ГУАП
КАФЕДРА №14

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ
ПРЕПОДАВАТЕЛЬ

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

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №3

по курсу: КОМПЬЮТЕРНАЯ ГРАФИКА

РАБОТУ ВЫПОЛНИЛ
СТУДЕНТ ГР.   1441   М.И. Лубинец

подпись, дата	инициалы, фамилия

Санкт-Петербург
2015

1. Формализация задачи

Нарисовать на экране многоугольник с заданным количеством граней (3, 4, 5 или 6)

1. Формализация задачи

Нарисовать на экране многоугольник с заданным количеством граней (3, 4, 5 или 6)

2. Листинги

Файл main.rs

```rust
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;

        // 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,
                _       => (),
            }
        }

        // 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);
        }

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

```rust
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);
}
```

Файл matrix.rs

```rust
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

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

impl Polygon {
    pub fn new(center: &Point2D, radius: f32, cnt: usize, color: Color) -> Polygon {
        let mut vertices = Vec::with_capacity(cnt);

        for i in 0..cnt {
            let n = cnt as f32;
            let i = i as f32;
            vertices.push(
                Point2D::new(
                    center.x + radius * (2.0 * consts::PI * i / n).cos(),
                    center.y + radius * (2.0 * consts::PI * i / n).sin(),
                )
            )
        }

        Polygon { vertices: vertices, anchor: center.clone(), color: color }
    }
}

impl Primitive2D for Polygon {
    fn to_matrix(&self) -> Matrix {
        let mut matrix = Vec::with_capacity(self.vertices.len());
        for vertex in &self.vertices {
            matrix.push([vertex.x, vertex.y, 1.0]);
        }

        Matrix::new(
            matrix
        )
    }

    fn from_matrix(&mut self, m: &Matrix) {
        for (i, row) in m.matrix.iter().enumerate() {
            self.vertices[i].x = row[0];
            self.vertices[i].y = row[1];
        }
    }

    fn anchor_point(&self) -> Point2D {
        self.anchor.clone()
    }

    fn set_anchor_point(&mut self, anchor: &Point2D) {
        self.anchor = anchor.clone();
    }

    fn draw(&self, renderer: &Renderer) {
        let mut lines = Vec::with_capacity(self.vertices.len() + 1);
        for i in 0..self.vertices.len() {
            let current_vertex = &self.vertices[i];
            let next_vertes = &self.vertices[(i+1) % self.vertices.len()];
            lines.push(
                Line::new(current_vertex, next_vertes, self.color)
            );
        }

        for line in lines {
            line.draw(renderer);
        }
    }
}
```

3. Скриншот