Real-Time 3D Wireframe Rendering Engine Quaternion Animation and Lambert Lighting

Project Documentation

January 12, 2026

Abstract

This project presents the design and implementation of a real-time software-based 3D wireframe rendering engine. The system implements a complete 3D graphics pipeline, including vector and matrix mathematics, perspective projection, depth-aware wireframe rendering, Lambert diffuse lighting, and smooth animation using cubic Bezier curves and quaternion-based rotation. All stages are implemented from first principles without relying on hardware graphics APIs, providing a clear mathematical understanding of real-time 3D rendering.

Contents

1 Introduction

Modern graphics pipelines rely heavily on linear algebra, geometry, and numerical methods. This project recreates a simplified but complete 3D rendering pipeline entirely in software. The goal is to understand the mathematical foundations behind 3D transformations, projection, lighting, and animation by implementing them manually.

The renderer draws 3D objects as wireframes on a 2D canvas and enhances visual quality through lighting and smooth motion. Advanced techniques such as spherical linear interpolation (SLERP), quaternion rotation, and Bezier animation are incorporated to avoid common issues like gimbal lock and discontinuous motion.

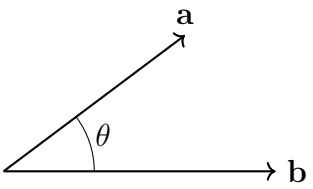
2 Mathematical Foundations

2.1 Vector Dot Product (Geometric View)

The dot product between two vectors $\mathbf{a}$ and $\mathbf{b}$ is defined as:

$$\mathbf{a} \cdot \mathbf{b} = |\mathbf{a}||\mathbf{b}| \cos\theta \tag{1}$$

This equation has a clear geometric interpretation: it measures how much one vector points in the direction of another.



If $\theta = 0°$, the vectors align and lighting is strongest. If $\theta = 90°$, the dot product becomes zero.

2.2 Vector Mathematics

A 3D vector is represented in Cartesian form as:

$$\mathbf{v} = (x, y, z) \tag{2}$$

The magnitude of a vector is:

$$|\mathbf{v}| = \sqrt{x^2 + y^2 + z^2} \tag{3}$$

Normalization produces a unit vector:

$$\hat{\mathbf{v}} = \frac{\mathbf{v}}{|\mathbf{v}|} \tag{4}$$

The dot product between two vectors is defined as:

$$\mathbf{a} \cdot \mathbf{b} = |\mathbf{a}||\mathbf{b}| \cos\theta \tag{5}$$

which is fundamental to lighting calculations.

2.3 Matrix Mathematics

All transformations are performed using 4×4 homogeneous matrices. A point is represented as:

$$\mathbf{p} = (x, y, z, 1) \tag{6}$$

Transformation matrices include translation, rotation, scaling, and projection. Multiple transformations are combined via matrix multiplication:

$$\mathbf{p}' = P \cdot V \cdot M \cdot \mathbf{p} \tag{7}$$

2.4 Model Transformation

The model matrix places an object into world space using scaling, rotation, and translation:

$$M = T \cdot R \cdot S \tag{8}$$

2.5 View Transformation

The view matrix transforms world coordinates into camera space. Conceptually, this is equivalent to moving the entire world so that the camera is at the origin.

2.6 Projection Transformation

Perspective projection converts 3D camera-space coordinates into clip space using a 4×4 matrix. The general asymmetric frustum projection matrix is:

$$P = \begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix} \tag{9}$$

The layout of this matrix is critical:

- The first two rows scale x and y into the view frustum
- The third row remaps depth into a non-linear range
- The -1 in the fourth row enables perspective division

After multiplication, perspective division is applied:

$$(x, y, z)_{\text{ndc}} = \left(\frac{x}{w}, \frac{y}{w}, \frac{z}{w}\right) \tag{10}$$

This step causes distant objects to appear smaller, producing the perspective effect.

3 Wireframe Rendering

Objects are rendered as wireframes by connecting edges between projected vertices. To ensure correct visual ordering, edges are sorted by average depth:

$$z_{edge} = \frac{z_1 + z_2}{2} \tag{11}$$

A painter's algorithm is used, drawing edges from back to front.

