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 периметра части выпуклой оболочки, расположенной внутри заданного треугольника. Нахождение суммы длин проекций полностью невидимых рёбер, центр которых находится строго внутри сферы $x^2 + y^2 + z^2 = 4$»

Группа 2362

Студент И.Г. Чернятьев

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

Москва 2014

Аннотация

Работа посвящена модификации проектов «Выпуклая оболочка» и «Изображение проекции полиэдра». В первом из этих проектов решалась задача на вычисление периметра части выпуклой оболочки, расположенной внутри заданного треугольника. Во втором проекте вычислялась сумма длин проекций полностью невидимых рёбер, 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 части выпуклой оболочки. Целью данной работы является определение периметра части выпуклой оболочки, расположенной внутри 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 полностью невидимых рёбер, 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 700, из которых более 157 были изменены или добавлены автором в процессе работы над задачами модификации.

2. Модификация проекта «Выпуклая оболочка»

Точная постановка задачи

Вычисляется периметр части выпуклой оболочки, расположенной внутри заданного треугольника.

Решение данной задачи и модификация кода

Эталонный проект до модификации индуктивно строит выпуклую оболочку по точкам. Если добавляется новая точка, то программа индуктивно перевычисляет периметр и площадь оболочки. Наша цель модифицировать эталонный проект таким образом, чтобы вычислялся периметр части выпуклой оболочки, расположенной внутри заданного треугольника.

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

1. точка лежит строго снаружи треугольника;
2. точка лежит на границе треугольника;
3. точка лежит строго внутри треугольника;
4. точка лежит на прямой;
5. точка лежит на отрезке.

```
def outside_tr?(point1, point2, point3)
  !inside_tr?(point1, point2, point3) and !on_board?(point1, point2, point3)
end
```

```
  def on_board?(point1, point2, point3)
    (self.on_segment?(point1, point2) or self.on_segment?(point2, point3) or
self.on_segment?(point3, point1))
  end

  def inside_tr?(point1, point2, point3)
    (!(self.light?(point1, point2) or self.light?(point2, point3) or
self.light?(point3, point1)) and !self.on_board?(point1, point2, point3))
  end

  def on_line?(point1, point2)
    (self.x-point1.x)*(point2.y-point1.y)==(self.y-point1.y)*(point2.x-point1.x)
  end

  def on_segment?(point1, point2)
    point1.dist(point2) == self.dist(point1) + self.dist(point2)
  end
```

Теперь модифицируем файл `convex/convex.rb`, добавляя в каждый класс три точки треугольника: `point1, point2, point3`. Рассмотрим ещё несколько методов в классе `Segment`, которые нам понадобятся для решения данной задачи.

Найдём точку пересечения прямых с помощью решения системы линейных уравнений:

$$\begin{cases} A_1x + B_1y + C_1 = 0 \\ A_2x + B_2y + C_2 = 0 \end{cases}$$

Добавим в файл `convex/convex.rb` метод `crosspoint` в класс `Segment`, который будет вычислять точку пересечения прямых. В этом методе будем рассматривать несколько случаев:

1. если первая прямая не вертикальна;
2. если первая вертикальна, а вторая нет.

```
def crosspoint(point1, point2)
  if point1.x != point2.x
    if self.p.x != self.q.x
      b1 = point1.y - (k1 = (point2.y - point1.y)/(point2.x - point1.x))*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 = point1.y - (k1 = (point2.y - point1.y)/(point2.x - point1.x))*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(point1.x, k2*point1.x + b2)
  end;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
```

Ещё нам необходимо проверить является ли найденная точка пересечением прямых или точкой пересечения отрезков. Чтобы этот случай учитывался добавим в файл `convex/convex.rb` метод `is_on_segments?` в класс `Segment`.

```
def is_on_segments?(point1,point2)
 j=crosspoint(point1,point2).inside?(point1,point2)
 k=crosspoint(point1,point2).inside?(@p,@q)
 if cross?(point1,point2)
  j and k
 else
  false
 end
end
```

Теперь можно приступить к созданию нового метода, который будет вычислять периметр части ребра, лежащего внутри или на границе заданного треугольника. Для этого создадим новый метод `part_perimeter` и добавим его в класс `Segment`.

