МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ИНДУСТРИАЛЬНЫЙ УНИВЕРСИТЕТ»
(ФГБОУ ВПО «МГИУ»)
КАФЕДРА ИНФОРМАЦИОННЫХ СИСТЕМ И ТЕХНОЛОГИЙ

КУРСОВАЯ РАБОТА

по дисциплине «Методы хранения и обработки информации»
на тему «Вычисление суммы углов, под которыми рёбра выпуклой оболочки пересекают заданный отрезок. Вычисление суммы площадей проекций граней, центр и все вершины которых находятся от плоскости $x = 2$ на расстоянии строго меньше 1»

Группа	2362
Студент	А.О. Дубинин
Руководитель работы к.ф.-м.н., доцент	Е.А. Роганов

Москва 2014

Аннотация

Работа посвящена модификации проектов «Выпуклая оболочка» и «Изображение проекции полиэдра». В проекте «Выпуклая оболочка» необходимо вычислить сумму углов, под которыми рёбра выпуклой оболочки пересекают заданный отрезок В проекте «Изображение проекции полиэдра» необходимо вычислить сумму площадей проекций граней, центр и все вершины которых находятся от плоскости $x = 2$ на расстоянии строго меньше 1.

Содержание

1. Введение

Проект «Выпуклая оболочка»[?] решает задачу индуктивного перевычисления выпуклой оболочки последовательно поступающих точек плоскости и таких её характеристик, как периметр и площади выпуклой оболочки. Целью данной работы является вычисление суммы углов, под которыми рёбра выпуклой оболочки пересекают заданный отрезок. Решение этой задачи требует знания теории индуктивных функций [?], основ аналитической геометрии, векторной алгебры и языка Ruby [?].

Проект «Изображение проекции полиэдра» [?] — пример классической задачи, для успешного решения которой необходимо знакомство с основами вычислительной геометрии. Задачей, решаемой в данной работе, является модификация эталонного проекта с целью Вычисление суммы площадей проекций граней, центр и все вершины которых находятся от плоскости $x = 2$ на расстоянии строго меньше 1. Для этого необходимы хорошее понимание ряда разделов аналитической геометрии и векторной алгебры, основ объектно-ориентированного программирования и языка Ruby.

Общее количество строк в рассмотренных проектах составляет около 630, из которых более 80 были изменены или добавлены автором в процессе работы над задачами модификации.

2. Модификация проекта «Выпуклая оболочка»

Постановка задачи

Модифицируйте эталонный проект таким образом, чтобы вычислялась сумма углов, под которыми рёбра выпуклой оболочки пересекают заданный отрезок.

После запуска программа запрашивает координаты вершин отрезка, для которого мы должны посчитать сумму углов его пересечений с рёбрами выпуклой оболочки. Далее в программу поступает последовательность координат вершин выпуклой оболочки, как это происходит в эталонном проекте. В процессе поступления новых вершин программа индуктивно подсчитывает и выводит нужную сумму.

Для последовательности точек $A(0, 0)$, $B(0, 1)$, $C(-0.5, 0.5)$ и $D(0.5, 0.5)$, где A и B — вершины задаваемого отрезка, а C и D — вершины выпуклой оболочки, программа выводит 90 в качестве угла, под которым отрезок AB пересекает выпуклую оболочку с вершинами B и C. При добавлении точки точки $E(0, 0)$ в выпуклую оболочку программа выводит 180. Этот результат получается при сложении значения суммы перед добавлением точки E и суммы углов пересечения AB с двумя новыми ребрами выпуклой оболочки: DE и CE. Стоит отметить, что из двух углов, образуемых пересечением отрезков, мы всегда выбираем наименьший.

Все методы, нужные для решения задачи, будут определены в файле convex.rb. Вся геометрическая часть проекта решается в классе `Segment`.

Решение

Для выполнения задания необходимо вычислять сумму углов, под которыми заданный отрезок пересекает рёбра выпуклой оболочки. Это означает, что для каждого ребра выпуклой оболочки мы должны вычислить угол его пересечения с заданным отрезком, зная координаты вершин ребра и отрезка. Вычисление угла, образованного

пересечением двух отрезков, является простой задачей, однако нам надо учитывать случай, когда отрезки не пересекаются вообще. Чтобы выяснить, пересекаются ли два отрезка, решим общую задачу: найдем точку пересечения прямых, на которых лежат наши отрезки.

