function vdot(a,b){ //produto escalar

return a[0]\*b[0] + a[1]\*b[1]

}

function rot90(v){

return [-v[1],v[0]]

}

function vmul(s,v){

return [s\*v[0], s\*v[1]]

}

function vadd(a,b){

return [a[0]+b[0], a[1]+b[1]]

}

function vadds(a,b,s){

return [a[0]+b[0]\*s, a[1]+b[1]\*s]

}

function vsub(a, b) {

return [a[0] - b[0], a[1] - b[1]];

}

class ponto{

constructor(){

this.p1 = [0,0]

}

edges(){

return [this.p1,this.p1]

}

vertices(){

return [this.p1];

}

render(){

colore(125,0,0)

circle(this.p1[0],this.p1[1],5)

}

}

class obb{

constructor(){

this.c = [0,0] // center

this.e = [25,25] // tamanho

this.angle = 0.93

this.u = [Math.cos(this.angle),Math.sin(this.angle)] // eixo x local

this.v = [-this.u[1],this.u[0]] // eixo y local

this.p1 = [0, 0];

this.p2 = [0, 0];

this.p3 = [0, 0];

this.p4 = [0, 0];

}

fit(pts){

this.pts = pts

}

randomize(){

this.angle = random(1)

this.u = [Math.cos(this.angle),Math.sin(this.angle)]

this.v = [-this.u[1],this.u[0]] // eixo y local

}

edges() {

return [

[this.p1, this.p2],

[this.p2, this.p3],

[this.p3, this.p4],

[this.p4, this.p1]]

}

vertices() {

return [this.p1, this.p2, this.p3, this.p4];

}

render(){

let pts = this.pts

let maxu = -Infinity

let minu = +Infinity

let maxv = -Infinity

let minv = +Infinity

colore(200)

for(let j=0;j<pts.length;j++){

let udotpi = vdot(this.u, pts[j]) // produto escalar ponto com eixo x local obb

let vdotpi = vdot(this.v, pts[j]) // produto escalar com eixo y local

maxu = max(udotpi,maxu) //maximo ou produto escalar ou o infinito

//ele define como max u qual o maior produto escalar, o do ponto anterior ou o desse

minu = min(udotpi,minu) // minimo ou produto escalar x ou infinito

maxv = max(vdotpi,maxv)

minv = min(vdotpi,minv)

let px = pts[j][0]

let py = pts[j][1]

let up = vmul(udotpi,this.u)

let vp = vmul(vdotpi, this.v)

//line(px,py,up[0],up[1])

// line(px,py, vp[0], vp[1])

}

let largurav = maxv - minv

let largura = maxu - minu

let medu = (maxu + minu) \* 0.5

let medv = (maxv + minv) \* 0.5

//o centro é o valor médio do máximo(max) e mínimo(min) em determinado eixo.

//neste caso, no x local (v)

this.c = vmul(medv, this.v) // medu\*this.u[0], \*this.u[1]

let centrou = vmul(medu,this.u)

this.c = vadd(this.c,centrou)

//this.u é vetor

//largura é o maximo menos o minimo daquele eixo

this.e[0] = largurav/2 // o e vai ser a metade da largura, multiplicando pelos valores

this.e[1] = largura/2

let w2 = this.e[0]

let h2 = this.e[1]

let p1 = vadds(vadds(this.c, this.v, +w2), this.u, +h2) // + h2\*this.v

let p2 = vadds(vadds(this.c, this.v, -w2), this.u, +h2)

let p3 = vadds(vadds(this.c, this.v, -w2), this.u, -h2)

let p4 = vadds(vadds(this.c, this.v, +w2), this.u, -h2)

this.p1 = p1;

this.p2 = p2;

this.p3 = p3;

this.p4 = p4;

colore(0)

noFill()

quad(

p1[0],p1[1],

p2[0],p2[1],

p3[0],p3[1],

p4[0],p4[1]

)

colore(64)

circle(this.c[0],this.c[1],5)

// os quatro vertices

circle(p1[0],p1[1],5)

circle(p2[0],p2[1],5)

circle(p3[0],p3[1],5)

circle(p4[0],p4[1],5)

//os eixos locais

colore(96,0,128)

let fu = vadds(this.c, this.u, w2)

let fv = vadds(this.c, this.v, h2)

//arrow(this.c[0],this.c[1],fu[0],fu[1])

colore(0,0,96)

//arrow(this.c[0],this.c[1],fv[0],fv[1])

}

}

function satAxes(obj) {

let edges = obj.edges();

let axes = [];

for (let j = 0; j < edges.length; j++) {

let ei = edges[j]; //lado da posição j

let dir\_ei = vsub(ei[1], ei[0]); // subtrai um ponto pelo outro, obtendo uma direção que a aresta segue

let nor\_ei = rot90(dir\_ei); //normal da direção da aresta, rotacionando 90

//normal a aresta i

axes.push(nor\_ei); //axes recebe a normal, que é um vetor rotacionado;

}

return axes;

