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 заданную прямую. Вычисление суммы площадей граней, все вершины которые расположены вне сферы $x^2 + y^2 + z^2 = 1$»

Группа 2362

Студент А.Ю. Дьячук

Руководитель работы
к.ф.-м.н., доцент Е.А. Роганов

Москва 2012

Аннотация

Курсовая работа посвящена дополнению проектов «Выпуклая оболочка» и «Изображение проекции полиэдра». В первом из этих проектов решалась задача: вычисление суммы углов, под которыми рёбра выпуклой оболочки пересекают заданную прямую. Во втором проекте решалась задача: вычисление суммы площадей граней, все вершины которые расположены вне сферы $x^2 + y^2 + z^2 = 1$.

Оглавление

1. Введение

В данной курсовой работе рассматриваются модификации двух эталонных проектов «Выпуклая оболочка» и «Изображение проекции полиэдра», реализованных на объектно-ориентированном языке программирования высокого уровня Ruby.

Целями работы являются:

- в проект «Выпуклая оболочка» добавить вычисление суммы внутренних углов выпуклой оболочки;
- «научить» программу определять и выводить сумму углов, под которыми рёбра выпуклой оболочки пересекают заданную прямую в проекте «Выпуклая оболочка»;
- добавить в проект «Изображение проекции полиэдра» вывод суммы площадей граней, все вершины которые расположены вне сферы $x^2 + y^2 + z^2 = 1$.

Для того чтобы выполнить полученные задания, необходимо было изучить особенности языка Ruby, подробно разобрать каждый эталонный проект и применить полученные знания в области информатики, компьютерной математики и аналитической геометрии на плоскости и в пространстве.

2. Модификация проекта «Выпуклая оболочка»

Постановка задачи

Модифицируйте эталонный проект таким образом, чтобы вычислять сумму углов, под которыми рёбра выпуклой оболочки пересекаю заданную прямую.

Решение

После запуска программа предлагает пользователю ввести последовательно координаты вершин выпуклой оболочки. Введённая точка индуктивно добавляется в выпуклую оболочку. Нам же необходимо вместе со значениями периметра и площади выпуклой оболочки выводить сумму острых внутренних углов. Это задание не требует серьёзной модификации исходного кода, все функции будут вписываться в файл проекта `convex.rb`.

Для выполнения задания необходимо вычислять внутренние углы, используя координаты вершин выпуклой оболочки. В языке Ruby имеется особый механизм множественного наследования `(mixin)` `Math`, которая содержит методы вычисления простейших тригонометрических функций, в частности, в данном проекте используется `Math.acos`.

Из аналитический геометрии известно, что угол между векторами $\overline{BA}$ и $\overline{BC}$ можно найти по формуле

$$\cos(B) = \frac{\overline{BA} \cdot \overline{BC}}{|\overline{BA}| \cdot |\overline{BC}|}.$$

Для точек $A(X_1; Y_1)$ и $B(X_2; Y_2)$ координаты вектора

$$AB = \{X_2 - X_1; Y_2 - Y_1\}.$$

Если $\overline{BA}\{X_1; Y_1\}$ и $\overline{BC}\{X_2; Y_2\}$, то скалярное произведение этих векторов

$$\overline{BA} \cdot \overline{BC} = X_1X_2 + Y_1Y_2,$$

а длина вектора

$$\sqrt{\overline{BA}^2} = |\overline{BA}| = \sqrt{X_1^2 + Y_1^2}.$$

Окончательная формула для угла B между векторами $\overline{BA}$ и $\overline{BC}$ выглядит так:

$$\cos(B) = \frac{X_1 X_2 + Y_1 Y_2}{\sqrt{X_1^2 + Y_1^2} \cdot \sqrt{X_2^2 + Y_2^2}}.$$

Метод `Math.acos` возвращает угол в радианах. Так как решено выводить ответ в градусах, воспользуемся формулой $\alpha^\circ = \alpha \times \frac{180}{\pi}$.