3.1 Circular Viewport Clipping

Only pixels inside a circular canvas are rendered. A point (x, y) is inside the viewport if:

$$(x - c_x)^2 + (y - c_y)^2 \leq r^2 \tag{12}$$

4 Lighting Model

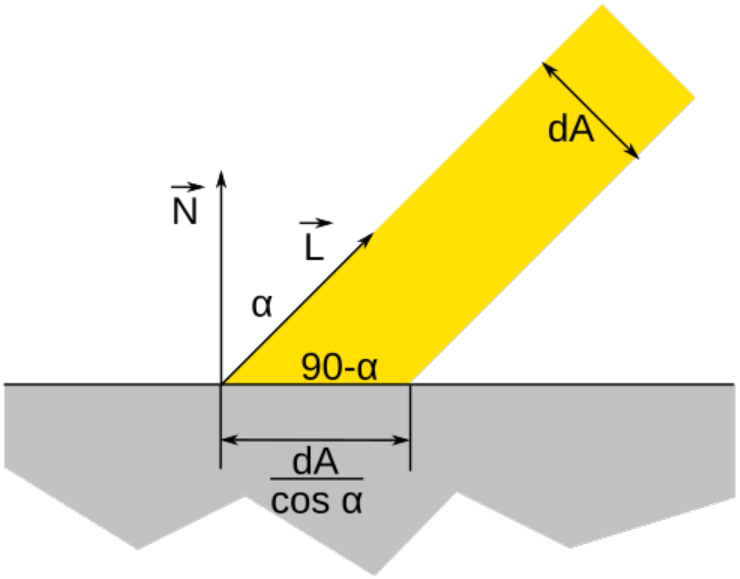


Figure 1: Lambert diffuse lighting based on angle between surface/edge direction and light direction

4.1 Lambert Diffuse Lighting

Lambert lighting models diffuse reflection using the cosine of the angle between a surface direction and light direction:

$$I = \max(0, \mathbf{n} \cdot \mathbf{l}) \tag{13}$$

4.2 Wireframe Edge Lighting

Since wireframes have no surface normals, edge direction is used instead:

$$I = \max(0, |\hat{\mathbf{e}} \cdot \hat{\mathbf{l}}|) \tag{14}$$

Multiple lights are supported by summing contributions:

$$I_{total} = \sum_{i=1}^{N} I_i \tag{15}$$

5 Animation System

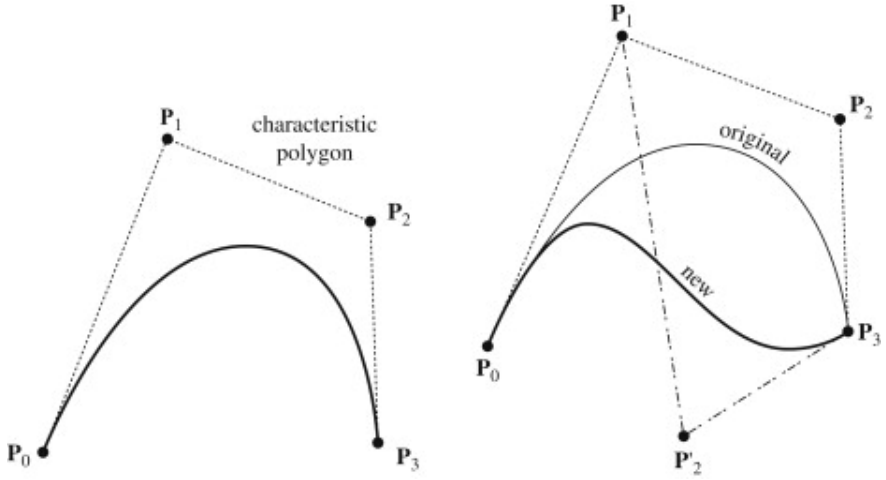


Figure 2: Cubic Bezier curve defined by four control points

5.1 Cubic Bezier Curves

Smooth animation is achieved using cubic Bezier curves defined by four control points:

$$B(t) = (1-t)^3 p_0 + 3(1-t)^2 t p_1 + 3(1-t)t^2 p_2 + t^3 p_3 \tag{16}$$

This ensures continuous position and velocity.

5.2 Looping and Synchronization

Time is wrapped into the interval $[0, 1]$ to ensure seamless looping. All animated objects share the same time parameter t, ensuring synchronized motion.

