Opt-FruitNinja：基于点线对偶算法的最优切水果游戏

王晗 2016213574　　姜流 2016311961　　刘英杰 2016213644

摘要

本次实验，我们搭建最优切水果游戏系统：Opt-FruitNinja，游戏系统由前端和后台构成，在游戏的每一帧，前端界面上随机地弹射出一系列的水果，水果用线段表示它们的骨架，这些线段随即被传给后台，利用点线对偶和区域求交算法，计算出与这些线段相交次数最多的直线，该直线反馈给前端，作为对水果进行切割的路径。从切出最多数量水果的意义来说，我们的后台算法计算出的直线是最优的。此外，为了测试后台算法的正确性和时间复杂度，我们采用不同规模的模拟线段集对算法进行测试，测试结果表明我们的算法可以在 20.4 秒内计算出 1024 条平行线段集的结果，以及 43.9 秒时间内计算出 1024 条非平行线段集的结果。

关键词

点线对偶　区域求交　凸包　实时交互

1. 简介

水果忍者是一款运行在 iOS、Windows Phone、Android、塞班等平台基于触摸屏的移动设备休闲游戏。此游戏由于耐玩性高，一直位于苹果 App Store 畅销的前十名，它在 2010 年风靡全球，在地铁，广场，大街小巷上随处可见水果忍者的玩家。在游戏当中，画面里会随机地弹射出一系列的水果，玩家尽可能砍掉所有的水果，就可以完成游戏规定的任务。如果玩家可以一刀砍下画面当中一连串的水果，则会有额外的奖励。

受此游戏的启发，我们想利用计算几何算法，搭建一个最优切水果游戏系统，Opt-FruitNinja (Optimal Fruit Ninja)，即用算法推荐一条穿过水果最多的直线，玩家只需采纳系统推荐的直线，便可以获得一刀切最高的分数。基于此想法，我们用线段来表示水果的骨架，这样，多个水果就可以用多条线段来表示，然后我们采用基于点线对偶的算法求出与这多条线段相交最多的直线，该直线作为结果反馈到游戏的前端界面上，此时沿着该直线进行一刀切就可以获得对当下出现的水果进行切割的最高分数。

为了说明我们基于点线对偶的算法的普适性和高效性，我们将其与基于凸包的算法进行对比，实验结果表明，我们的算法可以很好地处理线段不平行和平行的情况（基于凸包的算法只能处理线段平行的情况）。除此之外，我们在较大规模的线段数据集上对我们的算法的正确性进行测试，结果表明我们的可以在 20.4 秒内计算出 1024 条平行线段集的结果，43.9 秒时间内正确地完成 1024 条非平行线段数据集上的计算。我们还测试了游戏系统的实时性，测试结果表明我们的系统在 4 个水果下，后台返回结果的时间小于 200 毫秒，可以实时地计算出当前帧下的最优解。

2. 系统综述

Opt-FruitNinja 工作流如 Fig. 1 所示:

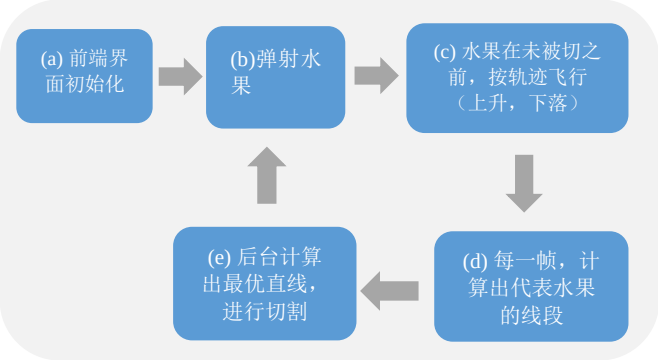

Figure 1. Opt-FruitNinja 工作流, 前端展示水果的飞行动画 (b-c)、计算代表水果的线段 (d)，后台负责计算最优直线 (e) 。

Opt-FruitNinja 系统界面如 Fig. 2 所示：

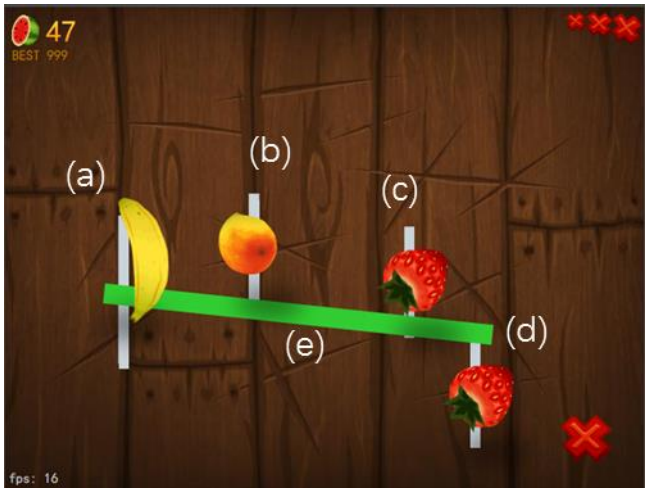

Figure 2. Opt-FruitNinja 界面, (a)-(d):代表水果的线段, (e): 与(a)-(d)线段相交最多的直线。

3. 几何问题描述

如 Fig. 3 所示， 在一个二维平面上，给定n条线段的集合 $\{s_1, s_2, \ldots, s_n\}$， 每个线段 l_i 由两个端点表示， s_i: $\{(x_{i1}, y_{i1}), (x_{i2}, y_{i2})\}$, 求出一条与给定线段集相交最多的直线 l_{opt}: $ax + by + c = 0$。

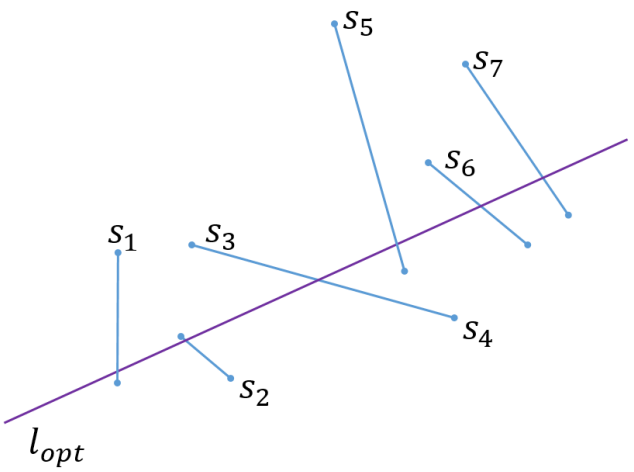

Figure 3. 7 条线段$(s_1, s_2, \dots, s_7)$以及与线段集合相交次数最多的直线l_{opt}

4. 基于凸包的算法

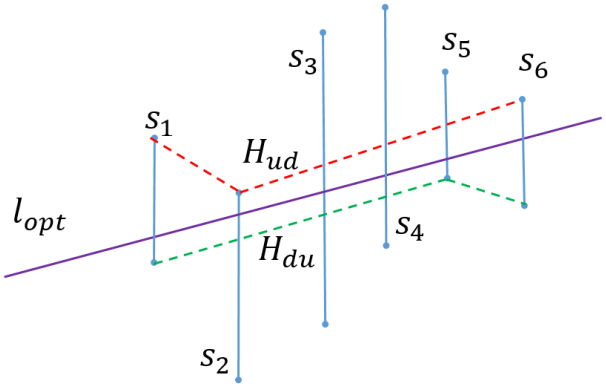

Figure 4. 基于凸包的算法示意图，H_{ud}是上端点集合的下凸壳，H_{du}是下端点集合的上凸壳，上凸壳以下和下凸壳以上的任意一条直线即为最优直线l_{opt}。

我们首先尝试基于凸包的算法求解最优的直线。此算法适用于给定线段是平行的情况，（这些平行的线段如果不与 x 轴垂直，则通过旋转一定的角度将所有的线段旋转至与 x 轴垂直），则线段s_i可以表示为$\{(x_i, y_{i1}), (x_i, y_{i2})\}$，不失一般性，假设$y_{i1} < y_{i2}$。

算法步骤如下（见 Fig. 4）：

Step 1：对每条线段s_i的两个端点进行分类，纵坐标大的为上端点，纵坐标小的为下端点。上端点的集合记为$P_u = \{(x_1, y_{12}), (x_2, y_{22}), \dots, (x_n, y_{n2})\}$，下端点的集合记为$P_d = \{(x_1, y_{11}), (x_2, y_{21}), \dots, (x_n, y_{n1})\}$。

Step 2：运行 Graham Scan，求出P_u的下凸壳H_{ud}和P_d的上凸壳H_{du}。

Step 3: 上凸壳和下凸壳围成了一个区域（中间窄两边宽的“隧道”），只要直线保持在上凸壳以下和下凸壳以上（即从中穿过）就一定满足，所以该“隧道”即是所有最优直线的集合。从中任意选取一条直线，作为最优直线输出。

该算法基于凸包求解，Graham Scan 的复杂度是 O(n log n)，Step 3 找满足上凸壳以下和下凸壳以上的直线是 O(n log n)（二分查找），然而在实际实现中，Step 3 用复杂度为$O(n^2)$的简单搜索算法即可满足要求。

我们尝试着将该算法迁移到处理非平行线段的情况，此时如果仍然沿用平行线段的情况时上下端点的定义方式是不妥的，比如有以下反例（Fig. 5）：如果沿用平行线段的情况时上下端点的定义方式，分别求出上下端点集合的下凸壳和上凸壳，此时上下凸壳是相交的，所以算法会判定没有直线与给定的线段都相交，但是，很明显最优直线l_{opt}是存在的，因此算法判断错误。

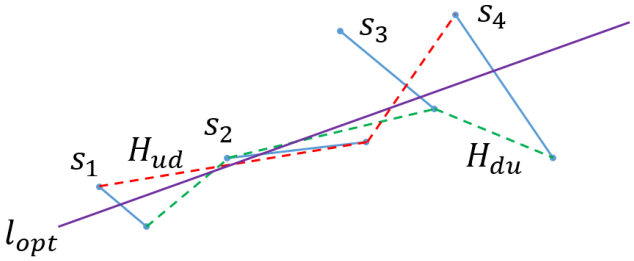

Figure 5. 反例示意图：基于凸包的算法无法直接迁移处理非平行线段。

因此如果想将该算法迁移到处理非平行线段的情况，必须仔细考虑上下端点的定义方式，比如上下端点的定义会和线段的斜率大小和方向有关。进一步的讨论已经超出本次实验的范围，在未来的工作里我们会探讨这一算法的在非平行线段的情况上的推广。

5. 基于点线对偶的算法

为了找到一个通用的（可以处理平行、不平行线段集）且高效的（时间复杂度低）的算法，我们借助点线对偶思想，将每一条线段对偶成区域，在对偶空间上找到各区域公共相交的部分，再将这些区域中的点对偶成直线，就可以找到与各线段都相交的最优直线。因此，基于点线对偶的算法步骤如下：

Step 1：对于给定的每一条线段s_i：$\{(x_{i1}, y_{i1}), (x_{i2}, y_{i2})\}$，将线段上的每一点对偶成直线，即$(x_0, y_0) \to y = x_0 x - y_0$，从宏观上，线段被对偶成了一个双翼（Double Wedge）或者条带（Gap），如 Fig. 6-7 所示：

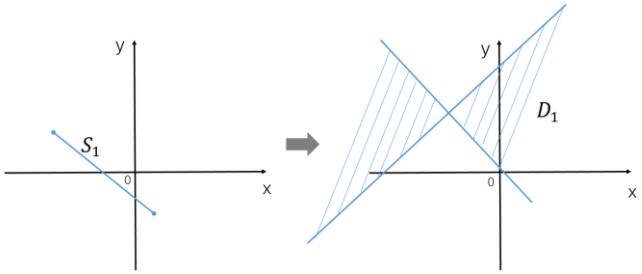

Figure 6. 斜率存在时，线段S_1被对偶为双翼D_1

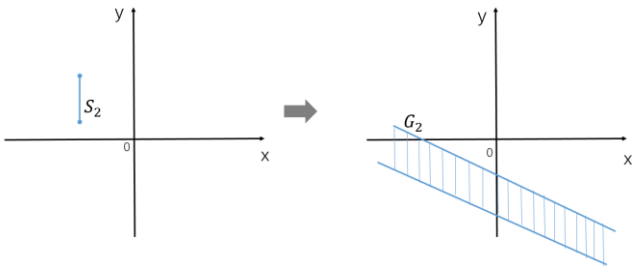

Figure 7. 斜率不存在时，线段 S_2 被对偶为条带 G_2

Step 2：如 Fig. 8 所示，在对偶平面上，我们采用 Sweep-Line 算法对多个双翼或者条带区域进行求交。

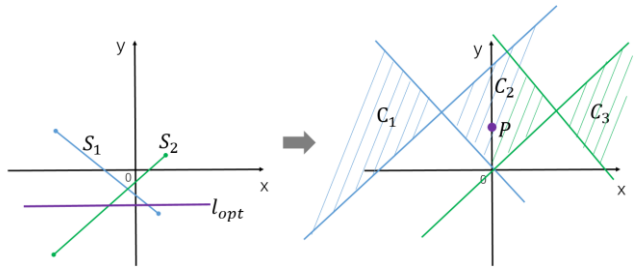

Figure 8. 右图蓝色双翼代表S_1的对偶区域，绿色代表S_2的对偶区域，两个对偶区域上求交得到 C_1，C_2，C_3，在C_2上取一点P，将P对偶回原平面得到左图的l_{opt}

Step 3：如 Fig. 8 所示，在对偶平面上，得到公共相交区域后，我们在该区域任取一点，再将该点对偶回原平面，即可得到最优的直线。

下面我们讨论各个 Step 的细节，分别介绍平行线段集的对偶、非平行线段集的对偶、区域求交。

5.1 平行线段集的对偶

如 Fig. 9 所示，当给定的平行线段不是与 x 轴垂直，此时对偶到对偶平面是一系列中心点横坐标相同的双翼。此时，为了降低对偶区域求交的复杂度，我们可以预先将所有的平行线段旋转到与 x 轴垂直，再对偶，此时在对偶平面上就是一系列规则的条带的 （Fig. 10），对这些条带区域进行求交是容易的，得到相交区域后并求出此时的最优直线，再将该直线按原来旋转的角度反着旋转即得到对于原线段集合最优的直线。

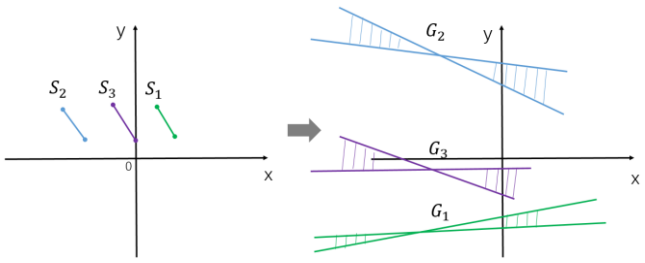

Figure 9. 平行但不垂直于 x 轴的线段的对偶

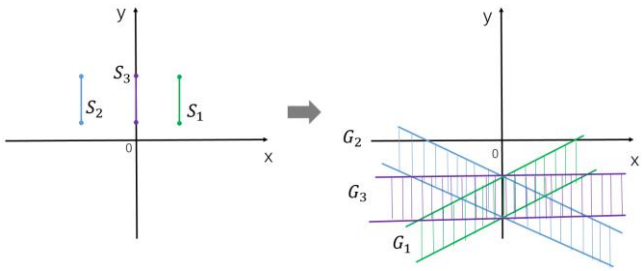

Figure 10. 平行且垂直于 x 轴的线段的对偶

5.2 非平行线段集的对偶

如 Fig. 11 所示，非平行线段集合的对偶是包含一系列双翼和条带的的区域。注意到，此时这些区域相交出来都是一系列凸包。

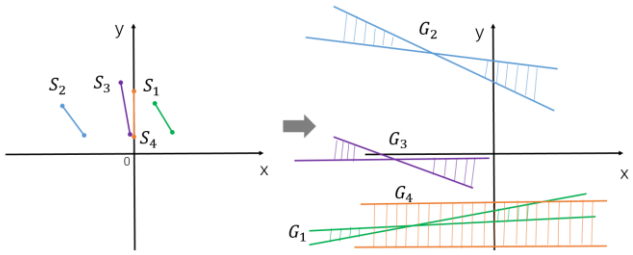

Figure 11. 非平行线段的对偶

5.3 区域求交

区域求交部分我们使用了几何求交的方法来实现。由于 Double Wedge 围成的区域是封闭的, 我们并不能直接使用现有的多边形求交算法。在课程第二章的最后, 老师曾经提到过这种不封闭的多边形求交实际上与封闭的多边形求交类似, 只要将原有的封闭多边形求交算法做一些改动就可以直接适用。

随后我们查找了相关资料, 发现可以通过添加包围盒的方法将不封闭的多边形剪切为封闭的多边形, 这种剪切需要保证原有多边形的端点必须落在包围盒内部。我们通过学习 CGAL 提供的算法库和相关论文, 我们发现了一种更适用不封闭区域求交的算法, 也就是 CGAL 中提供的 Nef Polyhedron 模型以及 Kurt Mehlhorn 和 Michael Seel 提出的使用 Infimaximal Frames 求交的算法。

数据结构方面，Nef Polyhedron 表示的是一系列由 Half-lane 进行二元运算(交, 并, 补)所得到的几何图形。两个 Half-plane 相交就能构成我们需要的一个 Single Wedge, 再由它们的补相交就能构成另一个 Single Wedge, 把这两个区域并起来就是我们需要表示的 Double Wedge。

算法方面, 在第二章几何求交的学习过程中我们曾学习过使用 Plane Sweeping, 这个算法非常适用于封闭图形求交。针对不封闭的图形求交, 一般也是先用包围盒将直线和射线 clip 掉, 将不封闭的情况变为封闭的情况, 然后使用 Plane Sweeping 算法进行求交。

Nef Polyhedron 中使用的也是 Plane Sweeping 算法, 但是与一般处理不封闭图形求交的做法不同, Nef Polyhedron 实现了另一种数据结构, 实现了一个可以动态变化的包围盒。

一般做法中, 包围盒一般是固定大小的, 这样的做法有一些缺陷, 比如为了使包围盒包围所有几何区域, 需要给定了输入之后才能确定包围盒的大小, 另外, 不同大小的包围盒计算的时候需要统一大小, 另外, 包围盒太大会导致在包围盒边界被 clip 掉的射线/直线会丢失精度。

Nef Polyhedron 算法实现了 Infimaximal Frames 是一个动态变化的包围盒, 这样就能解决上述的一些问题, 在实现过程中, 假设有一个无限大的包围盒包围着无限几何图形, 射线/直线都会在无限远出被 clip, 这个包围盒是一个无限大的方形, 由四个顶点 NW(-R,R), NE(R,R), SE(R,-R), SW(-R,-R)围成, 其中 R 代表无穷大，随后, Infimaximal Frames 针对点和线都进行了扩展定义，点被分为两类, standard point 和 non-standard point，standard point 指的是在无限大包围盒内部的点, non-standard point 指的是在无限大包围盒上的点，由点的定义可以引出线段/射线/直线的扩展定义, 线段由两个 standard point 构成, 射线由一个 standard point 和一个 non-standard point 构成, 直线由两个 non-standard point 构成。线也被分为两类, standard segment 和 non-standard segment, 在包围盒边界上的线是 non-standard segment, 其他的是 standard segment。通过这些定义, 所有的线段/射线/直线都能用扩展定义的线段(extended segments)来表示, 区别就是它们的端点类型不同. 后面 Infimaximal Frames 在论文中证明, 这种扩展了定义的线段求交和普通的线段求交的算法是相同的, 都是 4 次 toLeft 测试。

通过这种线段/射线/直线统一变为扩展定义线段的方法, 原来无限的几何区域在扩展意义下就是由很多扩展定义的线段构成的区域, 也就可以使用普通线段围成区域求交的算法—Plane Sweeping 了。

算法流程如下:

Step 1：将给定的线段两个端点通过对偶的形式转换为两条直线。

Step 2：使用 Nef Polyhedron 将两条直线构造为相应的 Double Wedge 区域，该区域在 Nef Polyhedron 内存储为扩展定义的线段(extended segments)。

Step 3：使用 Plane Sweeping 算法进行求交运算。

根据 Nef Polyhedron 求交算法复杂度, 一次 Nef Polyhedron 求交的复杂度为$O(n \log n)$, N 条线段需要调用 N 次求交算法, 所以算法整体复杂度渐进意义下是$O(n^2 \log n)$。

6. 实验与评估

为了评估我们的算法性能，我们做了三组数值实验以及一组游戏系统的使用主观实验。三组数值实验证明，基于凸包的算法可以高效地处理平行线段数据集，我们的对偶算法算法在既能处理平行线段数据集又能处理非平行线段数据集，而且在平行线段数据集上的求解比非平行线段数据集上的求解快一倍以上。游戏系统的使用主观实验的结果表明，我们的 Opt-FruitNinja 系统具有实时性、最优性和趣味性。

6.1 实验环境

操作系统：Windows 10

硬件配置： Intel Core i7 CPU (3.6GHz)，8GB 内存

开发环境：后台 Visual Studio 2013 (vc12, Release, C++)，前端 node.js

依赖库： CGAL 4.10，BOOST 1.64

6.2 数值实验

基于凸包算法的时间性能 根据我们在第 4 节的讨论，基于凸包的算法只能处理平行线段集, 我们在不同规模线段数据集（相互平行垂直于 x 轴）上测试我们基于凸包算法的时间性能，结果如 Table 1 所示，可以看出虽然我们实现的是$O(n^2)$的算法，但是基于凸包的算法处理平行直线且垂直于 x 轴的线段集合是非常之高效，在 1024 条线段下用 17.8 ms 就可以完成计算。

线段数量	2^1	2^2	2^3	2^4	2^5	2^6	2^7	2^8	2^9	2^10
组 1	0	0	0	0	1	2	2	4	9	17
组 2	0	0	0	0	0	1	2	3	15	23
组 3	0	0	0	0	0	1	3	5	9	16
组 4	0	0	1	0	1	1	2	4	9	15
组 5	0	0	0	0	1	1	2	5	8	18
平均	0	0	0.2	0	0.6	1.2	2.2	4.2	10	17.8

Table 1. 基于凸包算法的时间性能, 单位（毫秒）

对偶算法在平行线段数据集上时间性能 线段平行情况下，我们的对偶算法既可以处理垂直于 x 轴的也可以处理不垂直与 x 轴的线段（通过旋转变换或者不变换），下面我们在 Table 2-4 展示处理平行线段数据集的时间性能。

线段数量	2^1	2^2	2^3	2^4	2^5	2^6	2^7	2^8	2^9	2^10
组 1	42	97	158	313	622	1291	2431	4963	9638	19364
组 2	44	80	166	307	647	1271	2584	4794	9452	19382
组 3	43	83	156	320	618	1223	2535	4758	9576	19593
组 4	42	91	155	310	608	1258	2463	4746	9687	19048
组 5	51	87	156	308	607	1249	2491	4800	9701	18864
平均	44.4	87.6	158	311	620.	1258	2501	4812	9611	19250

