

一种易于硬件实现的嵌入式 GPU 三角形光栅化算法

刘青楠, 曾泽仓, 杜慧敏, 丁家隆

(西安邮电大学 电子工程学院, 陕西 西安 710121)

摘 要: 在嵌入式 GPU 中, 三角形光栅化的性能是决定图形处理器性能的一个重要因素. 传统的三角形光栅化算法需要处理大量的无关像素, 会降低 GPU 的处理速度. 本文针对采用瓦片渲染的嵌入式 GPU, 提出一种基于瓦片的递归的三角形光栅化算法, 三角形光栅化会引起图形的几何走样, 本文利用超级采样中的反走样算法, 只用加法和移位实现了 RGSS 反走样算法. 用 C++ 语言实现了所提出的三角形光栅化及反走样算法. 并在 Xilinx zc706 开发板上对算法进行了验证. 结果表明, 光栅化算法可完成对任意三角形的渲染, RGSS 反走样算法处理后的三角形抗锯齿效果明显, 该算法易于硬件实现, 适用于采用瓦片渲染的嵌入式 GPU 中.

关键词: 光栅化 边函数 瓦片 像素 反走样

中图分类号: TP391.41

文献标识码: A

文章编号: 1000-7180(2018)09-0026-06

An Embedded GPU Triangle Rasterization Algorithm for Hardware Implementation

LIU Qing-nan, ZENG Ze-cang, DU Hui-min, DING Jia-long

(School of electronic engineering, XI'AN University of Posts & telecommunications, Xi'an, 710121)

Abstract: In Embedded GPU, Triangle rasterization is an important factor in determining the performance of graphics processors. Traditional algorithms generate several redundant pixels and cut down the efficiency of GPU. A recursive triangular rasterization algorithm is proposed for tile-based embedded GPU. In addition, the triangular rasterization will cause the aliasing of the graph. The super-sampling anti-aliasing algorithm, RGSS, is used and implemented only by addition and shift operations. The C++ language is used to implement the triangular rasterization and anti-aliasing algorithm. The results show that the rasterization algorithm can complete the rendering of any triangle, and the triangular anti-aliasing effect of RGSS anti-aliasing algorithm is obvious. Anti-aliasing algorithm can significantly improve the rasterization of the anti-aliasing effect and suitable for using in tiled base rendering embedded GPU.

Key words: rasterization ;edge-function ;tile ;pixel ;anti-aliasing

1 引言

随着嵌入式设备的发展, 嵌入式 GPU 已经成为嵌入式设备中不可缺少的一部分. 与传统的桌面 GPU 采用的立即渲染构架不同, 嵌入式 GPU 一般采用基于瓦片(Tile-Based)架构渲染图形场景. 基于瓦片架构的 GPU 将显示屏幕划分为若干个固定大小被称为瓦片的像素块, 分别对瓦片单独进行

场景绘制渲染, 最后将各瓦片的绘制结果拼接到一起, 得到最终渲染结果^[1,2]. 基于瓦片架构的 GPU 渲染一次不是渲染整个场景, 而是渲染一个瓦片. 因此, 减少了 GPU 同主存之间的通信, 节省了大量的带宽资源和电能消耗, 成为嵌入式 GPU 的主流架构.

在嵌入式 GPU 渲染流水线中, 光栅化是一个重要的环节, 其效率的高低直接影响到 GPU 的性

能^[3,4]. 三角形是图元光栅化的最主要的基本单元,其光栅化包含了三角形的参数建立、属性平面方程的建立、边方程的建立和光栅化控制等. 三角形光栅化是将连续信息表示的三角形转换成离散的片元,由于三角形包含了无限的频率分量,因此,光栅化后,三角形会出现走样现象. 通常,会对光栅化之后的三角形进行反走样处理,以保持图形的真实感. 三角形光栅化及其反走样包含了大量的数据处理,是 GPU 渲染管线中非常耗时的部分,因此,一般采用硬件设计.

目前,最基本的三角形光栅化算法主要有边界扫描算法和边函数法^[5]. 基于边界扫描算法硬件实现,先对边进行光栅化,然后再产生三角形内部的所有像素. 文献[4]中,当一个三角形跨越多个瓦片的时候,需要计算三角形的边与 tile 块的交点,这时会引入除法运算,硬件的实现复杂度增加.