Для начала проверим с помощью метода `outside_tr?` лежат ли обе точки отрезка внутри или на границе треугольника (рис.1).

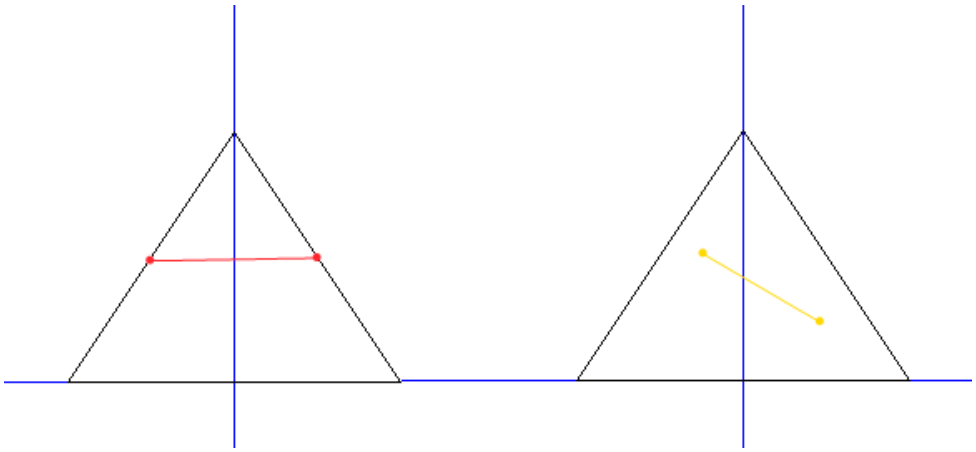

Рис. 1.

Если точки лежат, то возвращаем расстояние от одной точки до другой как периметр. Для этого используем метод `dist`, который вычисляет расстояние по формуле:

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

.

Если точки не лежат, то продолжаем проверку. Проверяем следующее условие, когда одна точка лежит снаружи, а другая находится строго внутри треугольника, тогда вычисляем расстояние от точки пересечения одной из сторон треугольника до точки лежащей строго внутри треугольника. Для нахождения точки пересечения используем метод `crosspoint` и выводим расстояние от найденой точки до точки лежащей строго внутри треугольника.

Если одна точка строго снаружи, а другая точка лежит на границе треугольника, тогда проверяем лежит ли одна точка на отрезке, а другая на прямой с помощью методов `on_segment? и on_line?`. Также необходимо учитывать, что обе точки могут лежать на одной прямой (рис. 2). Тогда выводим расстояние прямой лежащей в треугольнике или на его границе.

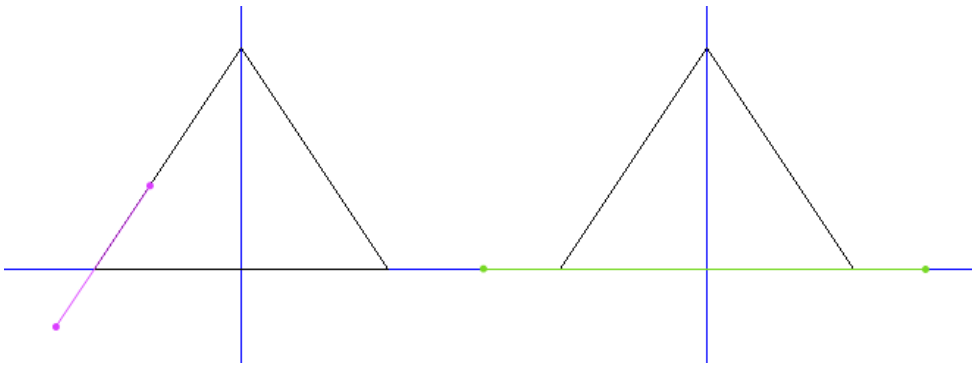

Рис. 2.

