НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Кафедра цифрових технологій в енергетиці

Розрахунково-графічна робота

З дисципліни: «Візуалізація графічної та геометричної інформації»

Варіант 2

**Виконав:**

студент 5 курсу

групи ТР-21мп, ТЕФ

Бойко Єгор Вікторович

**Перевірив:**

Демчишин А. А.

Київ – 2023

**Завдання**

* Нанести текстуру на поверхню з лабораторної роботи №2.
* Реалізувати обертання текстури навколо визначеної користувачем точки.
* Додати можливість переміщати точку вздовж простору поверхні (u, v) за допомогою клавіатури. Наприклад клавіші A і D переміщують точку вздовж параметра u, а клавіші W і S переміщують точку вздовж параметра v.

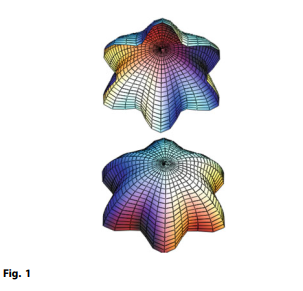
**Теоретична інформація**

Поверхня “Corrugated Sphere” має кругову синусоїду:



*Рис.1 Рівняння синусоїди*

в основі, де n – кількість вершин синусоїда на круговому плані, a — максимальна амплітуда звиви в основі поверхні (на екваторі), R – радіус сфери на екваторі, відносно якої кругова синусоїда будується, φ – кут, відрахований від осі Ox в напрямку осі Oy.



*Рис.2 «Corrugated Sphere»*

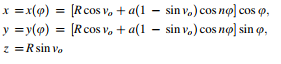
Параметрична форма визначення “Corrugated Sphere”:



*Рис.3 Параметрична форма x,y*

де v – кут площини xOy з віссю Oz;

; ;. У поперечних перерізах “Corrugated Sphere” площинами z = const, тобто при v = =const, кругові синусоїди



*Рис.4 Параметрична форма x,y,z*

мають місце.

Текстура - це спосіб надання поверхні 3D деталі — полігону: кольору, фактури, блиску, матовості та інших фізичних властивостей.

Поняття “текстура” є важливим елементом 3D-моделювання, оскільки дозволяє відтворити також малі об'єкти поверхні, створення яких полігонами виявилося б надмірно ресурсомістким. Наприклад, шрами на шкірі, складки на одязі, дрібні камені, предмети на поверхні стін і ґрунту та багато іншого. Отже, текстура використовується для заповнення поверхонь об'єктів і як шар для додання певного ефекту або зміни геометрії всьому зображенню або його частини.

Карта текстури застосовується для утворення певного параметру візуального відображення на поверхні заданої форми. Цей процес нагадує застосування візерунчастого паперу на звичайній білій коробці. Кожній вершині в 3D моделі присвоюється координати текстури (яка у разі 2D відома, як UV координата). Місця відбору проб зображення згодом інтерполюється по поверхні моделі з отриманням візуального результату.

**Деталі реалізації**

Для реалізації накладання текстури я використав мову програмування JavaScript та бібліотеку WebGL. Процес складається з таких етапів:

1. Спочатку треба додати код для завантаження текстур та налаштувати параметри текстурування після завантаження самої текстури, це робить функція loadTexture. У роботі була використана лише одна текстура, яка нанесена на поверхню. Нанесення текстури можна розбити на такі підпункти:
   1. Створюємо елемент Image, який буде виступати джерелом текстури та задаємо атрибут src текстури для даного елемента.
   2. Визначення методу onload, який міститиме логіку, яка спрацьовує при завантаженні зображення в елемент Image.
   3. Всередині методу onload визначаємо та створюємо об'єкт текстури за допомогою методу gl.createTexture().
   4. Потім прив'язуємо текстуру за допомогою методу gl.bindTexture().
   5. Налаштовуємо параметри текстури за допомогою методу gl.texParameteri().
   6. Наприкінці завантажуємо текстури у GPU за допомогою методу gl.texImage2D().

function loadTexture() {

  const image = new Image()

  image.crossOrigin = 'anonymous'

  image.src =

    'https://www.the3rdsequence.com/texturedb/download/258/texture/jpg/2048/yellow+bananas-2048x2048.jpg'

  image.onload = () => {

    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()

  }

1. Оновлюємо шейдер, додаємо в нього інформацію про текстуру та додаємо функцію для обертання текстури навколо точки.