本文提出一种基于瓦片的递归三角形光栅化算法:根据三角形边函数,判定一个瓦片四个顶点是否包含在拟光栅化的三角形中,如果是则输出该瓦片的包含的所有像素,否则对该瓦片进行水平和垂直方向折半细分处理,直到一个瓦片大小包含 2×2 个像素. 算法可以对每个瓦片独立处理,不需要考虑三角形跨越不同瓦片的情况,高度并行,易于硬件实现. 另外,三角形光栅化会引起图形的几何走样,本文只用加法和移位实现了 RGSS 反走样算法.

2 基于瓦片的层次化细分算法

2.1 点与三角形的关系

为了确定三角形内部像素并避免除法运算,文献[6]提出了边函数算法. 如图 1 左图所示,对于任意一个点,它相对于一个直线的位置有三种情况,左边、右边和直线上,通过直线矢量方程,可以判定点 (x, y) 和直线的位置关系. M, N 点的直线矢量 $\overrightarrow{MN}$ 的方程为公式(1):

$$l_{\overrightarrow{MN}}(x, y) = (x - x_M)(y_N - y_M) - (y - y_M)(x_N - x_M) \quad (1)$$

当 $l_{\overrightarrow{MN}}(x, y) > 0$, 点 $S(x, y)$ 在 $\overrightarrow{MN}$ 的左侧; 当 $l_{\overrightarrow{MN}}(x, y) < 0$, 点 $S(x, y)$ 在 $\overrightarrow{MN}$ 右侧; 当 $l_{\overrightarrow{MN}}(x, y) = 0$, 点 $S(x, y)$ 在 $\overrightarrow{MN}$ 的延长线上^[7]. 同理,可以通过判断三角形各边与点的位置关系从而判断该点是否在三角形内部. 如图 1 右图所示,对一个顺时针排列的三角形 ABC ,判断点 $S(x, y)$ 是否在三角形内部,可通过计算点 S 与三角形各边的边函数进行判断,如公式(2)、(3)、(4)所示.

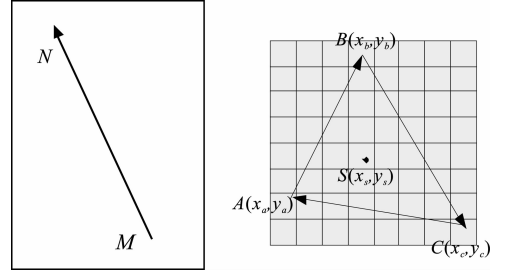


图 1 点与三角形矢量边的位置关系

$$l_{AB}(x, y) = (y - y_a)(x_b - x_a) - (x - x_a)(y_b - y_a) \quad (2)$$

$$l_{BC}(x, y) = (y - y_b)(x_c - x_b) - (x - x_b)(y_c - y_b) \quad (3)$$

$$l_{CA}(x, y) = (y - y_c)(x_a - x_c) - (x - x_c)(y_a - y_c) \quad (4)$$

当(2)、(3)、(4)均为非正时,像素点 S 在三角形内部或三角形的边上,否则点 S 在三角形外部.

我们可以将公式(2)~(4)展开,得到一般每个边函数的形式(5),其中 i 取 1、2、3 时分别对应 AB, BC, CA 三条边的边函数计算.

$$\begin{aligned} l_i(x, y) &= A_i \times x + B_i \times y + C_i \quad (i = 1, 2, 3) \\ A_1 &= y_a - y_b; B_1 = x_b - x_a; \\ C_1 &= y_a \times x_b - x_a \times y_b = x_a \times (y_b - y_a) + y_a \times (x_a - x_b) \\ A_2 &= y_b - y_c; B_2 = x_c - x_b \\ C_2 &= y_b \times x_c - x_b \times y_c = x_b \times (y_c - y_b) + y_b \times (x_b - x_c) \\ A_3 &= y_c - y_a; B_3 = x_a - x_c \\ C_3 &= y_c \times x_a - x_c \times y_a = x_c \times (y_a - y_c) + y_c \times (x_c - x_a) \end{aligned} \quad (5)$$

通过分析边函数,我们可以得出点 S 和三角形之间的位置关系:

情况 1: 当 $l_1 < 0, l_2 < 0, l_3 < 0$ 时,则 S 点位于三角形内部.

