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 количества вершин выпуклой оболочки, лежащих в 1-окрестности заданного заполненного треугольника. Сумма длин проекций невидимых частей частично видимых рёбер, образующих с горизонтальной плоскостью угол не более $\pi/7$, центр которых находится строго внутри сферы $x^2 + y^2 + z^2 = 4$ »

Группа 2362

Студент М.У. Бирюков

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 количество вершин выпуклой оболочки, лежащих в 1-окрестности заданного заполненного треугольника. Во проекте «Изображение проекции полиэдра» необходимо вычислить сумму длин проекций невидимых частей частично видимых рёбер, образующих с горизонтальной плоскостью угол не более $\pi/7$, центр которых находится строго внутри сферы $x^2 + y^2 + z^2 = 4$.

Содержание

1. Введение

Проект «Выпуклая оболочка»[1] 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 количества вершин выпуклой оболочки, которые находятся в заданном треугольнике или в единичной окрестности заданного треугольника. Решение этой задачи требует знания теории индуктивных функций [2], основ аналитической геометрии, векторной алгебры и языка Ruby [3].

Проект «Изображение проекции полиэдра» [4] — пример классической задачи, для успешного решения которой необходимо знакомство с основами вычислительной геометрии. Задачей, решаемой в данной работе, является модификация эталонного проекта с целью определения суммы длин проекций невидимых частей частично видимых рёбер, образующих с горизонтальной плоскостью угол не более $\pi/7$, центр которых находится строго внутри сферы $x^2 + y^2 + z^2 = 4$. 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 1180, из которых более 350 были изменены или добавлены автором в процессе работы над задачами модификации.

2. Модификация проекта «Выпуклая оболочка»

Точная постановка задачи

Вычисляется количество вершин выпуклой оболочки, лежащих в 1-окрестности заданного заполненного треугольника. 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 количество точек, попадающие в заданный заполненный треугольник или в его 1-окрестность. Для этого необходимо до начала ввода координат вершин выпуклой оболочки задать треугольник, потребовав на ввод 6 координат треугольника.

Это задание не требует серьёзной модификации исходного кода, все функции будут вписываться в файлы проекта `convex.rb` и `r2point.rb`.

Решение

Эталонный проект до модификации индуктивно строит выпуклую оболочку по точкам. При добавлении новой точки программа индуктивно перевычисляет периметр и площадь оболочки. Наша цель — модифицировать эталонный проект таким образом, чтобы вычислялось количество вершин выпуклой оболочки, лежащих в 1-окрестности заданного заполненного треугольника.

Для решения данной задачи необходимо проверять, принадлежат ли вершины выпуклой оболочки заданному треугольнику или 1-окрестности заданного треугольника. В языке Ruby имеется библиотека `Math`, которая содержит методы вычисления

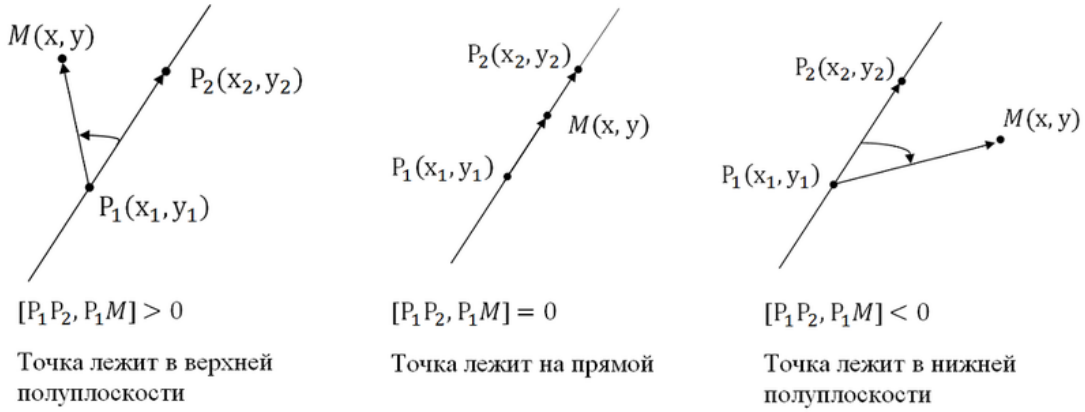
простейших тригонометрических функций, в частности, в данном проекте используется `Math.sqrt`.

