ГУАП

КАФЕДРА №14

ОТЧЕТ

ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

Должность, уч. степень, звание подпись, дата инициалы, фамилия

**ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №1**

по курсу: КОМПЬЮТЕРНАЯ ГРАФИКА

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. 1441 М.И. Лубинец

подпись, дата инициалы, фамилия

Санкт-Петербург

2015

1. Формализация задачи

Используя алгоритм Брезенхама и встроенную функцию рендерера, нарисовать 2 линии соответственно.

С помощью матриц, реализовать афинные преобразования над этими линиями.

2. Листинги

Файл main.rs

const WIDTH: u32 = 1280;

const HEIGHT: u32 = 720;

fn main() {

// Set initial lines coordinates

let startx = ((WIDTH/2) - 10) as f32;

let starty1 = (HEIGHT/4) as f32;

let starty2 = (HEIGHT/4\*3) as f32;

// Create lines

let mut line1 = line::Line::new(

Point2D::new(startx - 10.0, starty1),

Point2D::new(startx - 10.0, starty2),

Color::RGB(0, 255, 0)

);

let mut line2 = line::Line::new(

Point2D::new(startx + 10.0, starty1),

Point2D::new(startx + 10.0, starty2),

Color::RGB(255, 0, 0)

);

// Start main loop

'main:

loop {

// Initialize variables

let mut dx = 0.0;

let mut dy = 0.0;

let mut scale = 1.0;

let mut angle = 0.0;

// Poll presed keys

for key in events.keyboard\_state().pressed\_scancodes() {

match key {

W => dy -= 3.0,

S => dy += 3.0,

D => dx += 3.0,

A => dx -= 3.0,

Up => scale += 0.05,

Down => scale -= 0.05,

Left => angle -= 3.0,

Right => angle += 3.0,

\_ => (),

}

}

// Do affine transformations

line1.translate(dx, dy);

line1.scale(scale, scale);

line1.rotate(angle);

line2.translate(dx, dy);

line2.scale(scale, scale);

line2.rotate(angle);

// Clear render buffer

renderer.set\_draw\_color(Color::RGB(0, 0, 0));

renderer.clear();

// Draw lines

line1.draw(&renderer);

line2.draw\_builtin\_line(&renderer);

// Present render buffer

renderer.present();

}

}

Файл primitives.rs

pub struct Point2D {

pub x: f32,

pub y: f32,

}

impl Point2D {

pub fn new(x: f32, y: f32) -> Point2D {

Point2D {

x: x,

y: y,

}

}

}

pub trait Primitive2D {

fn to\_matrix(&self) -> Matrix;

fn from\_matrix(&mut self, m: &Matrix);

fn draw(&self, renderer: &Renderer);

fn translate(&mut self, dx: f32, dy: f32) {

let obj = self.to\_matrix();

let (x, y) = (obj.matrix[2][0], obj.matrix[2][1]);

self.from\_matrix(

&(obj \*

translation\_matrix(-x, -y) \*

translation\_matrix(dx, dy) \*

translation\_matrix(x, y))

);

}

fn scale(&mut self, sx: f32, sy: f32) {

let obj = self.to\_matrix();

let (x, y) = (obj.matrix[2][0], obj.matrix[2][1]);

self.from\_matrix(

&(obj \*

translation\_matrix(-x, -y) \*

scale\_matrix(sx, sy) \*

translation\_matrix(x, y))

);

}

fn rotate(&mut self, angle: f32) {

let obj = self.to\_matrix();

let (x, y) = (obj.matrix[2][0], obj.matrix[2][1]);

self.from\_matrix(

&(obj \*

translation\_matrix(-x, -y) \*

rotation\_matrix(angle) \*

translation\_matrix(x, y))

);

}

}

Файл line.rs

pub struct Line {

p1: Point2D,

p2: Point2D,

average: Point2D,

color: Color

}

impl Line {

pub fn new(point1: Point2D, point2: Point2D, color: Color) -> Line {

let average = Point2D::new(

(point2.x + point1.x) / 2.0,

(point2.y + point1.y) / 2.0,

);

Line { p1: point1, p2: point2, average: average, color: color }

}

fn draw\_bresenham\_line(&self, renderer: &Renderer) {

let mut x1 = self.p1.x as i16;

let mut y1 = self.p1.y as i16;

let mut x2 = self.p2.x as i16;

let mut y2 = self.p2.y as i16;

let steep = (y2 - y1).abs() > (x2 - x1).abs();

if steep {

swap(&mut x1, &mut y1);

swap(&mut x2, &mut y2);

}

if x1 > x2 {

swap(&mut x1, &mut x2);

swap(&mut y1, &mut y2);

}

let dx = x2 - x1;

let dy = (y2 - y1).abs();

let mut error = dx / 2;

let mut y = y1;

let ystep = if y1 < y2 { 1 } else { -1 };

for x in x1..x2 {

renderer.pixel(

if steep { y } else { x },

if steep { x } else { y },

self.color

).unwrap();

error -= dy;

if error < 0 {

y += ystep;

error += dx;

}

}

}

pub fn draw\_builtin\_line(&self, renderer: &Renderer) { /\* ... \*/ }

}

impl Primitive2D for Line {

fn to\_matrix(&self) -> Matrix {

matrix!( [[self.p1.x, self.p1.y, 1.0],

[self.p2.x, self.p2.y, 1.0],

[self.average.x, self.average.y, 1.0]] )

}

fn from\_matrix(&mut self, m: &Matrix) {

self.p1 = Point2D { x: m.matrix[0][0], y: m.matrix[0][1] };

self.p2 = Point2D { x: m.matrix[1][0], y: m.matrix[1][1] };

self.average = Point2D { x: m.matrix[2][0], y: m.matrix[2][1] };

}

#[inline]

fn draw(&self, renderer: &Renderer) {

self.draw\_bresenham\_line(renderer);

}

}

Файл matrix.rs

pub struct Matrix {

pub matrix: [[f32; 3]; 3],

}

macro\_rules! matrix {

($x: expr) => {

Matrix { matrix: $x }

};

}

impl Mul for Matrix {

type Output = Matrix;

fn mul(self, \_rhs: Matrix) -> Matrix {

let mut new = null\_matrix();

for row in 0..3 {

for col in 0..3 {

for inner in 0..3 {

new.matrix[row][col] += self.matrix[row][inner] \*

\_rhs.matrix[inner][col];

}

}

}

return new;

}

}

#[inline]

pub fn null\_matrix() -> Matrix {

matrix!([[0.0, 0.0, 0.0],

[0.0, 0.0, 0.0],

[0.0, 0.0, 0.0]])

}

#[inline]

pub fn identity\_matrix() -> Matrix {

matrix!([[1.0, 0.0, 0.0],

[0.0, 1.0, 0.0],

[0.0, 0.0, 1.0]])

}

#[inline]

pub fn translation\_matrix(dx: f32, dy: f32) -> Matrix {

matrix!([[1.0, 0.0, 0.0],

[0.0, 1.0, 0.0],

[dx, dy, 1.0]])

}

#[inline]

pub fn scale\_matrix(sx: f32, sy: f32) -> Matrix {

matrix!([[sx, 0.0, 0.0],

[0.0, sy, 0.0],

[0.0, 0.0, 1.0]])

}

#[inline]

pub fn rotation\_matrix(angle: f32) -> Matrix {

let a = angle.to\_radians();

matrix!([[ a.cos(), a.sin(), 0.0],

[-a.sin(), a.cos(), 0.0],

[0.0, 0.0, 1.0]])

}