Модификация кода

Метод, который находит уравнение прямой, заданной двумя точками:

```
def equation_from_segment(p1, p2)

  # Ax + By + C = 0
  x1, y1 = p1.x, p1.y
  x2, y2 = p2.x, p2.y

  a = y1 - y2
  b = x2 - x1
  c = x1 * y2 - x2 * y1
  [a, b, c]
end
```

Метод, который находит точку пересечения линия AB и отрезка PQ:

```
def cross_point(a, b, p, q)
  a1, b1, c1 = equation_from_segment(a, b)
  a2, b2, c2 = equation_from_segment(p, q)
  d = (a1*b2-a2*b1)

  # Если прямые не перескаются - завершаем работу:
  # отрезки тоже не будут пересекаться:
  return false if d == 0

  # Находим координаты пересечения прямых:
  x = -(c1*b2-c2*b1).to_f / d
  y = -(a1*c2-a2*c1).to_f / d

  # Точка пересечения прямых должна принадлежать отрезку PQ:
  m = R2Point.new(x, y)
  return false unless m.inside?(p, q)
  m
end
```

Метод нахождения угла B:

```
def angle (a, b, c)

  # Координаты векторов:
  ba_x = a.x - b.x
  ba_y = a.y - b.y
  bc_x = c.x - b.x
  bc_y = c.y - b.y

  dot_product = ba_x*bc_x + ba_y*bc_y
  module_ba = ba_x**2 + ba_y**2
  module_bc = bc_x**2 + bc_y**2
  cos = dot_product / Math.sqrt(module_ba * module_bc)
  cos *= -1 if cos < 0
  return Math.acos(cos)
end
```

Увидеть работу этой модификации можно на рис. 1.

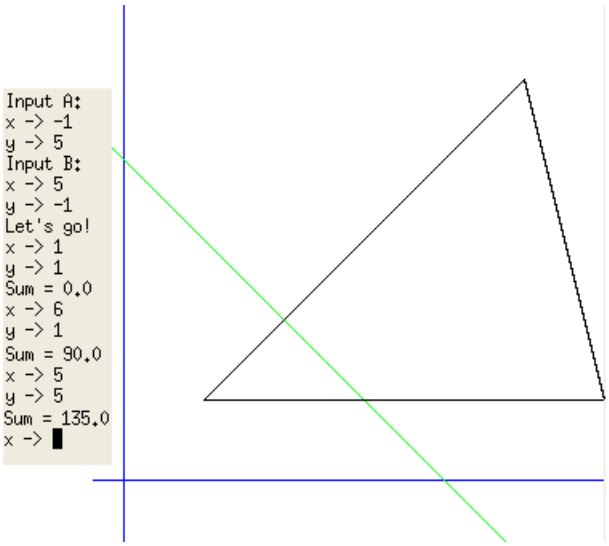

Рис. 1. Вычисление суммы углов, под которыми рёбра выпуклой оболочки пересекают заданную прямую

3. Модификация проекта «Изображение проекции полиэдра»

Постановка задачи

Модифицируйте эталонный проект таким образом, чтобы определялась и печаталась следующая характеристика полиэдра: сумма площадей граней, все вершины которые расположены вне сферы.

После запуска программы, программа вычисляет площадь полигонов, которые расположены вне сферы. Это задание не требует серьёзной модификации исходного кода, все функции будут вписываться в файл проекта `polyedr.rb`.

Решение

Для выполнения задания необходимо определить хорошую точку в пространстве,если она строго находится вне сферы $x^2+y^2+z^2 = 1$. Вычислить сумму площадей граней, все вершины которые расположены вне сферы $x^2 + y^2 + z^2 = 1$.

Модификация кода

Создадим метод модуля.

```
def length
   Math.sqrt(@x*@x + @y*@y + @z*@z)
end
```

Метод, который вычисляет вершины, который лежат вне сферы $x^2 + y^2 + z^2 = 1$:

```
def good?
   @x*@x + @y*@y + @z*@z > 1#
end
```

Метод, который вычисляет площадь грани:

```
def area
   result = 0.0

   for i in 1 ... (vertexes.size - 1)
     result += triangle_area(vertexes[0], vertexes[i], vertexes[i + 1])
   end

   return 0.5*result
end
```

Метод, который вычисляет сумму площадей треугольников:

```
def triangle_area(a, b, c)
   ((b - a).v(c - a)).length
end
```

Метод который вычисляет площадь грани, если она расположена вне сферы:

```
def good_area
   result = 0.0

   facets.each do |face|
     result += face.area if face.good?
   end

   return result
end
```

Результат работы модифицированной программы на рис. 2.

