vr-assignment2

18364107 叶微

December 3, 2020

1 泰森多边形

利用 python/matlab 求出以下 25 个二维离散点的泰森多边形区域并画图展示结果，坐标轴范围为 x [0, 50], y [0, 50]。

其中不同的泰森多边形用不同的颜色表示。

x = [43, 20, 34, 18, 12, 32, 40, 4, 44, 30, 6, 47, 23, 13, 38, 48, 36, 46, 50, 37, 21, 7, 28, 25, 10];

y = [3, 43, 47, 31, 30, 39, 9, 33, 49, 36, 21, 48, 14, 34, 41, 4, 1, 44, 18, 24, 20, 11, 27, 42, 13];

第一个离散点的坐标为（43,3），以此类推。

1.1 泰森多边形的特征

1) 每个泰森多边形内仅含有一个离散点数据。
2) 泰森多边形内的点到相应离散点的距离最近。
3) 位于泰森多边形边上的点到其两边的离散点的距离相等。

1.2 泰森多边形的实现方式

泰森多边形法，美国气候学家 A • H • Thiessen 提出了一种根据离散分布的气象站的降雨量，来计算平均降雨量的方法，即将所有相邻气象站连成三角形，作这些三角形各边的垂直平分线，将每个三角形的三条边的垂直平分线的交点（也就是外接圆的圆心）连接起来得到一个多边形。用这个多边形内所包含的一个唯一气象站的降雨强度来表示这个多边形区域内的降雨强度，并称这个多边形为泰森多边形。如图，其中虚线构成的多边形就是泰森多边形。泰森多边形每个顶点是每个三角形的外接圆圆心。泰森多边形也称为 Voronoi 图，或 dirichlet 图。

建立泰森多边形算法的关键是对离散数据点合理地连成三角网，即构建 Delaunay 三角网。建立泰森多边形的步骤如下：

1) 离散点自动构建三角网，即构建 Delaunay 三角网。对离散点和形成的三角形编号，记录每个三角形是由哪三个离散点构成的；

2) 找出与每个离散点相邻的所有三角形的编号，并记录下来。这只要在已构建的三角网中找出具有一个相同顶点的所有三角形即可；

3) 对与每个离散点相邻的三角形按顺时针或逆时针方向排序，以便下一步连接生成泰森多边形。设离散点为 o。找出以 o 为顶点的一个三角形，设为 A；取三角形 A 除 o 以外的另一顶点，设为 a，则另一个顶点也可找出，即为 f；则下一个三角形必然是以 of 为边的，即为三角形 F；三角形 F 的另一顶点为 e，则下一三角形是以 oe 为边的；如此重复进行，直到回到 oa 边；

4) 计算每个三角形的外接圆圆心，并记录之；

5) 根据每个离散点的相邻三角形，连接这些相邻三角形的外接圆圆心，即得到泰森多边形。对于三角网边缘的泰森多边形，可作垂直平分线与图廓相交，与图廓一起构成泰森多边形。

1.3 具体源码分析

1.3.1 设置初始点模块

```
N=25;
x =[43;20;34;18;12;32;40;4;44;30;6;47;23;13;38;48;36;46;50;37;21;7;28;25;10];
y =[3;43;47;31;30;39;9;33;49;36;21;48;14;34;41;4;1;44;18;24;20;11;27;42;13];
xdot =[x,y];
x = x *0.010
y = y *0.010
xdot1=[x,y];
N=size(xdot1,1);

```

1.3.2 Delaulay 三角形的构建

1.3.2.1 Delaulay 三角定义

三角剖分：假设 V 是二维实数域上的有限点集，边 e 是由点集中的点作为端点构成的封闭线段, E 为 e 的集合。那么该点集 V 的一个三角剖分 T=(V,E) 是一个平面图 G，该平面图满足条件：

1) 除了端点，平面图中的边不包含点集中的任何点。

2) 没有相交边。

3) 平面图中所有的面都是三角面，且所有三角面的合集是散点集 V 的凸包。

在实际中运用的最多的三角剖分是 Delaunay 三角剖分，它是一种特殊的三角剖分。先从 Delaunay 边说起：

1.3.2.2 Delaunay 边定义

Delaunay 边：假设 E 中的一条边 e（两个端点为 a,b），e 若满足下列条件，则称之为 Delaunay 边：存在一个圆经过 a,b 两点，圆内 (注意是圆内，圆上最多三点共圆) 不含点集 V 中任何其他的点，这一特性又称空圆特性。

1.3.2.3 Delaunay 三角剖分

Delaunay 三角剖分：如果点集 V 的一个三角剖分 T 只包含 Delaunay 边，那么该三角剖分称为 Delaunay 三角剖分。

优化处理: 理论上为了构造 Delaunay 三角网，Lawson 提出的局部优化过程 LOP(Local Optimization Procedure)，一般三角网经过 LOP 处理，即可确保成为 Delaunay 三角网，其基本做法如下所示：

1) 将两个具有共同边的三角形合成一个多边形。
2) 以最大空圆准则作检查，看其第四个顶点是否在三角形的外接圆之内。
3) 如果在，修正对角线即将对角线对调，即完成局部优化过程的处理。

LOP 处理过程如下图所示：

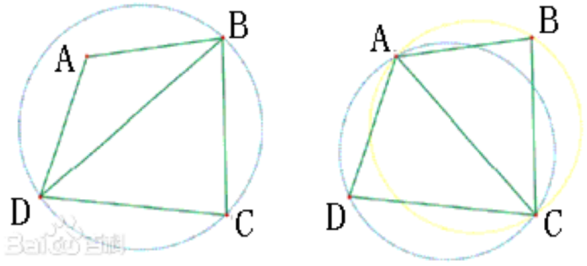

Figure 1: LOP 处理过程

1.3.2.4 计算方法-Bowyer-Watson 算法

Watson 算法的基本步骤是：

1) 构造一个超级三角形，包含所有散点，放入三角形链表。
2) 将点集中的散点依次插入，在三角形链表中找出外接圆包含插入点的三角形（称为该点的影响三角形），删除影响三角形的公共边，将插入点同影响三角形的全部顶点连接起来，完成一个点在 Delaunay 三角形链表中的插入。
3) 根据优化准则对局部新形成的三角形优化。将形成的三角形放入 Delaunay 三角形链表。
4) 循环执行上述第 2 步，直到所有散点插入完毕。

这一算法的关键的第 2 步图示如下：

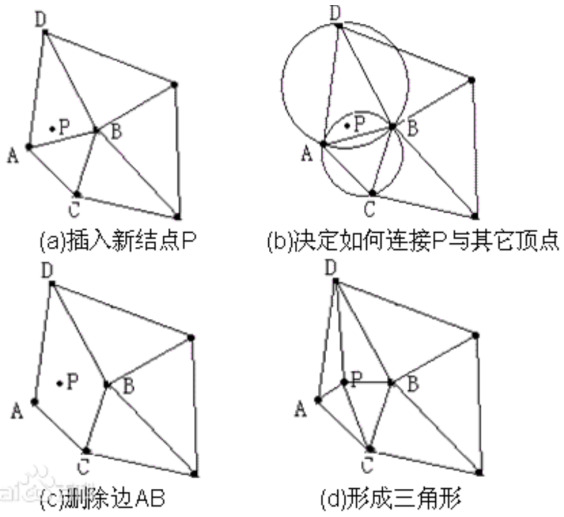

Figure 2: 关键步骤

1.3.2.5 Delaulay 三角形的构建 + 凸包检测

伪代码

```
        input: 顶点列表(vertices)                    //vertices为外部生成的随机或乱序顶点列表
        output:已确定的三角形列表(triangles)
            初始化顶点列表
            创建索引列表(indices = new Array(vertices.length))//indices数组中的值为0,1,2,3,.....,
    vertices.length-1
            基于vertices中的顶点x坐标对indices进行sort
        //sort后的indices值顺序为顶点坐标x从小到大排序（也可对y坐标，本例中针对x坐标）
            确定超级三角形
            将超级三角形保存至未确定三角形列表（temp triangles）
            将超级三角形push到triangles列表
            遍历基于indices顺序的vertices中每一个点//基于indices后，则顶点则是由x从小到大出现
                初始化边缓存数组（edge buffer）
                遍历temp triangles中的每一个三角形
                    计算该三角形的圆心和半径
                    如果该点在外接圆的右侧
                        则该三角形为Delaunay三角形，保存到triangles
                        并在temp里去除掉
                        跳过
                    如果该点在外接圆外（即也不是外接圆右侧）
                        则该三角形为不确定                          //后面会在问题中讨论
                        跳过
                    如果该点在外接圆内
                        则该三角形不为Delaunay三角形
                        将三边保存至edge buffer
                        在temp中去除掉该三角形
                对edge buffer进行去重
                将edge buffer中的边与当前的点进行组合成若干三角形并保存至temp triangles中
            将triangles与temp triangles进行合并
            除去与超级三角形有关的三角形
end
```

具体实现

```
        %1 Delaulay三角形的构建
*********************************************************************************

        %整理点，遵循从左到右，从上到下的顺序
        xdot1=sortrows(xdot1,[1 2]);

        %画出最大包含的三角形
        xmin=min(xdot1(:,1));xmax=max(xdot1(:,1));
        ymin=min(xdot1(:,2));ymax=max(xdot1(:,2));
        bigtri=[(xmin+xmax)/2-(xmax-xmin)*1.5,ymin-(xmax-xmin)*0.5;...
            (xmin+xmax)/2,ymax+(ymax-ymin)+(xmax-xmin)*0.5;...
           (xmin+xmax)/2+(xmax-xmin)*1.5,ymin-(xmax-xmin)*0.5];

        xdot1=[bigtri;xdot1];%点集
        edgemat=[1 2 xdot1(1,:) xdot1(2,:);...
            2 3 xdot1(2,:) xdot1(3,:);1 3 xdot1(1,:) xdot1(3,:)];%边集，每个点包含2个点，4个坐标值
        trimat=[1 2 3];%三角集,每个三角包含3个点
        temp_trimat=[1 2 3];
        for j=4:N+3
            pointtemp=xdot1(j,:);%循环每一个点
            deltemp=[];%初始化删除temp_trimat的点
            temp_edgemat=[];%初始化临时边
            for k=1:size(temp_trimat,1)%循环每一个temp_trimat的三角形
                panduan=whereispoint(xdot1(temp_trimat(k,1),:),...
                    xdot1(temp_trimat(k,2),:),xdot1(temp_trimat(k,3),:),pointtemp);%判断点在圆内0、圆外1、圆右侧2
                switch panduan
                    case 0
                        %点在圆内
                        %则该三角形不为Delaunay三角形
                        temp_edge=maketempedge(temp_trimat(k,1),temp_trimat(k,2),temp_trimat(k,3),j,xdot1);%把三条边暂时存放于临时边矩阵
                        temp_edgemat=[temp_edgemat;temp_edge];
                        deltemp=[deltemp,k];
                        ;
                    case 1
                        %点在圆外，pass
                        ;
                    case 2
                        %点在圆右
                        %则该三角形为Delaunay三角形，保存到triangles
                        trimat=[trimat;temp_trimat(k,:)];%添加到正式三角形中
                        deltemp=[deltemp,k];
                        %并在temp里去除掉
                        %别忘了把正式的边也添加进去
                        edgemat=[edgemat;makeedge(temp_trimat(k,1),temp_trimat(k,2),temp_trimat(k,3),xdot1)];%遵循12,13,23的顺序
                        edgemat=unique(edgemat,'stable','rows');

                end

            %三角循环结束
            end


```

```
            %除去上述步骤中的临时三角形
            temp_trimat(deltemp,:)=[];
            temp_trimat(~all(temp_trimat,2),:)=[];
            %对temp_edgemat去重复
            temp_edgemat=unique(temp_edgemat,'stable','rows');
            %将edge buffer中的边与当前的点进行组合成若干三角形并保存至temp triangles中
            temp_trimat=[temp_trimat;maketemptri(temp_edgemat,xdot1,j)];
            k=k;


        %点循环结束
        end

        %合并temptri
        trimat=[trimat;temp_trimat];
        edgemat=[edgemat;temp_edgemat];
        %删除大三角形
        deltemp=[];
        for j=1:size(trimat,1)
            if ismember(1,trimat(j,:))||ismember(2,trimat(j,:))||ismember(3,trimat(j,:))
                deltemp=[deltemp,j];
            end
        end
        trimat(deltemp,:)=[];
        edgemat=[trimat(:,[1,2]);trimat(:,[2,3]);trimat(:,[3,1])];
        edgemat=sort(edgemat,2);
        edgemat=unique(edgemat,'stable','rows');


        temp_edgemat=[];
        temp_trimat=[];
```

算法最终输出的三角形列表 trimat 包含三个点的编号。

1.3.2.6 Delaunay 三角算法的凸边形检测

伪代码

```
输入上一算法中得到的trimat
更新三角网格边缘三角形border_trimat
更新三角网格边缘点border_point
更新三角形网格之间的关系，即与每个三角形相邻的三角形trimat_con
整理边缘点border_point顺序，使得顺时针（或逆时针）成为当前图像最外边缘
依次循环边缘点的每一个点j
    按顺时针（或逆时针）做该点j与间隔点j+2的线段（border_point的顺序）
    求线段与点j+1相关的所有线段（或延长线）是否相交
    如果相交
        该点为图形边缘的突出点，忽略
    如果与所有点j+1相关的线段（或延长线）不相交
        该点为图形边缘凹陷点，连接
        返回最开始更新步骤
输出trimat_con
```

具体实现

```
    %1.5 凸包监测
**********************************************************************************************
    %思路是先找出边缘点（三角形只有1个或2个的），顺便整出一个三角形相互关系图，以后用。
    %然后顺时针，依次隔一个点连接出一条线段，如果这个和之前的线段相交，则不算；如果不交，则记录出三角形
    %更新完了以后，再监测一遍，直到没有新的为止。

    t_w=0;
    while t_w==0
        [~,border_point,~]=makebordertri(trimat);
        border_point=[border_point;border_point(1,:)];
        temp_edgemat=[];
        temp_trimat=[];
        for j=1:size(border_point,1)-1
            tempboderedge=[border_point(j,1),border_point(j+1,2)];
            tempboderdot=border_point(j,2);
            %寻找带tempboderdot的所有边
            tempdotex=edgemat(logical(sum(edgemat==tempboderdot,2)),:);
            %删除相邻边
            tempdotex(ismember(tempdotex,[tempboderdot,tempboderedge(1)],'rows'),:)=[];
            tempdotex(ismember(tempdotex,[tempboderedge(1),tempboderdot],'rows'),:)=[];
            tempdotex(ismember(tempdotex,[tempboderdot,tempboderedge(2)],'rows'),:)=[];
            tempdotex(ismember(tempdotex,[tempboderedge(2),tempboderdot],'rows'),:)=[];
            %检测tempdotex是否为空，如果是证明不用相连
            t_N=size(tempdotex,1);
            t_t=0;
            if t_N>0
                %依次检测是否相交，只要有一个相交就不算；如果都不想交，则相连
                for k=1:t_N
                    if tempdotex(k,1)==tempboderdot
                        t_xdot1no4=tempdotex(k,2);
                    else
                        t_xdot1no4=tempdotex(k,1);
                    end
                    tt_xdot1no4=xdot1(t_xdot1no4,:)-xdot1(tempboderdot,:);
                    xdot1no4=xdot1(tempboderdot,:)+tt_xdot1no4/sqrt(sum(tt_xdot1no4.^2))*(sqrt((xmax-xmin)^2+(ymax-ymin)^2));
                    panduan=crossornot(xdot1(tempboderedge(1),:),xdot1(tempboderedge(2),:),xdot1(tempboderdot,:),xdot1no4);
                    if panduan==1
                        t_t=t_t+1;
                        break
                    end
                end
                %t_t大于0说明有相交的线，略过
                if t_t==0
                    temp_edgemat=[temp_edgemat;tempboderedge];
                    temp_trimat=[temp_trimat;[tempboderedge,tempboderdot]];
                    break
                end
            end
        end
        trimat=[trimat;temp_trimat];
        edgemat=[edgemat;temp_edgemat];
        %删除重复的三角形
        trimat=sort(trimat,2);
        trimat=unique(trimat,'stable','rows');
        if j==size(border_point,1)-1
            t_w=1;
```

```
            end
        end
        %2 泰森多边形的建立步骤
    ********************************************************************************
        %求每个三角形的外接圆圆心

        trimatcenter=zeros(size(trimat,1),2);
        for j=1:size(trimat,1)
            [a,b,~]=maketricenter(xdot1(trimat(j,1),:),xdot1(trimat(j,2),:),xdot1(trimat(j,3),:));
            trimatcenter(j,:)=[a,b];
        end
        hold on
        scatter(trimatcenter(:,1),trimatcenter(:,2),5,[0.6,0,0],'filled')
        hold off

        %求三角形的相邻三角形个数
        [border_trimat,border_point,trimat_con]=makebordertri(trimat);
        Thi_edge1=[];
        for j=1:size(trimat,1)
            tempedge=[];
            %第一个相邻三角形
            if trimat_con(j,1)~=0
                tempedge=[tempedge;[j,trimat_con(j,1)]];
            end
            %第二个相邻三角形
            if trimat_con(j,2)~=0
                tempedge=[tempedge;[j,trimat_con(j,2)]];
            end
            %第三个相邻三角形
            if trimat_con(j,3)~=0
                tempedge=[tempedge;[j,trimat_con(j,3)]];
            end
            Thi_edge1=[Thi_edge1;tempedge];
        end



        %绘制边缘泰勒多边形
        %先逐个边试探，如果中心点在三角内，则做中心-边缘延长线
        %如果中心点在三角外，如果在屏幕外，忽略，如果在屏幕内，做边缘-中心延长线

        for j=1:size(border_point,1)
            %先找到边对应的三角
            temp_trimat=border_trimat(sum(border_trimat==border_point(j,1),2)+sum(border_trimat==
    border_point(j,2),2)==2,:);
            %判断中心点是否在三角形内
            [t_x1,t_y1,~]=maketricenter(xdot1(temp_trimat(1),:),xdot1(temp_trimat(2),:),xdot1(
    temp_trimat(3),:));%求中心

            panduan=pointintriangle(xdot1(temp_trimat(1),:),xdot1(temp_trimat(2),:),xdot1(
    temp_trimat(3),:),[t_x1,t_y1]);
            %求边的中点
            t_x2=(xdot1(border_point(j,1),1)+xdot1(border_point(j,2),1))/2;
            t_y2=(xdot1(border_point(j,1),2)+xdot1(border_point(j,2),2))/2;
            if panduan==1
                %做中心-边缘的延长线
                %这里用到了边缘在01这个条件
                t_xy3=[t_x1,t_y1]+[t_x2-t_x1,t_y2-t_y1]*sqrt(2)/sqrt((t_x2-t_x1)^2+(t_y2-t_y1)^2);
                plot([t_x1,t_xy3(1)],[t_y1,t_xy3(2)],'color',[0,0.4,0])
            elseif ~(t_x1<0||t_x1>1||t_y1<0||t_y1>1)
                %判断点是否在边与边框的三角内，如果在，做中心的延长线
                %如果不在，做中心-边缘的延长线
```

```
                %或者改成判断点是否在多边形内

                t_t=pointinmutiangle(xdot1,[border_point(1,1);border_point(:,2)],[t_x1,t_y1]);
                if t_t==1
                    t_xy3=[t_x1,t_y1]+[t_x2-t_x1,t_y2-t_y1]*sqrt(2)/sqrt((t_x2-t_x1)^2+(t_y2-t_y1)
^2);
                    plot([t_x1,t_xy3(1)],[t_y1,t_xy3(2)],'color',[0,0.4,0])
                else
                    t_xy3=[t_x1,t_y1]+[t_x1-t_x2,t_y1-t_y2]*1/sqrt((t_x2-t_x1)^2+(t_y2-t_y1)^2);
                    plot([t_x1,t_xy3(1)],[t_y1,t_xy3(2)],'color',[0,0.4,0])
                end
            end
        end

        scatter(xdot1(:,1),xdot1(:,2),5,[0,0.4,0],'filled')
        hold off

```

1.3.2.7 多边形面的识别

具体实现

```

    %第3部分   多边形面的识别
*********************************************************************************
    %1先规划出多边形预设矩阵（找出xdot1数组中出现次数最多的数），行个数等于三角形外接圆中心点的
个数
    %2选取中心点，然后依次找出一圈三角形（共点），之后依次连接这一圈三角形的外接圆就是泰森多边形
    %3对于边缘点，要结合边缘考虑，画出边缘线和延长射线的交点，就是了。还是依次画出，如果点在边缘
外，就忽略，在边缘内，就记录。
    %4选取上一步记录的三角形最外侧三角，做中垂线

```

具体函数与作图实现参考附录。

2 GJK 算法

2.1 GJK 算法详细介绍

2.1.0.1 概述

GJK 算法只对凸体有效。GJK 算法的优势是：通过 support 函数（后面会详细讲述），从而支持任何凸体形状之间的碰撞检测；

GJK 是一个迭代算法，但是如果事先给出穿透/分离向量，则它的收敛会很快，可以在常量时间内完成。GJK 算法的最初目的是计算两个凸体之间的距离，在两个物体穿透深度比较小的情况下，可用它判定物体之间的碰撞。它也可以和别的算法相结合，用来检测两个物体之间深度穿透时候的碰撞情况。

2.1.0.2 凸体 (凸多面体或凸多边形)

前面说过，GJK 算法只适用于凸体形状。凸体 (其实就是一条直线穿越凸体，和该凸体壳的交点不能超过 2 个).

2.1.0.3 明可夫斯基和 (Minkowski Sum)

GJK 算法中使用了明可夫斯基和的概念。明可夫斯基和很好理解，假设有两个物体，它们的明可夫斯基和就是物体 1 上的所有点和物体 2 上的所有点的和集。用公式表示就是：

$$A + B = a + b|a \in A, b \in B$$

如果两个物体都是凸体，它们的明可夫斯基和也是凸体。

对于减法，明可夫斯基和的概念也成立，这时也可称作明可夫斯基差。

$$A–B = A + (-B) = a + (–b)|a \in A, b \square B = a–b)|a \in A, b \in B$$

接着往下讲，在两个物体之间执行明可夫斯基差操作的解释如下：

如果两个物体重叠或者相交，它们的明可夫斯基差肯定包括原点。

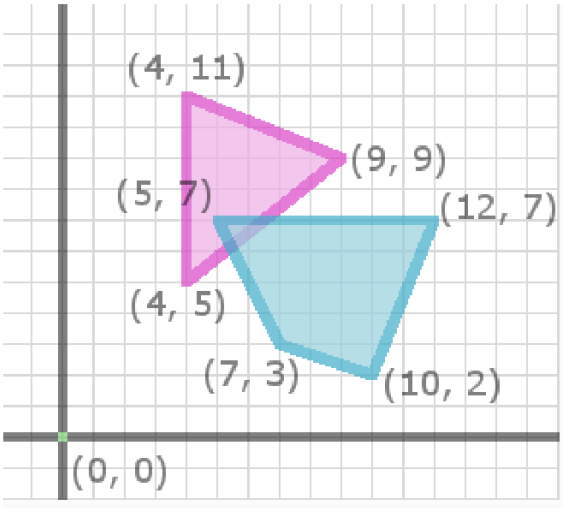

Figure 3: 两个凸体相交

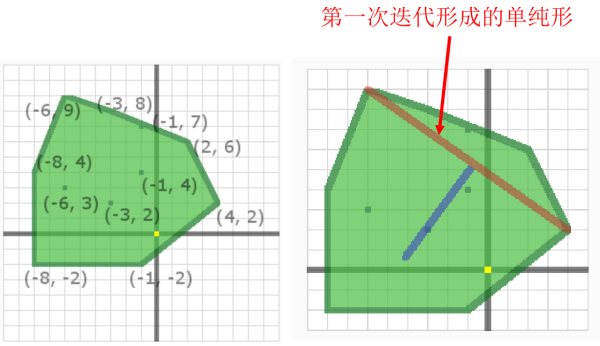

Figure 4: 明可夫斯基差

执行这些操作需要物体 1 的顶点数 * 物体 2 的顶点数 *3(在三维空间，是 *3，如果是向量减法数量就什么都不用乘了) 个减法操作。物体包含无穷多个点，但由于是凸体，我们可以只对它们的顶点执行明可夫斯基差操作。在执行 GJK 算法过程中，实际上我们并不需要显式计算物体之间明可夫斯基差，这也是 GJK 算法的优势所在。

2.1.0.4 单纯形 (Simplex)

我们不需要显式计算物体之间的明可夫斯基差，只要知道它们的明可夫斯基差是否包含原点就 ok 了。如果包含原点，物体之间就相交，否则，则不相交。

我们可以在明可夫斯基差形成的物体内迭代的形成一个多面体 (或多变形)，并使这个多面体尽量包围原点。如果这个多面体包含原点，显然明可夫斯基差形成的物体必然包括原点。这个多面体就称作单纯形。

2.1.0.5 Support 函数

下面我们讲述如何建立一个单纯形。首先看什么是 support 函数，给定两个凸体，该函数返回这两个凸体明可夫斯基差形状中的一个点。我们知道，物体 1 上的一个点，它的位置减去物体 2 上的一个点的位置，可以得到它们明可夫斯基差形状上的一个点，但我们不希望每次都得到相同的点。如何保证做到这一点呢？我们可以给 support 函数传递一个参数，该参数表示一个方向 (direction), 方向不同，得到的点也不同。

2.1.0.6 代码实例

```
function point = support(shape1,shape2,v)
%Support function to get the Minkowski difference.
point1 = getFarthestInDir(shape1, v);
point2 = getFarthestInDir(shape2, -v);
point = point1 - point2;
end
```

在某个方向上选择最远的点有重要作用，因为这样产生的单纯形包含最大的空间区域，增加了算法快速返回的可能。另外，通过这种方式返回的点都在明可夫斯基差形状的边上。如果我们不能通过一个过原点的方向在单纯形上增加一个点，则明可夫斯基差不过原点，这样在物体不相交的情况下，算法会很快退出。

具体代码示例在附录中给出。

3 AABB 包围盒算法

3.1 AABB 包围盒算法详细介绍

3.1.0.1 AABB 包围盒

在游戏中，为了简化物体之间的碰撞检测运算，通常会对物体创建一个规则的几何外形将其包围。

其中，AABB（axis-aligned bounding box）包围盒被称为轴对其包围盒。

二维场景中的 AABB 包围盒具备特点：

1) 表现形式为四边形，即用四边形包围物体。

2) 四边形的每一条边，都会与坐标系的轴垂直。

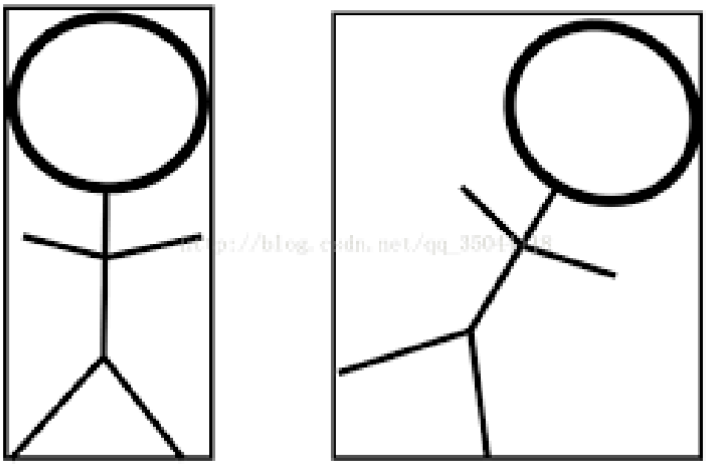

Figure 5: 2d

三维场景中的 AABB 包围盒特点：

1) 表现形式为六面体。

2) 六面体中的每条边都平行于一个坐标平面。

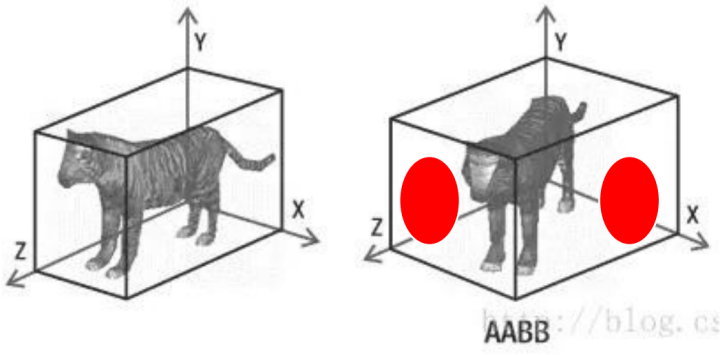

Figure 6: 3d

3.1.0.2 二维场景中的 AABB 碰撞检测原理

首先来看一张二维场景中的物体碰撞图：

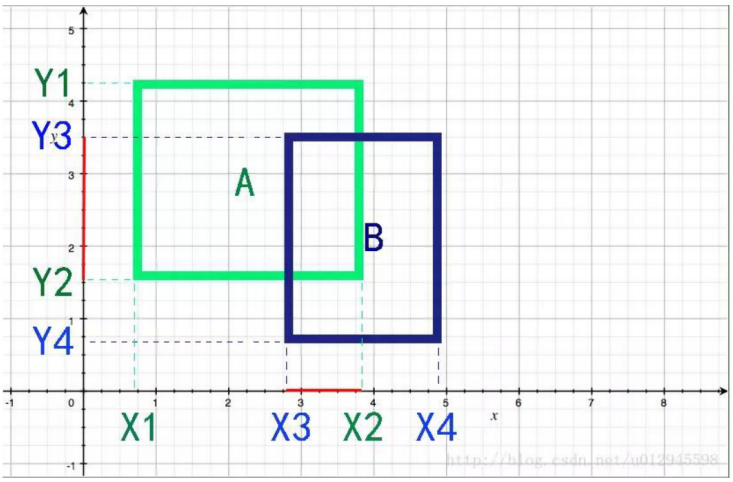

Figure 7: 2d collision

在图中，分别做物体 A 与物体 B 在 X,Y 轴方向的投影，物体 A 的 Y 轴方向最大点坐标为 Y1，最小点坐标 Y2，X 轴方向最小点坐标 X1，最大点坐标 X2，物体 B 同理。

图中红色区域为物体 A 与物体 B 投影的重叠部分。

可以看出，AABB 碰撞检测具有如下规则：

物体 A 与物体 B 分别沿两个坐标轴做投影，只有在两个坐标轴都发生重叠的情况下，两个物体才意味着发生了碰撞。

所以，在程序中做二维游戏的 AABB 碰撞检测时，只需验证物体 A 与物体 B 是否满足如下条件：

1) 物体 A 的 Y 轴方向最小值大于物体 B 的 Y 轴方向最大值；
2) 物体 A 的 X 轴方向最小值大于物体 B 的 X 轴方向最大值；
3) 物体 B 的 Y 轴方向最小值大于物体 A 的 Y 轴方向最大值；
4) 物体 B 的 X 轴方向最小值大于物体 A 的 X 轴方向最大值；

若满足上述条件，则证明物体 A 与物体 B 并未发生重合，反之，则证明物体 A 与物体 B 重合。

3.1.0.3 三维场景中的 AABB 碰撞检测原理

首先，再来看一下上图中的二维物体 A 和物体 B 的包围盒，可以发现实际上判断物体 A 与物体 B 是否发生重合只需要知道两个信息：

1) 物体 A 的最小点的信息，即图 2-1 中 A 的左下角点；以及物体 A 的最大点的信息，即图 2-1 中 A 的右上角点。
2) 物体 B 的最小点的信息，物体 B 的最大点的信息。

也就是说在二维场景的碰撞检测中，每个物体的顶点坐标信息都可以由两个坐标来确定，即两个坐标就可以标识一个物体了，所以两个物体的碰撞检测只需要获得到四个点坐标就可以了。

在之前的图中已经看到，三维场景中物体的 AABB 包围盒是一个六面体，其坐标系对于二维坐标系来讲只是多了一个 Z 轴，所以实际上在三维场景中物体的 AABB 碰撞检测依然可以采用四个点信息的判定来实现。即从物体 A 的八个顶点与物体 B 的八个顶点分别选出两个最大与最小的顶点进行对比。三维物体的 AABB 包围盒的八个顶点依旧可以用两个顶点来标识，如图所示：

只要确定了图中黑色点部分的坐标，就可以确定八个顶点的全部信息了。

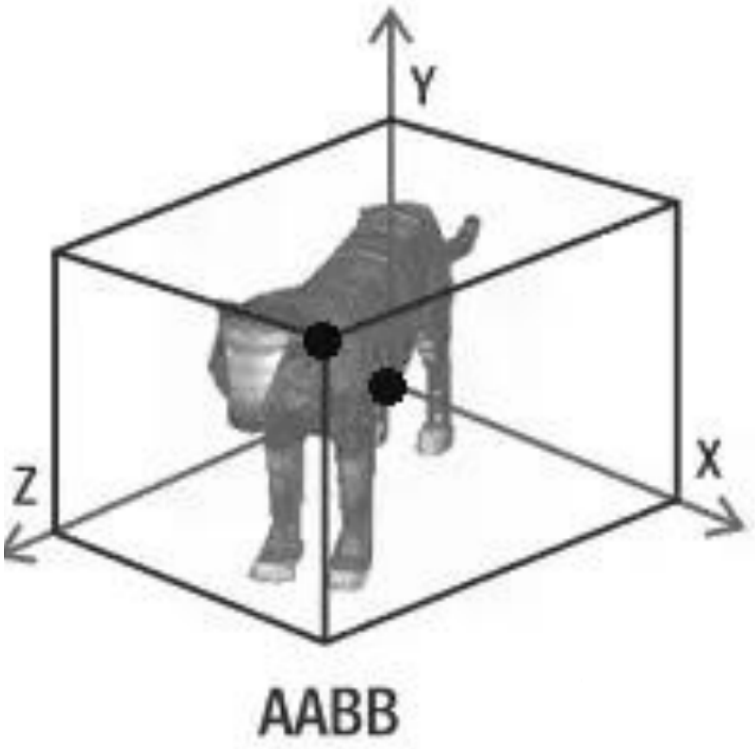

Figure 8: 3d collision

具体代码参考附录。

4 附录

4.1 参考源码与参考链接

Cocos2d-x 教程一三维物体 AABB 碰撞检测算法
[算法][包围盒] 球，AABB，OBB

碰撞检测 - GJK(2)
GJK(1)
3D 碰撞检测
GJK 算法详解
三角剖分算法 (delaunay)

