**National Technical University of Ukraine**

**"Kyiv Polytechnic Institute named after Igor Sikorsky"**

Educational and Scientific Institute of Nuclear and Thermal Energy

Department of Digital Technologies in Energy

Visualization of graphical and geometric information

**Calculation and graphics work**

Topic: «Operations on texture coordinates»

Variant 15

**Performed:**  
5th year student,

the TR-43mp group

Stepanenko M.F.

**Inspected:**

PhD Kaleniuk O. S.

Kyiv-2024

**TASK**

The objective of this assignment is to enhance texture mapping by implementing the scaling of texture coordinates around a user-specified point on a surface. This task builds upon the texture mapping functionality previously developed in the Control assignment, reusing it as a foundation for this work.

The implementation focuses on scaling the texture coordinates in the “u” and “v” parameter space. The scaling operation is performed relative to a dynamically adjustable center point, which can be moved across the surface. This introduces an interactive element, enabling users to modify the scaling behavior in real time based on the location of the transformation center.

To ensure user-friendly interaction, keyboard controls are provided for moving the center point along the surface. The following keys are assigned for this purpose:

* The **A** and **D** keys allow movement along the “u” parameter, corresponding to horizontal adjustments.
* The **W** and **S** keys enable movement along the “v” parameter, corresponding to vertical adjustments.

**THEORETICAL INFORMATION**

**WebGL** (Web Graphics Library) is a JavaScript API for rendering 2D and 3D graphics within web browsers. It is a standard designed to create visually rich and interactive web scenes and to integrate 3D graphics into websites without the need for plugins. WebGL is based on OpenGL ES, a standard for embedded systems primarily used in mobile devices.

Key features of WebGL include leveraging hardware-accelerated graphics for real-time rendering of 3D graphics, creating complex visual effects such as shadows and reflections, and enabling user interaction through input devices like a mouse and keyboard. WebGL is widely employed in web development to create impressive web interfaces and virtual reality experiences.

**Shaders** are programs written in GLSL (OpenGL Shading Language) used to handle computations related to rendering graphical objects in a 3D space. In WebGL, shaders are responsible for determining the positions of object vertices, their colors, and computing lighting effects and other graphical attributes.

Shaders are divided into two types: vertex shaders and fragment shaders. Vertex shaders process the vertices of objects, determining their positions in space, while fragment shaders compute the color of each pixel in the image, including lighting and other effects. These shaders form a critical part of WebGL’s programmable pipeline, enabling flexible and realistic rendering of graphical objects in a web browser.

**Vertex shaders** perform calculations for each vertex of an object in 3D space. Their primary tasks include transforming vertices from their local space to global space, defining their color and other attributes. These computations involve matrix operations to determine the precise position of vertices in space, considering camera and lighting effects. They also facilitate the transfer of data like normals and texture coordinates between vertices, generating output data used by fragment shaders to compute pixel colors.

**Fragment shaders** are responsible for determining the color of each pixel in the image. They operate on fragments generated after rasterizing the triangles processed by the vertex shader. The main goal of the fragment shader is to compute the final color for each pixel.

**Texture mapping** is a method of adding surface detail—such as color, texture, glossiness, or matte effects—to 3D objects (polygons). This technique simulates real-world materials like paper, wood, stone, or metal. The quality of a texture surface depends on texels, which refer to the number of pixels in the smallest unit of the texture. Since a texture is essentially an image, its resolution and format play a significant role in the overall visual quality of graphics in a 3D application.

Texture maps are applied to create specific visual properties on the surface of a given shape. This process is akin to wrapping patterned paper around a plain box. Each vertex in a 3D model is assigned texture coordinates (known as UV coordinates in 2D). The sampling locations of the image are then interpolated across the model's surface, resulting in the final visual effect.

**UV mapping** refers to the process in 3D modeling of projecting a 2D image onto a 3D model. The letters U and V represent the axes of the texture plane, as the letters X, Y, and Z are already used for spatial coordinates. UV mapping establishes the correspondence between coordinates on the surface of a 3D object (X, Y, Z) and coordinates on the texture (U, V). Typically, U and V values are normalized to range between 0 and 1.

Scaling around any point involves several steps using translation and scaling matrices. The process begins with creating a translation matrix (T) that moves the object from its initial position to the scaling point. Then, a scaling matrix (R) modifies the object’s size relative to this point. To scale around a specific point, the chosen point must first be shifted to the origin using the inverse of the translation matrix (T), denoted as . After scaling the object with matrix , the translation matrix is applied again to return the scaling point to its original position.

**IMPLEMENTATION DETAILS**

This implementation enhances a previously developed texture mapping solution by introducing scaling of texture coordinates around a user-specified point in the UV parameter space. The goal is to provide an interactive and dynamic scaling mechanism where the user can adjust the scaling behavior in real time.

The application consists of several integrated modules written in WebGL and JavaScript. The primary components are:

1. **HTML Interface**: Provides the canvas for rendering the 3D surface and controls for user interaction, such as sliders and key bindings.
2. **JavaScript Logic:** Manages the rendering pipeline, surface modeling, texture mapping, and user input handling.
3. **Shaders**: Custom GLSL vertex and fragment shaders that perform the core computations for vertex transformations and texture manipulation.

Key features include:

1. **Dynamic Texture Scaling**:
   1. Scaling is performed by modifying the UV coordinates within the vertex shader.
   2. The scaling operation is centered around a user-specified pivot point. This pivot point can be dynamically moved using keyboard controls (A/D for the U direction and W/S for the V direction).
2. **Interactive User Input**:
   1. The application uses keyboard and mouse input for interaction. Users can zoom in/out using the mouse wheel, rotate the surface with a mouse drag, and move the scaling pivot point with specific keys.
3. **Shader Logic**:
   1. Vertex shaders calculate the scaled texture coordinates and adjust their position in the UV space relative to the pivot.
   2. Fragment shaders apply texture sampling using the modified UV coordinates and handle lighting, shading, and highlighting.

**Surface Modeling**: The surface is modeled as a conical structure based on Cartesian Folium equations. The **ConicalSurfaceModel** class generates vertices, normals, tangents, bitangents, and UV coordinates. UV coordinates are normalized between 0 and 1, ensuring they map appropriately to the texture.

**Translation and Scaling Matrices**: The scaling operation is achieved using translation and scaling matrices. The UV coordinates are first translated to align the pivot point with the origin, scaled, and then translated back to their original position.

**Highlighting the Pivot Point**: To enhance user interaction, the pivot point is visually highlighted in red when it is close to the UV coordinates of a rendered pixel. The fragment shader calculates the distance between the current UV and the highlight position and blends the highlight color if within a threshold.

**Texture Loading**: The implementation uses three textures: diffuse, specular, and normal. These textures are loaded asynchronously, and rendering is deferred until all textures are ready.

