Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №3 по курсу «Компьютерная графика»
Тема: Основы построения фотореалистичных изображений.

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

Дата:
Оценка:
Подпись:

Москва, 2021

Содержание

1 Постановка задачи **2**

2 Описание программы **3**

3 Листинг программы **4**

4 Тесты **14**

1 Наборы тестов 14

2 Визуализация тестов 14

5 Выводы **15**

1 Постановка задачи

Лабораторная работа №3

Используя результаты Л.Р.№2, аппроксимировать заданное тело выпуклым многогранником. Точность аппроксимации задается пользователем. Обеспечить возможность вращения и масштабирования многогранника и удаление невидимых линий и поверхностей. Реализовать простую модель закраски для случая одного источника света. 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

```
package main

import (
"math"
"runtime"

"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"
)

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

var (
vertex = []float32{
  -0.5, 0.0, 0.0, 1.0, 0.0, 0.0,
  0.5, 0.0, 0.0, 0.0, 1.0, 0.0,
  0.0, -0.5, 0.0, 0.0, 0.0, 1.0,
  0.0, 0.5, 0.0, 1.0, 1.0, 0.0,
  0.0, 0.0, -2.5, 1.0, 0.0, 1.0,
  0.0, 0.0, 2.5, 0.0, 1.0, 1.0,
}

indices = []uint32{
  0, 4, 2,
  0, 2, 5,
  0, 3, 4,
  0, 5, 3,
  1, 2, 4,
  1, 5, 2,
  1, 4, 3,
  1, 3, 5,
}

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, 1, 0, 0, 0, 0, 1, 0)
  orto      = mgl32.Ortho(-1, 1, -1*aspect, 1*aspect, 0.1, 100)

  to_up    = mgl32.Rotate3DY(math.Pi / 6)
  to_down  = mgl32.Rotate3DY(-math.Pi / 6)
  to_right = mgl32.Rotate3DZ(math.Pi / 6)
  to_left  = mgl32.Rotate3DY(-math.Pi / 6)
  )

  func init() {
    runtime.LockOSThread()
    if err := glfw.Init(); err != nil {
      panic(err)
    }
    if err := gl.Init(); err != nil {
      panic(err)
    }
    imgui.CreateContext(nil).SetCurrent()
    imgui.CurrentIO().SetDisplaySize(imgui.Vec2{width, height})
    imgui.CurrentIO().Fonts().Build()
    vertex = calculate_points()

  }

  // initGlfw initializes glfw and returns a Window to use.
  func initGlfw() *glfw.Window {

    glfw.WindowHint(glfw.Resizable, glfw.True)
    glfw.WindowHint(glfw.ContextVersionMajor, 4)
    glfw.WindowHint(glfw.ContextVersionMinor, 1)
    glfw.WindowHint(glfw.OpenGLProfile, glfw.OpenGLCoreProfile)
    glfw.WindowHint(glfw.OpenGLForwardCompatible, glfw.True)

    window, err := glfw.CreateWindow(width, height, "Lab 3 Nedosekov Ivan", nil, nil)
    if err != nil {
      panic(err)
    }
    window.MakeContextCurrent()
```

```
86      window.SetKeyCallback(key_callback)
87      imgui.Version()
88      imgui.CreateContext(nil)
89      imgui.StyleColorsDark()
90
91      return window
92  }
93
94  // initOpenGL initializes OpenGL and returns an intiialized program.
95  func initOpenGL() uint32 {
96
97      vertexShader, err := compileShader(vertex_shader, gl.VERTEX_SHADER)
98      if err != nil {
99          panic(err)
100     }
101     fragment1, err := compileShader(shader1, gl.FRAGMENT_SHADER)
102     if err != nil {
103         panic(err)
104     }
105     fragment2, err := compileShader(shader2, gl.FRAGMENT_SHADER)
106     if err != nil {
107         panic(err)
108     }
109
110     shaders = []uint32{vertexShader, fragment1, fragment2}
111
112     prog := gl.CreateProgram()
113     gl.AttachShader(prog, vertexShader)
114     gl.AttachShader(prog, fragment1)
115     gl.LinkProgram(prog)
116     return prog
117 }
118
119 func main() {
120     window := initGlfw()
121     defer glfw.Terminate()
122     imgui.StyleColorsDark()
123
124
125     gl.Enable(gl.CULL_FACE)
126     gl.CullFace(gl.BACK)
127
128     program := initOpenGL()
129
```

```
130     _, vao, _ := makeVao()
131
132     for !window.ShouldClose() {
133       draw(vao, window, program)
134     }
135   }
136
137   // makeVao initializes and returns a vertex array from the points provided.
138   func makeVao() (uint32, uint32, uint32) {
139     var vao uint32
140     gl.GenVertexArrays(1, &vao)
141     gl.BindVertexArray(vao)
142
143     var vbo uint32
144     gl.GenBuffers(1, &vbo)
145     gl.BindBuffer(gl.ARRAY_BUFFER, vbo)
146     gl.BufferData(gl.ARRAY_BUFFER, 4*len(vertex), gl.Ptr(vertex), gl.STATIC_DRAW)
147
148     var ebo uint32
149     gl.GenBuffers(1, &ebo)
150     gl.BindBuffer(gl.ELEMENT_ARRAY_BUFFER, ebo)
151     gl.BufferData(gl.ELEMENT_ARRAY_BUFFER, 4*len(indices), gl.Ptr(indices),
      ↪  gl.STATIC_DRAW)
152
153     gl.VertexAttribPointer(0, 3, gl.FLOAT, false, 6*4, gl.PtrOffset(0))
154     gl.EnableVertexAttribArray(0)
155     gl.VertexAttribPointer(1, 3, gl.FLOAT, false, 6*4, gl.PtrOffset(4*3))
156     gl.EnableVertexAttribArray(1)
157
158     gl.BindVertexArray(0)
159     gl.BindBuffer(gl.ARRAY_BUFFER, 0)
160     gl.BindBuffer(gl.ELEMENT_ARRAY_BUFFER, 0)
161     return vbo, vao, ebo
162   }
163
164   func draw(vao uint32, window *glfw.Window, program uint32) {
165
166     rendedr_imgui()
167     gl.ClearColor(0.5, 0.5, 0.5, 1.0)
168     gl.Clear(gl.COLOR_BUFFER_BIT)
169     var res, model, view, proj int32
170     res = gl.GetUniformLocation(program, gl.Str("resolution\x00"))
171     model = gl.GetUniformLocation(program, gl.Str("model\x00"))
172     view = gl.GetUniformLocation(program, gl.Str("view\x00"))
```

```
173     proj = gl.GetUniformLocation(program, gl.Str("projection\x00"))
174
175     gl.UseProgram(program)
176     gl.Uniform2f(res, width, height)
177     gl.UniformMatrix4fv(model, 1, false, &transform[0])
178     gl.UniformMatrix4fv(view, 1, false, &view_matr[0])
179     gl.UniformMatrix4fv(proj, 1, false, &orto[0])
180     gl.BindVertexArray(vao)
181
182
183     gl.DrawArrays(gl.TRIANGLES, 0, int32(len(vertex)))
184
185
186     glfw.PollEvents()
187     window.SwapBuffers()
188   }
189
190   func rendedr_imgui() {
191     imgui.NewFrame()
192     imgui.BeginChild("sl")
193     imgui.SliderInt("Круги", &count_circle, 3, 100)
194     imgui.EndChild()
195     imgui.Render()
196
197   }
198
199   func key_callback(w *glfw.Window, key glfw.Key, scancode int, action glfw.Action,
      ↪  mods glfw.ModifierKey) {
200     rotate := mgl32.Rotate3DX(0)
201     switch key {
202       case glfw.KeyW:
203       rotate = to_up
204       case glfw.KeyS:
205       rotate = to_down
206       case glfw.KeyD:
207       rotate = to_right
208       case glfw.KeyA:
209       rotate = to_left
210       default:
211       rotate = mgl32.Rotate3DX(0)
212     }
213     transform = rotate.Mat4().Mul4(transform)
214
215   }
```

```

