# Assignment -2

Harsh Chandrakar—MT2024054

Instructor: Prof. Jaya Sreevalsan Nair

Date: 6 April 2025

## 1. Brief Overview

This assignment focuses on rendering and manipulating 3D models in WebGL. Key features implemented include:

- 3D Model Creation: Three models (a cup ,a cube and a random model) were designed using Blender with boolean operations.
- View Modes: A toggleable "Top View" (orthographic projection along the z-axis) and "3D View" (perspective projection with camera rotation).
- Object Manipulation: Translation along quadratic paths, rotation using quaternions, scaling, and object picking.
- Animation: Smooth interpolation along user-defined paths with adjustable speed.
- World Axes: RGB-colored axes (cylinder-cone structures) for spatial reference.

## 2. Implementation Details

### 2.1. 3D Model Creation

- Blender Operations: The cup model was created using a boolean subtraction to carve a hollow interior. The cube underwent subdivision and sculpting for rounded edges.
- Export: Models were saved as .obj files and imported into WebGL using webgl-objloader.

### 2.2. View Modes

- Top View: Camera positioned at (0, 5, 0) looking at the origin.
- 3D View: Virtual trackball implemented for camera rotation using spherical coordinates. Mouse drag adjusts theta (horizontal) and phi (vertical) angles.

## 2.3. Object Picking

- Ray Casting: In Top View, mouse clicks trigger ray-object intersection checks. Selected objects are highlighted orange.
- Screen-to-World Coordinates: NDC coordinates mapped using inverse view-projection matrices.

## 2.4. Path Animation

- Quadratic Bézier Curve: Defined by three points (p0, p1, p2). Coefficients solved using: $p(t)=(1-t)^2p_0+2(1-t)tp_1+t^2p_2$
- Speed Control: Animation duration adjustable via slider or keyboard ([/]).

## 2.5. Transformations

- Quaternions: Rotations applied using gl-matrix to avoid gimbal lock.
- Matrix Composition: Transformations ordered as Translate → Rotate → Scale to ensure correct spatial behavior.

# 3. Questions and Answers

3.1. Question: To what extent were you able to reuse code from Assignment 1?

Ans: The code reuse from Assignment 1 was moderate and primarily focused on the basic structure for web-based Open GL and event handling. In particular, the ShaderLoader class was reused as well as some parts of the render pipeline (e.g., creation of buffers, configuration of attributes) from my previous implementation of web based OpenGL. In addition, the event handling for user keyboard/mouse interactions (e.g., transitioning of views, selection of objects) and implementation of error handling were reused. The 3D specific components were significantly changed, e.g., generating proper perspective projection, implementing rotations in quaternion space, and implementing transformation based on matrices. The custom classes, VirtualTrackball and ViewManager were completely new classes. The OBJLoader and Renderer were implemented to work with 3D models and consider the depth when rendering the scene.

3.2. Question: What were the primary changes in the use of WebGL in moving from 2D to 3D?

Ans: The move to 3D necessitated some significant changes in WebGL program:

- Projection: orthographic projection to perspective projection using glMatrix.mat4.perspective, which provides depth in an animation.

- Depth and Culling: DEPTH_TEST and CULL_FACE are now enabled to allow for 3D viewing.
- Matrix Operations: Used 4x4 matrices for transformations (translation, rotation, scaling) via glMatrix to replace 2D affine transformations.
- Shader Complexity: The vertex shaders are now processing a 3D coordinate system, normals and lighting calculations.
- Quaternions: A new addition of quaternion-based rotations was introduced for in VirtualTrackball which would allow smooth 3D camera/orientation control without gimbal lock.

3.3. Question: How were the translate, scale and rotate matrices arranged? Can your implementation allow rotations and scaling during the movement?

Ans: Editing of the transformations occurs in a hierarchical manner:

- Translation: Object positions during animation are calculated using updateModelPosition, which offsets the vertices according to the target position.
- Rotation: Trackball quaternions are converted to rotation matrices (glMatrix.mat4.fromQuat) and applied prior to translation.
- Scaling: Vertices are uniformly scaled according to a scale factor relative to the centroid of the model.

Additionally, the system is capable of multiple transformations, while the animateAlongPath method updates positions dynamically during translation, it is also applying the current rotation matrix and scale factor to the vertices. This allows for smooth rotation and scaling applied during translation of the object.

3.4. Question: How did you choose a value for t1 in computing the coefficients of the quadratic curve? How would you extend this to interpolating through n points (n > 3) and still obtaining a smooth curve?

Ans: To perform interpolation for a quadratic Bézier curve, t was sampled uniformly (100 values between 0 and 1) to ensure smooth interpolation of control points. The parameter t was increased linearly and each intermediate point was computed with the following Bézier formula:

$$B(t) = (1-t)^2P_0 \quad 2(1-t)tP_1 \quad t^2P_2$$

If there are n > 3 spaces sampled, a Bézier spline or B-spline could be used. Each segment of the spline would use a subset of points (for example, cubic Bézier with four points per segment). This works without loss of generality, as each segment of the cubic Bézier will be continuous ($C^2$) due to the control points aligned from one segment to the next.

