基于硬件加速的 Voronoi 图并行生成研究

辛胜钧 (034964)，谢剑(034961)，袁仲达(034966)

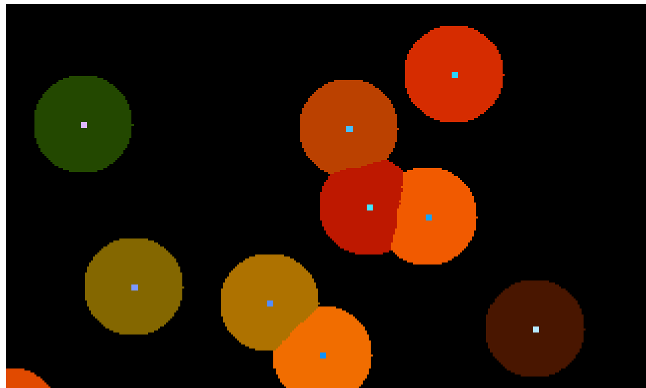
一、 问题背景：

给 Voronoi 图着色或生成加权的 Voronoi 图是一个富有挑战性的问题。我们的想法是用并行的算法生成加权的 Voronoi 图并对 Voronoi 图进行着色。

为了提高并行处理的速度，我们用 GPU 来处理。Cg（C for Graphics）是一种用来对 GPU 编程的高级语言，其语法结构，函数调用和绝大多数数据类型与 C 语言很相近。通过 CG 可以很方便的进行三维模型的变形处理（平移，旋转，拉伸）和渲染控制（着色，贴图变换）。我们使用的主要是其平移运算。利用 GPU 的图像运算能力，快速生成 Voronoi 图。

二、 问题分析：

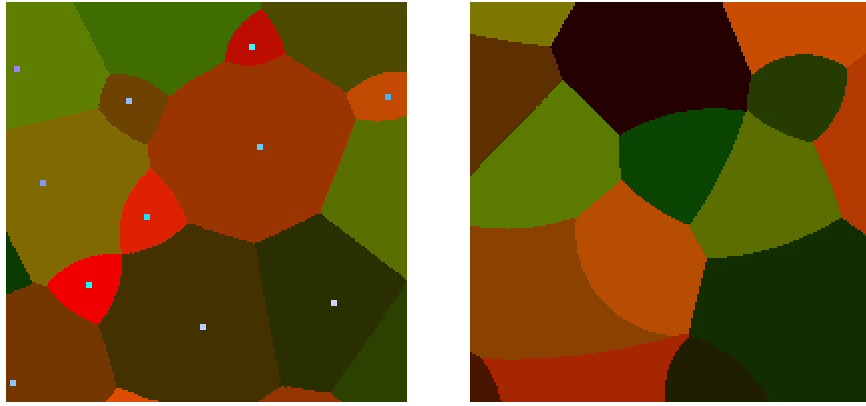
Voronoi 图已经有许多种生成算法，但是先生成 Voronoi 图再染色的话无疑会浪费染色所带来的附加信息。而加权的 Voronoi 图由于其边界是曲线，更是会增加问题的难度。为此我们参考了 Voronoi 图的原始提法：节点（Site）的控制域。每一个节点(Site)的控制域以等速度扩大直到在发生冲突时停下，这些停止点所连成的自然就是 Voronoi 图。



图一：增长中的节点控制区域

为此，我们采用了并行的方法来模拟每一个点的控制域的增长。因为进程之间的同步机制可以引入一定的随机性，不会出现恒定循环时对应的优先级领先现象。这种现象反映到离散的点空间上就是两个点之间的模糊区域（有离散误差引入）永远属于某一个点。

我们知道一个等高的圆锥族如果位于同一水平面上，他们两两相交的曲线在公共的地面上的投影就是一个 Voronoi 图，节点就是圆锥的顶点。如果某一个圆锥所在的水平面高于或者低于其他，就相当于对节点赋予了绝对权重（不同的初始范围，默认为 0）。如果某个圆锥的底边半径不等于其他的圆锥，则相当于对节点赋予了相对权重，对应于该节点和其他节点增长速度比例（默认为 1，即同步增长）。

