浙江大学

本科实验报告

课程名称：	基于 GPU 的绘制
实验名称：	Birds
姓　　名：	******
学　　院：	计算机学院
系：	数字媒体与网络技术
专　　业：	数字媒体技术
学　　号：	315010****
指导教师：	唐敏

2018 年 1 月 18 日

浙江大学实验报告

实验名称：Birds　　实验类型：操作实验

同组学生：无　　实验地点：外经贸楼

一、 实验目的与内容

a) 实验目的

综合使用 GPU 绘制和加速计算的知识制作演示动画。

b) 实验内容

制作鸟群动画。

二、 实验环境与配置

a) 实验环境

i. 开发环境

Windows 7 SP1 64-bit + CLion 2017.3 + MinGW 5.0 + CMake 3.9.1

ii. 硬件

NVIDIA Quadro K620 (OpenGL 4.5)

iii. 第三方库

GLEW 7.0 + GLUT 3.7 + GLM 0.9.8.5

b) 实验配置

本实验项目使用 C++编写，在项目中依次包含 GLEW, GLUT 和 GLM 的头文件以及其他必要的系统头文件，使用 CMake 生成可执行文件。

三、 实验方法与步骤

a) 编写窗口、事件、键鼠控制等基础模块

复用曾经编写的工程模块。

b) 编写 Shader 类封装着色器相关操作

复用曾经编写的工程模块。

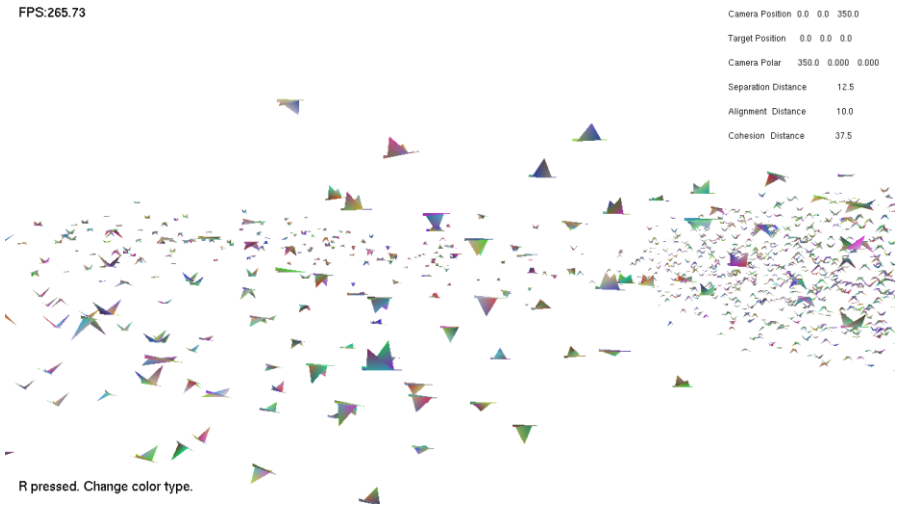
c) 编写着色器以实现功能

细节将在下一节中介绍。

四、 实验分析

a) Overall：实验效果

本次实验的最终效果如下图所示。在不干扰的情况下鸟群应沿环形轨道绕中心盘旋。



b) 个体运动算法介绍

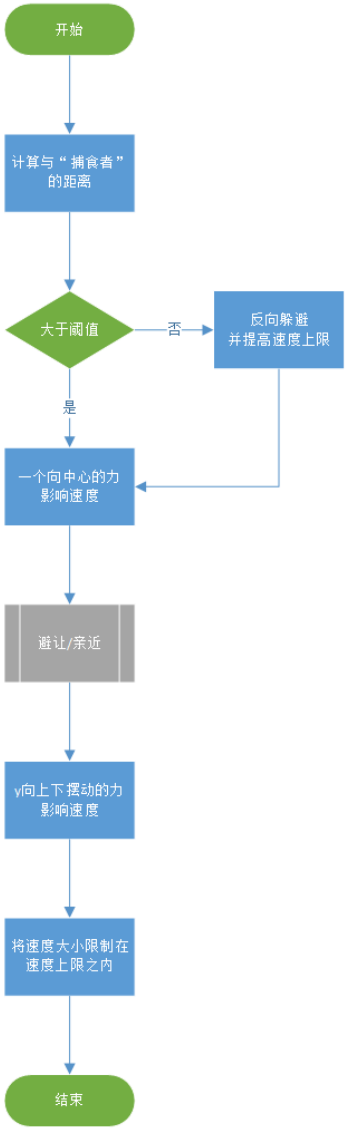


Figure 1 **速度的更新过程**

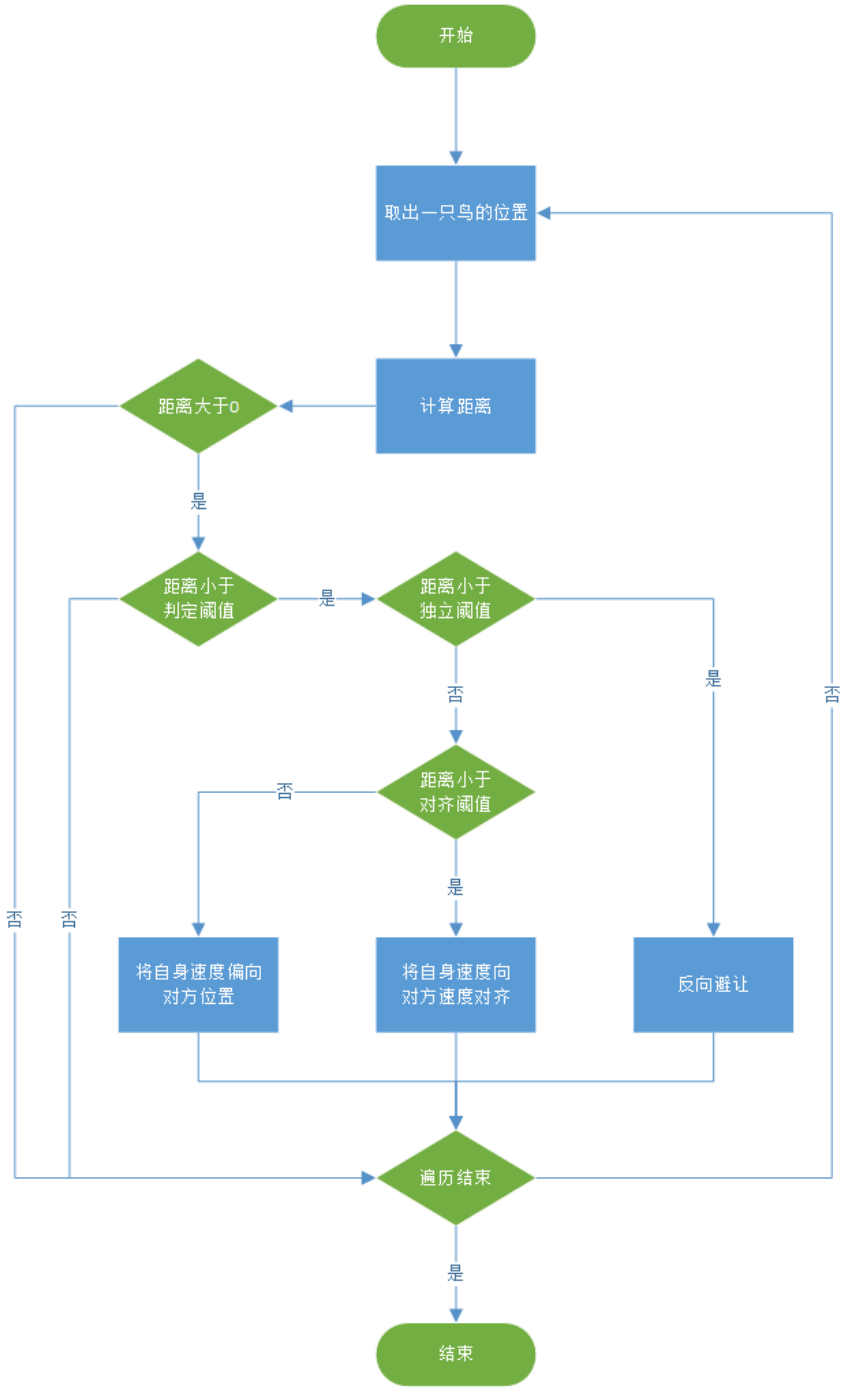


Figure 2 **子流程“避让/亲近”**

鸟群的位置和速度信息分别存放于两张纹理图片中，每次更新时（每秒约 60 次）读入旧的位置和速度信息，更新每一只鸟的位置和速度信息，再将新的位置应用于每一只鸟身上，重新绘制每一个顶点的位置。此节中将着重介绍更新过程中每一个个体所表现出的避让、附和和亲近过程，而忽略 OpenGL 和 GLSL 端的各种设置过程。

