Assignment 2

Rajan Khade — MT2024077

Instructor: Prof. Jaya & Prof. Srikanth *Date: April 10, 2025*

Implementation

The application allows for importing custom 3D models, selecting and transforming them, and animating their movement along a quadratic curve path.

Core Features Implemented

1. Multiple View Modes:
    - Top View mode for object selection and path definition
    - 3D View mode with rotatable camera for visualizing the scene
2. 3D Model Handling:
    - Successfully imported custom 3D models in common formats
    - Created a dedicated object-oriented architecture for managing model data
3. Object Transformations:
    - Scale, rotate, and translate operations on individual objects
    - Quaternion-based rotation to avoid gimbal lock
    - All transformations implemented relative to local coordinate systems
4. Path Definition and Movement:
    - Quadratic curve path definition through three points
    - Smooth object animation along the defined path
    - Variable speed control during movement
5. User Interaction:
    - Object picking in Top View mode
    - Camera rotation with both slider controls and mouse drag
    - Keyboard shortcuts for all transformations and view modes

Technical Implementation Details

3D Rendering Pipeline

The application uses WebGL with world, view, and projection matrices to create a complete 3D rendering pipeline. This extended the 2D implementation from Assignment 1 by adding proper 3D transformations and depth testing.

Transformation Matrix Arrangement

Transformation matrices follow the order:

`modelTransformMatrix = translationMatrix * rotationMatrix * scaleMatrix`.

This ensures that objects are first scaled, then rotated around their local origin, and finally translated to their world position.

Quadratic Curve Path

For the quadratic curve implementation, a parametric equation $p(t) = at^2 + bt + c$ was used where:

- t=0 corresponds to the starting position (p0)
- t=1 corresponds to the ending position (p2)
- t=0.5 was chosen for the middle control point (p1)

The coefficients were computed by solving a system of equations to ensure the curve passes through all three points.

Challenges Faced

1. Object Transformation Reference Frame:
    - Initially, transformations were happening relative to global axes instead of local object axes
    - Resolved by implementing proper quaternion-based rotations and maintaining local object coordinate systems
2. Object Picking:
    - I had to render a separate picking framebuffer that used unique color IDs and a depth buffer to determine which object was clicked.
    - Updated picking logic to account for object transformations
    - In current implementation of object picking using R channel, max 255 objects can be picked.
        i. This limitation can be easily removed by utilizing all 4 channels (RGBA)
3. Path Animation Smoothness:
    - Ensuring smooth movement along curved paths required precise coefficient calculation

Conclusion

The assignment successfully demonstrated key 3D graphics concepts including model transformation, view transformation, object picking, and animation along parametric curves. The implementation provides a solid foundation for more complex 3D graphics applications and effectively builds upon the concepts from Assignment 1.

By leveraging modern WebGL features and maintaining a clean architecture, the application achieves good performance even with multiple 3D models and complex transformations. The quaternion-based rotation system and local coordinate transformations ensure accurate and intuitive object manipulation.

Questions and Answers

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

My implementation successfully reused around 80% of Assignment 1 code. The core WebGL rendering pipeline and event handling framework remain the same.

I adapted the code by:

- Retaining the basic rendering and interaction framework from Assignment 1
- Adding 3D matrices (world, view, projection) in the vertex shader
- Implementing transformation matrices for 3D operations
- Creating object-oriented implementations for 3D models with dedicated classes to store object data
- Adding model importing functionality

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

The primary changes in WebGL for 3D rendering include:

- Implementing proper matrix transformations (world, view, projection) in the vertex shader
- Using perspective projection instead of orthographic projection
- Adding depth testing (gl.ENABLE_DEPTH_TEST) to handle occlusion of objects
- Implementing camera controls to navigate the 3D scene (rotation implementation)
- Using 3D coordinates (x,y,z) throughout the pipeline instead of 2D
- Adding quaternion-based rotation to properly handle 3D object rotation without gimbal lock
- Implementing proper lighting calculations in shaders for 3D rendering

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

The transformation matrices are arranged in the order:

modelTransformMatrix = translationMatrix * rotationMatrix * scaleMatrix.

This order ensures that:

1. Objects are first scaled relative to their local origin
2. Then rotated around their local origin
3. Finally translated to their world position

My implementation successfully allows for rotations and scaling during movement because:

- The operations are applied independently in the correct order
- I am using quaternions for rotation, which avoids gimbal lock issues
- The transformations are recalculated every frame during animation

- My code maintains a local reference frame for each object, allowing transforms to be applied consistently regardless of the object's current position or orientation

4. 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?

In the `computePathCoefficients` method, t1 was chosen as 0.5 (middle point) for the quadratic curve. This represents the parametric value at which the curve passes through the second control point. Choosing t1=0.5 creates a balanced curve where the second control point has equal influence on both halves of the curve.