情况 2: 当 $l_1 = 0, l_2 = 0, l_3 < 0$ 或 $l_1 < 0, l_2 = 0, l_3 = 0$ 或 $l_1 = 0, l_2 < 0, l_3 = 0$ 时,则 S 点为三角形顶点.

情况 3: 当 $l_1 = 0, l_2 < 0, l_3 < 0$ 时 S 点在 AB 边上,且不为三角形顶点.

当 $l_1 < 0, l_2 = 0, l_3 < 0$ 时 S 点在 BC 边上,且不为三角形顶点.

当 $l_1 < 0, l_2 < 0, l_3 = 0$ 时 S 点在 CA 边上,且不为三角形顶点.

情况 4: 如果不属于情况 1、2、3,则 S 点位于三

角形外部。

从硬件实现的角度看,我们利用边函数的符号位就很容易判定出判断 S 点与直线之间的关系,因此,将公式(5)四种情况变为两种:

情况 1:当 l_1, l_2, l_3 符号位都为 1 的时候: S 点位于三角形内部。

情况 2:不属于情况 1 下, S 点位于三角形外部或位于三角形上。

三角形的边函数的多项式系数即公式(5)中 $A_1, B_1, C_1, A_2, B_2, C_2, A_3, B_3, C_3$ 可以在三角形建立模块中计算出来,并且对于一个三角形只需要进行一次计算. 如果利用公式(1)判断一个点与三角形需要 6 次乘法,15 次加法,而采用公式(5)后只需进行 6 次乘法,9 次加法。

由公式(5),我们可以得到点 (x, y) 的邻接点 $(x+t_1, y+t_2)$ 的三个边函数 $l_{t_i}(t_1, t_2), l_i$ 为 (x, y) 点的边函数计算值,其中 i 为 1、2、3 的时候分别对应于三角形三条边。

$$\begin{aligned} l_{t_i}(x+t_1, y+t_2) &= \\ A_i \times (x+t_1) + B_i \times (y+t_2) + C_i &= \\ A_i \times x + B_i \times y + C_i + t_1 \times A_i + t_2 \times B_i &= \\ l_i + t_1 \times A_i + t_2 \times B_i \quad (i=1, 2, 3) \end{aligned} \quad (6)$$

本文设计中 t_1 和 t_2 为 2 或者 1/2 的倍数,所以 $t_1 \times A_i$ 和 $t_2 \times B_i$ 就由乘法器变为了对 A_i 和 B_i 的移位操作. 这样,我们可以很方便的计算出 (x, y) 的邻接点,也是几何反走样的基础。

2.2 基于 tile 的层次化光栅化算法

本文的算法的思想是对包含一个三角形区域分成四个子区域,按照左下,右下,右上,左上的顺序分别记为 $SPatch0, SPatch1, SPatch2, SPatch3$. 如果某个子区域不被该三角形所覆盖,则抛弃该子区域,否则对该子区域进行细分,依次递归地处理四个子区域,直到一个区域只包含了 2×2 个点。

下面给出判断一个子瓦片 $SPatch\ i (i=0, 1, 2, 3)$ 是否三角形所覆盖,如果符合其中任何一个判断条件,则保留该子瓦片:

条件 1: $SPatch_i$ 中的某个顶点(如图 2 中所标记)位于三角形内部,则保留该 $SPatch_i$;

条件 2:三角形顶点位于 $SPatch_i$ 内部,则保留该 $SPatch_i$;

条件 3:如果一个 $SPatch_i$ 的边与三角形的边有相交关系,则保留该 $SPatch_i$;

条件 4:一个 $SPatch_i$ 有两条及以上边有顶点,则保留该 $SPatch_i$;