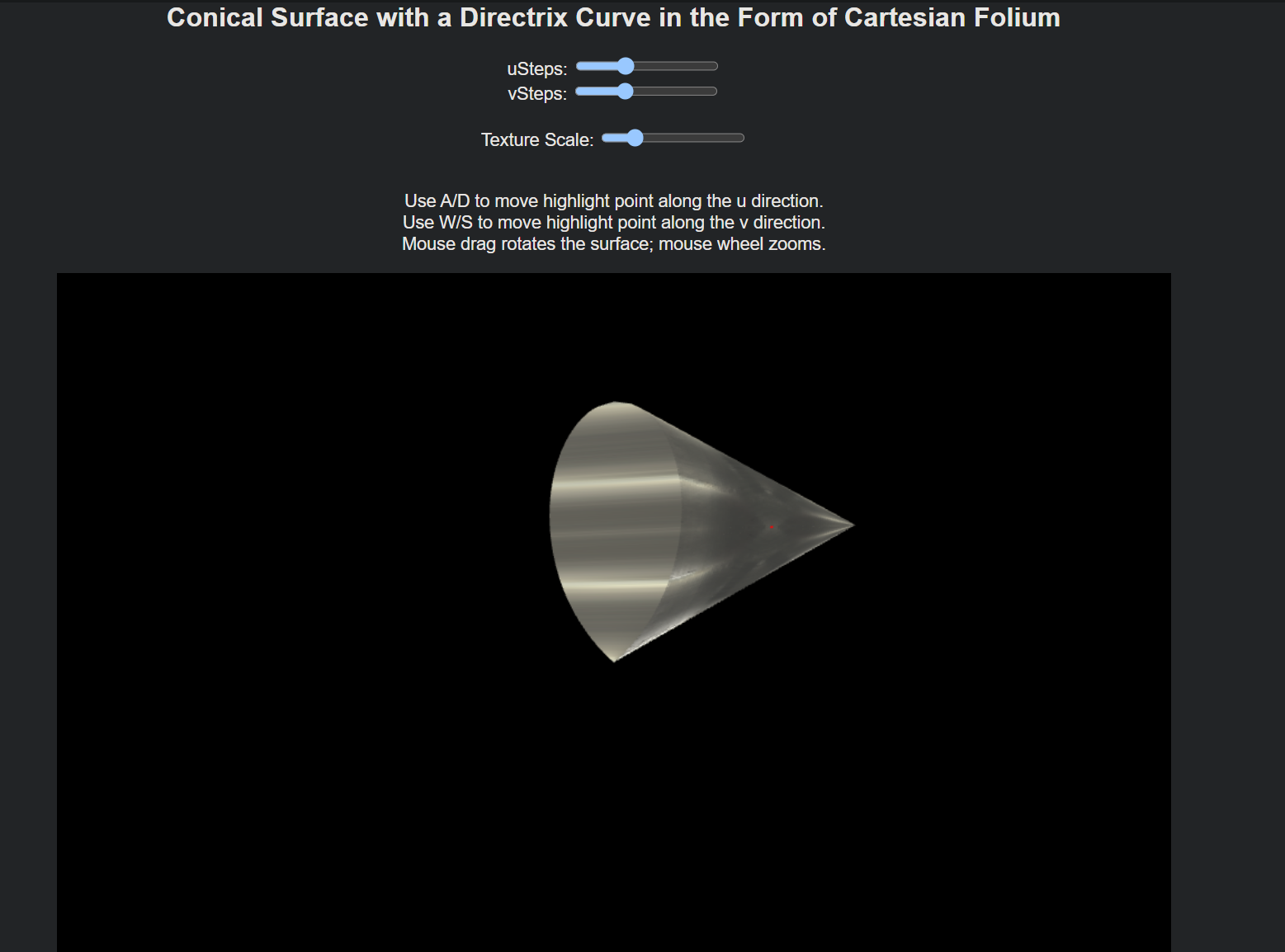
**User Interaction**: The application includes slider controls to adjust the surface's U and V steps and the scaling factor of the texture. Keyboard controls (A, D, W, S) allow the user to move the scaling pivot point interactively.

**Rendering Pipeline**: The rendering process is managed by WebGL. A perspective projection matrix and a model-view matrix are used to position the camera and surface. The rendering loop continuously updates the surface, handling lighting and user interactions.