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)

);

1. Для моделі фігури створюємо буфер для текстурних координат та додаємо вказівник на читання з буфера вершин.

gl.bindBuffer(gl.ARRAY\_BUFFER, this.iTextureBuffer)

gl.vertexAttribPointer(shProgram.iTextureCoords, 2, gl.FLOAT, false, 0, 0)

gl.enableVertexAttribArray(shProgram.iTextureCoords)

1. Додаємо управління точкою обертання натисненням клавіш “w”, “a”, “s”, “d”:

window.addEventListener('keydown', (event) => {

  switch (event.key) {

    case 'ArrowLeft':

      onArrowLeft()

      break

    case 'ArrowRight':

      onArrowRight()

      break

    case 'd':

      point.x = point.x + 0.1

      draw()

      break

    case 'a':

      point.x = point.x - 0.1

    case 'w':

      point.y = point.y + 0.1

      draw()

      break

    case 's':

      point.y = point.y - 0.1

      draw()

      break

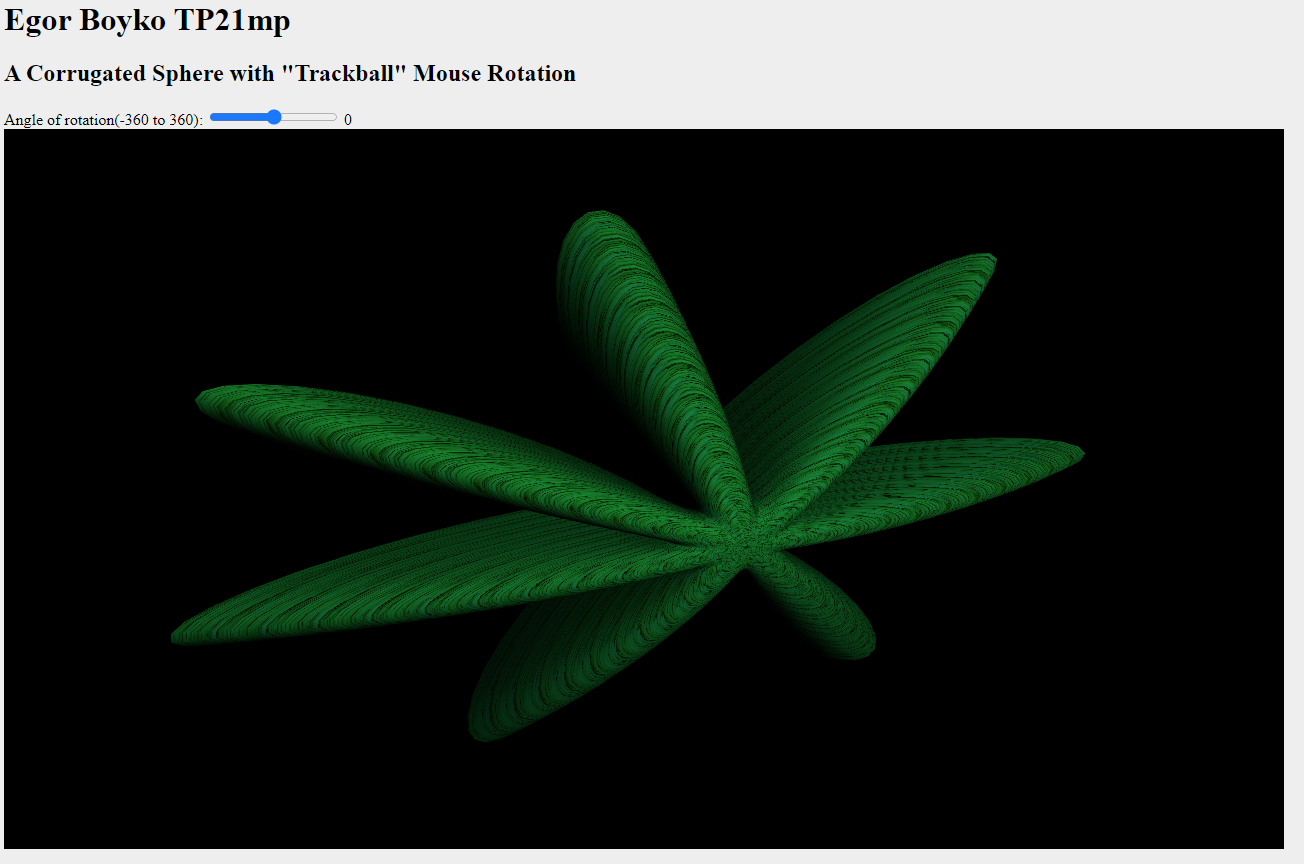
    default:

      break

  }

**Інструкція користувача**

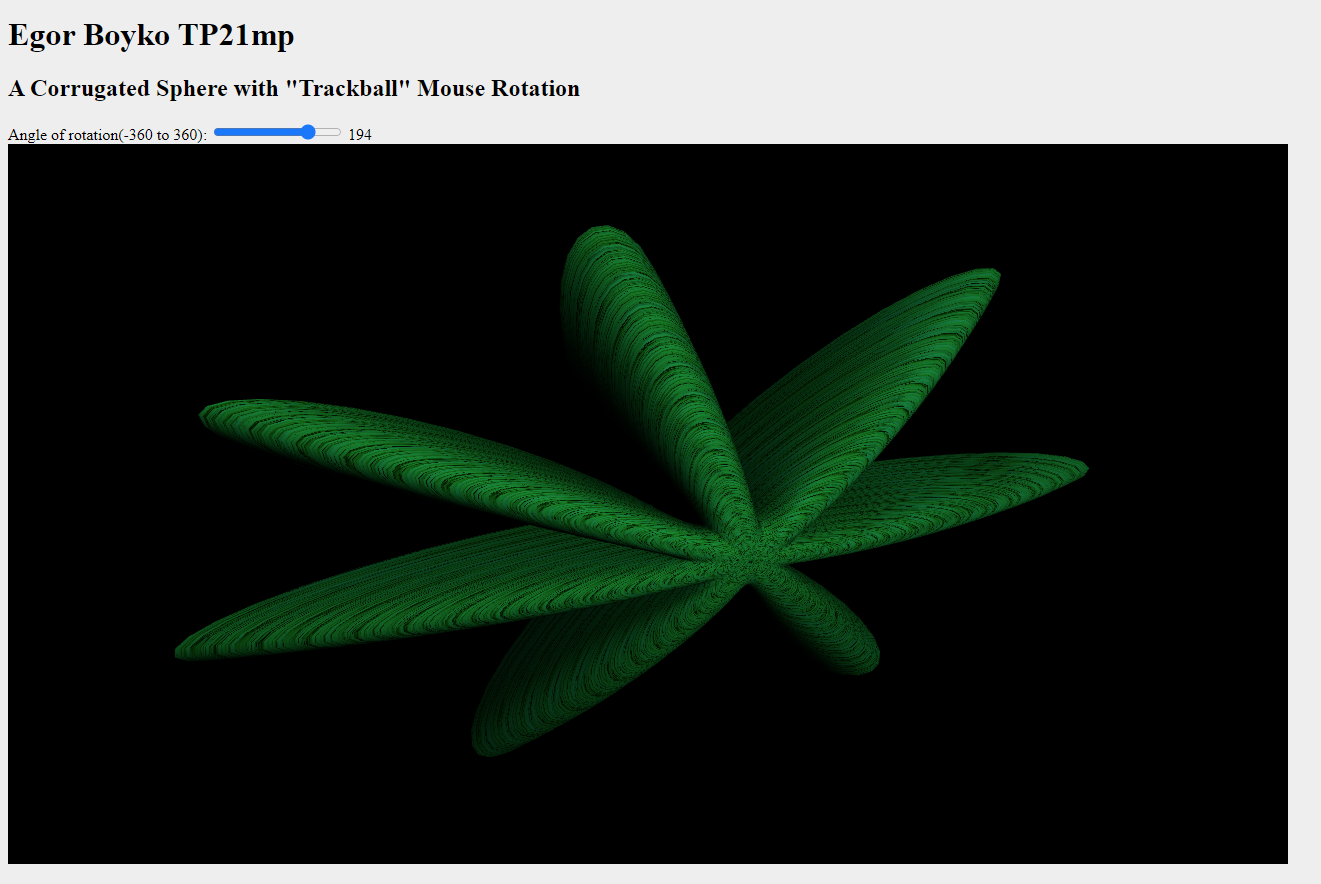
Нижче наведені зображення для демонстрації функціоналу та прикладу користування програмою.