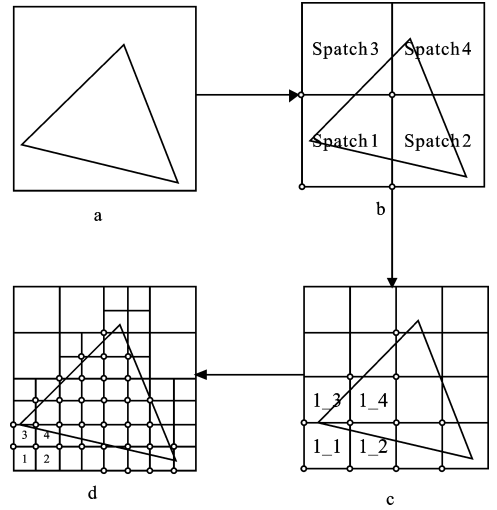


图 2 递归进行三角形光栅化

条件 5:三角形有两个顶点位于一个 $SPatch_i$ 的两个对角顶点上,则保留该 $SPatch_i$ 。

条件 6:三角形有两个顶点分别位于一个 $SPatch_i$ 的顶点和一个对角边上,则保留该 $SPatch_i$ 。

算法伪代码如下:

```

procedureTileRaster(SPatch, Triangle)
//SPatch 为瓦片的数据结构,保存瓦片的相关信息,如左下角坐标,瓦片大小等等;
//Triangle 为三角形的数据结构,保存三角形相关信息,如顶点,边函数系数等等.
begin
if sizeof_SPatch(SPatch) = 2
then
//递归结束条件,分别检测 SPatch 的四个顶点是否在三角形之内,若是输出相应顶点;
else if SPatch_in_Triangle(SPatch, Triangle) then
//SPatch 在一个三角形内,直接输出 SPatch 内的所有点;
else begin
    SPatch0 := Sub_SPatch(SPatch, 0);
    //第一个子 SPatch;
    if SPatch_cross_Triangle(SPatch0, Triangle) then
        TileRaster(SPatch0, Triangle);
    //递归调用
    SPatch1 := Sub_SPatch(SPatch, 1);
    //第二个子 SPatch
    if SPatch_cross_Triangle(SPatch1, Triangle) then
        TileRaster(SPatch1, Triangle);
    //递归
    SPatch2 := Sub_SPatch(SPatch, 2);
    if SPatch_cross_Triangle(SPatch2, Triangle) then
        TileRaster(SPatch2, Triangle)
    SPatch3 := Sub_SPatch(SPatch, 3);

```

```

if SPatch_cross_Triangle(SPatch3, Triangle) then
TileRaster(SPatch3, Triangle)
end; end;

```

上述伪代码中的函数定义如下:

```

functionSPatch_cross_Triangle
(SPatch, Triangle) : Boolean

```

由条件 1-条件 6, 计算边函数得到 SPatch 与 Triangle 的关系, 若 SPatch 需要保留返回 True 否则返回 False;

```

functionSPatch_in_Triangle
(SPatch, Triangle) : Boolean

```

由条件 1-条件 6, 计算边函数得到 SPatch 与 Triangle 的关系, 若 SPatch 的四个顶点都在 Triangle 之内返回 True 否则返回 False;

```

function sizeof_SPatch(SPatch) : integer

```

返回 SPatch 大小

```

function Sub_SPatch(SPatch, ind)

```

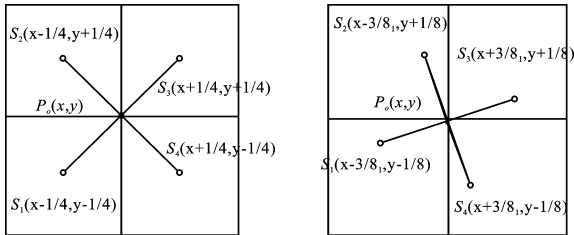
返回 SPatch 的第 ind 个子瓦片

3 RGSS 超采样算法优化设计

在光栅化阶段用有限的采样点表示图元时会出现走样现象, 为了减少走样现象, 文献[8]提出了反走样算法. 目前硬件使用较为广泛的有: Cook 的随机采样^[9], 超级采样^[10], 多重采样技术^[10,11], 自适应采样等. 本文采用超级采样算法对光栅化后的三角形进行反走样处理.

超采样是几何反走样中常用的方法之一, 它的核心思想是把一个像素细分为几个子像素进行处理, 对子像素进行采样, 并对采样的子像素颜色值进行加权获得该像素的颜色值. 常见的算法用 2×2 (4x) 或 4×4 (16x) 个子像素来构成一个像素点, 文献[12]指出只要进行 16x 的超级采样就可以完全消除视觉上的几何走样现象.

常见的超级采样算法有两种: OGSS (Ordered Grid Super-Sampling) 和 RGSS (Rotated Grid Super-Sampling), 本文采用 (4x) RGSS 算法对光栅化后的三角形做反走样处理.



