 Министерство образования Республики Беларусь

Учреждение образования  
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

                        Факультет      Информационных технологий и управления

                        Кафедра         Интеллектуальных информационных технологий

**ОТЧЁТ**

по дисциплине “Графический интерфейс интеллектуальных систем”

Лабораторная работа №5

Выполнил:

Кончатов С.Н., гр. 221703

Проверил:

Сальников Д.А.

Минск 2024

Задание: разработать элементарный графический редактор обработки полигонов.

Ход работы

Для создания программы использовалась библиотека python PyGame.

Нормаль к стороне (используется для нахождения точки пересечения)

    def fing\_norm(self,n:int)->list:

        if 0>self.edges[n].get\_v\_x()\*(self.edges[n+1].p\_2[1]-self.edges[n].p\_1[1])-\

                    self.edges[n].get\_v\_y()\*(self.edges[n+1].p\_2[0]-self.edges[n].p\_1[0]):

            return [self.edges[n].get\_v\_y(),-self.edges[n].get\_v\_x()]

        return [-self.edges[n].get\_v\_y(),self.edges[n].get\_v\_x()]

Пересечение отрезка со стороной полигона

    def find\_int\_point(self,line):

        int\_points=[]

        p1,p2=line

        d=[p2[0]-p1[0],p2[1]-p1[1]]

        for n in range(-2,len(self.edges)-2):

            n\_v=self.fing\_norm(n)

            f=self.edges[n].p\_1

            w=[p1[0]-f[0],p1[1]-f[1]]

            try:

                t=-(n\_v[0]\*w[0]+n\_v[1]\*w[1])/(n\_v[0]\*d[0]+n\_v[1]\*d[1])

                x=p1[0]+int((p2[0]-p1[0])\*t)

                y=p1[1]+int((p2[1]-p1[1])\*t)

                if 0<=t<=1:

                    try:

                        if not 0<=(x-f[0])/(self.edges[n].p\_2[0]-f[0])<=1:

                            continue

                    except:pass

                    try:

                        if not 0<=(y-f[1])/(self.edges[n].p\_2[1]-f[1])<=1:

                                continue

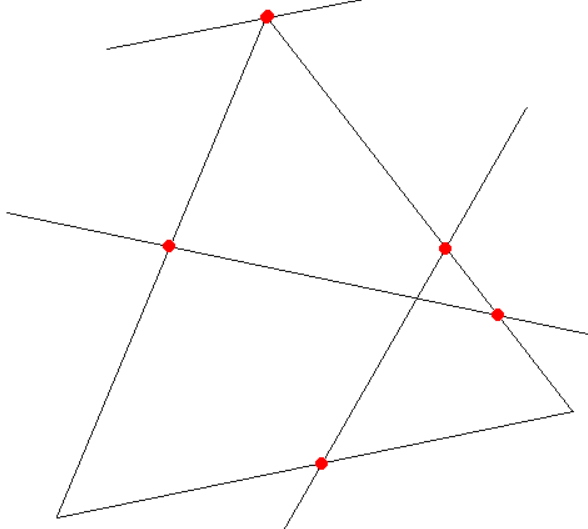
                    except:pass

                    int\_points.append([x,y])

            except:

                pass

        return int\_points



Проверка точки на принадлежность полигону (используется для определения выпуклый/вогнутый)

    def find\_point(self,x:int,y:int)->bool:

        if len(self.find\_int\_point([[x,y],[X,y]]))%2!=0:

            if self.convex\_check():

                print('ВЫПУКЛЫЙ')

            else:

                print('ВОГНУТЫЙ')

            return True

        else:

            return False

Проверка на выпуклый/вогнутый

    def convex\_check(self)->bool:

        v\_p=[(self.edges[n].get\_v\_y()\*self.edges[n-1].get\_v\_x()-\

              self.edges[n].get\_v\_x()\*self.edges[n-1].get\_v\_y()) for n in range(len(self.edges))]

        if v\_p[0]>0:

            for p in v\_p[1:]:

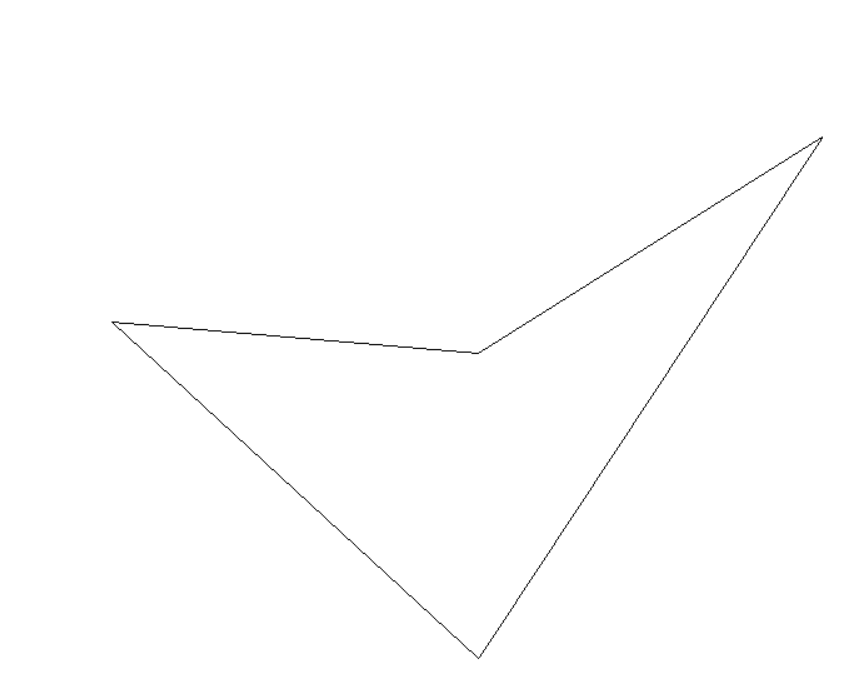
                if p<0:return False

        elif v\_p[0]<0:

            for p in v\_p[1:]:

                if p>0:return False

        return True





Алгоритм Грехема

class Greh\_alg():

    def find\_pol(points:list)->Polygon:

        def tan(point):

            try:

                f=math.atan((point[1]-p[1])/(point[0]-p[0]))

                if f<0:f+=math.pi

                return f

            except:

                return math.pi/2

        def shell\_build(p\_points:list,points:list):

            if not points:

                p\_points.append(Edge([p\_points[-1].p\_2,p\_points[0].p\_1]))

            elif (p\_points[-1].get\_v\_x())\*(points[0][1]-p\_points[-1].p\_2[1])\

                -(p\_points[-1].get\_v\_y())\*(points[0][0]-p\_points[-1].p\_2[0])>0:

                p\_points.append(Edge([p\_points[-1].p\_2,points[0]]))

                #print([point.get() for point in p\_points])

                shell\_build(p\_points,points[1:])

            else:

                p\_points.pop()

                #print([point.get() for point in p\_points])

                shell\_build(p\_points,points)

        p=points[0]

        for point in points[1:]:

            if point[1]<p[1]:

                p=point

        points.remove(p)

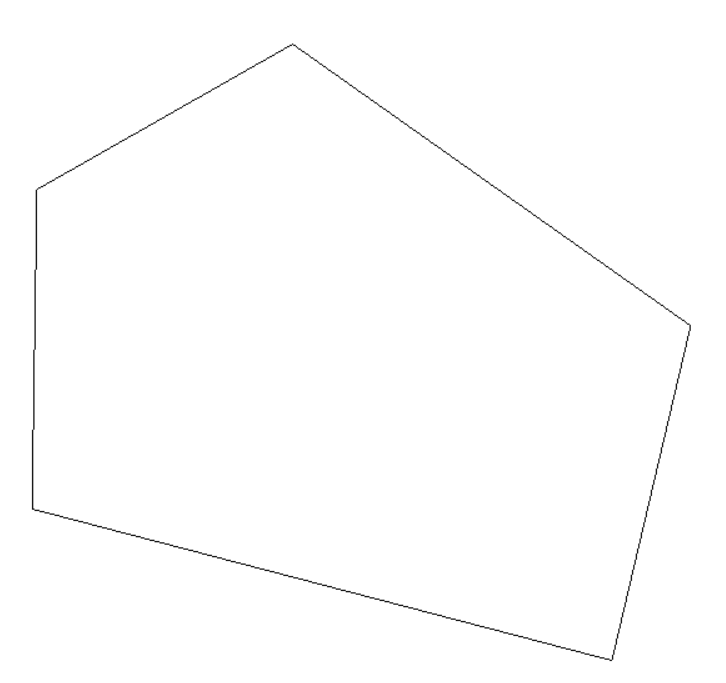
        points.sort(key=lambda point: tan(point))

