Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Московский государственный технический университет
имени Н.Э. Баумана»
(МГТУ им. Н.Э. Баумана)

Факультет: Информатика и системы управления
Кафедра: Теоретическая информатика и компьютерные технологии

Лабораторная работа №5
«Алгоритмы отсечения»
по курсу: «Алгоритмы компьютерной графики»

Выполнил:
Студент группы ИУ9-41Б
Гречко Г.В.

Проверил:
Цалкович П.А.

Москва, 2023

Цели

Получение навыков реализации алгоритмов отсечения для динамически вводимых данных

Задачи

1. Реализовать один из алгоритмов отсечения определенного типа в пространстве заданной размерности (в соответствии с вариантом).
   - Алгоритм внутреннего отсечения Коэна-Сазерленда для трехмерного пространства
2. Ввод исходных данных каждого из алгоритмов производится интерактивно с помощью клавиатуры и/или мыши.

Основная теория

Алгоритмом отсечения (отсечением) называется любая процедура, которая удаляет те точки изображения, которые находятся внутри (или снаружи) заданной области пространства.

Классификация алгоритмов отсечения

- по типу обрабатываемых объектов:
  - отсечение точки;
  - отсечение линии (отрезка);
  - отсечение области (многоугольника);
  - отсечение кривой;
  - отсечение текста (литер);
- по размерности:
  - двумерное отсечение;
  - трехмерное отсечение;
- по расположению результата отсечения относительно отсекателя:
  - внутреннее отсечение;
  - внешнее отсечение.

Алгоритмы отсечения отрезков:

- простой алгоритм отсечения
- алгоритм Коэна-Сазерленда (двумерного отсечения регулярной прямоугольной областью)
- алгоритм разбиения средней точкой
- алгоритм Кируса-Бека (двумерного параметрического отсечения выпуклым многоугольником)
- отсечение отрезка невыпуклым окном
- обобщение алгоритмов КоэнаСазерленда и Кируса-Бека для трехмерного случая

Отсечение многоугольников:

- алгоритм Сазерленда-Ходжмена (последовательного отсечения произвольного многоугольника выпуклым отсекателем)
- алгоритм Вейлера-Азертона (отсечения многоугольника произвольным отсекателем)
- модификации алгоритма Вейлера-Азертона для реализации булевских операций над многоугольниками.

