SPRAWOZDANIE

Zajęcia: Grafika komputerowa
Prowadzący: prof. dr hab. Vasyl Martsenyuk

Laboratorium 10
Data: 17 Maj 2021
Temat: " **Podstawy WebGL/GLSL**"

Imię Nazwisko: Klaudia Gruszczyk
Informatyka I stopień,
stacjonarne,
4 semestr,
Gr.1b

Celem jest zapoznanie się z podstawami WebGL/GLSL

1. Wprowadzane dane:

```
<script type="x-shader/x-vertex" id="vshader-source">
    attribute vec2 a_coords; // vertex position in standard canvas pixel coords
    attribute vec3 color;
    uniform float u_width;   // width of canvas
    uniform float u_height;  // height of canvas
    uniform float u_pointSize;
    uniform int u_type;
    varying vec3 outcolor;
    varying float type;
    void main() {
        float x,y;  // vertex position in clip coordinates
        x = a_coords.x/u_width * 2.0 - 1.0;  // convert pixel coords to clip
coords
        y = 1.0 - a_coords.y/u_height * 2.0;
        gl_Position = vec4(x, y, 0.0, 1.0);
        gl_PointSize = u_pointSize;
        outcolor = vec3(color);
        type = float(u_type);
    }
</script>
<script type="x-shader/x-fragment" id="fshader-source">
    #ifdef GL_FRAGMENT_PRECISION_HIGH
       precision highp float;
    #else
       precision mediump float;
    #endif

    varying vec3 outcolor;
    varying float type;

    const float pi=3.141592653589793;

    float polygon(float s, float apotheme, vec2 p){
        float ang=atan(p.x,p.y);
        ang-=floor(ang/pi/2.*s)/s*pi*2.-pi/s;
        return cos(atan(p.x,p.y)-floor(atan(p.x,p.y)/pi/2.*s)/s*pi*2.-
pi/s)*length(p)<apotheme?1.:0.;
    }

    void main() {
        float dist = distance( vec2(0.5), gl_PointCoord );

        gl_FragColor = vec4(outcolor, 1.0);
        if ( type > 4.0 ){
            if ( dist > polygon( type , 0.4, vec2(gl_PointCoord.x - 0.5,
gl_PointCoord.y- 0.5))) {
                discard;
            }
        }
    }
</script>

<script>
```

```
"use strict";

var canvas; // The canvas that is used for WebGL drawing; occupies the entire
window.
var gl;     // The webgl context.

var u_width_loc;        // Location of "width" uniform, which holds the width of
the canvas.
var u_height_loc;       // Location of "height" uniform, which holds the height of
the canvas.
var u_pointSize_loc;    // Location of "pointSize" uniform, which gives the size
for point primitives.
var a_coords_loc;       // Location of the a_coords attribute variable in the
shader program;
                        //    This attribute gives the (x,y) coordinates of the
points.
var a_color_loc;

var a_coords_buffer;    // Buffer to hold the values for a_coords (coordinates for
the points)
var a_color_buffer;

var u_type_loc;

var POINT_COUNT = 30;   // How many points to draw.
var POINT_SIZE = 64;    // Size in pixel of the square drawn for each point.

var nSides = 5;

var positions = new Float32Array( 2*POINT_COUNT );  // Position data for points.
var velocities = new Float32Array( 2*POINT_COUNT );
var color = new Float32Array( 3*POINT_COUNT );
 // Velocity data for points.
      // Note: The xy coords for point number i are in
positions[2*i],position[2*i+1].
      // The xy velocity compontents for point number i are in
velocities[2*i],velociteis[2*i+1].
      // Position coordinates are in pixels, and velocity components are in pixels
per frame.

var isRunning = true;  // The animation runs when this is true; its value is
toggled by the space bar.

function SetRandomColor(){
        for (let i = 0; i < color.length; i++) {
            color[i] = Math.random();
        }
 }

function changeType(){
    var num = prompt("Ile kątów?", "4");
    nSides = parseInt(num);
    gl.uniform1i(u_type_loc, nSides);
}
var isColorRandom = false;
/**
 *  Called by init() when the window is first opened, and by frame() to render
each frame.
 */
```

```
function render() {

    gl.clear(gl.COLOR_BUFFER_BIT);  // clear the color buffer before drawing

    // The position data changes for each frame, so we have to send the new values
    // for the position attirbute into the corresponding buffer in the GPU here,
    // in every frame.

    gl.bindBuffer(gl.ARRAY_BUFFER, a_coords_buffer);            // Select the
buffer we want to use.
    gl.bufferData(gl.ARRAY_BUFFER, positions, gl.STREAM_DRAW);  // Send the data.
    gl.vertexAttribPointer(a_coords_loc, 2, gl.FLOAT, false, 0, 0);  // Describes
the data format.

    if ( isColorRandom ){
