SPRAWOZDANIE

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

Laboratorium 7
12.05.2024
Temat: teksturyGL

Wariant 9

Gabriel Mrzygłód
Informatyka I stopień,
niestacjonarne,
4 semestr,

Gr.2a

1. Polecenie:

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

2. Wprowadzane dane:

Trzynastokąt

3. Wykorzystane komendy:

a) kod źródłowy

```
<!DOCTYPE html>
<html>
<!--
   Draws a 2D scene using OpenGL, then copies that scene to a
   texture image so that it can be applied as a texture to a
   3D object.  The scene can be animated.
-->
<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/teapot-model-IFS.js"></script>
<script src="../script/basic-object-models-IFS.js"></script>
<script>

var camera;

var canvas;  // the canvas on which we will draw.

var frameNumber = 0;  // frame number for the animation

var sphere, cubeModel, cylinder, cone, torus, teapot, pyramid13;  // model data for seven objects.

function createThirteenSidedPyramid() {
   var numSides = 13;
   var vertices = [];
   var indices = [];
   var normals = [];
   var texCoords = [];
   var angleStep = 2 * Math.PI / numSides;
   var radius = 0.5;

   // Base vertices and normals
   for (var i = 0; i < numSides; i++) {
      var angle = i * angleStep;
```

```
        vertices.push(radius * Math.cos(angle), radius * Math.sin(angle), 0);
        normals.push(0, 0, -1); // Normal for base is downwards
        texCoords.push((1 + Math.cos(angle)) / 2, (1 + Math.sin(angle)) / 2); // Texture coordinates for base
    }

    // Apex vertex and normal
    vertices.push(0, 0, 1);
    normals.push(0, 0, 1); // Normal for apex is upwards
    texCoords.push(0.5, 0.5); // Texture coordinate for apex

    // Indices for base
    for (var i = 1; i <= numSides; i++) {
        indices.push(0, i % numSides, (i + 1) % numSides);
    }

    // Indices for sides
    var apexIndex = numSides;
    for (var i = 0; i < numSides; i++) {
        indices.push(apexIndex, i, (i + 1) % numSides);
        // Calculate normals for sides
        var normal = calculateNormal(
            [vertices[3 * apexIndex], vertices[3 * apexIndex + 1], vertices[3 * apexIndex + 2]],
            [vertices[3 * i], vertices[3 * i + 1], vertices[3 * i + 2]],
            [vertices[3 * ((i + 1) % numSides)], vertices[3 * ((i + 1) % numSides) + 1], vertices[3 * ((i + 1) %
numSides) + 2]]
        );
        normals.push(...normal);
        normals.push(...normal);
        normals.push(...normal);
        // Add corresponding texture coordinates for sides
        texCoords.push(0.5, 1, (1 + Math.cos(i * angleStep)) / 2, (1 + Math.sin(i * angleStep)) / 2, (1 + Math.cos((i
+ 1) * angleStep)) / 2, (1 + Math.sin((i + 1) * angleStep)) / 2);
    }

    return {
        vertexPositions: new Float32Array(vertices),
        vertexNormals: new Float32Array(normals),
        vertexTextureCoords: new Float32Array(texCoords),
        indices: new Uint8Array(indices)
    };
}

function calculateNormal(v1, v2, v3) {
    var U = [v2[0] - v1[0], v2[1] - v1[1], v2[2] - v1[2]];
    var V = [v3[0] - v1[0], v3[1] - v1[1], v3[2] - v1[2]];
    var normal = [
        U[1] * V[2] - U[2] * V[1],
        U[2] * V[0] - U[0] * V[2],
        U[0] * V[1] - U[1] * V[0]
    ];
    var length = Math.sqrt(normal[0] * normal[0] + normal[1] * normal[1] + normal[2] * normal[2]);
    return [normal[0] / length, normal[1] / length, normal[2] / length]; // Normalize the normal
}

/*  The display function, which draws the content of the canvas.
 */
```

```
function draw() {

    var objectNumber = Number(document.getElementById("object").value);  // which object to draw.

    /* First, draw the 2D scene, using a 256-by256 viewport to get a power-of-two texture. */

    glDisable(GL_LIGHTING);
    glDisable(GL_DEPTH_TEST);
    glDisable(GL_TEXTURE_2D);
    glViewport(0,0,256,256);  // Note that canvas must be at least 256-by-256.
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho( 0,7, -1,5, -1,1 );  // Limits required by 2D scene
    glMatrixMode(GL_MODELVIEW);

    draw2DScene();

    if (objectNumber == 7) {
        return;   // Just show the 2D scene as the image in the canvas.
    }

    /* Copy the image into the texture. */

    glCopyTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, 0, 0, 256, 256, 0);

    /* Since we do not have mipmaps for the texture, we MUST set the texture
       minimization filter to GL_NEAREST or GL_LINEAR, since the default
       filter requires mipmaps. */

    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);

    /* Now, draw the shape, with the texture */

    glEnable(GL_LIGHTING);
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_TEXTURE_2D);
    glViewport(0,0,canvas.width,canvas.height);  // restore full viewport!

    camera.apply(); // (Sets up projection and viewing transforms.)

    glClearColor( 0, 0, 0, 1 );
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    switch(objectNumber) {
    case 0:
        glRotatef(-90,1,0,0);
        glScalef(1.5,1.5,1.5);
        drawModel(sphere);
        break;
    case 1:
        glScalef(1.2,1.2,1.2);
        drawModel(cubeModel);
        break;
    case 2:
        glRotatef(-90,1,0,0);
        glScalef(1.3,1.3,1.3);
        drawModel(cylinder);
        break;
    case 3:
        glRotatef(-90,1,0,0);
        glScalef(1.3,1.3,1.3);
        drawModel(cone);
        break;
    case 4:
        glScalef(1.6,1.6,1.6);
        drawModel(torus);
        break;
    case 5:
        glScalef(0.06, 0.06, 0.06);
        drawModel(teapot);
        break;
    case 6:
        glScalef(1.5, 1.5, 1.5);
        drawModel(pyramid13);
        break;
    }

}

/**
 *  Draws a model using glDrawElements.  The model data must be in the format produced by
 *  the functions in basic-object-models-IFS.js.
 */
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); // Ensure normals are normalized after transformations.

    // Adjust the light position and intensity.
    glLightfv(GL_LIGHT0, GL_POSITION, [1, 1, 1, 0]); // Change light position to shine from above
    glLightfv(GL_LIGHT0, GL_DIFFUSE, [1.5, 1.5, 1.5, 1]); // Increase light intensity

    // Add an ambient light for softer shadows.
    glLightModelfv(GL_LIGHT_MODEL_AMBIENT, [0.5, 0.5, 0.5, 1]); // Soft white ambient light

    glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE, [1, 1, 1, 1]); // Material 
reflecting the light
}

//----------------- Drawing the 2D scene ---------------------------------------
```

```
/*
 * Draw a 32-sided regular polygon as an approximation for a circular disk.
 * (This is necessary since OpenGL has no commands for drawing ovals, circles,
 * or curves.)  The disk is centered at (0,0) with a radius given by the
 * parameter.
 */
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();
}

/*
 * Draw a wheel, centered at (0,0) and with radius 1. The wheel has 15 spokes
 * that rotate in a clockwise direction as the animation proceeds.
 */
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();
}

/*
 * Draw a cart consisting of a rectangular body and two wheels.  The wheels
 * are drawn by the drawWheel() method; a different translation is applied to each
 * wheel to move them into position under the body.  The body of the cart
 * is a red rectangle with corner at (0,-2.5), width 5, and height 2.  The
 * center of the bottom of the rectangle is at (0,0).
 */
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();
}

/*
 * Draw a sun with radius 0.5 centered at (0,0).  There are also 13 rays which
 * extend outside from the sun for another 0.25 units.
 */
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);
}

/*
 * Draw a windmill, consisting of a pole and three vanes.  The pole extends from the
 * point (0,0) to (0,3).  The vanes radiate out from (0,3).  A rotation that depends
 * on the frame number is applied to the whole set of vanes, which causes the windmill
 * to rotate as the animation proceeds.  Note that this method changes the current
 * transform in the GL context gl!  The caller of this subroutine should take care
 * to save and restore the original transform, if necessary.
 */
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);
        glEnd();
    }
```

```
}

/*  Draws the entire 2D scene.
 */
function draw2DScene() {

    glClearColor( 0.7, 0.8, 1.0, 1.0 );
    glClear(GL_COLOR_BUFFER_BIT); // Fills the scene with blue.
    glLoadIdentity();

    /* Draw three green triangles to form a ridge of hills in the background */

    glColor3f(0, 0.6, 0.2);
    glBegin(GL_POLYGON);
    glVertex2f(-3,-1);
    glVertex2f(1.5,1.65);
    glVertex2f(5,-1);
    glEnd();
    glBegin(GL_POLYGON);
    glVertex2f(-3,-1);
    glVertex2f(3,2.1);
    glVertex2f(7,-1);
    glEnd();
    glBegin(GL_POLYGON);
    glVertex2f(0,-1);
    glVertex2f(6,1.2);
    glVertex2f(20,-1);
    glEnd();

    /* Draw a bluish-gray rectangle to represent the road. */

    glColor3f(0.4, 0.4, 0.5);
    glBegin(GL_POLYGON);
    glVertex2f(0,-0.4);
    glVertex2f(7,-0.4);
    glVertex2f(7,0.4);
    glVertex2f(0,0.4);
    glEnd();

    /* Draw a white line to represent the stripe down the middle
     * of the road. */

    glLineWidth(4);  // Set the line width to be 6 pixels.
    glColor3f(1,1,1);
    glBegin(GL_LINES);
    glVertex2f(0,0);
    glVertex2f(7,0);
    glEnd();
    glLineWidth(1);  // Reset the line width to be 1 pixel.

    /* Draw the sun.  The drawSun method draws the sun centered at (0,0).  A 2D translation
     * is applied to move the center of the sun to (5,3.3).   A rotation makes it rotate*/

    glPushMatrix();
    glTranslated(5.8,3,0);
    glRotated(-frameNumber*0.7,0,0,1);
    drawSun();
    glPopMatrix();

    /* Draw three windmills.  The drawWindmill method draws the windmill with its base
     * at (0,0), and the top of the pole at (0,3).  Each windmill is first scaled to change
     * its size and then translated to move its base to a different paint.  In the animation,
     * the vanes of the windmill rotate.  That rotation is done with a transform inside the
     * drawWindmill method. */

    glPushMatrix();
    glTranslated(0.75,1,0);
    glScaled(0.6,0.6,1);
    drawWindmill();
    glPopMatrix();

    glPushMatrix();
    glTranslated(2.2,1.6,0);
    glScaled(0.4,0.4,1);
    drawWindmill();
    glPopMatrix();

    glPushMatrix();
    glTranslated(3.7,0.8,0);
    glScaled(0.7,0.7,1);
    drawWindmill();
    glPopMatrix();

    /* Draw the cart.  The drawCart method draws the cart with the center of its base at
     * (0,0).  The body of the cart is 5 units long and 2 units high.  A scale is first
     * applied to the cart to make its size more reasonable for the picture.  Then a
     * translation is applied to move the cart horizontally.  The amount of the translation
     * depends on the frame number, which makes the cart move from left to right across
     * the screen as the animation progresses.  The cart animation repeats every 300
     * frames.  At the beginning of the animation, the cart is off the left edge of the
     * screen. */

    glPushMatrix();
    glTranslated(-3 + 13*(frameNumber % 300) / 300.0, 0, 0);
    glScaled(0.3,0.3,1);
    drawCart();
    glPopMatrix();


}  // end display


//------------------------------------------------------------------------------

var animating = false;

function frame() {
    if (animating) {
            frameNumber++;
            draw();
            setTimeout(frame,30);
    }
```

```
}

function doAnimate() {
    animating = document.getElementById("animate").checked;
    if (animating) {
            frame();
    }
}

function init() {
    try {
       canvas = document.getElementById("maincanvas");
       glsimUse(canvas,null); // ( The "null" gives an RGBA color buffer instead of RGB.
    }
    catch (e) {
        document.getElementById("canvas-holder").innerHTML="<p><b>Sorry, an error occurred:<br>" +
                e + "</b></p>";
        return;
    }
    initGL();
    document.getElementById("object").value = "1";
    document.getElementById("object").onchange = draw;
    document.getElementById("animate").checked = false;
    document.getElementById("animate").onchange = doAnimate;
    camera = new Camera();
    camera.setScale(1);
    camera.lookAt(2,2,5, 0,0,0, 0,1,0);
    camera.installTrackball(draw);
    sphere = uvSphere();
    cubeModel = cube();
    cylinder = uvCylinder();
    cone = uvCone();
    torus = uvTorus();
    teapot = teapotModel;  // (This one is just a variable, defined in teapot-model-IFS.js)
    pyramid13 = createThirteenSidedPyramid();  // Initialize the 13-sided pyramid model
    draw();
}

</script>
</head>
<body onload="init()">

<div id="content">

<h3 id="headline">Drawing a Texture</h3>

<div id="canvas-holder">
<canvas id="maincanvas" width="400" height="350"></canvas>
</div>

<br clear=all>

<p style="text-indent:30px"><b>Object</b>:
<select id="object">
   <option value="0">Sphere</option>
   <option value="1">Cube</option>
   <option value="2">Cylinder</option>
   <option value="3">Cone</option>
   <option value="4">Torus</option>
   <option value="5">Teapot</option>
   <option value="6">13-Sided Pyramid</option>
   <option value="7">SHOW 2D SCENE</option>
</select>
<label><input type="checkbox" id="animate" style="margin-left:30px"><b>Animate</b></label></p>
</div>



<div id="help-content" style="display:none">
<h3>About this demo...</h3>
<p>This program demonstrates the use of the OpenGL function <i>glCopyTexImage2D</i>,
which  copies an image from the color buffer (where OpenGL draws its images) into
a texture.  This makes it possible to draw an image with OpenGL and then use
the image as a texture on other objects.</p>
<p>A pop-up menu lets you select the object on which the texture image is
used.  The last entry in the menu is "SHOW 2D SCENE."  In that case,
you see the color buffer just after the 2D scene has been drawn, instead of
seeing a 3D object with the scene as a texture.  Note that the 2D scene
does not fill the canvas.  In fact, it is drawn in a 256-by-256 viewport, since
the width and height of a texture image should be powers of two.</p>
<p>If you turn on animation, the scene shown in the texture image is animated.
A new version of the scene is drawn and copied to the texture for each frame.
In this case, the texture image is a 2D cart-and-windmill animation that should
be familiar from other examples.</p>
<p>As usual, you can rotate the 3D objects using your mouse.</p>
</div>



<!-- support for help text -- do not change. -->
<div id="help-icon">
<img src="../image/question32.png" onclick="showDemoHelp()"
    title="Click here for information about this demo." width="32" height="32">
</div>
<div id="hide-help-icon">
<img src="../image/close32.png" onclick="showDemoHelp()"
    title="Click here to return to the demo." width="65" height="32">
</div>
<div id="helpBG" style="display:none"></div>
</body>
</html>
```

b) kod źródłowy

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
```

```
<title>Textures and Texture Transforms</title>
<link rel="stylesheet" href="demo.css">
<style>
#texcanvas {
    background-image: url("textures/NightEarth-512x256.jpg");
    background-size: 100px 100px;
}
</style>
<script src="script/demo-core.js"></script>
<script src="script/glsim.js"></script>
<script src="script/teapot-model-IFS.js"></script>
<script src="script/basic-object-models-IFS.js"></script>
<script src="script/slider-canvas.js"></script>
<script>

var camera;
var graphics;
var frameNumber = 0;
var sphere, cubeModel, cylinder, cone, torus, teapot, pyramid13;
var sliderScale, sliderTranslateX, sliderTranslateY, sliderRotate;
var scale = 1;
var translateX = 0;
var translateY = 0;
var rotate = 0;
var loadingImages = true;
var textureImages = new Array();
var textureImageURLs = [
    "textures/brick001.jpg",
    "textures/Earth-1024x512.jpg",
    "textures/NightEarth-512x256.jpg",
    "textures/marble.jpg",
    "textures/metal003.gif",
    "textures/mandelbrot.jpeg",
    "textures/clouds.jpg",     // New texture
    "textures/earth.jpg"       // New texture
];

function draw() {
    scale = sliderScale.value(0);
    rotate = sliderRotate.value(0);
    translateX = sliderTranslateX.value(0);
    translateY = sliderTranslateY.value(0);
    drawTextureCanvas();
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    if (loadingImages) {
        return;
    }
    var texnum = Number(document.getElementById("texture").value);
    var objectNumber = Number(document.getElementById("object").value);
    var image = textureImages[texnum];
    glEnable(GL_TEXTURE_2D);

glTexImage2D(GL_TEXTURE_2D,0,GL_RGBA,image.width,image.height,0,GL_RGBA,GL_
UNSIGNED_BYTE,image);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);

    glMatrixMode(GL_TEXTURE);
    glLoadIdentity();
    glTranslatef(translateX,translateY, 0);
    glRotatef(rotate,0,0,1);
    glScalef(scale,scale,1);
    glMatrixMode(GL_MODELVIEW);

    camera.apply();

    switch(objectNumber) {
    case 0:
        glRotatef(-90,1,0,0);
        glScalef(1.5,1.5,1.5);
        drawModel(sphere);
        break;
    case 1:
        glScalef(1.2,1.2,1.2);
        drawModel(cubeModel);
        break;
    case 2:
        glRotatef(-90,1,0,0);
        glScalef(1.3,1.3,1.3);
        drawModel(cylinder);
        break;
    case 3:
        glRotatef(-90,1,0,0);
        glScalef(1.3,1.3,1.3);
        drawModel(cone);
        break;
    case 4:
        glScalef(1.6,1.6,1.6);
        drawModel(torus);
        break;
    case 5:
        glScalef(0.06, 0.06, 0.06);
        drawModel(teapot);
        break;
    case 6:
        glScalef(1.5, 1.5, 1.5);
        drawModel(pyramid13);
        break;
    }
}

function drawTextureCanvas() {
    if (loadingImages) {
        graphics.fillStyle = "white";
        graphics.fillRect(0,0,300,300);
        graphics.fillStyle = "black";
        graphics.font = "14px serif";
        graphics.fillText("Waiting for images to load...", 10, 40);
        return;
    }
    graphics.clearRect(0,0,300,300);
    graphics.save();
    graphics.translate(100,200);
    graphics.scale(1,-1);
    graphics.translate(translateX*100,translateY*100);
    graphics.rotate(rotate/180 * Math.PI);
    graphics.scale(scale,scale);
    graphics.lineWidth = 5/scale;
    graphics.strokeStyle = "white";
    graphics.strokeRect(-.5,-.5,100,100);
    graphics.lineWidth = 1/scale;
    graphics.strokeStyle = "black";
    graphics.strokeRect(-.5,-.5,100,100);
    graphics.restore();
}

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_LIGHTING);
    glEnable(GL_LIGHT0);
```

```
   glEnable(GL_LIGHT1); // Enable a second light source
   glEnable(GL_NORMALIZE);
   glEnable(GL_DEPTH_TEST);
   glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE, [ 1, 1, 1, 1 ]);
   glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, [ 1, 1, 1, 1 ]);
   glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, 50.0); // Increase shininess
   glLightfv(GL_LIGHT0, GL_POSITION, [0, 0, 1, 0]); // Light from the front
   glLightfv(GL_LIGHT0, GL_DIFFUSE, [1, 1, 1, 1]); // White light
   glLightfv(GL_LIGHT1, GL_POSITION, [1, 1, 1, 0]); // Additional light from an angle
   glLightfv(GL_LIGHT1, GL_DIFFUSE, [0.8, 0.8, 0.8, 1]); // Brighter white light
   glClearColor(0,0,0,1);
}

function loadImages() {
   var loadedCt = 0;
   for (var i = 0; i < textureImageURLs.length; i++) {
      textureImages[i] = new Image();
      textureImages[i].onload = imageLoaded;
      textureImages[i].src = textureImageURLs[i];
   }
   function imageLoaded() {
      loadedCt++;
      if (loadedCt == textureImageURLs.length) {
         loadingImages = false;
         glEnable(GL_TEXTURE_2D);
         var texnum = Number(document.getElementById("texture").value);
         var image = textureImages[texnum];
         try {

glTexImage2D(GL_TEXTURE_2D,0,GL_RGBA,image.width,image.height,0,GL_RGBA,GL_
UNSIGNED_BYTE,image);
         } catch(e) {
            document.getElementById("headline").innerHTML="Can't access texture.<br>Note:
Some browsers can't use a file from a local disk."
            return;
         }
         glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
         draw();
         document.getElementById("object").disabled = false;
         document.getElementById("texture").disabled = false;
      }
   }
}

function changeTexture() {
   var texnum = Number(document.getElementById("texture").value);
   document.getElementById("texcanvas").style.backgroundImage = "url('" +
textureImageURLs[texnum] + "')";
```

```
   var image = textureImages[texnum];

glTexImage2D(GL_TEXTURE_2D,0,GL_RGBA,image.width,image.height,0,GL_RGBA,GL_
UNSIGNED_BYTE,image);
   draw();
}

function doReset() {
   sliderRotate.setValue(0,0);
   sliderScale.setValue(0,1);
   sliderTranslateX.setValue(0,0);
   sliderTranslateY.setValue(0,0);
   camera.lookAt(10,7,20);
   draw();
}

function createThirteenSidedPyramid() {
   var numSides = 13;
   var vertices = [];
   var indices = [];
   var normals = [];
   var texCoords = [];
   var angleStep = 2 * Math.PI / numSides;
   var radius = 0.5;

   for (var i = 0; i < numSides; i++) {
      var angle = i * angleStep;
      vertices.push(radius * Math.cos(angle), radius * Math.sin(angle), 0);
      normals.push(0, 0, -1);
      texCoords.push((1 + Math.cos(angle)) / 2, (1 + Math.sin(angle)) / 2);
   }

   vertices.push(0, 0, 1);
   normals.push(0, 0, 1);
   texCoords.push(0.5, 0.5);

   for (var i = 1; i <= numSides; i++) {
      indices.push(0, i % numSides, (i + 1) % numSides);
   }

   var apexIndex = numSides;
   for (var i = 0; i < numSides; i++) {
      indices.push(apexIndex, i, (i + 1) % numSides);
      var normal = calculateNormal(
         [vertices[3 * apexIndex], vertices[3 * apexIndex + 1], vertices[3 * apexIndex + 2]],
         [vertices[3 * i], vertices[3 * i + 1], vertices[3 * i + 2]],
         [vertices[3 * ((i + 1) % numSides)], vertices[3 * ((i + 1) % numSides) + 1], vertices[3 *
((i + 1) % numSides) + 2]]
```

```
        );
        normals.push(...normal);
        normals.push(...normal);
        normals.push(...normal);
        texCoords.push(0.5, 1, (1 + Math.cos(i * angleStep)) / 2, (1 + Math.sin(i * angleStep)) / 2,
(1 + Math.cos((i + 1) * angleStep)) / 2, (1 + Math.sin((i + 1) * angleStep)) / 2);
    }

    return {
        vertexPositions: new Float32Array(vertices),
        vertexNormals: new Float32Array(normals),
        vertexTextureCoords: new Float32Array(texCoords),
        indices: new Uint8Array(indices)
    };
}

function calculateNormal(v1, v2, v3) {
    var U = [v2[0] - v1[0], v2[1] - v1[1], v2[2] - v1[2]];
    var V = [v3[0] - v1[0], v3[1] - v1[1], v3[2] - v1[2]];
    var normal = [
        U[1] * V[2] - U[2] * V[1],
        U[2] * V[0] - U[0] * V[2],
        U[0] * V[1] - U[1] * V[0]
    ];
    var length = Math.sqrt(normal[0] * normal[0] + normal[1] * normal[1] + normal[2] *
normal[2]);
    return [normal[0] / length, normal[1] / length, normal[2] / length]; // Normalize the normal
}

function init() {
    try {
        glsimUse("maincanvas");
        var texcanvas = document.getElementById("texcanvas");
        graphics = texcanvas.getContext('2d');
    }
    catch (e) {
        document.getElementById("canvas-holder").innerHTML="<p><b>Sorry, an error
occurred:<br>" +
                e + "</b></p>";
        return;
    }
    document.getElementById("reset").onclick = doReset;
    document.getElementById("object").value = "1";
    document.getElementById("object").onchange = draw;
    document.getElementById("texture").value = "2";
    document.getElementById("texture").onchange = changeTexture;
    document.getElementById("object").disabled = true;
    document.getElementById("texture").disabled = true;
```

```
   sliderRotate = new SliderCanvas(document.getElementById("scRotate"));
   sliderRotate.addSlider({label:"rotate", min:-180, max:180, value:0});
   sliderScale = new SliderCanvas(document.getElementById("scScale"));
   sliderScale.addSlider({label:"scale", min: 0.5, max:2, step:0.01, value:1, decimals:2});
   sliderTranslateX = new SliderCanvas(document.getElementById("scTransX"));
   sliderTranslateX.addSlider({label:"x-trans.", min: -0.5, max:0.5, step:0.01, value:0,
decimals:2});
   sliderTranslateY = new SliderCanvas(document.getElementById("scTransY"));
   sliderTranslateY.addSlider({label:"y-trans.", min: -0.5, max:0.5, step:0.01, value:0,
decimals:2});
   sliderRotate.onChange = draw;
   sliderScale.onChange = draw;
   sliderTranslateX.onChange = draw;
   sliderTranslateY.onChange = draw;
   initGL();
   camera = new Camera();
   camera.setScale(1);
   camera.lookAt(10,7,20);
   camera.installTrackball(draw);
   sphere = uvSphere();
   cubeModel = cube();
   cylinder = uvCylinder();
   cone = uvCone();
   torus = uvTorus();
   teapot = teapotModel;
   pyramid13 = createThirteenSidedPyramid();
   sliderRotate.draw();
   sliderScale.draw();
   sliderTranslateX.draw();
   sliderTranslateY.draw();
   drawTextureCanvas();
   loadImages();
}

</script>
</head>
<body onload="init()">

<div id="content">

<h3 id="headline">Textures and Texture Transforms</h3>
<table border="0" cellspacing="0" cellpadding="8">
<tr>
<td><div id="canvas-holder"><canvas id="texcanvas" width="300"
height="300"></canvas></div></td>
<td><canvas id="maincanvas" width="300" height="300"></canvas></td>
</tr>
<tr align="center">
```

```
<td colspan="2">
    <b>Texture:</b>
    <select id="texture">
        <option value="0">Brick</option>
        <option value="1">Topographic Earth</option>
        <option value="2">Earth At Night</option>
        <option value="3">Marble</option>
        <option value="4">Metal</option>
        <option value="5">Mandelbrot</option>
        <option value="6">Clouds</option>    <!-- New option -->
        <option value="7">Earth</option>     <!-- New option -->
    </select>
    <b style="margin-left:40px">Object</b>:
    <select id="object">
       <option value="0">Sphere</option>
       <option value="1">Cube</option>
       <option value="2">Cylinder</option>
       <option value="3">Cone</option>
       <option value="4">Torus</option>
       <option value="5">Teapot</option>
       <option value="6">13-Sided Pyramid</option>
    </select>
    <button id="reset" style="margin-left:40px">Reset</button>
</td>
</tr>
<tr align=center>
<td>
    <canvas id="scScale" width="280" height="50"></canvas>
</td>
<td>
    <canvas id="scTransX" width="280" height="50"></canvas>
</td>
</tr>
<tr align=center>
<td>
    <canvas id="scRotate" width="280" height="50"></canvas>
</td>
<td>
    <canvas id="scTransY" width="280" height="50"></canvas></td>
</tr>
</table>

</div>

<div id="help-content" style="display:none">
<h3>About this demo...</h3>
<p>Textured objects are shown in the display at the upper right.
Use the pop-up menus to select the texture and the object that
```

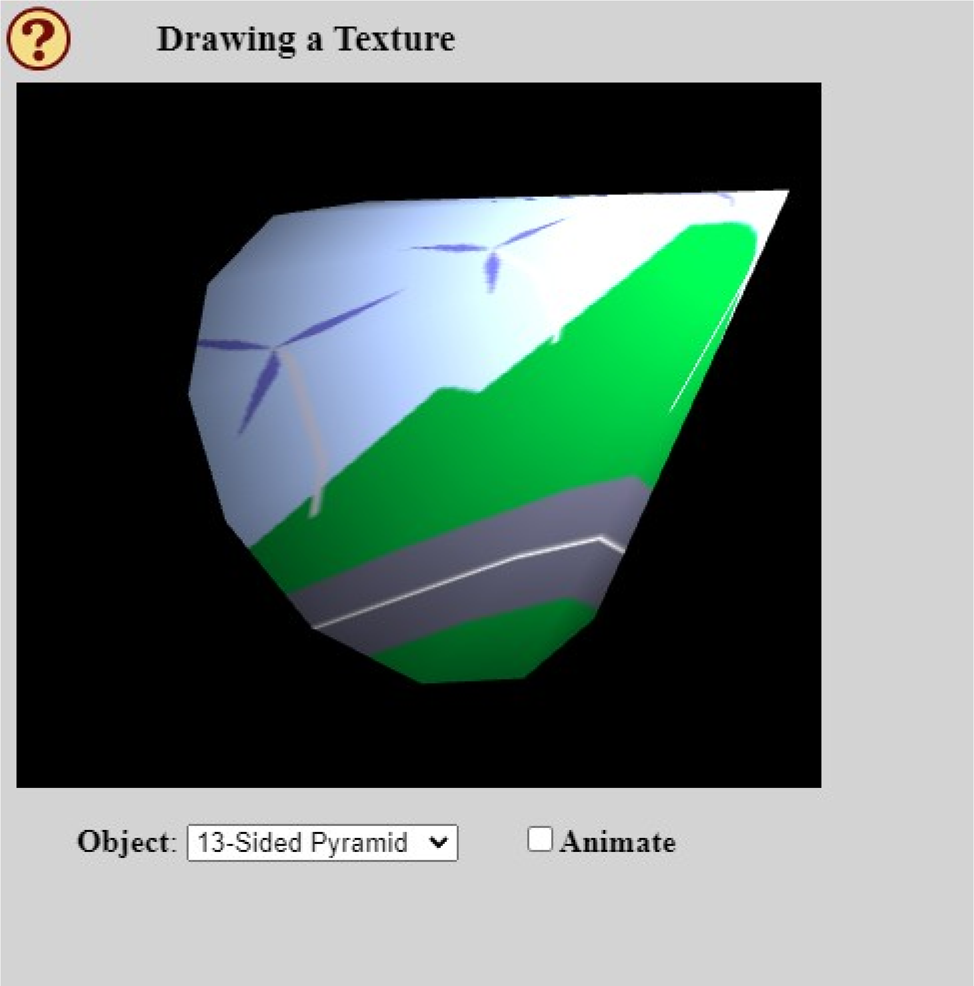
```
you want to view.  You can use your mouse to rotate the objects.</p>
<p>The display on the upper left shows the <i>st</i>-plane, where
the texture lives.  In the display, <i>s</i> and <i>t</i> range
from &minus;1 to 2.  A box is drawn around the original texture
image, with <i>s</i> and <i>t</i> ranging from 0 to 1.  The
point (0,0) is at the lower left corner of that box.  Note that
(0,0) is not at the center of the display.</p>
<p>Four sliders allow you apply texture transformations.  The
transform can be seen as a coordinate transformation in the <i>st</i>-plane, or
as a modeling transformation that applies to the box.  As a modeling transform,
the box is first scaled, then rotated, then translated.  </p>
<p>The same transform is also applied as a texture transformation on the textured
object.  The result
is easiest to see on the cube: <i>Each face of the cube shows a copy
of the part of the <i>st</i>-plane that is inside the box.</i> </p>
<p>Try adjusting just
one slider while the others are at their default values.
(Click "Reset" between experiments.)  Note that when the
box is translated to the left, the image on the cube moves to the right.
When the box grows, the image on the cube shrinks (because you are
seeing a larger region in the <i>st</i>-plane mapped to the same
area on the cube).  When the box rotates counterclockwise, the
image on the cube rotates clockwise.</p>
<p>Although the effect is easiest to understand on the cube, it's
fun to watch a texture moving around on an object.  Try it!
I especially like a rotating texture on the torus.</p>
</div>

<div id="help-icon">
<img src="../image/question32.png" onclick="showDemoHelp()"
   title="Click here for information about this demo." width="32" height="32">
</div>
<div id="hide-help-icon">
<img src="../image/close32.png" onclick="showDemoHelp()"
   title="Click here to return to the demo." width="65" height="32">
</div>
<div id="helpBG" style="display:none"></div>
</body>
</html>
```

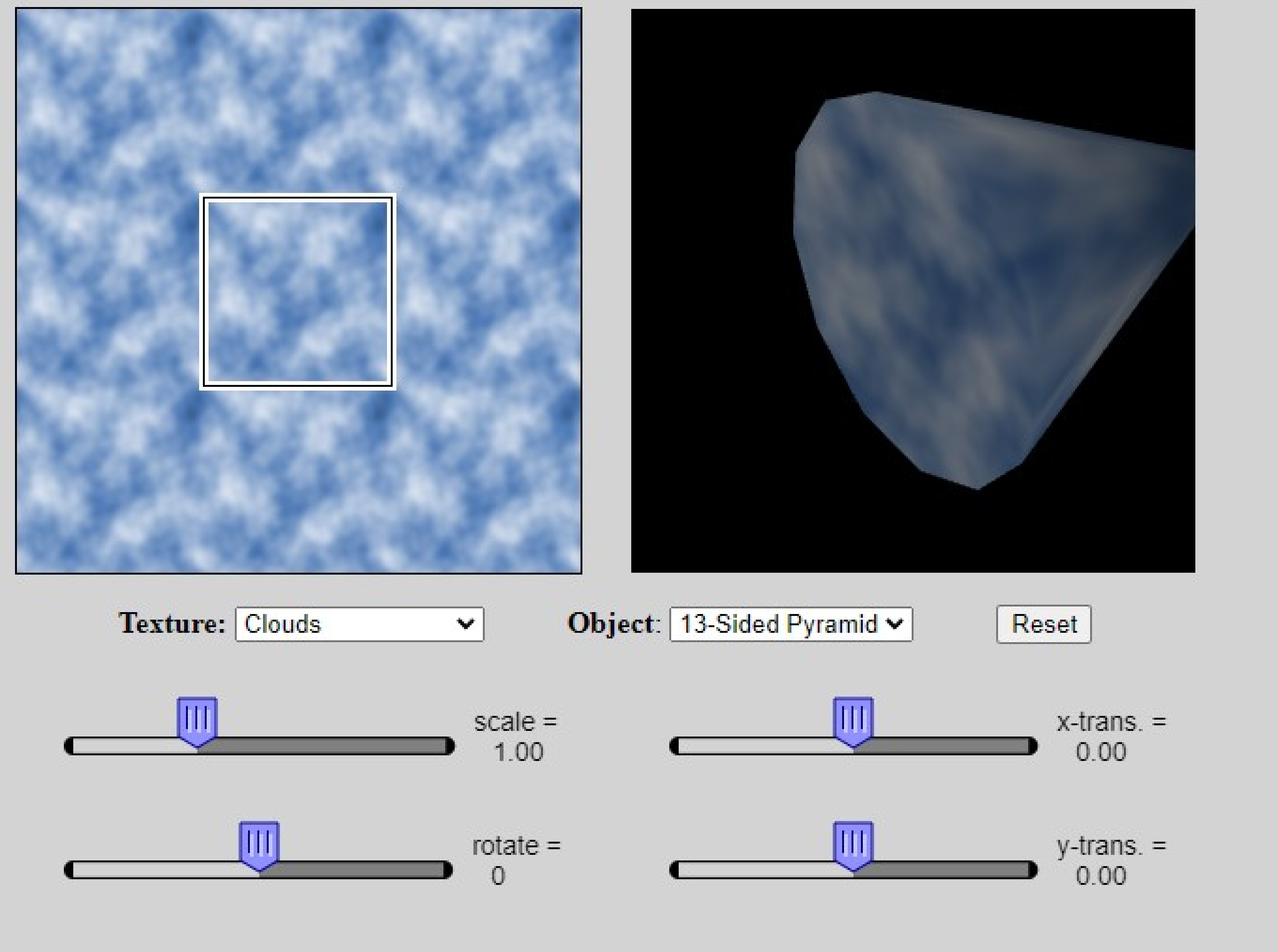
GitHub: https://github.com/GabrielMrzyglod/Grafika_Lab7

4. Wynik działania:

Zadanie a



Zadanie b



Texture: Clouds

Object: 13-Sided Pyramid

scale = 1.00

x-trans. = 0.00

rotate = 0

y-trans. = 0.00

5. Wnioski:

Ładowanie tekstury z buforu kolorów (rysowanie w Panelu) zapewnia wysoką wydajność, eliminując konieczność odczytu z pliku, co przyspiesza ładowanie tekstur. Umożliwia to również dynamiczne generowanie i modyfikowanie tekstur w czasie rzeczywistym, co jest przydatne w aplikacjach wymagających szybkiej zmiany wyglądu obiektów. Jednak ta metoda może być bardziej skomplikowana w implementacji, wymagając zaawansowanej wiedzy na temat grafiki komputerowej.

Ładowanie tekstury z pliku jest prostsze do zaimplementowania, ponieważ wykorzystuje gotowe obrazy jako tekstury. Umożliwia to użycie wysokiej jakości tekstur z szerokiej gamy źródeł, co może znacząco poprawić wizualną jakość sceny. Wadą tej metody jest dłuższy czas ładowania tekstur, szczególnie dla dużych plików, co może wpływać na wydajność aplikacji.