Pemanfaatan Quaternion dan Matriks Transformasi dalam Manipulasi Objek 3D di Godot

Ahmad Ibrahim - 13523089[1,2]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13523089@std.stei.itb.ac.id, [2]ahmaibrahim2020@gmail.com

***Abstract*—Makalah ini membahas pemanfaatan quaternion dan matriks transformasi dalam manipulasi objek 3D pada Godot, platform pengembangan game berbasis *open source*. Quaternion digunakan untuk meminimalkan masalah *gimbal lock* yang sering terjadi pada rotasi berbasis sudut euler, sementara matriks transformasi diterapkan untuk mengelola translasi, rotasi, dan skala objek secara efisien. Topik ini relevan bagi pengembang game dan aplikasi berbasis 3D dalam meningkatkan kualitas interaksi pada objek - objek 3D.**

***Keywords*—Godot Engine, Quaternion, 3D Transformation, Object Manipulation.**

I. PENDAHULUAN

Manipulasi objek 3D merupakan elemen fundamental dalam pengembangan aplikasi berbasis tiga dimensi, termasuk *game* dan simulasi virtual. Godot menyediakan berbagai alat yang memungkinkan pengembang untuk mengelola translasi, rotasi, dan skala objek dengan fleksibilitas tinggi. Salah satu fitur yang mendukung manipulasi tersebut adalah penggunaan quaternion dan matriks transformasi.

Quaternion menawarkan solusi efektif untuk mengatasi masalah pada rotasi berbasis sudut Euler, yaitu *gimbal lock.* Dalam Godot quaternion diimplementasikan melalui kelas Basis dan Quaternion, yang memungkinkan rotasi kontinu dengan presisi tinggi dan efisiensi komputasi. Di sisi lain, matriks transformasi digunakan secara luas dalam pengelolaan transformasi geometris, mencakup translasi, rotasi, dan skala, melalui struktur data seperti Transform3D.

Makalah ini bertujuan untuk mengeksplorasi dan mendemonstrasikan penerapan quaternion dan matriks transformasi dalam manipulasi objek 3D menggunakan Godot. Pendekatan ini dapat meningkatkan kualitas interaksi visual dan juga mendukung efisiensi pengelolaan objek 3D dalam proses yang kompleks. Dengan menggunakan referensi dari dokumentasi resmi Godot dan implementasi praktis di dalam *engine*, penelitian ini memberikan gambaran menyeluruh tentang keunggulan dan penerapan nyata di pendekatan tersebut.

Makalah ini terdiri dari beberapa bagian. Setelah pendahuluan ini, bagian dasar teori akan mengulas tentang dasar-dasar teori dan rumus. Selanjutnya, implementasi menjelaskan langkah implementasi quaternion dan matriks transformasi pada Godot. Bagian hasil dan pembahasan memaparkan percobaan, diakhiri dengan kesimpulan yang merangkum hasil penelitian serta peluang pengembangan lebih lanjut.

Referensi utama dalam makalah ini mencakup dokumentasi Godot dan repositori resmi di GitHub memastikan bahwa setiap langkah yang dijelaskan secara lebih akurat.

II. DASAR TEORI

A. Quaternion

Quaternion adalah representasi matematis yang digunakan untuk menggambarkan rotasi dalam ruang tiga dimensi. Secara struktural quaternion dapat dituliskan sebagai berikut:

$$q = a + bi + cj + dk$$

Dengan a adalah skalar dan $bi + cj + dk$ adalah vektor.

$$i^2 = j^2 = k^2 = ijk = -1$$

$$ij = k, jk = i, ki = j$$

$$ji = -k, kj = -i, ik = -j$$

Dalam hal ini, quaternion memiliki keunggulan signifikan dibandingkan representasi lain seperti sudut Euler, terutama karena kemampuannya untuk menghindari *gimbal lock. Gimbal lock* adalah kondisi ketika dua sumbu rotasi menjadi sejajar, yang menyebabkan hilangnya derajat kebebasan dan menimbulkan masalah dalam rotasi kompleks. Fenomena ini sering kali menghambat animasi atau manipulasi rotasi yang membutuhkan gerakan kontinu di ruang tiga dimensi.

