SPRAWOZDANIE

Zajęcia: Grafika komputerowa

Prowadzący: prof. dr hab. Vasyl Martsenyuk

Laboratorium 11

Data 29.05.2022

Temat: Grafika 3D w bibliotece WebGL/GLSL

Natalia Pierzchała

Informatyka I stopień,

stacjonarne,

4 semestr,

Gr. 2a

1. Polecenie:

Plik lab12.html pokazuje mały sześcian, który można obrócić, przeciągając myszą na płótnie. Zadaniem jest zastąpienie sześcianu dużym wiatrakiem siedzącym na prostokątnej podstawie, jak pokazano na rysunku. Łopatki wiatraka powinny obracać się po włączeniu animacji. Każda łopatka wiatraka powinna być zbudowana z dwóch stożków. (Dodanie czajniczka, który znajduje się na podstawie, jest konieczne dla uzyskania oceny "5")

Program zawiera trzy zmienne instancji reprezentujące podstawowe obiekty: cube, cone, cylinder. Te zmienne mają metody instancji cube.render(), cone.render(), cylinder.render(), które można wywołać w celu narysowania obiektów. Obiekty nietransformowane mają rozmiar 1 we wszystkich trzech kierunkach i mają swój środek na (0,0,0). Oś stożka i oś cylindra są wyrównane wzdłuż osi Z. Wszystkie obiekty na scenie powinny być przekształconymi wersjami podstawowych obiektów (lub podstawowego obiektu czajnika).