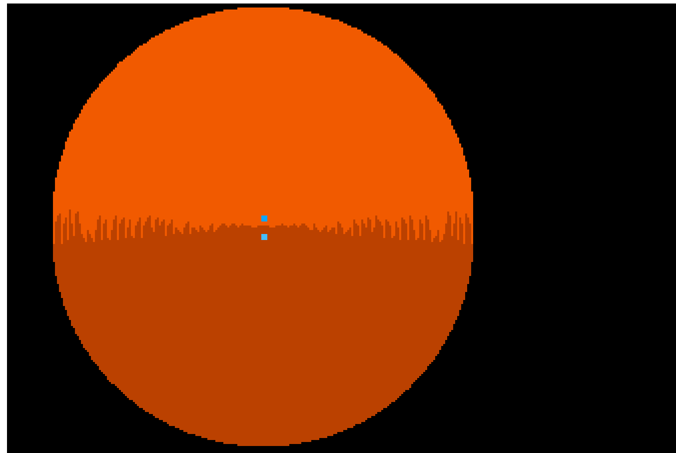

图二：相对权值不同（右）和绝对权值（左）不同的 Voronoi 图

三、 解决方案：

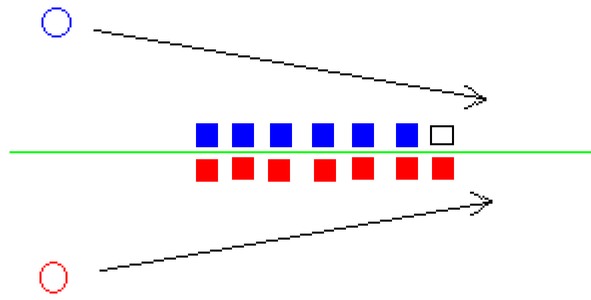
A, 软件的方法

采用并行算法，用离散的事件来模拟控制域的增长。给每一个 Site 关联一个同步的线程，这样就保证了扩大的停止线就一定在 Voronoi 图的附近，最多一个像素的误差。

然而这种离散化的一个代价就是在一定条件下误差难以控制。当两个点靠得很近时，其连线垂直平分线附近距离两个点都很远的时候，离散的点就会属于先对这个区间测试并占据的（参见下图）。

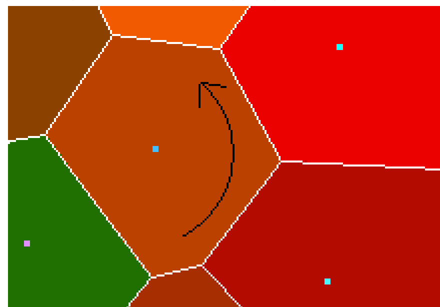

图三：偶极子的误差非常明显

下图更清楚地说明了这种现象：因为白色的空位到两个节点的距离相差很小，在两个节点控制区域同步增长时就成了谁先到归谁管。再加上进程同步的随机性，就出现了上图的那种锯齿现象。



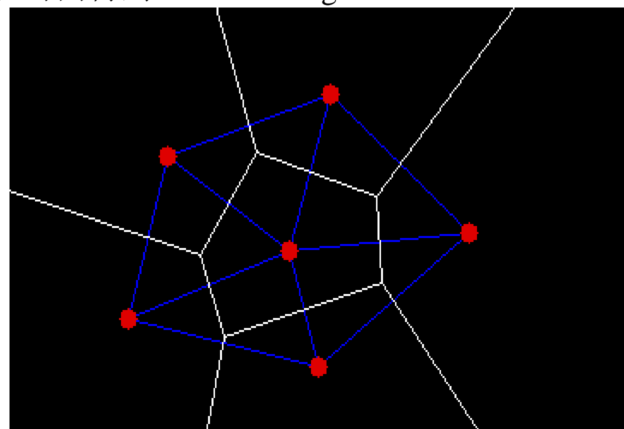
图四：节点对靠近公共区域离散点的争夺

得到染色的 Voronoi 图以后就可以利用边缘检测得到节点之间的相邻信息（Delaunay Triangulation）。因为我们知道节点的位置，亦即每一个凸的单色区域的内部节点，边缘检测就成了很容易的事情。



图五：边缘检测示意

从节点之间的相邻信息（Delaunay Triangulation）出发，利用对偶关系进行演算，就可以在线性时间得到 Voronoi Diagram。



图六：由三角剖分得到 Voronoi 图

赋予不同的 Site 不同的增长速度，就可以方便的模拟加权的 Voronoi 图生成过程。

为了方便的记录节点的信息，我们采用了类似 DCEL 的数据结构来存储顶点的信息。为了节省篇幅，变量的意义不一一介绍了，在源代码里有相关注释。

```
struct _DCEL_Face{
    int id;
    _DCEL_HalfEdge*    incEdge;
    _DCEL_Coord        siteX,siteY;
    COLORREF           siteColor;
    CPen               sitePen;
    CBrush             siteBrush;
    _GROWTH_PARA       range;
    bool               bGrowthing;
    int*               IDNeighbors;
    int                nNeighbor;
    CPoint**           pPoly;
    int                nPoly;
    int                oddPoly;
    double             weight;
}
```