```

vertex_calculator.go

```go
package main

import (
"log"
"math"

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

const (
R = 0.5
)

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

func r_sphere_cut_by_z(z float64) float64 {

  return math.Pow(math.Pow(R, 3)-math.Pow(z, 3), 1./3)
}

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
}

```

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

```
            }

        }
        return res

    }

    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)
        }
        log.Default().Print("z", z, "aaaaa", len(z))

        angel := (2 * math.Pi / float32(ccount_on_circle)) / 2
        for i := 2; i < ount_circle; i += 2 {
            circle_rotate(circles[i], angel)
        }
        log.Default().Print(circles[0])
        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}
        }
    }

    triang := find_triangels(circle3D, top, down)
    var res []float32
    for i, tr := range triang {
```

```
        log.Default().Print("Triangle", i, tr)
        res = append(res, tr.A.d[:]...)
        res = append(res, tr.A.c[:]...)

        res = append(res, tr.B.d[:]...)
        res = append(res, tr.B.c[:]...)

        res = append(res, tr.C.d[:]...)
        res = append(res, tr.C.c[:]...)

      }
      log.Default().Print(res[0:19])
      return res
    }
```

compile.go

```
package main

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

const (
shader1 = `
#version 330 core

in vec4 vColor;
in vec3 vertPos;
out vec4 FragColor;

void main() {
  FragColor = vColor;
}` + "\x00"


vertex_shader = `
#version 330 core

uniform vec2 resolution;
uniform mat4 model;
uniform mat4 view;
```

```
28  uniform mat4 projection;
29
30  layout (location = 0) in vec3 aPos;
31  layout (location = 1) in vec3 aColor;
32
33  out vec4 vColor;
34
35  void main() {
36    vColor = vec4(aColor, 1.0f);
37    gl_Position = projection * view * model * vec4(aPos, 1.0);
38    // gl_Position = vec4(aPos, 1.0);
39  }
40  ` + "\x00"
41  )
42
43  func compileShader(source string, shaderType uint32) (uint32, error) {
44    shader := gl.CreateShader(shaderType)
45
46    csources, free := gl.Strs(source)
47    gl.ShaderSource(shader, 1, csources, nil)
48    free()
49    gl.CompileShader(shader)
50
51    var status int32
52    gl.GetShaderiv(shader, gl.COMPILE_STATUS, &status)
53    if status == gl.FALSE {
54      var logLength int32
55      gl.GetShaderiv(shader, gl.INFO_LOG_LENGTH, &logLength)
56
57      log := strings.Repeat("\x00", int(logLength+1))
58      gl.GetShaderInfoLog(shader, logLength, nil, gl.Str(log))
59
60      return 0, fmt.Errorf("failed to compile %v: %v", source, log)
61    }
62
63    return shader, nil
64  }
65
66
```