После добавления вершины выпуклой оболочки возникает три случая, которые необходимо рассмотреть:

1. вершина находится в заполненном треугольнике;
2. вершина находится в единичной окрестности треугольника;
3. вершина находится вне треугольника и единичной окрестности.

Вершина находится в заполненном треугольнике

Для того, чтобы проверить, лежит ли точка в треугольнике мы будем использовать псевдоскалярное произведение [6]. 2 вершины треугольника образуют отрезок, который можно воспринять как луч. Этот луч делит всю плоскость на 2 полуплоскости. Берутся 2 вершины треугольника и обозреваемая точка, затем относительно них выполняется псевдоскалярное произведение.



Знак числа, которое получается при вычислении говорит нам, в какой из полуплоскостей относительно стороны треугольника лежит точка. Если все знаки чисел, полученных во время вычисления псевдоскалярных произведений относительно всех сторон треугольника, совпадают, значит точка лежит внутри треугольника, иначе, точка находится вне треугольника.

Вершина находится в единичной окрестности треугольника

Для определения принадлежности точки единичной окрестности мы будем считать расстояние от точки до сторон треугольника. Если хотя бы одно из этих расстояний находится в промежутке (0,1), то точка находится в единичной окрестности заданного треугольника. Для определения расстояния от точки надо сначала определить, лежит ли точка левее или правее отрезка. Вычисляется скалярное произведение от искомой точки до каждого из карев отрезка. Если одно из этих скалярных произведений меньше или равно нулю, то возвращается расстояние от точки до края отрезка, скалярное произведение с которым было меньше либо равно нулю, причем считается расстояние как гипотенуза прямоугольного треугольника. Если же скалярные произведения больше нуля, то расстояние считается по формуле, которая указана ниже [7].

$$d\big(P, L\big) = \frac{(y_0 - y_1)x + (x_1 - x_0)y + (x_0 y_1 - x_1 y_0)}{\sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2}}.$$

Модификация кода

Для начала модифицируем файл `convex/r2point.rb` добавив в класс `R2Point` методы, которые выполняют следующие действия:

- нахождение расстояния от точки до отрезка;
- проверка принадлежности точки треугольнику;
- проверка принадлежности точки окрестности и треугольнику.

```
def dist_segm(a,b)
  p1=(@x-a.x)*(b.x-a.x)+(@y-a.y)*(b.y-a.y)
  p2=(a.x-b.x)*(@x-b.x)+(a.y-b.y)*(@y-b.y)
  return Math.sqrt((@x-a.x)**2+(@y-a.y)**2) if p1<=0
  return Math.sqrt((@x-b.x)**2+(@y-b.y)**2) if p2<=0
  a=(a.y-b.y)*@x+(b.x-a.x)*@y+(a.x*b.y-b.x*a.y
  b=Math.sqrt((b.x-a.x)**2+(b.y-a.y)**2)
  return (a/b).abs
end
```

Метод `dist_segm(a,b)` возвращает расстояние от точки, к которой применяется этот метод до отрезка, концами которого являются точки a и b.

```
def inside_triangle?(a,b,c)
  l=(a.x-@x)*(b.y-a.y)-(b.x-a.x)*(a.y-@y)
  m=(b.x-@x)*(c.y-b.y)-(c.x-b.x)*(b.y-@y)
  n=(c.x-@x)*(a.y-c.y)-(a.x-c.x)*(c.y-@y)
  (l>0 && m>0 && n>0) || (l<0 && m<0 && n<0)
end
```