**USER`S INSTRUCTION**

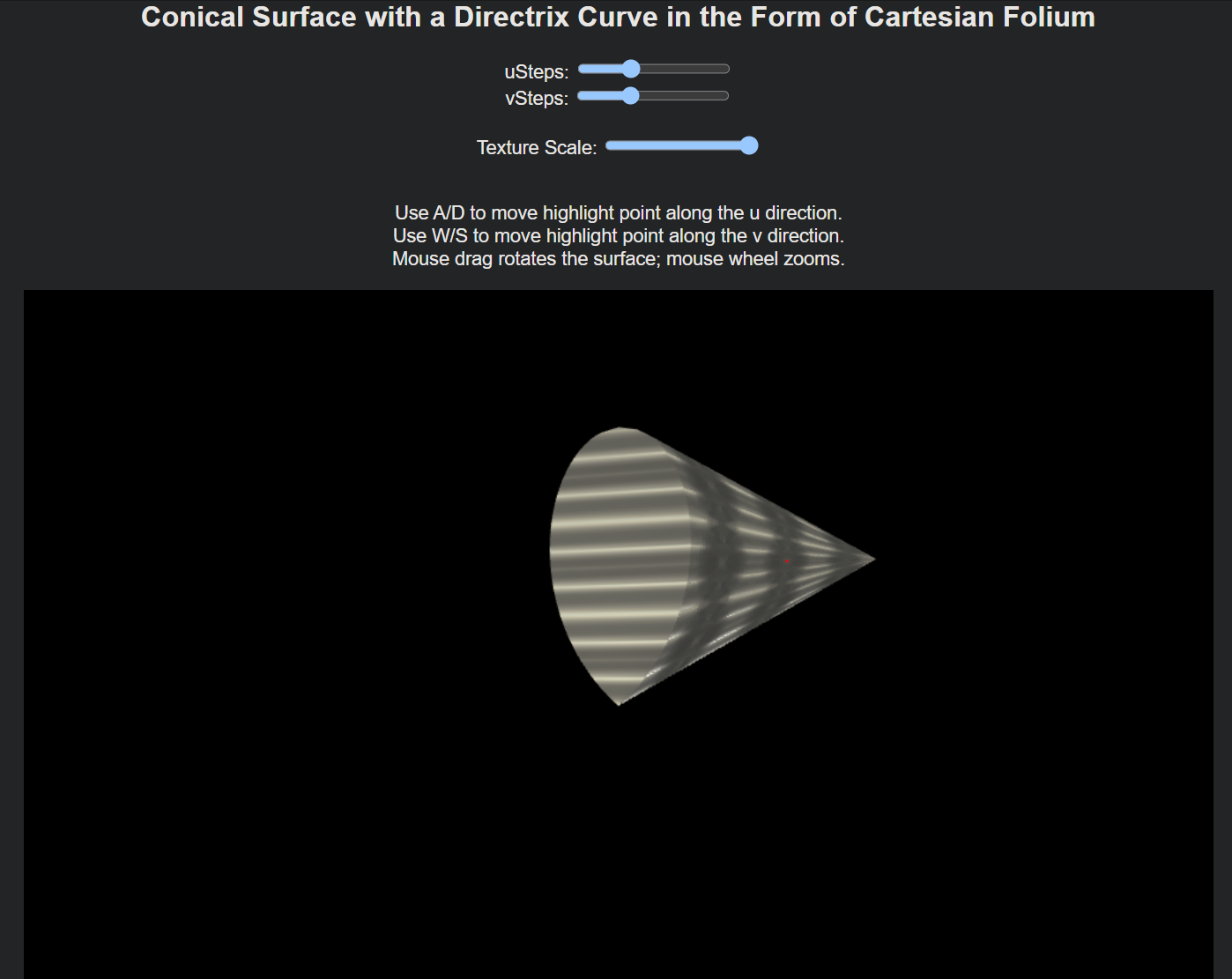
The application provides an interactive environment to explore texture scaling on a 3D surface. Upon launching, you will see the central canvas where the conical surface is rendered. Alongside, there are control elements including sliders to adjust parameters like surface resolution and texture scaling, and instructions for navigating the scene.

To begin, open the application in a browser that supports WebGL. The rendered surface will appear in the canvas area, surrounded by control elements. You can manipulate the resolution of the surface using the **uSteps** and **vSteps** sliders, which adjust the detail level of the rendered geometry. Moving these sliders alters the smoothness of the surface in real time, providing immediate feedback. Figure 1 demonstrates the application interface with default settings.



