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:

Probotiuk Anatoliy

group TR-23mp

Kyiv 2022

Description of the task

Subject: Operations on texture coordinates

Scale and rotate a texture around user specified point. Figure - Surface of Revolution of a General Sinusoid.

Requirements

- Map the texture over the surface from practical assignment #2.
- Implement texture scaling (texture coordinates) scaling / rotation around user specified point- odd variants implement scaling, even variants implement rotation
- 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

Surface of Revolution of a General Sinusoid

A surface of revolution of a general sinusoid

$$z = a\,\sin(n\pi x/R + \pi/2) = a\cos(n\pi x/R)$$

about an axis Oz is used in technics. General sinusoid in contrast to usual sinusoid (z = sin x) is elongated jaj times along the axis Oz and contracted R/(nπ) times along the axis Ox, where n is an integer, R is a dimension of an integer n of half-waves of the sinusoid, and is shifted to the left by a straight-line segment R/(2n). A period of the function is T = 2R/n. The points of intersection of the sine function with the Ox axis have the coordinates [(k + 1⁄2)R/n, 0]. A surface of revolution of a general sinusoid has the parts of positive and negative Gaussian curvatures. This surface can be reckoned in a subclass of waving or corrugated surfaces.

Forms of definition of the surface

(1) Parametrical equations (Fig. 1):

$$x = x(r,\beta) = r\cos\beta, \quad y = y(r,\beta) = r\sin\beta,$$
$$z = z(r) = a\cos\frac{n\pi r}{R}.$$

Coefficients of the fundamental forms of the surface and its principal curvatures:

$$A^2 = 1 + \frac{a^2n^2\pi^2}{R^2}\sin^2\frac{n\pi r}{R},$$
$$F = 0,\ B = r,$$

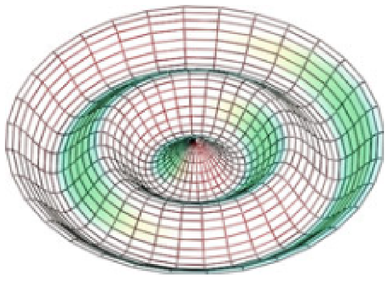


Fig. 1

$$L = -\frac{an^2\pi^2}{AR^2}\cos\frac{n\pi r}{R},$$
$$M = 0,\ N = -\frac{an\pi}{AR} r\sin\frac{n\pi r}{R},$$
$$k_1 = k_r = -\frac{an^2\pi^2}{A^3R^2}\cos\frac{n\pi r}{R},$$
$$k_2 = k_\beta = -\frac{an\pi}{rAR}\sin\frac{n\pi r}{R},$$
$$K = \frac{a^2n^3\pi^3}{2rA^4R^3}\sin\frac{2n\pi r}{R}.$$

The curvilinear coordinate net is put down to lines of principal curvatures.

(2) Parametrical equations (Fig. 2):

$$x = x(r,\beta) = r\cos\beta, \quad y = y(r,\beta) = r\sin\beta,$$
$$z = z(r) = a\sin\frac{n\pi r}{R}.$$

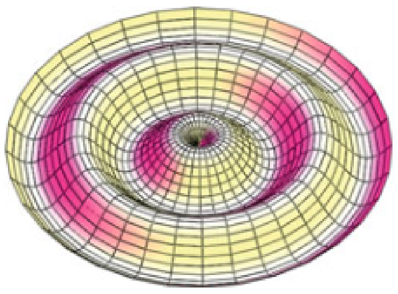


Fig. 2

Implementation details

To implement operations on texture coordinates with a surface of revolution of a general sinusoid, I followed these steps:

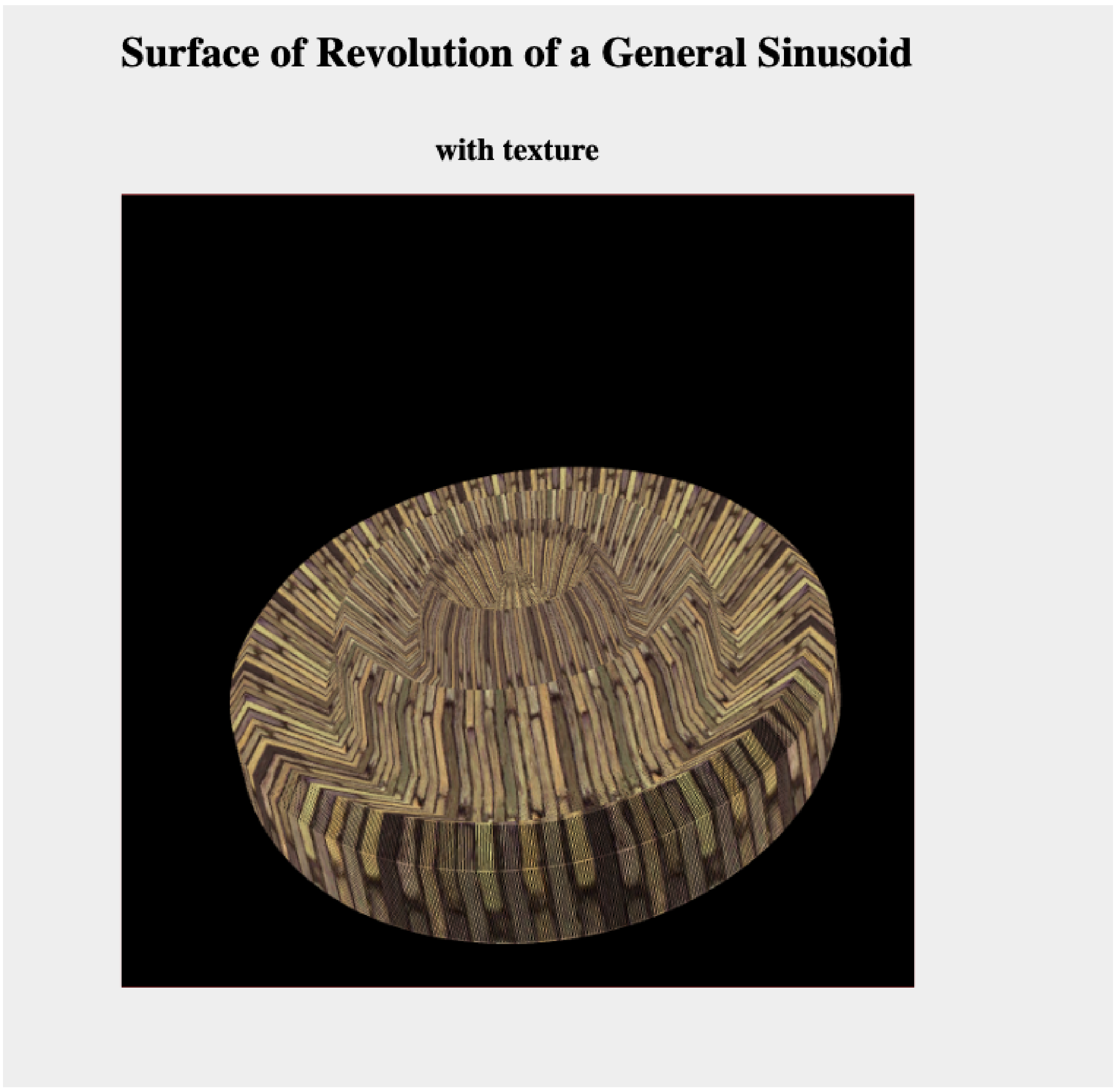
1. Defined the surface of revolution: In this step, I defined the surface of revolution by specifying the parameters of the general sinusoid and the resolution of the surface. I wrote functions to calculate the x, y, and z coordinates of the surface at each point.
2. Generated the vertices and texture coordinates of the surface and stored the resulting x, y, and z coordinates and texture coordinates in separate arrays.
3. Created a buffer object for the vertices and texture coordinates of the surface. A buffer object is an object that stores data in the GPU memory and is used to pass data to the GPU for rendering.
4. Binded the buffer object and passed the data to the GPU with gl.bindBuffer and gl.bufferData functions.
5. Drawed the surface by calling the gl.drawArrays function and specifying the type of primitive.

