WebGL, Projekcja stereoskopowa, Ładowanie modeli z plików OBJ

Część 1

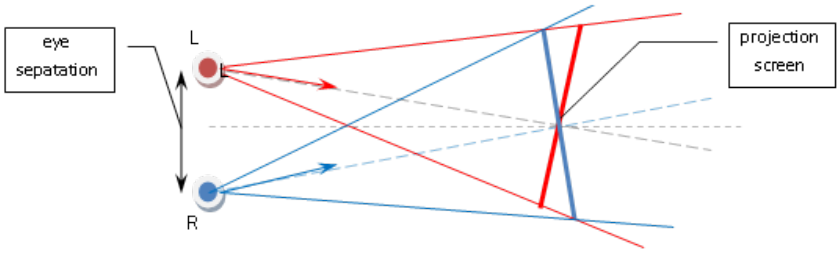
1. Cel ćwiczenia:

Zapoznanie z zagadnieniami projekcji stereoskopowej. Sposobem wyznaczania i stosowania zmodyfikowanej macierzy projekcji.

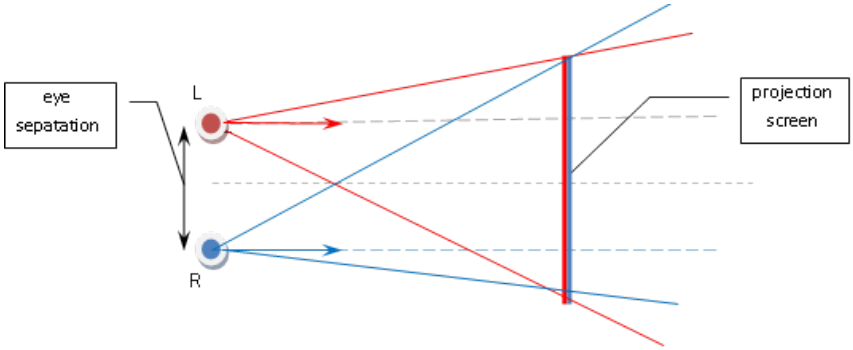
2. Wstęp:

Ze względu na szybki rozwój w dziedzinie wykorzystania nowoczesnych technik wizualnych oraz wzrostem dostępności tego typu rozwiązań na rynku można zaobserwować zainteresowanie wizualizacją stereoskopową. Tego typu rozwiązanie jest szeroko stosowane w różnych dziedzinach, począwszy od systemów obrazowania medycznego, procesów wizualizacji naukowej i rozrywki jak również w edukacji. Idea obrazowania stereoskopowego opiera się na założeniu tworzenia i prezentowania tej samej sceny z nieco innej perspektywy, jako oddzielnych obrazów dla każdego z oczu. Interpretacja obrazów przez ludzki mózg umożliwia uzyskanie efektu głębi. Stereoskopia nie jest nową koncepcją. W ostatnich latach można jednak zaobserwować znaczny wzrost zainteresowania tą metodą wizualizacji dzięki rozwojowi technik komputerowych i audio- wizualnych, a także wirtualnej rzeczywistości. Istnieje kilka metod generowania obrazów stereoskopowych oraz obliczania perspektywy.

Toe-in – to podejście stosuje się ze względu na prostotę oraz brak specjalnych wymagań co do zastosowanej kamery. Właściwe ustawienie kamer wymaga nie tylko przesunięcia względem siebie, ale również obrotu o pewien kąt. Niestety rozwiązanie to ma pewną wadę, ponieważ wprowadza pionową paralaksę, zwiększającą się wraz z odległością od centrum obrazu. Wprowadza to pewien dyskomfort obserwatora po dłuższym czasie. Pomimo przedstawionej wady, metoda ta jest nadal stosowana ze względu na łatwość korzystania z równoległych kamer.



Off-axis – jest prawidłowym sposobem tworzenia obrazów stereoskopowych, który nie powoduje pionowej paralaksy. W tym przypadku kamery są przesunięte równolegle względem siebie, jednocześnie modyfikowana jest macierz projekcji w celu uzyskania tzw. projekcji asymetrycznej.



3. Funkcja umożliwiająca przełączanie widoków dla trybu Off-axis:

```
function StereoProjection(_left, _right, _bottom, _top, _near, _far, _zero_plane, _dist, _eye)
{
        //    Perform the perspective projection for one eye's subfield.
        //    The projection is in the direction of the negative z-axis.
              //            _left=-6.0;
              //            _right=6.0;
              //            _bottom=-4.8;
              //             _top=4.8;
        //    [default: -6.0, 6.0, -4.8, 4.8]
        //    left, right, bottom, top = the coordinate range, in the plane of zero parallax setting,
        //         which will be displayed on the screen.
        //         The ratio between (right-left) and (top-bottom) should equal the aspect
        //    ratio of the display.
                   //                  _near=6.0;
                   //                  _far=-20.0;
        //    [default: 6.0, -6.0]
        //    near, far = the z-coordinate values of the clipping planes.
                   //                  _zero_plane=0.0;
        //    [default: 0.0]
        //    zero_plane = the z-coordinate of the plane of zero parallax setting.

        //    [default: 14.5]
                  //                    _dist=10.5;
        //   dist = the distance from the center of projection to the plane of zero parallax.

        //    [default: -0.3]
                  //                _eye=-0.3;
        //    eye = half the eye separation; positive for the right eye subfield,
        //    negative for the left eye subfield.

        let   _dx = _right - _left;
        let   _dy = _top - _bottom;

        let   _xmid = (_right + _left) / 2.0;
        let   _ymid = (_top + _bottom) / 2.0;

        let   _clip_near = _dist + _zero_plane - _near;
        let   _clip_far = _dist + _zero_plane - _far;

        let  _n_over_d = (_clip_near / _dist);

        let   _topw = _n_over_d * _dy / 2.0;
        let   _bottomw = -_topw;
        let   _rightw = _n_over_d * (_dx / 2.0 - _eye);
        let   _leftw = _n_over_d * (-_dx / 2.0 - _eye);/

        const proj = mat4.create();
        mat4.frustum(proj, _leftw, _rightw, _bottomw, _topw, _clip_near, _clip_far)
        mat4.translate(proj, proj, [-_xmid - _eye, -_ymid, 0]);
        let uniProj = gl.getUniformLocation(program, 'proj');
        gl.uniformMatrix4fv( uniProj, false, proj);
}
```

Kolejność wykonywanych poleceń w celu uzyskania obrazu stereoskopowego:

1. Ustalenie obszaru renderingu:

```
gl.viewport(0, 0, canvas.width, canvas.height);
```

2. Określenie widoku dla lewego oka:

```
StereoProjection(-6, 6, -4.8, 4.8, 12.99, -100, 0, 13, -0.1);
gl.colorMask(true,false,false,false);
.
.       Rysowanie sceny
.
```

Dla prawego oka:

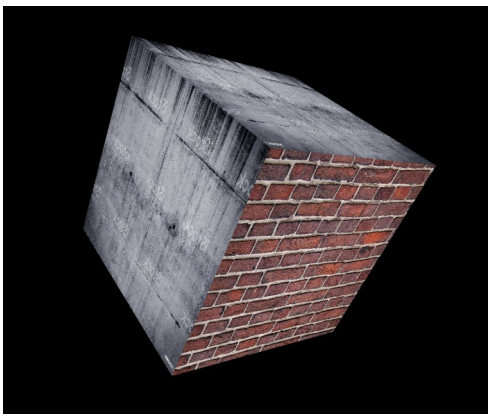
```
gl.clear(gl.DEPTH_BUFFER_BIT);
StereoProjection(-6, 6, -4.8, 4.8, 12.99, -100, 0, 13, 0.1);
gl.colorMask(false,false,true,false);
.
.       Rysowanie sceny
.
gl.colorMask(true,true,true,true);
```

5. Ćwiczenie:

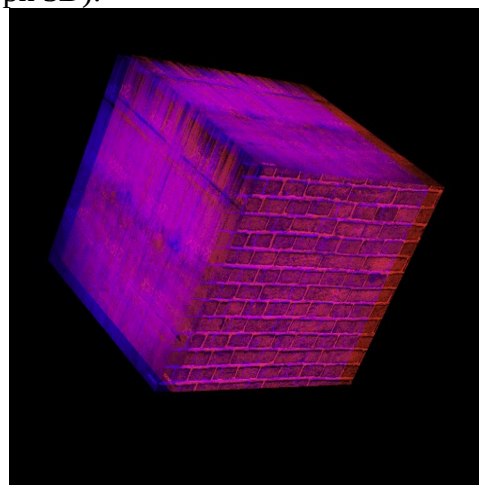
Należy przygotować projekt umożliwiający prezentowanie obrazu wykorzystując projekcję stereoskopową w trybie mono, stereo(anaglyph 3D) oraz side-by-side.

Program ma mieć możliwość przełączania widoku pomiędzy standardowym i stereoskopowym oraz możliwość regulacji parametrów projekcji (np. separacji) za pomocą klawiatury.

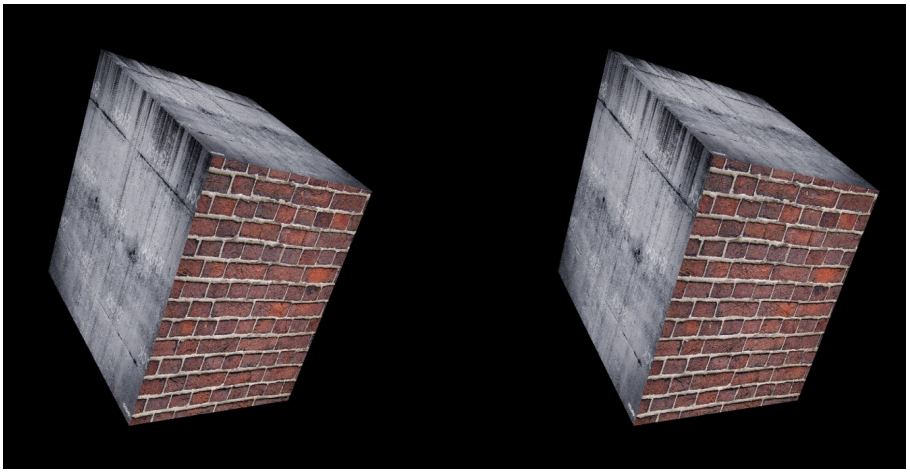
1. Widok mono



2. Widok stereo (anaglyph 3D):



3. Widok stereo (side-by-side):



Część 2
Ładowanie modeli z plików OBJ

1. Cel ćwiczenia:

Zapoznanie z zagadnieniami parsowania plików OBJ.

2. Wstęp:

W przypadku konieczności użycia w programie bardziej złożonych modeli jak np. pojazdy, budynki konieczne jest użycie modeli zbudowanych przy użyciu dedykowanych programów graficznych jak Blender 3D, 3DS czy Softimage.

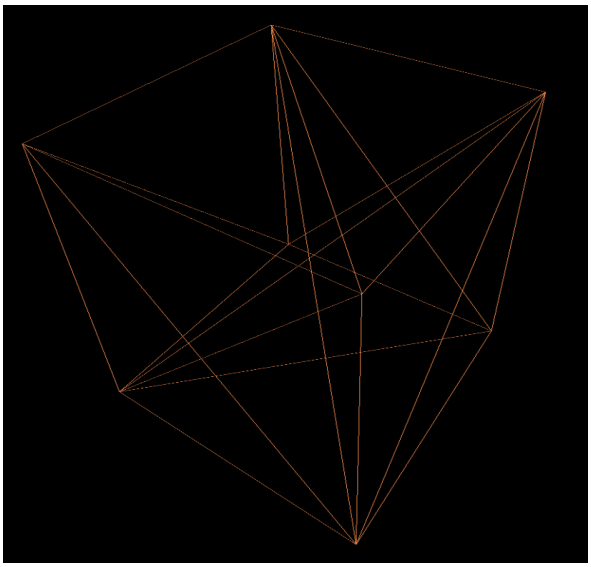
Modele składają się z dużej liczby wierzchołków na podstawie których budowane są siatki. Jednym z programów (darmowym) do budowy modeli jest Bledner 3D pozwalający na eksport obiektów do wielu formatów plików takich jak 3dS, fbx, obj…

Często stosowanym formatem zapisu modeli 3D ze względu na prostotę (plik tekstowy) jest OBJ. Zwiera współrzędne wierzchołków, wektory normalne, koordynaty tekstury, informacje o materiale i inne.

W najprostszym przypadku plik OBJ może zawierać tylko wierzchołki i powierzchnie. Dodatkowy plik MTL zawiera informacje o materiale.

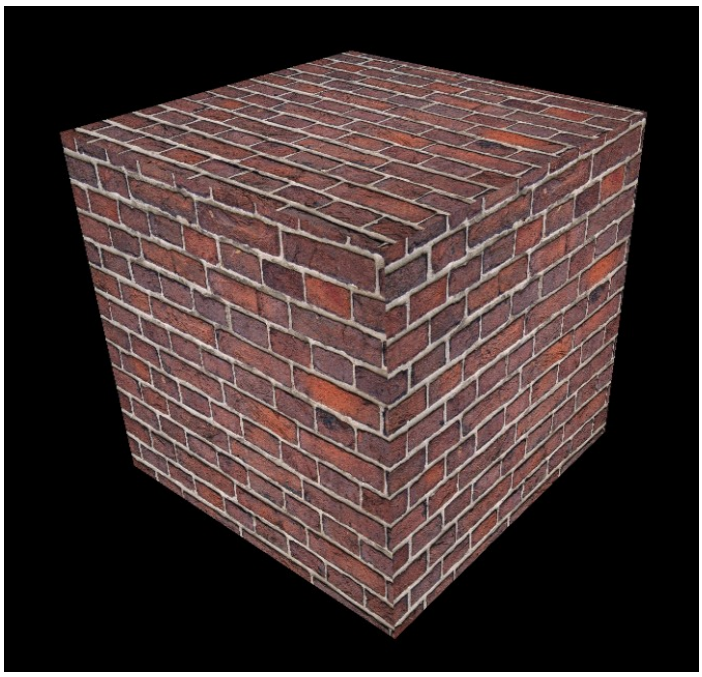
Przykładowy plik OBJ dla kostki:(triangulate faces)

```
# Blender v2.93.5 OBJ File: ''
# www.blender.org
o Cube
v 1.000000 1.000000 -1.000000
v 1.000000 -1.000000 -1.000000
v 1.000000 1.000000 1.000000
v 1.000000 -1.000000 1.000000
v -1.000000 1.000000 -1.000000
v -1.000000 -1.000000 -1.000000
v -1.000000 1.000000 1.000000
v -1.000000 -1.000000 1.000000
s off
f 5 3 1
f 3 8 4
f 7 6 8
f 2 8 6
f 1 4 2
f 5 2 6
f 5 7 3
f 3 7 8
f 7 5 6
f 2 4 8
f 1 3 4
f 5 1 2
```



Przykładowy plik OBJ dla kostki:(triangulate faces, include UVs, write normals)

```
# Blender v2.93.5 OBJ File: ''
# www.blender.org
o Cube
v 1.000000 1.000000 -1.000000
v 1.000000 -1.000000 -1.000000
v 1.000000 1.000000 1.000000
v 1.000000 -1.000000 1.000000
v -1.000000 1.000000 -1.000000
v -1.000000 -1.000000 -1.000000
v -1.000000 1.000000 1.000000
v -1.000000 -1.000000 1.000000
vt 0.000000 1.000000
vt 1.000000 0.000000
vt 1.000000 1.000000
vt 1.000000 1.000000
vt 0.000000 0.000000
vt 1.000000 0.000000
vt 0.000000 1.000000
vt 1.000000 0.000000
vt 1.000000 1.000000
vt 0.000000 1.000000
vt 0.000000 0.000000
vt 1.000000 0.000000
vt 0.000000 0.000000
vt 0.000000 0.000000
vt 1.000000 1.000000
vt 0.000000 1.000000
vn 0.0000 1.0000 0.0000
vn 0.0000 0.0000 1.0000
vn -1.0000 0.0000 0.0000
vn 0.0000 -1.0000 0.0000
vn 1.0000 0.0000 0.0000
vn 0.0000 0.0000 -1.0000
s off
f 5/1/1 3/2/1 1/3/1
f 3/4/2 8/5/2 4/6/2
f 7/7/3 6/8/3 8/5/3
f 2/9/4 8/5/4 6/10/4
f 1/3/5 4/11/5 2/12/5
f 5/1/6 2/12/6 6/13/6
f 5/1/1 7/14/1 3/2/1
f 3/4/2 7/7/2 8/5/2
f 7/7/3 5/15/3 6/8/3
f 2/9/4 4/6/4 8/5/4
f 1/3/5 3/16/5 4/11/5
f 5/1/6 1/3/6 2/12/6
```



Linie rozpoczynające się od znaków:

- # – to komentarze,
- o – obiekt,
- v – wierzchołki,
- f – powierzchnie zbudowane na wierzchołkach (cyfry określają numery wierzchołków tworzących daną powierzchnię/numer koordynaty tekstury/numer wektora normalnego),
- vt – koordynaty tekstury,
- vn – wektory normalne

3. Funkcja umożliwiająca import modeli OBJ :

```
async function loadFile(file) {
    text = await file.text();
    text=text.replaceAll('/', ' ');
    text=text.replaceAll('\n', ' ');
    let arrayCopy = text.split(' ');

const vertices = [[]];  let licz_vertices = 0;
const normals = [[]]; let licz_normals = 0;
const coords = [[]];  let licz_coords = 0;
const triangles = []; let licz_triangles = 0;
```

```
for (let i=0;i<arrayCopy.length-1;i++)
{
        if (arrayCopy[i]=='v') {
        vertices.push([]);
        vertices[licz_vertices].push(parseFloat(arrayCopy[i+1]));
        vertices[licz_vertices].push(parseFloat(arrayCopy[i+2]));
        vertices[licz_vertices].push(parseFloat(arrayCopy[i+3]));
        i+=3;
        licz_vertices++;
        }

        if (arrayCopy[i]=='vn') {
        normals.push([]);
        normals[licz_normals].push(parseFloat(arrayCopy[i+1]));
        normals[licz_normals].push(parseFloat(arrayCopy[i+2]));
        normals[licz_normals].push(parseFloat(arrayCopy[i+3]));
        i+=3;
        licz_normals++;
        }

        if (arrayCopy[i]=='vt') {
        coords.push([]);
        coords[licz_coords].push(parseFloat(arrayCopy[i+1]));
        coords[licz_coords].push(parseFloat(arrayCopy[i+2]));
        i+=2;
        licz_coords++;
        }

        if (arrayCopy[i]=='f') {
                triangles.push([]);
                for (let j=1;j<=9;j++)
                        triangles[licz_triangles].push(parseFloat(arrayCopy[i+j]));
        i+=9;
        licz_triangles++;
        }
}

let vert_array=[];

        for (let i = 0; i < triangles.length; i++)
        {
                vert_array.push(vertices[triangles[i][0] - 1][0]);
                vert_array.push(vertices[triangles[i][0] - 1][1]);
                vert_array.push(vertices[triangles[i][0] - 1][2]);
                vert_array.push(normals[triangles[i][2] - 1][0]);
                vert_array.push(normals[triangles[i][2] - 1][1]);
                vert_array.push(normals[triangles[i][2] - 1][2]);
                vert_array.push(coords[triangles[i][1] - 1][0]);
                vert_array.push(coords[triangles[i][1] - 1][1]);

                vert_array.push(vertices[triangles[i][3] - 1][0]);
                vert_array.push(vertices[triangles[i][3] - 1][1]);
                vert_array.push(vertices[triangles[i][3] - 1][2]);
                vert_array.push(normals[triangles[i][5] - 1][0]);
                vert_array.push(normals[triangles[i][5] - 1][1]);
                vert_array.push(normals[triangles[i][5] - 1][2]);
                vert_array.push(coords[triangles[i][4] - 1][0]);
                vert_array.push(coords[triangles[i][4] - 1][1]);

                vert_array.push(vertices[triangles[i][6] - 1][0]);
                vert_array.push(vertices[triangles[i][6] - 1][1]);
                vert_array.push(vertices[triangles[i][6] - 1][2]);
                vert_array.push(normals[triangles[i][8] - 1][0]);
                vert_array.push(normals[triangles[i][8] - 1][1]);
                vert_array.push(normals[triangles[i][8] - 1][2]);
                vert_array.push(coords[triangles[i][7] - 1][0]);
                vert_array.push(coords[triangles[i][7] - 1][1]);
        }

points=triangles.length*3;
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vert_array), gl.STATIC_DRAW);
}

W sekcji body
<input type="file" onchange="loadFile(this.files[0])">
```

5. Ćwiczenie:

Należy przygotować projekt umożliwiający wczytywanie modeli z pliku OBJ. Następnie przygotować w programie Blender model stołu i 4 krzeseł oraz modele Blendera typu mesh.