Для метода `is_on_segments?` создадим массив куда будем добавлять результат,

является ли найденная точка пересечения прямых - точкой пересечения отрезков. И выводим расстояние от одной точки до другой.

После рассмотренных случаев метод `part_perimeter` примет следующий вид:

```
class Segment < Figure
  ...
def part_perimeter()
  pt=!@p.outside_tr?(@point1,@point2,@point3)
  pt=!@q.outside_tr?(@point1,@point2,@point3)
  pq=@p.outside_tr?(@point1, @point2, @point3)
  qp=@q.inside_tr?(@point1, @point2, @point3)
  pv=@q.outside_tr?(@point1, @point2, @point3)
  qv=@p.inside_tr?(@point1, @point2, @point3)

  if pt and pt
    return @p.dist(@q)
  elsif pq and qp
    if is_on_segments?(@point1, @point2)
  return @p.dist(@q) - crosspoint(@point1, @point2).dist(@p)
elsif is_on_segments?(@point2, @point3)
  return @p.dist(@q) - crosspoint(@point2, @point3).dist(@p)
    elsif is_on_segments?(@point1, @point3)
  return @p.dist(@q) - crosspoint(@point1, @point3).dist(@p)
end
  elsif pv and qv
    if is_on_segments?(@point1, @point2)
  return @p.dist(@q) - crosspoint(@point1, @point2).dist(@q)
    elsif is_on_segments?(@point2, @point3)
  return @p.dist(@q) - crosspoint(@point2, @point3).dist(@q)
    elsif is_on_segments?(@point1, @point3)
  return @p.dist(@q) - crosspoint(@point1, @point3).dist(@q)
end
  else
    if @q.on_segment?(@point1, @point2) && @p.on_line?(@point1, @point2)
      return @p.dist(@q) - [@p.dist(@point1), @p.dist(@point2)].min
    elsif @q.on_segment?(@point2, @point3) && @p.on_line?(@point3, @point2)
      return @p.dist(@q) - [@p.dist(@point3), @p.dist(@point2)].min
    elsif @q.on_segment?(@point1, @point3) && @p.on_line?(@point3, @point1)
      return @p.dist(@q) - [@p.dist(@point3), @p.dist(@point1)].min
    end

      if @p.on_segment?(@point1, @point2) && @q.on_line?(@point1, @point2)
        return @p.dist(@q) - [@q.dist(@point1), @q.dist(@point2)].min
      elsif @p.on_segment?(@point2, @point3) && @q.on_line?(@point3, @point2)
        return @p.dist(@q) - [@q.dist(@point3), @q.dist(@point2)].min
      elsif @p.on_segment?(@point1, @point3) && @q.on_line?(@point3, @point1)
        return @p.dist(@q) - [@q.dist(@point3), @q.dist(@point1)].min
      end
```

```
      if @p.on_line?(@point1, @point2) && @q.on_line?(@point1, @point2)
     return @point1.dist(@point2)
      elsif @p.on_line?(@point3, @point2) && @q.on_line?(@point3, @point2)
    return @point3.dist(@point2)
      elsif @p.on_line?(@point1, @point3) && @q.on_line?(@point1, @point3)
    return @point1.dist(@point3)
  end

      array = []

  if is_on_segments?(@point1, @point2)
    array << crosspoint(@point1, @point2)
      elsif is_on_segments?(@point2, @point3)
    array << crosspoint(@point2, @point3)
      elsif is_on_segments?(@point1, @point3)
    array << crosspoint(@point1, @point3)
  end

      if array.size == 3
       if array[0] != array[1]
     return array[0].dist(array[1])
       elsif array[2] != array[1]
     return array[2].dist(array[1])
       elsif array[0] != array[2]
     return array[0].dist(array[2])
   end
      elsif array.size == 2
        return array[0].dist(array[1])
      else
        return 0.0
      end
    end
  end
...
end
```