Alternatively, if desired, a Catmull-Rom spline can interpolate all point values directly with tension parameters to control the smoothness of the interpolation. The smooth curve in either case is defined at the junction of the spaces with continuity in the derivatives of the provided samples to limit sudden directional changes in motion paths.

## 4. Screenshots

Adding all the relevant screenshots for better understanding of the project:

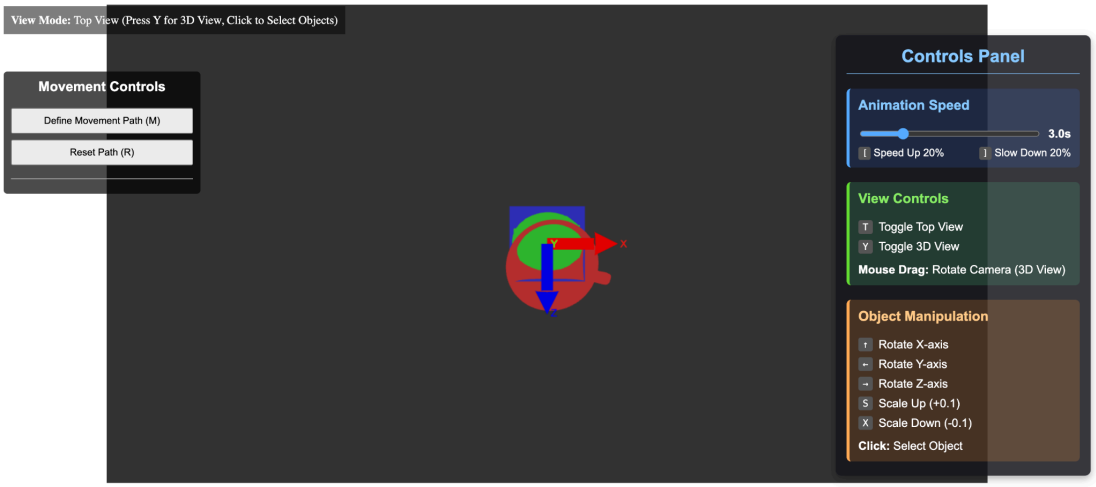


Figure 1: Top View with axes and models at origin.

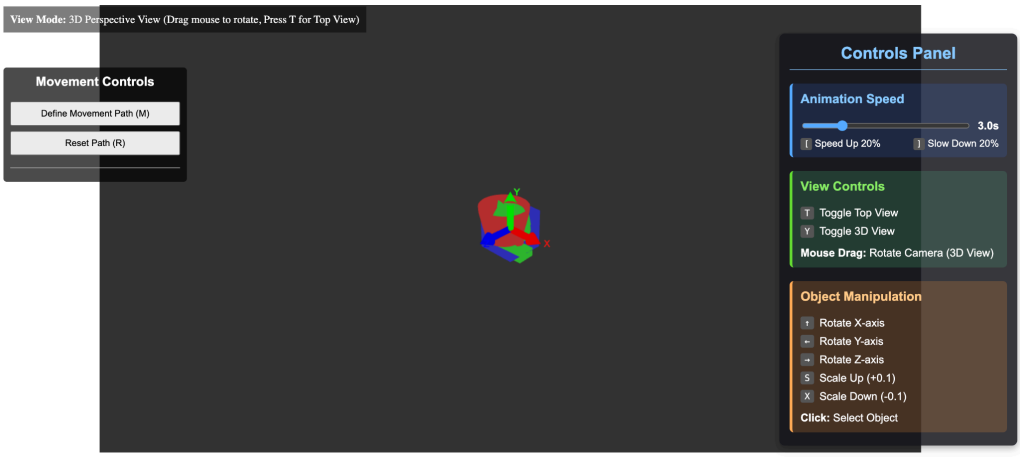


Figure 2: 3D View with camera rotated 45°.


Figure 3: Object selection (cup highlighted in orange).