In addition, I had the task of scaling the technology with the help of point movement along the surface (u,v) space using a keyboard. Keys A and D move the point along u parameter and keys W and S move the point along v parameter.

To scale and offset a texture in WebGL, I used the `u_scale` and `u_offset` uniform variables in my vertex shader and passed the values to the shader using gl.uniform1f. Then, I used these uniform variables to modify the texture coordinates passed to the fragment shader. And after clicking on “WASD” I redraw the canvas with my figure.

User's instruction

After all the work, my figure looks like this.

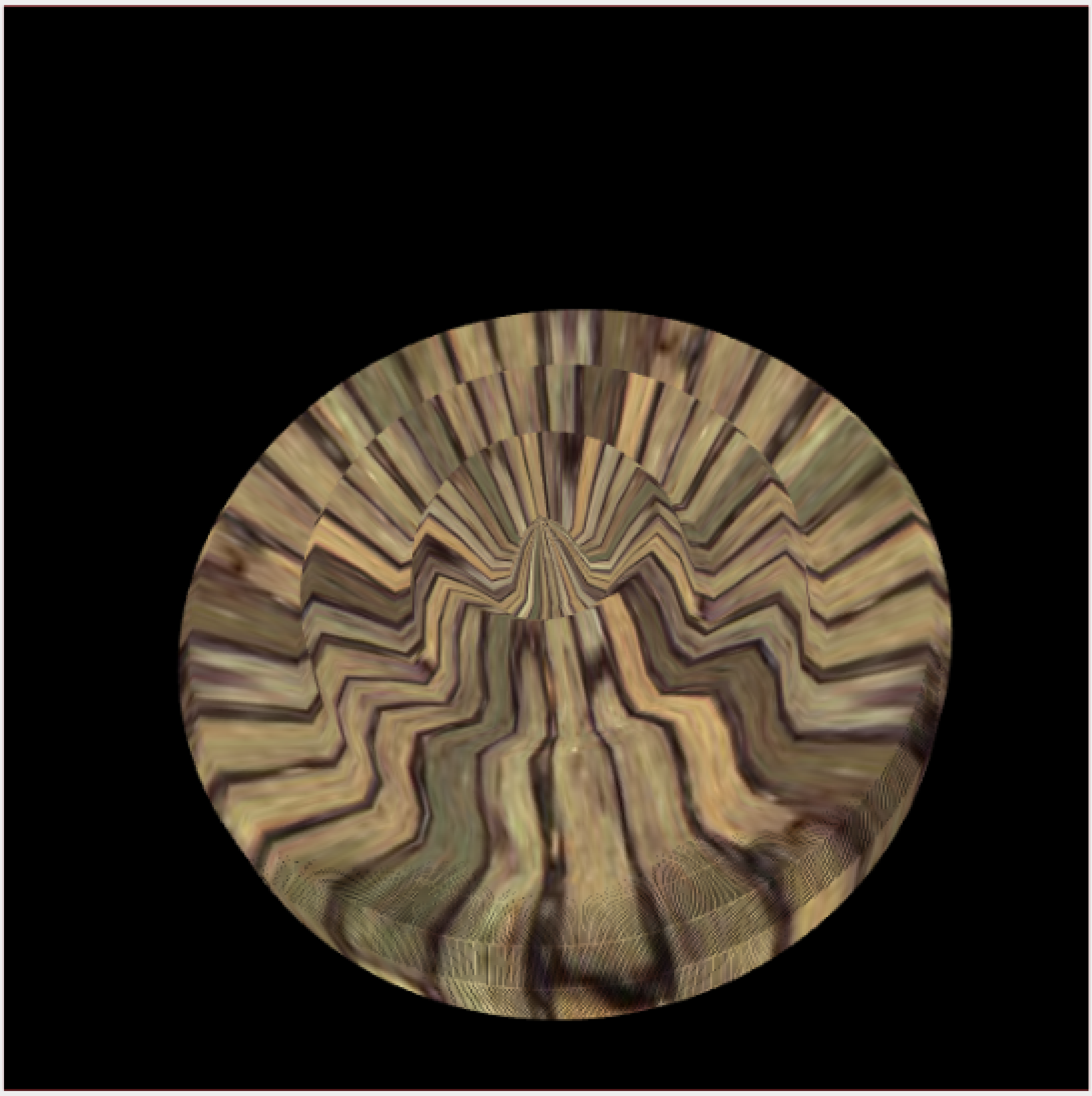


You can drag your mouse on the cube to rotate it or if you use touch screen you can use your finger.

If you click on A or D you can change the offset of the texture.
If you click on W or S you can change the figure scale.

Surface of Revolution of a General Sinusoid

with texture



Source code

Parametric function of the surface

```
function CreateSurfaceData() {

   let vertexCircles = []
   let texCoordsCircles = []

   for (let r = 0; r <= 7; r += Math.PI / 8) {
       let vertexCircle = []
       let texCoordsCircle = []
       for (let B = 0; B <= 2 * Math.PI; B += Math.PI / 700) {
           vertexCircle.push([x(r, B), y(r, B), z(r)])

           const u = r / 7;
           const v = B / 2 * Math.PI;
           texCoordsCircle.push([u, v]);
       }
       vertexCircles.push(vertexCircle)
       texCoordsCircles.push(texCoordsCircle)
   }

   let vertexLines = vertexCircles[0].map((col, i) => vertexCircles.map(row => row[i]));
   let texCoordsLines = texCoordsCircles[0].map((col, i) => texCoordsCircles.map(row =>
row[i]));

   vertexLines = vertexLines.map(vertexLine => [...vertexLine,
...vertexLine.slice().reverse()])
   vertexCircles = vertexCircles.map(vertexCircle => [...vertexCircle,
...vertexCircle.slice().reverse()])

   texCoordsLines = texCoordsLines.map(texCoordsLine => [...texCoordsLine,
...texCoordsLine.slice().reverse()])
   texCoordsCircles = texCoordsCircles.map(texCoordsCircle => [...texCoordsCircle,
...texCoordsCircle.slice().reverse()])

   const vertex = [...vertexLines.flat(Infinity), ...vertexCircles.flat(Infinity)]
   const texCoords = [...texCoordsLines.flat(Infinity),
...texCoordsCircles.flat(Infinity)]

   return [
       vertex, texCoords
   ]
}
```

Creating texture

```
function createTexture() {

   let texture = gl.createTexture();

   gl.bindTexture(gl.TEXTURE_2D, texture);

   gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_S, gl.REPEAT);
   gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_T, gl.REPEAT);
   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, 1, 1, 0, gl.RGBA, gl.UNSIGNED_BYTE,
       new Uint8Array([255, 255, 255, 255]));

   let img = new Image();
   img.crossOrigin = "Anonymous";
   img.src = 'https://upload.wikimedia.org/wikipedia/commons/4/41/Brickwall_texture.jpg';
   img.addEventListener('load', function() {
       gl.bindTexture(gl.TEXTURE_2D, texture);
       gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE, img);
       draw();
   });
}
```

Vertex Shader

```
const vertexShaderSource = `
#define GLSLIFY 1


attribute vec3 vertex;

attribute vec3 normal;

uniform mat4 normalMatrix;

uniform mat4 ModelViewProjectionMatrix;


uniform float shininess;

uniform vec3 ambientColor;

uniform vec3 diffuseColor;

uniform vec3 specularColor;


uniform vec3 lightPosition;
```

```
varying vec4 color;


attribute vec4 iTexposition;

attribute vec2 iTexCoord;

uniform mat4 u_modelViewProjectionMatrix;

uniform float u_scale;

uniform float u_offset;

varying vec2 v_texCoord;


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;

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

   gl_Position = u_modelViewProjectionMatrix * vec4(vertex, 1.0);

   v_texCoord = (iTexCoord * u_scale) + u_offset;

}`;
```

Fragment Shader

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

#define GLSLIFY 1

varying vec4 color;
varying vec2 v_texCoord;
uniform sampler2D tmu;

void main() {
   gl_FragColor = texture2D(tmu,v_texCoord);
}`;
```