# Московский авиационный институт (национальный исследовательский университет)

## Факультет информационных технологий и прикладной математики

## Кафедра вычислительной математики и программирования


**Курсовой проект по курсу «Компьютерная графика»**


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

Дата:
Оценка:
Подпись:

**Москва, 2021**

# Содержание

# 1 Постановка задачи

Составить и отладить программу, обеспечивающую каркасную визуализацию порции поверхности заданного типа. Исходные данные готовятся самостоятельно и вводятся из файла или в панели ввода данных. Должна быть обеспечена возможность тестирования программы на различных наборах исходных данных. Программа должна обеспечивать выполнение аффинных преобразований для заданной порции поверхности, а также возможность управлять количеством изображаемых параметрических линий. Для визуализации параметрических линий поверхности разрешается использовать только функции отрисовки отрезков в экранных координатах.

# 2 Описание программы

Программа написана на Golang[2] и OpenGL[1] все расчеты точек графика и отрисовки в vertex_calculator.go. В compile.go операции по работе с шейдерами.
Инструкция по установке:

- установить среду разработки Golang
- установить библиотеки для Golang (командой такого вида) `go get -v {репозиторий github}`
  - github.com/go-gl/gl/v3.3-core/gl
  - github.com/go-gl/glfw/v3.3/glfw
  - github.com/go-gl/mathgl/mgl32
  - github.com/inkyblackness/imgui-go/v4
- скопировать файлы main.go, vertex_calculator.go, compile.go
- перейти в директорию проекта и запустить через команду `go run .`

Описание полей настройки:

- node $N$ $(0 \leq N \leq 6)$ — изменение координаты $y$ у узла (изменение формы поверхности)
- count dots — количество точек на кривой которы берем для отрисовки поверхности (до вращения)
- count rotates — количество поворотов кривой (аппроксимация поверхности)

# 3 Листинг программы

main.go

```go
package main

import (
"log"

"fmt"

"runtime"

g "github.com/AllenDang/giu"
math "github.com/chewxy/math32"
"github.com/go-gl/gl/v3.3-core/gl"
mgl "github.com/go-gl/mathgl/mgl32"
)

var (
count_dots  int32 = 10
N           []func(float32) float32
r_vec       func(z float32) mgl.Vec2
l, r        float32
nodes       []mgl.Vec2
dot_sliders []g.Widget
count_rot   int32 = 12

mesh [][]mgl.Vec3
)

func FindMinMax(t []float32) (min_el, max_el float32) {
  if len(t) == 0 {
    log.Default().Print("Error zerro leght FindMinMax")
  }
  for _, el := range t {
    min_el = math.Min(el, min_el)
    max_el = math.Max(el, max_el)
  }
  return
}

func divided_difference(f func(float32) func(float32) float32, t ...float32)
↪  func(float32) float32 {

```

```
41    if len(t) == 2 {
42
43      res := func(z float32) float32 {
44        if t[1]-t[0] == 0 {
45          return 0
46        }
47        return (f(t[1])(z) - f(t[0])(z)) / (t[1] - t[0])
48
49      }
50      return res
51    } else if len(t) < 2 {
52      log.Fatal("len < 2")
53    }
54    n := len(t) - 1
55    d1 := divided_difference(f, t[1:n]...)
56    d2 := divided_difference(f, t[0:n-1]...)
57
58    return func(z float32) float32 {
59      return (d1(z) - d2(z)) / (t[n] - t[0])
60    }
61  }
62
63  func truncated_power_function(n int, t float32) func(float32) float32 {
64    n_float := float32(n)
65    return func(z float32) float32 {
66      if z <= t {
67        return 0.
68      }
69
70      return math.Pow((z - t), n_float)
71    }
72
73  }
74
75  func truncated_power_function_fixed_n(n int) func(float32) func(float32) float32 {
76
77    return func(t float32) func(float32) float32 {
78      return truncated_power_function(n, t)
79    }
80  }
81
82  func calculate_N(n, k int) (float32, float32, []func(float32) float32) {
83
84    t_max := n - k + 2
```

```
85      var t []float32
86      N := make([]func(float32) float32, n+1)
87
88      for i := 1; i < k; i++ {
89        t = append(t, 0)
90      }
91
92      for i := 0; i <= t_max; i++ {
93        t = append(t, float32(i))
94      }
95
96      for i := 1; i < k; i++ {
97        t = append(t, float32(t_max))
98      }
99      sigma := truncated_power_function_fixed_n(k - 1)
100     norm := float32(t_max - 0)
101     for i := range N {
102       tmp_i := i
103       d := divided_difference(sigma, t[tmp_i:tmp_i+k+1]...)
104       N[i] = func(z float32) float32 {
105         return norm * d(z)
106       }
107     }
108
109     return 0, float32(t_max), N
110
111   }
112
113   func format_r(node []mgl.Vec2) func(z float32) mgl.Vec2 {
114
115     res_f := func(z float32) mgl.Vec2 {
116       res := mgl.Vec2{z, 0}
117       for i, ri := range node {
118         j := i
119         tmp_ri := ri
120         res = res.Add(tmp_ri.Mul(N[j](z)))
121       }
122       return res
123     }
124
125     return res_f
126
127   }
128
```

```
129  func calculate_line(l, r float32, f func(float32) mgl.Vec2) []mgl.Vec2 {
130    res := make([]mgl.Vec2, count_dots)
131    for i := range res {
132      t := (r - l) * float32(i) / float32(count_dots)
133      res[i] = f(t)
134    }
135    return res
136
137  }
138
139  func init() {
140    nodes = []mgl.Vec2{
141      {0, 1},
142      {0, -1},
143      {0, 1},
144      {0, -1},
145      {0, 1},
146      {0, -1},
147      {0, 1},
148    }
149    runtime.LockOSThread()
150    if err := gl.Init(); err != nil {
151      panic(err)
152    }
153
154  }
155
156  func to_vec3(v []mgl.Vec2) []mgl.Vec3 {
157    res := make([]mgl.Vec3, len(v))
158
159    for i, el := range v {
160      res[i] = el.Vec3(0)
161    }
162
163    return res
164
165  }
166
167  func rotate(v []mgl.Vec3) [][]mgl.Vec3 {
168
169    res := make([][]mgl.Vec3, count_rot)
170    for i := int32(0); i < count_rot; i++ {
171      anggel := float32(i) / float32(count_rot) * 2 * math.Pi
172      res[i] = make([]mgl.Vec3, len(v))
```

```
173       rotate := mgl.Rotate3DY(anggel)
174       for j, el := range v {
175
176         res[i][j] = rotate.Mul3x1(el)
177       }
178     }
179     return res
180   }
181
182   func makeVao() (uint32, uint32) {
183     var vao_top uint32
184     gl.GenVertexArrays(1, &vao_top)
185     gl.BindVertexArray(vao_top)
186
187     var vbo_top uint32
188     gl.GenBuffers(1, &vbo_top)
189     gl.BindBuffer(gl.ARRAY_BUFFER, vbo_top)
190     gl.BufferData(gl.ARRAY_BUFFER, 4*len(vertex_v), gl.Ptr(vertex_v), gl.STATIC_DRAW)
191
192     gl.VertexAttribPointer(0, 3, gl.FLOAT, false, 3*4, gl.PtrOffset(0))
193     gl.EnableVertexAttribArray(0)
194     gl.BindBuffer(gl.ARRAY_BUFFER, 0)
195
196     return vbo_top, vao_top
197   }
198
199   func add_point(res []float32, e mgl.Vec3) []float32 {
200     res = append(res, e.X()/26, e.Y()/26, e.Z()/26)
201     return res
202   }
203
204   func mess_to_vec(m [][]mgl.Vec3) []float32 {
205     var res []float32
206     for _, el := range m {
207       for i, e := range el[:len(el)-1] {
208         res = add_point(res, e)
209         res = add_point(res, el[i+1])
210       }
211     }
212     for i := range m[0] {
213       for j := 0; j < len(m); j++ {
214         res = add_point(res, m[j][i])
215         res = add_point(res, m[(j+1)%len(m)][i])
216       }
```

```
    }
    log.Print("rec")

    return res
  }

  func main() {

    dot_sliders = make([]g.Widget, 7)
    for i := range nodes {
      dot_sliders[i] = g.SliderFloat(
      fmt.Sprintf("node %d", i),
      &nodes[i][1],
      -10,
      10,
      )

    }
    wnd := g.NewMasterWindow("KP Nedosekov ©", 1000, 800, 0)
    l, r, N = calculate_N(6, 3)
    r_vec = format_r(nodes)
    mesh = rotate(to_vec3(calculate_line(l, r, r_vec)))

    vertex_v = mess_to_vec(mesh)
    program = initOpenGL()
    log.Print(vertex_v)
    _, vertex = makeVao()

    register_key_callbacks(wnd)
    wnd.Run(draw)
  }

```

compile.go

```
package main

import (
"github.com/AllenDang/giu"
m "github.com/chewxy/math32"
"github.com/go-gl/gl/v3.3-core/gl"
mgl "github.com/go-gl/mathgl/mgl32"
)

```

```
const (
width          = 700
height         = 700
aspect float32 = float32(width) / height
)

var (
vertex_v  []float32
transform = mgl.Mat4{
  1, 0, 0, 0,
  0, 1, 0, 0,
  0, 0, 1, 0,
  0, 0, 0, 1,
}

view_matr    = mgl.LookAt(0, 0, 7, 0, 0, 0, 0, 1, 0)
orto         = mgl.Ortho(-1, 1, -1*aspect, 1*aspect, 0.1, 100)
to_up        = mgl.Rotate3DX(m.Pi / 6)
to_down      = mgl.Rotate3DX(m.Pi / 6)
to_right     = mgl.Rotate3DY(-m.Pi / 6)
to_left      = mgl.Rotate3DY(m.Pi / 6)
by_clock     = mgl.Rotate3DZ(-m.Pi / 6)
by_neg_clock = mgl.Rotate3DZ(m.Pi / 6)

vertex uint32

program uint32
)

func register_key_callbacks(window *giu.MasterWindow) {
  window.RegisterKeyboardShortcuts(
  giu.WindowShortcut{
    Key:      giu.KeyA,
    Modifier: giu.ModNone,
    Callback: func() {
      transform = transform.Mul4(to_right.Mat4())
    },
  },
  )
  window.RegisterKeyboardShortcuts(
  giu.WindowShortcut{
    Key:      giu.KeyD,
    Modifier: giu.ModNone,
    Callback: func() {
```

```
      transform = transform.Mul4(to_left.Mat4())
    },
  },
  )
  window.RegisterKeyboardShortcuts(
  giu.WindowShortcut{
    Key:      giu.KeyW,
    Modifier: giu.ModNone,
    Callback: func() {
      transform = transform.Mul4(to_up.Mat4())
    },
  },
  )
  window.RegisterKeyboardShortcuts(
  giu.WindowShortcut{
    Key:      giu.KeyS,
    Modifier: giu.ModNone,
    Callback: func() {
      transform = transform.Mul4(to_down.Mat4())
    },
  },
  )
  window.RegisterKeyboardShortcuts(
  giu.WindowShortcut{
    Key:      giu.KeyQ,
    Modifier: giu.ModNone,
    Callback: func() {
      transform = transform.Mul4(by_clock.Mat4())
    },
  },
  )
  window.RegisterKeyboardShortcuts(
  giu.WindowShortcut{
    Key:      giu.KeyE,
    Modifier: giu.ModNone,
    Callback: func() {
      transform = transform.Mul4(by_neg_clock.Mat4())
    },
  },
  )
}

func count_points(m []float32) int32 {
  return int32(len(m) / 3)
```

```
98  }
99
100 func draw() {
101
102   gl.ClearColor(0.2, 0.2, 0.2, 0.2)
103   gl.Clear(gl.COLOR_BUFFER_BIT)
104   var model, view, proj int32
105   model = gl.GetUniformLocation(program, gl.Str("model\x00"))
106   view = gl.GetUniformLocation(program, gl.Str("view\x00"))
107   proj = gl.GetUniformLocation(program, gl.Str("projection\x00"))
108
109   gl.UseProgram(program)
110   gl.UniformMatrix4fv(model, 1, false, &transform[0])
111   gl.UniformMatrix4fv(view, 1, false, &view_matr[0])
112   gl.UniformMatrix4fv(proj, 1, false, &orto[0])
113
114   gl.BindVertexArray(vertex)
115
116   gl.DrawArrays(gl.LINES, 0, count_points(vertex_v))
117
118   win := giu.Window("settings")
119   win.Layout(dot_sliders[0],
120   dot_sliders[1],
121   dot_sliders[2],
122   dot_sliders[3],
123   dot_sliders[4],
124   dot_sliders[5],
125   dot_sliders[6],
126   giu.SliderInt("count_dots", &count_dots, 10, 300),
127   giu.SliderInt("count rotetes", &count_rot, 12, 360),
128   )
129   mesh = rotate(to_vec3(calculate_line(l, r, r_vec)))
130   vertex_v = mess_to_vec(mesh)
131   _, vertex = makeVao()
132
133 }
134
135 func initOpenGL() uint32 {
136
137   vertexShader, err := compileShader(vertex_shader, gl.VERTEX_SHADER)
138   if err != nil {
139     panic(err)
140   }
141   fragment1, err := compileShader(shader1, gl.FRAGMENT_SHADER)
```

```
  if err != nil {
    panic(err)
  }

  prog := gl.CreateProgram()
  gl.AttachShader(prog, vertexShader)
  gl.AttachShader(prog, fragment1)
  gl.LinkProgram(prog)
  return prog
}
```

compile.go

```
  package main

  import (
  "fmt"
  "github.com/go-gl/gl/v3.3-core/gl"
  "strings"
  )

  const (
  shader1 = `
  #version 330 core
  out vec4 color;

  void main() {
    color = vec4(1,1,1, 1.0f);
  }
  ` + "\x00"

  vertex_shader = `
  #version 330 core
  layout (location = 0) in vec3 position;

  uniform mat4 model;
  uniform mat4 view;
  uniform mat4 projection;

  void main() {
    gl_Position = projection * view * model * vec4(position, 1.0f);
  }
  ` + "\x00"
  )
```

```
32
33  func compileShader(source string, shaderType uint32) (uint32, error) {
34    shader := gl.CreateShader(shaderType)
35
36    csources, free := gl.Strs(source)
37    gl.ShaderSource(shader, 1, csources, nil)
38    free()
39    gl.CompileShader(shader)
40
41    var status int32
42    gl.GetShaderiv(shader, gl.COMPILE_STATUS, &status)
43    if status == gl.FALSE {
44      var logLength int32
45      gl.GetShaderiv(shader, gl.INFO_LOG_LENGTH, &logLength)
46
47      log := strings.Repeat("\x00", int(logLength+1))
48      gl.GetShaderInfoLog(shader, logLength, nil, gl.Str(log))
49
50      return 0, fmt.Errorf("failed to compile %v: %v", source, log)
51    }
52
53    return shader, nil
54  }
55
```

# 4 Тесты

## 1 Наборы тестов

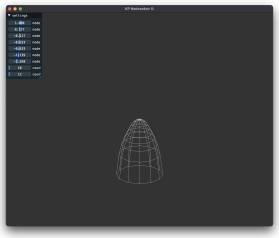

Рис. 1: 1ый тестовый набор

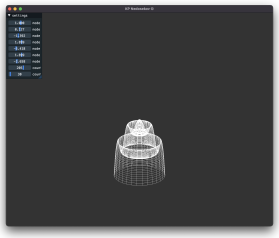

Рис. 2: 2ой тестовый набор

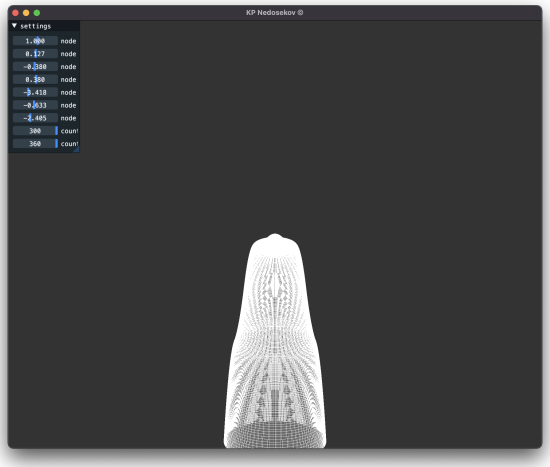


Рис. 3: 3ой тестовый набор

# 5 Выводы

Выполнив данный курсовой проект, я познакомился с OpenGl где есть более богатый встроенный инструментарий для отрисовки примитивов и кривых.

# Список литературы

[1] *Go bindings to various OpenGL*. URL: https://github.com/go-gl/gl (дата обр. 27.10.2021).

[2] *Golang — официальная документация*. URL: https://golang.org/ (дата обр. 27.10.2021).