如 Figure 1 和 Figure 2 所示，每一只鸟在更新速度信息时，都会在以自身为中心的一定范围 r 内寻找其他鸟，并根据它与自身的距离 d 与最大距离 r 的比值 persent 的处于哪一区域内，做出相应的判断：易知 percent 的大小处于(0, 1)，我们设定一个“独立阈值”（如 0.45）和“对齐阈值”（如 0.65），则当前鸟与目标鸟的位置关系可能有三种：小于“独立阈值”；大于等于“独立阈值”小于“对齐阈值”；大于等于“对齐阈值”。

(1) 小于“独立阈值”。此时两鸟的间距被判定为过小，当前鸟的在自身速度的基础上增加一个反向避让的速度。由于速度的大小会被最后的速度限制所处理，所以主要作用是影响速度方向。

```
if ( percent < separationThresh ) { // low
    // 低于separation阈值，靠得太近了
    // Separation - Move apart for comfort
    f = (separationThresh / percent - 1.0) * delta;
    velocity -= normalize(dir) * f;
```

Figure 3 180:13@compute.frag

(2) 大于等于“独立阈值”小于“对齐阈值”。在这个位置范围内的目标鸟将被当前鸟作为“对齐榜样”，目标鸟的速度矢量将被加成到当前鸟的速度上，造成的影响是使得当前鸟倾向于与目标鸟保持同一方向移动。

```
} else if ( percent < alignmentThresh ) { // high
    // 低于alignment阈值（separation的值也有影响）
    // Alignment - fly the same direction
    float threshDelta = alignmentThresh - separationThresh;
    float adjustedPercent = ( percent - separationThresh ) / threshDelta;
    // 取出参照鸟的速度，把参照鸟的速度方向加到自己身上
    birdVelocity = texture2D( textureVelocity, ref ).xyz;
    f = ( 0.5 - cos( adjustedPercent * PI_2 ) * 0.5 + 0.5 ) * delta;
    velocity += normalize(birdVelocity) * f;
```

Figure 4 185:13@compute.frag

(3) 大于等于“对齐阈值”。目标鸟处于一个不太远又不是很近的距离，我们让当前鸟的速度增加一个指向目标鸟的速度矢量，使它们靠得更近些以进入到“对齐阈值”的范围之内。

```
} else {
    // 否则：靠得有点远
    // Attraction / Cohesion - move closer
    float threshDelta = 1.0 - alignmentThresh;
    float adjustedPercent = ( percent - alignmentThresh ) / threshDelta;

    f = ( 0.5 - ( cos( adjustedPercent * PI_2 ) * -0.5 + 0.5 ) ) * delta;

    velocity += normalize(dir) * f;

}
```

Figure 5 194:13@compute.frag

而在实际应用过程中，我设置了三个值，分别为 Seperation Distance、Alignment Distance 和 Cohesion Distance。独立阈值 separationThresh 和对齐阈值 alignmentThresh 的计算方法如 Figure 6 所示，zoneRadius 即上文提到的以自身为中心的半径 r，这保证了在 Seperation Distance 和 Alignment Distance 都大于等于 0 的情况下，对齐阈值始终大于等于独立阈值。

```
separationThresh = seperationDistance / zoneRadius;
alignmentThresh = ( seperationDistance + alignmentDistance ) / zoneRadius;
```

Figure 6 101:5@compute.frag

仅此，一个个体就可以表现出一定的行为。在大一些的尺度来看，已经可以表现出一定的群体效果。我们最后为它加上一个对“捕食者”的躲避行为：在窗口中移动鼠标，每一只鸟将计算鼠标位置在空间中对应的坐标与自身的距离，若“捕食者”进入了自己的“索敌”范围，则进行避让。于是在窗口中移动鼠标可以干扰鸟群，产生有趣的效果。

```
// dir是捕食者相对当前鸟的位置
dir = predator * BOUNDS - selfPosition;
dir.z = 0.;
// 与捕食者的距离
dist = length( dir );
distSquared = dist * dist;

// 寻找捕食者的距离
float preyRadius = 150.0;
float preyRadiusSq = preyRadius * preyRadius;

// 与捕食者距离过远不会引发躲避行为
if (dist < preyRadius) {
    // move birds away from predator
    // 躲避速度的大小
    f = ( distSquared / preyRadiusSq - 1.0 ) * delta * 100.;
    // 躲避方向为远离捕食者方向
    velocity += normalize( dir ) * f;
    // 增加速度上限
    limit += 5.0;
}
```

