Politechnika
Wrocławska

GRAFIKA KOMPUTEROWA

SPRAWOZDANIE DO PROJEKTU

WEBGL – RENDEROWANIE INTERAKTYWNEJ GRAFIKI W PRZEGLĄDARCE INTERNETOWEJ

Autor:

Kamil CIEŚLIK, 226138

Prowadzący:

mgr inż. Szymon DATKO

Termin zajęć:

poniedziałek TP, 08:00 - 11:00

Wrocław, 2018 r.

Spis treści

1. Wprowadzenie

1.1. Cel projektu

Zadanie projektowe polegało na wykonaniu aplikacji internetowej renderującej grafikę 2D oraz 3D przy użyciu skryptów JavaScript w technologii opartej na elemencie HTML5 canvas. Aplikacja posiada następujące funkcjonalności:

- rysowanie kolorowego czworościanu z możliwością obrotu, zmiany rozmiaru oraz zatrzymywania ruchu obiektu w określonej pozycji,
- rysowanie obiektu będącego fraktalem – Dywan Sierpińskiego:
  - możliwość obrotu wg wskazanych osi,
  - losowe kolory,
  - regulowanie stopnia samopodobieństwa,
  - regulacja stopnia deformacji,
  - zmiana rozmiaru,
  - zatrzymywanie ruchu obiektu w określonej pozycji.
- dodatkowo – rysowanie oteksturowanego sześcianu,
- dodatkowo – rysowanie z różniącymi się od siebie kolorami ścian.

1.2. Wstęp teoretyczny

WebGL to interfejs programowania aplikacji języka JavaScript służący do renderowania interaktywnej grafiki 2D oraz 3D poprzez kompatybilną przeglądarkę internetową. WebGL może być wykorzystany do pracy z elementem canvas pozwalającym na dynamiczne, skryptowe renderowanie kształtów i obrazów bitmapowych.
Jedną z większych zalet obiektu canvas jest tworzenie dynamicznych dwu- i trójwymiarowych animacji i gier działających w przeglądarkach bez dodatkowych wtyczek [1].

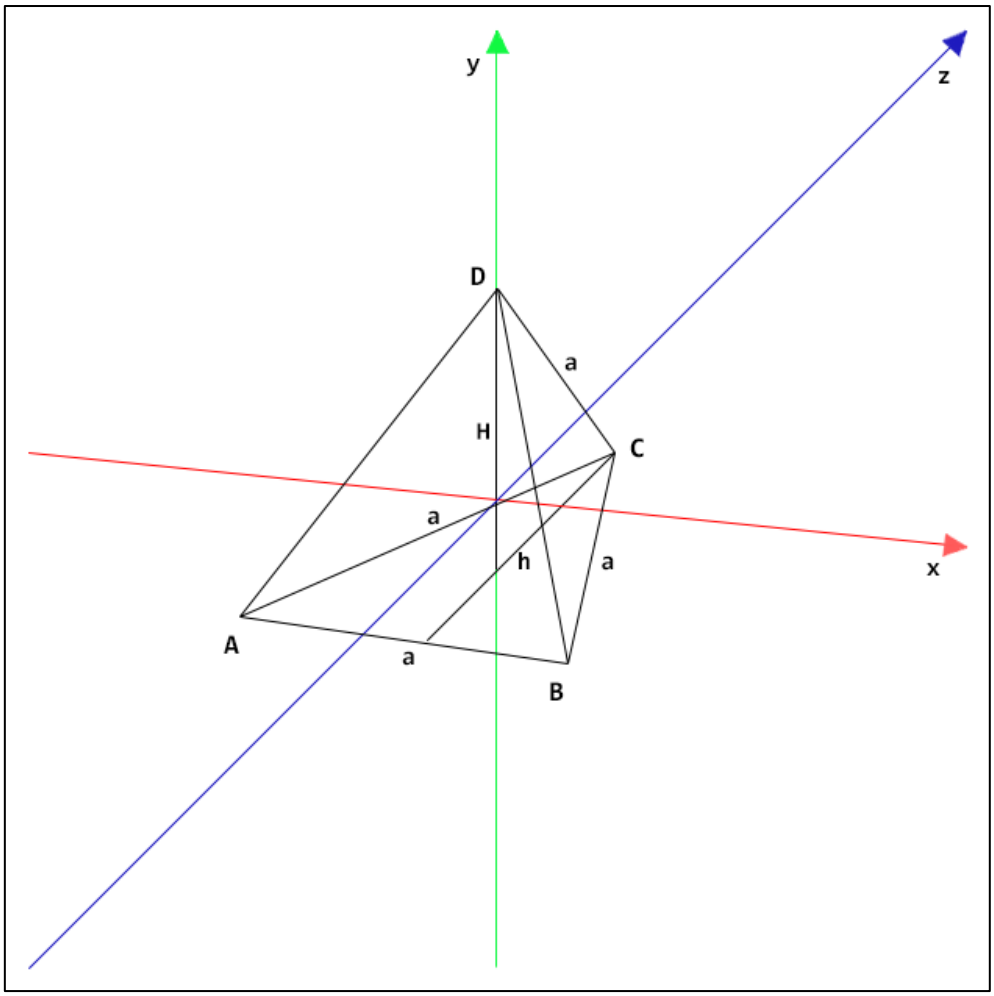
2. Przebieg ćwiczenia

2.1. Czworościan

W celu wyznaczenia uniwersalnej wartości współrzędnych czworościanu foremnego posłużono się wzorem Pitagorasa oraz wzorem na wys. ostrosłupa trójkątnego:

$$a^2 + b^2 = c^2,$$

$$H = a\sqrt{\frac{2}{3}}.$$



Rysunek 1. Przykładowy model ostrosłupa z oznaczeniem wierzchołków, boków oraz wysokości.

$$a = 2$$

$$h^2 + 1^2 = a^2 \rightarrow h = \sqrt{3}$$

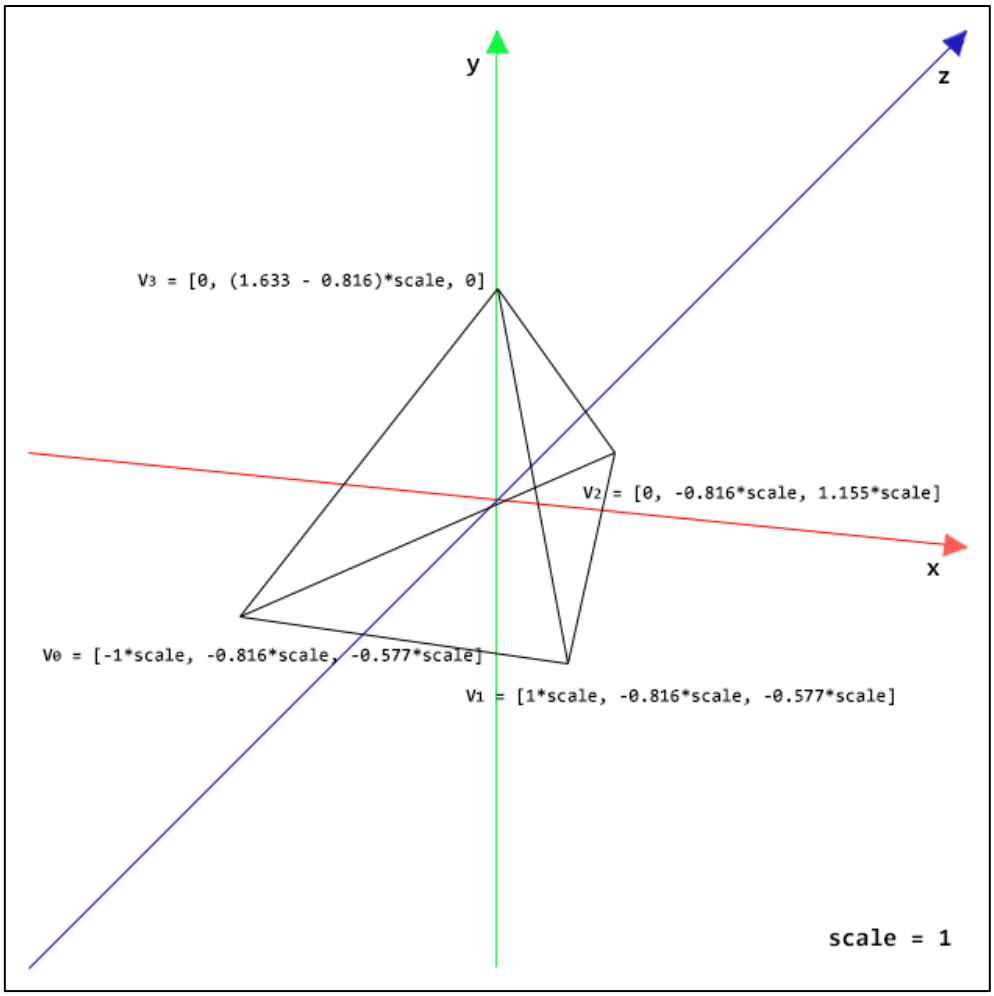
$$H = 2\sqrt{\frac{2}{3}} = \frac{2\sqrt{6}}{3}$$

$$A\left(-1, \frac{H}{2}, \frac{h}{3}\right) \rightarrow A(-1, -0.816, -0.577)$$

$$B\left(1, \frac{H}{2}, \frac{h}{3}\right) \rightarrow B(1, -0.816, -0.577)$$

$$C\left(0, \frac{H}{2}, \frac{2h}{3}\right) \rightarrow C(0, -0.816, 1.155)$$

$$D\left(0, \frac{H}{2}, 0\right) \rightarrow D(0, 0.816, 0)$$



Rysunek 2. Model ostrosłupa z wyznaczonymi współrzędnymi wierzchołków przy współczynniku skalowania = 1.

2.1.1. Najważniejsze fragmenty kodu JavaScript

2.1.1.a. Deklaracja zmiennych

```
var gl_canvas; // Element canvas z plik .html.
var gl_ctx; // Kontekst płotna.
var _position; // Położenie atrybutu uzyskane poprzez funkcję gl_ctx.getAttribLocation().
var _triangleVertexBuffer; // Bufor wierzchołków.
var _triangleFacesBuffer; // Bufor indeksów.
var _color;
var _PosMatrix; // Zmienna jednolita w shaderze.
var _MovMatrix; // Zmienna jednolita w shaderze.
var _ViewMatrix; // Zmienna jednolita w shaderze.
var _matrixProjection; // Wynik funkcji getProjection().
var _matrixMovement; // Macierz jednostkowa o wym. 4x4 zwrócona przez getIdentityMatrix().
var _matrixView; // Macierz jednostkowa o wym. 4x4 zwrócona przez getIdentityMatrix().
var animation; // Identyfikator żądania zwracany przez funkcję requestAnimationFrame().
var pause_animation; // Wstrzymywanie ruchu obiektu.
var dAngle; // Kąt obrotu obiektu.
var scale = 1; // Współczynnik skalowania obiektu (zmiana współrzędnych rysowania).
var rotationSpeed = 0.001;
var zoomRatio = -6;
var X, Y, Z; // Wartości obrotu wg osi: X, Y, Z
```

Kod programu 1. Początek skryptu: deklaracja zmiennych, inicjalizacja współczynnika skalowania, prędkości obrotu obiektu.

2.1.1.b. Funkcja główna

```
function runWebGL() {
    // Po wywołaniu funkcji wprawiającej obiekt w ruch zatrzymanie ruchu (jeżeli true)
    // przestanie obowiązywać.
    pause_animation = false;
    getRotation(); // Osie obrotu.
    gl_canvas = document.getElementById("glcanvas"); // Pobranie obiektu canvas.
    gl_ctx = gl_getContext(gl_canvas);
    gl_initShaders(); // Inicjalizacja shaderów.
    gl_initBuffers(); // Inicjalizacja buforów.
    gl_setMatrix(); // Inicjalizacja wartości zmiennych _matrixProjection...
    gl_draw(); // Renderowanie obiektu.
}
```

Kod programu 2. Główna funkcja skryptu runWebGL() - wywoływana w momencie zdarzenia kliknięcia przycisku „Uruchom”.

2.1.1.c. Funkcje zmiany parametrów programu wywoływane z poziomu UI

Funkcje związane z obsługą zdarzeń komponentów typu:

1. checkbox – getRotation(),
2. button – pauseAnimation(),
3. button – startedPosition(),
4. slider – changeScale().

```
// Osie obrotu.
function getRotation() {
    X = document.getElementById("rotateX").checked;
    Y = document.getElementById("rotateY").checked;
    Z = document.getElementById("rotateZ").checked;
}

// Zmiana zmiennej odpowiadającej za wstrzymywanie ruchu obiektu.
function pauseAnimation() {
    pause_animation = !pause_animation;
}

// Odznaczenie ruchu wokół osi obrotu i powrót obiektu do statycznej pozycji startowej.
function startedPosition() {
    X = document.getElementById('rotateX').checked = false;
    Y = document.getElementById('rotateY').checked = false;
    Z = document.getElementById('rotateZ').checked = false;
    runWebGL();
}

// Zmiana długości boku ostrosłupa foremnego.
function changeScale() {
    scale = document.getElementById('scaleRange').value/100;
    runWebGL();
}
```

Kod programu 3. Funkcje zmiany parametrów programu wywoływane z poziomu UI.

2.1.1.d. Pobranie kontekstu WebGL

Funkcja zwraca kontekst "webgl". Dodatkowo ustawia szerokość i wysokość kontekstu w oparciu o wartości tych parametrów w elemencie canvas.

```
function gl_getContext(canvas) {
    try {
        var ctx = canvas.getContext("webgl") || canvas.getContext("experimental-webgl");
        ctx.viewportWidth = canvas.width;
        ctx.viewportHeight = canvas.height;
    } catch (e) {
    }

    if (!ctx) {
        document.write("Nieudana inicjalizacja kontekstu WebGL.");
    }
    return ctx;
}
```

Kod programu 4. Funkcja gl_getContext() - pobranie kontekstu WebGL.

2.1.1.e. Inicjalizacja shader'ów

Shadery potrzebne są do wyrenderowania trójwymiarowego obiektu. W języku GLSL wyróżnia się dwa typy shaderów: wierzchołków oraz fragmentów. Oba rodzaje shaderów charakteryzują się tą samą strukturą.

```
function gl_initShaders() {
    var vertexShader = "\n\
      attribute vec3 position;\n\
      uniform mat4 PosMatrix;\n\
      uniform mat4 MovMatrix;\n\
      uniform mat4 ViewMatrix; \n\
      attribute vec3 color;\n\
      varying vec3 vColor;\n\
      void main(void) {\n\ // Zapis przetransformowanych wsp. wierzch. w gl_Position.
         gl_Position = PosMatrix * ViewMatrix * MovMatrix * vec4(position, 1.);\n\
         vColor = color;\n\
      }";

    var fragmentShader = "\n\
      precision mediump float;\n\
      varying vec3 vColor;\n\
      void main(void) {\n\ // Zapis shader'ów fragmentów w gl_FragColor.
         gl_FragColor = vec4(vColor, 1.);\n\
      }";

    var getShader = function (source, type, typeString) {
        var shader = gl_ctx.createShader(type);
        gl_ctx.shaderSource(shader, source);
        gl_ctx.compileShader(shader);

        if (!gl_ctx.getShaderParameter(shader, gl_ctx.COMPILE_STATUS)) {
            alert("error in" + typeString);
            return false;
        }
        return shader;
    };
    var shader_vertex = getShader(vertexShader, gl_ctx.VERTEX_SHADER, "VERTEX");
    var shader_fragment = getShader(fragmentShader, gl_ctx.FRAGMENT_SHADER, "FRAGMENT");
    var SHADER_PROGRAM = gl_ctx.createProgram();
    gl_ctx.attachShader(SHADER_PROGRAM, shader_vertex);
    gl_ctx.attachShader(SHADER_PROGRAM, shader_fragment);
    gl_ctx.linkProgram(SHADER_PROGRAM);
    _PosMatrix = gl_ctx.getUniformLocation(SHADER_PROGRAM, "PosMatrix");
    _MovMatrix = gl_ctx.getUniformLocation(SHADER_PROGRAM, "MovMatrix");
    _ViewMatrix = gl_ctx.getUniformLocation(SHADER_PROGRAM, "ViewMatrix");
    _position = gl_ctx.getAttribLocation(SHADER_PROGRAM, "position");
    _color = gl_ctx.getAttribLocation(SHADER_PROGRAM, "color");
    gl_ctx.enableVertexAttribArray(_position);
    gl_ctx.enableVertexAttribArray(_color);
    gl_ctx.useProgram(SHADER_PROGRAM);
}
```

Kod programu 5. Funkcja gl_initShaders() – inicjalizacja shader'ów.

2.1.1.f. Inicjalizacja buforów wierzchołków oraz indeksów

WebGL przechowuje dane wierzchołków w obiektach buforów. Bufor taki jest wypełniany wartościami. Gdy rozpocznie się renderowanie obiektów 3D, zostaną one przesłane do GPU. Za każdym razem gdy zmienione zostaną dowolne dane wierzchołków, należy załadować je do bufora.

```
function gl_initBuffers() {
    // Współrzędne wierzchołków ostrosłupa foremnego wyliczone
    // za pomocą wzorów Pitagorasa i wzoru na wysokość ostrosłupa.
    // Możliwość zmiany rozmiaru obiektu za pomocą współczynnika skalowania.
    var triangleVertices = [
        -1 * scale, -0.816 * scale, -0.577 * scale,
        0, 0, 1, // Kolor niebieski.
        1 * scale, -0.816 * scale, -0.577 * scale,
        0, 1, 0, // Kolor zielony,
        0, -0.816 * scale, 1.155 * scale,
        1, 0, 0, // Kolor czerwony.
        0, (1.633 - 0.816) * scale, 0,
        0.5, 0.5, 0.5 // Kolor szary.
    ];

    _triangleVertexBuffer = gl_ctx.createBuffer();
    gl_ctx.bindBuffer(gl_ctx.ARRAY_BUFFER, _triangleVertexBuffer);
    gl_ctx.bufferData(gl_ctx.ARRAY_BUFFER, new Float32Array(triangleVertices),
gl_ctx.STATIC_DRAW);

    // Połączenie wierzchołków poprzez podanie ich indeksów.
    var triangleFaces = [
        0, 1, 2, // Podstawa.
        0, 1, 3, // Ściana 1.
        1, 2, 3, // Ściana 2.
        2, 0, 3  // Ściana 3.
    ];

    _triangleFacesBuffer = gl_ctx.createBuffer();
    gl_ctx.bindBuffer(gl_ctx.ELEMENT_ARRAY_BUFFER, _triangleFacesBuffer);
    gl_ctx.bufferData(gl_ctx.ELEMENT_ARRAY_BUFFER, new Uint16Array(triangleFaces),
gl_ctx.STATIC_DRAW);
}
```

Kod programu 6. Funkcja gl_initBuffers() – inicjalizacja buforów wierzchołków oraz indeksów.

2.1.1.g. Inicjalizacja wartości zmiennych odpowiadających za operacje obrotu obiektu: _matrixProjection, _matrixMovement, _matrixView

```
function gl_setMatrix() {
    _matrixProjection = MATRIX.getProjection(40, gl_canvas.width / gl_canvas.height, 1,
      100);
    _matrixMovement = MATRIX.getIdentityMatrix();
    _matrixView = MATRIX.getIdentityMatrix();

    MATRIX.translateZ(_matrixView, zoomRatio);
}
```

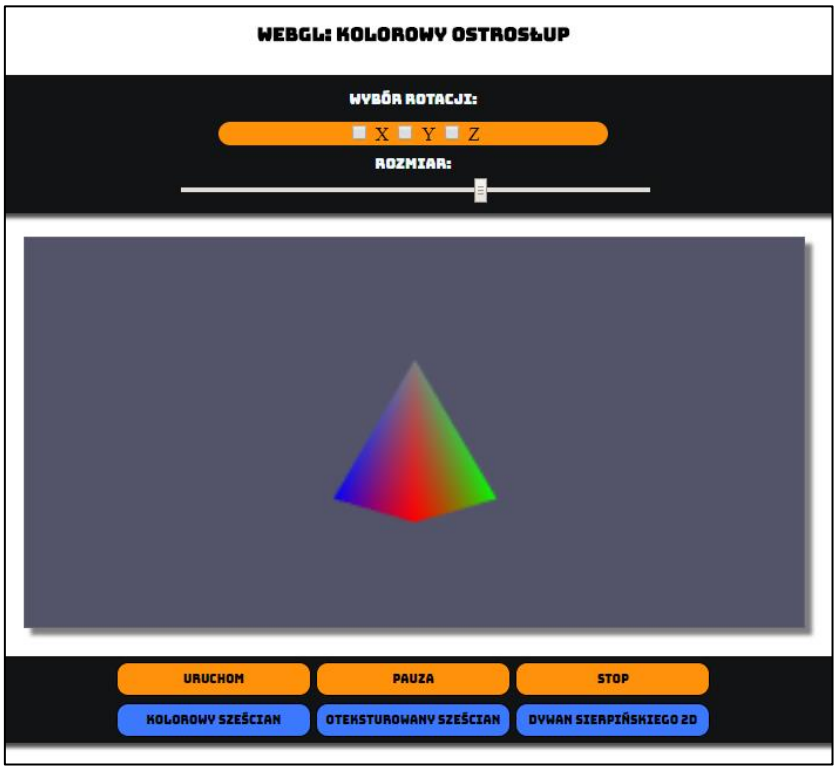
Kod programu 7. Funkcja gl_setMatrix() - inicjalizacja wartości zmiennych odpowiadających za operacje obrotu obiektu: _matrixProjection, _matrixMovement, _matrixView

2.1.1.h. Funkcja rysowania obiektu

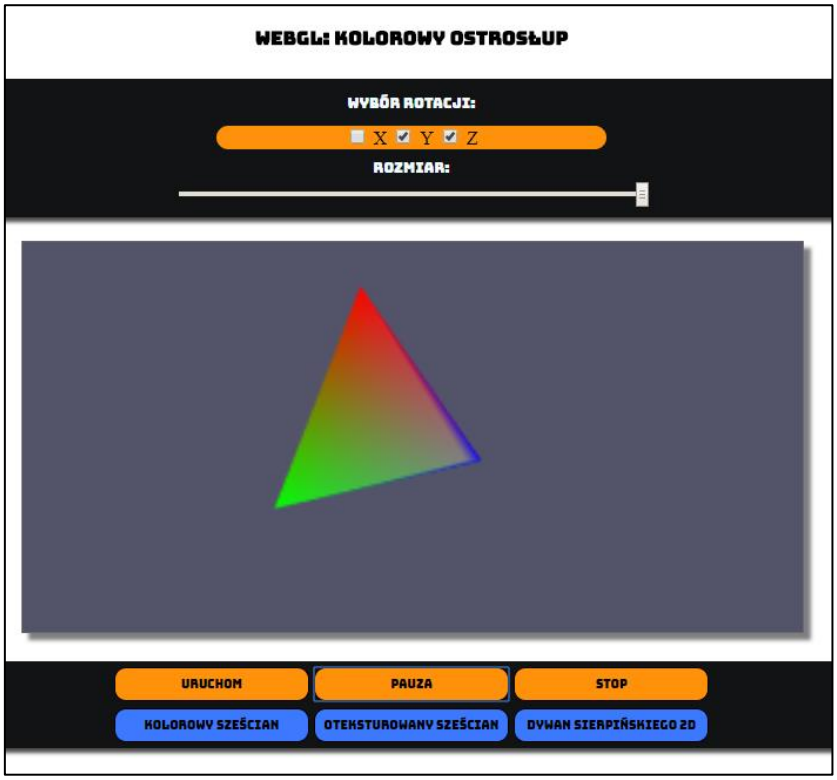
```
function gl_draw() {
    // Warunek powodujący, że prędkość obrotu obiektu jest zawsze jednakowa.
    // Anulowanie żądania wywołania funkcji do odświeżania w przypadku, kiedy
    // identyfikator żądania przyjmuje wartość inną niż 0 - wystąpiło odświeżenie anim.
    if (animation)
        window.cancelAnimationFrame(animation);

    gl_ctx.clearColor(0.0, 0.0, 0.0, 0.0);
    gl_ctx.enable(gl_ctx.DEPTH_TEST);
    gl_ctx.depthFunc(gl_ctx.LEQUAL);
    gl_ctx.clearDepth(1.0);
    var timeOld = 0;

    var animate = function (time) {
        dAngle = rotationSpeed * (time - timeOld);

        // Warunek odpowiadający za wykonanie obrotu - możliwość wstrzymania poruszania
        // obiektu po kliknięciu w odpowiedni przycisk.
        if (!pause_animation) {
            if (X)
                MATRIX.rotateX(_matrixMovement, dAngle);
            if (Y)
                MATRIX.rotateY(_matrixMovement, dAngle);
            if (Z)
                MATRIX.rotateZ(_matrixMovement, dAngle);
        }
        timeOld = time;

        gl_ctx.viewport(0.0, 0.0, gl_canvas.width, gl_canvas.height);
        gl_ctx.clear(gl_ctx.COLOR_BUFFER_BIT | gl_ctx.DEPTH_BUFFER_BIT);

        gl_ctx.uniformMatrix4fv(_PosMatrix, false, _matrixProjection);
        gl_ctx.uniformMatrix4fv(_MovMatrix, false, _matrixMovement);
        gl_ctx.uniformMatrix4fv(_ViewMatrix, false, _matrixView);

        // Ostrosłup - 3 współrzędne jednego wierzchołka, 3 współrzędne kolorów każdego
        // wierzchołka (RGB). Konieczność zmiany parametrów odchylenia - należy
        // rozpoznawać odpowiednie elementy tablicy triangleVertices jako współrzędne
        // wierzchołków a inne jako składowe kolorów.
        gl_ctx.vertexAttribPointer(_position, 3, gl_ctx.FLOAT, false, 4 * (3 + 3), 0);
        gl_ctx.vertexAttribPointer(_color, 3, gl_ctx.FLOAT, false, 4 * (3 + 3), 3 * 4);

        gl_ctx.bindBuffer(gl_ctx.ARRAY_BUFFER, _triangleVertexBuffer);
        gl_ctx.bindBuffer(gl_ctx.ELEMENT_ARRAY_BUFFER, _triangleFacesBuffer);

        // Ilość wierzchołków przypadająca na każdy poligon * ilość poligonów = 3 * 4.
        gl_ctx.drawElements(gl_ctx.TRIANGLES, 4 * 3, gl_ctx.UNSIGNED_SHORT, 0);
        gl_ctx.flush();

        // animate - funkcja do wywołania, gdy nadchodzi czas odświeżenia animacji przed
        // kolejnym odmalowaniem. animation - zwrócony przez funkcję parametr
        // identyfikatora żądania - niezerowa wartość.
        animation = window.requestAnimationFrame(animate);
    };

    animate(0);
}
```

Kod programu 8. Funkcja gl_draw() – rysowanie obiektu.

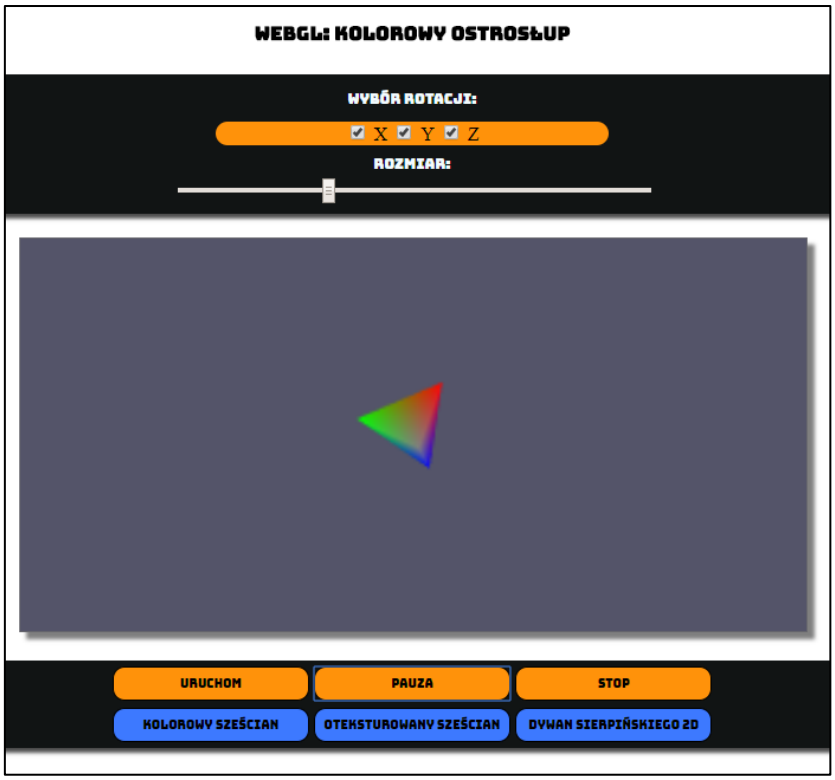
2.1.2. Zrzuty ekranu prezentujące działanie aplikacji



Rysunek 3. Zrzut ekranu prezentujący model ostrosłupa w początkowym statycznym położeniu.



Rysunek 4. Zrzut ekranu prezentujący powiększony model ostrosłupa wprawionego w ruch wg osi x oraz z.



Rysunek 5. Zrzut ekranu prezentujący pomniejszony model ostrosłupa wprawionego w ruch wg osi x, y oraz z.

2.2. Dwuwymiarowy Dywan Sierpińskiego

Rysowanie Dywanu Sierpińskiego odbywa się za pomocą funkcji rekurencyjnej wyliczającej kolejne współrzędne wierzchołków, dodającej je do tablicy wierzchołków oraz łączącej odpowiednie indeksy wierzchołków w trójkąty składowe każdego kwadratu.

2.2.1. Najważniejsze fragmenty kodu JavaScript

W poniższych listingach udokumentowane i omówione zostaną tylko części skryptu, który uległ istostnym zmianom względem kodu dotyczącego czworościanu.

2.2.1.a. Deklaracja wraz z inicjalizacją zmiennych

```
...
var elements = 0; // Ilość indeksów wierzchołków we wszystkich poligonach.
var vertex = 0; // Ilość wierzchołków.

var recursion = 3; // Stopień samopodobieństwa - rekurencji fraktalu.
var deformation = 0; // Stopień deformacji fraktalu.
var length = 4; // Długość boku fraktalu.
...
```

Kod programu 9. Początek skryptu: deklaracja wraz z inicjalizacją zmiennych.

2.2.1.b. Funkcja główna

```
function runWebGL() {
    // Jeżeli tablice wierzchołków zawierają wcześniej utworzone elementy
    // to następuje ich zerowanie. Zapobiega to nakładaniu się kolejnych warstw
    // składowych Dywanu Sierpińskiego na siebie po zmianie parametrów rysowania.
    while (triangleVertices.length > 0)
        triangleVertices.pop();
    while (triangleFaces.length > 0)
        triangleFaces.pop();
    elements = 0;
    vertex = 0;
    // Po wywołaniu funkcji wprawiającej obiekt w ruch zatrzymanie ruchu (jeżeli true)
    // przestanie obowiązywać.
    pause_animation = false;
    getRotation();
    gl_canvas = document.getElementById("glcanvas");
    gl_ctx = gl_getContext(gl_canvas);
    gl_initShaders();
    // Funkcja rekurencyjna obliczająca współrzędne kolejnych wierzchołków oraz
    // uzupełniająca tablice trangleVertices i triangleFaces.
    DrawSierpinskiCarpet(length / 2, length / 2, length, recursion);
    gl_initBuffers();
    gl_setMatrix();
    gl_draw();
}
```

Kod programu 10. Główna funkcja skryptu runWebGL() - wywoływana w momencie zdarzenia kliknięcia przycisku „Uruchom”.

2.2.1.c. Funkcje zmiany parametrów programu wywoływane z poziomu UI

Funkcje związane z obsługą zdarzeń komponentów typu:

1. combobox – changeRecursion(),
2. slider – changeDeformation().

```
...
// Zmiana stopnia samopodobieństwa - rekurencji Dywanu Sierpińskiego.
function changeRecursion() {
    recursion = document.getElementById("recursion-
      level").options[document.getElementById("recursion-level").selectedIndex].value;
    runWebGL();
}

// Zmiana stopnia deformacji.
function changeDeformation() {
    deformation = document.getElementById('deformationRange').value / 100;
    runWebGL();
}
...
```

Kod programu 11. Funkcje zmiany parametrów programu wywoływane z poziomu UI.

2.2.1.d. Funkcja rekurencyjna wyliczająca współrzędne składowych fraktalu

```
// Funkcja rekurencyjna wyliczająca współrzędne składowych Dywanu Sierpińskiego:
// - współrzędne (x, y), długość boku dywanu, poziom samopodobieństwa - rekurencji.
function DrawSierpinskiCarpet(x, y, _lengthOfSide, _recursionLevel) {
    if (_recursionLevel > 0) {
        _lengthOfSide = _lengthOfSide / 3;
        _recursionLevel -= 1;
        DrawSierpinskiCarpet(x - (2 * _lengthOfSide), y, _lengthOfSide, _recursionLevel);
        DrawSierpinskiCarpet(x - _lengthOfSide, y, _lengthOfSide, _recursionLevel);
        DrawSierpinskiCarpet(x, y, _lengthOfSide, _recursionLevel);

        DrawSierpinskiCarpet(x - (2 * _lengthOfSide), y - _lengthOfSide, _lengthOfSide,
           _recursionLevel);
        DrawSierpinskiCarpet(x, y - _lengthOfSide, _lengthOfSide, _recursionLevel);

        DrawSierpinskiCarpet(x - (2 * _lengthOfSide), y - (2 * _lengthOfSide),
           _lengthOfSide, _recursionLevel);
        DrawSierpinskiCarpet(x - _lengthOfSide, y - (2 * _lengthOfSide), _lengthOfSide,
           _recursionLevel);
        DrawSierpinskiCarpet(x, y - (2 * _lengthOfSide), _lengthOfSide, _recursionLevel);
    }
    else {
        // Wyliczenie właściwego przesunięcia deformacji.
        var def;
        var plus_or_minus;
        if (deformation !== 0)
            def = (Math.random() % (0.1 * length)) * (deformation * 0.1);
        else
            def = 0;

        // Sprawienie, że przesunięcie w lewo i prawo występują z tym samym
        // prawdopodobieństwem.
        plus_or_minus = Math.random();
        if (plus_or_minus < 0.5)
            def = -def;

        // Prawy-górny wierzchołek kwadratu.
        triangleVertices.push(x + def);
        triangleVertices.push(y + def);
        triangleVertices.push(Math.random());
        triangleVertices.push(Math.random());
        triangleVertices.push(Math.random());
        if (vertex !== 0)
            vertex = vertex + 1;

        // Prawy-dolny wierzchołek kwadratu.
        triangleVertices.push(x + def);
        triangleVertices.push(y - _lengthOfSide + def);
        triangleVertices.push(Math.random());
        triangleVertices.push(Math.random());
        triangleVertices.push(Math.random());
        vertex = vertex + 1;

        // Lewy-dolny wierzchołek kwadratu.
        triangleVertices.push(x - _lengthOfSide + def);
        triangleVertices.push(y - _lengthOfSide + def);
        triangleVertices.push(Math.random());
        triangleVertices.push(Math.random());
        triangleVertices.push(Math.random());
        vertex = vertex + 1;

        // Lewy-górny wierzchołek kwadratu.
        triangleVertices.push(x - _lengthOfSide + def);
        triangleVertices.push(y + def);
        triangleVertices.push(Math.random());
        triangleVertices.push(Math.random());
        triangleVertices.push(Math.random());
        vertex = vertex + 1;
```

Kod programu 12. Funkcja DrawSierpinskiCarpet() – wyliczająca współrzędne składowych Dywanu Sierpińskiego, część 1/2.

```
// Pierwsza połowa kwadratu (trójkąt po stronie prawej).
        triangleFaces.push(vertex - 3);
        triangleFaces.push(vertex - 2);
        triangleFaces.push(vertex - 1);

        // Druga połowa kwadratu (trójkąt po stronie lewej).
        triangleFaces.push(vertex - 3);
        triangleFaces.push(vertex - 1);
        triangleFaces.push(vertex);

        // Powiększenie ilości elementów bufora indeksów.
        elements = elements + 6;
    }
}
```

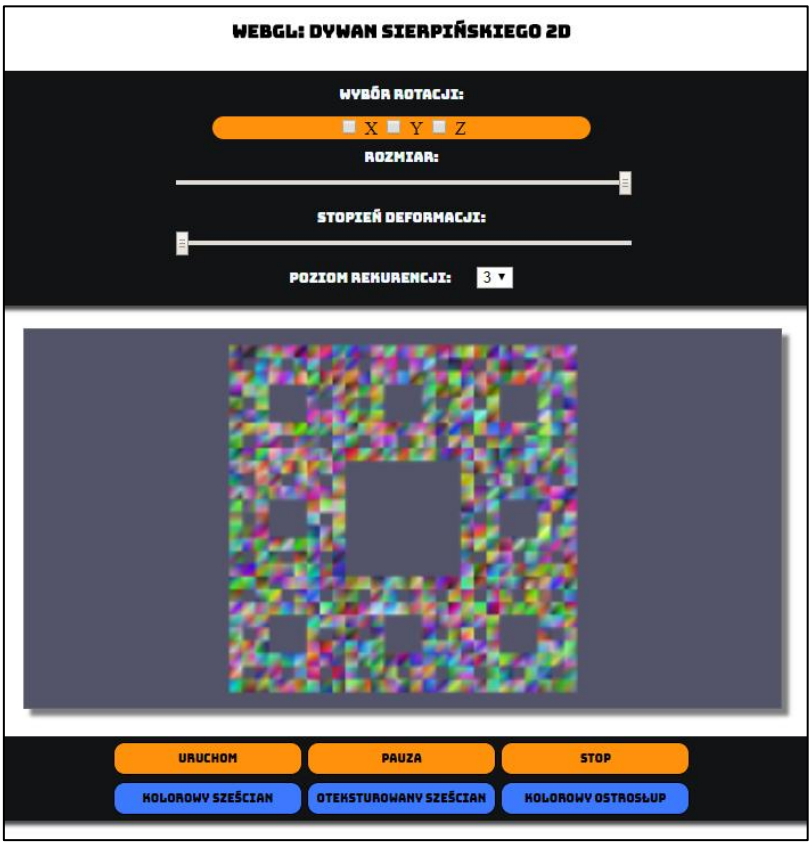
Kod programu 13. Funkcja DrawSierpinskiCarpet() – wyliczająca współrzędne składowych Dywanu Sierpińskiego, część 2/2.

2.2.1.e. Funkcja rysowania obiektu

```
function gl_draw() {
        ...
        // Fraktal 2D - 2 współrzędne jednego wierzchołka, 3 współrzędne kolorów każdego
        // wierzchołka (RGB). Konieczność zmiany parametrów odchylenia - należy
        // rozpoznawać odpowiednie elementy tablicy triangleVertices jako współrzędne
        // wierzchołków a inne jako składowe kolorów.
        gl_ctx.vertexAttribPointer(_position, 2, gl_ctx.FLOAT, false, 4 * (2 + 3), 0);
        gl_ctx.vertexAttribPointer(_color, 3, gl_ctx.FLOAT, false, 4 * (2 + 3), 2 * 4);

        gl_ctx.bindBuffer(gl_ctx.ARRAY_BUFFER, _triangleVertexBuffer);
        gl_ctx.bindBuffer(gl_ctx.ELEMENT_ARRAY_BUFFER, _triangleFacesBuffer);

        // Ilość wierzchołków przypadająca na każdy poligon * ilość poligonów - zmienna
        // elements inkrementowana podczas wykonywania funkcji rekurencyjnej obliczającej
        // kolejne współrzędne wierzchołków fraktalu.
        gl_ctx.drawElements(gl_ctx.TRIANGLES, elements, gl_ctx.UNSIGNED_SHORT, 0);
        gl_ctx.flush();
        animation = window.requestAnimationFrame(animate);
    };

    animate(0);
}
```

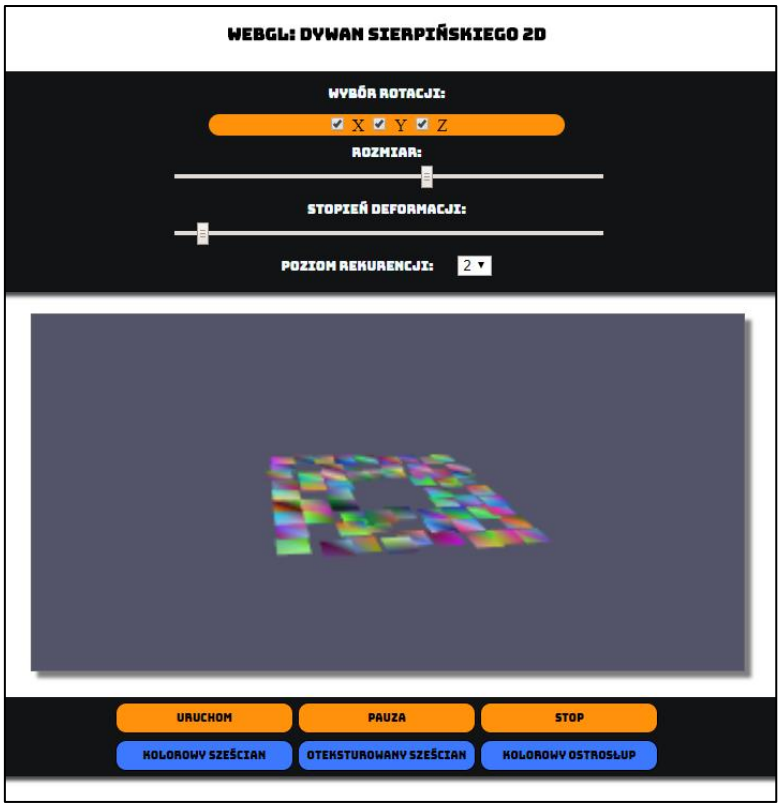
Kod programu 14. Funkcja gl_draw() – rysowanie obiektu.

2.2.2. Zrzuty ekranu prezentujące działanie aplikacji



Rysunek 6. Zrzut ekranu prezentujący dwuwymiarowy Dywan Sierpińskiego w początkowym statycznym położeniu.

Rysunek 7. Zrzut ekranu prezentujący dwuwymiarowy Dywan Sierpińskiego w początkowym statycznym położeniu po zmiane stopnia rekurencji oraz stopnia deformacji.



Rysunek 8. Zrzut ekranu prezentujący dwuwymiarowy Dywan Sierpińskiego wprawiony w ruch wg osi x, y oraz z po zmiane stopnia rekurencji oraz stopnia deformacji.

2.3. Dodatek - teksturowanie obiektu

Zaimplementowano dodatkową figurę – sześcian bez jednej ściany, będący imitacją prostego otwartego pudełka.

2.3.1. Najważniejsze fragmenty kodu JavaScript

2.3.1.a. Inicjalizacja shader'ów.

```
function gl_initShaders() {
    var vertexShader = "\n\
        attribute vec3 position;\n\
        uniform mat4 PosMatrix;\n\
        uniform mat4 MovMatrix;\n\
        uniform mat4 ViewMatrix;\n\
        attribute vec2 uv;\n\
        varying vec2 vUV;\n\
        void main(void) {\n\ // Zapis przetransformowanych wsp. wierzch. w gl_Position.
          gl_Position = PosMatrix * ViewMatrix * MovMatrix * vec4(position, 1.);\n\
          vUV = uv;\n\}";

    var fragmentShader = "\n\
        precision mediump float;\n\
        uniform sampler2D sampler;\n\
        varying vec2 vUV;\n\ // Zapis shader'ów fragmentów w gl FragColor.
      void main(void) {\n\ // Funkcja  texture2D() - pobierająca próbki z przekazanego..
            gl_FragColor = texture2D(sampler, vUV);\n\}"; // .. jej obrazu.
    ...
```

Kod programu 15. Funkcja gl_initShaders() – inicjalizacja shader'ów.

2.3.1.b. Załadowanie obrazu z teksturą

Obiekt tekstury musi zostać dowiązany do docelowego obiektu – gl_ctx.TEXTURE_2D, aby środowisko WebGL mogło ustalić, jak go wykorzystać.

```
function gl_initTexture() {
    var img = new Image();

    if (texture1)
        img.src = '../texture/cubetexture1.png';
    if (texture2)
        img.src = '../texture/cubetexture2.png';
    if (texture3)
        img.src = '../texture/cubetexture3.png';

    img.webglTexture = false;

    img.onload = function (e) {
        var texture = gl_ctx.createTexture();
        gl_ctx.pixelStorei(gl_ctx.UNPACK_FLIP_Y_WEBGL, true);
        gl_ctx.bindTexture(gl_ctx.TEXTURE_2D, texture);
        gl_ctx.texParameteri(gl_ctx.TEXTURE_2D, gl_ctx.TEXTURE_MIN_FILTER, gl_ctx.LINEAR);
        gl_ctx.texParameteri(gl_ctx.TEXTURE_2D, gl_ctx.TEXTURE_MAG_FILTER, gl_ctx.LINEAR);
        gl_ctx.texImage2D(gl_ctx.TEXTURE_2D, 0, gl_ctx.RGBA, gl_ctx.RGBA,
            gl_ctx.UNSIGNED_BYTE, img);
        gl_ctx.bindTexture(gl_ctx.TEXTURE_2D, null);
        img.webglTexture = texture;
    };
    return img;
}
```

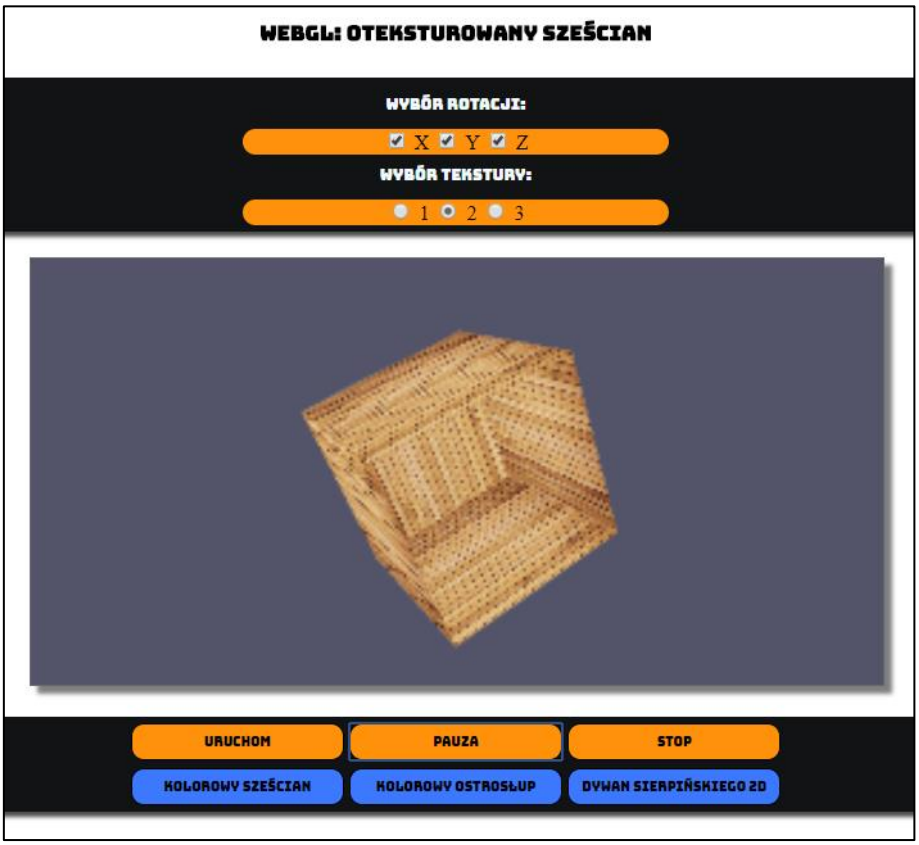
Kod programu 16. Funkcja gl_initTexture() – załadowanie obrazu z teksturą.

2.4. Funkcja rysowania obiektu

```
function gl_draw() {
        ...
        gl_ctx.vertexAttribPointer(_position, 3, gl_ctx.FLOAT, false, 4 * (3 + 2), 0);
        gl_ctx.vertexAttribPointer(_uv, 2, gl_ctx.FLOAT, false, 4 * (3 + 2), 3 * 4);
        ...
}
```

Kod programu 17. Funkcja gl_draw() - rysowanie obiektu.

2.4.1. Zrzuty ekranu prezentujące działanie aplikacji



Rysunek 9. Zrzut ekranu prezentujący oteksturowany otwarty sześcian wprawiony w ruch wg osi x, y.

2.5. Szablon dokumentu HTML

Szablon dokumentu HTML na przykładzie implementacji widoku strony dla Dywanu Sierpińskiego.

```
<!DOCTYPE html>
<html lang="pl-PL">
<head>
    <title>WebGL</title>
    <meta charset="UTF-8">
    <link rel="stylesheet" href="../css/style.css">
    <link href='https://fonts.googleapis.com/css?family=Bungee' rel='stylesheet'>
</head>

<body onload="runWebGL()">
<h2 class="heading-title">WebGL: Dywan Sierpińskiego 2D</h2>

<div class="configuration-block">
    <div>
        <label class="standard-label">WYBÓR ROTACJI:</label>
        <form class="rotate-buttons-form">
            <label for="rotateX"></label><input type="checkbox" id="rotateX"> X
            <label for="rotateY"></label><input type="checkbox" id="rotateY"> Y
            <label for="rotateZ"></label><input type="checkbox" id="rotateZ"> Z
        </form>
    </div>

    <label class="standard-label" for="scaleRange">ROZMIAR:</label><input type="range"
          min="100" max="400" value="400" class="slider" oninput="changeScale()"
          id="scaleRange">

    <label class="standard-label" for="deformationRange">STOPIEŃ DEFORMACJI:</label><input
          type="range" min="0" max="5000" value="0" class="slider"
          oninput="changeDeformation()" id="deformationRange">

    <label class="standard-label"> POZIOM REKURENCJI:
        <select class="combo-box" id="recursion-level" oninput="changeRecursion()">
            <option value=0>0</option>
            <option value=1>1</option>
            <option value=2>2</option>
            <option value=3 selected>3</option>
            <option value=4>4</option>
        </select>
    </label>
</div>

<canvas id="glcanvas">
    Brak wsparcia dla elementu HTML5 typu canvas
</canvas>

<div class="buttons-block">
    <div>
        <button class="figure-button" onclick="runWebGL()">Uruchom</button>
        <button class="figure-button" onclick="pauseAnimation()">Pauza</button>
        <button class="figure-button" onclick="startedPosition()">Stop</button>
    </div>
    <div class="pages-buttons">
        <button class="page-button"
            onclick="window.location.href='colored_cube.html'">Kolorowy sześcian</button>
        <button class="page-button"
            onclick="window.location.href='textured_cube.html'">Oteksturowany
            sześcian</button>
        <button class="page-button"
            onclick="window.location.href='colored_pyramid.html'">Kolorowy
            ostrosłup</button>
    </div>
</div>

<script src="../js/sierpinski_carpet.js"></script>
<script src="../js/matrix.js"></script>
</body>
</html>
```

Kod programu 18. Szablon dokumentu HTML na przykładzie widoku implementacji strony dla Dywanu Sierpińskiego.

2.6. Dyrektywy ustalające sposób wyświetlania komponentów dokumentu HTML

```css
.heading-title {
    font-family: 'Bungee';
    text-align: center;
    font-size: 16px;}

.configuration-block {
    height:auto;
    background-color: #111414;
    padding-top: 5px;
    padding-bottom: 5px;
    border-radius: 6px;
    box-shadow: 0px 4px 4px rgba(0, 0, 0, .7);}

.configuration-block .standard-label {
    font-family: 'Bungee';
    display: block;
    color: #f3fcff;
    text-align: center;
    margin-bottom: 4px;
    font-size: 12px;}

.configuration-block .rotate-buttons-form {
    width:30%;
    margin:0 auto;
    text-align:center;
    background-color: #ff920a;
    border-radius: 10px;}

.configuration-block .slider {
    display: inline;
    margin-top: -20px;
    position:relative;
    width: 400px;
    left: calc(50% - 200px);}

.configuration-block .combo-box {
    margin-left: 20px;}

canvas#glcanvas {
    border: 1px solid #66666D;
    background-color: #545469;
    width: 60%;
    margin-left: 20%;
    margin-top: 20px;
    box-shadow: 6px 6px 4px rgba(0, 0, 0, .5);}

.buttons-block {
    text-align:center;
    height:auto;
    background-color: #111414;
    margin-top: 20px;
    padding-top: 5px;
    padding-bottom: 5px;
    border-radius: 6px;
    box-shadow: 0px 4px 4px rgba(0, 0, 0, .7);}

.buttons-block .figure-button {
    font-family: 'Bungee';
    font-size: 10px;
    border:1px solid #000000;
    width:15%;
    background-color: #ff920a;
    border-radius: 10px;}

.buttons-block .pages-buttons {
    margin-top: 6px;}
```

Kod programu 19. Kod CSS dla komponentów HTML.

3. Wnioski

Projekt pozwolił na zapoznanie się z technologią renderowania interaktywnej grafiki 2D i 3D w przeglądarce internetowej, bazującej na elemencie HTML5 canvas. Po dokładnym zapoznaniu się ze sposobem renderowania obiektów za pomocą API JavaScript – WebGL kolejne etapy projektu sprawiały coraz mniejszą trudność.

4. Literatura

[1] MDN web docs, WebGL tutorial, https://developer.mozilla.org/pl/docs/Web/API/WebGL_API/Tutorial, 19.01.2018r.