Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №4-5 по курсу «Компьютерная графика»
Тема: Ознакомление с технологией OpenGL.

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

Москва, 2021

Содержание

1 Постановка задачи

Лабораторная работа №4-5

Создать графическое приложение с использованием OpenGL. Используя результаты Л.Р.№3, изобразить заданное тело (то же, что и в л.р. №3) с использованием средств OpenGL 2.1. Использовать буфер вершин. Точность аппроксимации тела задается пользователем. Обеспечить возможность вращения и масштабирования многогранника и удаление невидимых линий и поверхностей. Реализовать простую модель освещения на GLSL. Параметры освещения и отражающие свойства материала задаются пользователем в диалоговом режиме.

Вариант:
10. Шаровой сектор.

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 .`

3 Листинг программы

main.go

```go
package main

import (
"math"
"runtime"

"github.com/AllenDang/giu"

"github.com/go-gl/gl/v3.3-core/gl"
"github.com/go-gl/mathgl/mgl32"
)

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

var (
transform = mgl32.Mat4{
  1, 0, 0, 0,
  0, 1, 0, 0,
  0, 0, 1, 0,
  0, 0, 0, 1,
}

shaders      []uint32
view_matr            = mgl32.LookAt(0, 0, -3, 0, 0, 0, 0, 1, 0)
orto                 = mgl32.Ortho(-1, 1, -1*aspect, 1*aspect, 0.1, 100)
lightPos             = mgl32.Vec3{0, 0, -3}
lightForce   float32 = 0.3
to_up                = mgl32.Rotate3DX(math.Pi / 6)
to_down              = mgl32.Rotate3DX(math.Pi / 6)
to_right             = mgl32.Rotate3DY(-math.Pi / 6)
to_left              = mgl32.Rotate3DY(math.Pi / 6)
by_clock             = mgl32.Rotate3DZ(-math.Pi / 6)
by_neg_clock         = mgl32.Rotate3DZ(math.Pi / 6)

vertex_circles []float32
vertex_down    []float32
vertex_up      []float32
```

```
42
43  vao_down, vao_center, vao_top, program uint32
44  )
45
46  func init() {
47    runtime.LockOSThread()
48    if err := gl.Init(); err != nil {
49      panic(err)
50    }
51    calculate_points()
52  }
53
54  // initOpenGL initializes OpenGL and returns an intiialized program.
55  func initOpenGL() uint32 {
56
57    vertexShader, err := compileShader(vertex_shader, gl.VERTEX_SHADER)
58    if err != nil {
59      panic(err)
60    }
61    fragment1, err := compileShader(shader1, gl.FRAGMENT_SHADER)
62    if err != nil {
63      panic(err)
64    }
65
66    shaders = []uint32{vertexShader, fragment1 /*, fragment2*/}
67
68    prog := gl.CreateProgram()
69    gl.AttachShader(prog, vertexShader)
70    gl.AttachShader(prog, fragment1)
71    gl.LinkProgram(prog)
72    return prog
73  }
74
75  func main() {
76
77    window := giu.NewMasterWindow("lab4", width, height, 0)
78    register_key_callbacks(window)
79
80    program = initOpenGL()
81
82    _, vao_top, _, vao_center, _, vao_down = makeVao()
83
84    window.Run(draw)
85
```

```
86  }
87
88  // makeVao initializes and returns a vertex array from the points provided.
89  func makeVao() (uint32, uint32, uint32, uint32, uint32, uint32) {
90    var vao_top uint32
91    gl.GenVertexArrays(1, &vao_top)
92    gl.BindVertexArray(vao_top)
93
94    var vbo_top uint32
95    gl.GenBuffers(1, &vbo_top)
96    gl.BindBuffer(gl.ARRAY_BUFFER, vbo_top)
97    gl.BufferData(gl.ARRAY_BUFFER, 4*len(vertex_up), gl.Ptr(vertex_up),
      ↪  gl.STATIC_DRAW)
98
99    gl.VertexAttribPointer(0, 3, gl.FLOAT, false, 9*4, gl.PtrOffset(0))
100   gl.EnableVertexAttribArray(0)
101   gl.VertexAttribPointer(1, 3, gl.FLOAT, false, 9*4, gl.PtrOffset(4*3))
102   gl.VertexAttribPointer(2, 3, gl.FLOAT, false, 9*4, gl.PtrOffset(4*6))
103   gl.EnableVertexAttribArray(1)
104
105   gl.BindVertexArray(0)
106   gl.BindBuffer(gl.ARRAY_BUFFER, 0)
107
108   var vao_center uint32
109   gl.GenVertexArrays(1, &vao_center)
110   gl.BindVertexArray(vao_center)
111
112   var vbo_center uint32
113   gl.GenBuffers(1, &vbo_center)
114   gl.BindBuffer(gl.ARRAY_BUFFER, vbo_center)
115   gl.BufferData(gl.ARRAY_BUFFER, 4*len(vertex_circles), gl.Ptr(vertex_circles),
      ↪  gl.STATIC_DRAW)
116
117   gl.VertexAttribPointer(0, 3, gl.FLOAT, false, 9*4, gl.PtrOffset(0))
118   gl.EnableVertexAttribArray(0)
119   gl.VertexAttribPointer(1, 3, gl.FLOAT, false, 9*4, gl.PtrOffset(4*3))
120   gl.VertexAttribPointer(2, 3, gl.FLOAT, false, 9*4, gl.PtrOffset(4*6))
121   gl.EnableVertexAttribArray(1)
122
123   gl.BindVertexArray(0)
124   gl.BindBuffer(gl.ARRAY_BUFFER, 0)
125
126   var vao_down uint32
127   gl.GenVertexArrays(1, &vao_down)
```

```
128     gl.BindVertexArray(vao_down)
129
130     var vbo_down uint32
131     gl.GenBuffers(1, &vbo_down)
132     gl.BindBuffer(gl.ARRAY_BUFFER, vbo_down)
133     gl.BufferData(gl.ARRAY_BUFFER, 4*len(vertex_down), gl.Ptr(vertex_down),
        ↪  gl.STATIC_DRAW)
134
135     gl.VertexAttribPointer(0, 3, gl.FLOAT, false, 9*4, gl.PtrOffset(0))
136     gl.EnableVertexAttribArray(0)
137     gl.VertexAttribPointer(1, 3, gl.FLOAT, false, 9*4, gl.PtrOffset(4*3))
138     gl.VertexAttribPointer(2, 3, gl.FLOAT, false, 9*4, gl.PtrOffset(4*6))
139
140     gl.EnableVertexAttribArray(1)
141
142     gl.BindVertexArray(0)
143     gl.BindBuffer(gl.ARRAY_BUFFER, 0)
144
145     return vbo_top, vao_top, vbo_center, vao_center, vbo_down, vao_down
146   }
147
148   func count_points(m []float32) int32 {
149     return int32(len(m) / (3 * 3))
150   }
151
152   func draw() {
153
154     // rendedr_imgui()
155
156     gl.ClearColor(0.2, 0.2, 0.2, 1.0)
157     gl.Clear(gl.COLOR_BUFFER_BIT)
158     var res, model, view, proj, lpos, lf int32
159     res = gl.GetUniformLocation(program, gl.Str("resolution\x00"))
160     model = gl.GetUniformLocation(program, gl.Str("model\x00"))
161     view = gl.GetUniformLocation(program, gl.Str("view\x00"))
162     proj = gl.GetUniformLocation(program, gl.Str("projection\x00"))
163     lpos = gl.GetUniformLocation(program, gl.Str("lightPos\x00"))
164     lf = gl.GetUniformLocation(program, gl.Str("ambientStrenth\x00"))
165
166     gl.UseProgram(program)
167     gl.Uniform2f(res, width, height)
168     gl.UniformMatrix4fv(model, 1, false, &transform[0])
169     gl.UniformMatrix4fv(view, 1, false, &view_matr[0])
170     gl.UniformMatrix4fv(proj, 1, false, &orto[0])
```

```
    gl.Uniform3f(lpos, lightPos[0], lightPos[1], lightPos[2])
    gl.Uniform1f(lf, lightForce)

    gl.BindVertexArray(vao_top)

    gl.DrawArrays(gl.TRIANGLES, 0, count_points(vertex_up))

    gl.BindVertexArray(vao_down)
    gl.DrawArrays(gl.TRIANGLES, 0, count_points(vertex_down))

    gl.BindVertexArray(vao_center)
    gl.DrawArrays(gl.TRIANGLES, 0, count_points(vertex_circles))
    win := giu.Window("settings")
    win.Layout(
    giu.SliderInt("Count circles", &count_circle, 5, 150),
    giu.SliderInt("coun on circle", &ccount_on_circle, 3, 100),
    giu.SliderFloat("light", &lightForce, 0, 1),
    )
    calculate_points()
    _, vao_top, _, vao_center, _, vao_down = makeVao()

}

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())
      },
    },
    )
  }

