Assignment 1

Name: Abhishek Kumar [MT2024005]
Instructor: Prof. Jaya

February 8, 2025

1 Brief Overview

This project implements an interactive WebGL application that allows users to draw shapes, select them, modify their properties, and perform ear clipping triangulation for rendering. Key functionalities include:

- **Drawing Shapes:** Users can click to define polygon vertices.
- **Triangulation:** Ear clipping algorithm converts polygons into triangles for rendering.
- **Shape Selection & Manipulation:** Users can select and transform shapes (move, scale, rotate, change color).
- **Cursor Controls:** Option to toggle cursor visibility and interaction.
- **Z-Index Manipulation:** Adjust the rendering order of shapes.
- **Keyboard & Mouse Inputs:** Supports multiple interaction modes.

2 Functionalities and Implementation

2.1 WebGL Setup & Initialization

- The WebGL context is retrieved from an HTML `<canvas>` element.
- **Shaders:**
  - Vertex shader: Handles transformation of shape coordinates.
  - Fragment shader: Controls pixel colors using a uniform variable `uColor`.
- The shader program is compiled, linked, and used throughout the application.

2.2 Shape Drawing & Triangulation

- **Mouse Click Events:**
  - Clicking on the canvas adds a vertex to the current shape.
  - If a shape has three or more vertices, it can be converted into triangles via ear clipping.
- **Ear Clipping Algorithm:**
  - Iteratively removes triangles from the polygon while ensuring the convexity of remaining vertices.
  - The extracted triangles are stored and used for rendering.

2.3 Shape Selection & Manipulation

- **Click Detection (`getClickedShapeIndex`):**
  - Checks if a clicked point falls inside any triangle of an existing shape.
  - If true, sets `selectedShapeIndex` and updates the UI display.
- **Transformation Functions:**
  - `transformShape`: Applies translation, scaling, and rotation transformations to shapes.
  - `calculateCentroid`: Computes the centroid of a shape, used for scaling and rotation calculations.
  - `hexToRgb`: Converts hex color values to RGB format for WebGL rendering.
- **Rendering (`drawScene`):**
  - Clears the WebGL canvas (`gl.clear()`).
  - Sorts shapes by `zIndex`.
  - Draws each shape using `gl.drawArrays(gl.TRIANGLES, ...)`.

2.4 Cursor & Interaction Controls

- **Hiding Cursor (`toggleCursorButton`)**
  - Clicking the "Hide Cursor" button toggles visibility.
- **Disabling Cursor Interaction (`toggleCursorInteractionButton`)**
  - Clicking the "Disable Cursor" button prevents shape selection.

2.5 Z-Index Adjustment

- Shapes have a `zIndex` property that determines the rendering order.
- Users can use `b` (move backward) and `f` (move forward) to reorder shapes.
- The scene is re-rendered with shapes sorted by `zIndex`.

2.6 Rendering and Drawing Scene

- **Path Rendering (`drawSmoothLines`)**

– If a shape is being drawn, smooth lines are displayed using `gl.LINE`$_{S}TRIP$.

- **Star Indicator (`drawStar`)**
  - A star is drawn at the last clicked position for better visual feedback.

3 Screenshots and Significance

3.1 Drawing Shapes and Triangulation

Users define a shape by clicking, and ear clipping triangulation converts it into drawable triangles.

3.2 Shape Selection & Transformation

Selected shapes can be moved, scaled, rotated, and recolored dynamically.

3.3 Z-Index Manipulation

Users can adjust the layering order of shapes to control rendering priority.

4 Keyboard / Mouse Controls

- **Arrow Keys:** Move selected shape.
- **+ / -:** Scale shape up/down.
- **r / R:** Rotate clockwise/counterclockwise.
- **b / f:** Change shape rendering order.
- **c:** Cycle through colors.
- **Mouse Click:** Selects or adds points to a shape.
- **Hide Cursor Button:** Toggles cursor visibility.
- **Disable Cursor Button:** Prevents interaction.

5 Questions

1.) When we perform transformations on the scene, we treat the scene as a grouping of all the shapes as a single entity. How would you implement grouping and ungrouping shapes if the user would like to choose which shapes will be treated together as a group for transformations?

Ans: To implement grouping and ungrouping of shapes in WebGL, we can introduce a grouping mechanism where selected shapes share a common transformation matrix. When a user selects multiple shapes and groups them, we store references to these shapes in a group object that maintains a shared transformation matrix. Instead of transforming each shape individually, we apply transformations to the group's matrix, ensuring that all shapes in the group move, scale, or rotate together while preserving their relative positions. Each shape's local coordinates remain unchanged, and their final positions are computed using the group's matrix during rendering. For ungrouping, we simply remove shapes from the group, assigning them independent transformation matrices again. In WebGL, this can be efficiently implemented by maintaining an array of groups, modifying the vertex shader to apply group-level transformations, and updating the vertex buffer data accordingly. Additionally, UI elements such as a grouping button and a bounding box can help users visualize the grouped entities effectively.

2. Why is the use of centroid important in transforming a primitive or a group of primitives?

Ans: The centroid plays a crucial role in transforming a primitive or a group of primitives because it serves as the pivot point for transformations like scaling, rotation, and translation. When applying transformations in WebGL, they are typically performed around the origin (0,0,0) of the coordinate system. However, if we directly transform a shape without considering its centroid, the shape may not behave as expected—especially in rotations and scalings.For instance, if a shape is rotated around an arbitrary point instead of its centroid, it may appear to orbit rather than rotate in place. Similarly, when scaling, ignoring the centroid can lead to asymmetrical distortion rather than uniform growth or shrinkage. When transforming a group of primitives, calculating the centroid of the entire group ensures that all transformations are applied relative to a common reference point, preserving their spatial relationships. In WebGL, the centroid can be computed as the average of all vertex positions in the shape or group. By shifting the transformation matrix to center around the centroid, we achieve accurate and intuitive transformations.