(a) OGSS 超采样示意图 (b) RGSS 超采样示意图

图 3 RGSS 反走样

如图 3 所示, OGSS 算法是将一个原始像素平

均划分为四个子像素区域, 找出原始像素的中心点 $P_0(x, y)$, 再以 P_0 点为中心对四个子像素区域采样, 取得的四个采样点分别为 S_1, S_2, S_3, S_4 , 很容易获得这四个子像素相对于 P_0 的坐标, 如图 3(a) 所示.

RGSS 超级采样算法是在 OGSS 的基础上, 将子像素采样点顺时针或者逆时针旋转一个角度, 一般在 20 到 30 度之间^[13]. 如图 3 所示, 像素中心点为 $P_0(x, y)$, 子像素采样点旋转角度为 α ($20^\circ \leq \alpha \leq 30^\circ$). 为了能让硬件实现, 必须将取值旋转角度范围确定在一个固定值. 当旋转角度为 26.565° 时可确定四个点的坐标.

确定了出 RGSS 四个子像素坐标后, 利用公式 (5) 计算出 P 点边函数后, 利用计算出的系数, 可计算出 S_1, S_2, S_3, S_4 四个子像素边函数并判断出他们与三角形的位置关系.

$$\begin{aligned}
 S_1: l_{S_1} &= l_{P_0} + \frac{3}{8}A_i + \frac{1}{8}B_i \\
 S_2: l_{S_2} &= l_{P_0} - \frac{1}{8}A_i + \frac{3}{8}B_i \quad (i = 1, 2, 3) \\
 S_3: l_{S_3} &= l_{P_0} - \frac{3}{8}A_i - \frac{1}{8}B_i \\
 S_4: l_{S_4} &= l_{P_0} + \frac{1}{8}A_i - \frac{3}{8}B_i
 \end{aligned} \tag{7}$$

公式 (7) 中的乘 $3/8$ 可以用乘 $(1/4 + 1/8)$ 来代替, 即用移位与加法代替了运算中的乘法运算. 这也是本设计 RGSS 采用 26.565° 角的原因, 确保了系数都为 $1/2$ 的整数倍, 保证 4x RGSS 计算中公式 (6) 没有乘法运算.

如果一个子像素位于三角形 ABC 内, 那么该子像素参与对那么 P_0 点的颜色加权平均, 如果不在三角形内部, 则不参与对颜色的加权平均, 计算公式如公式 (8), C_{P_0} 代表颜色, s_i 表示第 i 个子像素, i 的取值范围为 $1 \sim 4$. 如果 s_i 属于三角形则 $s_i \in \Delta ABC$ 为 1, 否则为 0. 从硬件的实现角度看, 我们仅用加法和移位就实现了基于 RGSS 的几何反走样.

$$C_{P_0} = \sum_{i=1}^4 C_{s_i} \times (s_i \in \Delta ABC) \div 4 \tag{8}$$

4 光栅化及反走样算法的硬件实现

本文根据基于瓦片设计的 GPU 的特点, 提出的一种深度优先的光栅化硬件架构, 顶层框架如图 4 所示. 设计包含了三个模块:

(1) 边函数计算模块: 第一级根据公式 (5) 计算出三角形三条边函数系数 A_i, B_i 和 C_i ; 第二级

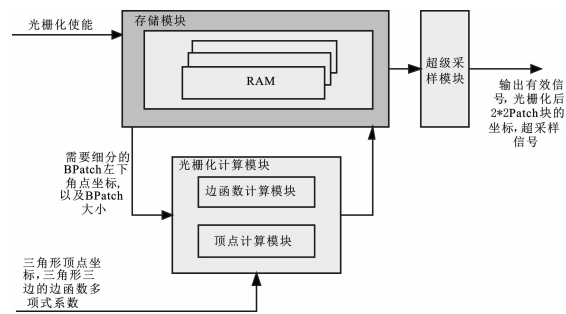


图 4 光栅化及反走样算法的硬件电路

根据公式(2)、(3)计算出四个子瓦片 Spatch i 各个顶点的边函数值;用该边函数值根据条件(1)~(6)判断 Spatch i 是否保留. 当 Spatch i 大小为 2×2 时,不再细分.