6 Quaternion Rotation

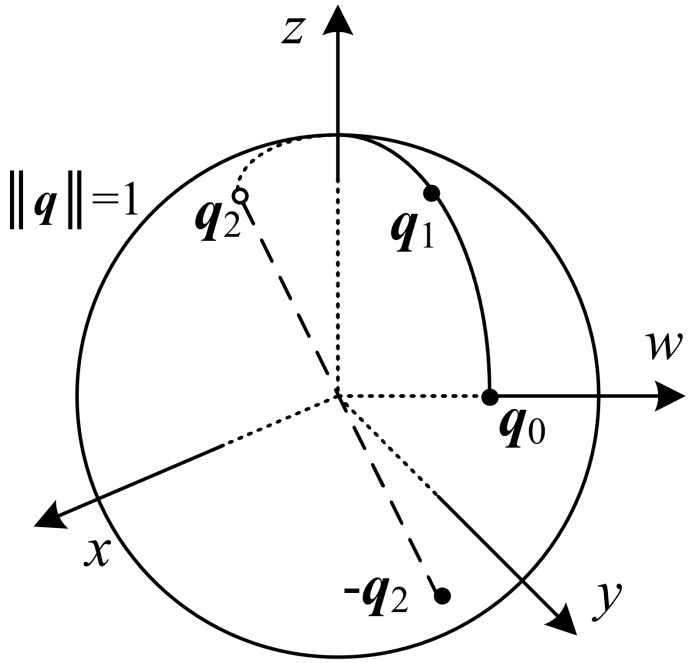


Figure 3: Quaternion-based rotation avoiding gimbal lock

Euler angle rotation suffers from gimbal lock. To avoid this, quaternions are used.

A rotation quaternion is defined as:

$$q = \left(\cos\frac{\theta}{2}, ; \hat{u}\sin\frac{\theta}{2}\right) \tag{17}$$

Rotation of a vector is performed using:

$$\mathbf{v}' = q, \mathbf{v}, q^{-1} \tag{18}$$

6.1 SLERP

Spherical Linear Interpolation provides smooth rotational motion:

$$\text{SLERP}(q_1, q_2, t) = \frac{\sin((1-t)\theta)}{\sin\theta} q_1 + \frac{\sin(t\theta)}{\sin\theta} q_2 \tag{19}$$

7 Demo and Results

The final demo renders multiple wireframe objects with Lambert lighting while animating them along Bezier paths. All objects move in sync and rotate smoothly using quaternion interpolation. The result is a visually coherent and mathematically accurate real-time 3D animation.

7.1 Bezier Animation Function

Listing 1: Cubic Bezier curve implementation

```
vec3_t bezier(vec3_t p0, vec3_t p1, vec3_t p2, vec3_t p3, float t)
{
    t = std::clamp(t, 0.0f, 1.0f);
    float u = 1.0f - t;

    return p0 * (u * u * u)
         + p1 * (3.0f * u * u * t)
         + p2 * (3.0f * u * t * t)
         + p3 * (t * t * t);
}
```

7.2 Lambert Edge Lighting

Listing 2: Lambert lighting for wireframe edges

```
float lambert_edge(vec3_t e, vec3_t l)
{
    e.normalize_fast();
    l.normalize_fast();

    return std::max(0.0f, std::fabs(dot(e, l)));
}
```

8 Conclusion

This project demonstrates that a full 3D rendering pipeline can be constructed using fundamental mathematics alone. By implementing vector algebra, matrix transformations, lighting models, and animation techniques manually, a deep understanding of real-time graphics systems is achieved. The techniques used here form the foundation of modern graphics engines and GPUs.

9 Further Improvements

9.1 Improved Animation and Camera Control

By adopting quaternions, both object animation and camera orientation can be enhanced. Smooth rotational animations can be achieved by interpolating between keyframe orientations using SLERP, resulting in more realistic motion compared to linear interpolation of Euler angles.

Additionally, quaternion-based camera control allows unrestricted free-look movement, making the system more suitable for real-time applications such as simulations and interactive visualizations.

—

9.2 Performance and Precision Considerations

Although quaternion operations introduce a small computational overhead, this cost is negligible compared to the benefits gained in stability and animation quality. Furthermore, quaternion normalization ensures rotation precision is maintained over long animation sequences, preventing error accumulation commonly observed with repeated matrix multiplications.