CUDA 加速分形火焰绘制 *

刘进锋
（宁夏大学 数学计算机学院，宁夏 银川 750021）

摘 要： 提出了一种基于 CUDA 的并行分形火焰绘制算法，该算法利用了 GPU 的单指令多线程的特点，将常用于迭代函数系统（IFS）的传统的混沌游戏（Chaos Game）算法作了并行化修改，并在图形输出时利用了 CUDA 与 OpenGL 互操作加速分形火焰绘制。实验证明，该并行方法比 CPU 上运行的普通算法快了 15 倍左右，能够实时绘制分形火焰图形。在上述基本算法的基础上，又进一步研究了消除分支分歧的改进算法，改进算法的运行时间具有相对于变换函数数量的恒定性，多数情况下比基本算法性能更优越。

关键词： 分形火焰；迭代函数系统；统一计算设备架构；图形处理器
中图分类号： TP391 **文献标识码：** A **文章编号：** 1674-7720(2013)23-0037-04

CUDA accelerated fractal flame rending

Liu Jinfeng
(School of Mathematics and Computer, Ningxia University, Yinchuan 750021, China)

Abstract: In this paper, a parallelized approach is proposed which exploits the SIMT (single-instruction multiple-thread) architecture of GPU to change the classical Chaos Game algorithm for IFS to a parallelized way, and utilizes CUDA and OpenGL interoperability to accelerate the fractal flame rendering. Experiments show that the parallelized approach can achieve about 15 times of speedup over the CPU counterpart, and can render fractal flame images in real-time. On the basis of the algorithm, an improved algorithm to eliminate branch divergence is further studied. The improved algorithm has constancy to the number of variation functions, and has a better performance than the basic algorithm in most cases.

Key words: fractal flame; IFS; CUDA; GPU

分形通常被定义为“一个粗糙或零碎的几何形状，可以分成数个部分，且每一部分都（至少近似地）是整体缩小后的形状”，即具有自相似的性质。分形一词于 1975 年由曼德博创造出，来自拉丁文“frāctus”，有“零碎”、“破裂”之意。分形有几种类型，可以分别依据表现出的精确自相似性、半自相似性和统计自相似性来定义。虽然分形是一个数学构造，但也可以在自然界中找到。分形在艺术作品、医学、土力学、地震学和技术分析中都有应用。

分形火焰是迭代函数系统 IFS（Iterated Function System）分形的一种更复杂的变种形式，与普通的 IFS 相比，分形火焰能产生变化多端、引人入胜的图形，但与此同时，其计算复杂度很高。

图形处理器（GPU）原本是处理计算机图形的专用设备，近十年来，由于高清晰度复杂图形实时处理的需求，GPU 发展成为高并行度、多线程、多核的处理器。目前，主流 GPU 的运算能力已超过主流通用 CPU，从发展趋势上来看，将来差距会越拉越大。GPU 卓越的性能对开发 GPGPU（使用 GPU 进行通用计算）非常具有吸引力。统一计算设备架构 CUDA（Compute Unified Device Architecture）是 NVIDIA 公司伴随着统一渲染架构而推出的一种通用的 GPU 编程模型，可以将 GPU 视为一个并行数据计算的设备，对所进行的计算进行分配和管理[1]。在 CUDA 的架构中，通用计算不再像过去的GPGPU 那样必须将计算映射到图形 API 中，开发者无需学习复杂的显示芯片的指令或是特殊的结构，CUDA 编程语言只是对标准的 C 语言作了少量扩展，因此开发门槛

* 基金项目：宁夏自然科学基金（NZ12163）

大大降低了。目前有很多基于 CUDA 的 GPU 计算的研究成果，有不少计算问题特别适合这种计算模式。关于 CUDA 的相关应用可参阅参考文献[2]和参考文献[3]。

CUDA 加速的分形绘制的研究并不多，其中 NVIDIA 的 SDK 提供了基于 CUDA 的加速 Mandelbrot、Julia 集生成的实例[4]。其算法的基本过程是平面图的每一个像素对应 CUDA 并行计算中的一个线程，该并行方法能比 CPU 上的普通方法加速几十倍。参考文献[5]描述了基于 CUDA 的并行分形 IFS 点递归绘制算法。