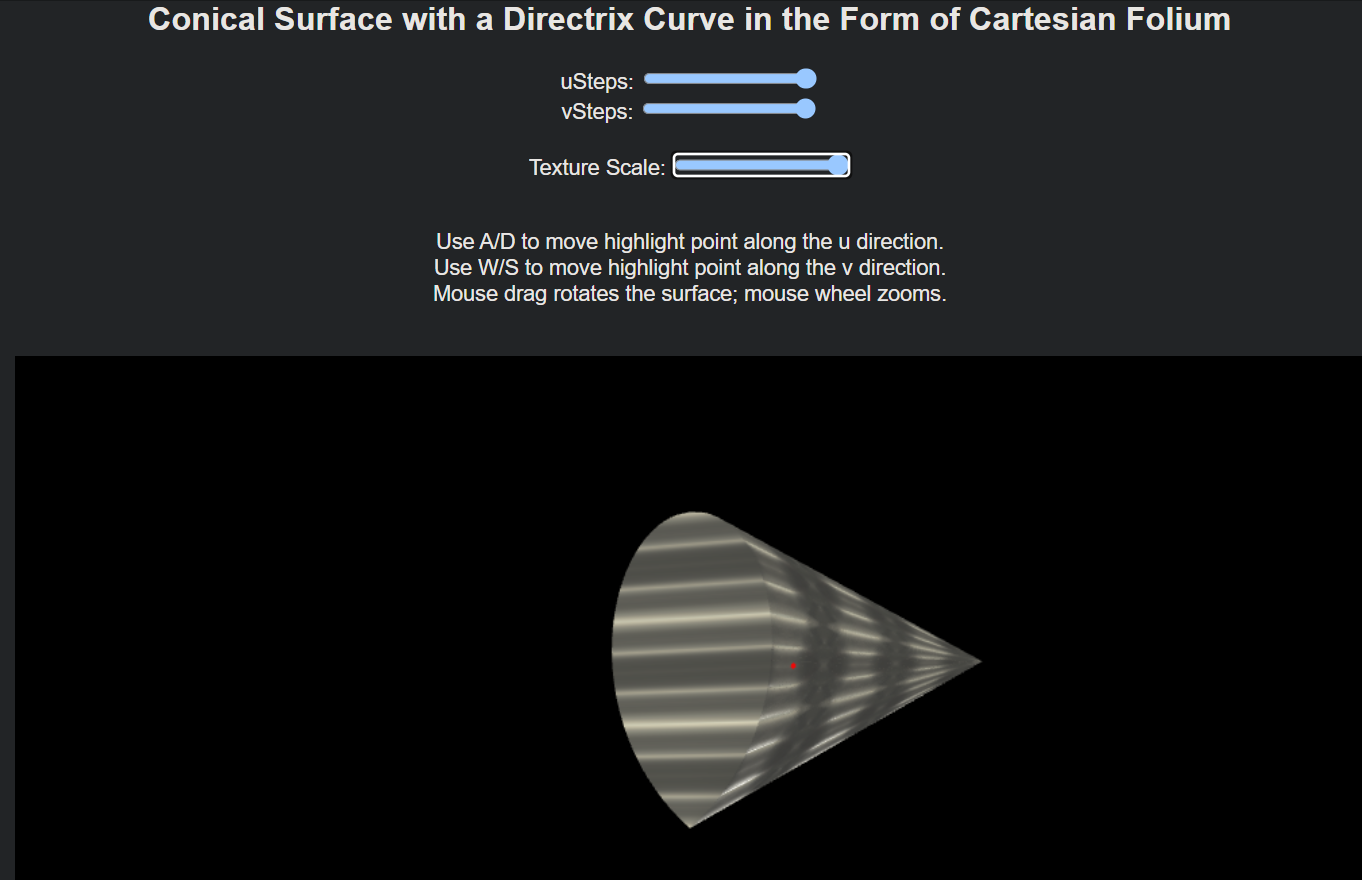
**Figure 1.** Application Interface

Adjusting the texture scaling is straightforward. The **Texture Scale** slider allows you to increase or decrease the size of the texture applied to the surface. Scaling is centered around a pivot point, which can be dynamically repositioned. Figure 2 illustrates how modifying the texture scale changes the appearance of the surface, showcasing the effect of real-time scaling.



**Figure 2.** Adjusting Texture Scale

Repositioning the pivot point is achieved using the keyboard. Pressing the A or D keys shifts the pivot along the horizontal (U) axis, while W or S moves it vertically along the V axis. Figure 3 shows figure with the pivot moved to the left using D key. This interactive feature allows precise control over the scaling transformation and makes it easy to experiment with various effects.