Figure 7 126:5@compute.frag

c) 使用纹理、FBO 和片段着色器更新位置和速度

i. 尝试使用 OpenGL 的 image variable，允许片段着色器对传入的纹理进行直接读写。

ii. “乒乓渲染”，是一种常用的渲染技巧。结合 image 变量，我生成了一张很大的纹理（如若有 32*32 只鸟，则创建一张 64*64 的纹理），分为 4 份，分别存储位置数据、速度数据和第二份位置和速度数据，每次更新都是在同一张纹理的不同部分进行操作。

iii. 使用了扩展的 GL_RGBA32F 颜色格式，为提高计算精度。其实 16F 已经足够。

五、 实验结果

a) 总体效果

i. 实验总体效果如 Figure 8 所示。在不干扰的情况下鸟群应沿环形轨道绕中心盘旋。

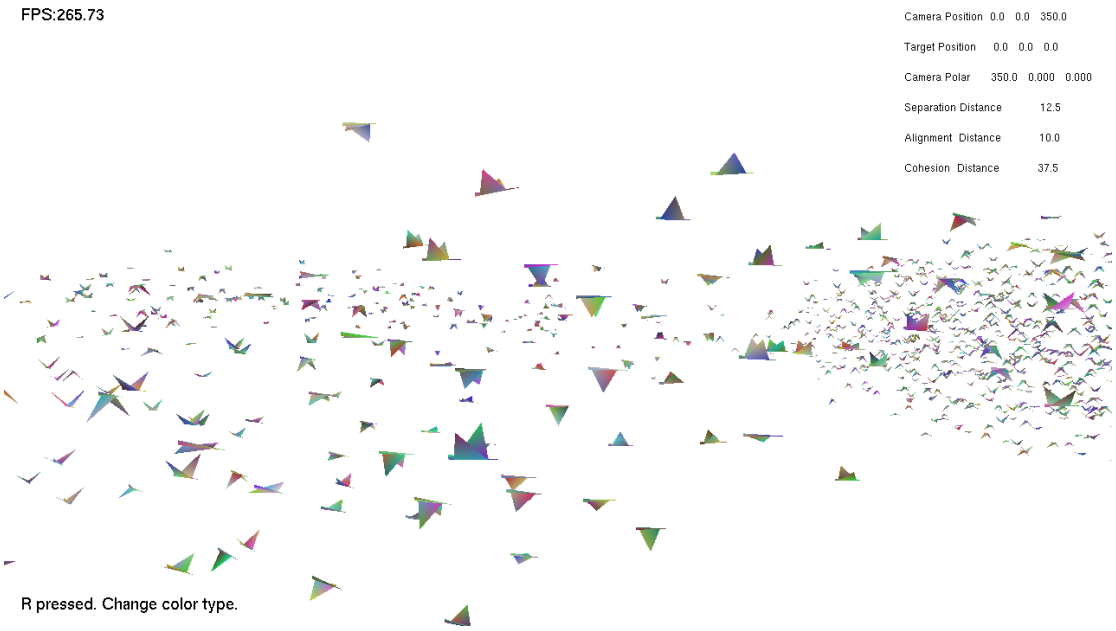


Figure 8 **总体效果**

b) 控制：暂停与恢复

i. 按下空格或将焦点切换到其他窗口时，计算会暂停。

Figure 9 **暂停与恢复**

c) 控制：干扰

i. 鼠标移入窗口，对鸟群产生干扰。

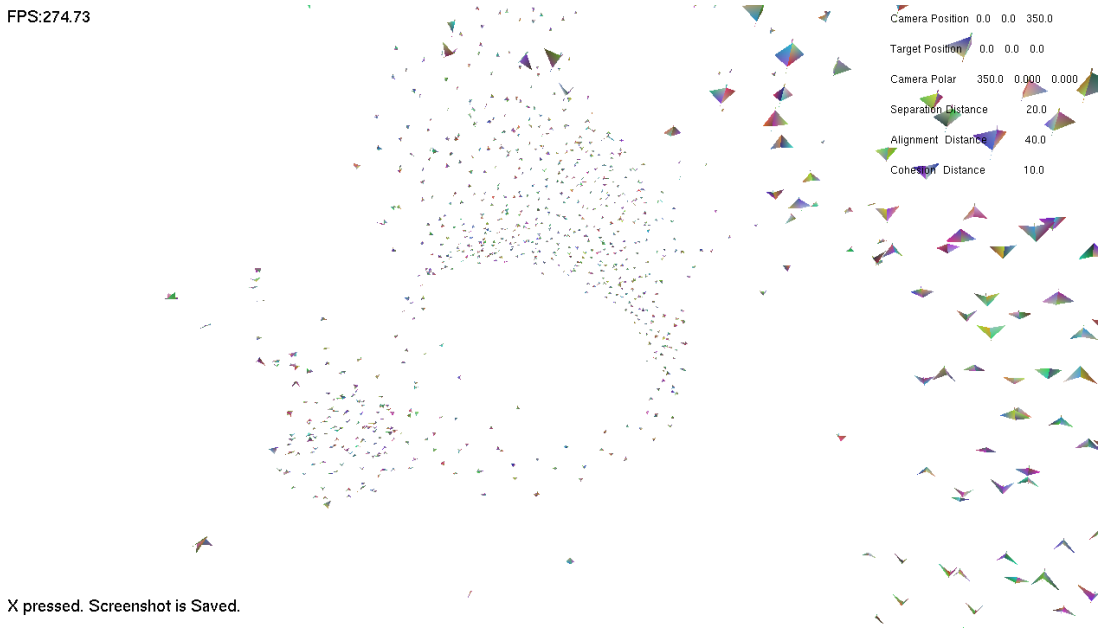


Figure 10“捕食者”干扰

d) 观察：改变颜色

i. 按下 R 键，可以切换鸟个体的颜色为随机颜色还是根据速度方向创建的颜色：若速度在 Z 轴的分量指向+Z，则颜色偏向红色，否则偏向蓝色。

Figure 11 改变颜色

e) 观察：改变视角

使用 WSADQE 键进行环绕和缩放观察，操作起来需要一段时间适应。按 C 进入第一人称漫游模式。再次按 C 恢复初始视角。请注意任何细小的视角操作都会影响(3)和(4)的效果，请尽可能在初始视角中使用以上功能。