4.2 泰森多边形的函数和作图实现代码

4.2.0.1 作图实现

```
% load('color_Sky02_h.mat')
clf;
xdot1=xdot1*100;
figure(1)
hold on
plot(xdot1(:,1),xdot1(:,2),'ko','MarkerFaceColor','k')
for j=1:size(trimat,1)
    plot([xdot1(trimat(j,1),1),xdot1(trimat(j,2),1)],[xdot1(trimat(j,1),2),xdot1(trimat(j,2),2)],'k-')
    plot([xdot1(trimat(j,1),1),xdot1(trimat(j,3),1)],[xdot1(trimat(j,1),2),xdot1(trimat(j,3),2)],'k-')
    plot([xdot1(trimat(j,3),1),xdot1(trimat(j,2),1)],[xdot1(trimat(j,3),2),xdot1(trimat(j,2),2)],'k-')
end
hold off
xlim([0,50]);ylim([0,50]);
xdot1=xdot1*1/100;

trimatcenter=trimatcenter*100;
xdot1=xdot1*100;
%绘制非边缘泰勒多边形
figure(2)
plot(xdot1(:,1),xdot1(:,2),'ko','MarkerFaceColor','k')
Thi_edge1=unique(Thi_edge1,'stable','rows');
xlim([0,50]);ylim([0,50]);
hold on
for j=1:size(Thi_edge1,1)
    plot(trimatcenter([Thi_edge1(j,1),Thi_edge1(j,2)],1),trimatcenter([Thi_edge1(j,1),Thi_edge1(j,2)],2),'color',[0,0.4,0])
end
trimatcenter=trimatcenter*1/100;
xdot1=xdot1*1/100;


figure(3)
% colormap(mycolor)
xlim([0,50]);ylim([0,50]);
%依次根据xdot1来循环
for j=4:size(xdot1,1)
    %trimatcenter
    [temp_trimat,temp_trimat_no,panduan]=findpoint2tri(j,trimat);%找出这一圈三角形
    if panduan
        %直接把temp_trimat_no的点当做泰森多边形即可
        trimatcenter=trimatcenter*100;
        patch('Faces',temp_trimat_no,'Vertices',trimatcenter,'FaceVertexCData',rand(1,1),'FaceColor','flat');
        'hyh';
```

```
            trimatcenter=trimatcenter*1/100;
        else
            %需要做边缘化处理
            %如果最终三角形不是边缘三角形，要注意不能适用内外部延长的准则，要用连接法则
            %选出中心点在边界内的一圈三角形
            [temp_trimat,temp_trimat_no,delleft,delright]=selecttemp_trimat(temp_trimat,
    temp_trimat_no,trimatcenter);
            t_t=1;
                % 如果selecttemp_trimat删除了边上的点，那么做边缘延长线的话就用和删除点之间作为连
    线，和边相交即可
                %即如果delleft,delright=0，照旧。如果不是0，利用temp_trimat_no(1)和delleft做连线（创
    建一个公式）
            tempboderedge=trimatcenter(temp_trimat_no,:);
            %边缘延长线1
            if delleft==0
                tempedge=temp_trimat(1,[1,2]);
                tempboderdot1=edgepointfind(tempedge,xdot1,trimatcenter(temp_trimat_no(1),:),
    border_point);%获取边缘点
                if size(tempboderdot1,1)==0
                    t_t=0;%边缘点超出，不再画面
                end
            else
                tempboderdot1=edgepointfind2(trimatcenter(temp_trimat_no(1),:),trimatcenter(delleft
    ,:));
            end
            tempboderedge=[tempboderdot1;tempboderedge];

            %边缘延长线2
            if delright==0
                tempedge=temp_trimat(end,[1,3]);
                tempboderdot2=edgepointfind(tempedge,xdot1,trimatcenter(temp_trimat_no(end),:),
    border_point);%获取边缘点
                if size(tempboderdot2,1)==0
                    t_t=0;%边缘点超出，不再画面
                end
            else
                tempboderdot2=edgepointfind2(trimatcenter(temp_trimat_no(end),:),trimatcenter(
    delright,:));
            end
            tempboderedge=[tempboderedge;tempboderdot2];

            %绘制边缘图形

            if t_t==1
                if tempboderdot1(1)~=tempboderdot2(1)
                    if tempboderdot1(2)~=tempboderdot2(2)
                        %边缘延长线3(边界角点
                        tempboderedge=[tempboderedge;maketempboderdot(tempboderdot1,tempboderdot2)];
                    end
                end
                tempboderedge = tempboderedge* 100;
                patch('Faces',1:length(tempboderedge),'Vertices',tempboderedge,'FaceVertexCData',
    rand(1,1),'FaceColor','flat');
                tempboderedge = tempboderedge* 1/100;
            end

        end

    end
```