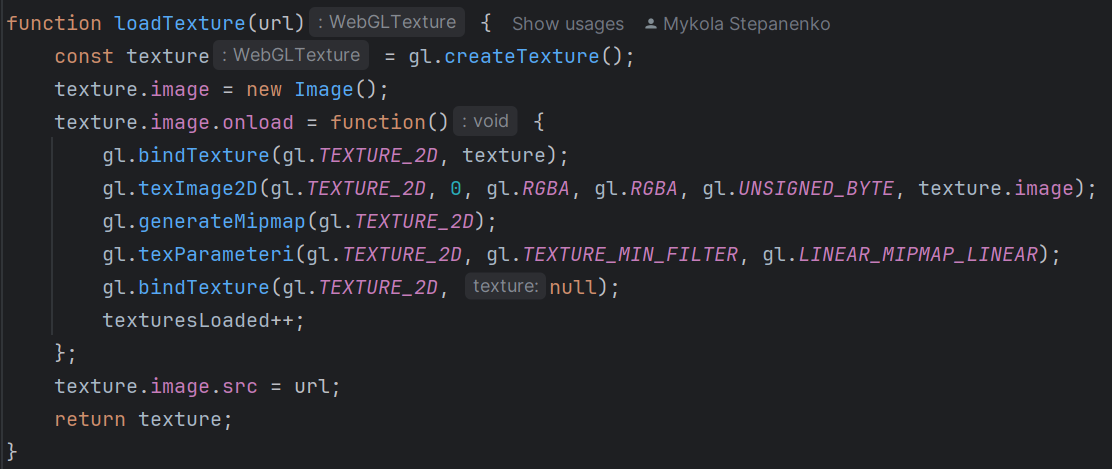


**Figure 3.** Pivot Point Highlighting

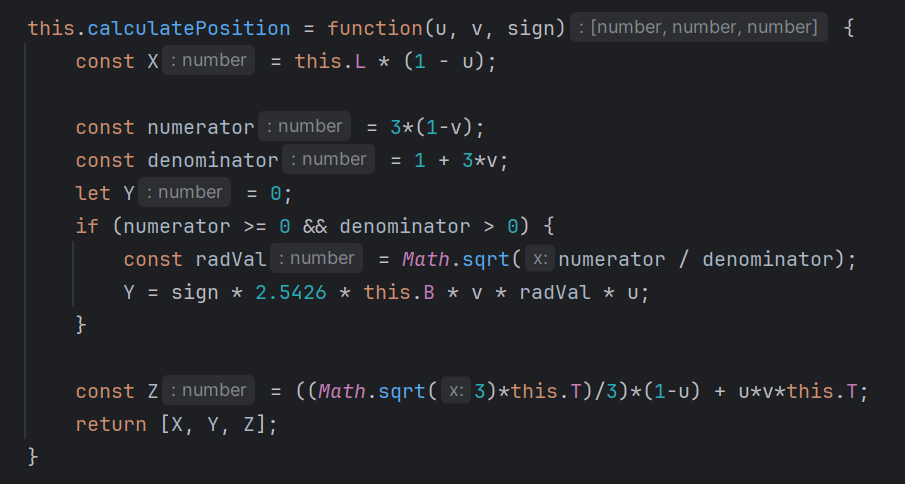
Through this interface, you can fully explore the dynamic behavior of texture scaling and gain a deeper understanding of how UV coordinates interact with the surface geometry.

**SOURCE CODE**

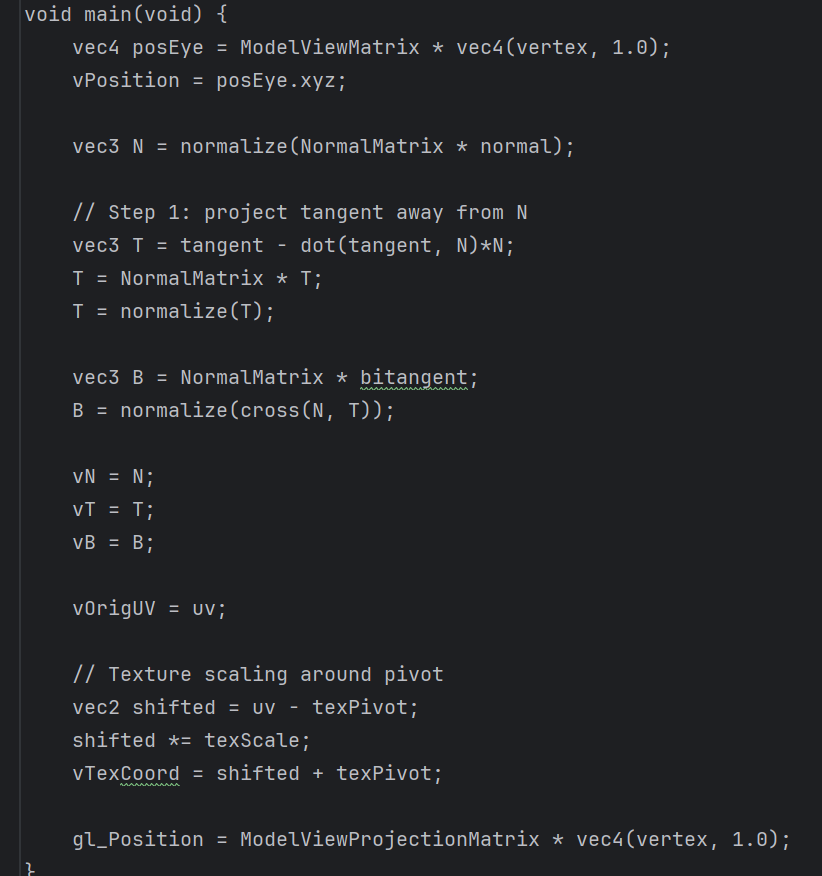
Function responsible for loading a texture:



Function responsible for calculating vertex positions on the conical surface:



Handling texture scaling around a pivot point in the vertex shader:



Highlighting the pivot point in the fragment shader:

