Міністерство Освіти й науки України

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Кафедра автоматизації проектування енергетичних процесів і систем

Звіт

«Візуалізація графічної та геометричної інформації»

Розрахунково-графічна робота

Варіант №2

Виконав:

студент 1-го курсу

групи ТР-22мп

НН ІАТЕ

Бачинський В.І.

Перевірив:

Демчишин А.А.

Київ-2022

Зміст

[**Опис завдання** 3](#_Toc124525091)

[**Опис теорії** 4](#_Toc124525092)

[**Деталі впровадження** 5](#_Toc124525093)

[**Інструкція користувача** 7](#_Toc124525094)

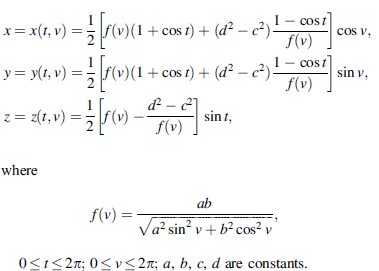
[**Вихідний код** 8](#_Toc124525095)

[**JS** 8](#_Toc124525096)

[**Shader** 9](#_Toc124525097)

**Опис завдання**

Необхідно нанести текстуру на поверхню (Virich Cyclic Surface), з такою формулою:



Для текстури необхідно реалізувати масштабування текстури (координати текстури) та масштабування навколо визначеної користувачем точки.

Повинна бути можливість переміщати точку вздовж простору поверхні за допомогою кнопок WASD на клавіатурі.

Потрібно підготувати звіт, з наступним змістом:

- титульна сторінка;

- розділ з описом завдання;

- розділ з описом теорії;

- розділ з описом деталей впровадження;

- розділ інструкції користувача зі скріншотами;

- зразок вихідного коду.

**Опис теорії**

WebGL (Web Graphics Library) — це API JavaScript, який дозволяє розробникам створювати інтерактивну 3D-графіку та візуальні ефекти у веб-браузерах. Він заснований на специфікації OpenGL ES 2.0, яка є широко використовуваним стандартом для створення 3D-графіки на мобільних пристроях і консолях.

WebGL підтримується більшістю сучасних веб-браузерів, включаючи Chrome, Firefox, Safari та Edge. Його можна використовувати для створення різноманітного інтерактивного 3D-контенту, включаючи ігри, візуалізації та симуляції.

Для використання WebGL розробники зазвичай використовують бібліотеку JavaScript, таку як Three.js або Babylon.js, яка забезпечує абстракцію WebGL API вищого рівня, що полегшує створення 3D-графіки без необхідності писати складний низькорівневий код.

WebGL також часто використовується в поєднанні з іншими веб-технологіями, такими як HTML, CSS і WebSockets, для створення більш складних інтерактивних програм.

За допомогою бібліотеки WebGL можна досить легко та зручно взаємодіяти з відображення текстур на поверхні будь якої складності. Для цього потрібно налаштувати вершинний шейдер так, щоб він використовував текстуру, а не кольори. Також у вершиному шейдері, якщо потрібно, можна змінювати координати текстури, масштабувати переносити та перетворювати їх за допомогою різних матриць. Після рефакторингу вершинного шейдера потрібно завантажити текстуру використовуючи JavaScript та зв'язати її з буфером координат текстури.

**Деталі впровадження**

Так як текстурування відбувається в фрагментному шейдері, то було додано змінну, для передачі в фрагментний шейдер, з ідентифікатором varying та атрибутом типу vec2, щоб отримувати координати текстури. В файлі main.js було створено функцію яка створює WebGl об'єкт текстури та об'єкт картинки, після чого картинка завантажуватися з мого репозиторію та прив’язується до об'єкту текстури. Потім було створено буфер для координат текстури та реалізована його прив'язка до його атрибуту. Після цього було створено функцію для переміщення точки заданої користувачем у вертексному шейдері:

mat4 translation(float tx, float ty, float tz) {

mat4 dst;

dst[0][0] = 1.0;

dst[0][ 1] = 0.0;

dst[0][ 2] = 0.0;

dst[0][ 3] = 0.0;

dst[1][ 0] = 0.0;

dst[1][ 1] = 1.0;

dst[1][ 2] = 0.0;

dst[1][ 3] = 0.0;

dst[2][ 0] = 0.0;

dst[2][ 1] = 0.0;

dst[2][ 2] = 1.0;

dst[2][ 3] = 0.0;

dst[3][ 0] = tx;

dst[3][ 1] = ty;

dst[3][ 2] = tz;

dst[3][ 3] = 1.0;

return dst;

А також функцію для обертання текстури навколо точки що задає користувач:

mat4 rotation(float angleInRadians) {

mat4 dst;

float c = cos(angleInRadians);

float s = sin(angleInRadians);

dst[0][0] = c;

dst[0][ 1] = s;

dst[0][ 2] = 0.0;

dst[0][ 3] = 0.0;

dst[1][ 0] = -s;

dst[1][ 1] = c;

dst[1][ 2] = 0.0;

dst[1][ 3] = 0.0;

dst[2][ 0] = 0.0;

dst[2][ 1] = 0.0;

dst[2][ 2] = 1.0;

dst[2][ 3] = 0.0;

dst[3][ 0] = 0.0;

dst[3][ 1] = 0.0;

dst[3][ 2] = 0.0;

dst[3][ 3] = 1.0;

return dst;

}

В результаті ми отримали поверхню (Virich Cyclic Surface), з точкою на ній:

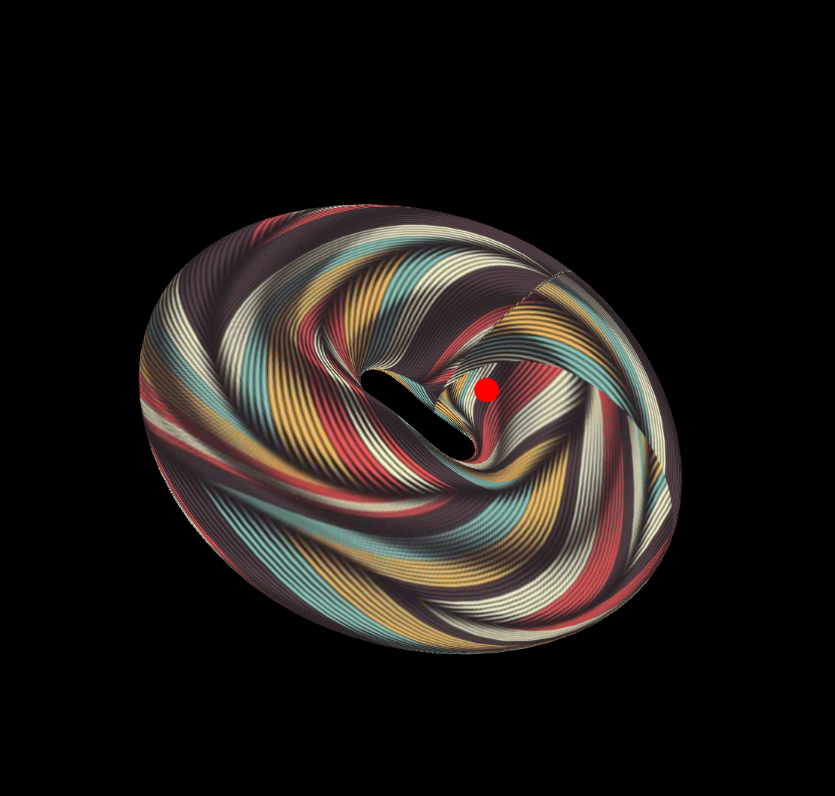


Рис 1 Virich Cyclic Surface with point and texture

**Інструкція користувача**

При затисканні лівої кнопки миші, користувач може здійснювати переорієнтацію фігури в просторі. Одночасно з цим буде відбуватись трансформація текстури. Трансформація текстури також відбувається якщо користувач просто буде водити мишею в горизонтальному напрямку. Зміну положення фігури зображено на рисках 2 та 3.

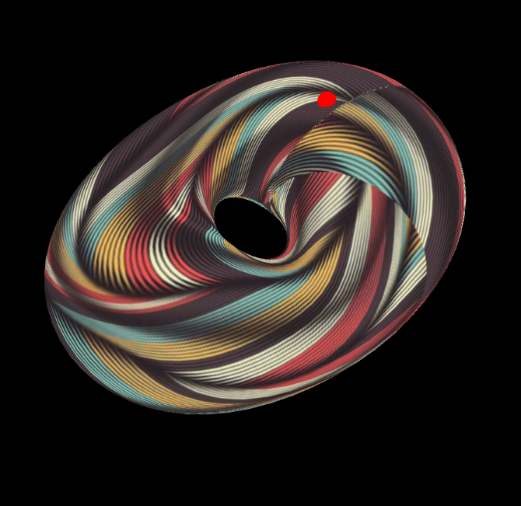
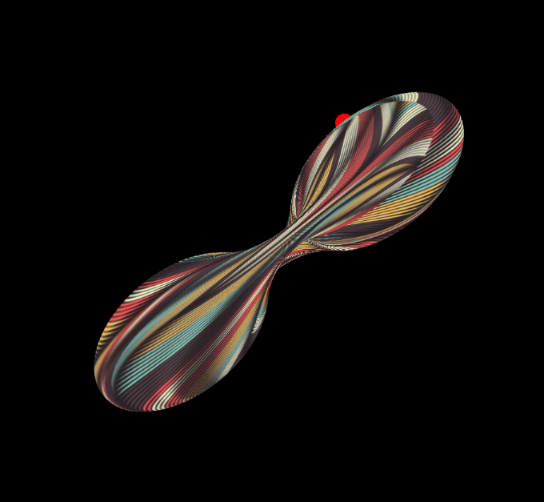
 

Рис 2 Положення 1 Рис 3 Положення 2

За допомогою введення з клавіатури клавішами W та S користувач може переміщувати точку відповідно до параметру v. За допомогою клавіш A та D здійснюється переміщення за параметром u. Відповідні переміщення точки можна побачити на рисунку 4 та 5.