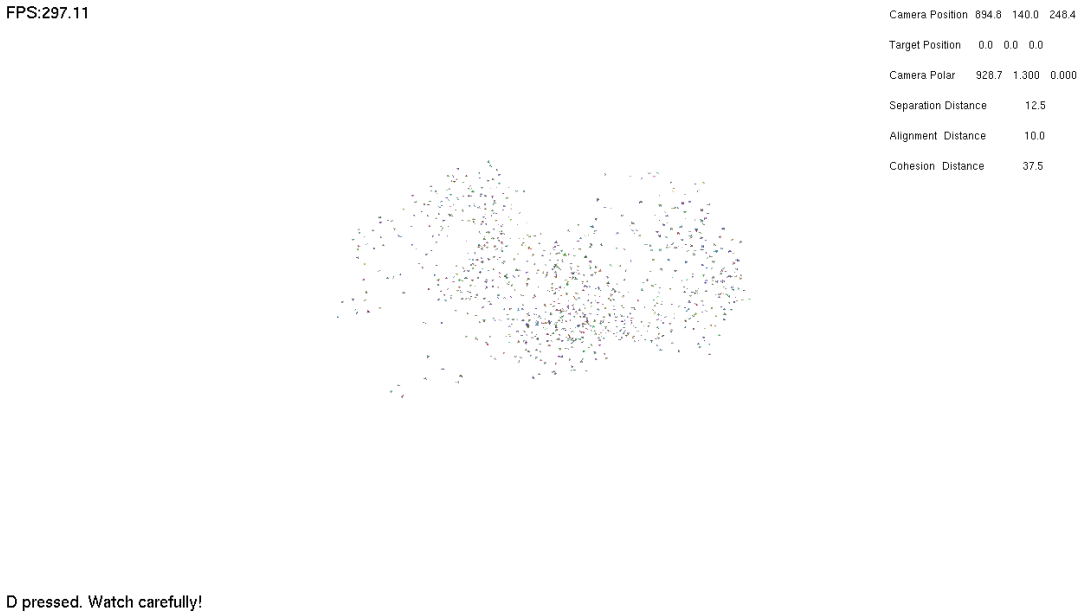


Figure 12 **改变视角**

f) 观察：截图

按 X 键保存当前窗口截图于运行目录。

Figure 13 **截图**

g) 参数：创建鸟的数量

可以创建的鸟的数量需为 2 的指数级的平方只，如输入 32，则会创建 32 * 32 = 1024 只。

```
Enter the base number of birds you want to create (like 2, 4, 8, 16 or 32)
32
Your decision: 32.
Framebuffer is complete
Focus is on this window.
Mouse moves to (906, 162).
Mouse moves to (906, 162).
Mouse moves to (906, 162).
Mouse moves to (906, 161).
Mouse moves to (881, 100).
Mouse moves to (869, 77).
Mouse moves to (854, 48).
Mouse moves to (835, 21).
Focus is on other window.
```