```

vertex_calculator.go

```
package main

import (
"log"
"math"
```

```
6
7   "github.com/go-gl/mathgl/mgl32"
8   )
9
10  const (
11  R = 0.5
12  )
13
14  var (
15  red                    = mgl32.Vec3{1, 0, 0}
16  green                  = mgl32.Vec3{0, 1, 0}
17  blue                   = mgl32.Vec3{0, 0, 1}
18  dark                   = mgl32.Vec3{0, 0, 0}
19  ccount_on_circle int32 = 4
20  count_circle     int32 = 20
21  top                    = DotRGB{mgl32.Vec3{0, 0, R}, dark}
22  down                   = DotRGB{mgl32.Vec3{0, 0, -R}, red}
23  )
24
25  func r_sphere_cut_by_z(z float64) float64 {
26
27    return math.Pow(math.Pow(R, 2)-math.Pow(z, 2), 1./2)
28  }
29
30  func find_points_on_circle(tmp_r float32, count int) []mgl32.Vec2 {
31    r := mgl32.Vec2{0, tmp_r}
32    rotate := mgl32.Rotate2D(2 * math.Pi / float32(count))
33    res := make([]mgl32.Vec2, count)
34    for i := 0; i < count; i++ {
35      res[i] = r
36      r = rotate.Mul2x1(r)
37    }
38    return res
39  }
40
41  func circle_rotate(circle []mgl32.Vec2, angel float32) {
42    rotate := mgl32.Rotate2D(angel)
43    for i := 0; i < len(circle); i++ {
44      circle[i] = rotate.Mul2x1(circle[i])
45    }
46  }
47
48  type DotRGB struct {
49    d, c mgl32.Vec3
```

```
}

type Triangle struct {
  A, B, C DotRGB
}

func find_triangels(circle3D [][]DotRGB, top, down DotRGB) []Triangle {
  var res []Triangle
  count_on_circle := int(ccount_on_circle)

  for i := 0; i < count_on_circle; i++ {
    res = append(res, Triangle{down, circle3D[0][i],
    ↪  circle3D[0][(i+1)%count_on_circle]})
    }

    for i := 0; i < count_on_circle; i++ {
      res = append(res, Triangle{circle3D[len(circle3D)-1][i], top,
      ↪  circle3D[len(circle3D)-1][(i+1)%count_on_circle]})
      }

      for j := 0; j < len(circle3D)-1; j++ {
        tops := circle3D[j+1]
        downs := circle3D[j]
        var shift int
        if j%2 == 0 {
        shift = 0
      } else {
        shift = 1
      }
      for i := 0; i < count_on_circle; i++ {
        if true {
          res = append(res, Triangle{downs[i], downs[(i+1)%count_on_circle],
          ↪  tops[(i+shift)%count_on_circle]})
          } else {
            res = append(res, Triangle{downs[i], tops[(i+shift)%count_on_circle],
            ↪  downs[(i+1)%count_on_circle]})
            }
          }
          shift = (shift + 1) % 2
          for i := 0; i < count_on_circle; i++ {
            if true {
              res = append(res, Triangle{tops[(i+1)%count_on_circle],
              ↪  downs[(i+shift)%count_on_circle], tops[i]})
              } else {
```

```
            res = append(res, Triangle{tops[(i+1)%count_on_circle], tops[i],
            ↪  downs[(i+shift)%count_on_circle]})
            }
          }

        }
        return res

      }

      func normal_of_triangl(A, B, C DotRGB) (N mgl32.Vec3) {
        X := B.d.Sub(A.d)
        Y := C.d.Sub(A.d)
        N = Y.Cross(X).Normalize()
        return
      }

      func append_dot(m []float32, dots ...DotRGB) []float32 {
        for _, dot := range dots {
          m = append(m, dot.d[:]...)
          m = append(m, dot.c[:]...)
        }
        return m
      }

      func find_new_triangl(circle3D [][]DotRGB, top, down DotRGB) {
        count_circle := int(ccount_on_circle)

        ///////////////
        vertex_down = nil
        N := normal_of_triangl(down, circle3D[0][0], circle3D[0][1])

        for i := count_circle - 1; i >= 0; i-- {
          N = normal_of_triangl(down, circle3D[0][i],
          ↪  circle3D[0][(i+1)%count_circle])

          vertex_down = append_dot(vertex_down, down)
          vertex_down = append(vertex_down, N[:]...)
          vertex_down = append_dot(vertex_down, circle3D[0][i])
          vertex_down = append(vertex_down, N[:]...)
          vertex_down = append_dot(vertex_down, circle3D[0][(i+1)%count_circle])
          vertex_down = append(vertex_down, N[:]...)
        }
        vertex_down = append_dot(vertex_down, circle3D[0][0])
```

```
131             N = normal_of_triangl(down, circle3D[0][0], circle3D[0][1])
132             vertex_down = append(vertex_down, N[:]...)
133
134             vertex_up = nil
135
136             for i, _ := range circle3D[len(circle3D)-1] {
137               N = normal_of_triangl(circle3D[len(circle3D)-1][i], top,
                  ↪  circle3D[len(circle3D)-1][(i+1)%count_circle])
138               vertex_up = append_dot(vertex_up, circle3D[len(circle3D)-1][i])
139               vertex_up = append(vertex_up, N[:]...)
140               vertex_up = append_dot(vertex_up, top)
141               vertex_up = append(vertex_up, N[:]...)
142               vertex_up = append_dot(vertex_up,
                  ↪  circle3D[len(circle3D)-1][(i+1)%count_circle])
143               vertex_up = append(vertex_up, N[:]...)
144
145             }
146
147             var shift int
148             vertex_circles = make([]float32, 0)
149             for i := 0; i < len(circle3D)-1; i++ {
150
151               shift = 0
152
153               for j, _ := range circle3D[i] {
154
155                 N = normal_of_triangl(circle3D[i][j],
                    ↪  circle3D[i+1][(j+shift)%count_circle], circle3D[i][j])
156                 vertex_circles = append_dot(vertex_circles,
157                 circle3D[i][j],
158                 )
159                 vertex_circles = append(vertex_circles, N[:]...)
160                 vertex_circles = append_dot(vertex_circles,
161                 circle3D[i+1][(j+shift)%count_circle],
162                 )
163                 vertex_circles = append(vertex_circles, N[:]...)
164                 vertex_circles = append_dot(vertex_circles,
165                 circle3D[i+1][(j+shift+1)%count_circle],
166                 )
167                 vertex_circles = append(vertex_circles, N[:]...)
168
169                 N = normal_of_triangl(circle3D[i][j],
170                 circle3D[i+1][(j+shift+1)%count_circle],
171                 circle3D[i][(j+shift+1)%count_circle],
```

```
            )

            vertex_circles = append_dot(vertex_circles,
            circle3D[i][j],
            )
            vertex_circles = append(vertex_circles, N[:]...)
            vertex_circles = append_dot(vertex_circles,
            circle3D[i+1][(j+shift+1)%count_circle],
            )
            vertex_circles = append(vertex_circles, N[:]...)
            vertex_circles = append_dot(vertex_circles,
            circle3D[i][(j+shift+1)%count_circle],
            )
            vertex_circles = append(vertex_circles, N[:]...)

          }

        }

      }

      func calculate_points() []float32 {
        circles := make([][]mgl32.Vec2, count_circle)
        z := make([]float32, count_circle)
        count_on_circle := int(ccount_on_circle)
        ount_circle := int(count_circle)

        for i := 0; i < ount_circle; i++ {
          z[i] = R * float32(i) / float32(count_circle)
          tmp_r := float32(r_sphere_cut_by_z(float64(z[i])))
          log.Default().Print(R, i, count_circle)
          circles[i] = find_points_on_circle(tmp_r, count_on_circle)
        }

        circle3D := make([][]DotRGB, count_circle)
        var color mgl32.Vec3
        for i, c := range circles {
          circle3D[i] = make([]DotRGB, ccount_on_circle)
          for j, coord := range c {

            if i%2 == 1 {
            color = blue
          } else {
            color = green
```

```
            }
            circle3D[i][j] = DotRGB{mgl32.Vec3{coord.X(), coord.Y(), z[i]}, color}
          }
        }

        find_new_triangl(circle3D, top, down)
        return vertex_circles

      }
```

compile.go

```
package main

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

const (
shader1 = `
#version 330 core

uniform vec3 lightPos;

in vec4 vColor;
in vec3 vertPos;
in vec3 Normal;
in vec3 FragPos;

uniform float ambientStrenth;
out vec4 FragColor;
vec3 lightColor = vec3(1.0,1.0,1.0);

void main() {
  vec3 norm = normalize(Normal);
  vec3 lightDir = normalize(lightPos - FragPos);
  // vec3 lightDir = normalize(FragPos - lightPos);
  float diff = max(dot(norm, lightDir), 0.0);
  vec3 diffuse = diff * lightColor;

  // float ambientStrenth = 0.1f;
  vec3 ambient = ambientStrenth * lightColor;
```

```
  vec3 light = ambient + diffuse;
  FragColor = vec4(light, 1.0f) * vColor;
}` + "\x00"

vertex_shader = `
#version 330 core

uniform vec2 resolution;
uniform mat4 model;
uniform mat4 view;
uniform mat4 projection;

layout (location = 0) in vec3 aPos;
layout (location = 1) in vec3 aColor;
layout (location = 2) in vec3 normal;

out vec4 vColor;
out vec3 Normal;
out vec3 FragPos;

void main() {
  vColor = vec4(aColor, 1.0f);
  Normal = normal;
  gl_Position = projection * view * model * vec4(aPos, 1.0);
  FragPos = vec3(model * vec4(aPos, 1.0));
  // gl_Position = vec4(aPos, 1.0);
}
` + "\x00"
)

func compileShader(source string, shaderType uint32) (uint32, error) {
  shader := gl.CreateShader(shaderType)

  csources, free := gl.Strs(source)
  gl.ShaderSource(shader, 1, csources, nil)
  free()
  gl.CompileShader(shader)

  var status int32
  gl.GetShaderiv(shader, gl.COMPILE_STATUS, &status)
  if status == gl.FALSE {
    var logLength int32
    gl.GetShaderiv(shader, gl.INFO_LOG_LENGTH, &logLength)

```

```
77      log := strings.Repeat("\x00", int(logLength+1))
78      gl.GetShaderInfoLog(shader, logLength, nil, gl.Str(log))
79
80      return 0, fmt.Errorf("failed to compile %v: %v", source, log)
81    }
82
83    return shader, nil
84  }
```

4 Тесты

1 Наборы тестов

1. 20 кругов 4 вершин на круге свет 0.488
2. 20 кругов 17 вершин на круге свет 0.820
3. 150 кругов 100 вершин на круге свет 0.820

2 Визуализация тестов

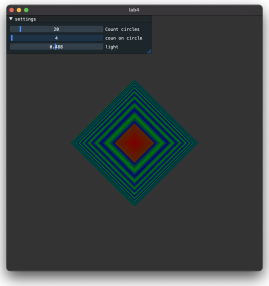

Рис. 1: 1ый тестовый набор

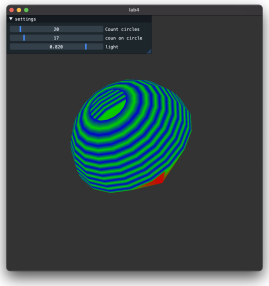

Рис. 2: 2ой тестовый набор

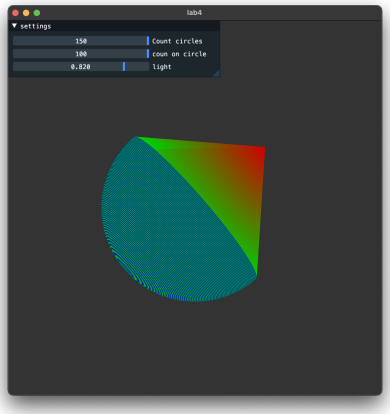

Рис. 3: 3ой тестовый набор

5 Выводы

Выполнив данную лабораторную работу, я познакомился с OpenGl где есть более богатый встроенный инструментарий для отрисовки примитивов.

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).