Для этого реализуем метод `cross?`, который будет проверять, могут ли пересекаться прямые, на которых лежат данные отрезки. Первоначально проверяется, вертикальны ли эти отрезки. Проверка выполняется по соответствию координат x. Если они равны у заданного отрезка (условие `self.point1.x == self.point2.x`), то заданный отрезок вертикальный. Вместе с ним проверяется ребро выпуклой оболочки (условие `self.p.x == self.q.x`). Если они оба вертикальны, метод сразу возвращает `false`. Если они не вертикальны, нужно проверить, не паралелльны ли они. Проверяем угловые коэффициенты отрезков. Угловой коэффициент отрезка $[a_1;\ a_2]$. считается по формуле

$$\frac{a_{2\mathrm{y}} - a_{1\mathrm{y}}}{a_{2\mathrm{x}} - a_{1\mathrm{x}}}$$

Если они совпадают, то отрезки параллельны и не пересекаются, следовательно, метод `cross?` вернет `false`. Если же отрезки не вертикальны и не параллельны, то прямые, на которых лежат эти отрезки, пересекаются и метод вернет `true`:

```
class Segment < Figure
  ...
  def cross?()
    if self.point1.x == self.point2.x and self.p.x == self.q.x
      false
    elsif ((self.point2.y - self.point1.y)/(self.point2.x - self.point1.x) ==
(self.q.y - self.p.y)/(self.q.x - self.p.x))
     false
    else
      true
    end
  end
  ...
```

Если две прямые могут пересечься, то нужно посчитать точку их пересечения. Для этого реализуем метод `cross_point`, возвращающий объект класса `R2Point` — точку, которая является точкой пересечения прямых, которые образованы данными отрезками. В данном методе вычисляются угловые коэффициенты прямых k и коэффициенты свободных членов b из уравнения прямой $y = kx + b$. Координаты x находятся как $\frac{b_2 - b_1}{k_1 - k_2}$, в то время как y находится подставлением k и b в уравнение прямой.

```
class Segment < Figure
  ...
  def cross_point()
    if self.point1.x != self.point2.x
      if self.p.x != self.q.x
        b1 = self.point1.y - (k1 = (self.point2.y - self.point1.y)/(self.point2.x -
self.point1.x))*self.point1.x
        b2 = self.p.y - (k2 = (self.q.y - self.p.y)/(self.q.x - self.p.x))*self.p.x
        xc = (b2 - b1)/(k1 - k2)
        return R2Point.new(xc, xc*k1 + b1)
      else
        b1 = self.point1.y - (k1 = (self.point2.y - self.point1.y)/(self.point2.x -
self.point1.x))*self.point1.x
        yc = k1*self.p.x + b1
        return R2Point.new(self.p.x, yc)
      end
    elsif self.p.x != self.q.x
        k2 = (self.q.y-self.p.y)/(self.q.x-self.p.x)
  b2 = self.p.y - k2*self.p.x
        return R2Point.new(self.point1.x, k2*self.point1.x + b2)
      end
    end
  end
  ...
```

После того, как мы нашли точку пересечения прямых, проверим, что найденная точка является точкой пересечения ребра выпуклой оболочки и заданного отрезка. Реализуем метод `is_on_segments?`. Если известно, что точка лежит на той же прямой, что и отрезок, то точка лежит на этом отрезке, если она находится внутри прямоугольника, в котором данный отрезок — диагональ.

```
class Segment < Figure
  ...
  def is_on_segments?()
    if cross?
      ((c = cross_point).inside?(self.point1, self.point2) and
c.inside?(self.p, self.q))
    else
      false
    end
  end
end
```