        #print(points)

        p\_points=[Edge([p,points[0]])]

        shell\_build(p\_points,points[1:])

        return Polygon(p\_points)



Алгоритм Джара

class Djarw\_alg():

    def find\_pol(points:list)->Polygon:

        def tan(t:list,point:list)->float:

            try:

                f=math.atan((point[1]-t[1])/(point[0]-t[0]))

                if t[0]>point[0]:f+=math.pi

                return f

            except:

                return math.pi/2

        p,m=points[0],points[0]

        for point in points[1:]:

            if point[1]<p[1]:

                p=point

            elif point[1]>m[1]:m=point

        points.remove(p)

        '''m=points[0]

        for point in points[1:]:

             if point[1]>m[1]:m=point'''

        t=[p]

        p\_points=[]

        while t[-1]!=m:

            f\_n=tan(t[-1],points[0])

            t.append(points[0])

            for point in points[1:]:

                f=tan(t[-2],point)

                if f<f\_n:

                    t[-1]=point

                    f\_n=f

            points.remove(t[-1])

            p\_points.append(Edge([t[-2],t[-1]]))

        #print([point.get() for point in p\_points])

        points.append(p)

        while t[-1]!=p:

            f\_n=tan([-t[-1][0],-t[-1][1]],[-points[0][0],-points[0][1]])

            t.append(points[0])

            for point in points[1:]:

                f=tan([-t[-2][0],-t[-2][1]],[-point[0],-point[1]])

                if f<f\_n:

                    t[-1]=point

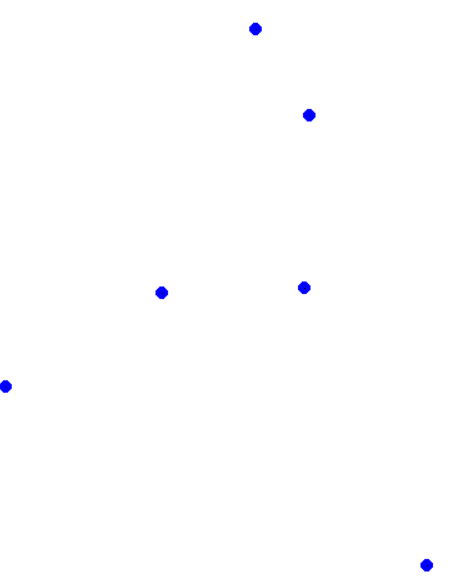
                    f\_n=f

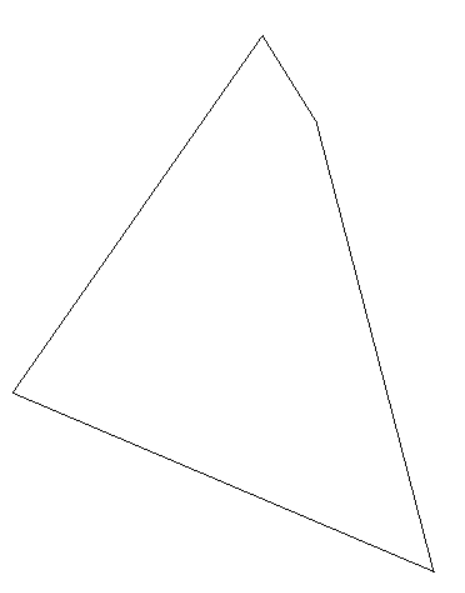
            points.remove(t[-1])

            p\_points.append(Edge([t[-2],t[-1]]))

        #print([point.get() for point in p\_points])

        return Polygon(p\_points)





Вывод

В ходе данной лабораторной работы были получены знания об алгоритмах объёмных моделей, данные знания были применены на практике и реализованы в среде Python.