计　算　几　何

大作业

三维凸包算法的实现及其动画演示

实验报告

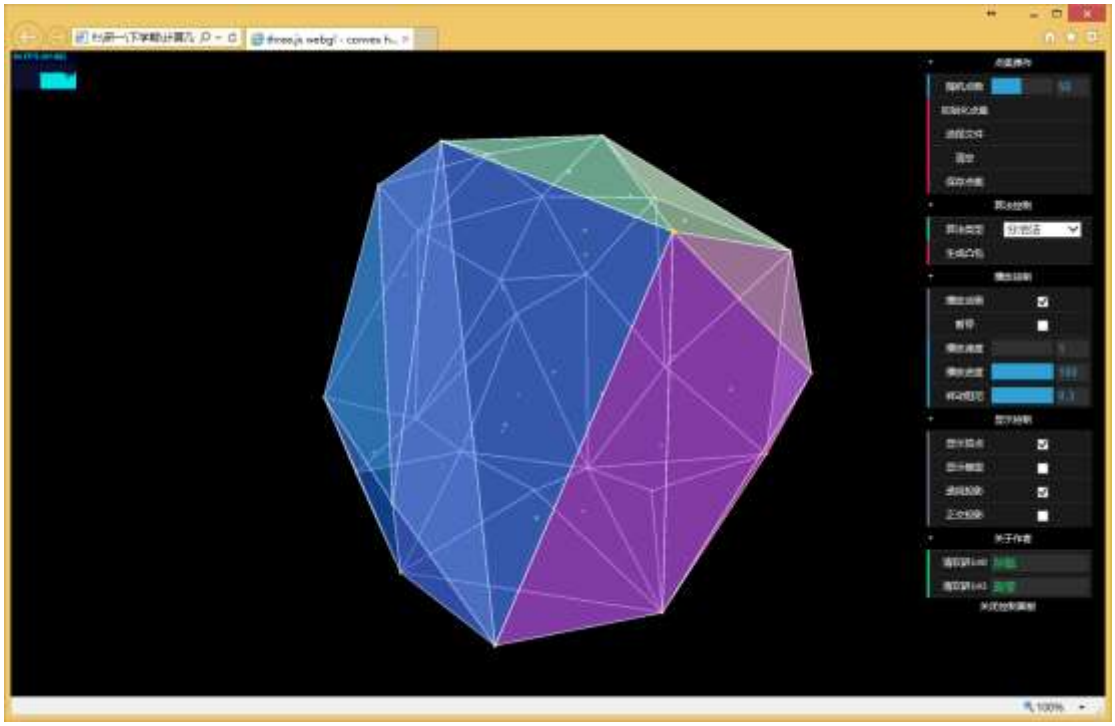
小组成员：

软件学院　　2014213460 孙 聪

软件学院　　2014213517 高 莹

目 录

1 程序演示说明



本次实验采用 js 和 html 编写。除了打开保存文件功能（需配置服务器方能正常执行）之外，其余功能均可通过直接使用浏览器打开 convex_demo.html 文件得以体现。

注：因使用 webgl 技术，所以请使用各主流浏览器的较新版本的浏览器浏览。否则，可能出现无法正常显示与执行的情况。经测试，在较新版本的 IE 及 Chrome 浏览器下表现正常。

无论实在透视投影还是正交投影下，鼠标左键拖动为旋转几何体；右键拖动为移动几何体；鼠标滚轮控制视点的远近。

1.1 网页服务器配置及项目运行说明

本项目在不同操作系统下，只需正确安装启动及设置 Web 服务器后，所有功能均可正常运行。

1. 软件安装

1） 软件下载地址 http://www.xampps.com/

2） 服务器启动问题说明：安装完成后，打开 xampp 控制面板，启动 Apache。

① 若因端口 80 占用启动失败，则点击 Config，选择 Apache(httpd.conf)打开，修改端口。

修改“Listen 80”为“Listen 8081”；

修改“ServerName localhost:80”为“ServerName localhost:8081”；

② 若因端口 443 占用启动失败，在 Apache 配置文件 httpd.conf 中，去掉 “Include "conf/extra/httpd-ssl.conf"”

或者将占用相应端口的进程杀死，则不用修改相应的配置文件。

2. 项目运行说明

1） 将工程目录放在 xampp 安装目录/htdocs 中

2） 启动 Apache 服务器。

3） 打开较新版本的 IE 及 Chrome 浏览器浏览器

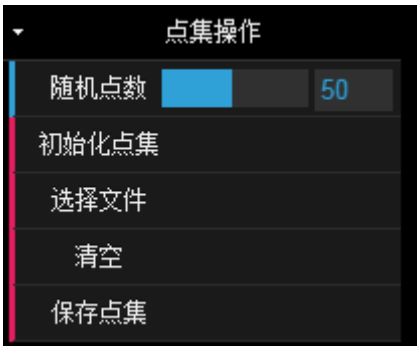
输入 localhost:80/ConvexHull/src/convex_demo.html

注：若前面修改过服务器端口，则需将 80 替换为修改的端口号

1.2 项目使用操作说明

1. 点集操作

点集控制部分可以初始化需要生成凸包的点。

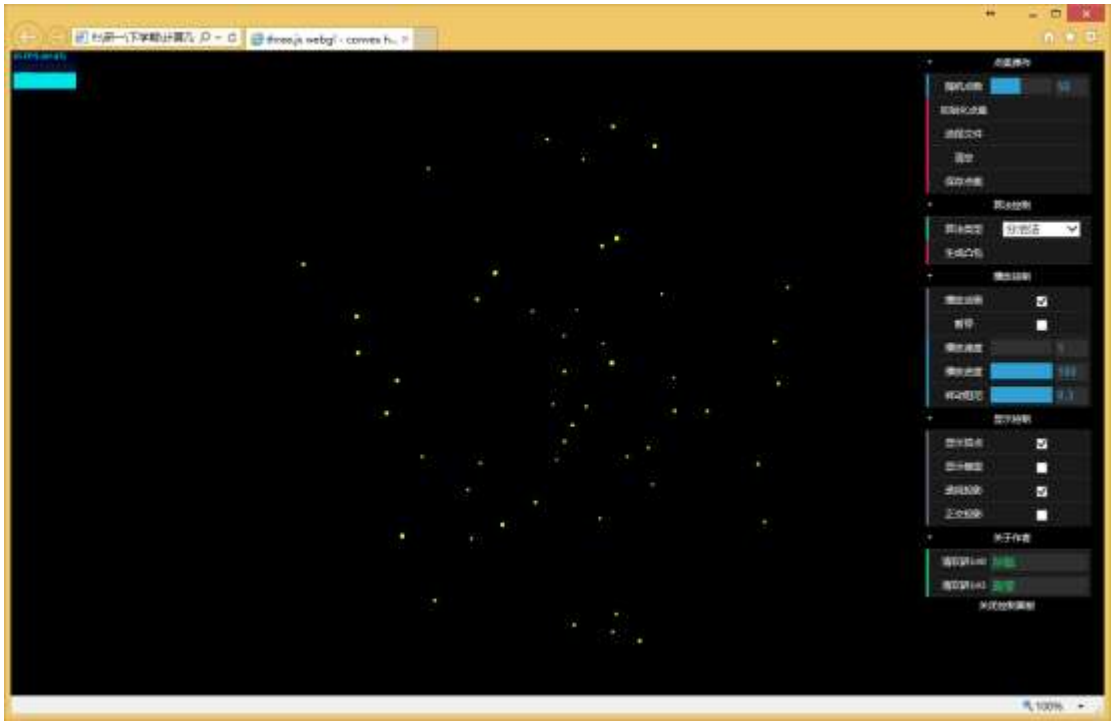


1) “随机点数”可以调节生成随机点的数目。该数目限制在 4~100 点之间。更少的点无法生成凸包；更多的点则会动画会很长，没有意义。

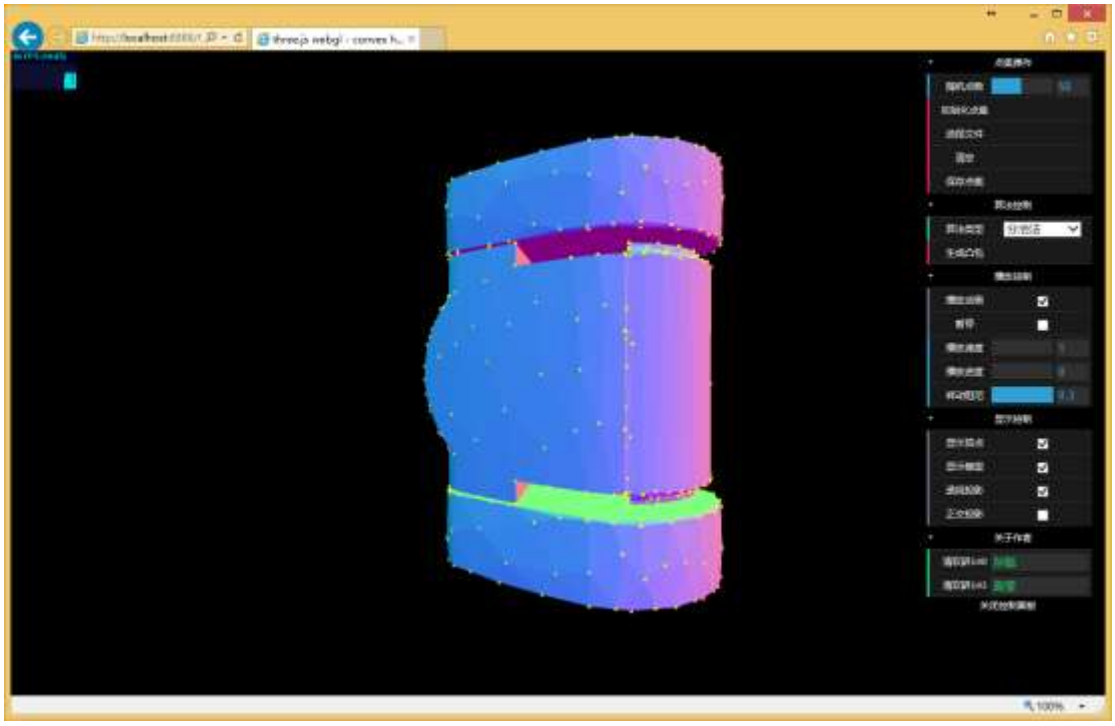
2) “初始化点集”则根据设置的随机点数生成一定量的随机点并显示。

3) “清空”可以在任意状况下使用。如果正在动画，则会中值动画。清空点集。

4) “选择文件”和“保存点集”两个按钮需要配置网页服务器才可以使用（现代主流浏览器为了安全性考虑，禁止本地网页程序操作本地文件）。“选择文件”可以选择 stl 文件或者普通数据文件，并导入当前环境作为初始点集。而“保存点集”则将当前点集信息保存下载到本地。



生成 50 个随机点的情况



载入 stl 文件的情况

2. 算法控制

1) “算法控制”可以选择当前要执行的凸包算法的类型。

2) “生成凸包”则按照选定的算法生成当前点集的凸包并演示动画。

3. 播放控制

播放控制面板可以控制动画播放的速度、进度等。

1) “播放动画”可以控制是否播放动画。当取消勾选的时候，直接显示点集的凸包；当勾选时，则动画演示凸包生成的过程。

2) “暂停”可以控制动画的播放与暂停。勾选时，动画暂停播放；取消勾选动画继续。实际上在动画进行的过程中，可以认为“播放动画”与“暂停”的功能类似，无非一个会显示最终结果以便对照，另一个会仅暂停而不现实结果。

3) “播放速度”可以控制播放的速度，以 1 最慢，10 最快。速度为 1 档的时候，大约每个动作持续 300ms。而速度为 10 档时，每个动作持续 30ms。

4) “播放进度”显示当前动画的进度，也可以通过“播放进度”快进快退或者定位到某一步。

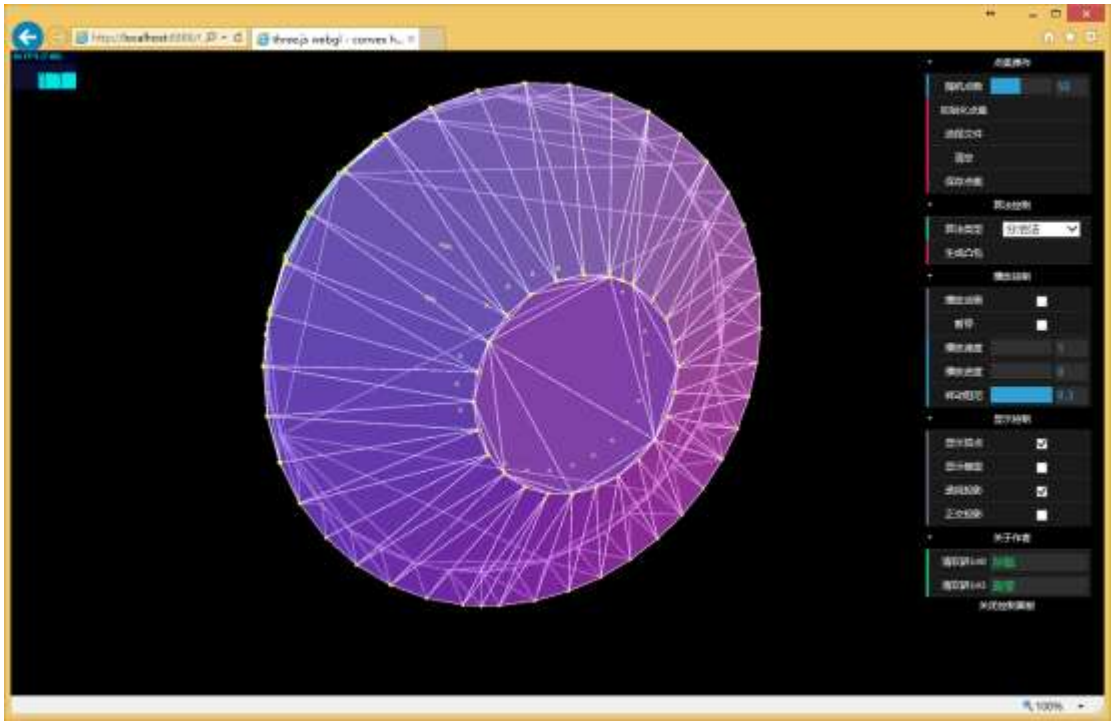
5) “转动阻尼”控制鼠标转动物体时的“惯性”。当阻尼为 0 时，任何转动操作都会让几何体不停地按照给定的方向和角度转动。随着阻尼的减小，“惯性”会减小，同样的操作几何体会停得更快。

4. 显示控制

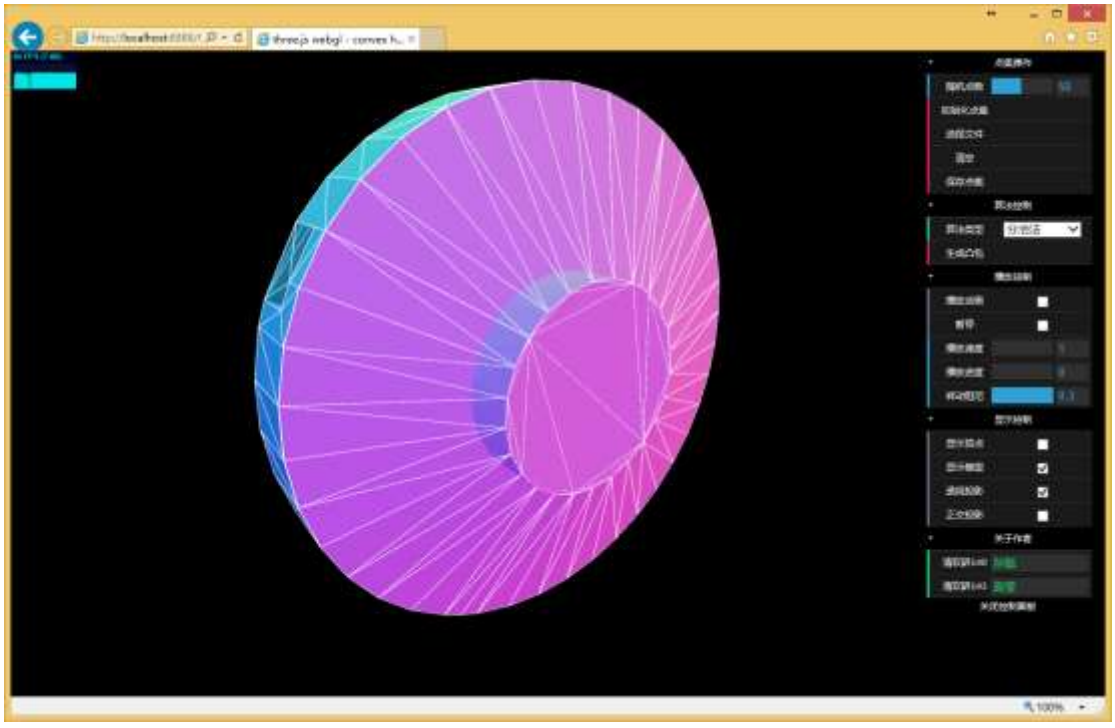
1) “显示顶点”控制顶点显示与否。勾选时会显示所有的顶点；取消勾选时则不显示顶点。

2) “显示模型”仅对载入 stl 模型时有效。勾选时显示 stl 模型本身的形状供参考；取消勾选时不显示 stl 模型本身的形状。

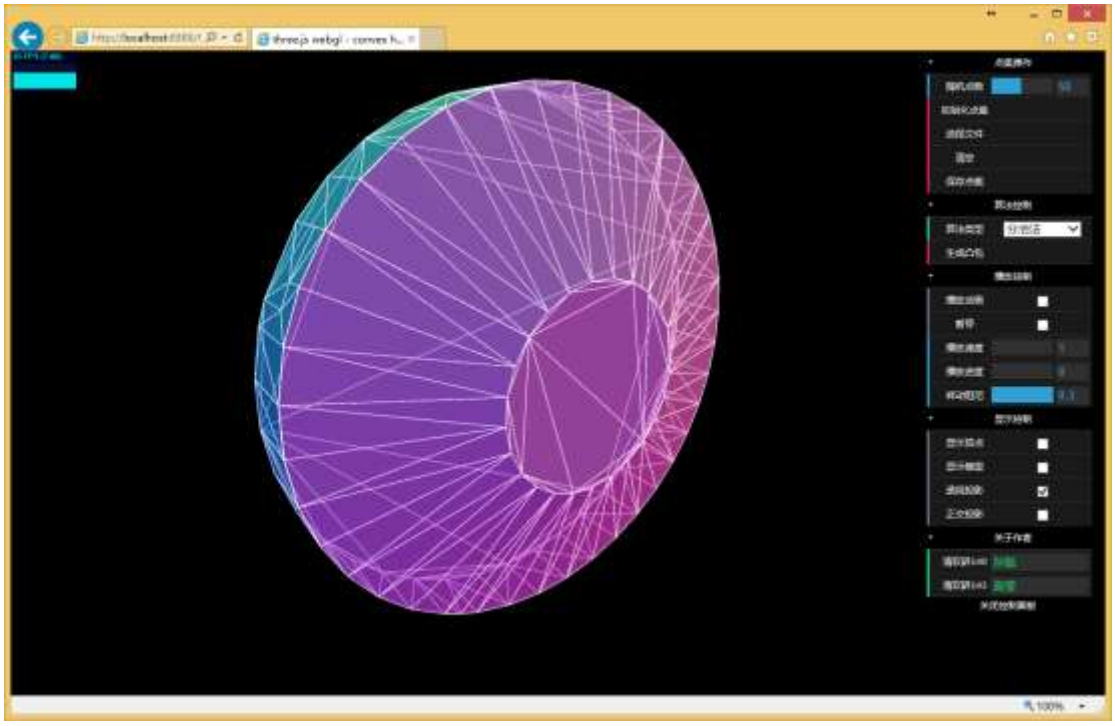
3) “透视投影”与“正交投影”二者仅能选一。选择透视投影的时候会体现透视关系，近大远小。而选择正交投影的时候不体现透视关系，远近一样大。可根据情况自由选择。



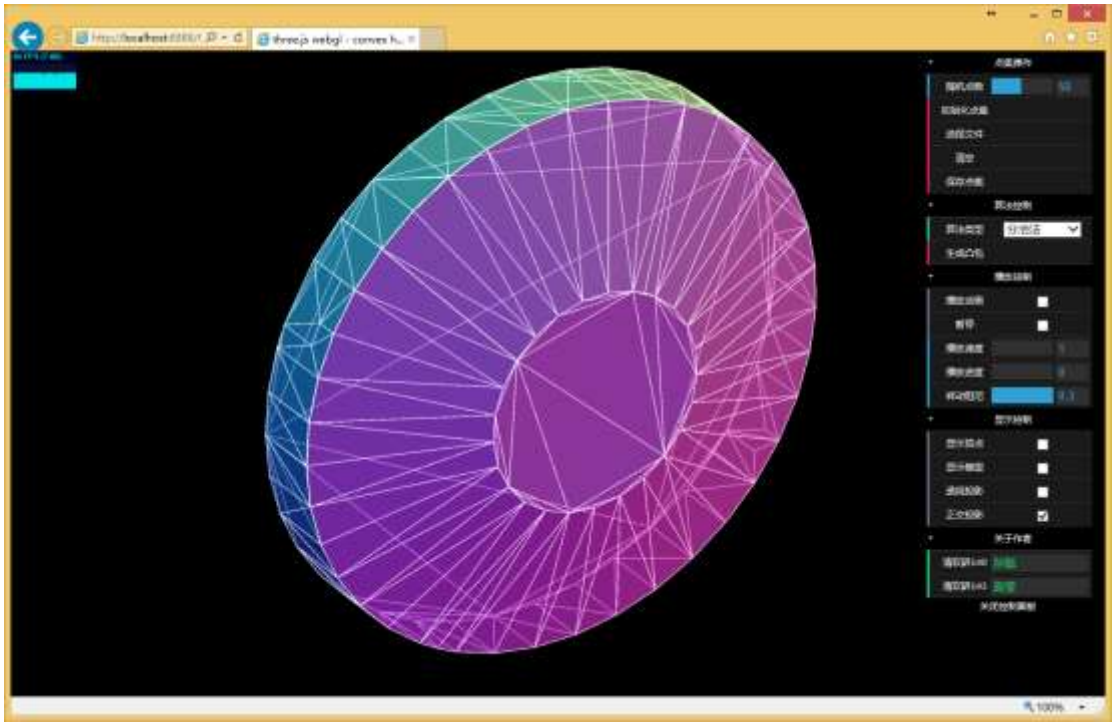
不显示模型，显示顶点



显示模型，不显示顶点。



透视投影



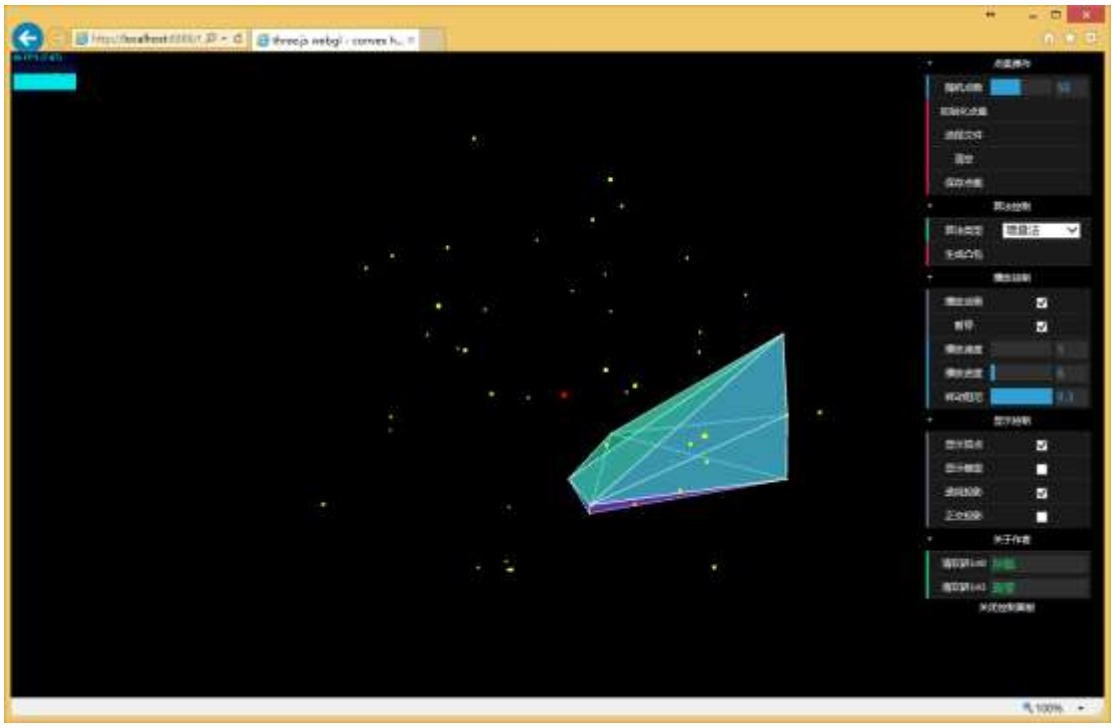
正交投影

1.3 动画说明

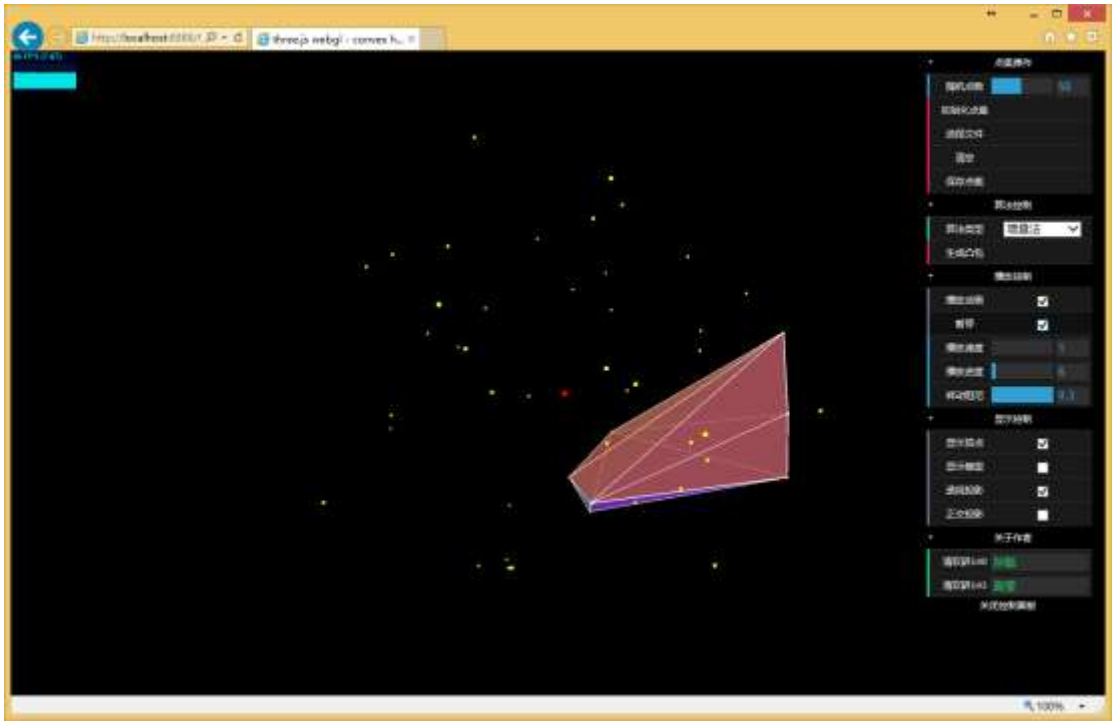
1. 增量法&冲突图法

二者凸包生成方式是类似的，仅仅是数据结构的差距造成了算法效率的不同，因此二者

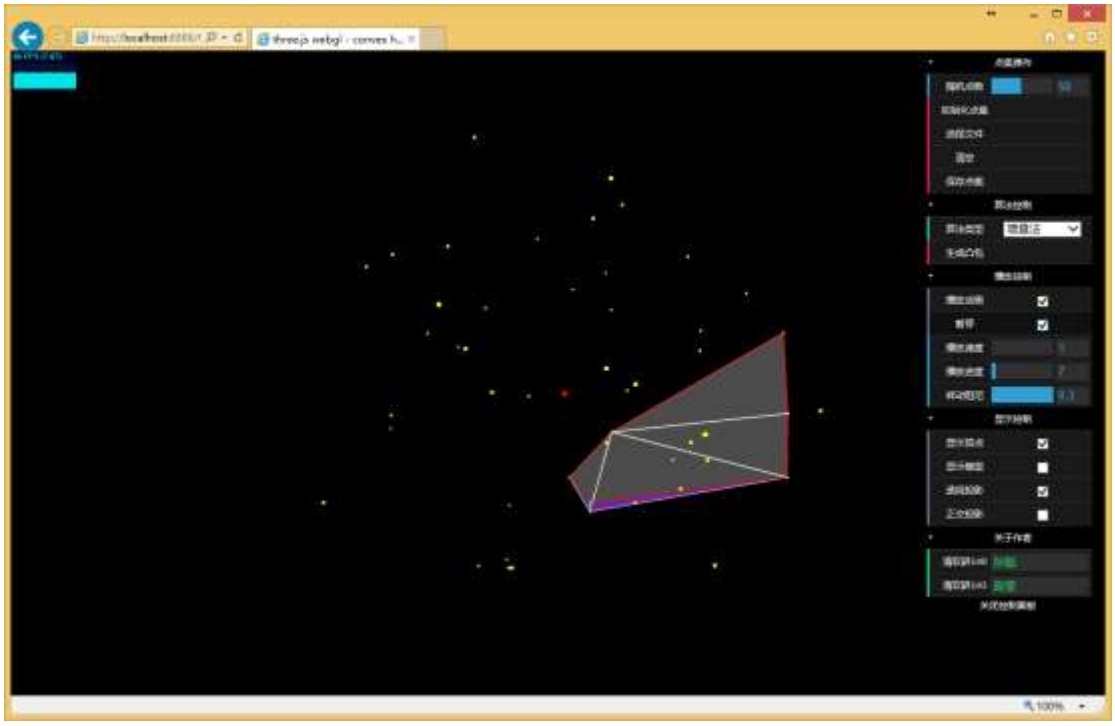
的动画演示是类似的。



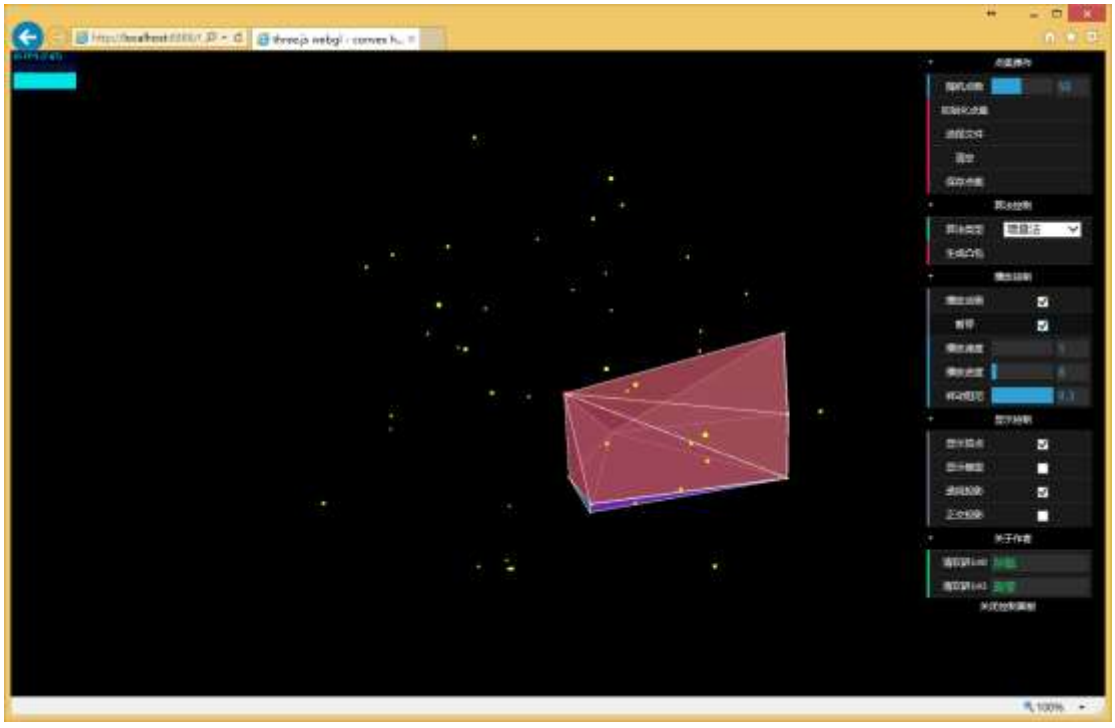
对于已经有的一个小凸包和要添加的新顶点



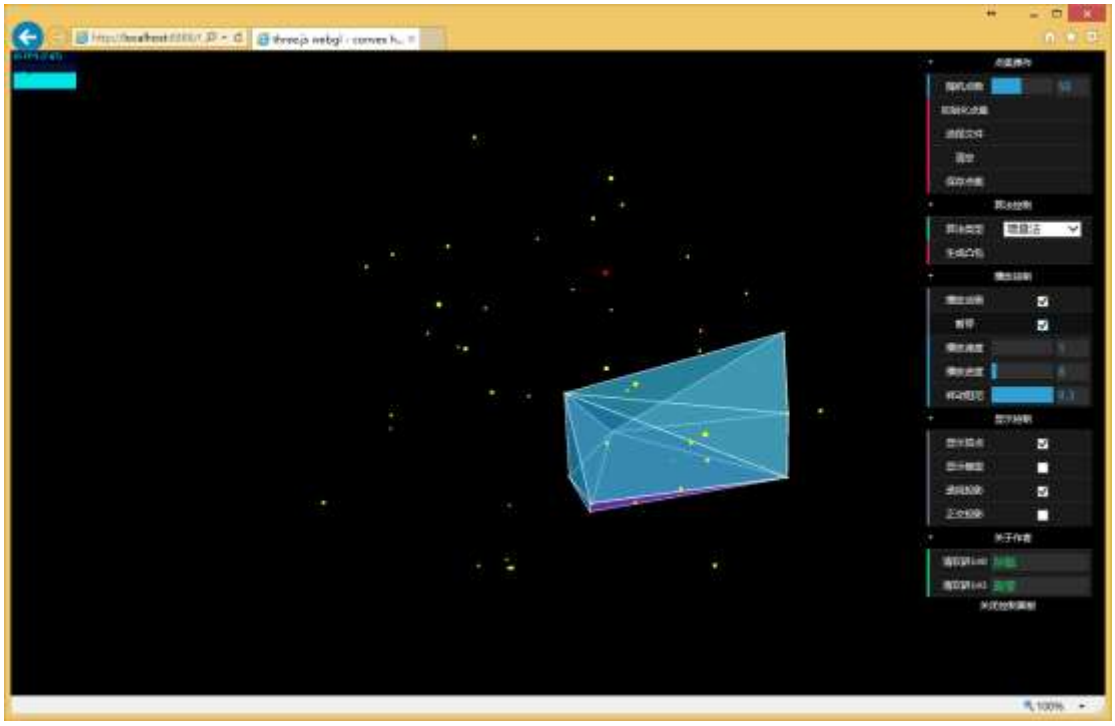
标红的面片为与当前点可见的面片



移除所有可见面片后勾出红色的边界



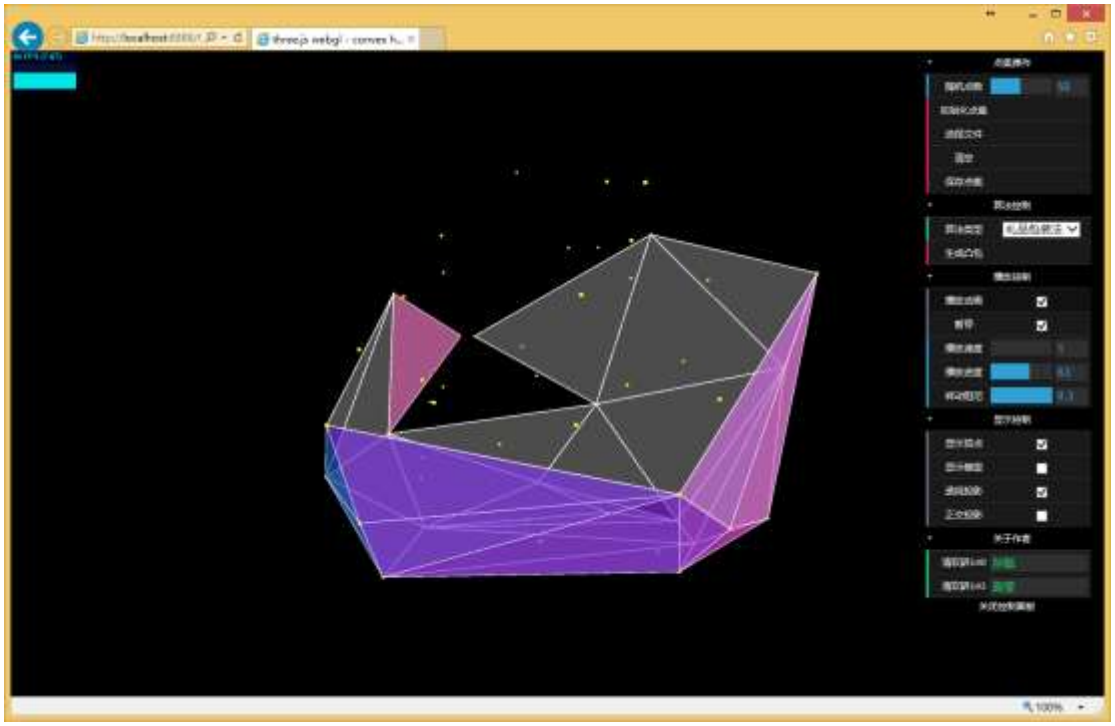
新增加的锥面用红色表示



下一轮开始

2. 礼品包装法

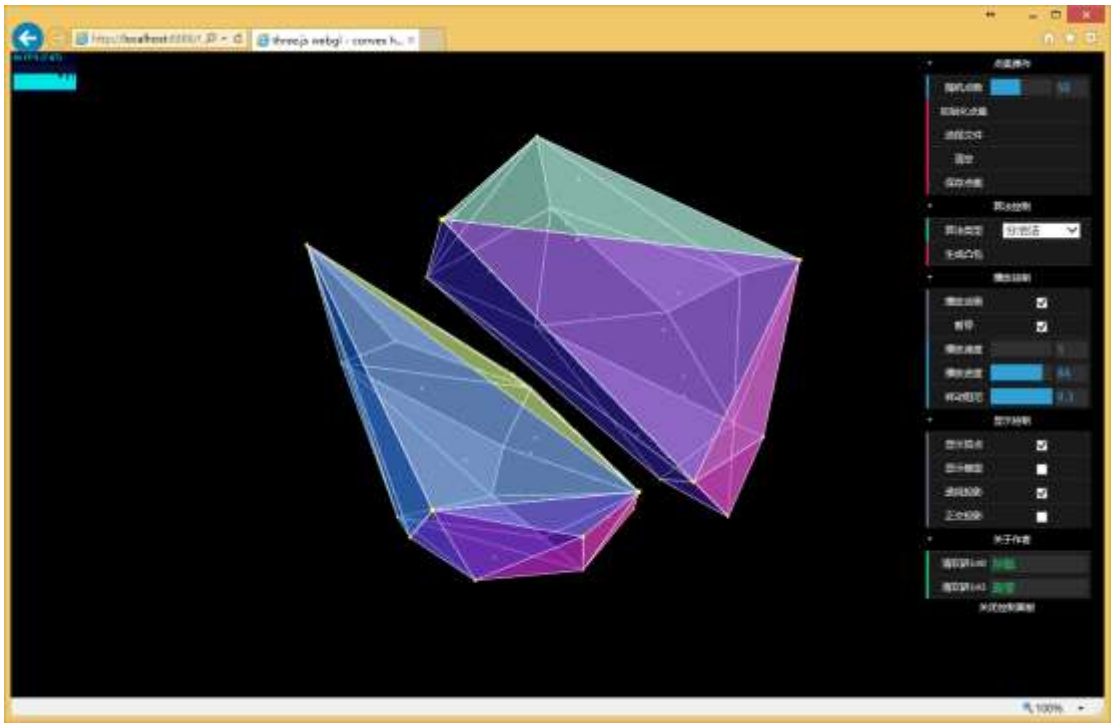
礼品包装法动画量比增量法少很多。



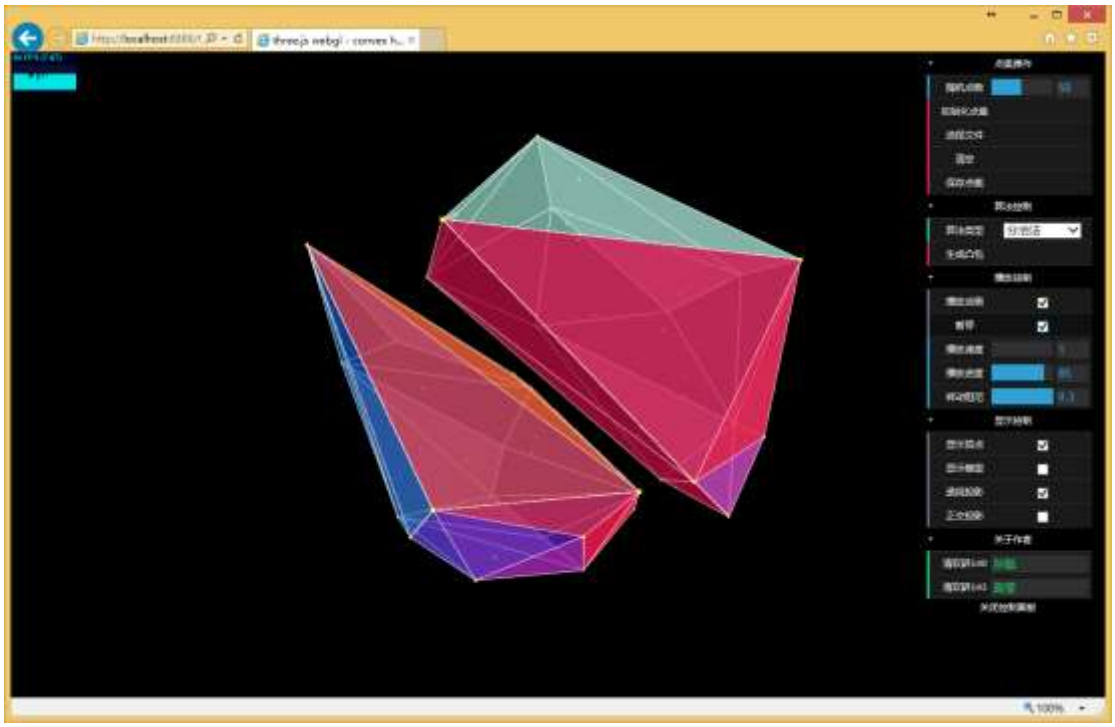
对于已经添加的面，外侧为法线纹理（彩色），内部为灰色。而将要被添加的面片则内外皆为法线纹理（彩色）。将要添加的面片从其依托面方向逐渐旋转到目标位置，变成已添加的面片。

3. 分治法

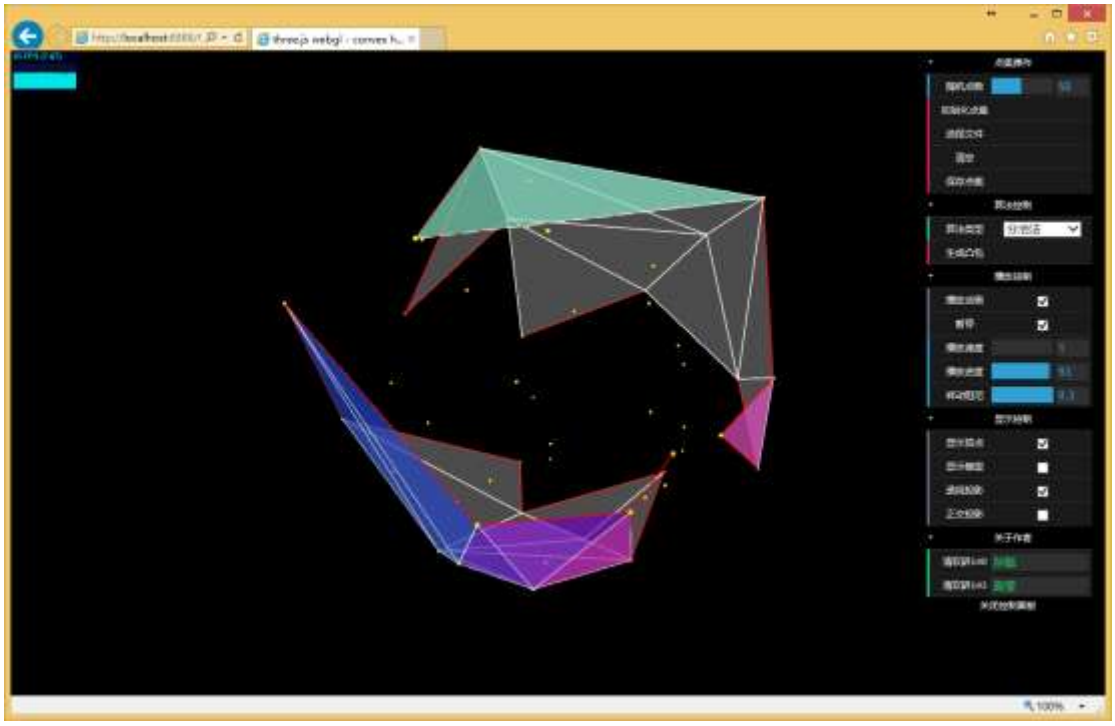
分治法的基础为增量法， 其动画演示也同增量法一致。这里中点说明合并凸包时候的情形。



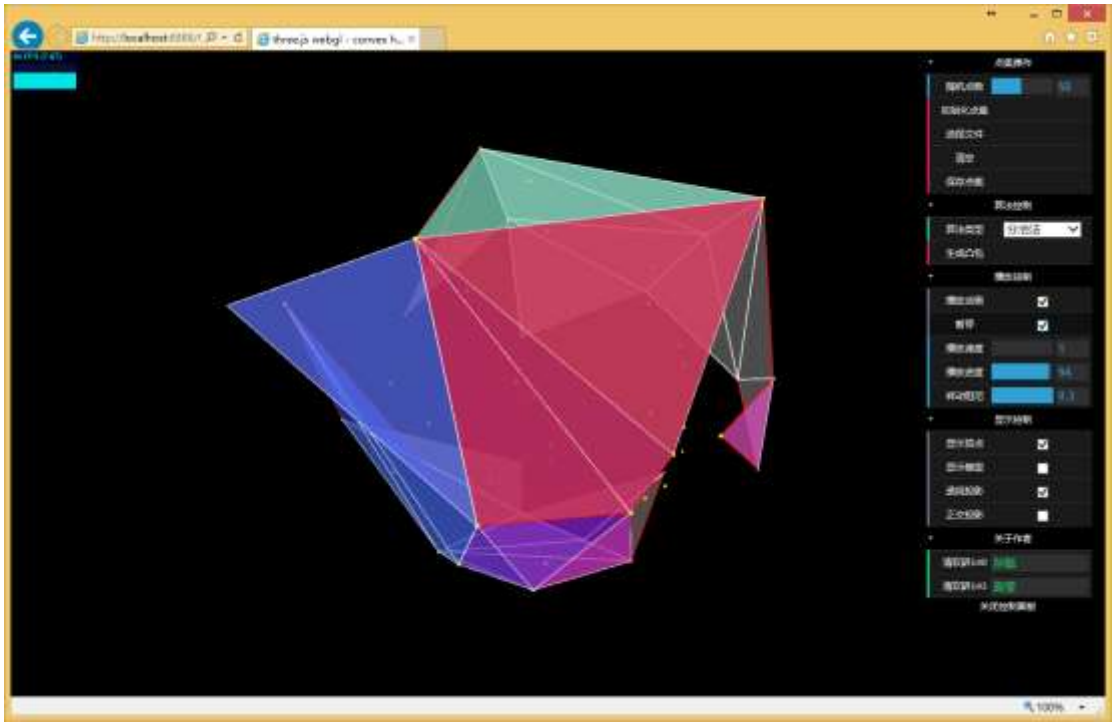
两个子凸包的样子



两个子凸包需要被移除的面被红色标出



移除所有面，并勾勒边界。



已经添加好的面用红色勾出，正在添加的面呈正常颜色，从基准面开始旋转，直到目标位置。

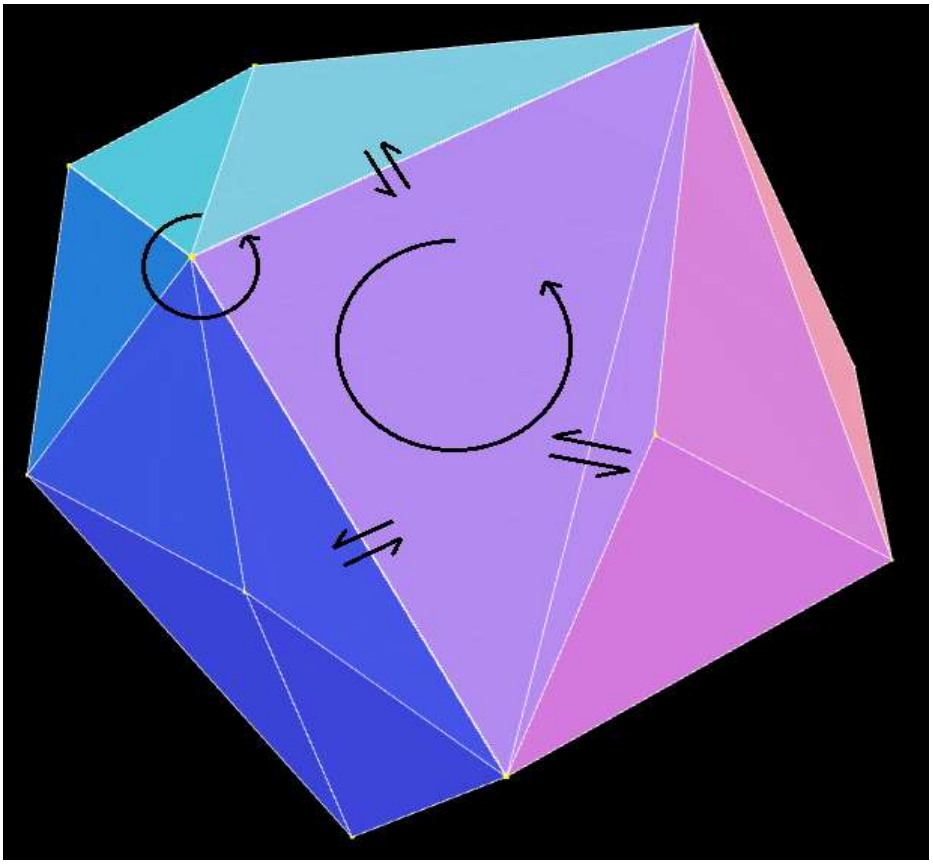
2 算法与数据结构

2.1 储存网格模型的数据结构

网格模型固定为三角网格模型。模型结构中仅有顶点和面两种结构，并不存在边的结构（边结构通过面结构中记录顶点的顺序确定）。

点被统一的储存在一个数组中，面同样被被储存在一个一个数组中，不过这个数组是一个变长数组，其长度随着新面片的加入而不断变长。一些算法（增量法，分治法等）执行过程中牵扯到面的增加与删除，而另外一些算法（礼品包装法）中每次都只添加凸包上的面。然而这些被删除的面在动画演示中是十分重要的，所以删除面的时候并不会直接从数组中移除面片，而是将其 valid 域置为 false，表示这是一个已经被删除的面片。而在最中生成用以渲染的凸包时，这些 valid 域为 false 的面片就不会参与到凸包的构成中来。

数组中的点与三角形均有一个 index 域。index 记录点/三角形在数组中的序号。根据算法的不同，点与三角形互相引用的时候引用的方式会不同。有时候会使用它们的索引，有时候则会直接引用点/三角形本身。第一种方式便于不同结构之间的交换（例如自己定义的结构与 THREE.js 的 Geometry 之间的交换），而第二种方式则便于计算（减少一次从点/三角形数组中获取元素的操作）。



对于每一个三角形，储存它的三个顶点和三个相邻的三角形。三个顶点的顺序实际上仅可能有两种（顺时针方向与逆时针方向，不考虑移位）。这里规定三角形的三个顶点固定为逆时针方向，这样就保证计算出面的法向固定为朝向几何体外。

而三个相邻的三角形的储存也是按照逆时针方向储存的。但是它们的移位并不是随意的。三角形的三个顶点按照[a,b,c]的顺序储存，那么三个相邻的三角形按照[ta,tb,tc]的顺序储存。即如果 a 顶点出现在 0 号顶点的位置，那么 0 号相邻三角形为 a 对面的那个相邻三角形。实际上规定其他的移位也是可以的，但是这样的规定更加直观。

顶点储存包含该顶点的所有三角形。同一个顶点可能包含任意多个个三角形，而储存它们的顺序并不做要求。而按次序访问某个顶点所有相邻三角形是通过其他方式实现的。

①对于某个顶点，获取与他相邻的任意三角形；

②获取该点在三角形中的序号，则该序号的下一位顶点所对的三角形即为下一个三角形。

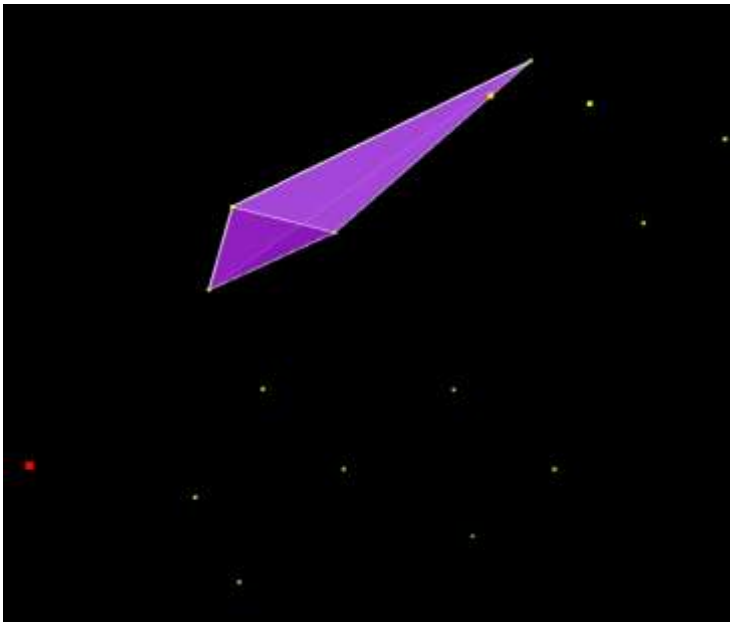
③重复以上步骤，直到回到第一个三角形。

由此可见，该结构所有关键步骤均可在常数或线性时间复杂度内完成，效率足够的高。而这一切得益于所有的面片均为三角面片。

2.2 朴素的增量法构造凸包

1. 构造初始三棱锥

初始三棱锥由四个点构成。这四个点要求两两不共点，三点不共线，四点不共面。否则，则不能生成初始三棱锥，会对后续增量构造凸包的方法带来问题。如果所有的点中不存在这样的四个点，则无法构造凸包，返回空。

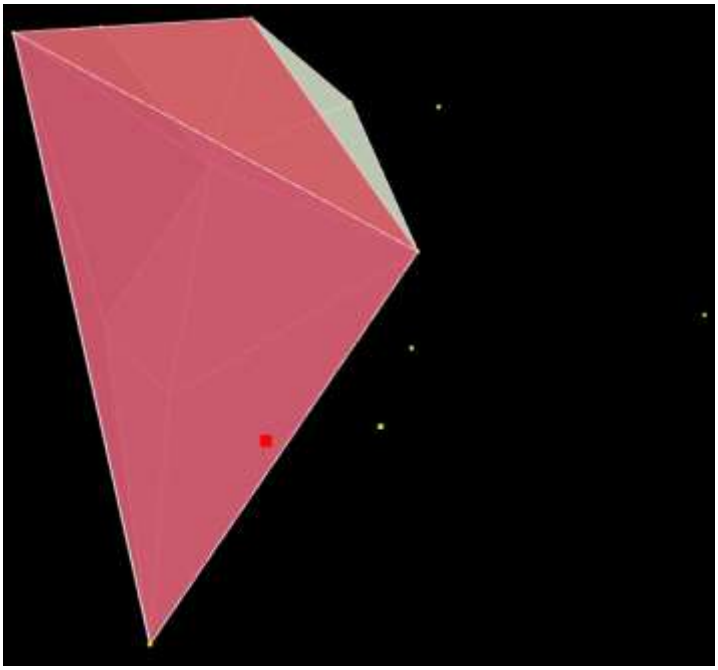


选取四个点构成初始三棱锥

2. 增量添加每一个顶点

1） 找到与该顶点可见的所有面。

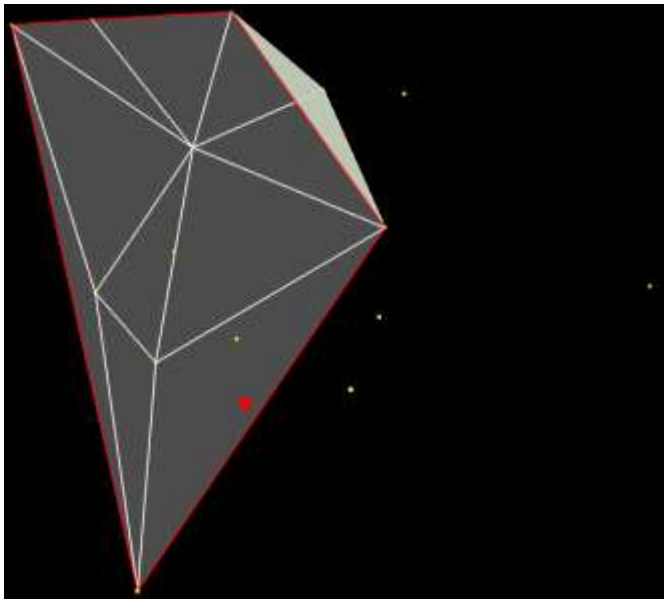
面的可见性判定可以通过面的法向来完成。首先计算面的法向 N，然后计算 N 与面上任意点到顶点的方向 M 的点积。如果该积大于零，则该点与该面互相可见，即该面属于需要被移除的面。实际算法中可以通过顺次计算所有面片与当前顶点的可见性来完成这一步，但是效率可能较低。本次试验中采用深度优先搜索的方法，仅需要找到第一个可见面，即可通关过面之间的连接关系找到所有可见的面。



与红色顶点可见的面被标成红色

2） 移除可见面并提取边界。

找到所有可见面之后，需要移除这些可见面并生成边界。如果一个三角形是可见三角形，但是与它相邻的三角形不是，那么它们之间公共的边就是边界。

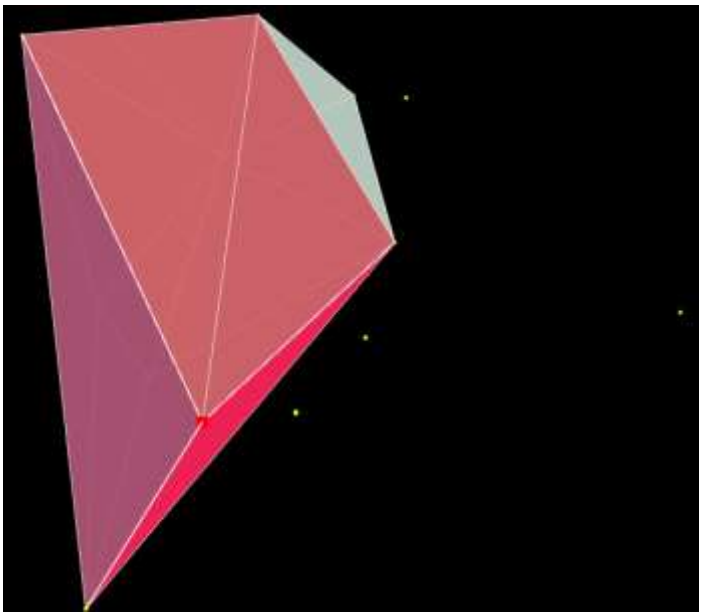


移除可见三角形并生成边界

3） 根据边界添加新的三角形。

边界上没一条边与当前新增加的顶点构成一个新的三角形。所有这些三角形构成一个锥面。而点的顺序关乎三角形的正反。因为每一条边均依附于一个已有的三角形，而两个相邻三角形中顶点的顺序是相反的。最后，需要维护这些新添加三角形之间互相邻

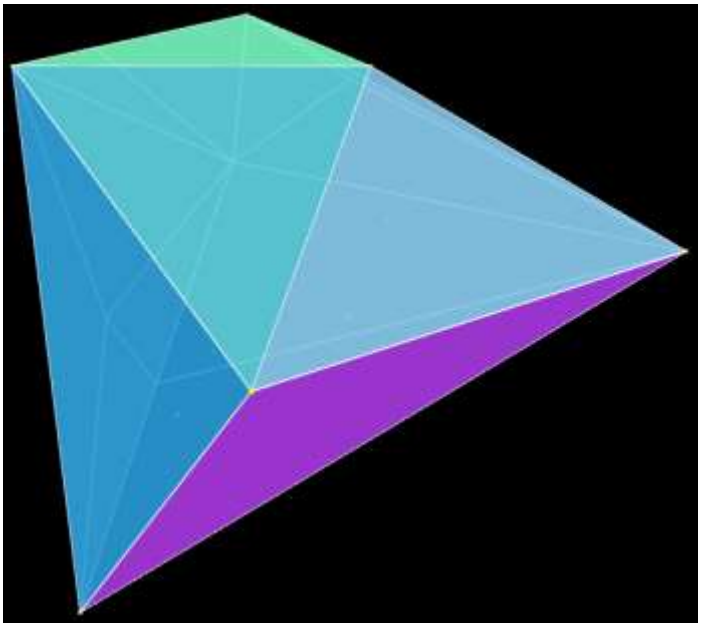
接关系。



新增加的锥面由红色标出

3 生成凸包

算法执行过程中移除面片并不是直接从数组中移除，而是将要移除的面片标记为已经被移除。所以，需要最后用线性的时间，将所有这些已经被移除的面片移除并构成心的数组。



最终生成凸包的样子

2.3 礼品包装法构造凸包

1. 找到凸包上的第一个面

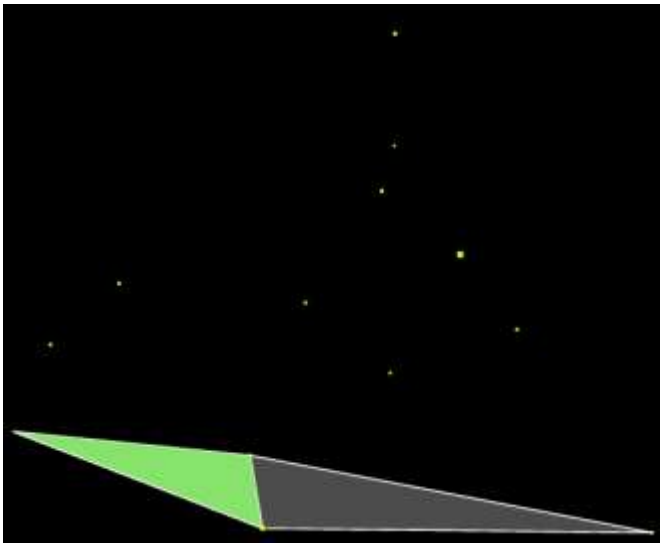
第一个面并不是随意构造的一个面，要保这个面是凸包上的面。首先，y 坐标最小的点

一定是凸包上的点，这个点就是凸包上第一个面。

第二个点为与与第一个点连线与 xz 平面所称角度最小的点。可以形象的理解为以第一个点为顶点的一个圆锥，其顶角由 180° 逐渐减小，直到碰到第一个点集中的点。这个点就是第二点。

现在考虑经过第一个点与第二个点的圆锥面的切平面。所有的点应当在该且平面的同侧。第三个点即为与之前两个点构成与切平面夹角最小三角形的点。

显然，上述三个点构成的面片为凸包上的一个面。

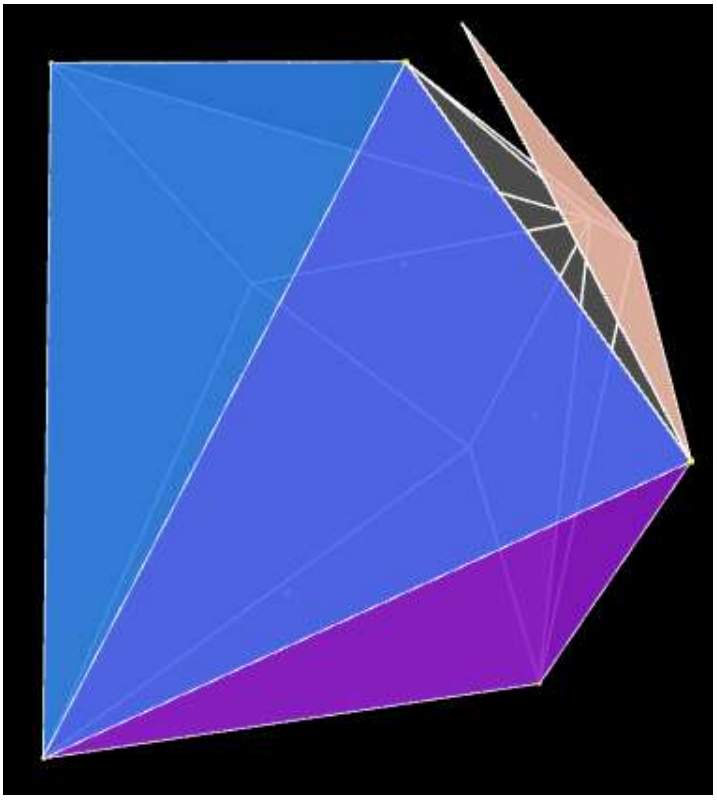


灰色面片为凸包上第一个面，绿色为将要“包裹”的下一个面片。

2. “包裹”面片

包裹面片从任意一个当前未封闭的三角面片开始。选取其任意一条未封闭的边，以该边为基础，从剩余顶点中选择一个点，使得未封闭边与该顶点构成的新面片与之前所依赖的面片夹角最小。显然，这样选出的面片也为凸包上的面。

如果新添加面片的顶点从未被选择过，那么增加面结束。如果新添加的面片以前被添加过，那么则需要维护三角形之间的邻接关系，即有一些边界需要被封闭，不需要继续“包裹”。

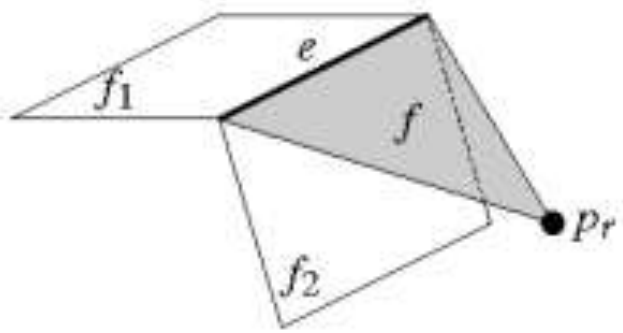


包裹最后一个面的情况。这个面一次性将三个未封闭的三角形。

2.4 冲突图法构造凸包

冲突图法是基于朴素增量法的。二者的不同在于冲突图法引入了冲突图这一概念，从而提升了算法的效率。下面仅介绍算法的流程和实现细节，对其复杂度的证明请参见《计算几何--算法与应用》11 章第三节。

就像在朴素增量法中描述的那样，如果一个面片与一个顶点互相可见，那么二者就是一对冲突。冲突的二者是不能共存于最终的凸包的，因此如果要添加一个新的顶点，只需要将于该顶点冲突的所有面片移除再添加新面片即可，不需要检索所有已有的面片。实际算法中，对于某个顶点，使用数组记录与它冲突的所有面；对于某个面，也使用一个数组记录与它冲突的所有顶点。



而更新与新增加面冲突的顶点，也不需要检索所有的顶点。如上图所示，新增加的面 f 一定是上面这种模式，e 是边界，f1 是被保留的面，而 f2 是被移除的面。仅需要检查与 f1、f2 冲突的顶点集合即可获得与面 f 冲突的顶点集合。

合并 f1，f2 冲突顶点几何可以使用类似于归并排序的算法。这样，可以保证合并的效率，排除重复的点，并且合并之后的集合仍然是有序的。

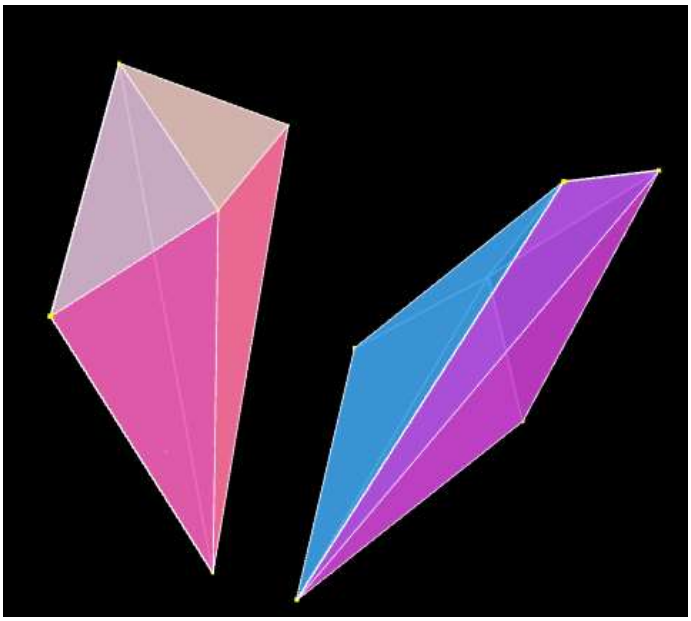
对于要移除的平面，理论上需要移除与之相关的所有冲突。实际上这是不必要的，只需要将被移除的面标记为已经被移除就好了。因为每个顶点的冲突表只会被使用一次，就是这个顶点被添加的时候。这个时候对于任何一个被移除的面片仅需要常数时间跳过即可。但是从数组或其他数据结构中移除当前顶点也至少需要常数时间。所以，移除面片时候不需要更新顶点的冲突面，只需要将面片标记为已移除即可。

2.5 分治法构造凸包

分治法建立在朴素增量法的基础上，使用分治的思想构造凸包，从而提高算法的效率。算法首先会将所有的点按照 x 坐标排序，划分的标准同样也是顶点的 x 坐标。这样，可以避免合并的凸包有重叠的部分，降低算法的复杂程度。

1. 点集划分

对于小于等于 7 个顶点的点集，使用增量法构造凸包。因为至少 4 个点才能构造出有意义的凸包，所以少于等于 7 个点的时候不再划分，直接使用增量法构造。而对于大于 7 个点的情况，则会将点集等分，分别构造小凸包，在进行合并凸包操作。



使用增量法构造的小凸包

2. 凸包合并

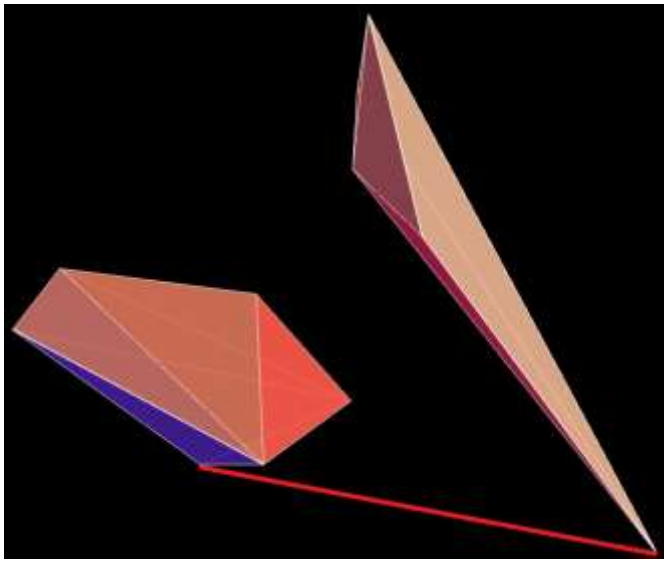
分治算法参考 *Convex Hulls of Finite Sets of Points in Two and Three Dimensions.* 重点参考其中凸包合并的部分。

凸包合并实际上是寻找包络两个较小凸包的“圆筒”，并移除被圆筒罩住的面片。这个“圆筒”一定是单面片序列，因为圆筒上的面片的三个顶点必然分列两个凸包上，使得面片总是横跨两个凸包。

1） 寻找“圆筒”上的第一条边。

论文中的方法是将两个凸包投影到一个平面上，寻找两个三维凸包的二维投影上的包络线。由于顶点事先按照 x 坐标排序，因此需要投影到 xy 平面或者 xz 平面。虽然从三维投影到了二维，但是顶点之间的连接关系仍然是存在的，所以仍可按照普通二维凸包寻找包络的方式寻找包络。

寻找包络的方法为：初始化为左侧最右点与右侧最左点。然后交替测试顶点的邻居，直到仰角达到极值。交替这个过程，直到左右两侧同时达到极值。这时，就找到了“圆筒”上的第一条边。