To extend this to interpolate through n points (n > 3) while maintaining curve smoothness, I could implement:

1. Piecewise Quadratic Curves: Connect multiple quadratic curves, ensuring C1 continuity (continuous first derivatives) at junction points. For each consecutive triplet of points (Pi, Pi+1, Pi+2), compute a quadratic curve segment. The challenge is maintaining smoothness at junctions.
2. Cubic Splines: Use cubic polynomials instead of quadratic, which better handle smoothness constraints. For n points, I would solve an (n-2) × (n-2) system to find the second derivatives at each point.
3. Catmull-Rom Splines: A special type of cubic spline that ensures the curve passes through all control points while maintaining C1 continuity. For points P0 to Pn, each segment between Pi and Pi+1 would be influenced by Pi-1 and Pi+2.
4. B-splines or NURBS: These provide even more control over smoothness and can interpolate any number of points while maintaining higher-order continuity. They use basis functions to blend control point influences.

For my implementation, the Catmull-Rom approach would be most natural to extend my current code, as it would maintain the property of interpolating through all points while providing the desired smoothness.

References

https://webglfundamentals.org/

For extracting vertex, normals, etc. from `.obj` file

https://github.com/frenchtoast747/webgl-obj-loader

https://webglfundamentals.org/webgl/lessons/webgl-picking.html

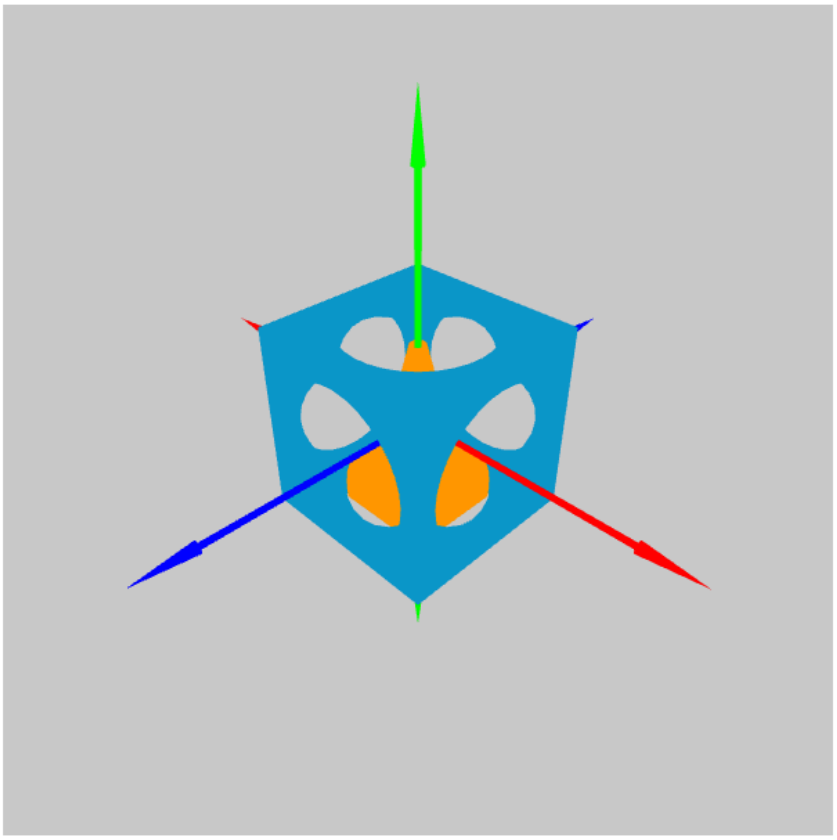
https://learnwebgl.brown37.net/11_advanced_rendering/selecting_objects.html

https://webglfundamentals.org/webgl/lessons/webgl-3d-lighting-point.html

Screenshots and Demo

https://youtu.be/Eem88xRh6R8

https://github.com/rajan-31/3D-Objects-on-a-Path



3D View [V] Top View [v]

X: 0 Y: 0 Z: 0 [t] Translate

Keyboard Actions

- **v/V:** Top/3D View
- **x/X:** Rotate Around X-axis (anti-clockwise / clockwise)
- **y/Y:** Rotate Around Y-axis (anti-clockwise / clockwise)
- **z/Z:** Rotate Around Z-axis (anti-clockwise / clockwise)
- **t:** Translate
- **←:** Scale Down
- **→:** Scale Up