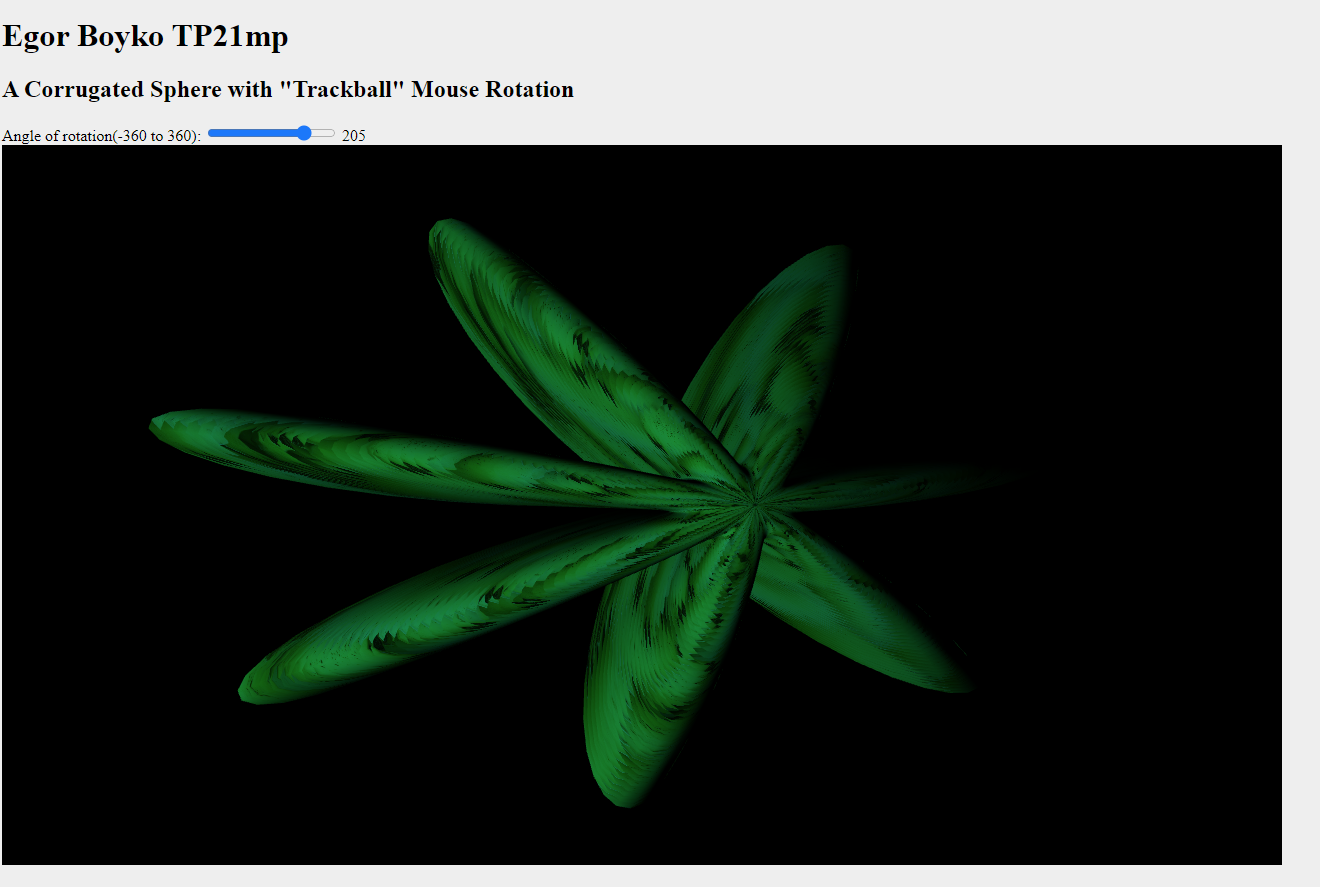
*Рис.5 Поверхня із стандартним значенням кута оберту текстури*

За допомогою клавіш “w”, “a”, “s”, “d” користувач може змінювати позицію точки навколо якої буде здійснюватися обертання. Саме ж обертання здійснюється за допомогою зміни повзунка “Angle of rotation” над зображенням. Поточне значення кута відображається поруч з повзунком.

Крім того, натискаючи клавішу стрілки вправо чи вліво, користувач може змінювати освітлення поверхні.



*Рис.6 Змінений параметр кута оберту текстури*



*Рис.7 Зміна точки оберту, освітлення та параметра кута оберту текстури*

**Зразок вихідного коду**

let gl // The webgl context.

let surface // A surface model

let shProgram // A shader program

let spaceball // A SimpleRotator object that lets the user rotate the view by mouse.

let t = 0.0

let point = { x: 0, y: 0 }

function deg2rad(angle) {

  return (angle \* Math.PI) / 180

}

// Constructor

function Model(name) {

  this.name = name

  this.iVertexBuffer = gl.createBuffer()

  this.iNormalBuffer = gl.createBuffer()

  this.iTextureBuffer = gl.createBuffer()

  this.count = 0

  this.BufferData = function ({ vertexList, normalList, textureList }) {

    gl.bindBuffer(gl.ARRAY\_BUFFER, this.iVertexBuffer)

    gl.bufferData(gl.ARRAY\_BUFFER, new Float32Array(vertexList), gl.STREAM\_DRAW)

    gl.bindBuffer(gl.ARRAY\_BUFFER, this.iNormalBuffer)

    gl.bufferData(gl.ARRAY\_BUFFER, new Float32Array(normalList), gl.STREAM\_DRAW)

    gl.bindBuffer(gl.ARRAY\_BUFFER, this.iTextureBuffer)

    gl.bufferData(

      gl.ARRAY\_BUFFER,

      new Float32Array(textureList),

      gl.STREAM\_DRAW

    )

    this.count = vertexList.length / 3

  }

  this.Draw = function () {

    gl.bindBuffer(gl.ARRAY\_BUFFER, this.iVertexBuffer)

    gl.vertexAttribPointer(shProgram.iAttribVertex, 3, gl.FLOAT, false, 0, 0)

    gl.enableVertexAttribArray(shProgram.iAttribVertex)

    gl.bindBuffer(gl.ARRAY\_BUFFER, this.iNormalBuffer)

    gl.vertexAttribPointer(shProgram.iNormal, 3, gl.FLOAT, true, 0, 0)

    gl.enableVertexAttribArray(shProgram.iNormal)

    gl.bindBuffer(gl.ARRAY\_BUFFER, this.iTextureBuffer)

    gl.vertexAttribPointer(shProgram.iTextureCoords, 2, gl.FLOAT, false, 0, 0)

    gl.enableVertexAttribArray(shProgram.iTextureCoords)

    gl.drawArrays(gl.TRIANGLE\_STRIP, 0, this.count)

  }

}

// Constructor

function ShaderProgram(name, program) {

  this.name = name

  this.prog = program

  // Location of the attribute variable in the shader program.

  this.iAttribVertex = -1

  // Location of the uniform specifying a color for the primitive.

  this.iColor = -1

  // Location of the uniform matrix representing the combined transformation.

  this.iModelViewProjectionMatrix = -1

  this.iNormal = -1

  this.iNormalMatrix = -1

  this.iLightPosition = -1

  this.iLightVec = -1

  this.iAmbientColor = -1

  this.iDiffuseColor = -1

  this.iSpecularColor = -1

  this.iShininess = -1

  this.iTextureCoords = -1

  this.iTextureU = -1

  this.iTextureAngle = -1

  this.iTexturePoint = -1

  this.Use = function () {

    gl.useProgram(this.prog)

  }

}

/\* Draws a colored cube, along with a set of coordinate axes.

 \* (Note that the use of the above drawPrimitive function is not an efficient

 \* way to draw with WebGL.  Here, the geometry is so simple that it doesn't matter.)

 \*/

function draw() {

  const angle = document.getElementById('rotAngle').value

  gl.clearColor(0, 0, 0, 1)

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

  gl.enable(gl.CULL\_FACE)

  gl.enable(gl.DEPTH\_TEST)

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

  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)

  let modelViewInverse = m4.inverse(matAccum1, new Float32Array(16))

  let normalMatrix = m4.transpose(modelViewInverse, new Float32Array(16))

  /\* 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

  )

  gl.uniformMatrix4fv(shProgram.iNormalMatrix, false, normalMatrix)

  gl.uniform3fv(shProgram.iLightPosition, lightCoordinates())

  gl.uniform3fv(shProgram.iLightDirection, [1, 0, 0])

  gl.uniform3fv(shProgram.iLightVec, new Float32Array(3))

  gl.uniform1f(shProgram.iShininess, 1.5)

  gl.uniform3fv(shProgram.iAmbientColor, [0.0, 0.0, 0.0])

  gl.uniform3fv(shProgram.iDiffuseColor, [0.034, 0.537, 0.85])

  gl.uniform3fv(shProgram.iSpecularColor, [0.0, 0.0, 0.0])

  /\* Draw the six faces of a cube, with different colors. \*/

  gl.uniform4fv(shProgram.iColor, [1, 1, 0, 1])

  gl.uniform1f(shProgram.iTextureAngle, deg2rad(+angle))

  gl.uniform2fv(shProgram.iTexturePoint, [

    calcX(point.x, point.y, 3, 2, 7),

    calcY(point.x, point.y, 3, 2, 7),

  ])

  gl.uniform1i(shProgram.iTextureU, 0)

  surface.Draw()

}