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 №4
«Алгоритмы растровой графики»
по курсу: «Алгоритмы компьютерной графики»

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

Проверил:
Цалкович П.А.

Москва, 2023

Цели

Получение навыков работы с растровой графикой и алгоритмами фильтрации на примере OpenGL.

Задачи

1. Реализовать алгоритм растровой развертки многоугольника
   - построчное сканирование многоугольника с упорядоченным списком ребер
2. Реализовать алгоритм фильтрации
   - целочисленный алгоритм Брезенхема с устранением ступенчатости
3. Реализовать необходимые вспомогательные алгоритмы (растеризации отрезка) с модификациями, обеспечивающими корректную работу основного алгоритма
4. Ввод исходных данных каждого из алгоритмов производится интерактивно с помощью клавиатуры и/или мыши. Предусмотреть также возможность очистки области вывода (отмены ввода).
5. Растеризацию производить в специально выделенном для этого буфере в памяти с последующим копированием результата в буфер кадра OpenGL. Предусмотреть возможность изменения размеров окна.

Основная теория

Растровые графические системы

- **принцип записи изображения:** построчное сканирование луча
- **примитив:** точка (пиксель, pixel = picture element)
- **необходима процедура растеризации геометрических примитивов (растровая развертка примитивов):**
  - растровая развертка в реальном времени
  - групповое кодирование (интенсивность + длина участка)
  - клеточное кодирование (шаблоны, например, кодирование литер в алфавитно-цифровом терминале)
  - использование буфера кадра (обеспечивает промежуточное хранение изображения - растеризованных графических примитивов).

Алгоритмы развертки растровых кривых

Основные требования к таким алгоритмам: - совпадение начальной и конечной точки с заданными - соблюдение формы кривой - постоянная яркость вдоль кривой (для отрезка: независимо от длины и наклона) - высокая производительность.

Примеры алгоритмом растровой развертки: - Цифровой дифференциальный анализатор - Вещественный алгоритм Брезенхема - Целочисленный алгоритм Брезенхема - Обобщение алгоритма Брезенхема для произвольных октантов

Ступенчатость

Типы искажений, связанных с ошибкой дискретизации при разложении в растр: - ступенчатость ребер, границ, кривых - некорректная визуализация тонких деталей и фактур - визуализация мелких объектов (проблема «мерцания» при анимации)

Методы устранения ступенчатости

- увеличение частоты дискретизации
- определение интенсивности пиксела путем вычисления площади его перекрывания с изображаемым объектом

Фильтрация

Это свертка (convolution) сигнала (изображения) с ядром свертки (функцией фильтра) – усреднение сигнала в некоторой области

Примеры фильтров

- размытие (простейшее усреднение, константное, гауссово)
- повышение четкости
- нахождение границ
- тиснение
- медианный фильтр

Постфильтрация

Это усреднение характеристик пикселя, бывает:

- равномерное
- взвешенное

Основные методы работы с пикселями в `OpenGL`

- определение режимов чтения/записи пикселей при передаче между буфером кадра и программным буфером:

```
glPixelStore (GLenum pname, GLtype param)
```

- определение режимов преобразования пикселей при передаче между буфером кадра и программным буфером:

```
glPixelTransfer (GLenum pname, GLtype param)
```

- задание текущей позиции в растре (подвергается преобразованиям):

```
glRasterPos[2 3 4][s i f d] [v] () //GL_CURRENT_RASTER_POSITION_VALID
```

- определение коэффициента масштабирования для пиксельных операций:

```
glPixelZoom (GLfloat xfactor, GLfloat yfactor)
```

- определение буфера кадра для операций над пикселями:

```
glReadBuffer( GLenum mode ) – для чтения
glDrawBuffer( GLenum mode ) – для записи
```

// GL_NONE, GL_FRONT_LEFT, GL_FRONT_RIGHT, GL_BACK_LEFT, GL_BACK_RIGHT, GL_FRONT, GL_BACK, GL_LEFT, GL_RIGHT, GL_FRONT_AND_BACK, GL_AUXi - операции над пикселями: - чтение из буфера кадра:

```
    glReadPixels( GLint x, GLint y, GLsizei width, GLsizei height, GLenum
↪   format, GLenum type, GLvoid *pixels )
```

```
- запись в буфер кадра:
```

```
    glDrawPixels( GLsizei width, GLsizei height, GLenum format, GLenum
↪   type, const GLvoid *pixels )
```

```
- копирование в текущую позицию растра:
```

```
    glCopyPixels( GLint x, GLint y, GLsizei width, GLsizei height, GLenum
↪   type)
```

