Ministry of Education and Science of Ukraine
National Technical University of Ukraine
"Kyiv Polytechnic Institute named after Igor Sikorsky"
Educational and Scientific Institute of Atomic and Thermal Energy
Department of Digital Technologies in Energy

Calculation and graphical work
of the discipline: «Visualization of graphical and geometric information»

Topic**:**
«Operations on texture coordinates»

Prepared by:
student of the TP-43mp group
Humeniuk A.O.

Checked by:
Demchyshyn A. A.

Kyiv – 2024

1. Task

1. Reuse texture mapping from Control task.
2. Implement texture scaling (texture coordinates) scaling / rotation around user specified point- odd variants implement scaling, even variants implement rotation
3. 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.

2. Theoretical information

2.1 Shading

Shading in computer graphics refers to the process of calculating the color and intensity of light at a particular point on a surface. This involves simulating the interaction of light with objects to achieve realism or a desired artistic style.

Shading determines how light interacts with surfaces, contributing to their appearance based on factors like material properties, light sources, and viewing angles.

Surface normal, a vector perpendicular to a surface at a given point. These normals are essential for calculating how light reflects or scatters across the surface, directly influencing shading results. Another important factor is the light source, which provides the illumination necessary to compute surface brightness. Light sources come in various types, such as point lights that emit light in all directions from a single point, directional lights that simulate parallel rays from a distant source, spotlights that create focused beams, and ambient light that uniformly illuminates a scene without directionality.

The viewer's position also plays a significant role in shading, as it affects how reflections and highlights are perceived, contributing to the appearance of depth and material realism. Additionally, the material properties of an object determine how it interacts with light. Surfaces exhibit different types of reflections, including diffuse reflection, which scatters light evenly to produce a matte look; specular reflection, which creates shiny highlights; and ambient reflection, which ensures objects are illuminated even in shadowed areas. Together, these concepts form the foundation of shading and allow for the creation of visually compelling and realistic graphics.

Common shading models defined as Flat, Gourand and Phong:

1. Flat - Each polygon is shaded using a single normal and color.
2. Gouraud - colors are calculated at each vertex and interpolated across the surface.

3. Phong - interpolates normals across the surface and calculates color for each pixel.

2.2 Textures

In computer graphics, textures are images or patterns applied to the surface of 3D models or 2D graphics to give them more detail, realism, or artistic style.

A texture is a 2D bitmap image or procedural pattern that is mapped onto the surface of a 3D object. Textures can represent various surface properties, such as:

- Color(diffuse maps)
- Roughness(specular maps)
- Depth details (bump or normal maps)

Textures are mapped using UV coordinates, a 2D coordinate system where U and V represent the horizontal and vertical axes. The surface of the 3D model is "unwrapped" into a flat 2D plane for applying the texture.

Each vertex in a 3D model is assigned UV coordinates, which dictate how the texture is applied. The texture image is "sampled" at these coordinates to color or define other properties of the model's surface.

In WebGL, texturing is a fundamental technique for adding visual detail to 3D scenes rendered in web browsers. Textures in WebGL are handled using the WebGL API, which allows to load image files, create texture objects, and apply them to 3D models.

The process involves binding a texture to a texture unit, setting parameters such as wrapping mode and filtering, and then using shaders to map the texture onto a 3D surface.

WebGL supports common features like mipmapping, texture compression, and various filtering modes. Since WebGL operates within the constraints of a browser, efficient texture handling and optimization (e.g., using compressed formats and small file sizes) are critical to ensure smooth performance and compatibility across devices. Textures are often managed using GLSL shaders,

enabling developers to create effects like animated textures, dynamic reflections, and procedural patterns directly in the browser.

3. Implementation Details

To scale a texture around a specific point, the UV coordinates are adjusted in the fragment shader. This is done by first translating the coordinates such that the pivot point becomes the origin (0, 0). Then, a scaling transformation is applied by dividing these translated coordinates by the scale factor. After scaling, the pivot point is restored to its original position by translating the UV coordinates back.

The pivot point itself is dynamic and can be moved in response to user input. This is handled in JavaScript by listening for keyboard events. When a key is pressed, such as 'A' or 'D' to move left or right, or 'W' or 'S' to move up or down, the corresponding UV coordinate of the pivot point is updated. The new coordinates are then clamped to the range of 0.0 to 1.0 to ensure they remain within the valid bounds of the texture.

The updated pivot coordinates are sent to the shader as a uniform variable. Both are passed to the fragment shader before each frame is rendered. In the fragment shader, the pivot and scale are used to compute the transformed UV coordinates. The shader adjusts the UV coordinates for each pixel fragment based on the transformation logic described earlier. The transformed UV coordinates are then used to sample the texture using the texture2D function, which fetches the corresponding color value from the texture map.

4. User's instruction

In order to spin up the project, user need to clone repository and serve all files with http server. After that open it in browser of choice on the correspondent resource. On Figure 4.1 displayed user interface.

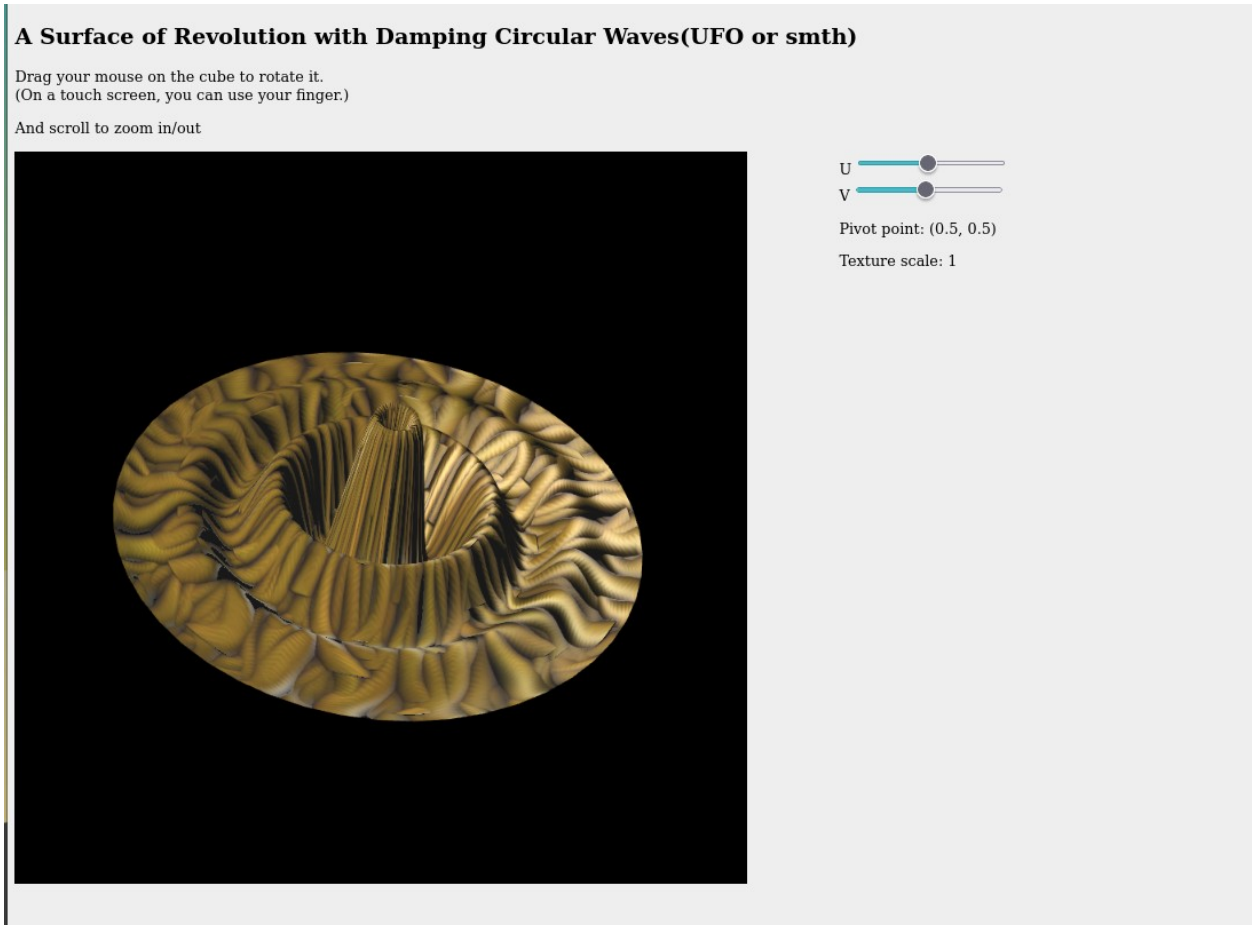

Figure 4.1 – User interface visual representation

To scale down/up texture – press keys ‘q’ and ‘e’ respectively. On the right panel updated value will be displayed next to label “Texture scale”. On the Figure 4.2 displayed 3x scaled texture.