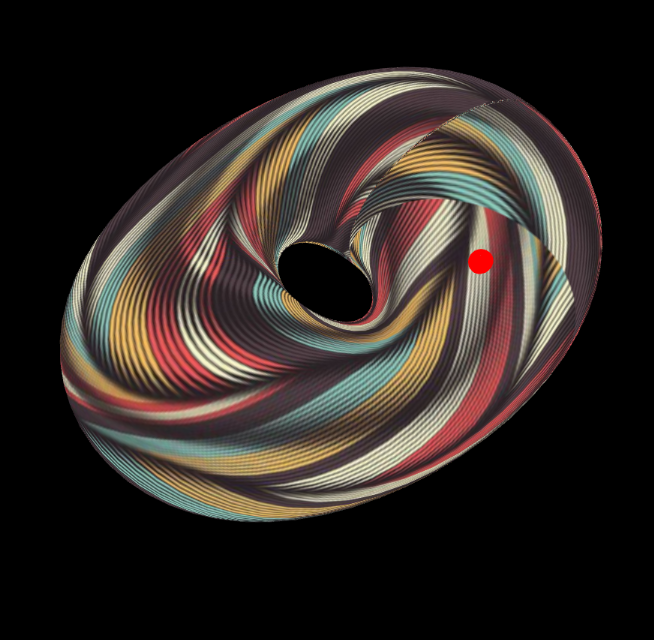
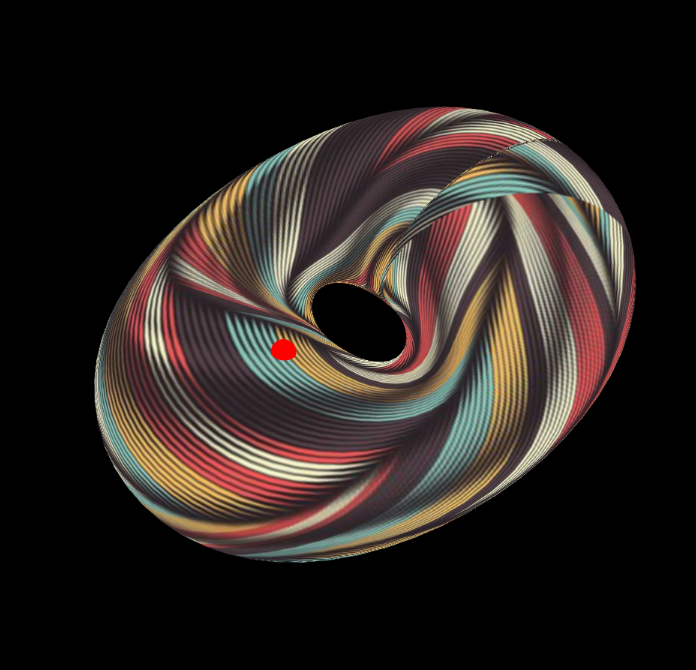
 

Рис 4 Положення точки №1 Рис 5 Положення точки №2

**Вихідний код**

## **JS**

function draw() {

  gl.clearColor(0, 0, 0, 1);

  gl.clear(gl.COLOR\_BUFFER\_BIT | gl.DEPTH\_BUFFER\_BIT);

  /\* Set the values of the projection transformation \*/

  // let projection = m4.perspective(Math.PI / 8, 1, 8, 12);

  let projection = m4.perspective(Math.PI / 8, 1, 8, 12);

  /\* Get the view matrix from the SimpleRotator object.\*/

  let modelView = spaceball.getViewMatrix();

  let rotateToPointZero = m4.axisRotation([0.707, 0.707, 0], 0.7);

  let translateToPointZero = m4.translation(0, 0, -10);

  let matAccum0 = m4.multiply(rotateToPointZero, modelView);

  let matAccum1 = m4.multiply(translateToPointZero, matAccum0);

  /\* Multiply the projection matrix times the modelview matrix to give the

       combined transformation matrix, and send that to the shader program. \*/

  let modelViewProjection = m4.multiply(projection, matAccum1);

  gl.uniformMatrix4fv(shProgram.iModelViewProjectionMatrix, false, modelViewProjection);

  //passing variables to the shader

  gl.uniform1i(shProgram.iTMU, 0);

  gl.enable(gl.TEXTURE\_2D);

  gl.uniform2fv(shProgram.iUserPoint, [userPointCoord.x, userPointCoord.y]);

  gl.uniform1f(shProgram.irotAngle, userRotAngle);

  gl.uniform2fv(shProgram.iUserPoint, [userPointCoord.x, userPointCoord.y]); //giving coordinates of user point

  gl.uniform1f(shProgram.irotAngle, userRotAngle);

  surface.Draw();

  let translation = virich(

    map(userPointCoord.x, 0, 1, 0, Math.PI \* 2),

    map(userPointCoord.y, 0, 1, 0, Math.PI \* 2),

  );

  gl.uniform3fv(shProgram.iUP, [translation.x, translation.y, translation.z]);

  gl.uniform1f(shProgram.irotAngle, 1100);

  point.DrawPoint();

}

function LoadTexture() {

  let 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);

  const image = new Image();

  image.crossOrigin = 'anonymus';

  image.src = 'https://raw.githubusercontent.com/vlad-beep/WebGL\_labs/PGW/texture.png';

  image.onload = () => {

    gl.bindTexture(gl.TEXTURE\_2D, texture);

    gl.texImage2D(gl.TEXTURE\_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED\_BYTE, image);

    console.log('imageLoaded');

    draw();

  };

}

## **Shader**

/// Vertex shader

const vertexShaderSource = `

attribute vec3 vertex;

attribute vec2 texture;

uniform mat4 ModelViewProjectionMatrix;

varying vec2 v\_texcoord;

uniform vec3 translateUP;

uniform vec2 userPoint;

uniform float rotA;

//function to create a displacement matrix

mat4 translation(float tx, float ty, float tz) {

mat4 dst;

dst[0][0] = 1.0;

dst[0][ 1] = 0.0;

dst[0][ 2] = 0.0;

dst[0][ 3] = 0.0;

dst[1][ 0] = 0.0;

dst[1][ 1] = 1.0;

dst[1][ 2] = 0.0;

dst[1][ 3] = 0.0;

dst[2][ 0] = 0.0;

dst[2][ 1] = 0.0;

dst[2][ 2] = 1.0;

dst[2][ 3] = 0.0;

dst[3][ 0] = tx;

dst[3][ 1] = ty;

dst[3][ 2] = tz;

dst[3][ 3] = 1.0;

return dst;

}

//function to create a rotation matrix

mat4 rotation(float angleInRadians) {

mat4 dst;

float c = cos(angleInRadians);

float s = sin(angleInRadians);

dst[0][0] = c;

dst[0][ 1] = s;

dst[0][ 2] = 0.0;

dst[0][ 3] = 0.0;

dst[1][ 0] = -s;

dst[1][ 1] = c;

dst[1][ 2] = 0.0;

dst[1][ 3] = 0.0;

dst[2][ 0] = 0.0;

dst[2][ 1] = 0.0;

dst[2][ 2] = 1.0;

dst[2][ 3] = 0.0;

dst[3][ 0] = 0.0;

dst[3][ 1] = 0.0;

dst[3][ 2] = 0.0;

dst[3][ 3] = 1.0;

return dst;

}

void main() {

mat4 rotMatrix = rotation(rotA);

mat4 transMatrix1 = translation(userPoint.x,userPoint.y,0.0);

mat4 transMatrix2 = translation(-userPoint.x,-userPoint.y,0.0);

vec4 texture1 = vec4(texture,0.0,0.0)\*transMatrix1;

//обертаємо текстуру

vec4 textureRotated = texture1\*rotMatrix;

vec4 texture2 = textureRotated\*transMatrix2;

v\_texcoord = vec2(texture2.x,texture2.y);

vec4 vertPos4 = ModelViewProjectionMatrix \* vec4(vertex, 1.0);

gl\_Position = vertPos4;

if(rotA>1000.0){

vec4 translatePoint = translation(translateUP.x,translateUP.y,translateUP.z)\*vec4(vertex,1.0);

gl\_Position = ModelViewProjectionMatrix\*translatePoint;

}

}`;

// Fragment shader

const fragmentShaderSource = `

#ifdef GL\_FRAGMENT\_PRECISION\_HIGH

precision highp float;

#else

precision mediump float;

#endif

varying vec2 v\_texcoord;

uniform sampler2D tmu;

uniform float rotA;

void main() {

//apply texture

vec4 texColor = texture2D(tmu, v\_texcoord);

gl\_FragColor = texColor;

if(rotA>1000.0){

gl\_FragColor = vec4(0.,0.,0.,0.);

}

}`;