После того, как мы нашли точку пересечения отрезков и убедились, что она лежит на отрезках, мы считаем угол пересечения. Этим занимается реализованный метод `angle`. Угол выражается через определение скалярного произведения.Для векторов $\overrightarrow{a}$ и $\overrightarrow{b}$ угол считается по формуле:

$$\alpha = \arccos \frac{\langle \mathbf{a}, \mathbf{b} \rangle}{\sqrt{\langle \mathbf{a}, \mathbf{a} \rangle \langle \mathbf{b}, \mathbf{b} \rangle}}.$$

```
class Segment < Figure
  ...
  def angle()
    if (cross? and is_on_segments?)
      angle = (Math.acos(((m1 = self.point2.x - self.point1.x)*(m2 = self.q.x -
self.p.x) + (n1 = self.point2.y - self.point1.y)*(n2 = self.q.y -
self.p.y))/(Math.sqrt(m1*m1 + n1*n1)*Math.sqrt(m2*m2 +
n2*n2)))*180.0/Math::PI).round(4)
    else
      angle = 0.0
    end
    angle > 180.0 - angle ? 180 - angle : angle
  end
  ...
end
```

На этом геометрическая часть решения заканчивается. Применяем методы для вычисления угла в оболочке. В классе `Figure` добавим метод `angle` и изменим инициализацию:

```
Class Figure
  ...
  def angle;     0.0 end
  def initialize(point1, point2)
    @point1 = point1; @point2 = point2
  end
```

где `@point1` и `@point2` — точки-координаты заданного отрезка. Так же изменятся инициализации во всех остальных классах:

```
class Void < Figure
  def add(p)
    Point.new(p, @point1, @point2)
  end
...
class Point < Figure
  def initialize(p, point1, point2)
    super(point1, point2)
    @p = p
  end
  ...
end
...
class Segment < Figure
  attr_reader :p, :q, :point1, :point2

  def initialize(p, q, point1 = @point1, point2 = @point2)
    super(point1, point2)
    @p, @q = p, q
  end
...
```

```
end
class Polygon < Figure
  attr_reader :points, :perimeter, :area, :angle

  def initialize(a, b, c, point1, point2)
    super(point1,point2)
    ...
    @angle     = Segment.new(a, b, @point1, @point2).angle +
 Segment.new(b, c, @point1, @point2).angle +
 Segment.new(c, a, @point1, @point2).angle

end
```

В классе `Polygon` при инициализации создается оболочка из трех вершин, при создании высчитывается сумма углов для трех вершин. Затем индуктивно высчитывается сумма углов для новых ребер:

```
class Polygon < Figure
  ...
  def add(t)
    ...
    if t.light?(@points.last, @points.first)
      ...
      @angle     -= Segment.new(@points.first, @points.last, @point1,
@point2).angle
      ...
      p = @points.pop_first
      while t.light?(p, @points.first)
        ...
        @angle     -= Segment.new(@points.last, p, @point1, @point2).angle
        ...
      end
      ...
      p = @points.pop_last
      while t.light?(@points.last, p)
        ...
        @angle     -= Segment.new(@points.last, p, @point1, @point2).angle
        ...
      end
      @angle     += Segment.new(@points.last, t, @point1, @point2).angle +
Segment.new(@points.first, t, @point1, @point2).angle
    ...
    end
    self
  end
end
```

На этом модификация файла `convex/convex.rb` завершена. Теперь необходимо добавить графическую часть. Модифицируем в файле `convex/tk_drawer.rb` метод `draw` класса `Figure`, чтобы рисовался заданный нами отрезок:

```
class Figure
  def draw
    TkDrawer.clean
    TkDrawer.draw_line(@point1,@point2)
  end
end
```

Для вывода результата на экран изменяем в файлах `convex/run_tkconvex.rb` и `convex/run_convex.rb` строку вывода.

```
fig = Void.new
  ...
    puts "S = #{fig.area}, P = #{fig.perimeter}, Angle = #{fig.angle}"
```

Модификация эталонного проекта «Выпуклая оболочка» завершена. Пример работы программы можно увидеть на (рис.1).