Также нам необходимо модифицировать класс `Polygon`, добавив небольшие изменения в методы `initialize и add(t)`, а именно:

Вычисляем периметр части выпуклой оболочки лежащей в заданном треугольнике.

```
class Polygon < Figure
  attr_reader :points, :perimeter, :area, :part_perimeter
  ...
  def initialize(a, b, c, point1, point2, point3)
    ...
    ao=Segment.new(a, b, @point1, @point2, @point3).part_perimeter
    bo=Segment.new(b, c, @point1, @point2, @point3).part_perimeter
    co=Segment.new(a, c, @point1, @point2, @point3).part_perimeter
```

```
    @part_perimeter = ao + bo + co;  end
```

Когда добавляем новую точку нужно рассмотреть несколько случаев:

1. если освещённых ребер нет, что означает новая точка попала внутрь или на границу старой выпуклой оболочки и нам нечего не нужно делать;
2. если хотя бы одно освещённое ребро есть, то их нужно удалить и соединить концы оставшейся ломаной с новой точкой.

```
def add(t)
   ...
  if t.light?(@points.last, @points.first)
    ...
  dl=Segment.new(@points.first,@points.last,@point1,@point2,@point3)
  @part_perimeter -= dl.part_perimeter
    ...
```

Для каждого освещённого ребра нам нужно уменьшить периметр оболочки на его длину.

```
  while t.light?(p, @points.first)
    ...
  det=Segment.new(@points.first, p, @point1, @point2, @point3).part_perimeter
  @part_perimeter -= det
    ...
  end
    ...

  while t.light?(@points.last, p)
    ...
    get=Segment.new(p, @points.last, @point1, @point2, @point3).part_perimeter
    @part_perimeter -= get
        ...
  end
```

Затем нам необходимо добавить сумму длин двух новых рёбер.

```
      ...
   ml=Segment.new(@points.first, t, @point1, @point2, @point3).part_perimeter
   nl=Segment.new(t, @points.last, @point1, @point2, @point3).part_perimeter
@part_perimeter += ml + nl
      ...
    end
    self
  end
end
```

Но для того чтобы рисовался треугольник нам нужно модифицировать файл `convex/tk_drawer.rb` метод `draw` класса `Figure`:

```
class Figure
  def draw
    ...
     TkDrawer.draw_line(@point1, @point2)
     TkDrawer.draw_line(@point2, @point3)
     TkDrawer.draw_line(@point3, @point1)
  end
end
```

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

```
  fig = Void.new(R2Point.new, R2Point.new, R2Point.new)
    ...
  puts "S = #{fig.area}, P = #{fig.perimeter} perimetr = #{fig.part_perimeter}"
```

Модификация эталонного проекта «Выпуклая оболочка» завершена. Пример работы программы можно увидеть на (рис.3).

