Міністерство освіти й науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Кафедра автоматизації проектування енергетичних процесів і систем

# Звіт
з циклу лабораторних робіт з дисципліни
«Візуалізація графічної та геометричної інформації»

Розрахунково-графічна робота
Варіант-19

Виконала:
студентка 5-го курсу
групи ТР-23мп НН ІАТЕ
Савчук Анна
Перевірив:
Демчишин А.А.

Київ-2023

Опис завдання

Нанести текстуру на поверхню “Kiss Surface”, що задана наступною формулою:

$$\begin{cases} x = x(u,z) = z^2 * \sqrt{1-z} * \cos(u) \\ y = y(u,z) = z^2 * \sqrt{1-z} * \sin(u) \\ z = z \end{cases}$$

Формула - 1. “Kiss Surface”

Реалізувати можливість масштабування текстури (координати текстури) масштабування/обертання навколо визначеної користувачем точки.

Повинна бути 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.

Опис теорії

Відображення текстури — це техніка визначення унікального кольору для кожного фрагмента, який утворює трикутник. Кольори походять із відображення. У математиці відображення — це функція, яка перетворює набір вхідних даних у вихідне значення.

GPU містить блоки текстур для підтримки відображення текстур. Блок текстури виконує обробку для відображення текстури. Об'єкт текстури зберігає дані, необхідні для відображення текстури. Разом блок текстури та об’єкт текстури можуть виконувати відображення текстури в шейдерній програмі . Може бути створено скільки завгодно об’єктів текстури, але кількість текстурних одиниць у графічному процесорі визначає, скільки текстурних карт може бути використано одночасно в програмі шейдера.

При побудові моделі:

- Виберіть відповідне зображення для накладання текстури.
- Призначте відповідну координату текстури (s,t) кожній вершині трикутника.

Попередня обробка JavaScript для візуалізації полотна:

- Завантажте зображення текстурної карти з сервера.
- Створіть і заповніть об’єкт текстури GPU зображенням.
- Встановіть параметри, які керують використанням зображення карти текстури.
- Отримайте розташування uniform Sample2Dзмінної з програми шейдера .

Налаштування JavaScript кожного разу, коли модель відображається за допомогою карти текстури:

- Прив’язати об’єкт текстури до блоку текстури
- Прив’яжіть блок текстури до uniformзмінної шейдера.

Шейдерна програма

- У вершинному шейдері створюється varyingзмінна, яка буде інтерполювати координати текстури по поверхні трикутника.
- У фрагментному шейдері використовуйте координати текстури для пошуку кольору із зображення текстури.

Виберіть відповідне зображення.

Будь-яке цифрове зображення можна використовувати як карту текстури, але якщо ширина та висота зображення не є ступенем 2, існують обмеження щодо використання зображення. Найкраще, щоб розміри зображення дорівнювали степеню 2, що робить відображення текстур більш ефективним і знімає будь-які обмеження на його використання.

Створення текстурних об’єктів у GPU.

Коли ми візуалізуємо модель, ми хочемо, щоб дані моделі зберігалися в пам’яті графічного процесора, щоб вони були безпосередньо доступні для шейдерної програми . Щоб використовувати зображення як таблицю пошуку значень кольорів, нам потрібно, щоб зображення також було доступним із пам’яті GPU. Оскільки відображення текстури є принципово іншою операцією порівняно з gl.drawArrays(), пам'ять, у якій зберігається зображення відображення текстури, називається об'єктом текстури замість об'єкта буфера . Об’єкт текстури зберігає зображення та всі пов’язані змінні стану, необхідні для створення відображення текстури. Ви можете створити стільки текстурних об’єктів , скільки має пам’ять графічний процесор.

Є три основні кроки для створення об’єкта текстури:

- Створіть новий об'єкт текстури
- Встановіть параметри, які керують використанням об’єкта текстури .
- Скопіюйте зображення в об’єкт текстури

Налаштування JavaScript для відображення текстур.

Ваша шейдерна програма матиме uniform Sampler2Dзмінну у своєму фрагментному шейдері . Ця змінна повинна вказувати, який блок текстури використовувати для відображення текстури. Але блоку текстури потрібні дані з об'єкта текстури . Тож ми прив’язуємо об’єкт текстури до блоку текстури, а потім встановлюємо uniform Sampler2Dзмінну для блоку текстури.

Шейдерні програми , що використовують відображення текстур.

Програми - шейдери, які виконують відображення текстури, є легкою частиною всього цього процесу. Вершиний шейдер просто копіює координати текстури вершини в varying змінну, щоб їх можна було інтерполювати на поверхні трикутника. Фрагментний шейдер використовує координати текстури для фрагмента для пошуку кольору в зображенні карти текстури. Це звичайна операція, вбудована у функціональність GLSL. Ви просто викликаєте texture2Dфункцію та вказуєте одиницю текстури, яку потрібно використовувати, і координати текстури (якими є vec2, два значення з плаваючою комою).

## Деталі реалізації

В функції loadTexture() створюємо текстуру, налаштовуємо фільтр збільшення текстури та мінімізації. Вказуємо адресу обраного зображення. Викликаємо метод при ініціалізції, в функції initGL()

```
function loadTexture()
{
    var 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)

    var image = new Image();
    image.crossOrigin = 'anonymous'
    image.src = "https://lh3.googleusercontent.com/9wkYryGX1XUiTq81UhqcCY-PDDcbNtGcV7n4iajZr9fgzAGXUPOJ4Xxr4a
    image.addEventListener( type: 'load',  listener: () => {
        gl.bindTexture(gl.TEXTURE_2D, texture)
        gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE, image)
        draw();
    });
}
```

В функції createTextureCoordinates() розраховуємо координати текстури. Поверхня залежить від декількох змінних: кут, який змінюється від 0 до 2П та межі, що змінюються від -1 до 1.

```
function createTextureCoordinates(){
    let resultCoordinates = []
    let stepU = 360 / (N - 1)
    let stepV = 2 / (N - 1)
    for (let u = 0; u < 360; u += stepU) {
        for (let v = -1; v <= 1; v += stepV) {
            resultCoordinates.push(deg2rad(u) / (2 * Math.PI), (v + 1) / 2)
            resultCoordinates.push(deg2rad( angle: u + stepU)  / (2 * Math.PI), (v + 1) / 2)
        }
    }
    return resultCoordinates
}
```

Vertex shader

```
// Vertex shader
const vertexShaderSource = `
attribute vec3 vertex;
attribute vec3 normal;
attribute vec2 textureCoord;
uniform mat4 ModelViewProjectionMatrix, normalMat;
varying vec3 normalInterp;
varying vec3 vertPos;
uniform vec2 scalePoint;
uniform float fScale;
varying vec2 texInter;
```

Створюємо нові змінні.
Далі в функції scale обчислюємо матрицю масштабування.

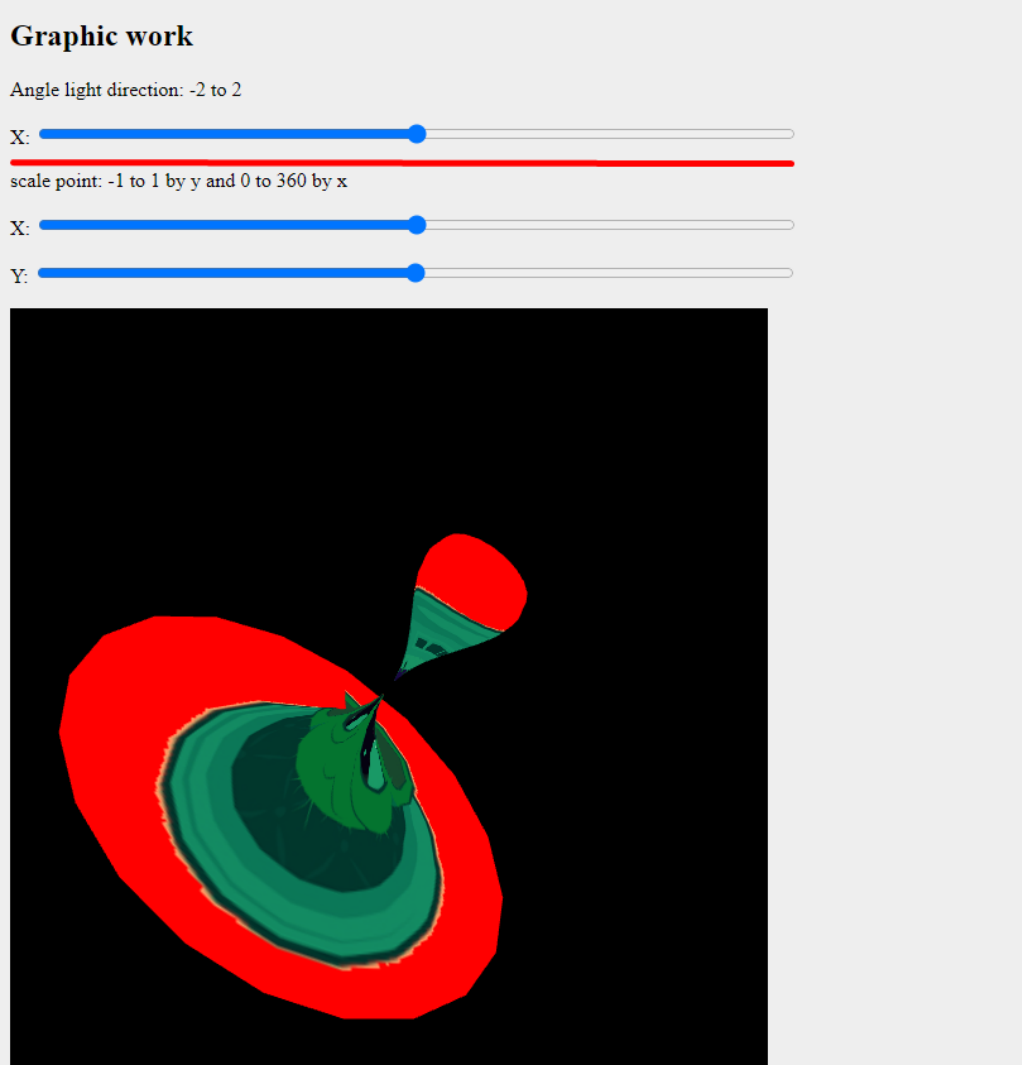
```
mat4 scale(float value) {
  mat4 scaleMatrix;

  for(int i = 0; i < 4;i++)  {
    for(int j = 0; j < 4;j++) {
      if(i != j) {
        scaleMatrix[i][j] = 0.0;
      } else {
        scaleMatrix[i][j] = value;
      }
    }
  }
  scaleMatrix[3][3] = 1.0;
  return scaleMatrix;
}
```

Fragment shader
У фрагментному шейдері викликаємо функцію texture2D для створення кольору фігури.

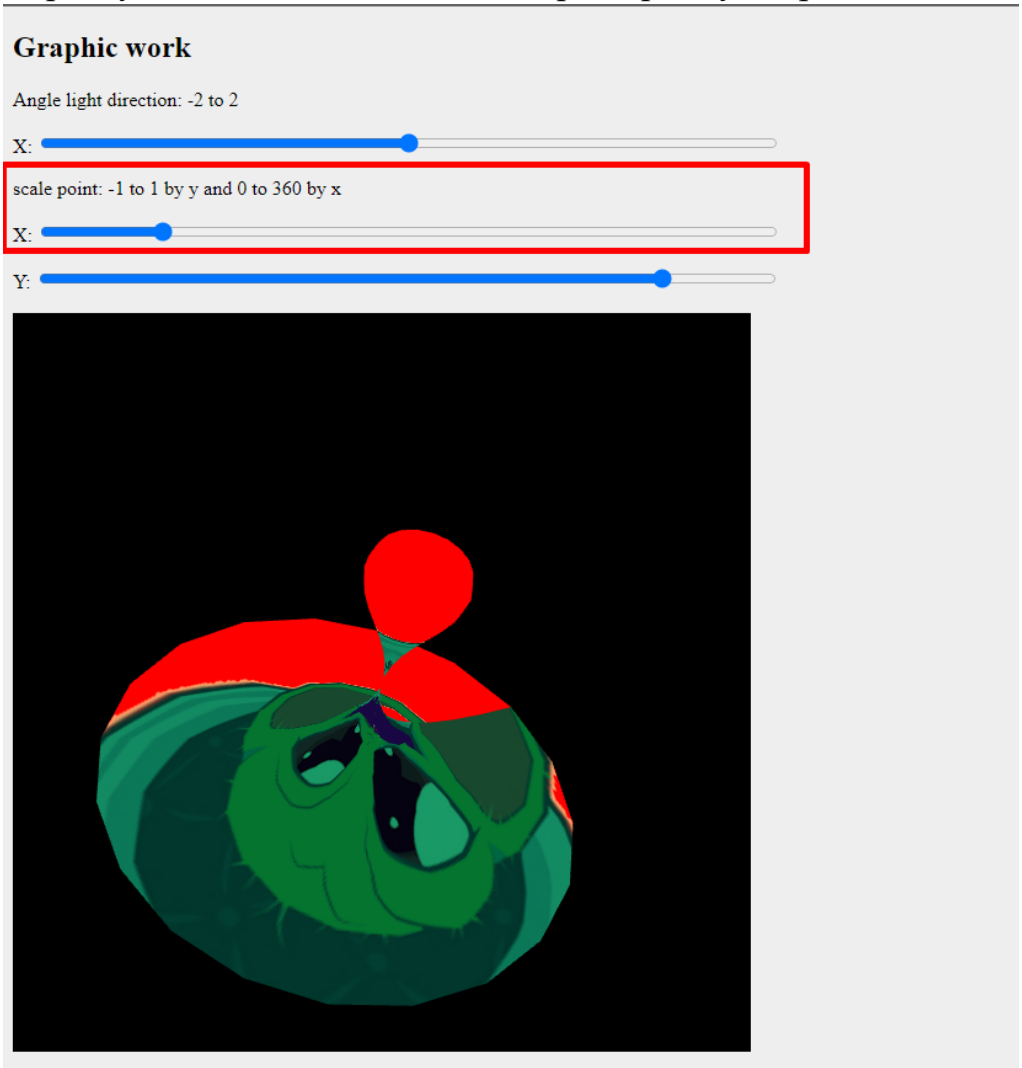
```
void main() {
  vec3 vNormal = normalize(normalInterp);
  vec3 light = normalize(lightPosition - vertPos);

  float dotProduct = max(dot(vNormal, light), 0.0);
  float specular = 0.0;

  if(dotProduct > 0.0) {
    vec3 reflect = reflect(-light, vNormal);
    vec3 n_vertPos = normalize(-vertPos);
    float specAngle = max(dot(reflect, n_vertPos), 0.0);
    specular = pow(specAngle, shininess);
  }

  vec4 lightColor = vec4(ambientCoefficient * ambientColor
                  + diffuseCoefficient * dotProduct * diffuseColor
                  + specularCoefficient * specular * specularColor, 1.0);
  vec4 texture = texture2D(sampler, texInter);
  gl_FragColor = texture * lightColor;
```

Інструкція користувача

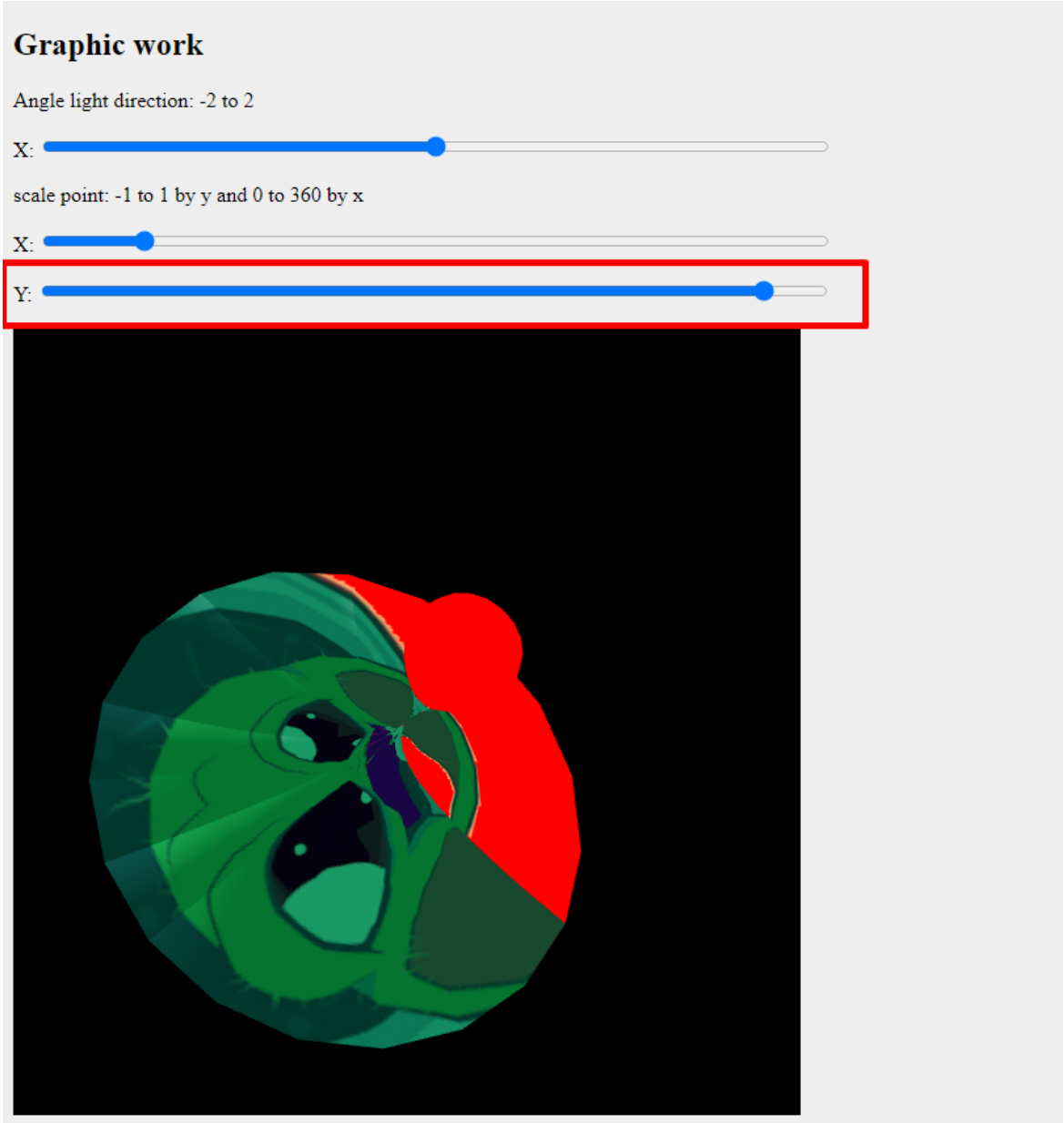
Користувач може змінювати вектор напрямку направленого світла як показано на скріншоті



Користувач може змінювати вектор напрямку зображення



Також користувач може змінювати масштаб зображення



Image

Приклад коду

Vertex Shader

```
// Vertex shader
const vertexShaderSource = `
attribute vec3 vertex;
attribute vec3 normal;
attribute vec2 textureCoord;
uniform mat4 ModelViewProjectionMatrix, normalMat;
varying vec3 normalInterp;
varying vec3 vertPos;
uniform vec2 scalePoint;
uniform float fScale;
varying vec2 texInter;

mat4 scale(float value)
{
  mat4 scaleMatrix;

  for(int i = 0; i < 4;i++) {
    for(int j = 0; j < 4;j++) {
      if(i != j) {
        scaleMatrix[i][j] = 0.0;
      }
      else {
        scaleMatrix[i][j] = value;
      }
    }
  }

  scaleMatrix[3][3] = 1.0;

  return scaleMatrix;
```

Fragment Shader

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

varying vec4 color;
precision mediump float;
varying vec3 normalInterp;  // Surface normal
varying vec3 vertPos;       // Vertex position
uniform float ambientCoefficient;   // Ambient reflection coefficient
uniform float diffuseCoefficient;   // Diffuse reflection coefficient
uniform float specularCoefficient;   // Specular reflection coefficient
uniform float shininess; // Shininess
uniform vec3 ambientColor;
uniform vec3 diffuseColor;
uniform vec3 specularColor;
uniform vec3 lightPosition;
uniform sampler2D sampler;
varying vec2 texInter;
```

```
void main() {
  vec3 vNormal = normalize(normalInterp);
  vec3 light = normalize(lightPosition - vertPos);

  float dotProduct = max(dot(vNormal, light), 0.0);
  float specular = 0.0;

  if(dotProduct > 0.0) {
    vec3 reflect = reflect(-light, vNormal);
    vec3 n_vertPos = normalize(-vertPos);
    float specAngle = max(dot(reflect, n_vertPos), 0.0);
    specular = pow(specAngle, shininess);
  }

  vec4 lightColor = vec4(ambientCoefficient * ambientColor
                    + diffuseCoefficient * dotProduct * diffuseColor
                    + specularCoefficient * specular * specularColor, 1.0);
  vec4 texture = texture2D(sampler, texInter);
  gl_FragColor = texture * lightColor;

}`;
```