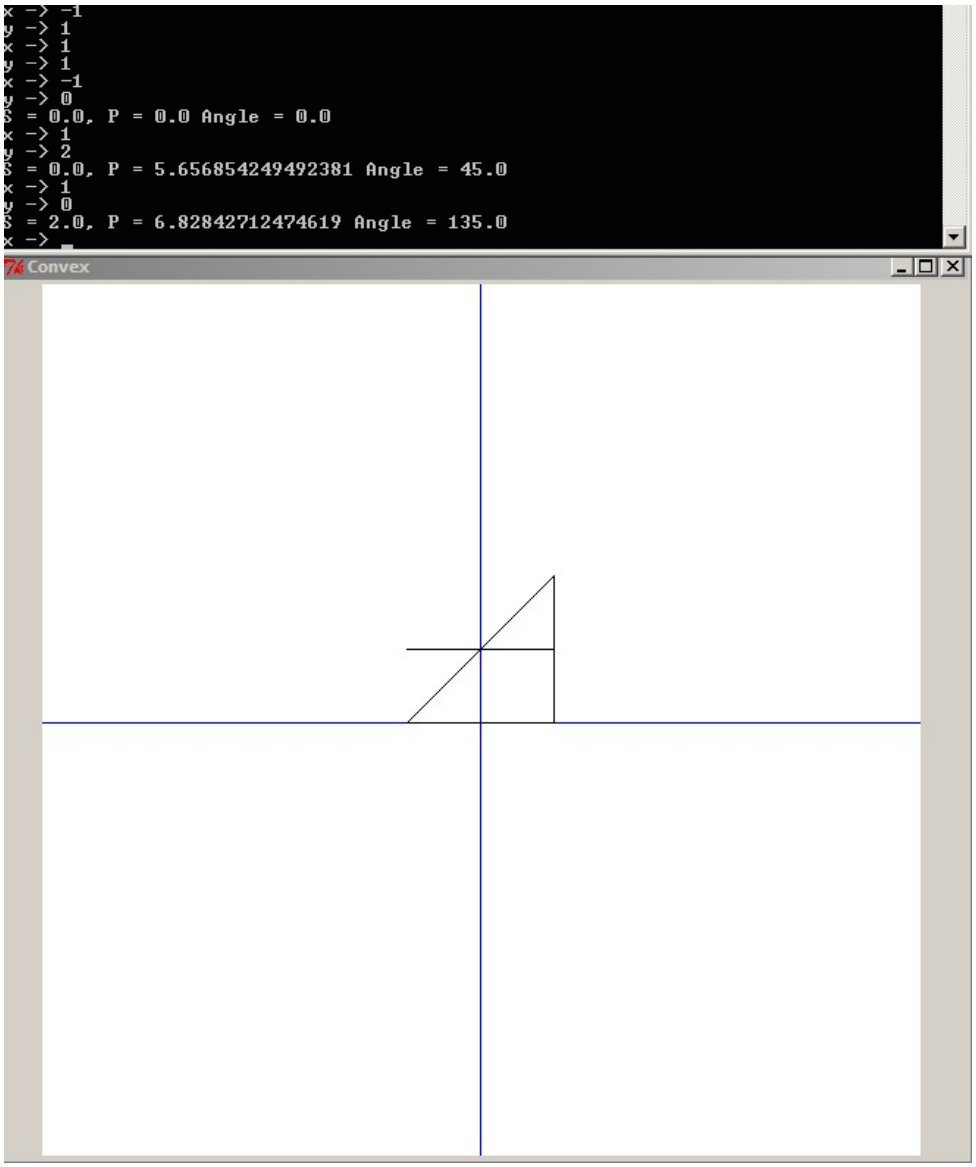

Рис. 1. Работа программы «Выпуклая оболочка»

3. Модификация проекта «Изображение проекции полиэдра»

Точная постановка задачи

Назовём точку в пространстве «хорошей», если она находится на расстоянии строго меньше 1 от плоскости $x = 2$ Модифицируйте эталонный проект таким образом, чтобы определялась и печаталась следующая характеристика полиэдра: сумма площадей проекций граней, центр и все вершины которых — «хорошие» точки.