Figure 14 更改鸟的数量

h) 参数：Seperation Distance

使用方向键的上↑和下↓控制 Seperation Distance 的大小。更大的值使得个体间的间距增加。

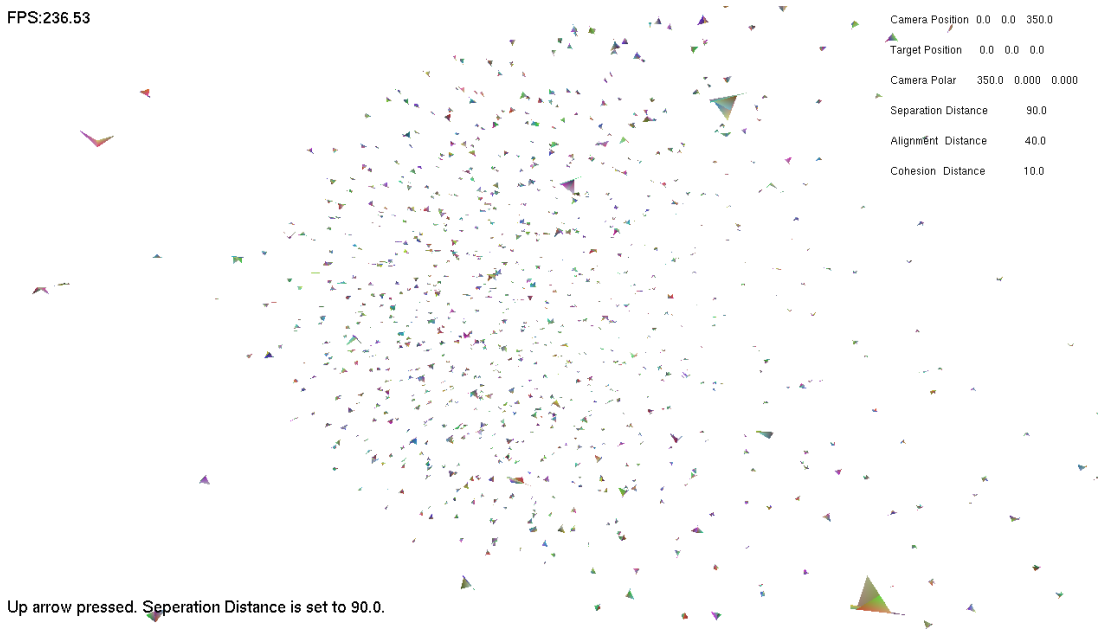


Figure 15 Seperation Distance

i) 参数：Alignment Distance

使用方向键的左←和右→控制 Alignment Distance 的大小。更大的值使得群体的方向统一性增加。

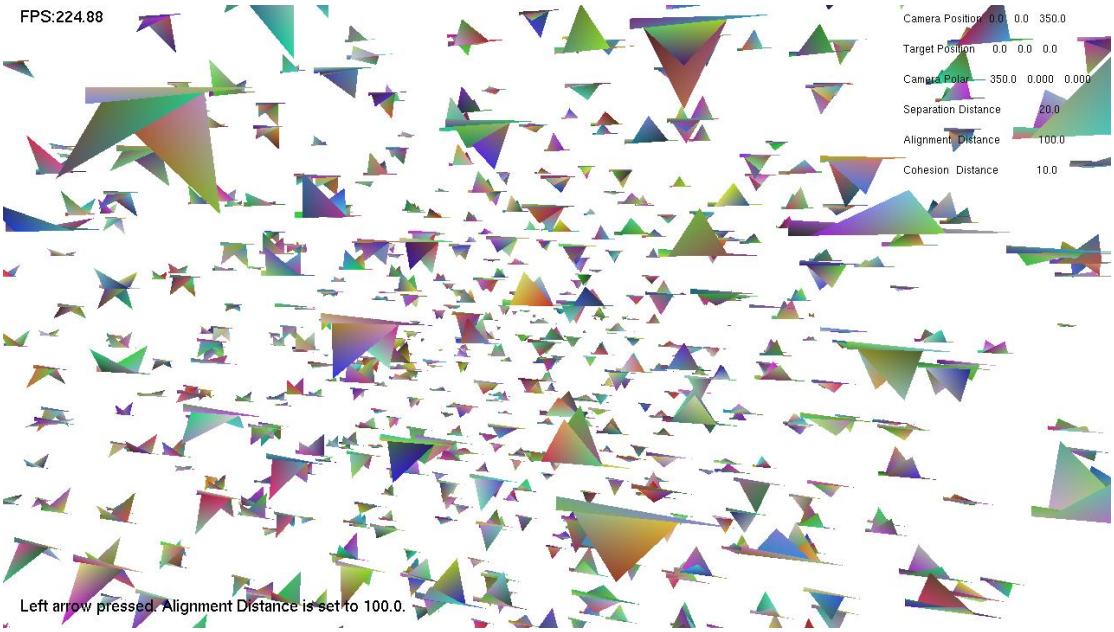


Figure 16 Alignment Distance