Практическая реализация

main.go

```
1 package main
2
3 import (
```

```
    "fmt"
    "log"

    "github.com/go-gl/gl/v2.1/gl"
    "github.com/go-gl/glfw/v3.2/glfw"
)

type point struct {
    x float32
    y float32
    z float32
}

type Area struct {
    min point // Левый верхний угол ближней грани
    max point // Правый нижкий угол дальней грани
}

var area = Area{
    min: point{
        -0.5, -0.5, -0.5,
    },
    max: point{
        0.5, 0.5, 0.5,
    },
}

var (
    window_width  = 1000
    window_height = 1000
)

const (
    BOTTOM = 1 << iota
    LEFT   = 1 << iota
    TOP    = 1 << iota
    RIGHT  = 1 << iota
    BACK   = 1 << iota
    FRONT  = 1 << iota

    scale = 0.2
)

var (
    verticies = [][]float32{
        {0.5, -0.5, -0.5},
        {0.5, 0.5, -0.5},
        {-0.5, 0.5, -0.5},
        {-0.5, -0.5, -0.5},
        {0.5, -0.5, 0.5},
        {0.5, 0.5, 0.5},
        {-0.5, -0.5, 0.5},
        {-0.5, 0.5, 0.5},
    }

    edges = [][]int{
        {0, 1},
        {0, 3},
        {0, 4},
        {2, 1},
        {2, 3},
        {2, 7},
```

```
        {6, 3},
        {6, 4},
        {6, 7},
        {5, 1},
        {5, 4},
        {5, 7},
    }

    to_cut = [][]float32{
        {1, 0.2, 1},
        {-0.1, 0, -1},
    }

    // to_cut = [][]float32{
    //  {0.5, 0.5, 1},
    //  {0.5, 0.5, -1},
    // }

    to_cut1 = [][]float32{
        {1, 0.2, 1},
        {-0.1, 0, -1},
    }

    angle1, angle2, angle3 = 10, 10, 10
)

func getCode(p point) int {
    code := 0

    if p.y > area.max.y {
        code |= TOP
    } else if p.y < area.min.y {
        code |= BOTTOM
    }

    if p.x > area.max.x {
        code |= RIGHT
    } else if p.x < area.min.x {
        code |= LEFT
    }

    if p.z > area.max.z {
        code |= FRONT
    } else if p.z < area.min.z {
        code |= BACK
    }

    return code
}

func CS_Clip() {
    a := to_cut[0]
    b := to_cut[1]
    x1 := a[0]
    x2 := b[0]
    y1 := a[1]
    y2 := b[1]
    z1 := a[2]
    z2 := b[2]
    code1 := getCode(point{a[0], a[1], a[2]})
    code2 := getCode(point{b[0], b[1], b[2]})
    accept := false
```

```
    for {
        code_out := 0
        if code1 == 0 && code2 == 0 {
            accept = true
            break
        } else if (code1 & code2) != 0 {
            break
        } else {
            x := float32(1.0)
            y := float32(1.0)
            z := float32(1.0)
            if code1 != 0 {
                code_out = code1
            } else {
                code_out = code2
            }
            if code_out&TOP != 0 {
                x = x1 + (x2-x1)*(area.max.y-y1)/(y2-y1)
                z = z1 + (z2-z1)*(area.max.y-y1)/(y2-y1)
                y = area.max.y
            } else if code_out&BOTTOM != 0 {
                x = x1 + (x2-x1)*(area.min.y-y1)/(y2-y1)
                z = z1 + (z2-z1)*(area.min.y-y1)/(y2-y1)
                y = area.min.y
            } else if code_out&RIGHT != 0 {
                y = y1 + (y2-y1)*(area.max.x-x1)/(x2-x1)
                z = z1 + (z2-z1)*(area.max.x-x1)/(x2-x1)
                x = area.max.x
            } else if code_out&LEFT != 0 {
                y = y1 + (y2-y1)*(area.min.x-x1)/(x2-x1)
                z = z1 + (z2-z1)*(area.min.x-x1)/(x2-x1)
                x = area.min.x
            } else if code_out&FRONT != 0 {
                x = x1 + (x2-x1)*(area.max.z-z1)/(z2-z1)
                y = y1 + (y2-y1)*(area.max.z-z1)/(z2-z1)
                z = area.max.z
            } else if code_out&BACK != 0 {
                x = x1 + (x2-x1)*(area.min.z-z1)/(z2-z1)
                y = y1 + (y2-y1)*(area.min.z-z1)/(z2-z1)
                z = area.min.z
            }
            if code_out == code1 {
                x1 = x
                y1 = y
                z1 = z
                code1 = getCode(point{x1, y1, z1})
            } else {
                x2 = x
                y2 = y
                z2 = z
                code2 = getCode(point{x2, y2, z2})
            }
        }
    }
    if accept {
        to_cut1[0][0] = x1
        to_cut1[0][1] = y1
        to_cut1[0][2] = z1
        to_cut1[1][0] = x2
        to_cut1[1][1] = y2
        to_cut1[1][2] = z2
    }
```

```
}

func DrawCube() {
    gl.PushMatrix()

    gl.Scalef(scale, scale, scale)
    gl.Rotatef(float32(angle1), 0, 0, 1)
    gl.Rotatef(float32(angle2), 0, 1, 0)
    gl.Rotatef(float32(angle3), 1, 0, 0)
    fmt.Println(to_cut, to_cut1)
    //[[1 0.2 1] [-0.1 0 -1]] [[0.5 0.10909091 0.09090912] [0.17499998
     ↪  0.05 -0.5]]

    gl.Color3f(1.0, 0.5, 0.0)
    gl.Begin(gl.LINES)
    for _, edge := range edges {
        for _, vertex := range edge {
            gl.Vertex3fv(&verticies[vertex][0])
        }
    }
    gl.End()

    gl.Color3f(0.0, 1.0, 0.0)
    gl.Begin(gl.LINES)
    for _, p := range to_cut {
        gl.Vertex3fv(&p[0])
    }
    gl.End()

    gl.Color3f(0.0, 0.0, 1.0)
    gl.Begin(gl.LINES)
    gl.Vertex3f(to_cut[0][0], to_cut[0][1], to_cut[0][2])
    gl.Vertex3f(to_cut1[0][0], to_cut1[0][1], to_cut1[0][2])
    gl.Vertex3f(to_cut[1][0], to_cut[1][1], to_cut[1][2])
    gl.Vertex3f(to_cut1[1][0], to_cut1[1][1], to_cut1[1][2])
    gl.End()

    gl.PopMatrix()
}

func display(w *glfw.Window) {
    gl.Clear(gl.COLOR_BUFFER_BIT)
    gl.ClearColor(0.0, 0.0, 0.0, 0.0)

    CS_Clip()
    DrawCube()
}

func keyCallback(w *glfw.Window, key glfw.Key, scancode int, action
 ↪  glfw.Action, mods glfw.ModifierKey) {
    if action == glfw.Press {
        switch key {
        case glfw.KeyEscape:
            w.SetShouldClose(true)
        case glfw.KeyA:
            angle1 -= 10
        case glfw.KeyD:
            angle1 += 10
        case glfw.KeyW:
            angle2 -= 10
        case glfw.KeyS:
            angle2 += 10
```

```
250             case glfw.KeyQ:
251                 angle3 -= 10
252             case glfw.KeyE:
253                 angle3 += 10
254             }
255     }
256 }
257 
258 func main() {
259     if err := glfw.Init(); err != nil {
260         log.Fatal("failed to initialize glfw:", err)
261     }
262 
263     window, err := glfw.CreateWindow(window_width, window_height, "Lab
    ↪ 5", nil, nil)
264 
265     if err != nil {
266         glfw.Terminate()
267         log.Fatal("failed to create glfw window:", err)
268     }
269 
270     if err := gl.Init(); err != nil {
271         log.Fatal("failed to initialize gl:", err)
272     }
273 
274     window.MakeContextCurrent()
275     glfw.SwapInterval(1)
276     window.SetKeyCallback(keyCallback)
277 
278     for !window.ShouldClose() {
279         display(window)
280 
281         window.SwapBuffers()
282         glfw.PollEvents()
283     }
284 
285     window.Destroy()
286     glfw.Terminate()
287 }
```

Заключение

В ходе лабораторной работы был реализован алгоритм отсечения отрезка, который был протестирован на различных входных данных. Так же был реализован трехмерный просмотр результата работы алгоритма с возможностью смещения точки обзора для возможности удостовериться в корректной работе с разных ракурсов.