ГУАП

КАФЕДРА №14

ОТЧЕТ

ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

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

**ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №2**

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 концентрические круги и с помощью системного таймера замерить скорость отрисовки своей реализации алгоритма и встроенной.

2. Листинги

Файл main.rs

const WIDTH: u32 = 1600;

const HEIGHT: u32 = 800;

fn main() {

// Set first circle center and radius

let startx1 = (WIDTH / 100 \* 25) as f32;

let startx2 = (WIDTH / 100 \* 75) as f32;

let starty = (HEIGHT / 2) as f32;

// Set result text coordinates

let textx = (WIDTH - 200) as i16;

let texty = 10;

// Create circles

let mut bresenham\_circles = Vec::with\_capacity(200);

let mut builtin\_circles = Vec::with\_capacity(200);

for r in (4..).step\_by(2).take(198) {

bresenham\_circles.push(

Circle::new(

Point2D::new(startx1, starty),

r,

Color::RGB(0, 255, 0)

)

);

builtin\_circles.push(

Circle::new(

Point2D::new(startx2, starty),

r,

Color::RGB(255, 0, 0)

)

);

}

// Start main loop

loop {

// Clear render buffer

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

renderer.clear();

// Plot circles

let builtin\_duration = stopwatch(|| {

for c in &builtin\_circles {

c.draw\_builtin\_circle(&renderer);

}

});

let bresenham\_duration = stopwatch(|| {

for c in &bresenham\_circles {

c.draw(&renderer);

}

});

/\* Render time for each circles set here \*/

// Present render buffer

renderer.present();

}

}

fn stopwatch<F>(mut closure: F) -> Duration

where F: FnMut() {

use std::time::SystemTime;

let before\_time = SystemTime::now();

closure();

let after\_time = SystemTime::now();

after\_time.duration\_since(before\_time).unwrap()

}

Файл circle.rs

pub struct Circle {

center: Point2D,

radius: i16,

color: Color

}

impl Circle {

pub fn new(center: Point2D, radius: i16, color: Color) -> Circle {

Circle { center: center, radius: radius, color: color }

}

#[inline]

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

let x0 = self.center.x as i16;

let y0 = self.center.y as i16;

let mut x = 0;

let mut y = self.radius;

let mut dp = 1 - self.radius;

let color = self.color.as\_u32();

while x < y+1 {

unsafe {

ll::pixelColor(renderer.raw(), x0 + x, y0 + y, color);

ll::pixelColor(renderer.raw(), x0 - x, y0 + y, color);

ll::pixelColor(renderer.raw(), x0 + x, y0 - y, color);

ll::pixelColor(renderer.raw(), x0 - x, y0 - y, color);

ll::pixelColor(renderer.raw(), x0 + y, y0 + x, color);

ll::pixelColor(renderer.raw(), x0 - y, y0 + x, color);

ll::pixelColor(renderer.raw(), x0 + y, y0 - x, color);

ll::pixelColor(renderer.raw(), x0 - y, y0 - x, color);

}

x += 1;

if dp < 0 {

dp = dp + 2 \* x + 3;

} else {

y -= 1;

dp = dp + 2 \* x - 2 \* y + 5;

}

}

}

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

}