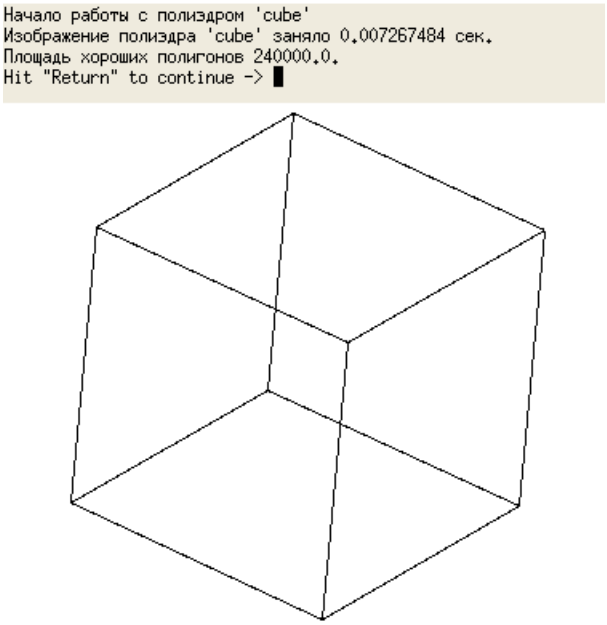

Рис. 2. Начало работы с полиэдром куб

Список литературы и интернет-ресурсов

[1] `http://ru.wikipedia.org/wiki/LaTeX` — Википедия (свободная энциклопедия) о системе LaTeX.

[2] `http://www.mgena.chat.ru/latex/indru.html` — Система TeX/LaTeX(конспект).

[3] `http://edu.msiu.ru/files/571-convex.zip` — Исходные файлы эталонного проекта «Выпуклая оболочка». Доступны только при авторизации на образовательном портале МГИУ.

[4] `http://edu.msiu.ru/files/1821-polyedr.zip` — Исходные файлы эталонного проекта «Изображение проекции полиэдра». Доступны только при авторизации на образовательном портале МГИУ.

[5] М.Я. Выгодский. *Справочник по высшей математике.* — М., АСТ: Астрель, 2008.