Path Movement Controls

Step 1: Select an object in Top View [v] mode

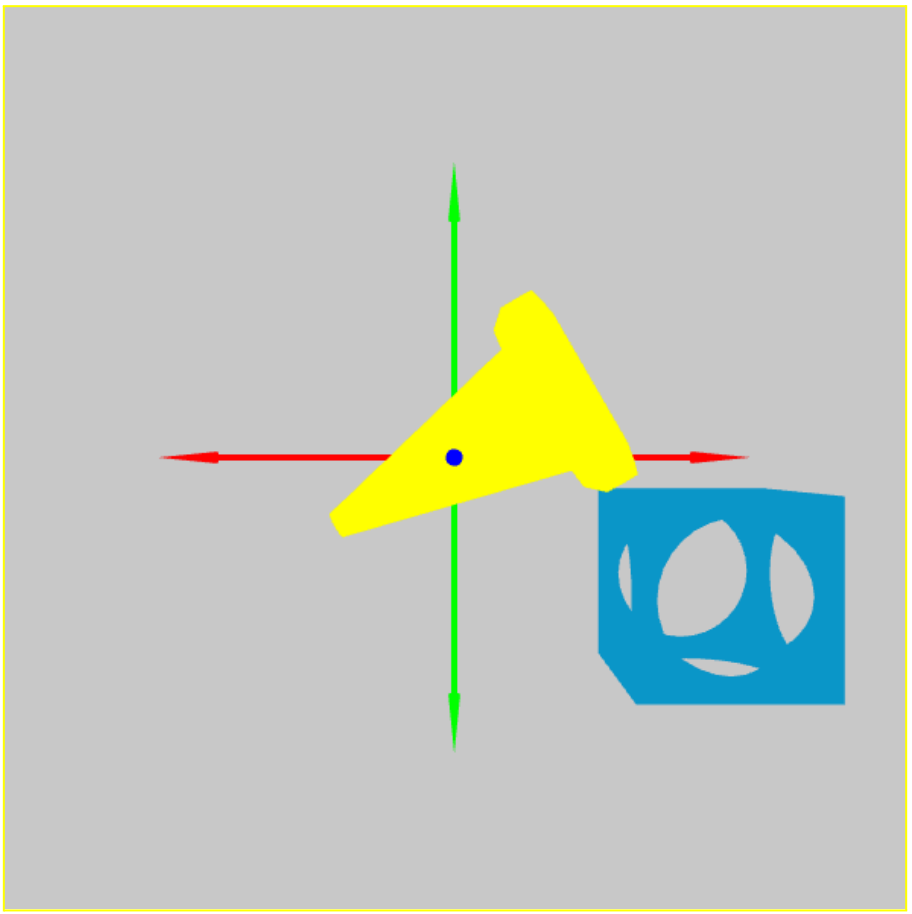
Step 2: Press [p] to start path definition

Step 3: Click on the canvas to define two more points (p1 and p2)

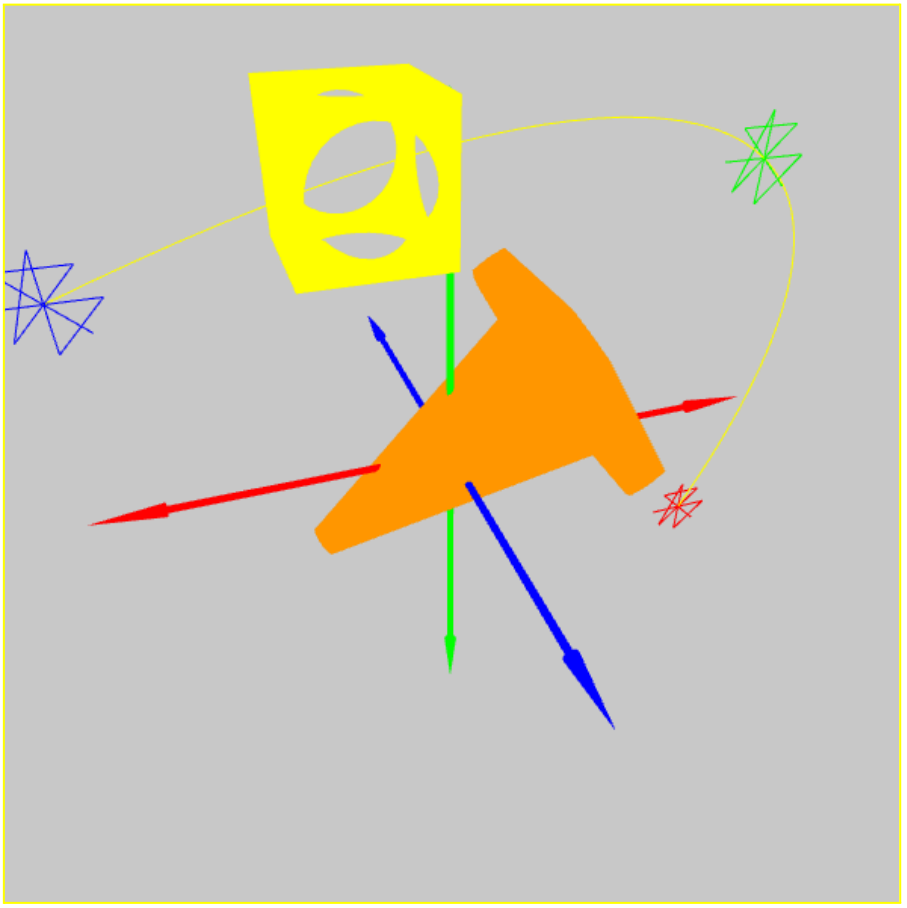
During movement:

- Press [↑] to increase speed
- Press [↓] to decrease speed
- Press [Esc] to cancel path definition

Select, Rotate, Translate, Scale



3D object on a path



Lighting