}

function project(axis, obj) {

let points = obj.vertices();

let mi = +Infinity,

ma = -Infinity;

for (let i = 0; i < points.length; i++) {

let pi = points[i];

let proj = vdot(axis, pi); //produto escalar do eixo (normal) com um ponto do objeto

mi = min(mi, proj); // vai sempre atualizando o mi e ma, pra saber qual o ponto com maior e menor valor de produto escalar com a normal

ma = max(ma, proj);

}

return [mi, ma]; //retorna os valores mínimo e maximo

}

function separates(axis, obj1, obj2) {

let ab = project(axis, obj1);

let cd = project(axis, obj2);

let a = ab[0],

b = ab[1];

let c = cd[0],

d = cd[1];

return b < c || d < a ;

}

function sat(obj1, obj2) {

//

let axes = satAxes(obj1);

for (let i = 0; i < axes.length; i++) {

if (separates(axes[i], obj1, obj2)) {

return false

}

}

axes = satAxes(obj2);

for (let j = 0; j < axes.length; j++) {

if (separates(axes[j], obj1, obj2)) {

return false

}

}

return true; //ha colisao

}

let b = new obb()

let pontos = []

let b2 = new obb()

let pontos2 = []

let p1 = new ponto;

var mouseXC, mouseYC = 0

function keyPressed(){

b.randomize()

b2.randomize()

}

function setup() {

createCanvas(400, 400);

let z = random(50) // variáveis para gerar nuvens de pontos próximos

let q = random(50) // entre si, porém em posições aleatórias

for(let i=0;i<15;i++){

if (z<25){

pontos.push([random(-50,33),random(20,133)])

} else {

pontos.push([random(-70,20),random(-100,0)])

}

if (q<25){

pontos2.push([random(-150,-52),random(-77,50)])

}

else{

pontos2.push([random(-100,-17),random(-77,50)])

}

} //criando pontos

}

function mouseClicked() { //atualizando posição do ponto

p1.p1[0] = mouseX-width/2;

p1.p1[1] = -(mouseY-height/2);

}

function draw() {

goCartesian()

print (b.angle)

// pontos

colore(196,128,64)

pontos.forEach(p=>circle(p[0],p[1],3))

pontos2.forEach(p=>circle(p[0],p[1],3))

b.fit(pontos)

b2.fit(pontos2)

b2.render()

b.render()

p1.render(); //render do ponto

if (sat(b, b2)) {

texto("Colisao entre OBBs", -width / 2, height / 2 - 18);

}

if (sat(p1,b2) || sat(p1,b)){

colore(0,125,125)

texto("Colisão entre ponto e OBB", -width / 2, height / 2 - 36);

}

// pontos

//colore(196,128,64)

//pontos.forEach(p=>circle(p[0],p[1],3))

// b.fit(pontos)

// b.render()

}

function goCartesian()

{

background(255)

mouseXC = mouseX - width/2

mouseYC = height/2 - mouseY

colore(128,0,0)

arrow(0,height/2,width, height/2)

colore(0,128,0)

arrow(width/2,height,width/2, 0)

translate(width/2,height/2)

scale(1,-1,1)

}

// atualiza as variáveis globais com as coordenadas do mouse no plano cartesiano

function grabMouse()

{

mouseXC = mouseX - width/2

mouseYC = height/2 - mouseY

}

// renderiza texto corretamente no plano cartesiano

function texto(str,x,y)

{

push()

resetMatrix();

translate(width/2,height/2)

// desenha o texto normalmente

text(str,x,-y)

pop()

}

/\* Define as cores de preenchimento e de contorno com o mesmo valor.

\* Há várias opções de trabalho em RGB nesse caso:

\* - caso c1,c2,c3 e c4 sejam passados, o efeito padrão é uma cor RGBA

\* - caso c1,c2 e c3 sejam passados, tem-se uma cor RGB.

\* - caso c1 e c2 sejam passados, c1 é um tom de cinza e c2 é opacidade.

\* - caso apenas c1 seja passado, c1 é um tom de cinza.

\*/

function colore(c1,c2,c3,c4)

{

if(c4 != null)

{

fill(c1,c2,c3,c4)

stroke(c1,c2,c3,c4)

return

}

if(c3 != null)

{

fill(c1,c2,c3)

stroke(c1,c2,c3)

return

}

if(c2 == null )

{

fill(c1)

stroke(c1)

}

else

{

fill(c1,c1,c1,c2)

stroke(c1,c1,c1,c2)

}

}

/\* Desenha um segmento de reta com seta do ponto (x1,y1) para (x2,y2)

\*/

function arrow(x1,y1,x2,y2)

{

line(x1,y1,x2,y2)

var dx = x2-x1, dy = y2-y1

var le = sqrt(dx\*dx + dy\*dy)

var vx = dx/le, vy = dy/le

var ux = -vy

var uy = vx

triangle(x2,y2,

x2-5\*vx+2\*ux, y2-5\*vy+2\*uy,

x2-5\*vx-2\*ux, y2-5\*vy-2\*uy)

}