Метод `inside_triangle?(a,b,c)` использует псевдоскалярное произведение для определения, находится ли точка внутри треугольника. Если точка, к которой применяется этот метод находится внутри треугольника, то метод возвращает true, иначе возвращается false

```
def is_inside?(a,b,c)
  return true if self.inside_triangle?(a,b,c) #если в треугольнике
  return true if (self.dist_segm(a,b)<=1 && self.dist_segm(a,b)>=0) ||
  (self.dist_segm(b,c)<=1 && self.dist_segm(b,c)>=0) ||
  (self.dist_segm(c,a)<=1 && self.dist_segm(c,a)>=0) ||
  (self.dist_segm(a,c)<=1 && self.dist_segm(a,c)>=0)
  return false
end
```

Метод `is_inside?(a,b,c)` выполняет финальную проверку на принадлежность точки треугольнику или окрестности треугольника. Первоначально, он проверяет, лежит ли точка внутри треугольника, вызывая метод `inside_triangle?(a,b,c)`. Если этот метод вернул true, то и метод `is_inside?(a,b,c)` возвращает true, говоря о том, что точка удовлетворяет условию. Если же был получен false, то идет проверка окрестности. Проверяется расстояние от точки до каждой из сторон при помощи метода `dist_segm(a,b)`. Если хотя бы до одной стороны расстояние больше нуля и меньше единицы, то возвращается true, опять же говоря о том, что точка удовлетворяет условию. Во всех остальных ситуациях (если точка не лежит ни в треугольнике, ни в окрестности) возвращается false, что означает, что точка не удовлетворяет условию и увеличивать количество точек, удовлетворяющих условию, не стоит.

Теперь модифицируем файл `convex/convex.rb`. Добавим в класс `Figure` инициализацию объекта, метод создания треугольника, метод проверки на принадлежность треугольнику и метод получения количество вершин выпуклой оболочки, которые удовлетворяют нашему условию.

```
def initialize; @ins=0 end
```

Теперь при инициализации абстрактной фигуры создается переменная экземпляра класса `Figure @ins` . Так как в нашем проекте создается только одна выпуклая оболочка, мы можем использовать именно этот тип переменной.

```
def set_triangle(a,b,c)
  @@a, @@b, @@c = a,b,c
end
```

Задание треугольника производится тремя точками, которые будут храниться в переменных класса `Figure @@a, @@b, @@c` соответственно. Так как при добавлении вершин выпуклой оболочки может произойти создание объекта другого класса, нам придется использовать именно переменные класса, ибо все объекты классов `Void, Point, Segment, Polygon` наследуются от класса `Figure`, а следовательно, наследуют и точки нашего треугольника.

```
def intr?(p) ; (p.is_inside?(@@a,@@b,@@c)) ? 1 : 0 ;end
```

Метод `intr?(p)` проверяет принадлежность точки, получаемой в качестве аргумента, заданному ранее треугольнику используя метод `is_inside?(a,b,c)`, который возвращает true, в случае, если точка лежит либо в заданном треугольнике, либо в его единичной окрестности и false во всех остальных случаях. Метод `intr?(p)`, в случае, если точка попадает в заданный треугольник или его окрестность, возвращает число 1, во всех остальных случаях он возвращает 0.

```
def inside_points; @ins; end
```

Метод `inside_points` возвращает переменную экземпляра `@ins`, которая хранит количесто вершин, лежащих в заданном треугольнике или его единичной окрестности.

Теперь рассмотрим класс `Void`. Так как объект именно этого класса создается при начале работы программы, то задавать треугольник лучше всего именно в нем. Для этого добавим инициализацию объекта класса `Void`.

```
def initialize(a=R2Point.new, b=R2Point.new, c=R2Point.new)
  set_triangle(a,b,c)
end
```

Если при создании объекта класса `R2Point` не указать аргументы, то координаты точки будут запрашиваться через ручной ввод, мы этим воспользуемся. Зададим 3 точки `a, b, c` и передадим их в метод `set_triangle(a,b,c)`, чтобы задать эти точки как переменные класса `Figure`, которые в будущем будут доступны из всех классов.