4 Тесты

1 Наборы тестов

1. 10 кругов 10 вершин на круге
2. 10 кругов 4 вершин на круге

2 Визуализация тестов

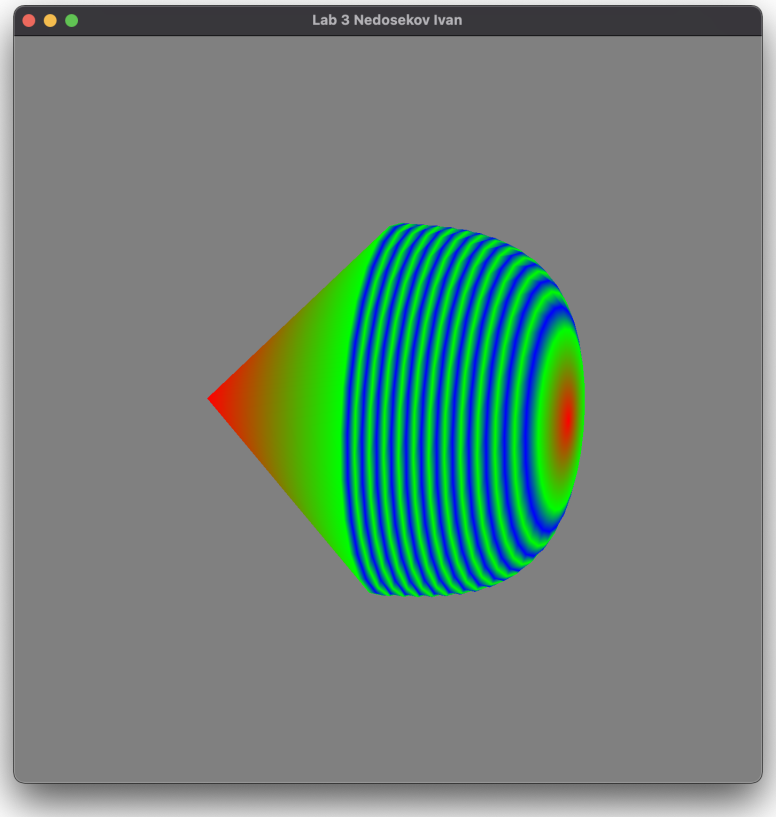

Рис. 1: 1ый тестовый набор

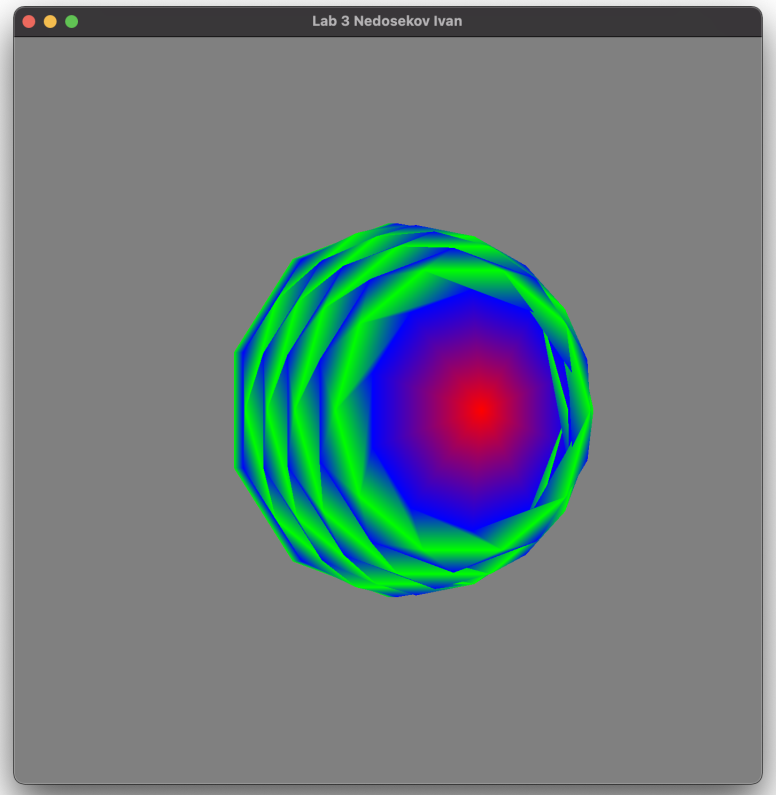

Рис. 2: 2ой тестовый набор

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).