Решение данной задачи

Для выполнения поставленной задачи необходимо модифицировать код в файле `common/polyedr.rb` Методы проверки центра грани, всех ее вершин и вычисление площади проекции грани выполняется в методе `draw` класса `Polyedr` в файле `shadow/polyedr.rb`:

```
class Polyedr
  attr_reader :edges, :facets, :total_area
  def initialize(file)
    @total_area = 0.0
    ...
    nf.times do
      ...
     @total_area += facet.area if facet.all_good?()
      end
    puts "Нужная сумма: #{@total_area}"
    end
  end
end
```

Модификация кода

В файле `common/polyedr.rb` добавим метод `is_good?`, который позволит проверять точку на расстояние от плоскости $x = 2$. Расстояние от точки $M(M_x,\ M_y,\ M_z)$ для плоскости, заданной уравнением $Ax + By + Cz + D = 0$, от точки вычисляется по формуле:

$$d = \frac{|AM_x + BM_y + CM_z + D|}{\sqrt{A^2 + B^2 + C^2}}.$$

Так как в условии задачи плоскость задана уравнением $x = 2$, то уравнение принимает вид

$$d = \frac{|M_x + 0 + 0 - 2|}{\sqrt{1 + 0 + 0}} = |M_x - 2|.$$

Если расстояние больше, либо равно единице, то метод вернет `false`, иначе, метод вернет `true` :

```
class R3
  ...
  def is_good?()
    (x - 2).abs < 1
  end
end
```

Затем добавим методы в класс `Facet`. Метод `all_good?` возвращает `true`, когда все вершины грани и ее центр — хорошие точки, и `false`, когда не все. Координаты центра грани находятся с помощью метода `center`, который используется в файле `shadow/polyedr.rb` эталонного проекта.

```
class Facet
  ...
  def all_good?()
    @vertexes.all?{|i| i.is_good?()} and center.is_good?()
  end
  ...
end
```

Далее напишем вспомогательный метод `triangle_area(a, b, c)`, который считает площадь треугольника, заданного вершинами `a, b` и `c`. В данном методе используется формула вычисления площади треугольника по вершинам при помощи определителя матрицы:

$$S = \frac{1}{2}\begin{vmatrix} a_x - c_x & a_y - c_y \\ b_x - c_x & b_y - c_y \end{vmatrix}.$$

```
class Facet
  ...
  def triangle_area(a, b, c)
    0.5*((a.x-c.x)*(b.y-c.y)-(a.y-c.y)*(b.x-c.x)).abs
  end
end
```

Затем реализуем метод `area`, который будет основным для подсчета площади проекции грани. Данный метод будет базироваться на вспомогательном методе `triangle_area(a, b, c)`, где в качестве агрумента `c` будет передаваться центр грани. Циклом пройдем по всем вершинам и просуммируем площади всех треугольников, из которых состоит грань, тем самым получим площадь проекции грани.

```
class Facet
  ...
  def area
    area = 0.0; c = center
    (-1...@vertexes.size - 1).each do |i|
      area += triangle_area(@vertexes[i], @vertexes[i + 1], c)
    end
    area
  end
  ...
end
```

На этом решение поставленной завершено. Пример работы программы с модифицированным файлом `test3.geom` можно увидеть на рис.2 и его содержание представленно ниже.