4.2.0.2 函数实现

```
    %判断点在三角形外接圆的哪个部分
```

```
function panduan=whereispoint(xy1,xy2,xy3,xy0)
%判断点在三角形外接圆的哪个部分
[a,b,r2]=maketricenter(xy1,xy2,xy3);
x0=xy0(1);y0=xy0(2);
if a+sqrt(r2)<x0
    %x0在圆的右侧
    panduan=2;
elseif (x0-a)^2+(y0-b)^2<r2
    %x0在圆内
    panduan=0;
else
    %在圆外
    panduan=1;
end
end

%做出三角形三点与内部1点之间的线段
function temp_edge=maketempedge(dot1,dot2,dot3,dot0,xdot)
%做出连接点与三角形之间的线
%每行包含2个点，4个坐标值，共3行
%xy1和xy0组成线段
temp_edge=zeros(3,6);
if xdot(dot1,1)<xdot(dot0,1)
    temp_edge(1,:)=[dot1,dot0,xdot(dot1,:),xdot(dot0,:)];
elseif xdot(dot1,1)==xdot(dot0,1)
    if xdot(dot1,2)<xdot(dot0,2)
        temp_edge(1,:)=[dot1,dot0,xdot(dot1,:),xdot(dot0,:)];
    else
        temp_edge(1,:)=[dot0,dot1,xdot(dot0,:),xdot(dot1,:)];
    end
else
    temp_edge(1,:)=[dot0,dot1,xdot(dot0,:),xdot(dot1,:)];
end
%xy2和xy0组成线段
if xdot(dot2,1)<xdot(dot0,1)
    temp_edge(2,:)=[dot2,dot0,xdot(dot2,:),xdot(dot0,:)];
elseif xdot(dot2,1)==xdot(dot0,1)
    if xdot(dot2,2)<xdot(dot0,2)
        temp_edge(2,:)=[dot2,dot0,xdot(dot2,:),xdot(dot0,:)];
    else
        temp_edge(2,:)=[dot0,dot2,xdot(dot0,:),xdot(dot2,:)];
    end
else
    temp_edge(2,:)=[dot0,dot2,xdot(dot0,:),xdot(dot2,:)];
end
%xy3和xy0组成线段
if xdot(dot3,1)<xdot(dot0,1)
    temp_edge(3,:)=[dot3,dot0,xdot(dot3,:),xdot(dot0,:)];
elseif xdot(dot3,1)==xdot(dot0,1)
    if xdot(dot3,2)<xdot(dot0,2)
        temp_edge(3,:)=[dot3,dot0,xdot(dot3,:),xdot(dot0,:)];
    else
        temp_edge(3,:)=[dot0,dot3,xdot(dot0,:),xdot(dot3,:)];
    end
else
    temp_edge(3,:)=[dot0,dot3,xdot(dot0,:),xdot(dot3,:)];
end

end

%做出一些列固定点发散的线段外点组成的三角形
function temp_trimat=maketemptri(temp_edgemat,xdot,dot0)
%将edge buffer中的边与当前的点进行组合成若干三角形
%temp_edgemat是新边，x是中心点
```

```
%思路是计算各个边对应角度，然后排序相连

A=temp_edgemat(:,1:2);
pointline=A(A~=dot0);
N=length(pointline);
pointaxe=xdot(pointline,:);
img_pointaxe=pointaxe(:,1)+1i*pointaxe(:,2);
d_img_pointaxe=img_pointaxe-xdot(dot0,1)-1i*xdot(dot0,2);
angle_d_img_pointaxe=angle(d_img_pointaxe);
[~,index]=sort(angle_d_img_pointaxe);
index=[index;index(1)];%排序，然后依次串起来
temp_trimat=zeros(N,3);
for j=1:N
    temp_trimat(j,:)=[pointline(index(j)),pointline(index(j+1)),dot0];
end


end

%将三个点构成3条边
function edgemat=makeedge(dot1,dot2,dot3,xdot)
%将dot1 2 3这三个点构成三条边
%每行包含2个点，4个坐标值，共3行
edgemat=zeros(3,6);
%点12
if xdot(dot1,1)<xdot(dot2,1)
    edgemat(1,:)=[dot1,dot2,xdot(dot1,:),xdot(dot2,:)];
elseif xdot(dot1,1)==xdot(dot2,1)
    if xdot(dot1,2)<xdot(dot2,2)
        edgemat(1,:)=[dot1,dot2,xdot(dot1,:),xdot(dot2,:)];
    else
        edgemat(1,:)=[dot2,dot1,xdot(dot2,:),xdot(dot1,:)];
    end
else
    edgemat(1,:)=[dot2,dot1,xdot(dot2,:),xdot(dot1,:)];
end
%点13
if xdot(dot1,1)<xdot(dot3,1)
    edgemat(2,:)=[dot1,dot3,xdot(dot1,:),xdot(dot3,:)];
elseif xdot(dot1,1)==xdot(dot3,1)
    if xdot(dot1,2)<xdot(dot3,2)
        edgemat(2,:)=[dot1,dot3,xdot(dot1,:),xdot(dot3,:)];
    else
        edgemat(2,:)=[dot3,dot1,xdot(dot3,:),xdot(dot1,:)];
    end
else
    edgemat(2,:)=[dot3,dot1,xdot(dot3,:),xdot(dot1,:)];
end
%点23
if xdot(dot3,1)<xdot(dot2,1)
    edgemat(3,:)=[dot3,dot2,xdot(dot3,:),xdot(dot2,:)];
elseif xdot(dot3,1)==xdot(dot2,1)
    if xdot(dot3,2)<xdot(dot2,2)
        edgemat(3,:)=[dot3,dot2,xdot(dot3,:),xdot(dot2,:)];
    else
        edgemat(3,:)=[dot2,dot3,xdot(dot2,:),xdot(dot3,:)];
    end
else
    edgemat(3,:)=[dot2,dot3,xdot(dot2,:),xdot(dot3,:)];
end
% edgemat
end

%求三角形外接圆圆心
```

```
function [a,b,r2]=maketricenter(xy1,xy2,xy3)
x1=xy1(1);y1=xy1(2);
x2=xy2(1);y2=xy2(2);
x3=xy3(1);y3=xy3(2);
a=((y2-y1)*(y3*y3-y1*y1+x3*x3-x1*x1)-(y3-y1)*(y2*y2-y1*y1+x2*x2-x1*x1))/(2.0*((x3-x1)*(y2-y1)-(x2-x1)*(y3-y1)));
b=((x2-x1)*(x3*x3-x1*x1+y3*y3-y1*y1)-(x3-x1)*(x2*x2-x1*x1+y2*y2-y1*y1))/(2.0*((y3-y1)*(x2-x1)-(y2-y1)*(x3-x1)));
r2=(x1-a)*(x1-a)+(y1-b)*(y1-b);
end

%求边缘三角形
function [border_trimat,border_point,trimat_con]=makebordertri(trimat)
N=size(trimat,1);
border_trimat=[];
border_point=[];
trimat_con=zeros(N,3);
for j=1:N
    %tempborder_trimat=zeros(3,3);
    temptri=trimat(j,:);
    %计算temptri中12点边对应的三角形有哪些
    edgetrimat=find(sum(trimat==temptri(1),2)+sum(trimat==temptri(2),2)==2);
    edgetrimat(edgetrimat==j)=[];
    if size(edgetrimat,2)==0
        %这个边没有三角形相连，是个临边。
        border_point=[border_point;[temptri(1),temptri(2)]];

    elseif size(edgetrimat,2)==1
        %这个边没有三角形相连，是个临边。
        %tempborder_trimat(1,:)=trimat(edgetrimat,:);%记录三角形三点坐标
        trimat_con(j,1)=edgetrimat;%trimat_con记录上相邻三角形
    end

    %计算temptri中23点边对应的三角形有哪些
    edgetrimat=find(sum(trimat==temptri(2),2)+sum(trimat==temptri(3),2)==2);
    edgetrimat(edgetrimat==j)=[];
    if size(edgetrimat,2)==0
        border_point=[border_point;[temptri(2),temptri(3)]];

    elseif size(edgetrimat,2)==1
        %tempborder_trimat(2,:)=trimat(edgetrimat,:);
        trimat_con(j,2)=edgetrimat;
    end

    %计算temptri中31点边对应的三角形有哪些
    edgetrimat=find(sum(trimat==temptri(3),2)+sum(trimat==temptri(1),2)==2);
    edgetrimat(edgetrimat==j)=[];
    if size(edgetrimat,2)==0
        border_point=[border_point;[temptri(3),temptri(1)]];

    elseif size(edgetrimat,2)==1
        %tempborder_trimat(3,:)=trimat(edgetrimat,:);
        trimat_con(j,3)=edgetrimat;
    end

    %tempborder_trimat(all(tempborder_trimat==0,2),:)=[];%删除0行
    if ~all(trimat_con(j,:))
        %如果边缘三角少于3个,就添加
        border_trimat=[border_trimat;temptri];
    end

end

%把边首尾排序一遍,输出border_point
```

```
    for j=1:size(border_point,1)-1
        border_pointtemp=find(sum(border_point==border_point(j,2),2)==1);
        border_pointtemp(border_pointtemp==j)=[];%删除自己
        border_point([j+1,border_pointtemp],:)=border_point([border_pointtemp,j+1],:);
        if border_point(j,2)==border_point(j+1,2)
            border_point(j+1,[1,2])=border_point(j+1,[2,1]);
        end
    end

    end


    %判断两个线段是否相交
    function panduan=crossornot(l1xy1,l1xy2,l2xy1,l2xy2)
    l1x1=l1xy1(1);l1y1=l1xy1(2);
    l1x2=l1xy2(1);l1y2=l1xy2(2);
    l2x1=l2xy1(1);l2y1=l2xy1(2);
    l2x2=l2xy2(1);l2y2=l2xy2(2);
    %先快速判断
    if    (max(l2x1,l2x2)<min(l1x1,l1x2))||(max(l2y1,l2y2)<min(l1y1,l1y2))||...
          (max(l1x1,l1x2)<min(l2x1,l2x2))||(max(l1y1,l1y2)<min(l2y1,l2y2))
        %如果判断为真，则一定不会相交
        panduan=0;
    else
        %如果判断为假，进一步差积判断
        if ((((l1x1-l2x1)*(l2y2-l2y1)-(l1y1-l2y1)*(l2x2-l2x1))*...
            ((l1x2-l2x1)*(l2y2-l2y1)-(l1y2-l2y1)*(l2x2-l2x1)))>0 ||...
            (((l2x1-l1x1)*(l1y2-l1y1)-(l2y1-l1y1)*(l1x2-l1x1))*...
            ((l2x2-l1x1)*(l1y2-l1y1)-(l2y2-l1y1)*(l1x2-l1x1)))>0)
            %如果判断为真，则不会相交
            panduan=0;
        else
            panduan=1;
        end
    end
    end

    %两个向量做差积
    function t=crossdot(xy1,xy2)
    x1=xy1(1);y1=xy1(2);
    x2=xy2(1);y2=xy2(2);
    t=x1*y2-y1*x2;
    end

    %点是否在三角形内
    function panduan=pointintriangle(xy1,xy2,xy3,xy0)
    x1=xy1(1);y1=xy1(2);
    x2=xy2(1);y2=xy2(2);
    x3=xy3(1);y3=xy3(2);
    x0=xy0(1);y0=xy0(2);
    PA=[x1-x0,y1-y0];PB=[x2-x0,y2-y0];PC=[x3-x0,y3-y0];
    %利用差积同正或同负号来判断是否在三角内
    t1=crossdot(PA,PB);
    t2=crossdot(PB,PC);
    t3=crossdot(PC,PA);
    if abs(sign(t1)+sign(t2)+sign(t3))==3
        panduan=1;
    else
        panduan=0;
    end

    end

    %点是否在多边形内
```

```
function panduan=pointinmutiangle(xdot,d_no,xy0)
%d_no符合12341的格式，收尾相连
Ndot=xdot(d_no,:);
PN=[Ndot(:,1)-xy0(1),Ndot(:,2)-xy0(2)];
tn=zeros(length(d_no)-1,1);
for j=1:length(d_no)-1
    tn(j)=crossdot(PN(j,:),PN(j+1,:));
end
%利用差积同正或同负号来判断是否在三角内

if abs(sum(sign(tn)))==length(d_no)-1
    panduan=1;
else
    panduan=0;
end

end

%做出同时具有一个中心点的一圈三角形，按照相接顺序排序
function [temp_trimat,temp_trimat_no,panduan]=findpoint2tri(j,trimat)
temp_trimat_no=1:size(trimat,1);
panduan=1;
temp_trimat=trimat(logical(sum(trimat==j,2)),:);
temp_trimat_no=temp_trimat_no(logical(sum(trimat==j,2)));
%把j放到每行第一个
for k=1:size(temp_trimat,1)
    temp_trimat(k,[find(temp_trimat(k,:)==j),1])=temp_trimat(k,[1,find(temp_trimat(k,:)==j)]);
end
%如果有一个点只出现过一次，把这个点包含的三角形放到第一行
tN2=tabulate(reshape(temp_trimat,[],1));
tN3=tN2((tN2(:,2)==1),1);
%其实如果中心点不在范围内也最好单列出来，挖坑

if size(tN3,1)~=0
    tN3=tN3(1);
    tN=find(sum(temp_trimat==tN3,2));
    temp_trimat([1,tN],:)=temp_trimat([tN,1],:);
    temp_trimat_no([1,tN])=temp_trimat_no([tN,1]);
    panduan=0;
    if temp_trimat(1,3)==tN3
        temp_trimat(1,[2,3])=temp_trimat(1,[3,2]);
    end
end


%把边首尾排序一遍
for k=2:size(temp_trimat,1)

    tN=find(sum(temp_trimat==temp_trimat(k-1,3),2));
    tN(tN==k-1)=[];
    temp_trimat([tN,k],:)=temp_trimat([k,tN],:);
    if temp_trimat(k-1,3)~=temp_trimat(k,2)
        temp_trimat(k,[2,3])=temp_trimat(k,[3,2]);
    end
    temp_trimat_no([tN,k])=temp_trimat_no([k,tN]);
end
end

%做出一点对边缘的中垂线，得到边缘点坐标
function tempboderdot=edgepointfind(tempedge,xdot,xy0,border_point)

x0=xy0(1);y0=xy0(2);
%判断中心点是否在大图形内
d_no=[border_point(1,1);border_point(:,2)];
```

```
    panduan=pointinmutiangle(xdot,d_no,xy0);
    %求边的中点
    xz=(xdot(tempedge(1),1)+xdot(tempedge(2),1))/2;
    yz=(xdot(tempedge(1),2)+xdot(tempedge(2),2))/2;
    %做中心点延长线，得到一条超长线段定为2得了，用到了边界为1这个条件
    panduan2=true;
    if panduan
        %在内部，做中心到边缘延长线即可
        t_xy1=[x0,y0]+[xz-x0,yz-y0]*2/sqrt((xz-x0)^2+(yz-y0)^2);

    else
        if ~(x0<0||x0>1||y0<0||y0>1)
        %在外部,但是没超出边界，做中心到反向边缘
            t_xy1=[x0,y0]-[xz-x0,yz-y0]*2/sqrt((xz-x0)^2+(yz-y0)^2);
        else
            %在边界外
            panduan2=false;
            tempboderdot=[];%返回空矩阵
        end
    end

    if panduan2
        %判断是4个边哪一个
        [xy3,xy4]=select4edge(xy0,t_xy1);
        %做4点求交点运算
        tempboderdot=crosspoint(xy0,t_xy1,xy3,xy4);
    end
    %       panduan
    %       d_no
    %       xy0
    %       t_xy1
    %       tempboderdot
    end

    %做出边缘界外连线，得到边缘点坐标
    function tempboderdot1=edgepointfind2(xy1,xy2)
    [xy3,xy4]=select4edge(xy1,xy2);
    tempboderdot1=crosspoint(xy1,xy2,xy3,xy4);
    end
    %判断射线会与哪条边相交
    function [xy3,xy4]=select4edge(xy0,xy1)
    deg01=angle((0-xy0(1))+(0-xy0(2))*1i);%左下00
    deg02=angle((1-xy0(1))+(0-xy0(2))*1i);%10
    deg03=angle((1-xy0(1))+(1-xy0(2))*1i);%11
    deg04=angle((0-xy0(1))+(1-xy0(2))*1i);%01
    deg00=angle((xy1(1)-xy0(1))+(xy1(2)-xy0(2))*1i);
    [~,I]=sort([deg00,deg01,deg02,deg03,deg04]);
    k=find(I==1);
    switch k
        case 1
            xy3=[0,0];xy4=[0,1];
        case 2
            xy3=[0,0];xy4=[1,0];
        case 3
            xy3=[1,0];xy4=[1,1];
        case 4
            xy3=[0,1];xy4=[1,1];
        case 5
            xy3=[0,0];xy4=[0,1];
    end
    end

    %求两条线交点
    function xy0=crosspoint(xy1,xy2,xy5,xy6)
```

```
a1=(xy2(2)-xy1(2));b1=(xy1(1)-xy2(1));c1=xy1(1)*xy2(2)-xy2(1)*xy1(2);
a2=(xy6(2)-xy5(2));b2=(xy5(1)-xy6(1));c2=xy5(1)*xy6(2)-xy6(1)*xy5(2);
xy0=[det([c1,b1;c2,b2])/det([a1,b1;a2,b2]),det([a1,c1;a2,c2])/det([a1,b1;a2,b2])];
end


%删除所有中心点超出边界的三角形
function [temp_trimat,temp_trimat_no,delleft,delright]=selecttemp_trimat(temp_trimat,
temp_trimat_no,trimatcenter)
delleft=0;delright=0;
for j=1:size(temp_trimat,1)
    centerdot=trimatcenter(temp_trimat_no(j),:);
    if and(and(0<centerdot(1),centerdot(1)<1),and(0<centerdot(2),centerdot(2)<1))
        break
    end
end
if j~=1
    delleft=temp_trimat_no(j-1);
end
temp_trimat(1:j-1,:)=[];
temp_trimat_no(1:j-1)=[];

%倒着来一遍
temp_trimat=flipud(temp_trimat);
temp_trimat_no=fliplr(temp_trimat_no);
for j=1:size(temp_trimat,1)
    centerdot=trimatcenter(temp_trimat_no(j),:);
    if and(and(0<centerdot(1),centerdot(1)<1),and(0<centerdot(2),centerdot(2)<1))
        break
    end
end
if j~=1
    delright=temp_trimat_no(j-1);
end
temp_trimat(1:j-1,:)=[];
temp_trimat_no(1:j-1)=[];

temp_trimat=flipud(temp_trimat);
temp_trimat_no=fliplr(temp_trimat_no);

end


%利用两个边缘点求角点
function tempboderdot3=maketempboderdot(tempboderdot1,tempboderdot2)
tempboderdot3=zeros(1,2);
%第一个边
if abs(tempboderdot1(1)-0)<1e-10
    tempboderdot3(1)=0;
end
if abs(tempboderdot1(1)-1)<1e-10
    tempboderdot3(1)=1;
end
if abs(tempboderdot1(2)-0)<1e-10
    tempboderdot3(2)=0;
end
if abs(tempboderdot1(2)-1)<1e-10
    tempboderdot3(2)=1;
end
%第二个边
if abs(tempboderdot2(1)-0)<1e-10
    tempboderdot3(1)=0;
end
if abs(tempboderdot2(1)-1)<1e-10
```

```
        tempboderdot3(1)=1;
    end
    if abs(tempboderdot2(2)-0)<1e-10
        tempboderdot3(2)=0;
    end
    if abs(tempboderdot2(2)-1)<1e-10
        tempboderdot3(2)=1;
    end
    end
```

4.3 GJK 算法

4.3.0.1 函数实现-GJK.m

```
function flag = GJK(shape1,shape2,iterations)
% GJK Gilbert-Johnson-Keerthi Collision detection implementation.
% Returns whether two convex shapes are are penetrating or not
% (true/false). Only works for CONVEX shapes.
%
% Inputs:
%   shape1:
%   must have fields for XData,YData,ZData, which are the x,y,z
%   coordinates of the vertices. Can be the same as what comes out of a
%   PATCH object. It isn't required that the points form faces like patch
%   data. This algorithm will assume the convex hull of the x,y,z points
%   given.
%
%   shape2:
%   Other shape to test collision against. Same info as shape1.
%
%   iterations:
%   The algorithm tries to construct a tetrahedron encompassing
%   the origin. This proves the objects have collided. If we fail within a
%   certain number of iterations, we give up and say the objects are not
%   penetrating. Low iterations means a higher chance of false-NEGATIVES
%   but faster computation. As the objects penetrate more, it takes fewer
%   iterations anyway, so low iterations is not a huge disadvantage.
%
% Outputs:
%   flag:
%   true - objects collided
%   false - objects not collided
%
%
%   This video helped me a lot when making this: https://mollyrocket.com/849
%   Not my video, but very useful.
%
%   Matthew Sheen, 2016
%

%Point 1 and 2 selection (line segment)
%v = [0.8 0.5 1];
v = [1,0,0; 0,1,0;0,0,1;-1,0,0;0,-1,0;0,0,-1;]
for  i=1:6
     v1=v(i,:);
    [a,b] = pickLine(v1,shape2,shape1);
    %Point 3 selection (triangle)
    [a,b,c,flag] = pickTriangle(a,b,shape2,shape1,iterations);
     if flag==1
        break;
     end
end
```

```
49
50
51      %Point 4 selection (tetrahedron)
52      if flag == 1 %Only bother if we could find a viable triangle.
53          [a,b,c,d,flag] = pickTetrahedron(a,b,c,shape2,shape1,iterations);
54      end
55
56      end
57
58      function [a,b] = pickLine(v,shape1,shape2)
59      %Construct the first line of the simplex
60      b = support(shape2,shape1,v);
61      a = support(shape2,shape1,-v);
62      end
63
64      function [a,b,c,flag] = pickTriangle(a,b,shape1,shape2,IterationAllowed)
65      flag = 0; %So far, we don't have a successful triangle.
66
67      %First try:
68      ab = b-a;
69      ao = -a;
70      v = cross(cross(ab,ao),ab); % v is perpendicular to ab pointing in the general direction of the
        origin.
71
72      c = b;
73      b = a;
74      a = support(shape2,shape1,v);
75
76      for i = 1:IterationAllowed %iterations to see if we can draw a good triangle.
77          %Time to check if we got it:
78          ab = b-a;
79          ao = -a;
80          ac = c-a;
81
82          %Normal to face of triangle
83          abc = cross(ab,ac);
84
85          %Perpendicular to AB going away from triangle
86          abp = cross(ab,abc);
87          %Perpendicular to AC going away from triangle
88          acp = cross(abc,ac);
89
90          %First, make sure our triangle "contains" the origin in a 2d projection
91          %sense.
92          %Is origin above (outside) AB?
93          if dot(abp,ao) > 0
94              c = b; %Throw away the furthest point and grab a new one in the right direction
95              b = a;
96              v = abp; %cross(cross(ab,ao),ab);
97
98             %Is origin above (outside) AC?
99          elseif dot(acp, ao) > 0
100             b = a;
101             v = acp; %cross(cross(ac,ao),ac);
102
103         else
104             flag = 1;
105             break; %We got a good one.
106         end
107         a = support(shape2,shape1,v);
108     end
109     end
110
111     function [a,b,c,d,flag] = pickTetrahedron(a,b,c,shape1,shape2,IterationAllowed)
```

```
    %Now, if we're here, we have a successful 2D simplex, and we need to check
    %if the origin is inside a successful 3D simplex.
    %So, is the origin above or below the triangle?
    flag = 0;

    ab = b-a;
    ac = c-a;

    %Normal to face of triangle
    abc = cross(ab,ac);
    ao = -a;

    if dot(abc, ao) > 0 %Above
        d = c;
        c = b;
        b = a;

        v = abc;
        a = support(shape2,shape1,v); %Tetrahedron new point

    else %below
        d = b;
        b = a;
        v = -abc;
        a = support(shape2,shape1,v); %Tetrahedron new point
    end

    for i = 1:IterationAllowed %Allowing 10 tries to make a good tetrahedron.
        %Check the tetrahedron:
        ab = b-a;
        ao = -a;
        ac = c-a;
        ad = d-a;

        %We KNOW that the origin is not under the base of the tetrahedron based on
        %the way we picked a. So we need to check faces ABC, ABD, and ACD.

        %Normal to face of triangle
        abc = cross(ab,ac);

        if dot(abc, ao) > 0 %Above triangle ABC
            %No need to change anything, we'll just iterate again with this face as
            %default.
        else
            acd = cross(ac,ad);%Normal to face of triangle

            if dot(acd, ao) > 0 %Above triangle ACD
                %Make this the new base triangle.
                b = c;
                c = d;
                ab = ac;
                ac = ad;
                abc = acd;
            elseif dot(acd, ao) < 0
                adb = cross(ad,ab);%Normal to face of triangle

                if dot(adb, ao) > 0 %Above triangle ADB
                    %Make this the new base triangle.
                    c = b;
                    b = d;
                    ac = ab;
                    ab = ad;
                    abc = adb;
                else
```

```
                    flag = 1;
                    break; %It's inside the tetrahedron.
                end
            end
        end

        %try again:
        if dot(abc, ao) > 0 %Above
            d = c;
            c = b;
            b = a;
            v = abc;
            a = support(shape2,shape1,v); %Tetrahedron new point
        else %below
            d = b;
            b = a;
            v = -abc;
            a = support(shape2,shape1,v); %Tetrahedron new point
        end
    end

    end

    function point = getFarthestInDir(shape, v)
    %Find the furthest point in a given direction for a shape
    XData = shape.XData; % Making it more compatible with previous MATLAB releases.
    YData = shape.YData;
    ZData = shape.ZData;
    dotted = XData*v(1) + YData*v(2) + ZData*v(3);
    [maxInCol,rowIdxSet] = max(dotted);
    [maxInRow,colIdx] = max(maxInCol);
    rowIdx = rowIdxSet(colIdx);
    point = [XData(rowIdx,colIdx), YData(rowIdx,colIdx), ZData(rowIdx,colIdx)];
    end

    function point = support(shape1,shape2,v)
    %Support function to get the Minkowski difference.
    point1 = getFarthestInDir(shape1, v);
    point2 = getFarthestInDir(shape2, -v);
    point = point1 - point2;
    end
```

4.3.0.2 函数实现-MAIN$_{e}xample_{c}ompatible.m$

```
% Example script for GJK function
%   Animates two objects on a collision course and terminates animation
%   when they hit each other. Loads vertex and face data from
%   SampleShapeData.m. See the comments of GJK.m for more information
%
%   Most of this script just sets up the animation and transformations of
%   the shapes. The only key line is:
%   collisionFlag = GJK(S1Obj,S2Obj,iterationsAllowed)
clc;clear all;close all

%How many iterations to allow for collision detection.
iterationsAllowed = 6;

% Make a figure
fig = figure;
hold on

% Load sample vertex and face data for two convex polyhedra
SampleShapeData;
```

```
20
21      % Make shape 1
22      S1.Vertices = V1;
23      S1.Faces = F1;
24      S1.FaceVertexCData = jet(size(V1,1));
25      S1.FaceColor = 'interp';
26      S1Obj = patch(S1);
27
28      % Make shape 2
29      S2.Vertices = V2;
30      S2.Faces = F2;
31      S2.FaceVertexCData = jet(size(V2,1));
32      S2.FaceColor = 'interp';
33      S2Obj = patch(S2);
34
35      % Make shape 3
36      S3.Vertices = V3;
37      S3.Faces = F3;
38      S3.FaceVertexCData = jet(size(V3,1));
39      S3.FaceColor = 'interp';
40      S3Obj = patch(S3);
41
42      hold off
43      axis equal
44      axis([-5 5 -5 5 -5 5])
45      ax = get(fig,'Children');
46      set(ax,'Visible','off'); % Turn off the axis for more pleasant viewing.
47      set(fig,'Color',[1 1 1]);
48      rotate3d on;
49
50      %Move them through space arbitrarily.
51      S1Coords = get(S1Obj,'Vertices');
52      S2Coords = get(S2Obj,'Vertices');
53      S3Coords = get(S2Obj,'Vertices');
54
55
56
57      % Animation loop. Terminates on collision.
58      for i = 3:-0.01:0.2;
59
60          % Do collision detection
61          collisionFlag1 = GJK(S1Obj,S2Obj,iterationsAllowed);
62          collisionFlag2 = GJK(S1Obj,S3Obj,iterationsAllowed);
63          collisionFlag3 = GJK(S3Obj,S2Obj,iterationsAllowed);
64          drawnow;
65
66          if collisionFlag1
67              t = text(3,3,3,'1-Collision!','FontSize',30);
68              break;
69          %else
70              %t = text(3,3,3,'No 1-Collision!','FontSize',30);
71              %break;
72          end
73          if collisionFlag2
74              t = text(3,3,3,'2-Collision!','FontSize',30);
75              break;
76          end
77          if collisionFlag3
78              t = text(3,3,3,'3-Collision!','FontSize',30);
79              break;
80          end
81      end
```

4.3.0.3 函数实现-SampleShapeData.m

```
V1 = [
     0 0 0
     1 0 0
     1 1 0
     0 1 0
     0 0 1
     1 0 1
     1 1 1
     0 1 1];

F1 = [1 4 5 8
     1 2 3 4
     1 2 5 6
     2 3 6 7
     3 4 7 8
     5 6 7 8
     ];

 V2 = [
      3 3 2
      5 3 2
 4 5 2
4 4 4
];

F2 = [       1       2       3
     1 2   4
     1 3 4
     2 3 4
];


 V3 = [
 1 1 1
 2 1 1
 2 2 1
1 2 1
1 1 2
2 1 2
2 2 2
1 2 2
];

F3 = [1 4 5 8
     1 2 3 4
     1 2 5 6
     2 3 6 7
     3 4 7 8
     5 6 7 8
];
```

4.3.0.4 函数实现-aabb.py

```
import numpy
def collides_with(lower_bound,upper_bound,lower_bound_compare,upper_bound_compare):

    a=lower_bound[0]
    b=lower_bound[1]
    c=upper_bound[0]
    d=upper_bound[1]
    e=lower_bound_compare[0]
```

```
        f=lower_bound_compare[1]
        g=upper_bound_compare[0]
        h=upper_bound_compare[1]


        x1=a<=e<=c
        x2=a<=g<=c
        y1=b<=f<=d
        y2=b<=h<=d

        collision=False

        if (x1 or x2) and (y1 or y2): collision=True

        return collision

    v1 = [[0,0,0],[1,0,0],[1,1,0],[0,1,0],[0,0,1],[1,0,1],[1,1,1],[0,1,1]]
    v1=numpy.array(v1)
    v1xmin=min(v1[:,0])
    v1xmax=max(v1[:,0])
    v1ymin=min(v1[:,1])
    v1ymax=max(v1[:,1])
    v1zmin=min(v1[:,2])
    v1zmax=max(v1[:,2])
    F1 = [[1,4,5,8],[1,2,3,4],[1,2,5,6],[2,3,6,7],[3,4,7,8],[5,6,7,8]];
    v2 = [[3,3,2],[5,3,2],[4,5,2], [4,4,4]];
    v2=numpy.array(v2)
    v2xmin=min(v2[:,0])
    v2xmax=max(v2[:,0])
    v2ymin=min(v2[:,1])
    v2ymax=max(v2[:,1])
    v2zmin=min(v2[:,2])
    v2zmax=max(v2[:,2])
    F2 = [[1,2,3],[1,2,4],[1,3,4],[2,3,4]];
    v3 = [[1,1,1],[2,1,1],[2,2,1],[1,2,1],[1,1,2],[2,1,2],[2,2,2],[1,2,2]];
    v3=numpy.array(v3)
    F3 = [[1,4,5,8],[1,2,3,4],
        [1,2,5,6],
        [2,3,6,7],
        [3,4,7,8],
        [5,6,7,8],
    ];
    v3xmin=min(v3[:,0])
    v3xmax=max(v3[:,0])
    v3ymin=min(v3[:,1])
    v3ymax=max(v3[:,1])
    v3zmin=min(v3[:,2])
    v3zmax=max(v3[:,2])

    v1lower_bound=(v1xmin,v1ymin,v1zmin)
    v1upper_bound=(v1xmax,v1ymax,v1zmax)
    v2lower_bound=(v2xmin,v2ymin,v2zmin)
    v2upper_bound=(v2xmax,v2ymax,v2zmax)
    v3lower_bound=(v3xmin,v3ymin,v3zmin)
    v3upper_bound=(v3xmax,v3ymax,v3zmax)
    v1.__init__()


    collision1=collides_with(v1lower_bound,v1upper_bound,v2lower_bound,v2upper_bound)
    collision2=collides_with(v1lower_bound,v1upper_bound,v3lower_bound,v3upper_bound)
    collision3=collides_with(v2lower_bound,v2upper_bound,v3lower_bound,v3upper_bound)

    if collision1:
        print(collision1)
```

```
73          print('collision between object 1 and object 2')
74      if collision2:
75          print(collision2)
76          print('collision between object 1 and object 3')
77      if collision3:
78          print(collision3)
79          print('collision between object 2 and object 3')
```