本文的研究是利用 GPU 的计算能力加速分形火焰绘制，使之能更加实用。

1 背景及相关知识

1.1 经典的迭代函数系统

数学中，迭代函数系统是构造分形的一种方法，由此产生的结构是自相似的。IFS 分形可以是任何维度[6]，但通常在二维空间中计算和绘制。分形由一些自身的拷贝联合构成，每个拷贝根据一个函数来作变换。函数通常是收缩的，意思是它们会使图像点更靠近，使形状更小。因此，IFS 分形的形状由一些可能重叠的自身拷贝组成，其中每个拷贝也由自身的一些拷贝组成，如此无限下去。这也是自相似分形特性的来源。

IFS 正式定义：迭代函数系统是一组有限的完备度量空间上的收缩映射。

$\{f_i: X \to X | i=1,2,\cdots,N\}$，$N \in \mathbf{N}$ 是一个迭代的函数系统，如果每个 f_i 对完备度量空间 X 是收缩的。

性质：哈钦森指出，对于度量空间 R^n，这种函数系统有唯一的紧凑固定集 S。一种构建固定集的方法是从一个初始点或集合 S_0 开始，并对 f_i 迭代，S_{n+1} 是图像 S_n 在 f_i 下的并集；取 S 为 S_n 并集的闭包。这唯一的固定集（非空紧缩）$S \subseteq X$ 具有这样的属性：

$$S=\bigcup_{i=1}^{N} f_i(S) \tag{1}$$

f_i 可以是多种函数，但主要使用仿射变换函数。

对于二维空间上的点，仿射变换具有如下的形式：

$$f_i(x,y)=(ax+by+c, dx+ey+f) \tag{2}$$

其中 a、b、c、d、e、f 为放射变换系数。

1.2 分形火焰简介

分形火焰是 1992 年由 DRAVES S 提出的[7]，它可以说是分形迭代函数的一种形式，但有以下 3 方面不同：

（1）迭代时使用非线性函数而不是仿射变换；

（2）色调映射显示是对数密度而不是线性的；

（3）颜色是根据结构（即通过采取的递归路径）决定的，而不是采用单色的或根据点的密度决定。

色调映射和着色旨在显示尽可能多的分形的详细信息，这样通常会产生更美观、更绚丽的图像。

分形火焰的每个函数具有如下的形式：

$$f_i(x,y)=\sum_{V_k \in \text{Variations}} w_k V_k(a_i x+b_i y+c_i, d_i x+e_i y+f_i) \tag{3}$$

其中，w_k 是权重；V_k 函数是一组预定义的非线性变换函数，可以多达 20 多个。

2 通常绘制分形火焰的过程

绘制分形火焰的通常方法包括两个步骤[8]：创建直方图及绘制图像。其中，创建直方图的核心是使用随机迭代算法，也称为混沌游戏[8]。由于分形火焰和常见的 IFS 之间的差异，该算法相比于普通混沌游戏算法有一些变化，创建直方图的伪代码如下：

```
选择初始点的 p(p.x,p.y,p.c);
                    //第 3 个坐标表示该点当前的颜色
For(i=1;i<=max_no_of_iterations;i++)
{//max_no_of_iterations 表示最大迭代次数。产生一个随
                    机概率 pj，选择概率是 pj 的一个 Fi 函数
(p.x,p.y)=fi(p.x,p.y);
p.c=(p.c+(Fi)color)/2;
                    //(Fi)color 是与函数 Fj 关联的颜色
if i>12{    //前十几次迭代的数据不在直方图中统计
histogram_frequency[x][y]++;
histogram_color[x][y]=(histogram_color[x][y]+p.c)/2;
}}
```

绘制图像时为了增加图像的品质，可以使用超级采样来降低噪声。其方法是创建一个比本身图像大的直方图，这样每个像素会从多个数据点采集数据。例如，为了绘制 100×100 像素的图像，创建具有 300×300 个单元的直方图，这样每个像素将使用 3×3 的直方图组来计算其值。对于每个在最终图像中的像素 (x,y)，执行以下计算：

```
frequency[x][y]=周边超采样点 histogram_frequency[x][y]
的平均值;
color[x][y]=周边超采点 histogram_color[x][y]的平均值;
alpha[x][y]=log(frequency[x][y])/log(frequency_max);
                    //frequency_max 为 frequency[x][y]的最大值
final_pixel_color[x][y]=color[x][y]*alpha[x][y]^(1/gamma)
```

上述的算法使用了灰度校正，能使颜色更亮。

3 分形火焰并行算法详细描述

在分形火焰直方图生成阶段，使用串行混沌游戏算法的过程是：选择一个点开始，并根据概率挑选一个函数将该点代入来计算，得到下一个点，然后用新点继续进行下一次迭代，如此不断迭代下去。

本文提出的并行算法是传统混沌游戏算法的扩展：算法开始时选择 n 个而不是一个起始点，参考文献[9]说明了选择 n 个起始点的方法开始迭代和选择 1 个起始点开始迭代得到的结果一致。并行的实现是通过将一个线程分配给某一个起始点，n 个线程同时分别独立执行混沌游戏。通过这种方式，并行算法提高了速度。

当然，上述只是并行算法的主要思想，因为分形火焰比较复杂，具体的算法还有很多细节问题需要考虑。实现主要包括 3 个在 GPU 上并行执行的内核函数：warm_up、iterate_batch 和 output_for_rending。与常见的串行混沌游戏一样，在 warm_up 函数中的每个线程选择一个起始点，迭代十几次，但只保持最后的结果，放弃前面迭代得到的值。warm_up 函数计较简单，没必要描述实现细节。Iterate_batch 函数接收 warm_up 函数生成的点并迭代数十或数百遍，然后计算得到的每个点的直方图。Iterate_batch 函数的核心思想就是 n 个点同时迭代的并行混沌游戏算法，也是生成分形火焰图形的核心。其伪代码如下：

```
//输入：ractalInfo 存储该 fractal flame 信息的结构体；
iterPosStateBuffer 是 warm_up 后存储点位置的数组；iterColorStateBuffer 是 warm_up 后存储颜色的数组；randBuffer 用户指定的每个函数选择的概率
//输出：accumBuffer 存储累计直方图信息，该信息在绘制阶段会用到
Iterate_batch ( )
{lid=threadIdx.x;
gid=(blockIdx.x*blockDim.x+threadIdx.x);
pos=iterPosStateBuffer[gid];
color=iterColorStateBuffer[gid];
for(iter=0;iter<ITERCOUNT;iter++)
{fIndex=chooseRandomBranch(randBuffer+lid,fractalInfo);
                                    //根据 randBuffer 选择一个函数
iterate(&pos,&color,fIndex,fractalInfo);
                                    //迭代生成新的点和颜色
screenPos=getPos(fractalInfo,pos);
        //将计算得到的点的位置转换为屏幕上点的位置
calhistogram(color,screenPos,accumBuffer);
 //计算每个屏幕点的统计直方图和颜色，保存到accumBuffer
}}
```

Output_for_rending 函数主要完成图形绘制功能，它接收从 iterate_batch 传递来的直方图信息，作色调映射和伽玛校正，并将每个像素的最终显示颜色放在 outputBuffer。其伪代码如下：

```
//输入：ractalInfo 存储了该 fractal flame 信息的结构体；
accumBuffer 是从 Iterate_batch 函数传递过来的，保存了直方图信息
//输出：outputBuffer 保存了最终数据，用于通过 OpenGL 绘制最终图形
output_for_rending(ractalInfo,accumBuffer)
{x=(blockIdx.x*blockDim.x+threadIdx.x);
y=(blockIdx.y*blockDim.y+threadIdx.y);
pix=tonemap(fractalInfo,accumBuffer,x,y);
//根据直方图信息，使用色调映射和伽玛校正产生实际
                                    //显示的颜色
setoutput(outputBuffer,x,y,pix);
                    //将每个像素的显示颜色保存到 outputBuffer
}
```

根据实验观察，图像绘制阶段大约消耗总时间的 70%，为了加速绘制过程，需要利用 CUDA 和 Open GL 互操作[10]。互操作的基本方法是将 Open GL 缓冲区映射到 CUDA 的内存空间。CUDA 用于计算和数据生成，Open GL 用来绘制像素或顶点，因为它们共享同一内存空间，无需在 CPU 内存和 GPU 内存间移动数据，所以速度非常快。调用 output_for_rending 函数之前，需要做一些工作使 CUDA 和 Open GL 相关联，并设置 outputBuffer 与 CUDA 和 Open GL 共享。

4 对基本并行算法的改进

分形火焰的基本并行实现是每个线程对应一个数据点，各线程随机选择一个变换函数计算，用计算得到的数据点替换原数据点，这个过程重复 m 次。该并行算法需要每个线程根据随机数选择变换函数，因而会导致 CUDA 严重的分支分歧[11]。变换函数越多，越有可能绘制出更有趣的图像，因此，应该尽可能允许有更多的变换函数。然而用到的变换函数越多，分支分歧问题就越严重。

本文提出一种改进的算法，该算法通过预分类能移除随机选择函数，可以消除分支分歧。具体方法是通过随机数据访问来替代随机选定函数，即每个线程分配一个固定的函数，在每次迭代中随机选择一个数据点。这种选择是通过数据和线程索引之间的一个随机双射函数映射实现的。通过这种方法，指令能够以最佳方式静态地分配给线程，预先计算好固定的置换，它们不依赖于动态数据。每个线程使用分配的函数，通过置换索引间接地访问数据数组，然后将该函数的计算结果写回。为了避免写访问时的条件竞争，要么结果数据写回读取的位置，要么需要第二个数组来存储数据点，每次迭代挑选一个新的置换。

上述的过程需要通过创建多个包含数据点索引及随机数的数组来进行置换，随机数可由 Mersenne Twister 方法生成[11]，数组根据随机数排序。所有的变换函数是在一个大的 switch 语句中实现的。用一个结构描述变化索引及缩放因子，该结构存储在常量内存中。

5 实验结果

上述并行分形火焰算法是在 NVIDIA GeForce GTX9800+ 上实现的，该 GPU 有 128 个频率为 1.836 GHz 的流处理器，分为 16 个 SM，896 MB 显存，装配在 CPU 为 Intel Core 2 Duo E7400 2.8 GHz，内存为 1 GB 的计算机上。为了作对比，同时实现了 CPU 上运行的串行版本。实验结果表明，本文提出的基于 CUDA 的基本并行算

法在大多数情况下比 CPU 上的串行算法快了约 15 倍，能做到分形火焰实时绘制。

实验表明，消除分支分歧的算法与基本并行算法相比，基本并行算法的运行时间与变换函数的数目呈现线性增长关系，而消除分支分歧的改进算法运行时间恒定，多数情况下性能都比基本算法高。在变换函数数目为 20 个时，消除了分支分歧的算法比基本算法快了约两倍。

本文提出了一种并行的分形火焰算法，该方法利用了 GPU 的计算能力及 CUDA 和 Open GL 互操作方式，速度快到足够实时高帧速率绘制。该算法使得在生成分形火焰图形时，能实时更改参数并观察绘制效果，可以有效地帮助加深对迭代函数系统的认识。本文方法对其他图像实时绘制的应用也有很高的参考价值。

参考文献

[1] NVIDIA Corporation. NVIDIA CUDA programming guide version 3.2[EB/OL]. (2011-03). http://developer.nvidia.com/cuda.

[2] Hwu Wenmei, RODRIGUES C, RYOO S, et al. Compute unified device architecture application suitability[J]. Computing in Science and Engineering, 2009,11(3): 16-26.

[3] Hwu Wenmei. GPU computing gems [M]. New York: Morgan Kaufmann,2011.

[4] GRANGER M. Mandelbrot CUDA SDK code samples [EB/OL].http://developer.nvidia.com/cuda.2011-03.

[5] KWANIEWSKI B. CUDA based parallel version of point recursive rendering algorithm of IFS attractor [C]. 12th International PhD Workshop OWD, 2010: 23-26.

[6] WHITELAW M. Metacreation: art and artificial life[M]. MIT Press, 2004.

[7] DRAVES S, RECKASE E. The fractal flame algorithm[EB/OL].http://flam3.com/flame draves.pdf. 2008-11.

[8] NIKIEL S. Iterated function systems for real-time image synthesis[M]. London: Springer, 2007.

[9] DUTIL N. Construction of fractal objects with iterated function systems[EB/OL]. (2000-10).http://www.cs.mcgill.ca/~ndutil/project.pdf .

[10] 刘进锋，郭雷.CUDA 和 OpenGL 互操作的实现及分析[J]. 微型机与应用，2011(23)：40-43.

[11] MATSUMOTO M, NISHIMURA T. Mersenne twister: a 623-dimensionally equidistributed uniform pseudorandom number generator[J]. ACM Transactions on Modeling and Computer Simulation, 1998,8(1):3-30.

（收稿日期：2013-09-03）

作者简介：

刘进锋，男，1971 年生，副教授，主要研究方向：GPU 通用计算，图形学，图像处理。