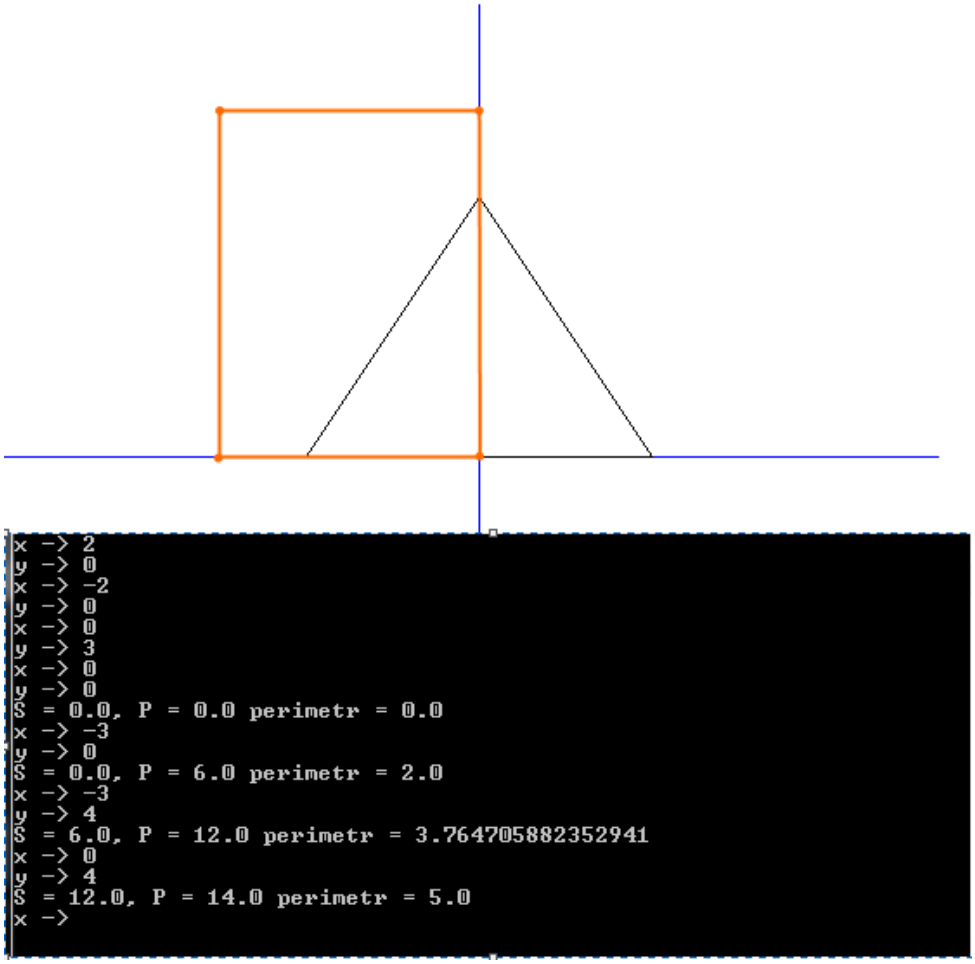

Рис. 3. Работа программы «Выпуклая оболочка»

3. Модификация проекта «Изображение проекции полиэдра»

Точная постановка задачи

Модифицируйте эталонный проект таким образом, чтобы определялась и печаталась следующая характеристика полиэдра: сумма длин проекций полностью невидимых рёбер, центр которых находится строго внутри сферы $x^2 + y^2 + z^2 = 4$.

Решение данной задачи и модификация кода

Для решения данной задачи модифицируем код файла `shadow/polyedr.rb`. Проверка ребра на невидимость, а также проверка центра ребра на попадание в сферу и вычисление периметра суммы длин проекций выполняется в методе `draw` класса `Polyedr`:

```
class Polyedr
  ...

  def draw
    ...
    p=0.0
      edges.each do |e|
        facets.each{|f| e.shadow(f)}
          e.gaps.each{|s| TkDrawer.draw_line(e.r3(s.beg), e.r3(s.fin))}
            if e.invisible?
              if e.func?(c)
                p+=e.perimeter(c)
              end
            end
      end
  return p
  end
end
```

Чтобы проверить, является ли ребро невидимым, нужно проверить является ли массив `@gaps` пустым. Для этого добавим в файл `shadow/polyedr.rb` метод `invisible?` в классе `Edge`:

```
class Edge
  ...
  def invisible?
    @gaps.size==0 ? true : false
  end
  ...
end
```

В файле `common/polyedr.rb` дополним метод `initialize(file)` в классе `Polyedr` заведем константу с коэффициентом гомотетии:

```
class Polyedr
  ...
  def initialize(file)
    ...
@c=c
    ...
  end
end
```

