Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №6 по курсу «Компьютерная графика»
Тема: Создание шейдерных анимационных эффектов в OpenGL

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

Москва, 2021

Содержание

1 Постановка задачи

Лабораторная работа №6

Для поверхности, созданной в л.р. №5, обеспечить выполнение следующего шейдерного эффекта

Вариант:
8. Прозрачность вершины обратно пропорциональна расстоянию от заданной точки

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,
}

al float32 = 0.5

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
```

```
42  vertex_down    []float32
43  vertex_up      []float32
44
45  vao_down, vao_center, vao_top, program uint32
46  )
47
48  func init() {
49    runtime.LockOSThread()
50    if err := gl.Init(); err != nil {
51      panic(err)
52    }
53    calculate_points()
54  }
55
56  // initOpenGL initializes OpenGL and returns an intiialized program.
57  func initOpenGL() uint32 {
58
59    vertexShader, err := compileShader(vertex_shader, gl.VERTEX_SHADER)
60    if err != nil {
61      panic(err)
62    }
63    fragment1, err := compileShader(shader1, gl.FRAGMENT_SHADER)
64    if err != nil {
65      panic(err)
66    }
67
68    shaders = []uint32{vertexShader, fragment1}
69
70    prog := gl.CreateProgram()
71    gl.AttachShader(prog, vertexShader)
72    gl.AttachShader(prog, fragment1)
73    gl.LinkProgram(prog)
74    return prog
75  }
76
77  func main() {
78
79    window := giu.NewMasterWindow("lab6 Nedosekov", width, height, 0)
80    register_key_callbacks(window)
81
82    gl.Disable(gl.CULL_FACE)
83    gl.Enable(gl.BLEND)
84    gl.BlendFunc(1, 1)
85    program = initOpenGL()
```

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

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

```
171
172     gl.UseProgram(program)
173     gl.Uniform2f(res, width, height)
174     gl.UniformMatrix4fv(model, 1, false, &transform[0])
175     gl.UniformMatrix4fv(view, 1, false, &view_matr[0])
176     gl.UniformMatrix4fv(proj, 1, false, &orto[0])
177     gl.Uniform3f(lpos, lightPos[0], lightPos[1], lightPos[2])
178     gl.Uniform1f(lf, lightForce)
179     gl.Uniform1f(a, al)
180
181     gl.BindVertexArray(vao_top)
182
183     gl.DrawArrays(gl.TRIANGLES, 0, count_points(vertex_up))
184
185     gl.BindVertexArray(vao_down)
186     gl.DrawArrays(gl.TRIANGLES, 0, count_points(vertex_down))
187
188     gl.BindVertexArray(vao_center)
189     gl.DrawArrays(gl.TRIANGLES, 0, count_points(vertex_circles))
190     win := giu.Window("settings")
191     win.Layout(
192     giu.SliderInt("Count circles", &count_circle, 5, 150),
193     giu.SliderInt("coun on circle", &ccount_on_circle, 3, 100),
194     //giu.SliderFloat("light", &lightForce, 0, 1),
195     //giu.SliderFloat("alpha", &al, 0, 1),
196     giu.SliderFloat("pos x", &lightPos[0], -10, 10),
197     giu.SliderFloat("pos y", &lightPos[1], -10, 10),
198     giu.SliderFloat("pos z", &lightPos[2], -10, 10),
199     giu.SliderFloat("R", &R, 0, 10),
200     )
201     calculate_points()
202     _, vao_top, _, vao_center, _, vao_down = makeVao()
203
204 }
205
206 func register_key_callbacks(window *giu.MasterWindow) {
207     window.RegisterKeyboardShortcuts(
208     giu.WindowShortcut{
209       Key:      giu.KeyA,
210       Modifier: giu.ModNone,
211       Callback: func() {
212         transform = transform.Mul4(to_right.Mat4())
213       },
214     },
```

```
    )
    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"

"github.com/go-gl/mathgl/mgl32"
)

var (
R                float32 = 0.5
red                      = mgl32.Vec3{1, 0, 0}
green                    = mgl32.Vec3{0, 1, 0}
blue                     = mgl32.Vec3{0, 0, 1}
dark                     = mgl32.Vec3{0, 0, 0}
ccount_on_circle int32   = 4
count_circle     int32   = 20
top                      = DotRGB{mgl32.Vec3{0, 0, R}, dark}
down                     = DotRGB{mgl32.Vec3{0, 0, -R}, red}
)

func r_sphere_cut_by_z(z float64) float64 {

  return math.Pow(math.Pow(float64(R), 2)-math.Pow(z, 2), 1./2)
}

func find_points_on_circle(tmp_r float32, count int) []mgl32.Vec2 {
  r := mgl32.Vec2{0, tmp_r}
  rotate := mgl32.Rotate2D(2 * math.Pi / float32(count))
  res := make([]mgl32.Vec2, count)
  for i := 0; i < count; i++ {
    res[i] = r
    r = rotate.Mul2x1(r)
  }
  return res
}

```

```
38 func circle_rotate(circle []mgl32.Vec2, angel float32) {
39   rotate := mgl32.Rotate2D(angel)
40   for i := 0; i < len(circle); i++ {
41     circle[i] = rotate.Mul2x1(circle[i])
42   }
43 }
44
45 type DotRGB struct {
46   d, c mgl32.Vec3
47 }
48
49 type Triangle struct {
50   A, B, C DotRGB
51 }
52
53 func find_triangels(circle3D [][]DotRGB, top, down DotRGB) []Triangle {
54   var res []Triangle
55   count_on_circle := int(ccount_on_circle)
56
57   for i := 0; i < count_on_circle; i++ {
58     res = append(res, Triangle{down, circle3D[0][i],
     ↪  circle3D[0][(i+1)%count_on_circle]})
59     }
60
61     for i := 0; i < count_on_circle; i++ {
62       res = append(res, Triangle{circle3D[len(circle3D)-1][i], top,
       ↪  circle3D[len(circle3D)-1][(i+1)%count_on_circle]})
63       }
64
65       for j := 0; j < len(circle3D)-1; j++ {
66         tops := circle3D[j+1]
67         downs := circle3D[j]
68         var shift int
69         if j%2 == 0 {
70         shift = 0
71       } else {
72         shift = 1
73       }
74       for i := 0; i < count_on_circle; i++ {
75         if true {
76           res = append(res, Triangle{downs[i], downs[(i+1)%count_on_circle],
           ↪  tops[(i+shift)%count_on_circle]})
77           } else {
78             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])
          N = normal_of_triangl(down, circle3D[0][0], circle3D[0][1])
          vertex_down = append(vertex_down, N[:]...)

          vertex_up = nil

          for i, _ := range circle3D[len(circle3D)-1] {
            N = normal_of_triangl(circle3D[len(circle3D)-1][i], top,
            ↪  circle3D[len(circle3D)-1][(i+1)%count_circle])
            vertex_up = append_dot(vertex_up, circle3D[len(circle3D)-1][i])
            vertex_up = append(vertex_up, N[:]...)
            vertex_up = append_dot(vertex_up, top)
            vertex_up = append(vertex_up, N[:]...)
            vertex_up = append_dot(vertex_up,
            ↪  circle3D[len(circle3D)-1][(i+1)%count_circle])
            vertex_up = append(vertex_up, N[:]...)

          }

          var shift int
          vertex_circles = make([]float32, 0)
          for i := 0; i < len(circle3D)-1; i++ {

            shift = 0

            for j, _ := range circle3D[i] {

              N = normal_of_triangl(circle3D[i][j],
              ↪  circle3D[i+1][(j+shift)%count_circle], circle3D[i][j])
              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)%count_circle],
              )
              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[:]...)

                N = normal_of_triangl(circle3D[i][j],
                circle3D[i+1][(j+shift+1)%count_circle],
                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);
  float l = length(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 ambient = (1/l) * lightColor;

  vec3 light = ambient; //+ diffuse;
  FragColor = vec4(light, 1.0f) * vColor;


}` + "\x00"

// shader2 = `
// #version 330 core

// void main(){
  //   float ambientStrenth = 0.1f;
  //   vec3 ambient = ambientStrenth * lightColor;

  //   vec3 res = ambient * objectColor;
  //   color = vec4(res, 0.1f);
  // }
// ` + "\x00"

vertex_shader = `
#version 330 core

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

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, al);
  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)

    log := strings.Repeat("\x00", int(logLength+1))
    gl.GetShaderInfoLog(shader, logLength, nil, gl.Str(log))

    return 0, fmt.Errorf("failed to compile %v: %v", source, log)
  }

  return shader, nil
}
```

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

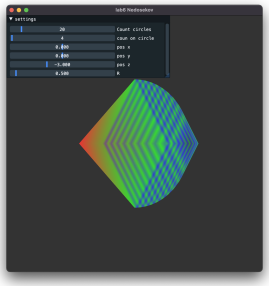

Рис. 1: 1ый тестовый набор

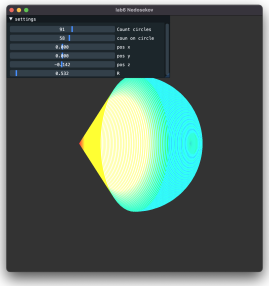

Рис. 2: 2ой тестовый набор

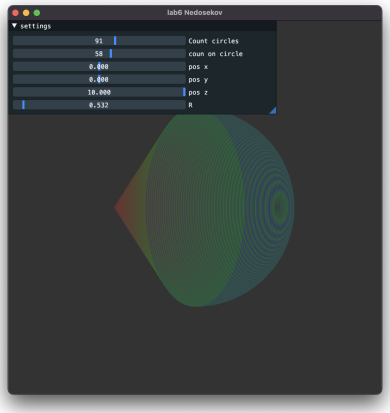

Рис. 3: 3ой тестовый набор

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).