ГУАП
КАФЕДРА №14

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ
ПРЕПОДАВАТЕЛЬ

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

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №4

по курсу: КОМПЬЮТЕРНАЯ ГРАФИКА

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. 1441		М.И. Лубинец
	подпись, дата	инициалы, фамилия

Санкт-Петербург
2015

1. Формализация задачи

На основе лабораторной работы №3, реализовать алгоритм закраски нарисованной фигуры

2. Листинги

Файл main.rs

```
const WIDTH: u32 = 1280;
const HEIGHT: u32 = 720;

fn main() {
    let w0 = WIDTH as f32  / 100.0;
    let h0 = HEIGHT as f32 / 100.0;

    // Create polygons
    let mut polygons = vec![
        Polygon::new(&Point2D::new(w0 * 10.0, h0 * 50.0), 30.0,  3, Color::RGB(255, 0, 0)),
        Polygon::new(&Point2D::new(w0 * 25.0, h0 * 50.0), 50.0,  4, Color::RGB(0, 0, 255)),
        Polygon::new(&Point2D::new(w0 * 50.0, h0 * 50.0), 100.0, 6, Color::RGB(0, 255, 0)),
        Polygon::new(&Point2D::new(w0 * 75.0, h0 * 50.0), 50.0,  5, Color::RGB(0, 0, 255)),
        Polygon::new(&Point2D::new(w0 * 90.0, h0 * 50.0), 30.0,  3, Color::RGB(255, 0, 0)),
    ];

    // Start main loop
    loop {
        // Initialize variables
        let mut dx = 0.0;
        let mut dy = 0.0;
        let mut scale = 1.0;
        let mut angle = 0.0;
        let mut fill = false;

        // 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 += 0.005,
                Down    => scale -= 0.005,
                Left    => angle -= 3.0,
                Right   => angle += 3.0,
                F       => fill = true,
                _       => (),
            }
        }

        // Clear render buffer
        renderer.set_draw_color(Color::RGB(0, 0, 0));
        renderer.clear();


        // Do affine transformations and draw
        for poly in &mut polygons {
            poly.translate(dx, dy);
            poly.rotate(angle);
            poly.scale(scale, scale);
            poly.draw(&renderer);
            if fill {
                poly.fill(&renderer);
            }
        }

        // Present render buffer
        renderer.present();
    }
}
```

Файл primitives/mod.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 anchor_point(&self) -> Point2D;
    fn set_anchor_point(&mut self, anchor: &Point2D);

    fn translate(&mut self, dx: f32, dy: f32) {
        let obj = self.to_matrix();
        let mut anchor = self.anchor_point();

        self.from_matrix(
          &(obj *
            Matrix::translation_matrix(-anchor.x, -anchor.y) *
            Matrix::translation_matrix(dx, dy) *
            Matrix::translation_matrix(anchor.x, anchor.y))
        );

        // Moving anchor point by dx and dy
        anchor.x += dx;
        anchor.y += dy;

        self.set_anchor_point(&anchor);
    }

    fn scale(&mut self, sx: f32, sy: f32) {
        let obj = self.to_matrix();
        let anchor = self.anchor_point();

        self.from_matrix(
          &(obj *
            Matrix::translation_matrix(-anchor.x, -anchor.y) *
            Matrix::scale_matrix(sx, sy) *
            Matrix::translation_matrix(anchor.x, anchor.y))
        );
    }

    fn rotate(&mut self, angle: f32) {
        let obj = self.to_matrix();
        let anchor = self.anchor_point();

        self.from_matrix(
          &(obj *
            Matrix::translation_matrix(-anchor.x, -anchor.y) *
            Matrix::rotation_matrix(angle) *
            Matrix::translation_matrix(anchor.x, anchor.y))
        );
    }

    /* Draw the object on screen */
    fn draw(&self, renderer: &Renderer);

    fn fill(&seld, renderer: &Renderer);
}
```

Файл matrix.rs

```
pub struct Matrix {
    pub matrix: Vec<[f32; 3]>,
}

impl Matrix {
    pub fn new(rows: Vec<[f32; 3]>) -> Matrix {
        Matrix {
            matrix: rows
        }
    }
}

impl Mul for Matrix {
    type Output = Matrix;

    fn mul(self, _rhs: Matrix) -> Matrix {
        let rows_cnt = self.matrix.len();
        let mut new = Matrix::null_matrix(rows_cnt);
        for row in 0..rows_cnt {
            for col in 0..3 {
                for inner in 0..3 {
                    new.matrix[row][col] += self.matrix[row][inner] *
                                           _rhs.matrix[inner][col];
                }
            }
        }
        return new;
    }
}

impl Matrix {
    #[inline]
    pub fn null_matrix(rows: usize) -> Matrix {
        let mut vec = Vec::with_capacity(rows);
        for _ in 0..rows {
            vec.push([0.0, 0.0, 0.0]);
        }
        Matrix::new(vec)
    }

    #[inline]
    pub fn identity_matrix() -> Matrix { /* ... */ }

    #[inline]
    pub fn translation_matrix(dx: f32, dy: f32) -> Matrix { /* ... */ }

    #[inline]
    pub fn scale_matrix(sx: f32, sy: f32) -> Matrix { /* ... */ }

    #[inline]
    pub fn rotation_matrix(angle: f32) -> Matrix { /* ... */ }
```

Файл polygon.rs

```
pub struct Polygon {
    vertices: Vec<Point2D>,
    anchor:   Point2D,
    color:    Color
}

impl Polygon {
    pub fn new(center: &Point2D, radius: f32, cnt: usize, color: Color) -> Polygon { /* ... */ }

impl Primitive2D for Polygon {
    fn to_matrix(&self) -> Matrix { /* ... */ }

    fn from_matrix(&mut self, m: &Matrix) { /* ... */ }

    fn anchor_point(&self) -> Point2D { /* ... */ }

    fn set_anchor_point(&mut self, anchor: &Point2D) { /* ... */ }

    fn draw(&self, renderer: &Renderer) { /* ... */ }

    fn fill(&self, renderer: &Renderer) {
            let is_inside = |x: i16, y: i16| -> bool {
                let v_cnt = self.vertices.len();
                let mut j = v_cnt - 1;
                let mut c = false;
                let p = &self.vertices;

                for i in 0..v_cnt {
                    let pix = p[i].x as i16;
                    let piy = p[i].y as i16;
                    let pjx = p[j].x as i16;
                    let pjy = p[j].y as i16;

                    if (((piy <= y) && (y < pjy)) || ((pjy <= y) && (y < piy)))
                    && (x > (pjx - pix) * (y - piy) / (pjy - piy) + pix) {
                        c = !c;
                    }
                    j = i;
                }
                return c;
            };

            let minx = self.vertices.iter().minX() as i16;
            let miny = self.vertices.iter().minY() as i16;
            let maxx = self.vertices.iter().maxX() as i16;
            let maxy = self.vertices.iter().minY() as i16;

            let color = self.color.as_u32();

            for y in miny..maxy {
                for x in minx..maxx {
                    if is_inside(x, y) {
                        unsafe {
                            ll::pixelColor(renderer.raw(), x, y, color);
                        }
                    }
                }
            }
        }
```