Чтобы проверить находится ли центр полностью невидемых ребер строго внутри сферы $x^2 + y^2 + z^2 = 4$ с учетом коэффициента гомотетии. Используем следующую формулу для проверки центра ребра:

$$x^2 + y^2 + z^2 < 4$$

Надо добавить в файле `shadow/polyedr.rb` метод `func?(c)` в класс `Edge`:

```
class Edge
  ...
  def func?(c)
    point=r3(0.5)
    point.x**2+point.y**2+point.z**2 < 4*c**2
  end
  ...
end
```

После чего нам необходимо посчитать суммы длин проекций. Суммы длин проекций с учетом коэффициента гомотетии будем искать с помощью формулы:

$$\frac{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}}{\mathrm{c}}$$

.

Добавим в файле `shadow/polyedr.rb` метод `perimetr(c)` в класс `Edge`, позволяющий вычислить длину проеции.

```
class Polyedr
  ...
  def perimeter(c)
    vx=@fin.x-@beg.x
    vy=@fin.y-@beg.y
    (Math.sqrt(vx**2+vy**2))/c
  end
  ...
end
```

Заключительная 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")
  puts "Сумма длин проекций #{a.draw}"
  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 `cube1.geom` можно увидеть на (рис.4) и его содержание представленно ниже.

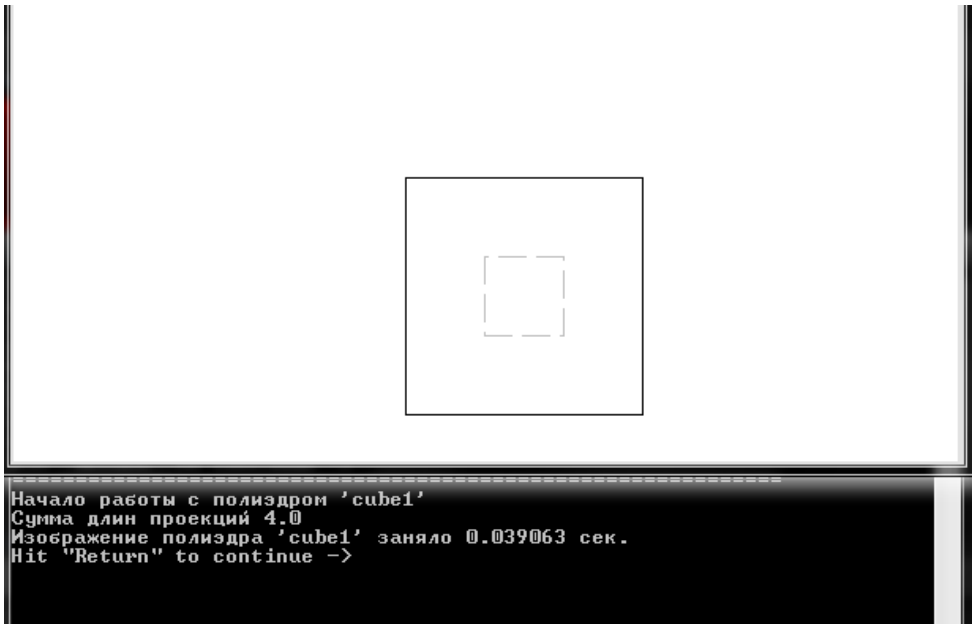

Рис. 4. Работа программы «Изображение проекции полиэдра»

```
50.0  0.0  0.0  0.0
 8  2  8
 0.0  0.0  0.0
 1.0  0.0  0.0
 1.0  1.0  0.0
 0.0  1.0  0.0
-1.0 -1.0  1.0
 2.0 -1.0  1.0
 2.0  2.0  1.0
-1.0  2.0  1.0
4  1  2  3  4
4  5  6  7  8
```

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».