SPRAWOZDANIE

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

Laboratorium II
Data
Temat: Zadanie_teksturyGL

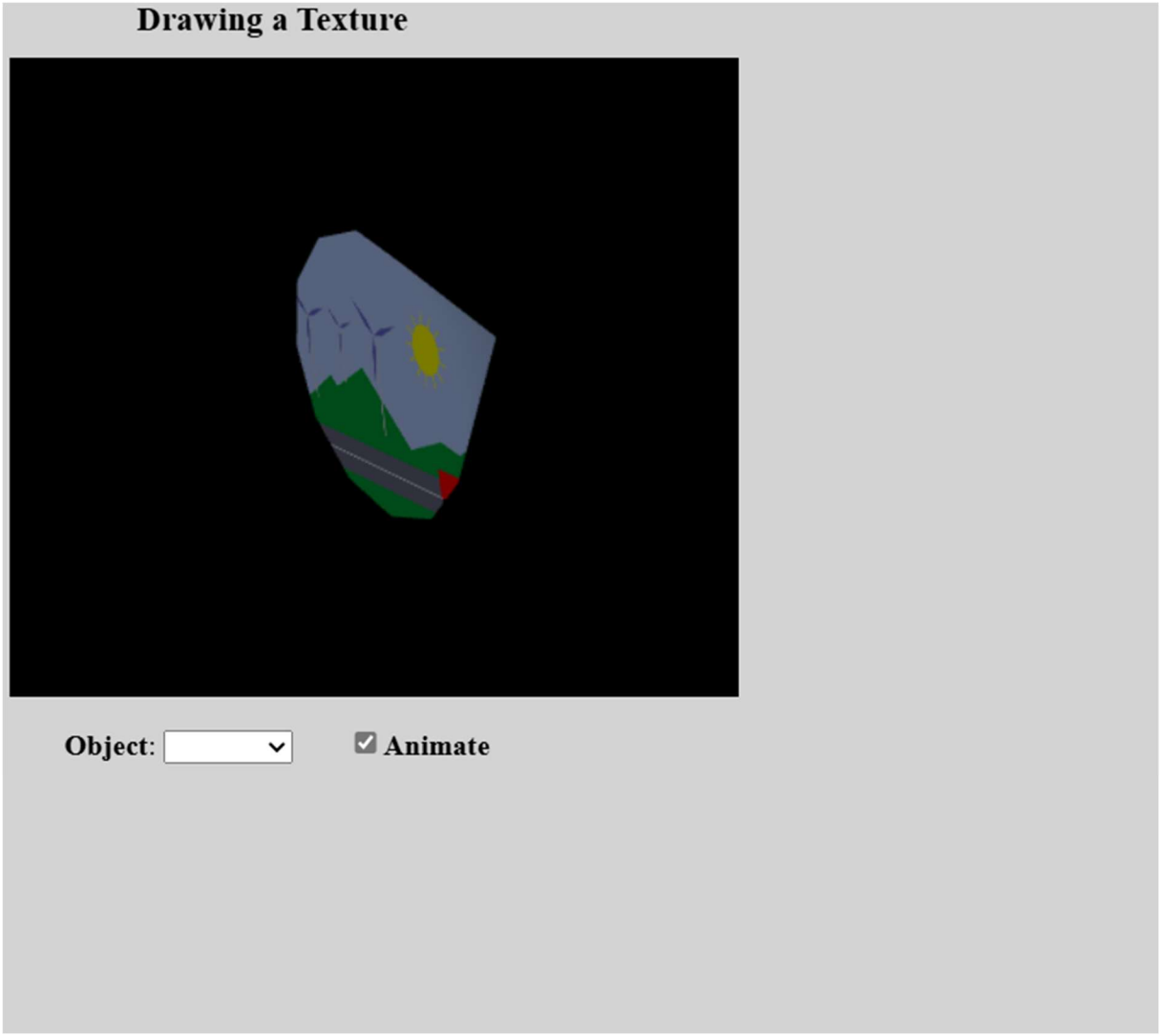
Wariant 8 + 4

Jakub Bąk
Informatyka I stopień,
stacjonarne,
4 semestr,
Gr.3b

Polecenie

Celem jest teksturowanie piramidy z użyciem dwóch sposobów ładowania tekstur: użycie tekstury z buforu kolorów (rysowanie w Panel); ładowanie tekstury z pliku (trzy pliki przykładowe do pobrania)

1. Wyniki zadania:



2. Wykorzystane komendy:

Link do github: https://github.com/Szeladin/grafika.git

Kod Programu:

```
1. <!DOCTYPE html>
2. <html>
```

```
<head>
<meta charset="UTF-8">
<title>Texture From Color Buffer</title>
<link rel="stylesheet" href="../demo.css">
<script src="../script/demo-core.js"></script>
<script src="../script/glsim.js"></script>
<script src="../script/pyramid-model-IFS.js"></script>
<script src="../script/basic-object-models-IFS.js"></script>
<script>
var camera;
var canvas;
var frameNumber = 0;
var pyramid;

function draw() {

    var objectNumber = Number(document.getElementById("object").value);
    glDisable(GL_LIGHTING);
    glDisable(GL_DEPTH_TEST);
    glDisable(GL_TEXTURE_2D);
    glViewport(0,0,256,256);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho( 0,7, -1,5, -1,1 );
    glMatrixMode(GL_MODELVIEW);
    draw2DScene();
    if (objectNumber == 6) {
        return;
    }
    glCopyTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, 0, 0, 256, 256, 0);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glEnable(GL_LIGHTING);
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_TEXTURE_2D);
    glViewport(0,0,canvas.width,canvas.height);
    camera.apply();
    glClearColor( 0, 0, 0, 1 );
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    switch(objectNumber) {
    case 0:
     glScalef(0.06, 0.06, 0.06);
     drawModel(pyramid);
     break;
    }
}
function drawModel(model) {
    glEnableClientState(GL_VERTEX_ARRAY);
    glVertexPointer(3,GL_FLOAT,0,model.vertexPositions);
    glEnableClientState(GL_NORMAL_ARRAY);
    glNormalPointer(GL_FLOAT, 0, model.vertexNormals);
    glEnableClientState(GL_TEXTURE_COORD_ARRAY);
    glTexCoordPointer(2,GL_FLOAT,0,model.vertexTextureCoords);
    glDrawElements(GL_TRIANGLES, model.indices.length, GL_UNSIGNED_BYTE, model.indices);
    glDisableClientState(GL_VERTEX_ARRAY);
    glDisableClientState(GL_NORMAL_ARRAY);
    glDisableClientState(GL_TEXTURE_COORD_ARRAY);
}
function initGL() {
    glEnable(GL_LIGHT0);
    glEnable(GL_NORMALIZE);
    glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE, [ 1, 1, 1,1 ]);// white material
for texturing.

}
function drawDisk(radius) {
    var d;
    glBegin(GL_POLYGON);
    for (d = 0; d < 32; d++) {
        var angle = 2*Math.PI/32 * d;
        glVertex2d( radius*Math.cos(angle), radius*Math.sin(angle));
```

```
    }
    glEnd();
}
function drawWheel() {
    var i;
    glColor3f(0,0,0);
    drawDisk(1);
    glColor3f(0.75, 0.75, 0.75);
    drawDisk(0.8);
    glColor3f(0,0,0);
    drawDisk(0.2);
    glRotatef(frameNumber*20,0,0,1);
    glBegin(GL_LINES);
    for (i = 0; i < 15; i++) {
        glVertex2f(0,0);
        glVertex2d(Math.cos(i*2*Math.PI/15), Math.sin(i*2*Math.PI/15));
    }
    glEnd();
}
function drawCart() {
    glPushMatrix();
    glTranslatef(-1.5, -0.1, 0);
    glScalef(0.8,0.8,1);
    drawWheel();
    glPopMatrix();
    glPushMatrix();
    glTranslatef(1.5, -0.1, 0);
    glScalef(0.8,0.8,1);
    drawWheel();
    glPopMatrix();
    glColor3f(1,0,0);
    glBegin(GL_POLYGON);
    glVertex2f(-2.5,0);
    glVertex2f(2.5,0);
    glVertex2f(2.5,2);
    glVertex2f(-2.5,2);
    glEnd();
}
function drawSun() {
    var i;
    glColor3f(1,1,0);
    for (i = 0; i < 13; i++) { // Draw 13 rays, with different rotations.
        glRotatef( 360 / 13, 0, 0, 1 ); // Note that the rotations accumulate!
        glBegin(GL_LINES);
        glVertex2f(0, 0);
        glVertex2f(0.75, 0);
        glEnd();
    }
    drawDisk(0.5);
    glColor3f(0,0,0);
}
function drawWindmill() {
    var i;
    glColor3f(0.8, 0.8, 0.9);
    glBegin(GL_POLYGON);
    glVertex2f(-0.05, 0);
    glVertex2f(0.05, 0);
    glVertex2f(0.05, 3);
    glVertex2f(-0.05, 3);
    glEnd();
    glTranslatef(0, 3, 0);
    glRotated(frameNumber * (180.0/46), 0, 0, 1);
    glColor3f(0.4, 0.4, 0.8);
    for (i = 0; i < 3; i++) {
        glRotated(120, 0, 0, 1);  // Note: These rotations accumulate.
        glBegin(GL_POLYGON);
        glVertex2f(0,0);
        glVertex2f(0.5, 0.1);
        glVertex2f(1.5,0);
        glVertex2f(0.5, -0.1);
```

```
142.         glEnd();
143.     }
144. }
145. function draw2DScene() {
146.     glClearColor( 0.7, 0.8, 1.0, 1.0 );
147.     glClear(GL_COLOR_BUFFER_BIT);
148.     glLoadIdentity();
149.     glColor3f(0, 0.6, 0.2);
150.     glBegin(GL_POLYGON);
151.     glVertex2f(-3,-1);
152.     glVertex2f(1.5,1.65);
153.     glVertex2f(5,-1);
154.     glEnd();
155.     glBegin(GL_POLYGON);
156.     glVertex2f(-3,-1);
157.     glVertex2f(3,2.1);
158.     glVertex2f(7,-1);
159.     glEnd();
160.     glBegin(GL_POLYGON);
161.     glVertex2f(0,-1);
162.     glVertex2f(6,1.2);
163.     glVertex2f(20,-1);
164.     glEnd();
165.     glColor3f(0.4, 0.4, 0.5);
166.     glBegin(GL_POLYGON);
167.     glVertex2f(0,-0.4);
168.     glVertex2f(7,-0.4);
169.     glVertex2f(7,0.4);
170.     glVertex2f(0,0.4);
171.     glEnd();
172.     glLineWidth(4);
173.     glColor3f(1,1,1);
174.     glBegin(GL_LINES);
175.     glVertex2f(0,0);
176.     glVertex2f(7,0);
177.     glEnd();
178.     glLineWidth(1);
179.     glPushMatrix();
180.     glTranslated(5.8,3,0);
181.     glRotated(-frameNumber*0.7,0,0,1);
182.     drawSun();
183.     glPopMatrix();
184.     glPushMatrix();
185.     glTranslated(0.75,1,0);
186.     glScaled(0.6,0.6,1);
187.     drawWindmill();
188.     glPopMatrix();
189.     glPushMatrix();
190.     glTranslated(2.2,1.6,0);
191.     glScaled(0.4,0.4,1);
192.     drawWindmill();
193.     glPopMatrix();
194.     glPushMatrix();
195.     glTranslated(3.7,0.8,0);
196.     glScaled(0.7,0.7,1);
197.     drawWindmill();
198.     glPopMatrix();
199.     glPushMatrix();
200.     glTranslated(-3 + 13*(frameNumber % 300) / 300.0, 0, 0);
201.     glScaled(0.3,0.3,1);
202.     drawCart();
203.     glPopMatrix();
204. }
205. var animating = false;
206. function frame() {
207.     if (animating) {
208.      frameNumber++;
209.      draw();
210.      setTimeout(frame,30);
211.     }
```

```
212. }
213. function doAnimate() {
214.     animating = document.getElementById("animate").checked;
215.     if (animating) {
216.      frame();
217.     }
218. }
219. function init() {
220.     try {
221.        canvas = document.getElementById("maincanvas");
222.            glsimUse(canvas,null);
223.     }
224.     catch (e) {
225.         document.getElementById("canvas-holder").innerHTML="<p><b>Sorry, an error
occurred:<br>" +
226.                 e + "</b></p>";
227.         return;
228.     }
229.     initGL();
230.     document.getElementById("object").value = "1";
231.     document.getElementById("object").onchange = draw;
232.     document.getElementById("animate").checked = false;
233.     document.getElementById("animate").onchange = doAnimate;
234.     camera = new Camera();
235.     camera.setScale(1);
236.      camera.lookAt(2,2,5, 0,0,0, 0,1,0);
237.     camera.installTrackball(draw);
238.     sphere = uvSphere();
239.     cubeModel = cube();
240.     cylinder = uvCylinder();
241.     cone = uvCone();
242.     torus = uvTorus();
243.     pyramid = pyramidModel;
244.     draw();
245. }
246. </script>
247. </head>
248. <body onload="init()">
249. <div id="content">
250. <h3 id="headline">Drawing a Texture</h3>
251. <div id="canvas-holder">
252. <canvas id="maincanvas" width="400" height="350"></canvas>
253. </div>
254. <br clear=all>
255. <p style="text-indent:30px"><b>Object</b>:
256. <select id="object">
257.    <option value="0">Pyramid</option>
258. </select>
259. <label><input type="checkbox" id="animate" style="margin-
left:30px"><b>Animate</b></label></p>
260. </div>
261. </body>
262. </html>
263.
```

Program to aplikacja internetowa wykorzystująca WebGL do renderowania sceny 3D i 2D na płótnie HTML `<canvas>`. Umożliwia użytkownikowi wybór obiektu (np. piramidy) do wyświetlenia oraz opcjonalne animowanie sceny. Scena 2D zawiera elementy takie jak słońce, wiatraki, wózek i krajobraz, natomiast scena 3D renderuje wybrany obiekt z teksturą. Program obsługuje kamerę z możliwością manipulacji (np. skalowanie, obrót) oraz animację w pętli.

3. Wnioski:

Kod demonstruje wykorzystanie tekstur w WebGL poprzez kopiowanie zawartości bufora kolorów (`glCopyTexImage2D`) i używanie jej jako tekstury na obiektach 3D. Tekstury są generowane dynamicznie na podstawie sceny 2D, co pozwala na tworzenie złożonych efektów wizualnych. Ustawienia takie jak `GL_LINEAR` dla filtrowania tekstur zapewniają płynne przejścia między pikselami.