(2)存储模块:这个模块包含三个 RAM,分别保存 $8 \times 8, 4 \times 4, 2 \times 2$ 大小需要进一步细化的 Spatch i 的相关信息.

(3)反走样模块:根据公式(7)计算出四个子像素 l_s 的值. 然后根据条件 1~6 判断所计算的四个子像素是否在三角形内部,最后根据子像素的颜色加权值计算得到原像素颜色值.

5 实验结果与分析

为了测试本文算法的正确性,用 C/C++ 语言对本文算法进行软件模拟,我们在 VS 下添加 EGE(Easy Graphics Engine)图形库,进行算法仿真.

这里提供 2 种情况下的测试结果图如图 5 所示. 图中的虚线框代表 16×16 的 tile,每个实线框为 2×2 的 Patch. 图像中深色的块代表软件计算结果所得的三角形覆盖块,所有被三角形覆盖的块都为深色,没有被覆盖的块都为白色.

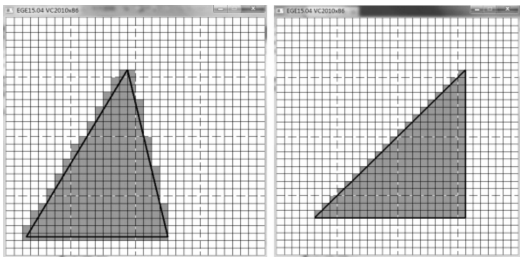


图 5 软件测试的三角形光栅化算法

添加 4x RGSS 后的测试结果如图 6、7 所示,可以看出开启超级采样后边缘的走样现象减少了.

本设计在 Xilinx 的 ZC706 开发平台进行了系统验证上,结果表 1 如下.

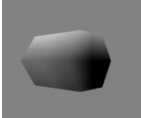
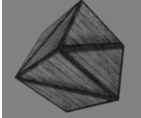


图 6 开启 4xRGSS 的三角形



图 7 RGSS 反走样算法的对比

表 1 对不同复杂度的场景进行测试

场景	旋转的正六边形柱体	旋转的立方体盒子	旋转的彩色兔子
ZC706 板上运行结果			
绘制点的个数	72	36	337152
绘制速率	86.5FPS	83.1FPS	6.8FPS

6 结束语

本文在基于边函数三角形光栅化算法的基础上,提出了一种基于边函数的层次化的三角形光栅化及旋转超采样易于硬件实现算法,通过层次化方法一次对多个像素进行舍弃判断,提高了光栅化的速度,同时利用光栅化中的计算结果,通过移位和加法实现了 RGSS.

参考文献:

[1] Lindholm E, Nickolls J, Oberman S, et al. NVIDIA Tesla: A Unified Graphics and Computing Architecture[J]. IEEE Micro, 2008, 28(2):39-55.

[2] Garber L. GPUs Go Mobile [J]. Computer. 2013, 46(2): 16-19.

[3] 符鹤, 谢永芳. TBR 架构 GPU 中三角形高效光栅化[J]. 中国图象图形学报, 2015, 20(4):527-532.

[4] 杜慧敏, 郝哲, 王鹏超, 等. 三角形的光栅化与反走样算法[J]. 西安邮电大学学报, 2015, 20(3):33-38.

[5] Fuchs H, Goldfeather J, Hultquist J P, et al. Fast spheres, shadows, textures, transparencies, and image enhancements in pixel-planes[C]// Conference on Computer Graphics and Interactive Techniques. ACM, 1985:111-120.

- [6] Popescu V, Rosen P. Forward rasterization[J]. ACM Transactions on Graphics (TOG), 2006, 25(2): 375-411.
- [7] Sun C H, Tsao Y M, Lok K H, et al. Universal Rasterizer with edge equations and tile-scan triangle traversal algorithm for graphics processing units[C]//IEEE International Conference on Multimedia and Expo. IEEE Press, 2009:1358-1361.
- [8] Akeninemoller T, Haines E, Hoffman N. Real-time rendering[J]. A. k. peters Limited, 2008, 32(2):163 - 173.
- [9] Cook R L. Stochastic sampling in computer graphics [J]. Acm Transactions on Graphics, 1986, 5(1): 51-72.
- [11] Hraa. high-resolution antialiasing through multisampling[C]//USENIX Technical Conference. 2001.
- [12] 浅谈抗锯齿技术. 中关村在线. <http://hard.zol.com.cn/labs/2002/1007/48701.shtml>

- [13] Popescu V, Rosen P. Forward rasterization[J]. ACM Transactions on Graphics (TOG), 2006, 25(2): 375-411.
- [14] Shen Y L, Seo S W, Zhang Y, et al. A Low Hardware Cost 2D Vector Graphic Rendering Algorithm for Supersampling Antialiasing[J]. Computer Engineering & Applications, 2010, 1(31):141-144.

作者简介:

刘青楠 男,(1992-),硕士研究生. 要研究方向为计算机图形学. E-mail:330743434@qq.com.

曾泽沧 男,(1967-),讲师. 研究方向电路系统设计.

杜慧敏 女,(1967-),博士,教授,硕士生导师. 研究方向集成电路设计.

(上接第25页)

搜索精度以及鲁棒性等方面有了较大改善. 最后将改进算法用于 WSN 网络节点的部署优化,得到了满意的优化效果. 此外,作为一种较新的算法,ALO还有许多改进方向,哪些改进策略能以较低的复杂度显著提升算法性能,是下一步需要研究的问题.

参考文献:

- [1] Mirjalili S. The ant lion optimizer[J]. Advances in Engineering Software, 2015, 83(C):80-98.
- [2] 栗然, 张凡, 靳保源, 等. 基于改进蚁狮算法的电力系统最优潮流计算[J]. 电力科学与工程, 2017, 33(9):15-22.
- [3] 赵世杰, 高雷阜, 于冬梅, 等. 带混沌侦查机制的蚁狮优化算法优化 SVM 参数[J]. 计算机科学与探索, 2016, 10(5):722-731.
- [4] 李宗妮, 吴伟民, 林志毅. 一种采用改进蚁狮优化算法的图像增强方法[J]. 计算机应用研究, 2018, 34(3):1-2.
- [5] Gergel V, Grishagin V, Gergel A. Adaptive nested optimization scheme for multidimensional global search[J]. Journal of Global Optimization, 2015, 66(1):1-17.
- [6] Dai X, Yuan X, Wu L. A novel harmony search algorithm with gaussian mutation for multi-objective optimization [J]. Soft Computing, 2017, 21(6): 1549-1567.
- [7] Hakl H, Uğuz H. A novel particle swarm optimization algorithm with Levy flight[J]. Applied Soft Computing Journal, 2014, 23(5):333-345.

- [8] 吴伟民, 张晶晶, 林志毅, 等. 双重反馈机制的蚁狮算法[J]. 计算机工程与应用, 2017, 53(12):31-35.
- [9] 赵辉, 李牧东, 翁兴伟. 具有自适应全局最优引导快速搜索策略的人工蜂群算法[J]. 控制与决策, 2014, 28(11):2041-2047.
- [10] 李煜, 裴宇航, 刘景森. 融合均匀变异与高斯变异的蝙蝠优化算法[J]. 控制与决策, 2017, 32(10): 1775-1781.
- [11] 丁旭, 吴晓蓓, 黄成. 基于改进粒子群算法和特征点集的无线传感器网络覆盖问题研究[J]. 电子学报, 2016, 44(4): 967-973.
- [12] 赖锦辉, 梁松. 基于改进人工萤火虫算法的无线传感器网络覆盖优化[J]. 计算机测量与控制, 2014, 22(6): 1862-1864.
- [13] 王明亮, 闵新力, 薛君志. 基于改进人工鱼群算法的 WSN 覆盖优化策略[J]. 微电子学与计算机, 2015, 33(6):78-81.
- [14] 廖灿星, 张平, 李行善, 等. 基于混合人工鱼群算法的传感器网络优化[J]. 北京航空航天大学学报, 2010, 36(3): 373-377.

作者简介:

王若安 男,(1993-),硕士研究生. 研究方向为自动测试系统设计. E-mail:550861874@qq.com.

周越文 男,(1972-),硕士,教授. 研究方向为测试原理.

韩博 男,(1991-),博士研究生. 研究方向为测试软件设计.

李剑峰 男,(1993-),硕士研究生. 研究方向为故障诊断与健康管理.

刘强 男,(1993-),硕士研究生. 研究方向为故障预测.