        gl.enableVertexAttribArray(a_color_loc);
    } else {
        gl.disableVertexAttribArray(a_color_loc);
        gl.vertexAttrib3f (a_color_loc, 1, 0, 0)
    }
    // Now, draw the points as a primitive of type gl.POINTS

    gl.drawArrays(gl.POINTS, 0, POINT_COUNT);

    if (gl.getError() != gl.NO_ERROR) {
        console.log("During render, a GL error has been detected.");
    }
} // end render()


/**
 * Called once in init() to create the data for the scene. Creates point positions
and
 * velocities.  All points start at the center of the canvas, with random
velocity.
 * The speed is between 2 and 6 pixels per frame.
 */
function createData() {
    SetRandomColor();
    for (var i = 0; i < POINT_COUNT; i++) {
        positions[2*i] = canvas.width/2;
        positions[2*i+1] = canvas.height/2;
        var speed = 2 + 4*Math.random();
        var angle = 2*Math.PI*Math.random();
        velocities[2*i] = speed*Math.sin(angle);
        velocities[2*i+1] = speed*Math.cos(angle);
    }
} // end createData()


/**
 * Called by frame() before each frame is rendered.  Adds velcities
 * to point positions.  If the point moves past the edge of the canvas,
 * it bounces.
 */
function updateData() {
    for (var i = 0; i < POINT_COUNT; i++) {
       positions[2*i] += velocities[2*i];
       if ( positions[2*i] < POINT_SIZE/2 && velocities[2*i] < 0) {
```

```
            positions[2*i] += 2*(POINT_SIZE/2 - positions[2*i]);
            velocities[2*i] = Math.abs(velocities[2*i]);
        }
        else if (positions[2*i] > canvas.width - POINT_SIZE/2 && velocities[2*i] >
0){
            positions[2*i] -= 2*(positions[2*i] - canvas.width + POINT_SIZE/2);
            velocities[2*i] = - Math.abs(velocities[2*i]);
        }
        positions[2*i+1] += velocities[2*i+1];
        if ( positions[2*i+1] < POINT_SIZE/2  && velocities[2*i+1] < 0) {
            positions[2*i+1] += 2*(POINT_SIZE/2 - positions[2*i+1]);
            velocities[2*i+1] = Math.abs(velocities[2*i+1]);
        }
        else if (positions[2*i+1] > canvas.height - POINT_SIZE/2  &&
velocities[2*i+1] > 0){
            positions[2*i+1] -= 2*(positions[2*i+1] - canvas.height +
POINT_SIZE/2);
            velocities[2*i+1] = - Math.abs(velocities[2*i+1]);
        }
    }
} // end updateData()


/* Called when the user hits a key */
function doKey(evt) {
    var key = evt.keyCode;
    console.log("key pressed with keycode = " + key);
    if ( key == 49){
        isColorRandom == false ? isColorRandom = true : isColorRandom = false;
    }
    if ( key == 50) {
        nSides = 4;
        changeType();
    }
    if (key == 32) {  // space bar
        if (isRunning) {
            isRunning = false;  // stops the animation
        }
        else {
            isRunning = true;
           requestAnimationFrame(frame);  // restart the animation
        }
    }
} // end doKey();


/* Initialize the WebGL context.  Called from init() */
function initGL() {

    var prog = createProgram(gl,"vshader-source", "fshader-source", "a_coords");
    gl.useProgram(prog);

    /* Get locations of uniforms and attributes. */

    u_width_loc = gl.getUniformLocation(prog,"u_width");
    u_height_loc = gl.getUniformLocation(prog,"u_height");
    u_pointSize_loc = gl.getUniformLocation(prog,"u_pointSize");
    a_coords_loc = gl.getAttribLocation(prog,"a_coords");
    a_color_loc = gl.getAttribLocation(prog, "color");
```

```
    u_type_loc = gl.getUniformLocation(prog, "u_type");
    /* Assign initial values to uniforms. */

    gl.uniform1f(u_width_loc, canvas.width);
    gl.uniform1f(u_height_loc, canvas.height);
    gl.uniform1f(u_pointSize_loc, POINT_SIZE);
    /* Create and configure buffers for the attributes. */


    a_coords_buffer = gl.createBuffer();
    gl.enableVertexAttribArray(a_coords_loc); // data from the attribute will come
from a buffer.

    a_color_buffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, a_color_buffer);
    gl.bufferData(gl.ARRAY_BUFFER, color, gl.STATIC_DRAW);
    gl.vertexAttribPointer(
        a_color_loc,
        3,
        gl.FLOAT,
        false,
        0,
        0);

    /* Configure other WebGL options. */

    gl.clearColor(0,0,0,1);  // gl.clear will fill canvas with black.

    if (gl.getError() != gl.NO_ERROR) {
        console.log("During initialization, a GL error has been detected.");
    }
} // end initGL()
```

2. Wynik działania: