National Technical University of Ukraine
“Igor Sikorsky Kyiv Polytechnic Institute”
Educational and Scientific Institute of Atomic and Thermal Energy

Visualization of graphical and geometric information

Calculation and graphics work

(Operations on texture coordinates)

Prepared by:
group TR-23mp
Voronin Oleksii

Kyiv - 2023

Description of the task

Operations on texture coordinates

Scale and rotate a texture around user specified point.

Requirements

- Map the texture over the surface from practical assignment #2.
- Implement texture scaling (texture coordinates) scaling / rotation around user specified point- <u>odd variants implement scaling, even variants implement rotation</u>
- It has to be possible to move the point along the surface (u,v) space using a keyboard. E.g. keys **A** and **D** move the point along u parameter and keys **W** and **S** move the point along v parameter.

Theory

The Klein Surface (The Klein Bottle)

The most known variant of the Klein bottle presents a bent tube of a variable diameter (Fig. 1). The narrower end goes through the wall of the tube and goes out through another wider opening of the tube, so that both boundary circles dispose concentrically. Bending the wide end of the tube inside and the narrow end outside one can join the both ends of the tube without peculiarities.

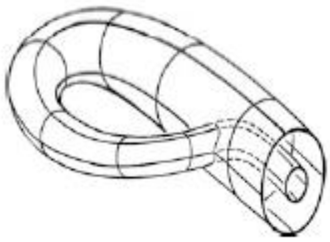


Fig. 1

Forms of definition of the Klein surface

(1) Parametrical equations:

$$x = x(u,v) = \left(a + \cos\frac{u}{2}\sin v - \sin\frac{u}{2}\sin 2v\right)\cos u,$$
$$y = y(u,v) = \left(a + \cos\frac{u}{2}\sin v - \sin\frac{u}{2}\sin 2v\right)\sin u,$$
$$z = z(u,v) = \sin\frac{u}{2}\sin v + \cos\frac{u}{2}\sin 2v,$$

where $a > 2, \quad 0 \le u \le 2\pi; \quad 0 \le v \le 2\pi$

(2) Vector equation:

$$\boldsymbol{r} = \boldsymbol{r}(u,v) = \left(R_x, R_y, R_z\right),$$

where Rx, Ry, Rz are components at Cartesian coordinates of the following vector-function

$$\boldsymbol{R}(u,v) = \boldsymbol{R}_0(u) + \rho(u)[e_1(u)\cos v + e_2 \sin v],$$
$$\boldsymbol{R}_0(u) = \begin{pmatrix} a\sin 2u \\ 0 \\ b\cos u \end{pmatrix}, \quad \boldsymbol{e}_1(u) = \begin{pmatrix} b\sin u \\ 0 \\ a\cos 2u \end{pmatrix}\frac{1}{|R_0'|}, \quad \boldsymbol{e}_2 = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix},$$
$$\rho = \rho(u) = \rho_0 + \rho_1 \sin^{2n}(u - u_0),$$

where e1 and e2 are the unit vectors orthogonal to R00. Here, for example, we may take

$$\rho_0 = 0.1;\ \rho_1 = 0.6; \quad a = 0.25;\ b = 1;$$
$$u_0 = 0.3;\ n = 4;$$
$$0 \le u \le \pi;\ 0 \le v \le 2\pi.$$

In Fig. 2, some modifications of the Klein surface are shown. A one-sided topological surface in question has no inside or outside.

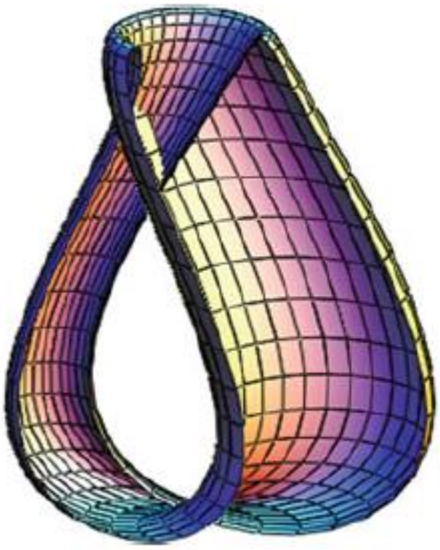


Fig. 2

Implementation details

The Klein Surface surface is a type of mathematical surface that has minimal surface area for a given boundary. It is a continuous function of two variables, and it can be represented by a parametric equation in three-dimensional space.

To implement The Klein Surface minimal surface in WebGL, I did the following:

1. Wrote a function to generate surface vertices: first wrote a function that generates surface vertices using the parametric equation for the Richmond minimum surface. This function takes two parameters, u and v, which represent the coordinates in the parametric domain of the surface. It returns an array of three-dimensional vertices, each represented as an array of three coordinates.

2. Wrote a function to generate surface normals: Wrote a function that generates surface normals for each vertex. They are used to calculate the illumination on the surface.

3. Wrote a function to generate surface texture coordinates: to display a texture on a surface, also wrote a function that generates texture coordinates for each vertex. The texture coordinates determine the display of the texture image on the surface.

4. Loaded the texture image: inserted a reference to the texture image using the image element, then created a texture object from the image using gl.createTexture(). I also bound the texture object to the gl.TEXTURE_2D target using gl.bindTexture() and set the texture parameters using gl.texParameteri().

5. Adjusted texture and coordinate vertex buffers: Created texture and coordinate vertex buffers and filled them with data using gl.bufferData(). I also bound the buffers and set the vertex attribute pointers using gl.vertexAttribPointer() to pass the data to the shader program.

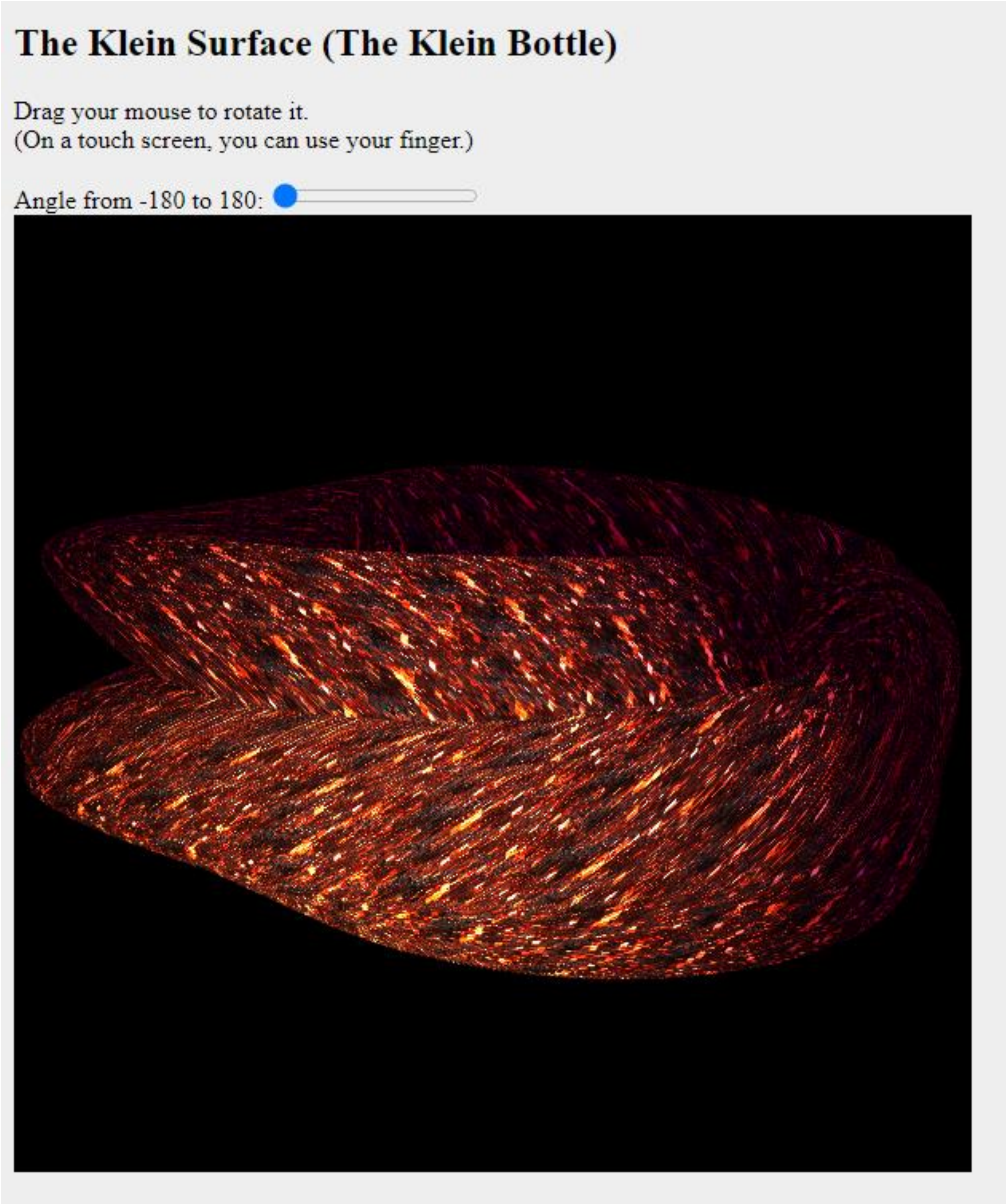
6. Test in the shader program: enabled texturing in the vertex and fragment shaders by declaring the sampler2D uniform and the vTexCoord attribute and using the texture2D() function to sample the texture value in texture coordinates. Bind the texture object to the texture block using gl.activeTexture() and gl.uniform1i() and passed the texture block to the shader program as a uniform.

7. Drawn a surface with texturing enabled: In the Draw() method, enabled texturing in the WebGL backend by enabling the gl.TEXTURE_2D capability and

binding the texture object to the gl.TEXTURE_2D target. I also configured the texture coordinate attribute and enabled it.

8. Lighting: From previous lab work in the fragment shader, I calculated the lighting on the surface using surface normals, light position and surface material properties.

User's instruction with screenshots



Keys **A** and **D** - move the point along u parameter

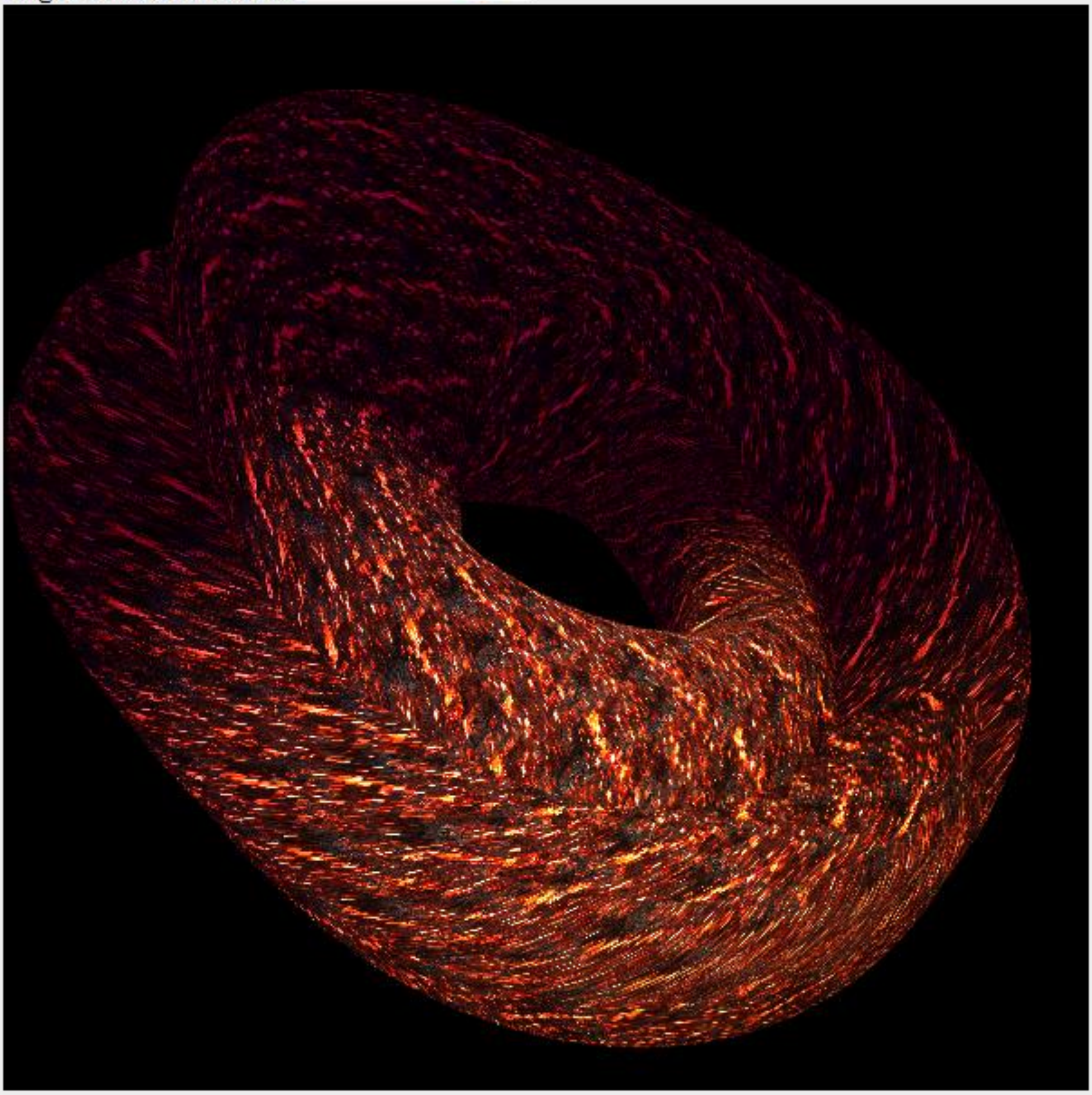
Keys **W** and **S** - move the point along v parameter

Left/right keyboard keys - move the light

The Klein Surface (The Klein Bottle)

Drag your mouse to rotate it.
(On a touch screen, you can use your finger.)

Angle from -180 to 180:



Source code

```
// Vertex shader
const vertexShaderSource = `
attribute vec3 vertex;
attribute vec3 normal;
attribute vec2 textureCoords;

uniform mat4 normalMatrix;
uniform mat4 ModelViewProjectionMatrix;

uniform float shininess;
uniform vec3 ambientColor;
uniform vec3 diffuseColor;
uniform vec3 specularColor;

uniform vec3 lightPosition;

uniform float textureAngle;
uniform vec2 texturePoint;

varying vec4 color;
varying vec2 vTextureCoords;

mat4 getRotateMatix(float angleRad) {
  float c = cos(angleRad);
  float s = sin(angleRad);

  return mat4(
    vec4(c, s, 0.0, 0.0),
    vec4(-s, c, 0.0, 0.0),
    vec4(0.0, 0.0, 1.0, 0.0),
    vec4(0.0, 0.0, 0.0, 1.0)
  );
}

mat4 getTranslateMatrix(vec2 point) {
  return mat4(
    vec4(1.0, 0.0, 0.0, point.x),
    vec4(0.0, 1.0, 0.0, point.y),
    vec4(0.0, 0.0, 1.0, 0.0),
    vec4(0.0, 0.0, 0.0, 1.0)
  );
}

void main() {
    vec4 vertexPosition4 = ModelViewProjectionMatrix * vec4(vertex, 1.0);
    vec3 vertexPosition = vec3(vertexPosition4) / vertexPosition4.w;
    vec3 normalInterpolation = vec3(normalMatrix * vec4(normal, 0.0));
    gl_Position = vertexPosition4;

    vec3 normal = normalize(normalInterpolation);
    vec3 lightDirection = normalize(lightPosition - vertexPosition);
```

```
    float nDotLight = max(dot(normal, lightDirection), 0.0);
    float specularLight = 0.0;
    if (nDotLight > 0.0) {
        vec3 viewDirection = normalize(-vertexPosition);
        vec3 halfDirection = normalize(lightDirection + viewDirection);
        float specularAngle = max(dot(halfDirection, normal), 0.0);
        specularLight = pow(specularAngle, shininess);
    }
    vec3 diffuse = nDotLight * diffuseColor;
    vec3 ambient = ambientColor;
    vec3 specular = specularLight * specularColor;

    mat4 rotatedMatrix = getRotateMatix(textureAngle);
    mat4 translatedMatrix = getTranslateMatrix(-texturePoint);
    mat4 translatedBackMatrix = getTranslateMatrix(texturePoint);

    vec4 vTranslatedMatrix = translatedMatrix * vec4(textureCoords, 0, 0);
    vec4 vRotatedMatrix = vTranslatedMatrix * rotatedMatrix;
    vec4 vTranslatedBackMatrix = vRotatedMatrix * translatedBackMatrix;

    vTextureCoords = vec2(vTranslatedBackMatrix);

    color = vec4(diffuse + ambient + specular, 1.0);
}`;
```

Fragment shader

```
const fragmentShaderSource = `
#ifdef GL_FRAGMENT_PRECISION_HIGH
    precision highp float;
#else
    precision mediump float;
#endif

varying vec4 color;
varying vec2 vTextureCoords;
uniform sampler2D textureU;

void main() {
  vec4 texture = texture2D(textureU, vTextureCoords);
  gl_FragColor = texture * color;
```

Parametric function of the surface

```
function CreateSurfaceData()
{
    let vertexList = [];
    let textureList = [];
    let splines = 20;

    let maxU = Math.PI;
    let maxV = 2 * Math.PI;
    let stepU = step(maxU, splines);
    let stepV = step(maxV, splines);

    let getU = (u) => {
      return u / maxU;
    };

    let getV = (v) => {
      return v / maxV;
    };

    const POINTS = 60
    let a = 2.000001

    for (let u = 0; u <= POINTS; u += stepU) {

        const U = u * 2 * Math.PI / POINTS;

        for (let v = 0; v <= POINTS; v += stepV) {

            const V = v * 2 * Math.PI / POINTS;

            const x = (a + Math.cos(U / 2) * Math.sin(V) - Math.sin(U / 2) * Math.sin(2 * V)) * Math.cos(U)
            const y = (a + Math.cos(U / 2) * Math.sin(V) - Math.sin(U / 2) * Math.sin(2 * V)) * Math.sin(U)
            const z = Math.sin(U / 2) * Math.sin(V) + Math.cos(U / 2) * Math.sin(2 * V)

            vertexList.push(x/2,y/2,z/2)
            textureList.push(getU(u), getV(v));

            const x1 = (a + Math.cos(U / 2 + stepU) * Math.sin(V + stepV) - Math.sin(U / 2 + stepU) * Math.sin(2 * V + stepV)) * Math.cos(U + stepU)
            const y1 = (a + Math.cos(U / 2 + stepU) * Math.sin(V + stepV) - Math.sin(U / 2 + stepU) * Math.sin(2 * V + stepV)) * Math.sin(U + stepU)
            const z1 = Math.sin(U / 2 + stepU) * Math.sin(V + stepV) + Math.cos(U / 2 + stepU) * Math.sin(2 * V + stepV)

            vertexList.push(x1/2,y1/2,z1/2)
            textureList.push(getU(u + stepU), getV(v + stepV));
        }

    }

    return { vertexList, textureList };
}
```

Parametric function of the texture

```
const loadTexture = () => {
  const image = new Image();
  image.crossOrigin = "anonymous";
  image.src =
    "https://www.the3rdsequence.com/texturedb/download/259/texture/jpg/1024/burning+hot+lava-1024x1024.jpg";

  image.addEventListener("load", () => {
    const texture = gl.createTexture();
    gl.bindTexture(gl.TEXTURE_2D, texture);
    gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR);
    gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.LINEAR);
    gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE, image);
    draw();
  });
};
```