Практическая реализация

main.go

```
1  package main
2
3  import (
4      "fmt"
5      "math"
6      "sort"
7      "unsafe"
8
9      "github.com/go-gl/gl/v2.1/gl"
10     "github.com/go-gl/glfw/v3.2/glfw"
11     "go.uber.org/zap"
```

```
)

type point struct {
    x float64
    y float64
}

var (
    window_width  = 1280
    window_height = 720
    is_displayed  = false
    smoothing     = true
)

var logger *zap.Logger

var (
    points []point
    pixels []uint8
    edges  [][2]point
    list   map[int][]int
)

func isExtrema(y, y1, y2 float64) bool {
    return (y > y1 && y > y2) || (y < y1 && y < y2)
}

func vertexCountTwice(i, j int) bool {
    l := len(edges)
    return isExtrema(
        edges[i][j].y,
        edges[i][(j+1)%2].y,
        edges[(i-1+l)%l][j].y)
}

func addToList(x, y float64) {
    list[int(math.Floor(y))] = append(list[int(math.Floor(y))],
    ↪ int(math.Floor(x)))
}

func drawLine(y, x1, x2 int) {
    for i := x1 + 1; i <= x2; i++ {
        pos := (i + (window_height-y)*window_width)
        pixels[pos] = 255
    }
}

func OrderedEdgesList() {
    for i, edge := range edges {
        if vertexCountTwice(i, 0) {
            addToList(edge[0].x, edge[0].y)
        }
        addToList(edge[1].x, edge[1].y)

        dy := edge[1].y - edge[0].y
        dx := edge[1].x - edge[0].x

        if dy == 0 {
            continue
        }

        count := int(math.Ceil(math.Abs(dy)))
```

```
        dy = dy / float64(count)
        dx = dx / float64(count)

        checkEndFunc := func(i float64) bool {
            if dy > 0 {
                return edge[0].y+i*dy < edge[1].y
            } else {
                return edge[0].y+i*dy > edge[1].y
            }
        }

        for i := float64(1); checkEndFunc(i); i++ {
            addToList(edge[0].x+i*dx, edge[0].y+i*dy)
        }
    }
}

func ProcessLines() {
    for y := range list {
        sort.Ints(list[y])
        for i := 0; i < len(list[y]); i += 2 {
            drawLine(y, list[y][i], list[y][i+1])
        }
    }
}

func brezenhem0(p1, p2 point) {
    I := 255

    dx := p2.x - p1.x
    dy := p2.y - p1.y

    x := p2.x
    y := p2.y

    swap := 0

    m := dy / dx

    sx := -1
    if dx < 0 {
        sx = 1
        dx *= -1
    }

    sy := -1
    if dy < 0 {
        sy = 1
        dy *= -1
    }

    if m > 1 {
        dx, dy = dy, dx

        m = 1 / m
        swap = 1
    }

    e := float64(I) / float64(2)

    m = m * float64(I)
    w := float64(I) - m
```

```
135
136     for i := float64(1); i <= dx+1; i++ {
137         if e <= w {
138             if swap == 0 {
139                 x += float64(sx)
140             } else {
141                 y += float64(sy)
142             }
143             e += m
144         } else {
145             y += float64(sy)
146             x += float64(sx)
147             e -= w
148             color := 255 - e
149             // fmt.Println(color, x, y)
150             pos := (int(x) + (window_height-int(y))*window_width)
151             pixels[pos] = uint8(math.Floor(color))
152         }
153     }
154
155 }
156
157 func getPos(x, y float64) int {
158     return int(x) + (window_height-int(y))*window_width
159 }
160
161 func brezenhem(p1, p2 point) {
162     I := float64(255)
163     x := p1.x
164     y := p1.y
165
166     dx := p2.x - p1.x
167     dy := p2.y - p1.y
168
169     m := I * (dy / dx)
170
171     w := I - m
172
173     e := m / 2
174
175     pos := getPos(x, y)
176
177     pixels[pos] = uint8(m / 2)
178
179     for x < p2.x {
180         x++
181         if e >= w {
182             y++
183             e -= w
184         } else {
185             e += m
186         }
187         pos := getPos(x, y)
188         pixels[pos] = uint8(e)
189     }
190 }
191
192 func filtrate() {
193     for _, edge := range edges {
