实时球面化结果

Fig. 9 Result of GPU-based real-time spherizing algorithm



(a) 代表使用传统双线性插值; (b) 代表使用快速离散化双线性插值; (c) 代表使用基于 GPU 的球面化算法

图 10 CPU/GPU 球面化速度对比

Fig. 10 Speed comparison of CPU-based spherizing and GPU-based spherizing

(3) 基于 GPU 的球面化算法受图像尺寸的影响明显小于传统插值算法,且图像尺寸越大越能显示出算法的优越性;

(4) 对整个半球进行映射时,立体投影模型所需图像大小为 1200×1200 ,而改进的模型所需图像尺寸为 600×600 ,对比 I52430(c) 与 LE1100(c) 中对应的数据,可以看出图像大小为 600×600 的计算速度明显更快。

这是由于在 CUDA 计算中内存复制时间占主要时间所造成的,图像越大内存复制时间越长,改进模型减小了图像尺寸,有效地减少内存复制时间,保证了球面化处理的实时性。

4 结束语

本文针对传统插值速度过慢,提出了基于 GPU 的球面化算法,改进了基于立体投影法的半球面纹理映射模型,将纹理平面移动到赤道平面,使得模型在满足点的相邻性和长宽比不变性的同时近似地满足了面积比不变性准则,同时降低了模型计算的复杂度,减小了 CUDA 内存复制的时间,提升了球面化速度。在 CPU 主频较低的主机上,使用基于 GPU 的球面化算法,可获得 10 倍左右的加速比,极大地节约了计算时间,甚至优于主流 CPU 上传统算法的计算时间,提升了球面化的运行速度,利用人眼的“视觉暂留”特性,使球面化算法的输出画面更具有流畅感。该算法适合于实时性要求较高的应用场合。

参考文献:

- [1] Bier E A, Sloan Jr K R. Two-part texture mappings[J]. IEEE Computer Graphics and Application, 1986, 6(9): 40-53.
- [2] 江巨浪, 张佑生. 一种应用面积等比约束的半球面纹理映射算法[J]. 系统仿真学报, 2004, 16(9): 1982-1984.
Jiang J L, Zhang Y S. Algorithm of hemisphere texture mapping applying constraint of equal ratio of areas[J]. Journal of System Simulation, 2004, 16(9): 1982-1984. (in Chinese)
- [3] 江巨浪, 张佑生. 一种适用于球面局部区域的纹理映射算法[J]. 中国图象图形学报, 2004, 9(9): 1113-1116.
Jiang J L, Zhang Y S. An algorithm of texture mapping applicable to partial area on spheres[J]. Journal of Image and Graphics, 2004, 9(9): 1113-1116. (in Chinese)
- [4] 刘晓梅, 秦文虎, 赵正旭. 一种基于面积比不变性约束的曲面纹理算法[J]. 计算机技术与发展, 2006, 16(9): 1-3.
Liu X M, Qin W H, Zhao Z X. An algorithm of surface texture mapping based on invariability of ratio of areas[J]. Computer Technology and Development, 2006, 16(9): 1-3. (in Chinese)
- [5] 唐勇, 刘连军, 吕梦雅. 应用长宽等比约束的半球面

- 纹理映射算法[J]. 系统仿真学报, 2007, 19(18): 4209-4211.
- Tang Y, Liu L J, Lv M Y. Algorithm of hemisphere texture mapping applying constraint of equal ratio of length and breadth[J]. Journal of System Simulation, 2007, 19(18): 4209-4211. (in Chinese)
- [6] 张舒, 褚艳利. GPU 高性能运算之 CUDA [M]. 北京: 中国水利水电出版社, 2009: 1-13.
- Zhang S, Chu Y L. High-performance computing of GPU with CUDA [M]. Beijing: China Water Power Press, 2009: 1-13. (in Chinese)
- [7] 陈宝平, 赵俊岚, 尹志凌. 双线性插值算法的一种快速实现方式[J]. 北京电子科技学院学报, 2004, 12(4): 21-23.
- Chen B P, Zhao J L, Yin Z L. A fast approach to realize bilinear interpolation algorithm[J]. Journal of Beijing Electronic Science and Technology Institute, 2004, 12(4): 21-23. (in Chinese)
- [8] 陈良, 高成敏. 快速离散化双线性插值算法[J]. 计算机工程与设计, 2007, 28(15): 3787-3790.
- Chen L, Gao C G. Fast discrete bilinear interpolation algorithm[J]. Computer Engineering and Design, 2007, 28(15): 3787-3790. (in Chinese)
- [9] 田立国, 杜君. 基于 GPU 通用计算的分析与研究[J]. 制造业自动化, 2011, 33(1): 173-175.
- Tian L G, Du J. General computing of gpu: A new high-performance computing solutions[J]. Manufacturing Automation, 2011, 33(1): 173-175. (in Chinese)

A graphics processing unit (GPU)-based real-time spherizing algorithm

HUANG JianBiao CHEN GuoHua ZHANG AiJun ZHOU LiYing

(College of Mechanical and Electrical Engineering, Beijing University of Chemical Technology, Beijing 100029, China)

Abstract: The cause of the low speed of a sphere mapping algorithm has been analyzed. In order to reduce the interaction between speed and accuracy, which is common in traditional interpolation methods with existing sphere mapping algorithms, an improved hemisphere texture mapping model based on stereoscopic projection has been proposed, and a graphics processing unit (GPU)-based spherizing algorithm has been put forward, in which CUDA parallel programming was utilized to complete the parallel computing of bilinear interpolation. The experiments showed that the computing speed could be significantly increased with the new method, whilst ensuring output accuracy. The method gave a speedup factor of almost 10, and it could be employed in fast real-time applications.

Key words: real-time spherizing; hemisphere texture mapping; stereoscopic projection; parallel computing