После задания треугольника можно начать работать с точками выпуклой оболочки. В классе `Point` изменим инициализацию, добавив строчку, которая будет проверять, лежит ли она в заданном ранее треугольнике или его единичной окрестности.

```
class Point < Figure
  def initialize(p)
    @p = p
    @ins=intr?(@p)
  end
  ...
end
```

Если точка удовлетворяет условию, то переменная `@ins` будет равна 1, в противном случае, она будет равна 0, так как метод `intr?(p)` возвращает 1, если точка удовлетворяет условию и 0, если не удовлетворяет.

По тому же принципу изменим инициализацию класса `Segment` и класса `Polygon`

```
class Segment < Figure
  def initialize(p, q)
    @p, @q = p, q
    @ins=intr?(@p)+intr?(@q)
  end
  ...
end

class Polygon < Figure
  attr_reader :points, :perimeter, :area
  ...
  @area      = R2Point.area(a, b, c).abs
  @ins=intr?(a)+intr?(b)+intr?(c)
  end
```

Также нам нужно будет проверить пересекаются ли прямые, на которых лежит ребро выпуклой оболочки и заданный отрезок. Для этого добавим в файл

convex/convex.rb метод cross? в класс Segment, который будет проверять данное условие.

```
def cross?(point1,point2)
 if (point1.x==point2.x && self.p.x==self.q.x)
  return false
 else
  an=(point2.y-point1.y)/(point2.x-point1.x)
  bn=(self.q.y-self.p.y)/(self.q.x-self.p.x)
  if an == bn
   return false
  else
   return true
  end
 end
end
```

Теперь необходимо модифицировать класс Polygon, изменив инициализацию объекта и метод add(t):

Объект класса Polygon создается тогда, когда 3 вершины выпуклой оболочки образуют треугольник. Поэтому в инициализации мы будем проверять эти 3 вершины

```
class Polygon < Figure
  attr_reader :points, :perimeter, :area
  def initialize(a, b, c)
    ...
    @ins=intr?(a)+intr?(b)+intr?(c)
  end
```

При добавлении вершины выпуклой оболочки проверяется, нет ли из этой точки освещенных ребер. Если их нет, значит добавленная вершина находится внутри или на границе прежней оболочки и она "поглощается" старой оболочкой, а значит, пересчитывать ничего не надо. Если же хотя бы одно ребро освещено из этой точки, то все освещенные ребра удаляются, а концы получившейся ломаной соединяется с новой вершиной. При этом могут быть удалены точки, которые удовлетворяли условию. Для этого в циклах удаления точек надо проверять, подходили ли они нам ранее.

```
  while t.light?(p, @points.first)
    ...
    @ins -= intr?(p)
    ...
  end
  ...

  while t.light?(@points.last, p)
   ...
   @ins -= intr?(p)
   ...
  end
```

После этого проверяем, подходит ли новая вершина оболочки нашему условию.

```
      ...
      @ins += intr?(t)
      end
    self
  end
end
```

На этом модификация файла `convex/convex.rb` завершена. Теперь необходимо добавить графическую часть. Необходимо модифицировать в файле `convex/tk_drawer.rb` метод `draw` класса `Figure` и добавить методы `draw_poly`, `draw_line_1` и `draw_circle`:

```
class Figure
  def draw
    ...
    TkDrawer.draw_poly(@@a,@@b,@@c)
    TkDrawer.draw_line_1(@@a,@@b)
    TkDrawer.draw_line_1(@@b,@@c)
    TkDrawer.draw_line_1(@@a,@@c)
    TkDrawer.draw_circle(@@a)
    TkDrawer.draw_circle(@@b)
    TkDrawer.draw_circle(@@c)
    TkDrawer.draw_line(@@a,@@b)
    TkDrawer.draw_line(@@b,@@c)
    TkDrawer.draw_line(@@a,@@c)
  end
end
```

Все вышеуказанные методы получают на вход переменные класса `Figure`, точки заданного треугольника.

