**SPRAWOZDANIE**

Zajęcia: Grafika komputerowa

Prowadzący: mgr Mikołaj Grygiel

**Laboratorium 11**

23.06.2024

**Temat:** Podstawy WebGL/GLSL

**Wariant**: 1

Radosław Skrzypczyński

Informatyka I stopień,

niestacjonarne,

4 semestr,

Gr. 1B

1. **Polecenie:**

Program w lab11.html pokazuje wiele ruchomych czerwonych kwadratów, które odbijają się od krawędzi płótna. Płótno wypełnia cały obszar zawartości przeglądarki internetowej. Kwadraty odpowiadają również na ruchy myszy: Jeśli klikniesz lewym przyciskiem myszy lub klikniesz lewym przyciskiem myszy i przeciągniesz na płótnie, cały kwadrat będzie kierowany w stronę pozycji myszy. Jeśli klikniesz lewym przyciskiem myszy, dane punktów zostaną ponownie zainicjowane, więc zaczną się od środka. Możesz wstrzymać i ponownie uruchomić animację, naciskając spację.

Kwadraty są w rzeczywistości częścią jednego prymitywu WebGL typu gl.POINTS. Każdy kwadrat odpowiada jednemu z wierzchołków prymitywu. Oczywiście renderowanie jest wykonywane przez moduł shadera wierzchołków i moduł shadera fragmentu. Kod źródłowy shaderów jest w dwóch fałszywych „skryptach” w górnej części pliku html.

Będziesz modyfikował kod modułu shadera i kod JavaScript, aby zaimplementować kilka różnych stylów dla prymitywu punktu. Na przykład możliwe będzie rysowanie kwadratów w różnych kolorach, rysowanie wielokątów zamiast kwadratów i tak dalej. Użytkownik będzie kontrolował program, naciskając klawisze na klawiaturze. Do ciebie należy decyzja, których klawiszy użyć, ale proszę udokumentować interfejs w odpowiednim komentarzu do funkcji doKey() lub na górze programu.

Program ma dwie funkcje, nad którymi będziesz musiał pracować: Funkcja initGL() jest wywoływana, gdy program jest uruchamiany po raz pierwszy, a funkcje updateForFrame() i render() są wywoływane dla każdej ramki animacji. Ten sam zestaw poleceń byłby legalny we wszystkich tych poleceniach, ale initGL() jest najlepszym miejscem do ustawiania rzeczy, które nie zmieniają się w trakcie działania programu, takich jak położenie zmiennych i zmiennych atrybutów w module shadera; updateForFrame() jest przeznaczony do aktualizacji zmiennych JavaScript, które zmieniają się z ramki na ramkę; i render() ma na celu wykonanie rzeczywistego rysunku WebGL ramki.

Atrybut koloru

W oryginalnej wersji programu wszystkie kwadraty są czerwone. Pierwsze ćwiczenie polega na umożliwieniu przypisania innego koloru do każdego kwadratu. Ponieważ kwadraty są naprawdę wierzchołkami w pojedynczym prymitywie typu gl.POINTS, można użyć zmiennej atrybutu dla koloru. Atrybut może mieć inną wartość dla każdego wierzchołka.

Pierwszym zadaniem jest dodanie zmiennej kolorowej typu vec3 do modułu shadera wierzchołka i użycie wartości atrybutu do pokolorowania kwadratów. Będziesz także musiał pracować po stronie JavaScript. Będziesz potrzebował Float32Array do przechowywania wartości kolorów po stronie JavaScript, a będziesz potrzebował bufora WebGL dla tego atrybutu. Program ma już jeden atrybut, który jest używany do współrzędnych wierzchołków. Będziesz robił coś podobnego do atrybutu color (poza tym, że możesz to zrobić w initGL(), ponieważ wartości kolorów nie zmieniają się po ich utworzeniu). Można użyć losowych wartości w zakresie od 0,0 do 1,0 dla składników koloru.

Po uruchomieniu wielokolorowych kwadratów powinieneś ustawić kolory jako opcjonalne. Możesz włączać i wyłączać użycie tablicy wartości atrybutów za pomocą następujących poleceń, gdzie a\_color\_loc to identyfikator atrybutu color w programie shader:

gl.enableVertexAttribArray(a\_color\_loc); // użyj bufora atrybutów kolorów

gl.disableVertexAttribArray(a\_color\_loc); // nie używaj bufora

Gdy tablica atrybutów jest włączona, każdy wierzchołek otrzymuje swój własny kolor z bufora atrybutów. Gdy tablica atrybutów jest wyłączona, wszystkie wierzchołki otrzymują ten sam kolor, a tę wartość można ustawić za pomocą rodziny funkcji gl.vertexAttrib\*. Na przykład, aby ustawić wartość używaną, gdy tablica atrybutów kolorów jest wyłączona, można użyć:

gl.vertexAttrib3f(a\_color\_loc, 1, 0, 0); // ustaw kolor attrbute na czerwony

Pozwól użytkownikowi na naciśnięcie określonego klawisza, aby włączyć lub wyłączyć losowe kolory. Program ma funkcję doKey(), która jest już skonfigurowana do reagowania na wprowadzanie z klawiatury. Będziesz dodawać do programu kilka typów interakcji z klawiaturą. Aby odpowiedzieć na klawisz, musisz znać numeryczny kod klawiszy. Funkcja doKey() wysyła kod do konsoli za każdym razem, gdy użytkownik uderza klawisz, i możesz użyć tej funkcji, aby odkryć wszystkie inne kody klawiszy, których potrzebujesz.

Styl punktów

Powinieneś dodać opcję używania stylu wyświetlania dla punktów w postaci wielokąta. Pozwól użytkownikowi wybrać styl za pomocą klawiatury; na przykład, naciskając klawisze numeryczne. Style będą musiały zostać zaimplementowane w shaderze fragmentu, a będziesz potrzebował nowej zmiennej jednolitej, aby powiedzieć modułowi shadera fragmentu, którego stylu użyć. Dodaj jednolitą zmienną typu int do shadera fragmentu, aby kontrolować styl punktu, i dodaj kod do modułu cieniującego fragmentu, aby zaimplementować różne style. Będziesz także musiał dodać zmienną po stronie JavaScript dla lokalizacji zmiennej jednolitej, a będziesz musiał wywołać glUniform1i, gdy chcesz zmienić styl.

Na przykład, żeby narysować punkt jako dysk, odrzucając niektóre piksele:

float dist = distance(vec2(0.5), gl\_PointCoord);

if (dist > 0.5) {

discard;

}

Powinieneś również wykorzystać przezroczystość alfa w niektórych stylach. Aby umożliwić korzystanie ze składnika alpha, musisz dodać następujące linie do funkcji initGL():

Skopiuj kod

gl.enable(gl.BLEND);

gl.blendFunc(gl.SRC\_ALPHA, gl.ONE\_MINUS\_SRC\_ALPHA);

Dzięki tym ustawieniom wartość alfa koloru będzie używana do przezroczystości w zwykły sposób. W szczególności jeden z twoich stylów powinien pokazywać punkt jako wielokąt, który zanika z całkowicie nieprzezroczystego w środku wielokąta do całkowicie przezroczystego na krawędzi.

**2. Wprowadzane dane:**

Spacja – zatrzymanie animacji

0-9 – zmiana kształtu

A-Z – zmiana koloru

**3. 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>*

*<script type="x-shader/x-vertex" id="vshader-source">*

*attribute vec2 a\_coords; // vertex position in standard canvas pixel coords*

*attribute vec3 a\_color; // vertex color*

*varying vec3 v\_color; // color passed to fragment shader*

*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;*

*v\_color = a\_color;*

*}*

*</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 v\_color; // color passed from vertex shader*

*uniform int u\_style; // style for point rendering*

*void main() {*

*vec2 coords = gl\_PointCoord.xy - vec2(0.5);*

*float dist = length(coords);*

*if (u\_style == 0) { // Square style*

*gl\_FragColor = vec4(v\_color, 1.0);*

*} else if (u\_style == 1) { // Circle style*

*if (dist > 0.5) {*

*discard;*

*}*

*gl\_FragColor = vec4(v\_color, 1.0);*

*} else if (u\_style == 2) { // Gradient transparency*

*float alpha = 1.0 - dist \* 2.0;*

*if (dist > 0.5) {*

*discard;*

*}*

*gl\_FragColor = vec4(v\_color, alpha);*

*} else if (u\_style == 3) { // Triangle*

*if (coords.x < 0.0 || coords.y < 0.0 || coords.x + coords.y > 0.5) {*

*discard;*

*}*

*gl\_FragColor = vec4(v\_color, 1.0);*

*} else if (u\_style == 4) { // Star*

*float angle = atan(coords.y, coords.x) \* 5.0;*

*float radius = cos(angle);*

*if (dist > 0.5 || dist < 0.3 \* radius) {*

*discard;*

*}*

*gl\_FragColor = vec4(v\_color, 1.0);*

*} else if (u\_style == 5) { // Hexagon*

*float angle = atan(coords.y, coords.x) \* 3.0;*

*float radius = cos(angle);*

*if (dist > 0.5 || dist < 0.3 \* radius) {*

*discard;*

*}*

*gl\_FragColor = vec4(v\_color, 1.0);*

*} else if (u\_style == 6) { // Diamond*

*if (abs(coords.x) + abs(coords.y) > 0.5) {*

*discard;*

*}*

*gl\_FragColor = vec4(v\_color, 1.0);*

*} else if (u\_style == 7) { // Pentagon*

*float angle = atan(coords.y, coords.x) \* 5.0 / 3.0;*

*float radius = cos(angle);*

*if (dist > 0.5 || dist < 0.3 \* radius) {*

*discard;*

*}*

*gl\_FragColor = vec4(v\_color, 1.0);*

*} else if (u\_style == 8) { // Octagon*

*float angle = atan(coords.y, coords.x) \* 8.0 / 3.0;*

*float radius = cos(angle);*

*if (dist > 0.5 || dist < 0.3 \* radius) {*

*discard;*

*}*

*gl\_FragColor = vec4(v\_color, 1.0);*

*} else if (u\_style == 9) { // Heart*

*float r = length(coords);*

*float theta = atan(coords.y, coords.x);*

*if (r > 0.5 || r < 0.3 \* abs(cos(theta))) {*

*discard;*

*}*

*if (coords.y > 0.0 && (r > 0.5 || r < 0.3 \* abs(sin(theta)))) {*

*discard;*

*}*

*gl\_FragColor = vec4(v\_color, 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 u\_style\_loc; // Location of "style" uniform, which gives the rendering style for points.*

*var a\_coords\_loc; // Location of the a\_coords attribute variable in the shader program;*

*var a\_color\_loc; // Location of the a\_color attribute variable in the shader program.*

*var a\_coords\_buffer; // Buffer to hold the values for a\_coords (coordinates for the points)*

*var a\_color\_buffer; // Buffer to hold the values for a\_color (colors 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.*

*var colors = new Float32Array( 3\*POINT\_COUNT ); // Color data for points.*

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

*var pointStyle = 0; // The current style of points: 0 for square, 1 for circle, etc.*

*var colorVariant = 0; // The current color variant.*

*function render() {*

*gl.clear(gl.COLOR\_BUFFER\_BIT); // clear the color buffer before drawing*

*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.*

*gl.bindBuffer(gl.ARRAY\_BUFFER, a\_color\_buffer);*

*gl.bufferData(gl.ARRAY\_BUFFER, colors, gl.STATIC\_DRAW);*

*gl.vertexAttribPointer(a\_color\_loc, 3, gl.FLOAT, false, 0, 0);*

*gl.enableVertexAttribArray(a\_color\_loc);*

*gl.uniform1i(u\_style\_loc, pointStyle);*

*gl.drawArrays(gl.POINTS, 0, POINT\_COUNT);*

*if (gl.getError() != gl.NO\_ERROR) {*

*console.log("During render, a GL error has been detected.");*

*}*

*}*

*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);*

*setColor(i);*

*}*

*}*

*function setColor(index) {*

*if (colorVariant == 0) { // Random colors*

*colors[3\*index] = Math.random();*

*colors[3\*index+1] = Math.random();*

*colors[3\*index+2] = Math.random();*

*} else if (colorVariant == 1) { // Shades of red*

*colors[3\*index] = Math.random();*

*colors[3\*index+1] = 0.0;*

*colors[3\*index+2] = 0.0;*

*} else if (colorVariant == 2) { // Shades of green*

*colors[3\*index] = 0.0;*

*colors[3\*index+1] = Math.random();*

*colors[3\*index+2] = 0.0;*

*} else if (colorVariant == 3) { // Shades of blue*

*colors[3\*index] = 0.0;*

*colors[3\*index+1] = 0.0;*

*colors[3\*index+2] = Math.random();*

*} else if (colorVariant == 4) { // Red to yellow*

*colors[3\*index] = 1.0;*

*colors[3\*index+1] = Math.random();*

*colors[3\*index+2] = 0.0;*

*} else if (colorVariant == 5) { // Green to cyan*

*colors[3\*index] = 0.0;*

*colors[3\*index+1] = 1.0;*

*colors[3\*index+2] = Math.random();*

*} else if (colorVariant == 6) { // Blue to magenta*

*colors[3\*index] = Math.random();*

*colors[3\*index+1] = 0.0;*

*colors[3\*index+2] = 1.0;*

*} else if (colorVariant == 7) { // Shades of gray*

*var gray = Math.random();*

*colors[3\*index] = gray;*

*colors[3\*index+1] = gray;*

*colors[3\*index+2] = gray;*

*} else if (colorVariant == 8) { // Rainbow*

*colors[3\*index] = Math.sin(0.1 \* index + 0) \* 0.5 + 0.5;*

*colors[3\*index+1] = Math.sin(0.1 \* index + 2) \* 0.5 + 0.5;*

*colors[3\*index+2] = Math.sin(0.1 \* index + 4) \* 0.5 + 0.5;*

*} else if (colorVariant == 9) { // Pastel colors*

*colors[3\*index] = Math.random() \* 0.5 + 0.5;*

*colors[3\*index+1] = Math.random() \* 0.5 + 0.5;*

*colors[3\*index+2] = Math.random() \* 0.5 + 0.5;*

*}*

*}*

*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]);*

*}*

*}*

*}*

*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*

*}*

*} else if (key >= 48 && key <= 57) { // '0'-'9' keys*

*pointStyle = key - 48; // set point style (0 for square, 1 for circle, etc.)*

*} else if (key >= 65 && key <= 90) { // 'A'-'Z' keys*

*colorVariant = key - 65; // set color variant (A for 0, B for 1, etc.)*

*for (var i = 0; i < POINT\_COUNT; i++) {*

*setColor(i);*

*}*

*}*

*}*

*function initGL() {*

*var prog = createProgram(gl,"vshader-source", "fshader-source", "a\_coords");*

*gl.useProgram(prog);*

*u\_width\_loc = gl.getUniformLocation(prog,"u\_width");*

*u\_height\_loc = gl.getUniformLocation(prog,"u\_height");*

*u\_pointSize\_loc = gl.getUniformLocation(prog,"u\_pointSize");*

*u\_style\_loc = gl.getUniformLocation(prog, "u\_style");*

*a\_coords\_loc = gl.getAttribLocation(prog,"a\_coords");*

*a\_color\_loc = gl.getAttribLocation(prog, "a\_color");*

*gl.uniform1f(u\_width\_loc, canvas.width);*

*gl.uniform1f(u\_height\_loc, canvas.height);*

*gl.uniform1f(u\_pointSize\_loc, POINT\_SIZE);*

*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.enable(gl.BLEND);*

*gl.blendFunc(gl.SRC\_ALPHA, gl.ONE\_MINUS\_SRC\_ALPHA);*

*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.");*

*}*

*}*

*function createProgram(gl, vertexShaderID, fragmentShaderID, attribute0) {*

*function getTextContent( elementID ) {*

*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;*

*}*

*function frame() {*

*if (isRunning) {*

*updateData();*

*render();*

*requestAnimationFrame(frame); // Arrange for function to be called again*

*}*

*}*

*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();*

*}*

*}*

*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);*

*}*

*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()">*

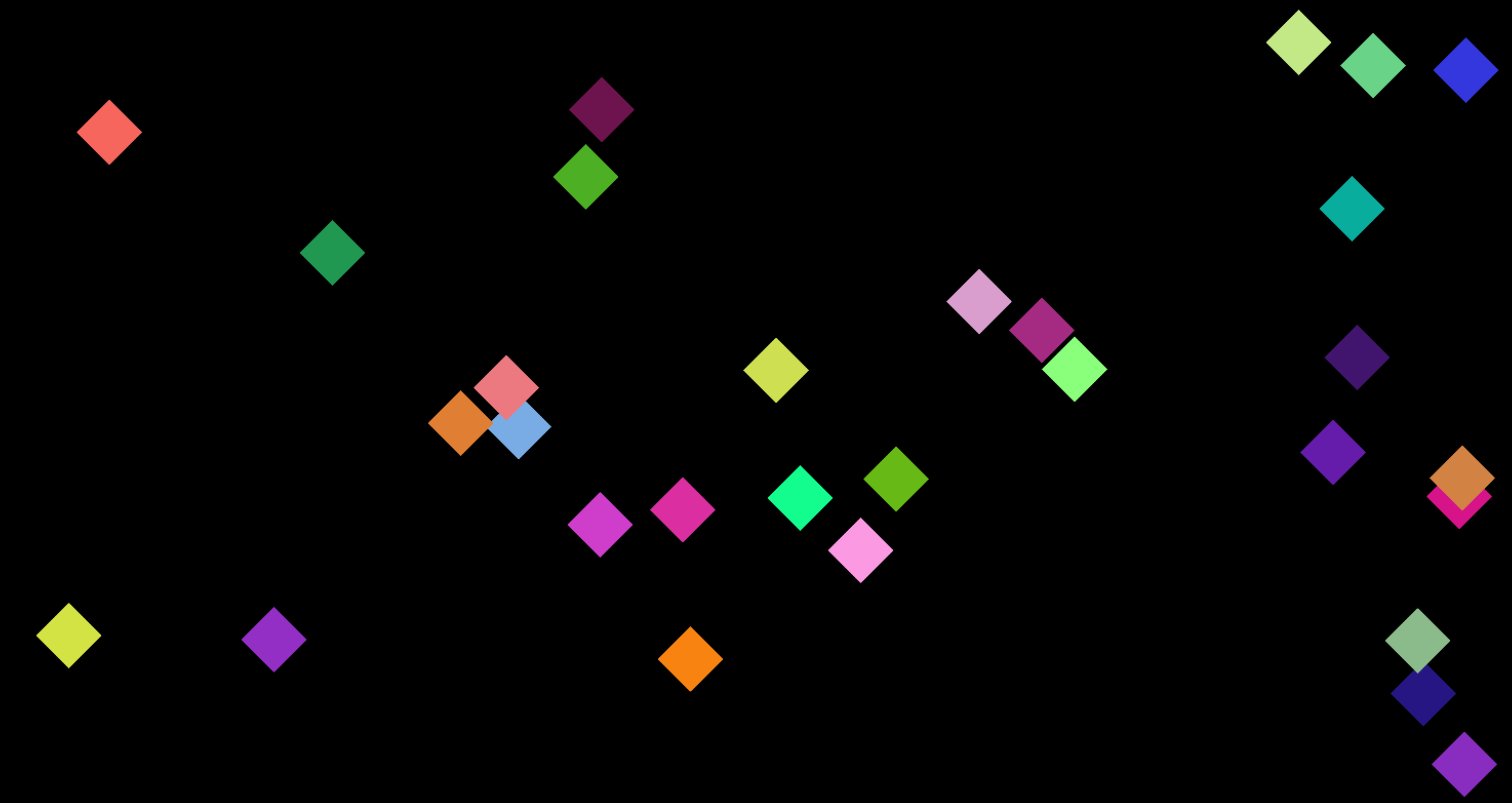
*<noscript>Sorry, this page requires JavaScript.</noscript>*

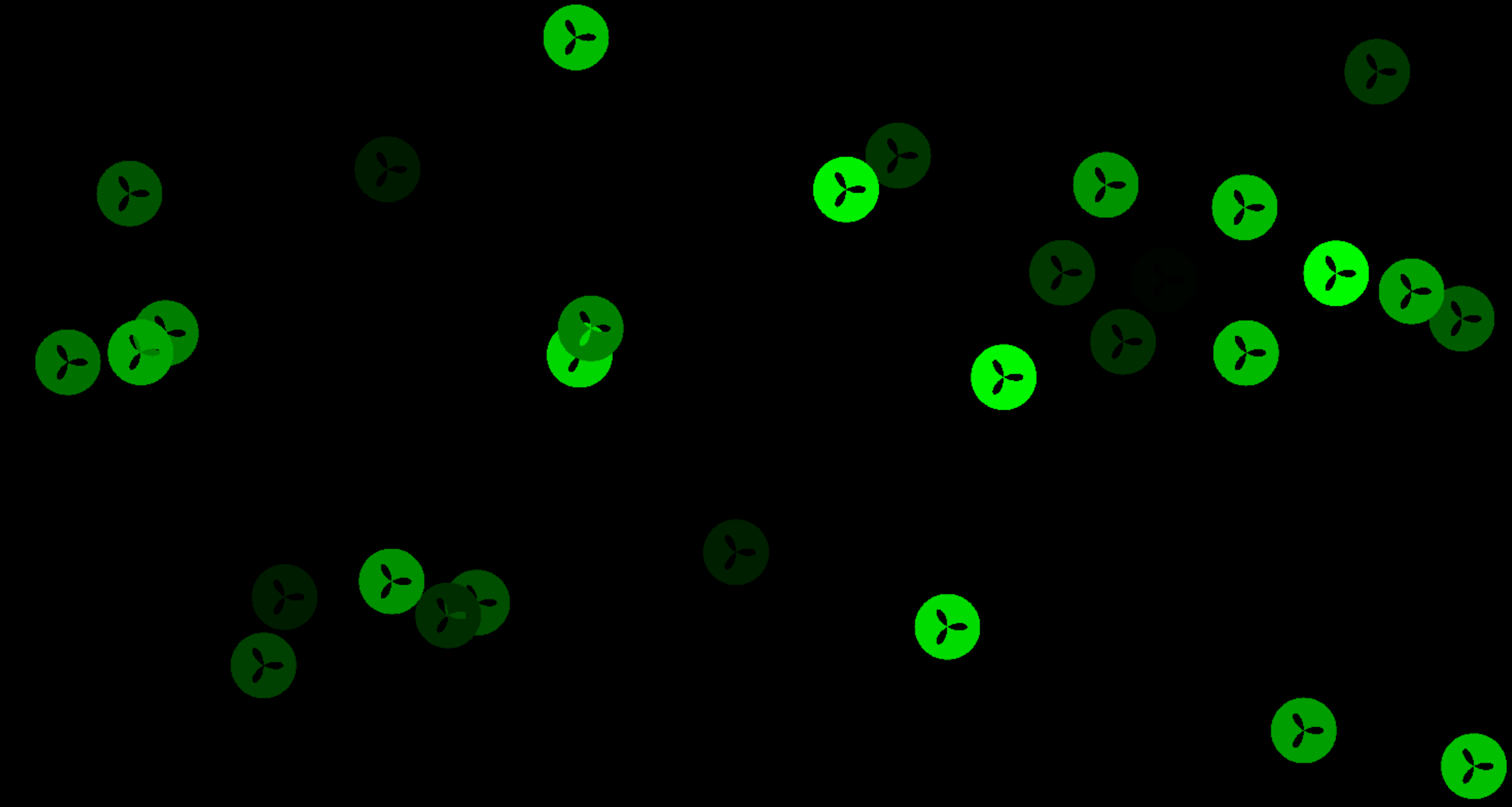
*</body>*

*</html>*

Link do zdalnego repozytorium: https://github.com/Slayzerus/UBB\_GrafikaKomputerowa/tree/main/Lab%2011

**4. Wynik działania:**

****

****

**5. Wnioski:**

Przez bezpośrednie użycie WebGL, uzyskaliśmy pełną kontrolę nad sprzętem graficznym, co pozwala na bardziej zaawansowane i zoptymalizowane wykorzystanie zasobów GPU. Mimo że bezpośrednie programowanie w WebGL jest bardziej skomplikowane niż korzystanie z bibliotek takich jak Three.js, jest to cenne doświadczenie, które wprowadza nas w nowoczesne techniki grafiki komputerowej.

Podczas tego zadania nauczyliśmy się, jak tworzyć kontekst graficzny WebGL oraz jak pisać i kompilować programy shaderów w GLSL, co jest niezbędne dla programowania grafiki. Poznaliśmy strukturę programu WebGL, przepływ danych w potoku przetwarzania i sposób komunikacji między JavaScriptem a shaderami.

Dzięki modyfikacjom programu lab11.html udało się wprowadzić różne style i kolory punktów, które użytkownik może kontrolować za pomocą klawiatury, co zwiększa interaktywność i pokazuje elastyczność WebGL. Te umiejętności i wiedza są podstawą do dalszego zgłębiania grafiki 3D oraz bardziej zaawansowanych technik w przyszłych zajęciach.