# 太空（一）一种恒星分布的建模方案

本文主要探讨模拟构建一种星系中恒星的分布，并选取若干可以使用的恒星并将其相互关联，使游戏化的恒星位置贴近一种真实的物理分布。如果你需要制定一个关于银河系&太空的恒星分布规则，这篇建模方案可能对你有所帮助。里面涉及到的知识点主要包括，高斯分布、几何曲线、三角剖分等。

<QQ20220313-193824-HD.mp4>

## 选定建模对象

目前已知的星系类型主要包括旋涡星系、椭圆星系、壳层星系、棒旋星系（银河系属于此类）、超量旋涡星系。



根据哈伯分类方案的星系类型： E表示椭圆星系，S是漩涡星系，SB是棒旋星系



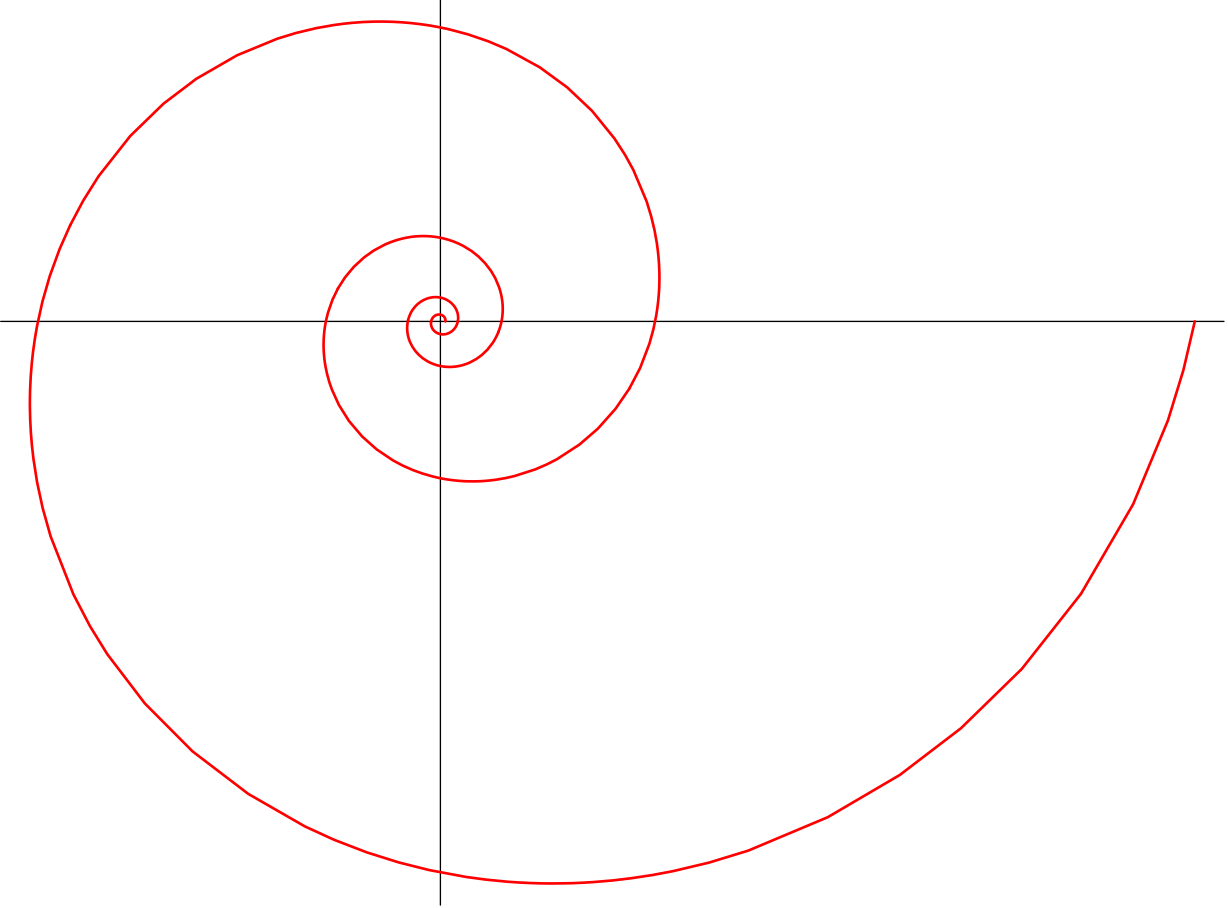
《旋涡星系》与《棒旋星系》

本文选用旋涡星系作为建模对象。

### 对数螺旋

#### 基础建模

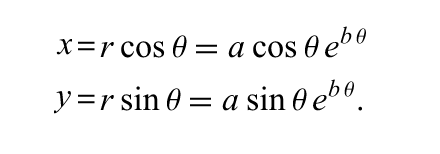
对于旋涡星系，可以考虑使用对数螺旋对其恒星位置进行建模，不过初始曲线的选用也可以使用其他，或者自己改造也行。下面是对数螺旋的相关信息：



极坐标方程：



参数方程：



用代码实现角度在[0,2PI]区间，并将半径归一化到[0,1]区间：

var t = UnityEngine.Random.value;*//[0-1]*float value = 2f \* (float) Mathf.**PI** \* t;*//[0,2PI]*

var r = (Mathf.Pow((float) Math.**E**, (1.0f \* t - Mathf.Floor(1.0f \* t)) \* 1.0f) - 1) / (float) (Math.**E-1)**;

float num6 = r\*Mathf.Cos(value);  
float num7 = r\*Mathf.Sin(value);

在代码中实现得到效果：



#### 半径周期性变化

根据图片效果，我们需要四五条对数螺旋对螺旋星系进行模拟,可以选着对半径进行周期性循环，这里直接进行取余重复，修改半径的方程：

Var numberOfArms = 5;

var r = (Mathf.Pow((float) Math.**E**, (numberOfArms \* t - Mathf.Floor(numberOfArms \* t)) \* 1.0f) - 1) /  
 (float) (Math.**E***-1***)**;*//[0,1]*



#### 螺旋线延长

上面的拖尾太短，两个螺旋之间的间隔比较少，这里加快旋转速度即可延长每条螺旋的延伸长度：

float num6 = r\*Mathf.Cos(value\*4);  
float num7 = r\*Mathf.Sin(value\*4);



### 离散化

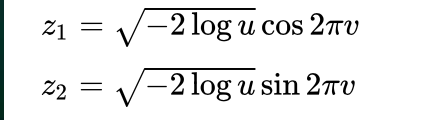
显然，我们不希望恒星都击中在螺旋线上，而是在螺旋线上分布,考虑到图上的效果，我们希望的是在螺旋线上分布的恒星应该是基于正态分布的，即靠近螺旋线的恒星越多、远离螺旋线的恒星越少，所以我们应该使用正态分布，而不是均匀分布了。这里直接使用高斯分布。

#### 高斯随机

常用的正态分布算法：

1. Box-Muller算法，

a.其中u和v在区间[0,1]的均匀分布



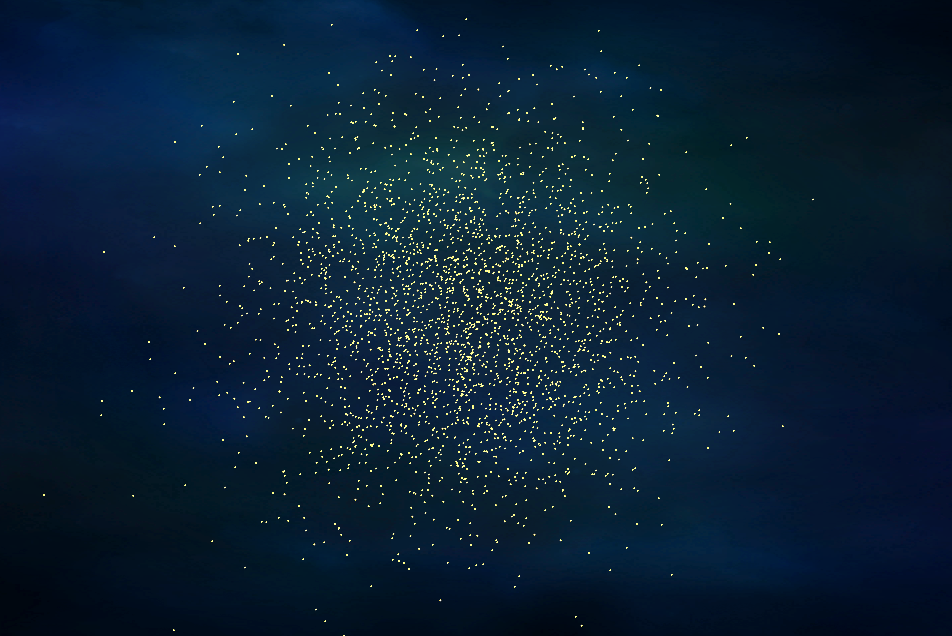
b.极坐标形式，速度更快，且更具有鲁棒性

float x1;  
float w;  
do  
{  
 x1 = 2f \* UnityEngine.Random.value - 1f;  
 float x2 = 2f \* UnityEngine.Random.value - 1f;  
 w = x1 \* x1 + x2 \* x2;  
}  
while ((double)w >= 1.0 || w == 0.0);  
float num4 = Mathf.Sqrt(-2f \* Mathf.Log(w) / w);  
return x1 \* num4;

1. zigurat算法

效率最高，TODO

使用正态分布生成一张星空图

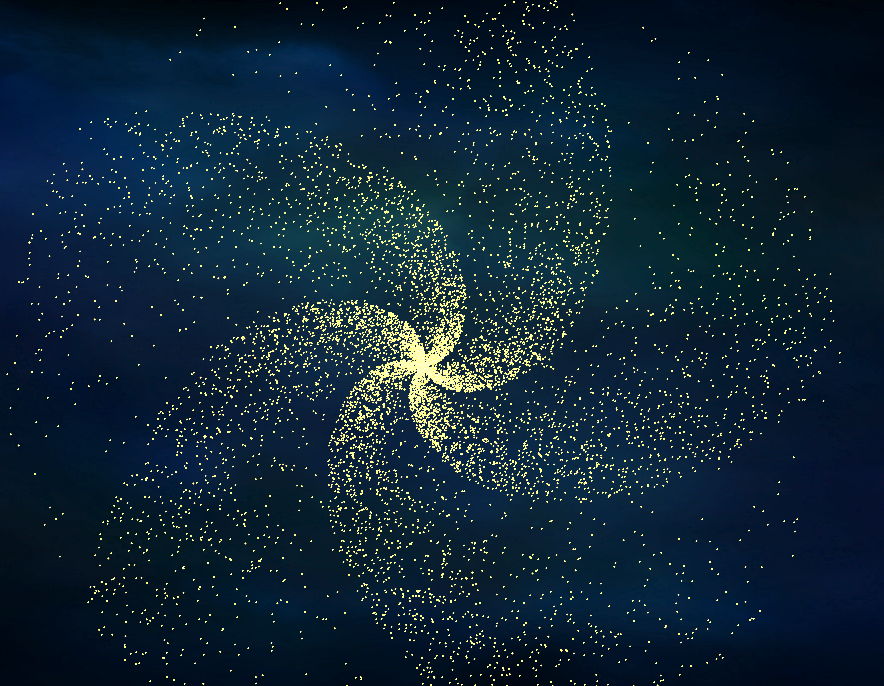


#### 螺旋线离散恒星

有了正态分布后，我们对螺旋线进行正态分布偏移。

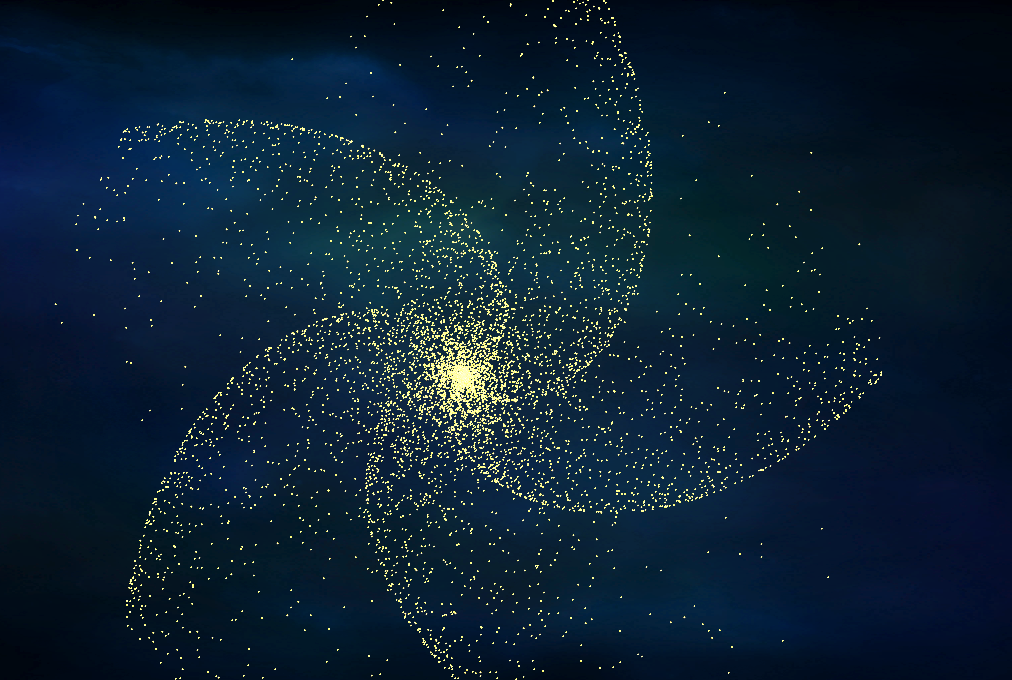
float offset = MathUtil.GaussianRandom \* armOffsetMax;*//[-2,2]*offset /= 2f;*//[-1,1]*if (offset < 0f)  
{  
 offset \*= -1f;  
}