2. Wprowadzane dane i wykorzystane komendy:

```
<!DOCTYPE html>
<meta charset="UTF-8">
<html>
<head>
<title>WebGL Intro</title>
<style>
   html, body {
      margin: 0;  /* Make sure that there is no margin around the canvas */
      overflow: hidden;  /* Make sure we don't get scroll bars. */
   }
   canvas {
      display: block; /* The default display, inline, would add a small margin below the canvas
*/
   }
</style>

<!--
   A 2D WebGL app in which "points" move around in the browser window, bouncing
   off the edges.  The animation can be paused and restarted by pressing the
   space key.
      If the user clicks or clicks-and-drags with the mouse, all of the
   points head towards the mouse position, except if the user shift-clicks, the
   positions and velocities of the points are re-initialized.
-->

<script type="x-shader/x-vertex" id="vshader-source">
   attribute vec2 a_coords; // vertex position in standard canvas pixel coords
   uniform float u_width;   // width of canvas
   uniform float u_height;  // height of canvas
   uniform float u_pointSize;
   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;
   }
</script>

<script type="x-shader/x-fragment" id="fshader-source">
   #ifdef GL_FRAGMENT_PRECISION_HIGH
      precision highp float;
   #else
      precision mediump float;
   #endif
   void main() {
      gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);
   }
</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_coords_buffer;    // Buffer to hold the values for a_coords (coordinates for the points)

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

var positions = new Float32Array( 2*POINT_COUNT );  // Position data for points.
var velocities = new Float32Array( 2*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.


/**
 *  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.

   // 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() {
   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 == 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");

    /* 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.

    /* 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()


/**
 * Creates a program for use in the WebGL context gl, and returns the
 * identifier for that program.  If an error occurs while compiling or
 * linking the program, an exception of type String is thrown.  The error
 * string contains the compilation or linking error.  If no error occurs,
 * the program identifier is the return value of the function.
 *    The second and third parameters are the id attributes for <script>
 * elements that contain the source code for the vertex and fragment
 * shaders.
 *    If the third parameter is present, it should be the name of an
 * attribute variable in the shader program, and the attribute should be
 * one that is always used.  The attribute will be assigned attribute
 * number 0.  This is done because it is suggested that there should
 * always be an attribute number 0 in use.
 */
function createProgram(gl, vertexShaderID, fragmentShaderID, attribute0) {
   function getTextContent( elementID ) {
         // This nested function retrieves the text content of an
         // element on the web page.  It is used here to get the shader
         // source code from the script elements that contain it.
      var element = document.getElementById(elementID);
      var node = element.firstChild;
      var str = "";
      while (node) {
         if (node.nodeType == 3) // this is a text node
            str += node.textContent;
         node = node.nextSibling;
      }
      return str;
   }
   try {
      var vertexShaderSource = getTextContent( vertexShaderID );
      var fragmentShaderSource = getTextContent( fragmentShaderID );
   }
   catch (e) {
      throw "Error: Could not get shader source code from script elements.";
   }
   var vsh = gl.createShader( gl.VERTEX_SHADER );
   gl.shaderSource(vsh,vertexShaderSource);
   gl.compileShader(vsh);
   if ( ! gl.getShaderParameter(vsh, gl.COMPILE_STATUS) ) {
      throw "Error in vertex shader:  " + gl.getShaderInfoLog(vsh);
    }
   var fsh = gl.createShader( gl.FRAGMENT_SHADER );
   gl.shaderSource(fsh, fragmentShaderSource);
   gl.compileShader(fsh);
   if ( ! gl.getShaderParameter(fsh, gl.COMPILE_STATUS) ) {
     throw "Error in fragment shader:  " + gl.getShaderInfoLog(fsh);
   }
   var prog = gl.createProgram();
   gl.attachShader(prog,vsh);
   gl.attachShader(prog, fsh);
   if (attribute0) {
      gl.bindAttribLocation(prog,0,attribute0);
   }
   gl.linkProgram(prog);
   if ( ! gl.getProgramParameter( prog, gl.LINK_STATUS) ) {


      throw "Link error in program:  " + gl.getProgramInfoLog(prog);
   }
   return prog;
}

/**
 *  A function to drive the animation, which runs continuously while the global
 *  variable isRunning is true.  The value of this variable is toggled by pressing
 *  the space bar.  If the animation is still running, this fucntion calls
 *  updateData(), then calls render(), then calls requestAnimationFrame to
 *  schedule the next call to the same function.
 */
function frame() {
   if (isRunning) {
      updateData();
      render();
      requestAnimationFrame(frame);  // Arrange for function to be called again
   }
}

/**
 * When the window is resized, we need to resize the canvas, reset the
 * OpenGL viewport to match the size, and reset the values of the uniform
 * variables in the shader that represent the canvas size.
 */
 function doResize() {
     canvas.width = window.innerWidth;
     canvas.height = window.innerHeight;
     gl.viewport(0, 0, canvas.width, canvas.height);
     gl.uniform1f(u_width_loc, canvas.width);
     gl.uniform1f(u_height_loc, canvas.height);
     if (!isRunning) {
        render();
     }
 }
```

```
/**
 * Responds to left mouse click on canvas; points all head toward mouse location
 * when mouse is clicked and as it is dragged.  However if shift key is down,
 * all the data is reinitialized instead.
 */
function doMouse(evt) {
   function headTowards(x,y) {
      for (var i = 0; i < POINT_COUNT; i++) {
         var dx = x - positions[2*i];
         var dy = y - positions[2*i+1];
         var dist = Math.sqrt(dx*dx + dy*dy);
         if (dist > 0.1) { // only if mouse and point are not too close.
            var speed = Math.sqrt( velocities[2*i]*velocities[2*i] +
velocities[2*i+1]*velocities[2*i+1] );
            velocities[2*i] = dx/dist * speed;
            velocities[2*i+1] = dy/dist * speed;
         }
      }
   }
   function move(evt) {
      headTowards(evt.clientX,evt.clientY);
   }
   function up() {
      canvas.removeEventListener("mousemove", move, false);
      document.removeEventListener("mouseup", up, false);
   }
   if (evt.which != 1) {
      return;  // only respond to left mouse down
   }
   if (evt.shiftKey) {
      createData();
      return;
   }
   headTowards(evt.clientX,evt.clientY);
   canvas.addEventListener("mousemove", move);
   document.addEventListener("mouseup", up);
}

/**
 * initialization function that will be called when the page has loaded.
 */
function init() {
   try {
      canvas = document.createElement("canvas");
      canvas.width = window.innerWidth;
```

```
        canvas.height = window.innerHeight;
        var options = {
            alpha: false,   // The color buffer doesn't need an alpha component
            depth: false,   // No need for a depth buffer in this 2D program
            stencil: false  // This program doesn't use a stencil buffer
          };
        gl = canvas.getContext("webgl", options);
        if ( ! gl ) {
           throw "Browser does not support WebGL";
        }
    }
    catch (e) {
        var message = document.createElement("p");
        message.innerHTML = "Sorry, could not get a WebGL graphics context.  Error: " + e;
        document.body.appendChild(message);
        return;
    }
    try {
        createData();  // create data for points (in case it's needed in initGL())
        initGL();  // initialize the WebGL graphics context.
    }
    catch (e) {
        var message = document.createElement("p");
        message.innerHTML =
           "<pre>Sorry, could not initialize graphics context.  Error:\n\n" + e + "</pre>";
        document.body.appendChild(message);
        return;
    }
    document.body.appendChild(canvas);
    window.addEventListener("resize", doResize);
    canvas.addEventListener("mousedown",doMouse);
    document.addEventListener("keydown",doKey);
    requestAnimationFrame(frame);
}

</script>
</head>
<body onload="init()">
<noscript>Sorry, this page requires JavaScript.</noscript>
</body>
</html>
```

3. Wykorzystane komendy:

`https://github.com/99lucky8/Grafika-komputerowa.git`

4. Wyniki działania