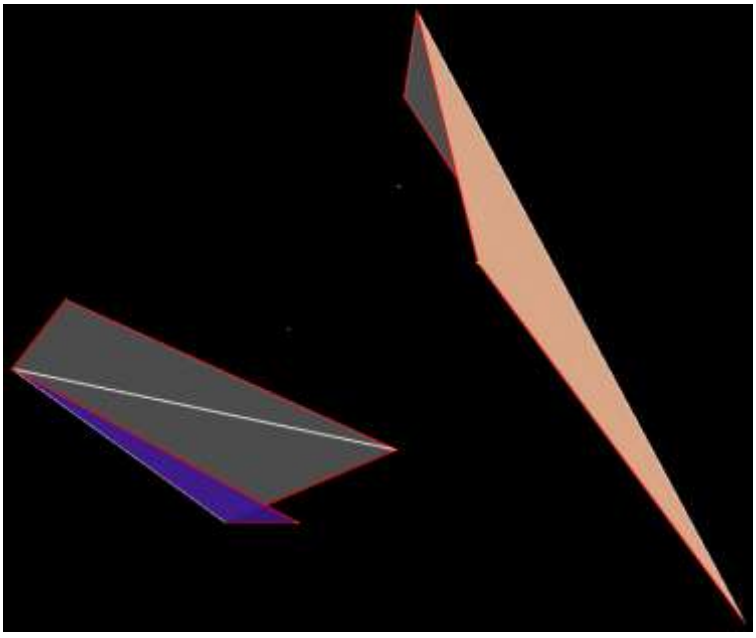
平行投影下沿 z 方向观察到的两个小凸包。红色线即为“圆筒”上的一条边。

2）顺次生成圆筒上的所有面片。

这个过程类似于礼品包装法。与礼品包装法不同的是，礼品包装法每次需要检查所有的顶点，而分治法这里不需要检查所有的顶点，只需要检查与顶点相邻的顶点即可。从中找出生成面片与原有面片夹角最小的一个，就是“圆筒”上的下一个面片。

生成第一个面片的时候，并没有上一个平面。这时候，如果投影到 xy 平面，那么就以过第一条边，平行于 z 轴的平面作为基准面。

3）移除被盖住的面片。



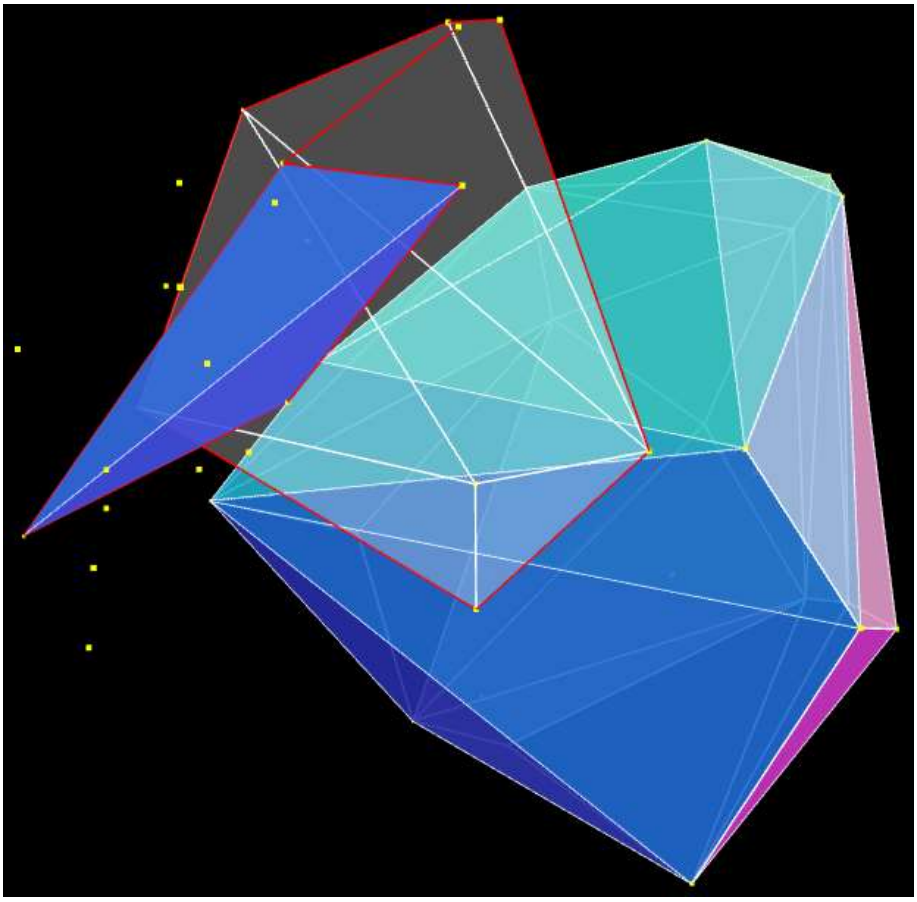
圆筒生成之后，需将圆筒添加到凸包结果中，并移除所有被挡住的面。

在生成“圆筒”的过程中，实际上已经能够拿到所有的边界，也就可以判断边界两边的

面片哪一个是被删除的，哪一个是被保留的。通过这些被删除的种子点，即可获得所有被删除的面片。

这里有几种极端的情况。例如某一侧的小凸包仅贡献一个点，或一条边，不贡献任何一个面。这种情况程序也能很好的处理。

仅贡献一条边的情况有一种奇异的推广，即移除面片后的边界出现一条伸出的边。这种情况似乎很古怪，但是如果想象该伸出的边处是一个极为细小的三角形，则不难理解。



边界中伸出一条边。

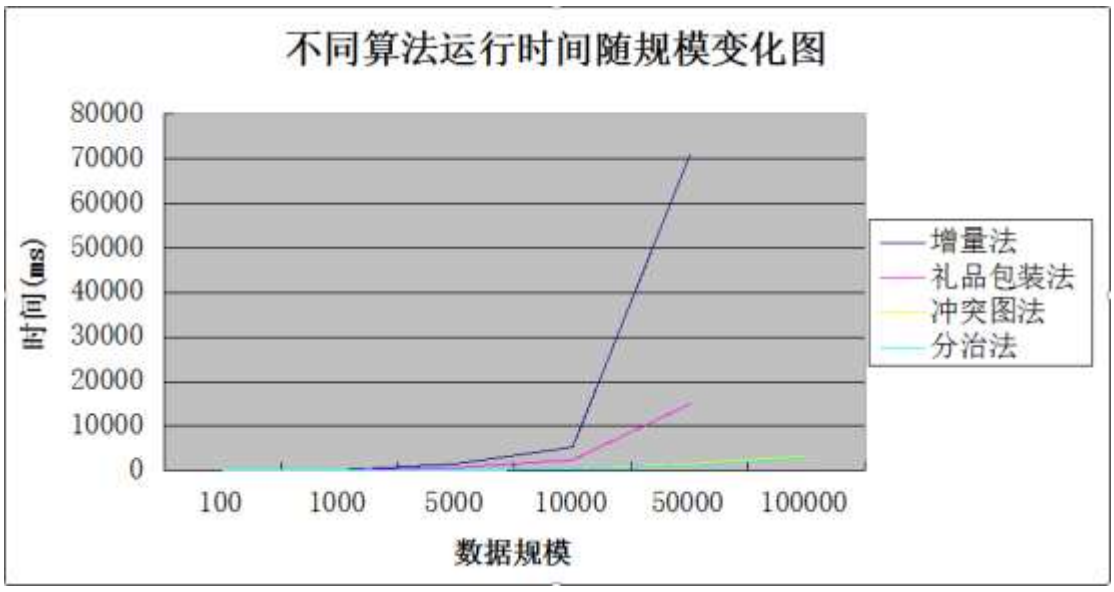
3 性能测试

本次作业考虑到易于展示与易于分发和复用，因而使用了 html 和 javascript 完成。因此整体来说代码执行的效率是不高的。下面列出了四种不同算法在不同数据规模下的执行时间。

横向为数据规模，纵向为使用的方法。表中时间的单位是 ms。

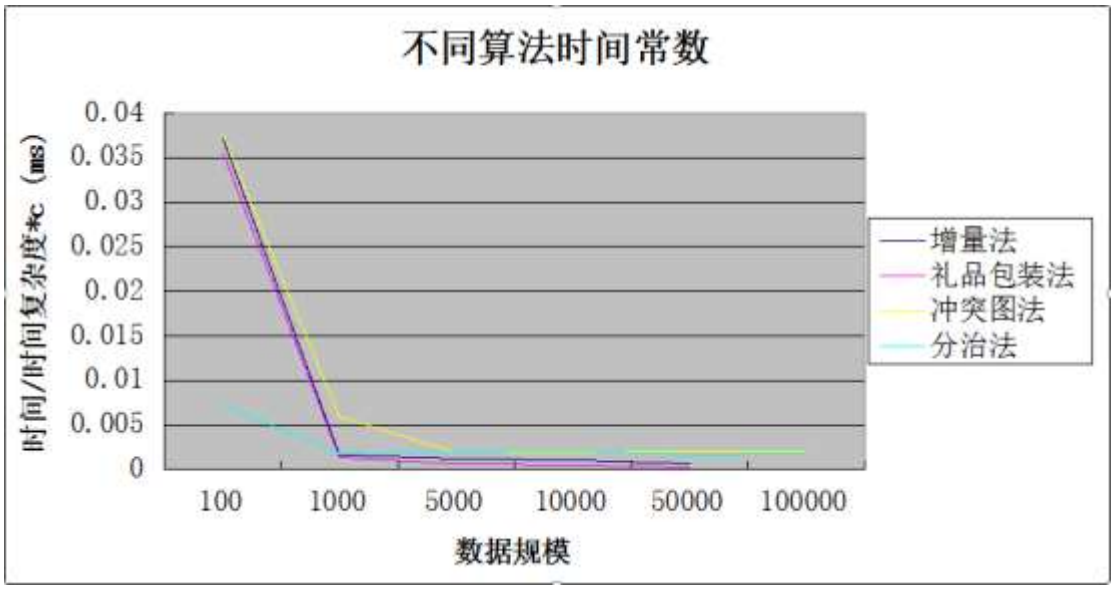
	100	1000	5000	10000	50000	100000
增量法	18.6	83.2	1429.6	5277.6	70834	
礼品包装法	17.8	67.8	815.6	2386.6	15134	
冲突图法	25	59.4	113.2	253	1598	3365.2
分治法	4.8	17.6	120.2	278.6	1303	2946.6

下图为运行时间随数据规模变化图。



算法运行时间随数据规模变化图。

下图为时间常数随数据规模变化情况。时间常数的计算公式为运行时间÷时间复杂度×常数 c。冲突图法和分治法时间常数较大，常数 c 为 1；增量法和礼品包装法时间常数较小，常数 c 为 20。



算法时间常数对比。可以看到当数据规模较大时，曲线趋于平稳。

4 参考文献

《计算几何--算法与应用》:Mark de Berg,Orfried Cheong,Marc van Kreveld, Mark Overmars 著, 邓俊辉 译; 第 11 章

Convex Hulls of Finite Sets of Points in Two and Three Dimensions F. P. Preparata and S. J. Hong

University of Illinois at Urbana-Champaign,1977.