float value = 2f \* (float) Mathf.**PI** \* (t)+offset;*//*



恒星分布太规律，我们是希望越靠近中心，由于恒星越大，引力越大，分布的恒星越多，越远离中心，恒星越收敛：

offset \*= 1/r;*//半斤越小 偏振越大*offset = Mathf.Pow(offset, 2f);*//收敛*



以上、一个漩涡星系的恒星分布数据基本就生成了。

### 恒星选取

#### 距离筛选

如果在游戏设计中，我们肯定不需要使用这么多恒星，我们会选取部门恒星作为实际使用的恒星，这里的原则就属于设计化的范畴了。比如通过简单的距离判定，选取距离超过一定值的恒星作为游戏中使用的恒星，因为距离太近的恒星引力过大，导致互相撞击后消失。

for (int j = 0; j < solarCount; j++)  
{  
 int destroyNum = 0;  
 while (allOrgPos.Count > 0)  
 {  
 int index = UnityEngine.Random.Range(0, allOrgPos.Count);  
 Vector3 vector = allOrgPos[index];  
 allOrgPos.RemoveAt(index);  
 if (!isCloseWithOther(vector, gameSolarPos))  
 {  
 gameSolarPos.Add(vector);  
 break;  
 }

}



#### 恒星网格化

恒星之间存在相互联系，我们为他们之间打通穿梭通道，实现过程为通过对离散2维点建立网格。这里直接使用[Delaunay三角剖析](https://zhuanlan.zhihu.com/p/459884570)算法建立恒星点之间的三角形。

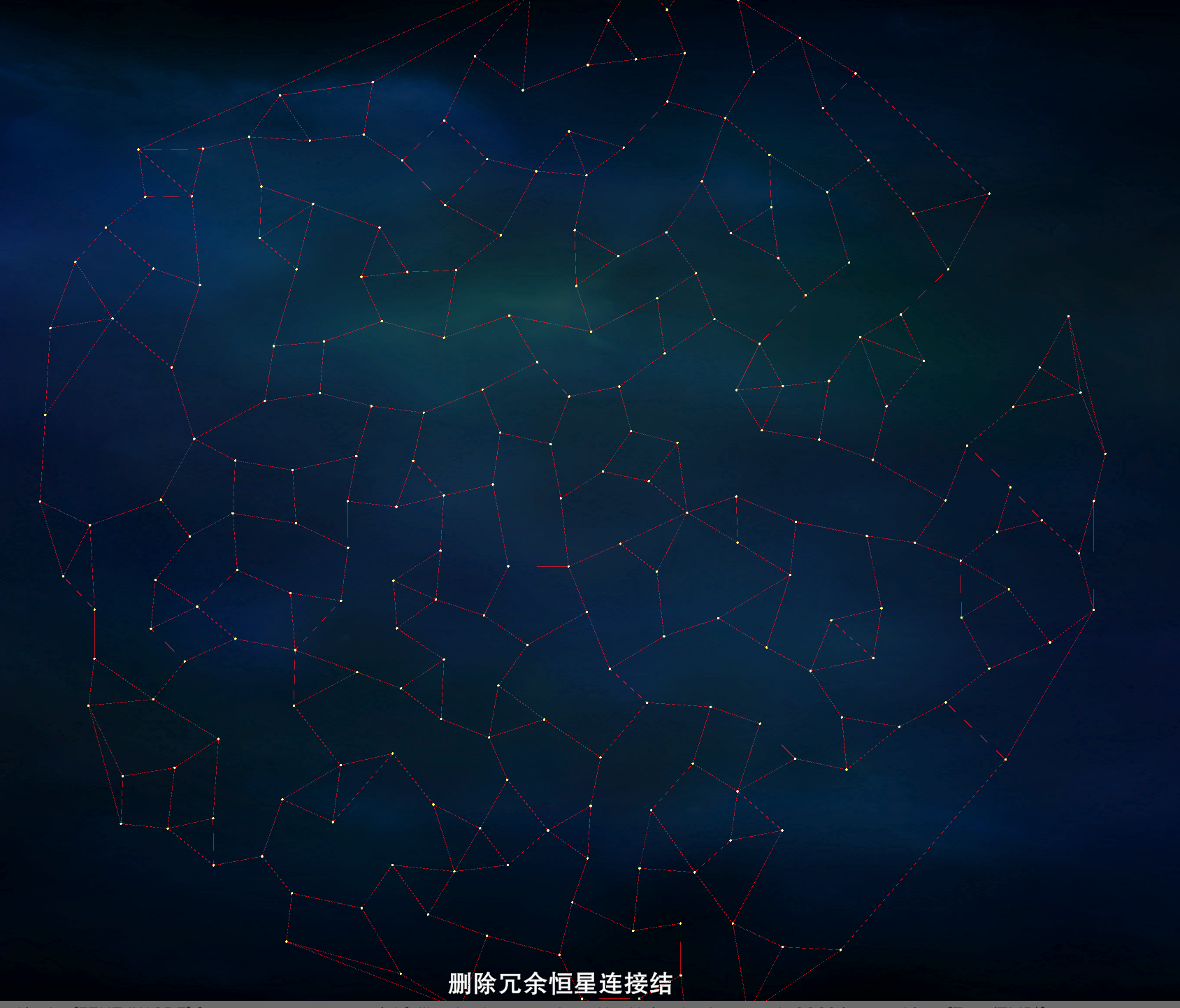
该算法有很多优化版本，这里直接使用比较成熟的[开源库](https://github.com/nol1fe/delaunator-sharp),获取恒星点之间的连接信息。

List<IPoint> list2 = gameSolarPos.Select((Vector3 point) => new Vector2(point.x, point.y)).ToPoints().ToList();  
Delaunator delaunator = new Delaunator(list2.ToArray());  
Dictionary<int, List<int>> dictionary = MathUtil.GenPathConnectionsByMesh(delaunator.Triangles);



作为一种合理的游戏设计，我们不能让恒星与太多恒星之间相互进行直接关联，所以我们删除部分恒星之间冗余的连接。

foreach (int key in posConnects.Keys)  
{  
 List<int> list = posConnects[key];  
 Vector3 pos = posDatas[key];  
 if (list.Count <= minCount)  
 {  
 continue;  
 }  
 list.Sort(delegate(int left, int right)  
 {  
 Vector3 vector = posDatas[left];  
 Vector3 vector2 = posDatas[right];  
 float sqrMagnitude = (pos - vector).sqrMagnitude;  
 float sqrMagnitude2 = (pos - vector2).sqrMagnitude;  
 return sqrMagnitude.CompareTo(sqrMagnitude2);  
 });  
 while (list.Count > minCount)  
 {  
 int key2 = list[list.Count - 1];  
 List<int> list2 = posConnects[key2];  
 if (list2.Count <= minCount)  
 {  
 break;  
 }  
 list.Remove(key2);  
 list2.Remove(key);  
 }  
}



### 银河系的星星

最后那些我们不需要的恒星，将他们还原成星空中的星星，点亮宇宙的星空。使用unity自带的particlesystem自定义位置信息。

stars = new ParticleSystem.Particle[posList.Length];  
particles = GetComponent<ParticleSystem>();  
var main = particles.main;  
main.simulationSpeed = 0;  
for (int i = 0; i < posList.Length; i++)  
{  
 float num = Random.Range(startSizeRange \* 0.5f, startSizeRange \* 1.5f);  
 stars[i].position = posList[i];  
 stars[i].startSize = starSize \* num;  
 stars[i].startColor = color;  
}  
particles.SetParticles(stars, stars.Length);



以上

几何曲线参考：https://mathworld.wolfram.com/topics/Spirals.html

三角行剖分参考：https://github.com/nol1fe/delaunator-sharp