用软件模拟时，如果在一部分节点增长一定时间以后再引进新的节点，这些新引进的节点就会具备相对较低的绝对权值，他们控制的区域要比先前的节点在半径上少一个和延迟时间成正比的值。这一现象可以在插图二中看到。

B, 硬件加速的方法

基于圆锥交线投影的 Voronoi 图生成算法需要在三维空间构造很多个圆锥体，这些圆锥体的交线如果完全由 CPU 计算的话，势必会占用很多计算资源，将问题复杂化。所以这个算法必须使用硬件加速。

在构造场景时，我们使用了同样参数的绘图函数来建造圆锥，这些物体都处于同一个位置。CG 的顶点映射功能负责将他们整体的搬移到应该在的位置。CG 是对 GPU 直接编程控制的语言，可以较方便的完成这一功能。关于 GPU 进行顶点映射的工作原理，请参见文献[2]的相关网站和文档说明。

场景构造完成以后，将圆锥的公共底面设置为平行投影的目标平面，生成的图像就是用不同颜色标志了的 Voronoi 图了。

下面就是利用硬件系统加速的简易流程图：

1，初始化图形系统，并检查硬件是否支持 Cg。

2，使用 cgCreateContext 创建 Context 环境，使用 cgCreateProgramFromFile 创建 Cg 程序，由 cgGLLoadProgram 载入 Cg 程序。

3，初始化 Cg 程序的参数输入接口。

4，设定坐标系的变换，使生成的图像精确对应于各个节点的屏幕坐标。主要用到的是 glViewport 和 glOrtho;

5，使用 cgGLSetStateMatrixParameter 设定 Cg 变换矩阵。

6，在生成每一幅图像时，首先绑定 Cg 程序 cgGLBindProgram，设置顶点映射规则，然后打开顶点映射功能。

7，用普通的 OpenGL 绘图历程创建圆锥，这时的图像输出就已经是经过 Cg 加速以后生成 Voronoi 图了。

8，关闭顶点映射功能。将图像转存以备处理。

Cg 核心程序(cg_vonronoi.cg)：

1， 输入参数：

包含三个元素的位置向量

in float3 position : POSITION,

包含四个元素的颜色向量

in float4 color : COLOR0,

二维变换矩阵

const uniform float4x4 modelViewProj,

顶点偏移向量

const uniform float4 Offset,

顶点颜色向量

const uniform float4 SiteColor

2， 输出参数：

顶点映射以后的输出位置向量

out float4 positionO : POSITION,

顶点映射以后的输出颜色向量

out float4 colorO : COLOR0,

3， Cg 处理：

将位置向量的元素由三个扩充为四个

```
float4 tempPosition = float4( position.x,position.y,position.z,1.0f );
```

根据图形系统的当前变换矩阵计算出顶点的当前位置

```
positionO = mul( modelViewProj, tempPosition);
```

根据输入的偏移向量求出顶点的最终位置

```
positionO += Offset;
```

给顶点着色

```
colorO = SiteColor;
```

四、 遇到的问题和解决方法

1，进程的同步。

按照最初的设计方法，每个节点对应的进程计算该节点的控制区域，并标定每一个尚未被标志的离散点为该节点的控制区域。但是，实际上由于进程存在优先级上个细微差别，如果不加以控制，会出现某一个节点进程的 Starvation 现象，及该节点根本的不到应用的增长机会。

所以我们采用了定期同步的手段，即在运行一个适当的时间间隔以后，让所有进程停下，等待下一个开始运行的信号灯。这个时间间隔我们取得是一个离散像素的增长时间。

2，相邻信息的提取。

同步算法生成的是染色以后的 Voronoi 图，没有节点的相互几何关系也没有 Voronoi Cell 的几何描述，所以说并不是真正的 Voronoi 图。我们最初的设计是在节点增长时实时的记录下每一个增长时发生的冲突事件，但在是显示发觉这样做的效率非常底。因为每两个相邻节点之间的冲突事件会发生很多次，并且如果不控制节点区域增长峰线的方向的话，就很可能会出现不相邻节点的冲突信息，这是不正确的。而且，如果使用硬件加速的话，只能得到染色的最后结果，而无法控制图形硬件的中间运行状态。