Метод `draw_poly(a,b,c)` изображает заполненный полигон без контура, создавая объект класса `TkcPolygon` библиотеки `Tk`.

```
  def TkDrawer.draw_poly(a,b,c)
    TkcPolygon.new(CANVAS, x(a), y(a), x(b), y(b), x(c), y(c), 'fill'=>'green')
  end
```

Метод `draw_line_1(a,b)` изображает отрезок с толщиной равной 2SCALE(по SCALE с левой и правой стороны треугольника соответственно). Так мы будем изображать окрестность сторон треугольника.

```
  def TkDrawer.draw_line_1(p,q)
    TkcLine.new(CANVAS, x(p), y(p), x(q), y(q), 'width'=>2*SCALE, 'fill'=>'green' )
  end
```

Метод `draw_circle` изображает круг радиуса SCALE, который является единичной окрестностью вершин заданного треугольника.

```
 def TkDrawer.draw_circle(p)
    TkcOval.new(CANVAS, x(p) + SCALE, y(p) + SCALE, x(p) - SCALE, y(p) - SCALE,
    'fill'=>"green")
  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},
    count inside points = #{fig.inside_points}"
```

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

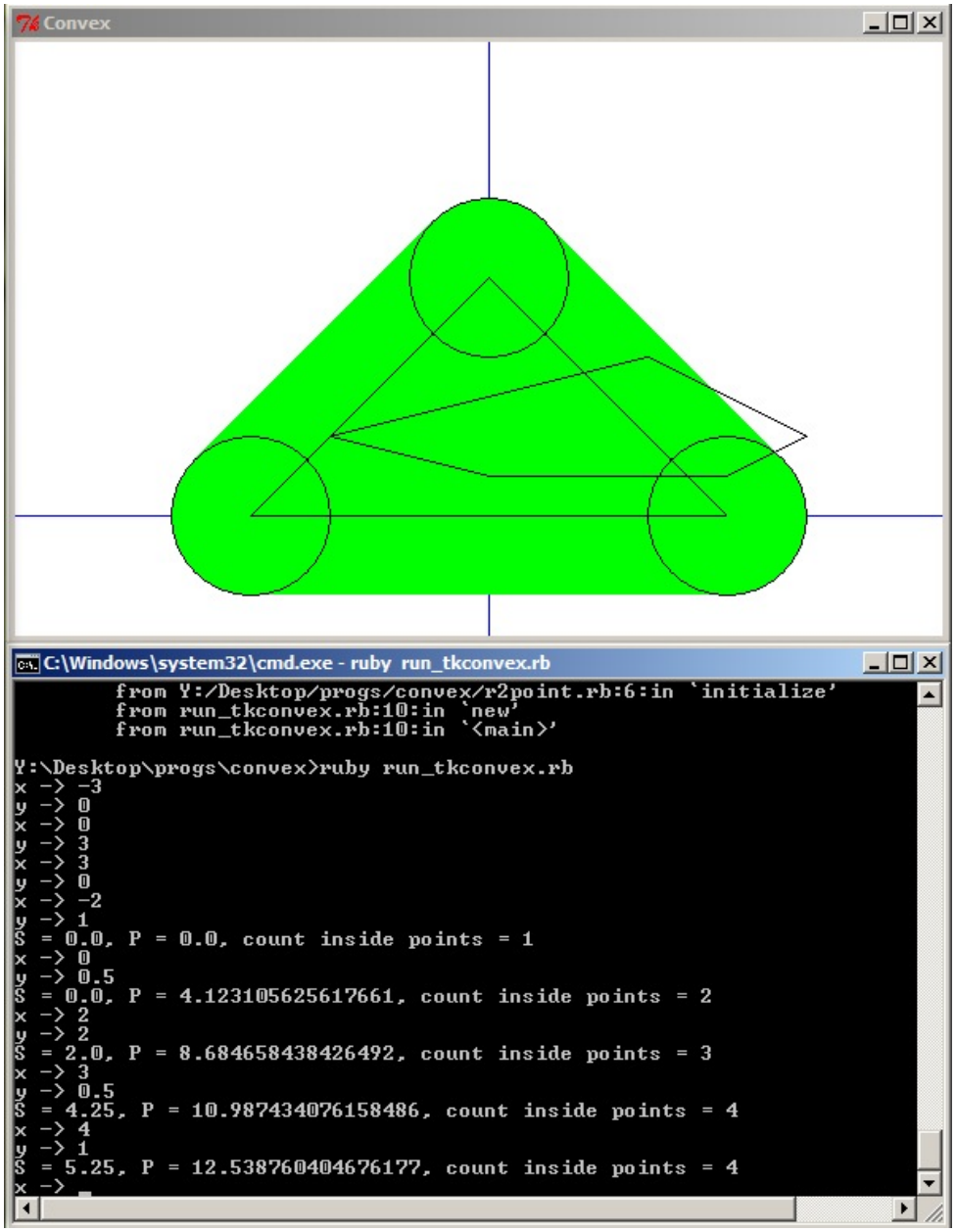


Рис. 1. Работа программы «Выпуклая оболочка»

3. Модификация проекта «Изображение проекции полиэдра»

Точная постановка задачи

Модифицируйте эталонный проект таким образом, чтобы определялась и печаталась следующая характеристика полиэдра: сумма длин проекций невидимых частей частично видимых рёбер, образующих с горизонтальной плоскостью угол не более $\pi/7$, центр которых находится строго внутри сферы $x^2 + y^2 + z^2 = 4$.

Решение данной задачи

Чтобы успешно решить данную задачу необходимо изменить код в файлах `shadow/polyedr.rb`, `common/polyedr.rb` и `shadow/run_polyedr.rb` Методы проверки ребра на частичную видимость, на проверку угла наклона к горизонтальной плоскости, на попадание центра ребра в заданную сферу и для вычисления длины проекции выполняется в методе `magic` класса `Polyedr` в файле `shadow/polyedr.rb`:

```
class Polyedr
  ...
 def magic()
    result = 0
    my_edges=edges.dup
    my_edges.each{|e| facets.each{|f| e.shadow(f)}}.uniq!
    my_edges.each do |e|
      if e.is_good?
        last=0.0
        e.gaps.each{|s| result+=e.r3(last).proection(e.r3(s.beg)); last=s.fin}
        result+=e.r3(last).proection(e.r3(1.0))
      end
    end
    result
  end
```

Модификация кода

В файле `common/polyedr.rb` дополним инициализацию и добавим методы в классе `Edge`. Объекты этого класса теперь будут хранить коэффициент гомотетии. Чтобы модификация не портила прохождение тестов для эталонного проекта, мы сделаем аргумент `coef` по умолчанию равным единице:

```
class Edge
  ...
  def initialize(b, f, coef=1.0)
    @beg, @fin, @coef = b, f, coef
  end
    ...
end
```

Так как в некоторых фигурах (например, в кубе) одно и то же ребро может принадлежать разным граням, необходимо реализовать метод `uniq` для класса `Edge`. Для этого нужно добавить 2 метода `eql?` и `hash`:

```
class Edge
  ...
  def eql?(other)
    (@beg==other.beg && @fin==other.fin) or (@beg==other.fin && @fin==other.beg)
  end
  def hash
    @beg.hash + @fin.hash
  end
end
```

В данной реализации приметяется оператор == для объектов класса `R3`, где этот метод не реализован. Чтобы этот оператор работал в классе `R3` добавим метод `eql?`:

```
class R3
  ...
  def eql?(other)
    @x==other.x && @y==other.y && @z==other.z
  end
end
```

Для выяснения, больше ли угол наклона к горизонтальной плоскости чем $\pi/7$, реализуем метод `angle_with_xy`. В данном методе используется метод `atan` библиотеки математических функций Ruby `Math`. Этот метод возвращает арктангенс угла. В нашем случае, арктангенс нужного нам угла находится как отношение координаты z к проекции на плоскость Oxy. Метод `atan` возвращает значения в диапазоне $[-\pi/2,\ \pi/2]$.

```
class R3
  ...
  def angle_with_xy
    atan(@z/sqrt(@x**2+@y**2))<=::PI/7
  end
  ...
end
```

Так же добавим метод `proection(other)`, который будет вычислять длину проекции отрезка, в котором началом является объект класса `R3`, от которого вызывается метод, а концом — объект класса `R3`, который передается в качестве аргумента.

```
class R3
  ...
  def proection(other)
    sqrt((other.x-@x)**2+(other.y-@y)**2)
  end
  ...
end
```

На этом модификация файла `common/polyedr.rb` завершена.Теперь необходимо модифицировать файл `swadow/polyedr.rb`.

Для проверки ребра на частичную видимость необходимо проверить массив `@gaps`. Так как элементами массива `@gaps` являются объекты класса `Segment` и в этом классе есть доступ к чтению аттрибутов `beg` и `fin`, которые соответственно являются началом и концом видимого отрезка, то мы можем получить эти значения. Значения `beg` и `fin` могут варьироваться в промежутке от [0,1]. В методе `is_good?` мы суммируем длины всех видимых участков и если сумма не равна нулю (полностью невидимое ребро) и не равна 1(полностью видимое ребро), то проверяемое ребро является частично видимым. Так как вычисления производятся на компьютере, используется погрешность 0.000001:

```
class Edge
  ...
  def is_good?()
    sum=0
    @gaps.each{|x| sum=+x.fin-x.beg}
    sum > 0.000001 && sum < 0.999999
  end
  ...
end
```

Проверку центра ребра на принадлежность сфере реализует метод `is_center_good?` из класса `Edge`. Он базируется на базовых знаниях векторной алгебры. Координаты середины отрезка $[a, b]$ вычисляются по формуле:

$$\frac{a_x + b_x}{2};\ \frac{a_y + b_y}{2};\ \frac{a_z + b_z}{2}.$$

Из-за коэффициента гомотетии вероятность того, что центр ребра попадет в сферу радиуса 4, становится очень малой, поэтому мы делим каждую координату на коэффициент гомотетии.

```
  def is_center_good?()
    xc = (@beg.x + @fin.x) / (2.0*@coef)
    yc = (@beg.y + @fin.y) / (2.0*@coef)
    zc = (@beg.z + @fin.z) / (2.0*@coef)
    return ((xc**2+yc**2+zc**2)<4)
  end
```

На этом модификация файла `shadow/polyedr.rb` завершена. Последняя модификация в файле `shadow/run_polyedr.rb` – вывод результата на экран:

```
#!/usr/bin/env ruby
require_relative './polyedr'
require_relative '../common/tk_drawer'
TkDrawer.create
%w(ccc box cube king).each do |name|
  puts '============================================================'
  puts "Начало работы с полиэдром '#{name}'"
  start_time = Time.now
  a=Polyedr.new("../data/#{name}.geom")
  a.draw
  puts "summ = #{a.magic}"
  puts "Изображение полиэдра '#{name}' заняло #{Time.now - start_time} сек."
  print 'Hit "Return" to continue -> '
  gets
end
```

Модификация эталонного проекта «Изображение проекции полиэдра» завершена. Пример работы программы с модифицированным файлом `test1.geom` можно увидеть на рис.2 и его содержание представленно ниже.

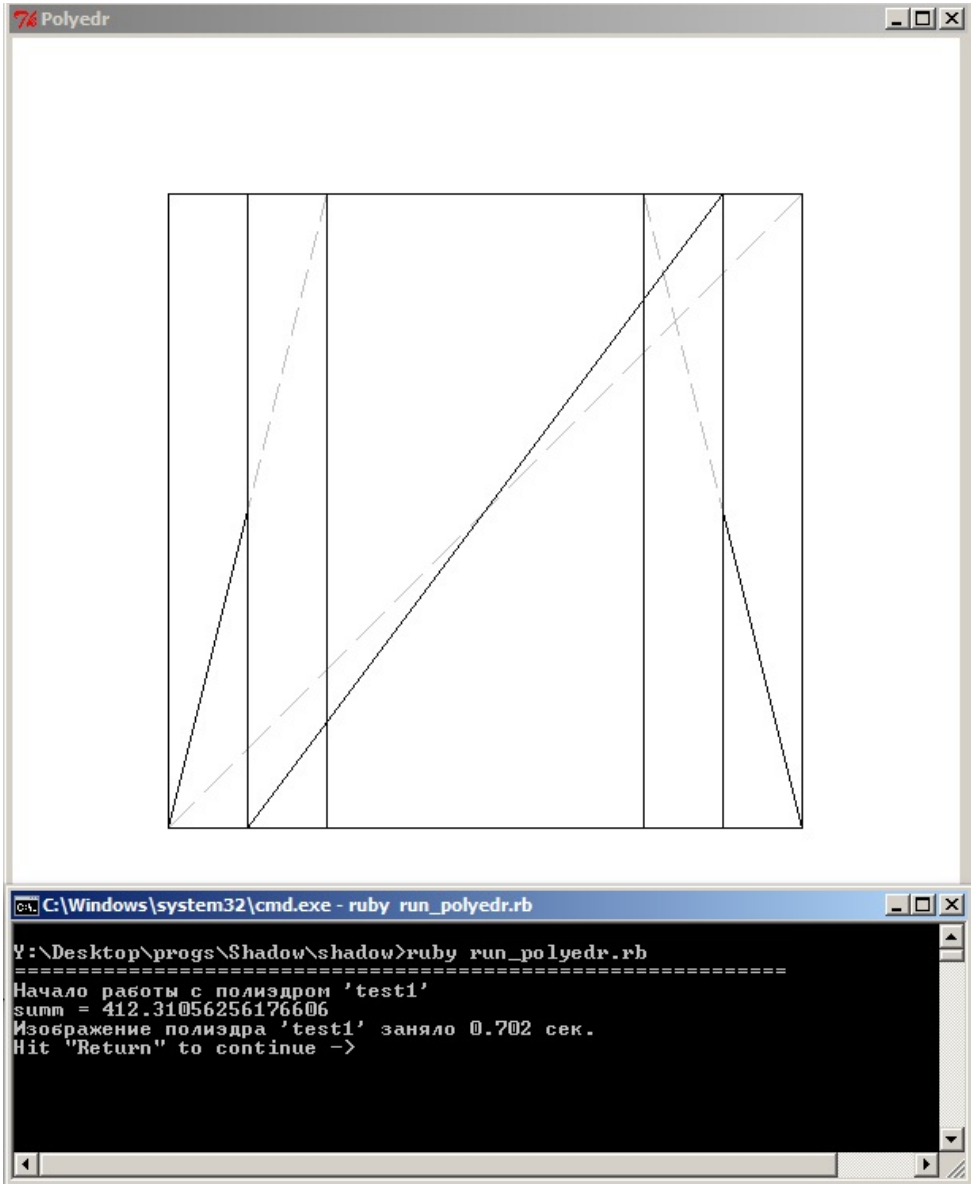


Рис. 2. Работа программы «Изображение проекции полиэдра»

```
200.0 0 0 0
12 12 36
-1 -1 0
-1 1 0
1 1 0
1 -1 0
-0.5 -1 0.23
-0.5 1 0.23
0.5 1 0.23
0.5 -1 0.23
-0.75 -1 0.23
-0.75 1 0.23
0.75 1 0.23
```

```
0.75 -1 0.23
3 1 2 3
3 1 4 3
3 1 2 6
3 1 5 6
3 1 4 8
3 1 5 8
3 4 3 7
3 4 8 7
3 2 3 7
3 2 6 7
3 9 10 11
3 9 12 11
```

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/Псевдоскалярное_произведение` — Статья о псевдоскалярном произведении

[7] `http://algolist.manual.ru/maths/geom/distance/pointline.php` — Расстояние от точки до отрезка