Table 2. 对偶算法处理平行且垂直与 x 轴线段集的时间性能, 单位（毫秒）

线段数量	2^1	2^2	2^3	2^4	2^5	2^6	2^7	2^8	2^9	2^10
组 1	41	79	158	321	620	1261	2497	5013	10210	20735
组 2	41	78	154	305	609	1328	2526	5040	10398	20301
组 3	38	79	152	308	609	1223	2423	5262	10216	20829
组 4	45	84	161	321	700	1377	2570	5068	10409	20789
组 5	53	80	152	315	618	1235	2484	4993	10119	19562
平均	43.6	80	155.4	314	631.2	1285	2500	5075	10270.	20443

Table 3. 对偶算法通过旋转变换处理平行但不垂直与 x 轴线段集的时间性能, 单位（毫秒）

线段数量	2^1	2^2	2^3	2^4	2^5	2^6	2^7	2^8	2^9	2^10
组 1	80	159	326	665	1257	2557	5143	9786	19240	38898
组 2	76	165	323	630	1242	2516	5074	10130	20086	41853
组 3	77	166	330	652	1255	2533	5075	10319	20222	47438
组 4	98	204	412	783	1627	3062	5453	11275	22689	53780
组 5	104	229	425	784	1753	3382	6735	13334	27256	50510
平均	87	185	363	703	1427	2810	5496	10969	21899	46496

Table 4. 对偶算法（不进行旋转变换）直接处理平行但不垂直与 x 轴线段集的时间性能, 单位（毫秒）

Table 2-4 结果表明我们的算法可以很好地处理平行线段集合，对于平行且垂直与 x 轴线段集合。我们的对偶算法慢与基于凸包的算法，这是因为我们需要做$O(n \log n)$复杂度的区域求交。对于平行但不垂直与 x 轴线段，我们采用旋转变换预处理的对偶算法比不进行旋转变换的算法要快一倍，这是因为旋转变换将斜的线段旋转为垂直与 x 轴的线段，对偶过去是一个条带区域而不是 Double Wedge（内含两个区域），因此求交时会快一倍。

对偶算法在非平行线段数据集上时间性能 Table 5 展示了对偶算法处理不平行线段集的时间性能，与 Table 4 对比，可知对偶算法（不进行旋转变换）直接处理平行但不垂直与 x 轴线段集的时间性能与处理不平行线段集的时间性能相似，可以在 43.9 秒时间内计算出 1024 条非平行线段集的结果。

线段数量	2^1	2^2	2^3	2^4	2^5	2^6	2^7	2^8	2^9	2^10
组 1	86	167	341	622	1371	2696	5386	11217	22375	41597
组 2	91	191	363	695	1402	2848	5570	10614	21813	45973
组 3	86	152	348	707	1373	2662	5449	10744	21427	43167
组 4	86	173	355	649	1553	2819	5435	11013	21398	44245
组 5	93	174	353	704	1381	2854	5403	10651	21327	44316
平均	88.4	171	352	675	1416	2776	5449	10848	21668	43860

Table 5. 对偶算法处理不平行线段集的时间性能,

单位（毫秒）

6.3 游戏系统的使用研究

我们让不同的用户使用我们的游戏系统，用户反馈为 Opt-FruitNinja 趣味性强，操作简单，并且系统时延低，可以在人难以判断一刀切的方向时有效地推荐出最优直线，获得最高分数。Fig. 12-14 是一些玩家在使用过程的截图。图中白色的线段代表水果，绿色的线段表示可以一次切掉所有水果的线段。游戏可以支持平行模式（代表水果的线段是平行的）和任意模式（代表水果的线段是非平行的）。

Figure 12. Opt-FruitNinja 启动界面

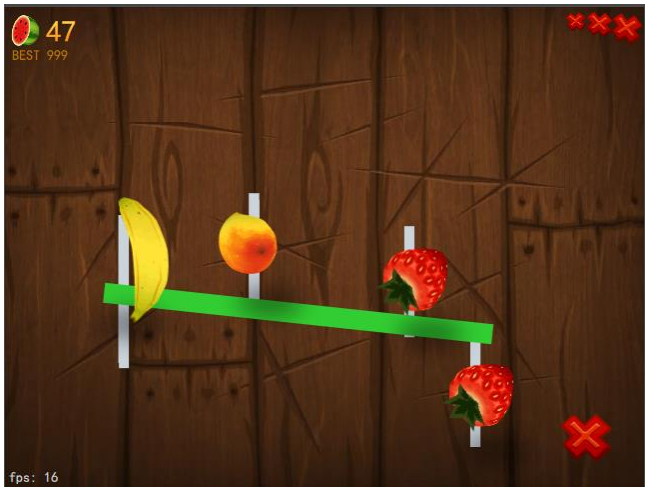

Figure 13. Opt-FruitNinja 游戏进行中，后台推荐出最优直线

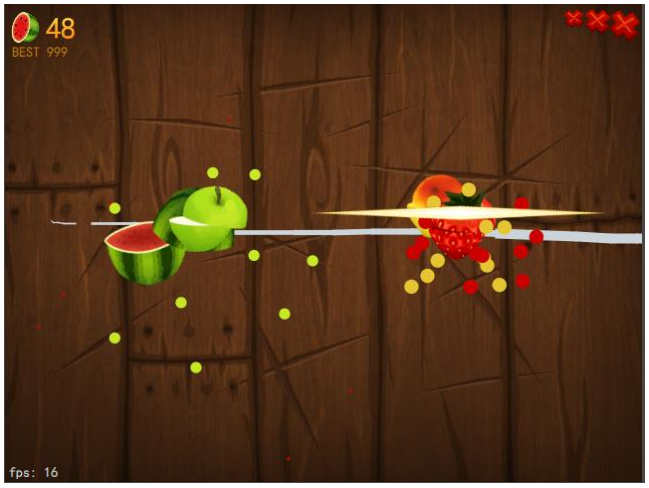

Figure 14. Opt-FruitNinja 游戏进行中，玩家一刀切碎所有水果

7. 讨论

本小节我们对算法的局限性以及可能改进的地方进行总结。

1. 平行的 case 下, 我们找到了更好的算法: 求线段上端点构成的下凸壳和线段下端点构成的上凸壳, 如果两个凸壳不相交的话就可以找到穿过所有线段的直线, 这样的算法的时间复杂度为O(nlogn)；

2.非平行 case 下，在查找资料的过程中, 我们发现 double wedge 相交构成的区域是一系列不相交的 convex polygon (有可能 unbounded), 这样就可以使用 unbounded convex polygons 的求交算法, 而不是通用的 Nef Polyhedron 求交算法；

3. Nef Polyhedron 提供了三种不同的核(Kernel)来进行求交运算 Extended_cartesian<FT>, Extended_homogeneous<RT>, 和 Filtered_extended_homogeneous<RT>. 其中运算最快的核是 Filtered_extended_homogeneous<RT>. 我们使用了 Extended_cartesian<FT>进行计算, 这对算法最终的效率产生了一些影响。

8. 参考文献

[1] Granados, Miguel, et al. "Boolean operations on 3D selective Nef complexes: Data structure, algorithms, and implementation." European Symposium on Algorithms. Springer Berlin Heidelberg, 2003.

[2] Seel, Michael. Research Report: Implementation of Planar Nef Polyhedra. MPI Informatik, Bibliothek & Dokumentation, 2001.

[3] Mehlhorn, Kurt, and Michael Seel. "Infimaximal frames: A technique for making lines look like segments." International Journal of Computational Geometry & Applications 13.03 (2003): 241-255.

[4] Mehlhorn, Kurt, and Stefan Näher. LEDA: a platform for combinatorial and geometric computing. Cambridge university press, 1999.

[5] De Berg, Mark, et al. "Computational geometry." Computational geometry. Springer Berlin Heidelberg, 2000. 1-17.

[6]http://www.cs.uu.nl/docs/vakken/ga/slides8.pdf

[7]http://cgm.cs.mcgill.ca/~athens/cs507/Projects/2001/DG/CompGeom/Index_files/geom_preliminaries.htm