Pivot point position also updates based on the key pressed:

- ‘a’ and ‘d’ to adjust ‘u’ coordinate
- ‘w’ and ‘s’ to adjust ‘v’ coordinate

On the right panel updated value will be displayed next to label “Pivot point”. On the Figure 4.3 displayed figure with moved pivot point.

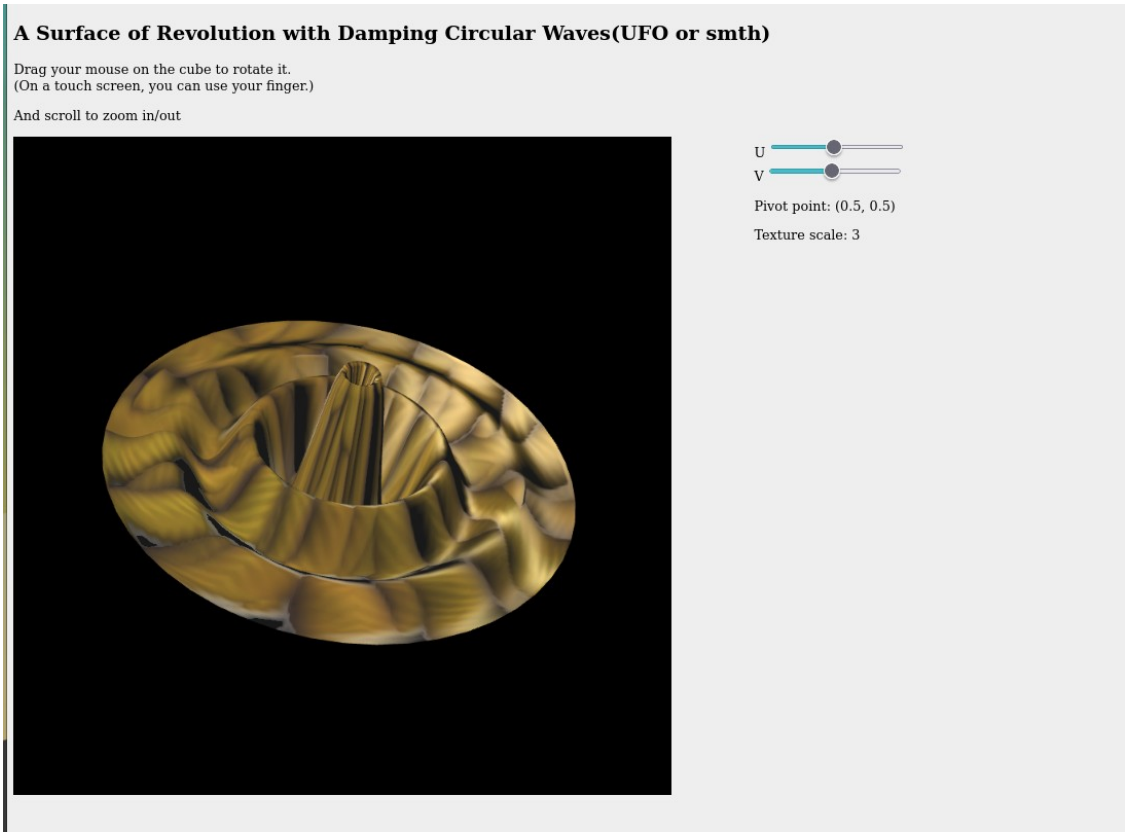

Figure 4.2 – 3x scaled texture

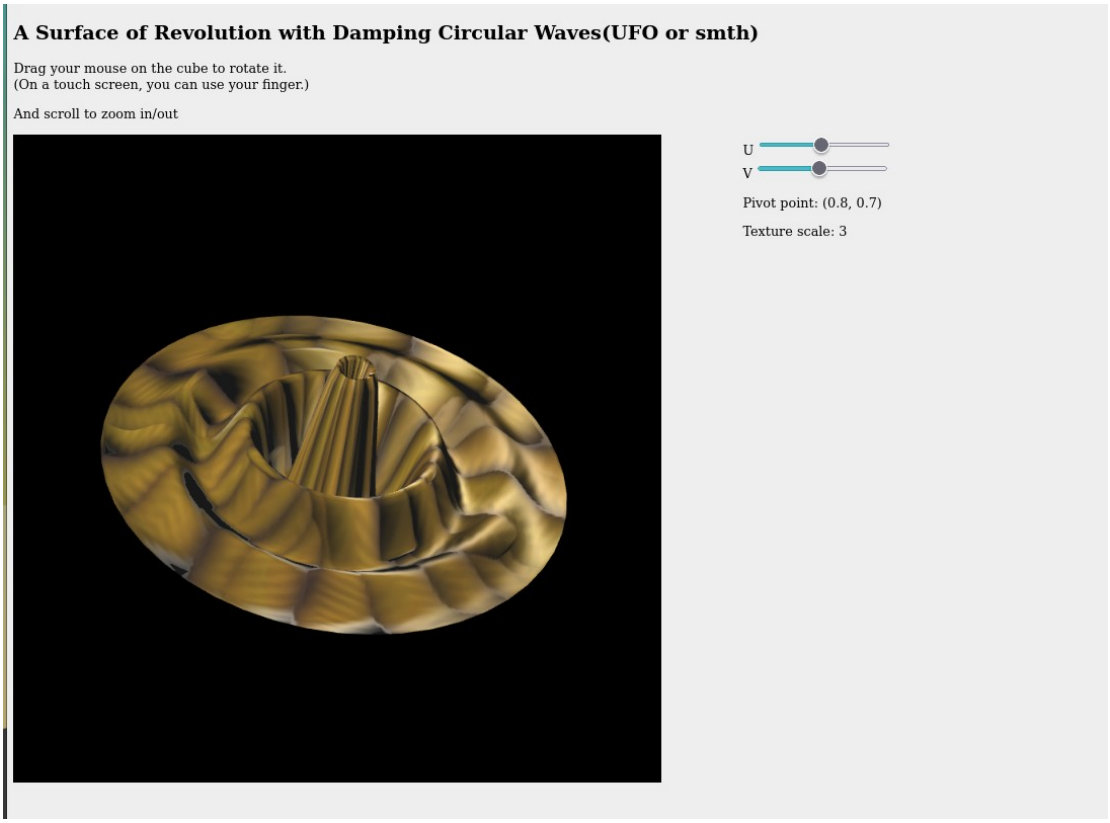

Figure 4.3 – texture with moved pivot point

5. Sample of source code

// shader.glsl

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

uniform mat4 ModelViewProjectionMatrix;
uniform mat4 ModelViewMatrix;
uniform mat3 NormalMatrix;
uniform vec3 scale;

varying vec3 vEyePos;
varying vec2 vTexCoord;
varying mat3 TBN;
void main() {
   vec3 scaled = vertex * scale;

   vec4 eyePos = ModelViewMatrix * vec4(scaled, 1.0);
   vEyePos = eyePos.xyz;

   vec3 bitangent = cross(normal, tangent);
   TBN = mat3(normalize(NormalMatrix * tangent),
          normalize(NormalMatrix * bitangent),
      normalize(NormalMatrix * normal)
   );
   vTexCoord = texCoord;

   gl_Position = ModelViewProjectionMatrix * vec4(scaled, 1.0);
}
`;
// Fragment shader
const fragmentShaderSource = `
#ifdef GL_FRAGMENT_PRECISION_HIGH
  precision highp float;
#else
```

```
  precision mediump float;
#endif

varying vec3 vEyePos;
varying vec2 vTexCoord;
varying mat3 TBN;

uniform vec4 color;
uniform vec3 lightDir;

uniform float Ka;
uniform float Kd;
uniform float Ks;
uniform float Sh;

uniform sampler2D diffuseMap;
uniform sampler2D specularMap;
uniform sampler2D normalMap;

uniform vec2 pivot;
uniform float texScale;
void main() {
        vec2 scaledTexCoord = pivot + (vTexCoord - pivot) / texScale;

        vec3 diffuseColor = texture2D(diffuseMap, scaledTexCoord).rgb;
        vec3 specularColor = texture2D(specularMap, scaledTexCoord).rgb;
        vec3 normalMapData = texture2D(normalMap, scaledTexCoord).rgb;

        vec3 N = normalize(normalMapData * 2.0 - 1.0);
        N = normalize(TBN * N);

        vec3 L = normalize(lightDir);
        vec3 V = normalize(-vEyePos);

        vec3 ambient = Ka * color.rgb;

        float lambertian = max(dot(N, L), 0.0);
        vec3 diffuse = Kd * diffuseColor * lambertian;
```