所以，我们放弃了最初的冲突记录思想，改用边缘检测算法在已经染色的 Voronoi 图上查找相邻信息。由于每一个节点必然处于包含它的一个单色图区域里面，我们只需要沿着该单色区域的边界遍历，同时记录所遇到的各种不同区域就得到了和该节点相邻的所有节点,构造初此节点族的三角剖分（Delaunay Triangulation）。根据对偶关系，计算 Voronoi 就是一件很容易的事情了。

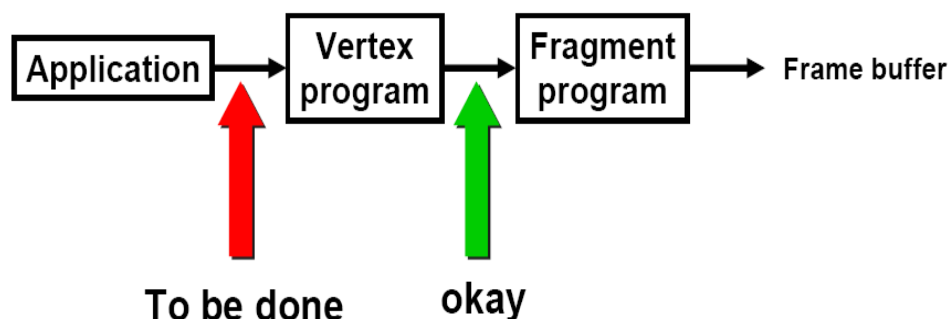
3，动态生成 Voronoi 图。

以往的算法大都用到所谓的 Dynamic Program 技术，既当某一个顶点的信息改变以后只需要重新计算和它相邻的节点关系就可以保持整个图的正确性。这种方法的一个缺陷就是不能高效快速的处理多个节点同时变动的情况。而我们的算法是利用全部顶点信息的，所以处理一个节点的改变和处理很多节点的改变是一样的。

根据导师的提示，我们设计了一个对顶点加上随机扰动，并实时生成的 Voronoi 图的演示。

4，关于 Cg 的使用。

最初我们并不希望用Cg做显示而是希望把数据传给GPU，让它计算，等它计算好了再把结果取出来给CPU用，这可以利用GPU在计算点偏移和着色速度快的优点来提高程序的运行速度，这是我们最初的GPU加速的思想，可是我们试了很多种方法都无法把结果从GPU中拿出来，后来仔细研究了一下Cg和其他一些Cg源代码才知道GPU把计算出来的结果送到帧缓存然后直接显示，根本就不把结果再返回给CPU。明白了这点后，我们最终就用Cg作了显示。GPU和CPU的连接关系如下图所示：



图七，GPU和CPU的连接关系

五、 进一步的工作

1，点的选取算法 — 哈希表

2，在查询节点的信息时，我们采用了简单的线性查找的技术。为了更好的进行点的查询，可以改用哈希表或者借用点定位的一些成熟算法。

3，节点参数的动态改变（交互式）

在点定位的基础上可以增加顶点参数实时改变的功能，目前只有在添加节点的时候设置位置和权值，使用起来不是很方便。后面的随机扰动也是不受用户控制的，所以应该提供这个接口。

4，在现有的基础上，可以增加节点的有规则运动，这样就可以到达更加引人入胜的演示效果。例如我们可以让一部分节点向右移动而另一部分向左移动，这个比较适合我们的这个平台实现。

六、 难点分析：

1， 由于我们要使用并行算法来生成Voronoi图，因此各个进程的同步问题是我们必须要解决好的。

2， 我们要生成加权的Voronoi图，因此经典的Voronoi图生成算法将不再适用，我们必须自己设计一种算法可以生成动态和静态的加权Voronoi图。

3， 对于加权的Voronoi图，区域边界将不再是直线，而是曲线，具体的说是一段圆弧。因此，在生成加权的Voronoi图时，区域边界将很复杂，不再像经典的Voronoi图那样简单和直观了。

4， 我们以前都没有使用过对GPU编程的CG语言，因此熟练使用CG对我们来说是一个很大的挑战。

七、 参考文献：

[1].Optimal Parallel Randomized Algorithms for the Voronoi Diagram of Line Segments in the Plane and Related Problems, Sanguthevar

Rajasekaran Suneeta Ramaswami,Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA 19104.
[2].User's Manual A Developer's Guide to programmable Graphics，来自 http://developer.nvidia.com/cg.
[3].Voronoi图的Beachline扫描算法的三维图形演示,王益，王垠，潘达.
[4].Computational Geometry, Algorithms and Applications.