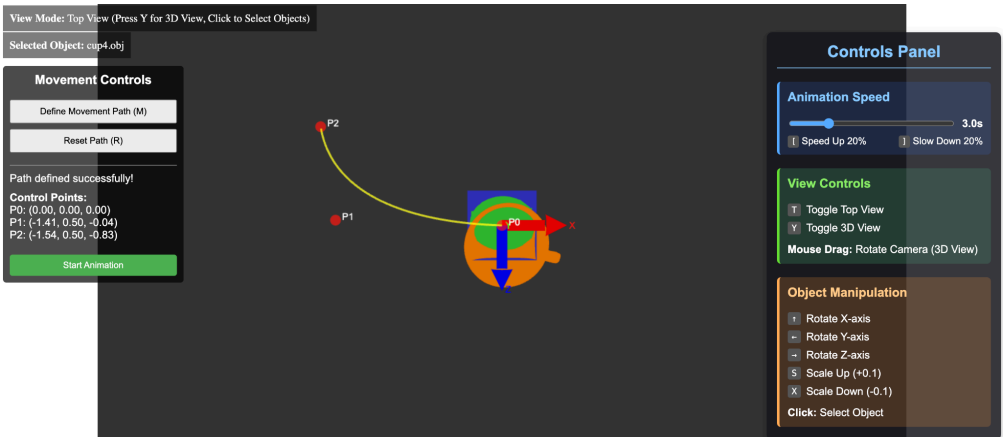


Figure 4: Quadratic path visualization (yellow curve) with control points P0, P1, P2.

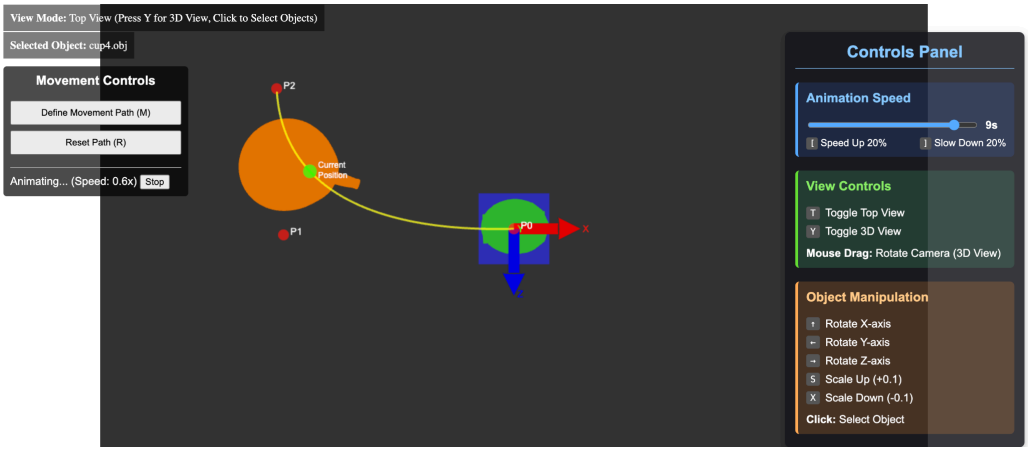


Figure 5: Cup animating along the path with a green position marker.

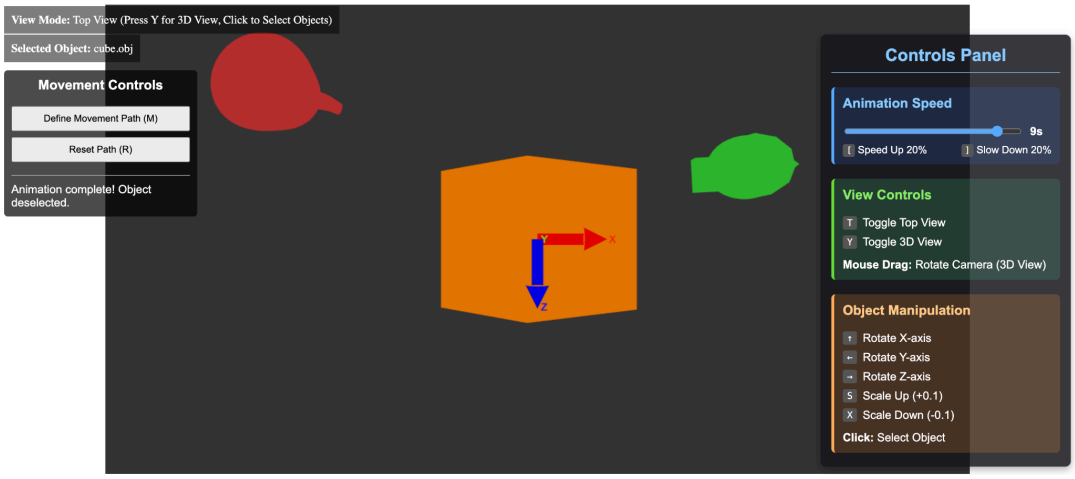


Figure 6: Scaled and rotated cube in Top View.

## 5. Keyboard/Mouse Controls

1. View Modes: T (Top View), Y (3D View).
2. Object Manipulation:
   ↑/←/→: Rotate selected object about X/Y/Z axes.
   S/X: Scale up/down.
3. Path Controls:
   M: Start path drawing.
   R: Reset path.
4. Camera: Mouse drag in 3D View to rotate.

## 6. Video Presentation Link

Link : https://youtu.be/KOg3IvJoAdE