j) 参数：Cohesion Distance

使用 Home 和 End 键控制 Cohesion Distance 的大小，更大的值将减弱前面两者的影响。

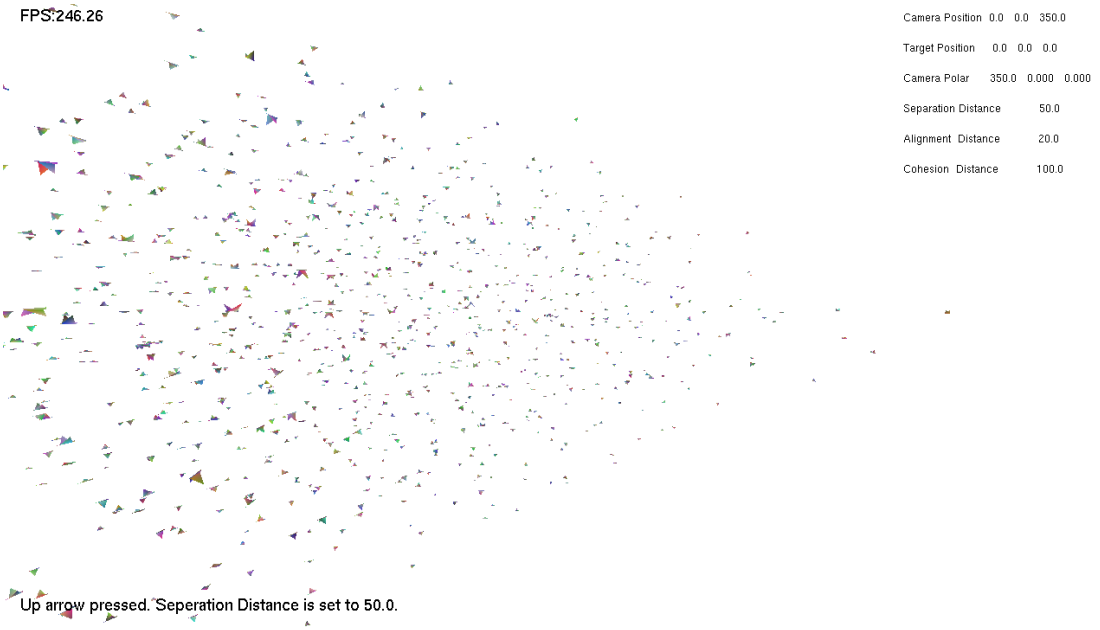


Figure 17 Cohesion Distance

六、 实验总结与心得

终于完成了，中间遇到了很多麻烦，选题一换再换，最终选择了这个。看上去比较有趣，利用 GPU 来做计算听起来也很棒，就是太难调试了。考试周太忙没时间仔细解释使用的方法，但程序都是自己一行一行组建起来的。

如有疑问，可以在考试周结束后联系我：********** / yunzhe@zju.edu.cn。