Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №2 по курсу «Компьютерная графика»
Тема: Каркасная визуализация выпуклого многогранника. Удаление невидимых линий

Студент: И. Д. Недосеков
Преподаватель: Чернышов Л. Н.
Группа: М8О-306Б-19
Дата:
Оценка:
Подпись:

Москва, 2021

Содержание

1 Постановка задачи

Лабораторная работа №2

Разработать формат представления многогранника и процедуру его каркасной отрисовки в ортографической и изометрической проекциях. Обеспечить удаление невидимых линий и возможность пространственных поворотов и масштабирования многогранника. Обеспечить автоматическое центрирование и изменение размеров изображения при изменении размеров окна

Вариант:

17. Гранная прямая правильная усеченная пирамида

2 Описание программы

Программа написана на Golang[2] и Gtk[1] для пользовательского интерфейса. Разметка интерфейса хранится в main.glade, все расчеты точек графика и отрисовки в render.go. Проверка на видимость граней происходит через алгоритм Робертса[3]. В matrices.go операции по работе с координатами фигуры, таких как нахождение уравнение поверхности, поворот фигуры и тд.

Инструкция по установке:

- установить библиотеку GTK
- установить среду разработки Golang
- установить библиотеки для Golang (командой такого вида) `go get -v {репозиторий github}`
  - goGtk github.com/gotk3/gotk3/gtk
  - goCairo github.com/gotk3/gotk3/cairo
- скопировать файлы main.go, render.go, matrices.go, main.glade
- перейти в директорию проекта и запустить через команду `go run .`

3 Листинг программы

main.go

```
package main

import (
"log"

"github.com/gotk3/gotk3/gtk"
)

type Edge struct {
  u, v Point
}

type GlobalObj struct {
  figure                      []Edge
  figure_matr                 [][]float64
  figure_center, camera_point Point
}

var settings GlobalObj

func buildEdges(points []Point) (res []Edge) {
  for i, point := range points[:4] {
    res = append(res,
    Edge{point, points[i+4]},
    Edge{point, points[(i+1)%4]})
    }
    for i, point := range points[4:] {
      res = append(res, Edge{point, points[(i+1)%4+4]})
      }
      return
    }

    func matr2Points(matr [][]float64) (res []Point) {
      res = make([]Point, len(matr))
      for i, point := range matr {
        res[i] = Point{point[0], point[1], point[2]}
      }
      return
    }

    func calculateFigure(halfDist float64) ([]Edge, [][]float64, Point) {
```

```
42          upperHalf := halfDist / 2
43          points := []Point{
44            {halfDist, halfDist, 1}, {-halfDist, halfDist, 1}, {-halfDist, -halfDist,
              ↪  1}, {halfDist, -halfDist, 1},
45            {upperHalf, upperHalf, upperHalf + 20}, {-upperHalf, upperHalf, upperHalf +
              ↪  20}, {-upperHalf, -upperHalf, upperHalf + 20}, {upperHalf, -upperHalf,
              ↪  upperHalf + 20},
46          }
47
48          res := buildEdges(points)
49          res_matr := fig2matr(points)
50          center := getInnerPoint(res_matr)
51          return res, res_matr, center
52
53        }
54
55        func fig2matr(fig []Point) (matr [][]float64) {
56          matr = make([][]float64, len(fig))
57          for i, p := range fig {
58            matr[i] = []float64{p.x, p.y, p.z}
59          }
60          return
61        }
62
63        func rotateFig(matr [][]float64) {
64          settings.figure_matr, _ = multiplayMatr(settings.figure_matr, matr)
65          settings.figure = buildEdges(matr2Points(settings.figure_matr))
66          log.Println("Figure rotated by", matr)
67        }
68
69        func init() {
70          gtk.Init(nil)
71          settings.figure, settings.figure_matr, settings.figure_center =
            ↪  calculateFigure(50)
72          settings.camera_point = Point{0, 0, -5}
73          log.Println("Init past, fig", settings.figure)
74        }
75
76        func bindButton(builder *gtk.Builder, name string, callback func()) {
77          obj, err := builder.GetObject(name)
78          if err != nil {
79            log.Fatal(err)
80          }
81          button := obj.(*gtk.Button)
```

```
82          button.Connect("clicked", func() {
83            log.Println(name, "clicked")
84            callback()
85          })
86        }
87
88        func main() {
89
90          builder, err := gtk.BuilderNew()
91          if err != nil {
92            log.Fatal(err)
93          }
94
95          err = builder.AddFromFile("main.glade")
96          if err != nil {
97            log.Fatal(err)
98          }
99
100         obj, err := builder.GetObject("draw_area")
101         if err != nil {
102           log.Fatal(err)
103         }
104
105         da := obj.(*gtk.DrawingArea)
106         da.Connect("draw", drawCanwas)
107         da.Connect("draw", draw_figure)
108
109         bindButton(builder, "up_b", func() { rotateFig(rotateUp) })
110         bindButton(builder, "down_b", func() { rotateFig(rotateDown) })
111         bindButton(builder, "left_b", func() { rotateFig(rotateLeft) })
112         bindButton(builder, "right_b", func() { rotateFig(rotateRight) })
113
114         obj, err = builder.GetObject("main_window")
115         if err != nil {
116           log.Fatal(err)
117         }
118
119         win := obj.(*gtk.Window)
120         win.Connect("destroy", func() {
121           gtk.MainQuit()
122         })
123         win.SetTitle("LR 2 Ivan Nedosekov")
124
125         win.ShowAll()
```

```
126         gtk.Main()
127
128     }
129
```