194         fmt.Println(edge)
195         brezenhem0(edge[0], edge[1])
196     }
```

```
}

func calcEdges() {
    edges = make([][2]point, len(points))
    for i, p := range points {
        nextP := points[(i+1)%len(points)]
        edges[i] = [2]point{p, nextP}
    }
}

func calcPixels() {
    pixels = make([]uint8, window_width*window_height)
    list = make(map[int][]int)

    calcEdges()
    OrderedEdgesList()
    ProcessLines()
    if smoothing {
        filtrate()
    }
}

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.KeyD:
            if len(points) > 2 {
                is_displayed = true

                calcPixels()

                logger.Info("drawing pixels...")
            } else {
                logger.Error("can't draw pixels, array is empty")
            }
        case glfw.KeyC:
            is_displayed = false
            points = []point{}
            pixels = []uint8{}
            edges = [][2]point{}
            list = nil

            logger.Info("cleared pixels buffer")
        case glfw.KeyS:
            smoothing = !smoothing

            logger.Info("toggled smooting", zap.Bool("new value",
 ↪  smoothing))
        }
        if key == glfw.KeyEscape {
            w.SetShouldClose(true)
        }
    }
}

func mouseCallback(w *glfw.Window, button glfw.MouseButton, action
 ↪  glfw.Action, mod glfw.ModifierKey) {
    if action == glfw.Press && button == glfw.MouseButtonLeft {
        x, y := w.GetCursorPos()
```

```
256         points = append(points, point{x, y})
257 
258         logger.Info("Mouse click: ", zap.Float64("x", x),
    ↪  zap.Float64("y", y))
259     }
260 }
261 
262 func sizeCallback(w *glfw.Window, width, height int) {
263     window_width, window_height = width, height
264     gl.Viewport(0, 0, int32(width), int32(height))
265 
266     points = []point{}
267     pixels = []uint8{}
268     is_displayed = false
269 }
270 
271 func display(w *glfw.Window) {
272     gl.Clear(gl.COLOR_BUFFER_BIT)
273     // gl.ClearColor(0.0, 0.9, 0.9, 1.0)
274 
275     if is_displayed {
276         gl.DrawPixels(int32(window_width), int32(window_height),
277             gl.RED, gl.UNSIGNED_BYTE,
278             unsafe.Pointer(&pixels[0]),
279         )
280     }
281 }
282 
283 func main() {
284     logger, _ = zap.NewDevelopment()
285 
286     // points = append(points, point{87, 104}, point{190, 624},
     ↪  point{1148, 441})
287 
288     if err := glfw.Init(); err != nil {
289         logger.Fatal("failed to initialize glfw:", zap.Error(err))
290     }
291 
292     window, err := glfw.CreateWindow(window_width, window_height, "Lab
    ↪  4", nil, nil)
293 
294     if err != nil {
295         glfw.Terminate()
296         logger.Fatal("failed to create glfw window:", zap.Error(err))
297     }
298 
299     if err := gl.Init(); err != nil {
300         logger.Fatal("failed to initialize gl:", zap.Error(err))
301     }
302 
303     window.MakeContextCurrent()
304     glfw.SwapInterval(1)
305     window.SetKeyCallback(keyCallback)
306     window.SetMouseButtonCallback(mouseCallback)
307     window.SetFramebufferSizeCallback(sizeCallback)
308 
309     w, h := window.GetFramebufferSize()
310     sizeCallback(window, w, h)
311 
312     for !window.ShouldClose() {
313         display(window)
314 
```

```
315             window.SwapBuffers()
316             glfw.PollEvents()
317         }
318
319         window.Destroy()
320         glfw.Terminate()
321     }
322
```

Заключение

В ходе лабораторной работы были реализованы алгоритмы для сканирования фигуры и ее растеризации, а так же реализованы алгоритм фильтрации Брезенхема. Все алгоритмы были реализованы в виде модульных компонентов.

Так же были изучены функции OpenGL для динамического ввода данных и изменения размеров окна без перезапуска программы.