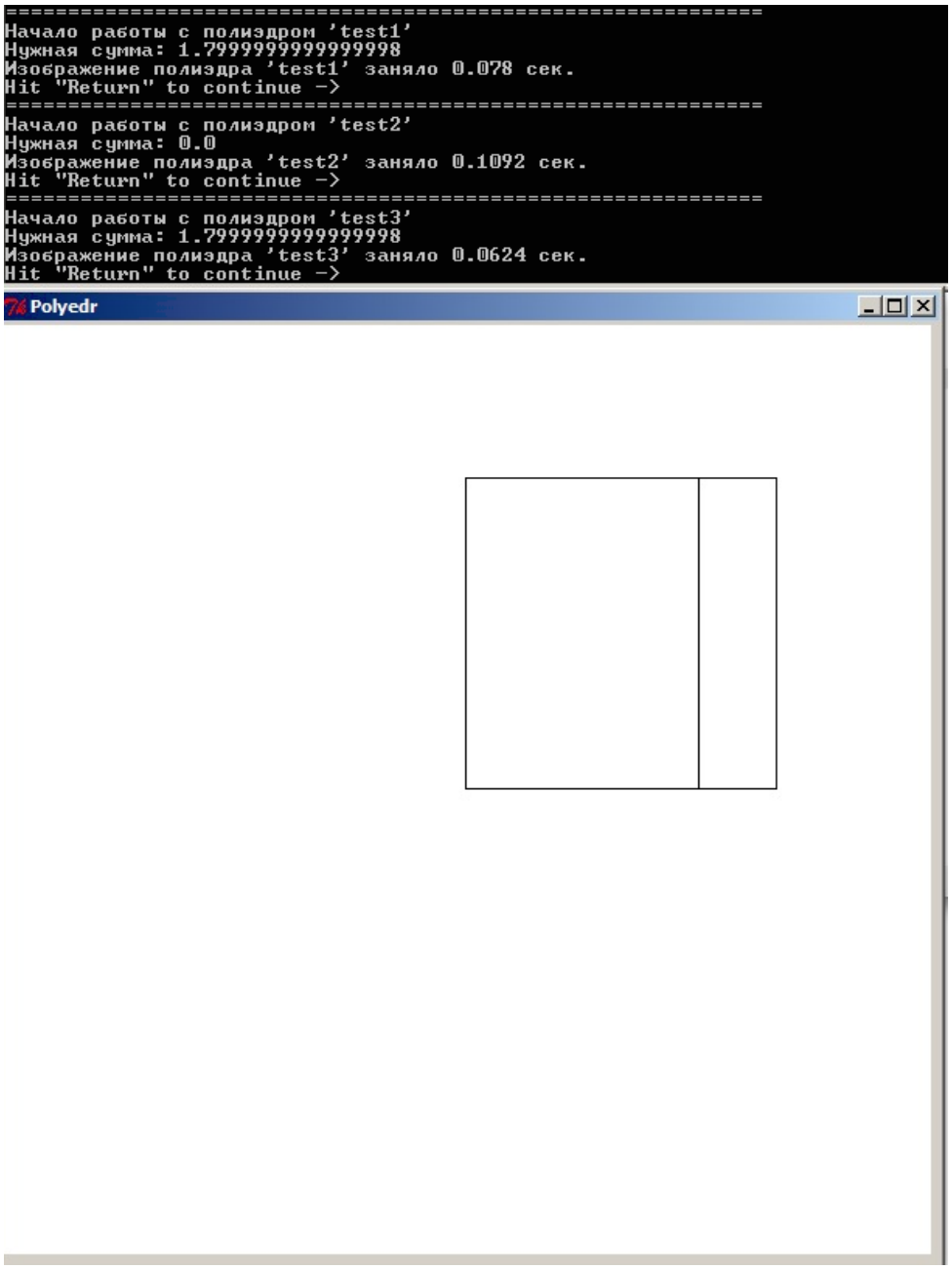

Рис. 2. Работа программы «Изображение проекции полиэдра»

```
1 0 0 0
10 3 12
2.0 0 0
2.0 2 0
1.5 0 2
1.5 2 2
1.1 0 0
1.1 2 0
0 0 0
```

```
0 2 0
1.5 0 2
1.5 2 2
4 1 2 4 3
4 5 6 4 3
4 7 8 4 3
```

Список литературы и интернет-ресурсов

[1] http://edu.msiu.ru//files/25029-lecture.html — Описание проекта «Выпуклая оболочка».

[2] Е.А. Роганов *Основы информатики и программирования.* — М., МГИУ, 2002.

[3] http://ru.wikipedia.org/wiki/Ruby — Википедия (свободная энциклопедия) о языке Ruby.

[4] http://edu.msiu.ru/files/26490-lecture.html, http://edu.msiu.ru/files/26929-lecture.html — Описание проекта «Изображение проекции полиэдра».

[5] http://www-sbras.nsc.ru/win/docs/TeX/LaTex2e/Text_in_LaTeX.pdf — Справочник по командам LATEX.

[6] http://ru.wikipedia.org/wiki/Скалярное_произведение — Статья о скалярном произведении

[7] http://ru.onlinemschool.com/math/library/analytic_geometry/p_plane/ — Расстояние от точки до плоскости