render.go

```
1   package main
2
3   import (
4   "log"
5   "math"
6
7   "github.com/gotk3/gotk3/cairo"
8   "github.com/gotk3/gotk3/gtk"
9   )
10
11  func drawCanwas(da *gtk.DrawingArea, cr *cairo.Context) {
12    cr.SetSourceRGB(255, 255, 255)
13
14    w, h := da.GetAllocatedWidth(), da.GetAllocatedHeight()
15
16    cr.Rectangle(0, 0, float64(w), float64(h))
17    cr.Fill()
18    log.Printf("Draw Canvas w=%d, h=%d", w, h)
19  }
20
21  func calculateArrowPoints(startX, startY, endX, endY, legth float64) (x1, y1, x2,
    ↪  y2 float64) {
22    const wingsAngel = math.Pi / 10
23    angle := math.Atan2(endY-startY, endX-startX) + math.Pi
24    x1 = endX + legth*math.Cos(angle-wingsAngel)
25    y1 = endY + legth*math.Sin(angle-wingsAngel)
26    x2 = endX + legth*math.Cos(angle+wingsAngel)
27    y2 = endY + legth*math.Sin(angle+wingsAngel)
28    log.Printf("Calculated x1=%f, y1=%f, x2=%f, y2=%f, ang1=%f, ang2=%f", x1, y1, x2,
      ↪  y2, math.Cos(angle-wingsAngel), math.Cos(angle+wingsAngel))
29    return
30  }
31
32  func drawArrows(cr *cairo.Context, startX, startY, endX, endY float64) {
33    cr.MoveTo(startX, startY)
34    cr.LineTo(endX, endY)
35
```

```
    x1, y1, x2, y2 := calculateArrowPoints(startX, startY, endX, endY, 10)
    cr.MoveTo(endX, endY)
    cr.LineTo(x1, y1)
    cr.MoveTo(endX, endY)
    cr.LineTo(x2, y2)
    cr.Stroke()
}

func rotatePoint(angle, cx, cy, x, y float64) (float64, float64) {
    s := math.Sin(angle)
    c := math.Cos(angle)

    // translate point back to origin:
    x -= cx
    y -= cy

    // rotate point
    xnew := x*c - y*s
    ynew := x*s + y*c

    // translate point back:
    x = xnew + cx
    y = ynew + cy
    return x, y
}

func drawAxis(da *gtk.DrawingArea, cr *cairo.Context) {
    cr.SetSourceRGB(0, 0, 0)
    cr.SetLineWidth(1)
    w, h := float64(da.GetAllocatedWidth()), float64(da.GetAllocatedHeight())

    horizontalPadding := w * 0.02
    verticalPadding := h * 0.05

    startX, startY := horizontalPadding, h-verticalPadding
    endX, endY := w-horizontalPadding, h-verticalPadding

    drawArrows(cr, startX, startY, endX, endY)

    startX, startY = horizontalPadding, h-verticalPadding
    endX, endY = horizontalPadding, verticalPadding

    drawArrows(cr, startX, startY, endX, endY)

```

```
80    startX, startY = horizontalPadding, h-verticalPadding
81    endX, endY = rotatePoint(-math.Pi/600, horizontalPadding, h-verticalPadding,
      ↪  (w-horizontalPadding)*1/7, (h-verticalPadding)*1/7)
82
83    drawArrows(cr, startX, startY, endX, endY)
84
85    log.Printf("Axis rendered: w=%f h=%f hpad=%f\n", w, h, horizontalPadding)
86  }
87
88  type Point struct {
89    x, y, z float64
90  }
91
92  func draw_figure(da *gtk.DrawingArea, cr *cairo.Context) {
93
94    cr.SetSourceRGB(0, 0, 0)
95    cr.SetLineWidth(1)
96    w, h := float64(da.GetAllocatedWidth()), float64(da.GetAllocatedHeight())
97    cX, cY := w/2, h/2
98
99    //tmp_fig := make([]Edge, len(settings.figure))
100   //copy(tmp_fig, settings.figure)
101
102   type args struct{ a, b Edge }
103   var planes []args
104   for i := 0; i < 8; i += 2 {
105     planes = append(planes, args{settings.figure[i], settings.figure[i+1]})
106   }
107   // donw plane
108   planes = append(planes, args{settings.figure[1], settings.figure[5]})
109   // upper plane
110   planes = append(planes, args{settings.figure[8], settings.figure[9]})
111   //log.Println("given figure", settings.figure)
112
113   log.Println("constructed plane by edges")
114   for i, p := range planes {
115     log.Println("plane", i, p)
116   }
117
118   var A, B, C, D float64
119   visible_edge := make(map[Edge]struct{})
120
121   for i, plane := range planes {
122     A, B, C, D = findPlane(plane.a, plane.b, settings.figure_center)
```

```
123
124       if !is_visible(A, B, C, D, settings.figure_center, settings.camera_point) {
125         log.Println("Plane", i, A, B, C, D, "skipped")
126         continue
127       }
128       switch i {
129         case 5:
130         log.Println("Upper visible")
131         for j := 8; j < 12; j++ {
132           visible_edge[settings.figure[j]] = struct{}{}
133         }
134         case 4:
135         log.Println("Down visible")
136         for j := 1; j < 8; j += 2 {
137           visible_edge[settings.figure[j]] = struct{}{}
138         }
139         default:
140         log.Println(i, "plane visible")
141         j := i * 2
142         visible_edge[settings.figure[j]] = struct{}{}
143         visible_edge[settings.figure[j+1]] = struct{}{}
144         visible_edge[settings.figure[(j+2)%8]] = struct{}{}
145         visible_edge[settings.figure[i+8]] = struct{}{}
146
147       }
148     }
149
150     tmp_fig := make(map[Edge]Edge)
151     for edge := range visible_edge {
152       u := edge.u
153       v := edge.v
154       tmp_fig[edge] = Edge{Point{cX + u.x, cY + u.y, -1}, Point{cX + v.x, cY + v.y,
        ↪  -1}}
155     }
156     log.Println("Pased norming", tmp_fig)
157
158     for _, edge := range tmp_fig {
159       log.Println("Draw edge", edge)
160       cr.MoveTo(edge.u.x, edge.u.y)
161       cr.LineTo(edge.v.x, edge.v.y)
162     }
163     cr.Stroke()
164
165   }
```

matrices.go

```go
package main

import (
"errors"
"log"
"math"
)

func multiplayMatr(a, b [][]float64) ([][]float64, error) {
  if len(a[0]) != len(b) {
    return nil, errors.New("not multiplicatable martcies")
  }
  res := make([][]float64, len(a))
  for i := 0; i < len(a); i++ {
    res[i] = make([]float64, len(b[0]))
    for j := 0; j < len(b); j++ {
      for r := 0; r < len(a[0]); r++ {
        res[i][j] += a[i][r] * b[r][j]
      }
    }
  }
  return res, nil

}

const alpha = math.Pi / 6

var rotateUp = [][]float64{
  {1, 0, 0},
  {0, math.Cos(alpha), -math.Sin(alpha)},
  {0, math.Sin(alpha), math.Cos(alpha)}}

var rotateDown = [][]float64{
  {1, 0, 0},
  {0, math.Cos(-alpha), -math.Sin(-alpha)},
  {0, math.Sin(-alpha), math.Cos(-alpha)}}

var rotateLeft = [][]float64{
  {math.Cos(alpha), -math.Sin(alpha), 0},
  {math.Sin(alpha), math.Cos(alpha), 0},
  {0, 0, 1}}

```

```
43  var rotateRight = [][]float64{
44    {math.Cos(-alpha), -math.Sin(-alpha), 0},
45    {math.Sin(-alpha), math.Cos(-alpha), 0},
46    {0, 0, 1}}
47
48  func is_inner_dot(A, B, C, D float64, p Point) bool {
49    return A*p.x+B*p.y+C*p.z+D >= 0
50  }
51  func findPlane(a, b Edge, innerPoint Point) (A, B, C, D float64) {
52
53    p := []Point{a.u, a.v, b.v}
54    log.Println("find plane to points", p)
55
56    det := [3][3]float64{
57      {},
58      {p[1].x - p[0].x, p[1].y - p[0].y, p[1].z - p[0].z},
59      {p[2].x - p[0].x, p[2].y - p[0].y, p[2].z - p[0].z}}
60
61    det1 := det[1][1]*det[2][2] - (det[1][2] * det[2][1])
62    det2 := det[1][0]*det[2][2] - (det[1][2] * det[2][0])
63    det3 := det[1][0]*det[2][1] - (det[1][1] * det[2][0])
64
65    // log.Printf("de1 %f\n det2 %f\n det3 %f\n", det1, det2, det3)
66
67    A = det1
68    D = -p[0].x * det1
69    B = -det2
70    D -= -p[0].y * det2
71    C = det3
72    D += -p[0].z * det3
73    log.Println("Finded coefs", A, B, C, D)
74    if is_inner_dot(A, B, C, D, innerPoint) {
75      log.Print("Calculated positive normal to dot")
76      return A, B, C, D
77    } else {
78      log.Print("Calculated negative normal to dot")
79      return -A, -B, -C, -D
80    }
81  }
82
83  type Vector struct{ i, j, k float64 }
84
85  func dotProduct(a, b Vector) float64 {
86    res := a.i*b.i + a.j*b.j + a.k*b.k
```

```
87    log.Println("Product of", a, b, "is", res)
88    return res
89  }
90
91  func is_visible(A, B, C, D float64, innerPoint, cameraPoint Point) bool {
92    N := Vector{A, B, C}
93    Cam := Vector{innerPoint.x - cameraPoint.x, innerPoint.y - cameraPoint.y,
      ↪  innerPoint.z - cameraPoint.z}
94    log.Println("Checking visible N", N, "Cam", Cam, "camPoint", cameraPoint,
      ↪  "inner", innerPoint, "D", D)
95    return dotProduct(N, Cam) > 0
96  }
97
98  func getInnerPoint(figure [][]float64) Point {
99    var res Point
100   for _, point := range figure {
101     res.x += point[0]
102     res.y += point[1]
103     res.z += point[2]
104   }
105   count := float64(len(figure))
106   res.x, res.y, res.z = res.x/count, res.y/count, res.z/count
107   return res
108 }
```

4 Тесты

1 Наборы тестов

1. вид снизу
2. вид сбоку
3. вид с нижнего угла

2 Визуализация тестов

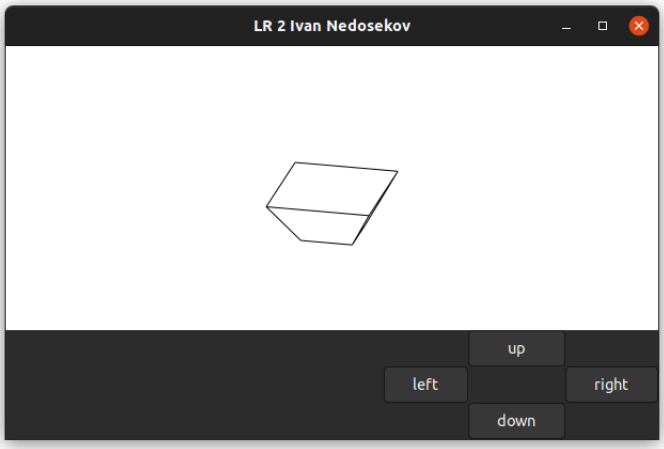

Рис. 1: 1ый тестовый набор

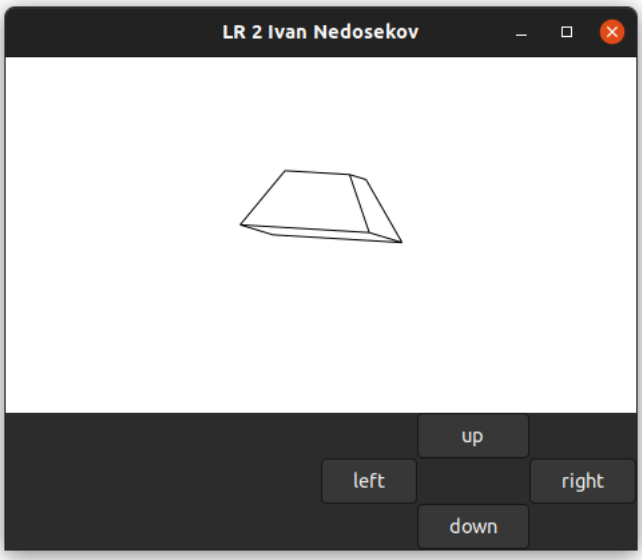

Рис. 2: 2ой тестовый набор

Рис. 3: 3ий тестовый набор

5 Выводы

Выполнив данную лабораторную работу, я познакомился с алгоритмом вычисления видимости граней Робертса. Но написание и дебаг математики не самое приятное для программиста занятие, поэтому нужно переходить на OpenGl где скорее всего есть более богатый встроенный инструментарий для отрисовки примитивов.

Список литературы

[1] *GTK — официальная документация.*
URL: www.gtk.org

[2] *Golang — официальная документация.*
URL: golang.org

[3] *Алгортм Робертса*
URL: algolist.ru/graphics/roberts.php