4. Приложение

Вычисляется сумма углов, под которыми рёбра выпуклой оболочки пересекают заданную прямую.

```
# encoding: UTF-8
require "./r2point"
require "./deq"
# Вычисляется сумма углов, под которыми рёбра
# выпуклой оболочки пересекают заданную прямую.

# Абстрактная фигура
class Figure
  def perimeter; 0.0 end
  def area;      0.0 end
  def sum;    get_sum end

  #создаем метод суммы
  def get_sum
    #если текущий метод не отвечает (points),
    #то мы возвращам 0(нет ребер)
    return 0.0 unless self.respond_to? :points
    #возвращает точки выпуклой оболочки
    pts = self.points
    p pts
    #сумма углов 0
    s = 0.0
    #н раз проходим по всем ребрам нашей выпуклой оболочки
    pts.size.times do |i|
      #находим вершины ребра
      p1,p2 = pts[i-1], pts[i]
      # находим точку пересечения ребра и прямой
      m = cross_point $points[0], $points[1], p1, p2
      #если точка пересечения существует (ребра и прямой)
      if m
        #выбераем точку на ребре не совпадающей с точкой пересечения
        cp = (m == p1) ? p2 : p1
        #выбераем точку на прямой не совпадающей с точкой пересечения
        cl = (m == $points[0]) ? $points[1] : $points[0]
        #вычисляем угл между ребром и премой и добавляем его в сумму
        s += angle(cp, m, cl).abs unless cp == cl
      end
    end
    s#возвраем сумму всех углов (пробежалисъ в цикле по всем ребрам)
  end
  ##############################################
  # находит уравнение прямой, заданной двумя точками
  def equation_from_segment(p1, p2)
    # Ax + By + C = 0
    x1, y1 = p1.x, p1.y
```

```
      x2, y2 = p2.x, p2.y

      a = y1 - y2
      b = x2 - x1
      c = x1 * y2 - x2 * y1
      [a, b, c]
    end
    ##########################################################3
    # точка пересечения линии AB и отрезка PQ
    def cross_point(a, b, p, q)
      a1, b1, c1 = equation_from_segment(a, b)
      a2, b2, c2 = equation_from_segment(p, q)

      d = (a1*b2-a2*b1)
      # Если прямые не перескаются - завершаем работу,
      # отрезки тоже не будут пересекаться
      return false if d == 0

      # Находим координаты пересечения прямых
      x = -(c1*b2-c2*b1).to_f / d
      y = -(a1*c2-a2*c1).to_f / d

      # Точка пересечения прямых должна принадлежать отрезку PQ
      m = R2Point.new(x, y)
      return false unless m.inside?(p, q)
      m
    end
    ##########################################################
    # ищем угол B
    def angle (a, b, c)
      # координаты векторов
      ba_x = a.x - b.x
      ba_y = a.y - b.y
      bc_x = c.x - b.x
      bc_y = c.y - b.y

      dot_product = ba_x*bc_x + ba_y*bc_y # скалярное произведение
      module_ba = ba_x**2 + ba_y**2
      module_bc = bc_x**2 + bc_y**2
      cos = dot_product / Math.sqrt(module_ba * module_bc)
      cos *= -1 if cos < 0
      return Math.acos(cos)
    end

end

# "Нульугольник"
class Void < Figure
  def add(p)
    Point.new(p)
```

```
  end
end

# "Одноугольник"
class Point < Figure
  def initialize(p)
    @p = p
  end
  def add(q)
    @p == q ? self : Segment.new(@p, q)
  end
end

# "Двуугольник"
class Segment < Figure
  def initialize(p, q)
    @p, @q = p, q
  end
  def perimeter
    2.0 * @p.dist(@q)
  end
  def add(r)
    return Polygon.new(@p, @q, r) if R2Point.triangle?(@p, @q, r)
    return Segment.new(@p, r)     if @q.inside?(@p, r)
    return Segment.new(r, @q)     if @p.inside?(r, @q)
    self
  end
  def points#функция возвращает
    [@p, @q]#две точки ввиде массива
  end
end

# "Многоугольник"
class Polygon < Figure
  attr_reader :perimeter, :area


  def initialize(a, b, c)
    @points     = Deq.new
    @points.push_first(b)
    if b.light?(a,c)
      @points.push_first(a)
      @points.push_last(c)
    else
      @points.push_last(a)
      @points.push_first(c)
    end
    @perimeter = a.dist(b) + b.dist(c) + c.dist(a)
    @area      = R2Point.area(a, b, c).abs
    @sum = get_sum# накапливаем сумму в методе суммы
```

```
end

# добавление новой точки
def add(t)

  # поиск освещённого ребра
  @points.size.times do
    break if t.light?(@points.last, @points.first)
    @points.push_last(@points.pop_first)
  end

  # хотя бы одно освещённое ребро есть
  if t.light?(@points.last, @points.first)

    # учёт удаления ребра, соединяющего конец и начало дека
    @perimeter -= @points.first.dist(@points.last)
    @area      += R2Point.area(t, @points.last, @points.first).abs

    # удаление освещённых рёбер из начала дека
    p = @points.pop_first
    while t.light?(p, @points.first)
      @perimeter -= p.dist(@points.first)
      @area      += R2Point.area(t, p, @points.first).abs
      p = @points.pop_first
    end
    @points.push_first(p)

    # удаление освещённых рёбер из конца дека
    p = @points.pop_last
    while t.light?(@points.last, p)
      @perimeter -= p.dist(@points.last)
      @area      += R2Point.area(t, p, @points.last).abs
      p = @points.pop_last
    end
    @points.push_last(p)

    # добавление двух новых рёбер
    @perimeter += t.dist(@points.first) + t.dist(@points.last)
    @points.push_first(t)
  end

  self
end

def points
  pts = []

  @points.size.times do
    p = @points.pop_first
    pts << p
```

```
        @points.push_last(p)
      end

      return pts
    end
  end
```

Вычисляется сумма площадей граней, все вершины которые расположены вне сферы.

```
# encoding: UTF-8
require "../common/polyedr"

class R3
  def length#
    Math.sqrt(@x*@x + @y*@y + @z*@z)#
  end

  def good?#вершина хорошая, если она лежит вне сферы
    @x*@x + @y*@y + @z*@z > 1#
  end
end

class Facet
  def area#метод, вычисляющий площадь грани
    result = 0.0#

    for i in 1 ... (vertexes.size - 1)#цикл от первой вершины до последней
      result += triangle_area(vertexes[0], vertexes[i], vertexes[i + 1])
    end

    return 0.5*result#деление результата на 2
  end

  def good?#метод вершин хороших
    vertexes.each do |vertex|#если вершины хорошие, то
      return false if !vertex.good?#вернем false если вершина не хорошая
    end

    return true#вернем правду
  end

  private

  def triangle_area(a, b, c)#площать треугольников рассматривать
    ((b - a).v(c - a)).length#модуль векторов length-модуль
  end
end

class Polyedr
  def draw
```

```
    TkDrawer.clean
    edges.each{|e| TkDrawer.draw_line(e.beg, e.end)}
  end

  def good_area#метод площади возвращающий метод хороших граней
    result = 0.0#начало суммы 0

    facets.each do |face|#цикл по всем граням
    #если грань хорошая то добавляем площадь грани в нашу сумму
      result += face.area if face.good?
    end

    return result#возвращаем сумму
  end
end
```