Keunggulan quaternion terletak pada efisiensi komputasi, stabilitas dalam rotasi kontinu, dan kemampuannya untuk diinterpolasi dengan metode seperti spherical linear interpolation (SLERP). Hal ini membuat

quaternion sangat berguna dalam aplikasi 3D, termasuk game, simulasi, dan visualisasi. Penggunaan quaternion memungkinkan pengembang untuk mengelola rotasi kompleks dengan presisi tinggi, tanpa menghadapi keterbatasan yang umum pada representasi lainnya. Dalam Godot, quaternion diimplementasikan melalui kelas Quaternion, yang dirancang untuk memberikan solusi praktis bagi pengembang dalam mengelola rotasi objek 3D dengan presisi tinggi. Dokumentasi resmi Godot menjelaskan secara rinci fitur dan metode yang dapat digunakan untuk memanfaatkan quaternion secara maksimal.

B. Normalisasi Quaternion

Normalisasi Quaternion penting untuk memastikan bahwa quaternion tetap berada dalam ruang unit, yang diperlukan untuk menjaga keakuratan rotasi dalam aplikasi yang sensitif terhadap presisi matematis.

Rumus Normalisasi quaternion didapat dari:

$$||q|| = \sqrt{qq^*} = \sqrt{q^* q} = \sqrt{a^2 + b^2 + c^2 + d^2}$$

Dengan $q^* = a - bi - cj + dk$ yaitu konjugat dari quaternion.

C. Rotasi dengan quaternion

Vektor 3D yang direpresentasikan sebagai quaternion, dan adalah inversi quaternion. Inversi ini memungkinkan quaternion untuk secara efisien menerapkan rotasi balik, yang berguna dalam berbagai manipulasi 3D. Operasi ini banyak diterapkan dalam animasi karakter, simulasi fisika, dan bahkan pengendalian kamera.

D. Matriks Transformasi

Matriks transformasi adalah representasi linier yang digunakan untuk melakukan translasi, rotasi, dan skala objek dalam ruang tiga dimensi. Matriks ini biasanya dinyatakan dalam bentuk matriks 4x4 dalam representasi homogen, yang memungkinkan integrasi transformasi linier dan translasi dalam satu operasi tunggal. Matriks transformasi memberikan fleksibilitas dalam mengelola berbagai transformasi geometris secara terpadu. Dalam Godot Engine, struktur data seperti Transform3D dirancang untuk mempermudah manipulasi posisi, orientasi, dan skala objek, yang membuatnya menjadi komponen vital dalam pengelolaan objek 3D di dunia virtual.

Bentuk Umum Matriks Transformasi:

1. Matriks translasi

Matriks ini digunakan untuk memindahkan objek ke posisi baru dalam ruang tiga dimensi, seperti memindahkan karakter di dalam dunia game atau menggeser posisi kamera.

2. Matriks rotasi (misalnya, untuk sumbu)

Matriks rotasi memungkinkan pengelolaan rotasi objek di sekitar sumbu tertentu dengan sudut , yang berguna dalam berbagai aplikasi seperti rotasi roda kendaraan atau perputaran kamera di game.

3. Matriks skala

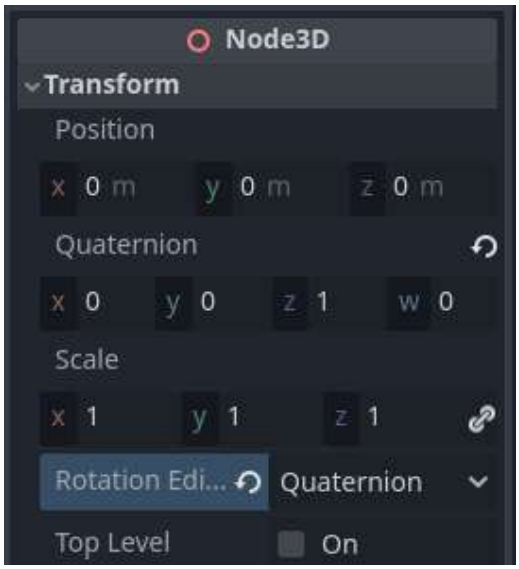
Matriks ini digunakan untuk mengubah ukuran objek dengan faktor pada masing-masing sumbu. Ini sering diterapkan dalam situasi di mana objek perlu diperbesar atau diperkecil secara dinamis.

Kombinasi dari translasi, rotasi, dan skala ini memungkinkan pengelolaan transformasi geometris yang kompleks dalam satu struktur data yang komprehensif. Hal ini memberikan fleksibilitas luar biasa dalam pengelolaan adegan tiga dimensi.

E. Implementasi di Godot Engine

Godot Engine menggunakan quaternion dan matriks transformasi sebagai bagian integral dari sistem manipulasi 3D. Fitur utama implementasi ini mencakup:

Fitur Quaternion digunakan untuk mengelola rotasi objek dengan efisien melalui metode seperti rotasi lokal dan global. Penggunaan quaternion memungkinkan rotasi yang lebih halus dan bebas dari masalah seperti gimbal lock, yang membuatnya ideal untuk aplikasi yang memerlukan animasi rotasi yang kompleks. Animasi seperti rotasi kamera dan gerakan robotik seringkali memanfaatkan keuntungan ini.



Gambar 2.1 Tampilan untuk transformasi menggunakan quaternion

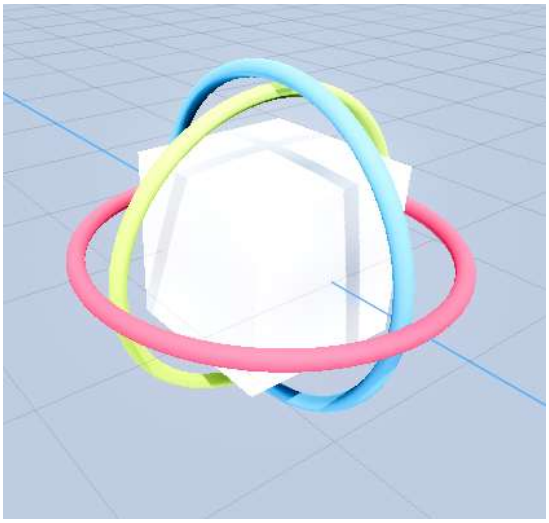
Struktur data Transform3D mengintegrasikan translasi, rotasi, dan skala dalam satu kerangka kerja, sehingga mempermudah pengelolaan transformasi kompleks dalam aplikasi 3D. Transform3D sering digunakan untuk mendefinisikan transformasi dunia objek dalam proyek Godot, termasuk kamera, cahaya, dan objek lain. Struktur ini mendukung pengelolaan transformasi hierarkis yang sering diterapkan dalam animasi kompleks dan skenario permainan.

Basis merupakan bagian dari Transform3D yang secara spesifik menangani rotasi dan skala dalam bentuk matriks 3x3. Basis memberikan fleksibilitas tambahan dalam mengelola orientasi dan dimensi objek, yang penting dalam simulasi fisika atau efek visual yang membutuhkan presisi tinggi.

Dengan dasar teori ini, quaternion dan matriks transformasi menjadi alat yang sangat penting dalam menyederhanakan manipulasi objek di ruang tiga dimensi. Implementasi yang baik dari kedua pendekatan ini memungkinkan pengembang untuk meningkatkan efisiensi, kualitas visual, dan interaksi pengguna dalam aplikasi berbasis Godot. Kombinasi matematis dan praktis ini menjadikan quaternion dan matriks transformasi sebagai elemen tak terpisahkan dalam pengembangan aplikasi 3D modern, memberikan fondasi yang kuat untuk menciptakan pengalaman pengguna yang imersif dan realistis.

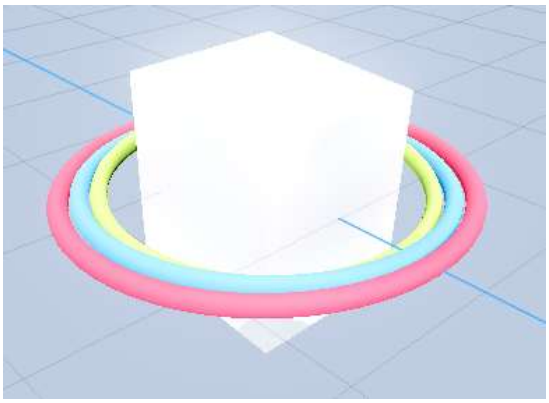
III. IMPLEMENTASI

Dalam pengembangan aplikasi berbasis tiga dimensi, transformasi rotasi memainkan peran yang sangat krusial dalam menentukan bagaimana suatu objek dapat bergerak dan berinteraksi di dalam ruang virtual. Dua pendekatan yang paling umum digunakan untuk mengelola rotasi adalah metode berbasis Euler angles dan quaternion. Kedua pendekatan ini memiliki karakteristik yang sangat berbeda, baik dalam hal kompleksitas implementasi, fleksibilitas penggunaannya, maupun keandalan hasil yang diperoleh dalam berbagai kasus praktis.



Gambar 3.1 Transformasi menggunakan Euler

Euler angles mendasarkan rotasi pada tiga sumbu utama, yaitu X, Y, dan Z. Dalam pendekatan ini, setiap rotasi dilakukan secara berurutan, yang berarti urutan pelaksanaan rotasi memiliki pengaruh langsung terhadap hasil akhir. Sebagai contoh, jika suatu objek pertama kali dirotasi pada sumbu X, lalu dilanjutkan pada sumbu Y, dan terakhir pada sumbu Z, orientasi akhirnya akan berbeda jika urutan rotasi ini diubah. Kompleksitas semakin bertambah dalam kasus transformasi rotasi yang melibatkan banyak langkah, di mana interaksi antar sumbu dapat mempersulit pengelolaan rotasi secara presisi. Sebagai ilustrasi, rotasi 45 derajat pada sumbu X akan mengubah sistem koordinat untuk rotasi berikutnya pada sumbu Y, sehingga pengembang harus mempertimbangkan perubahan ini dalam perhitungan mereka. Kesulitan ini sering kali menyebabkan kebingungan, terutama dalam proyek yang memerlukan banyak manipulasi rotasi secara bersamaan.



Gambar 3.2 Terjadi *Gimbal Lock* ketika menggunakan Euler

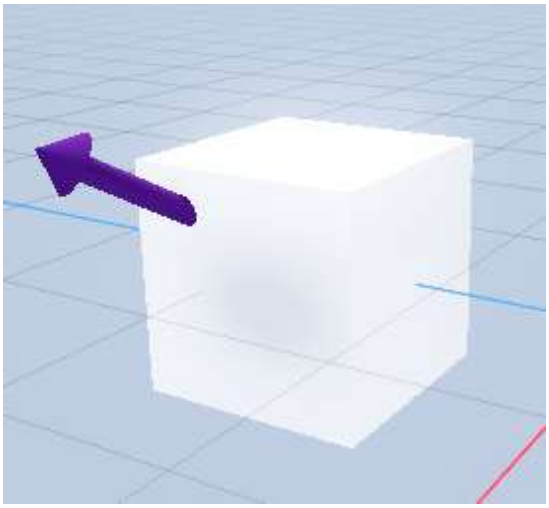
```
func _on_x_slider_value_changed(value):
    x.rotation_degrees.x = value

func _on_y_slider_value_changed(value):
    y.rotation_degrees.x = value - 90

func _on_z_slider_value_changed(value):
    z.rotation_degrees.z = value - 90
```

Gambar 3.3 Kode untuk mengubah sudut menggunakan Euler

Slider mengontrol rotasi pada sumbu tertentu, dengan nilai yang diterapkan langsung pada sumbu X untuk objek pertama (x), sedangkan sumbu X pada objek kedua (y) dan sumbu Z pada objek ketiga (z) dikurangi 90 derajat. Pengurangan 90 derajat dilakukan untuk menyesuaikan orientasi awal objek sehingga posisi "netral" pada slider mencerminkan orientasi yang diinginkan di dunia 3D, atau untuk mengatasi perbedaan sistem koordinat antara model dan engine. Namun, menggunakan rotasi berbasis Euler rentan terhadap *gimbal lock*, yang terjadi ketika dua sumbu rotasi menjadi sejajar, biasanya saat rotasi pada salah satu sumbu mencapai ±90 derajat. Dalam kondisi ini, rotasi pada salah satu sumbu kehilangan derajat kebebasan, menyebabkan rotasi yang tidak terprediksi atau tidak sesuai dengan kontrol slider. Masalah ini dapat

terjadi pada kode ini jika slider, misalnya, menyebabkan rotasi sumbu X atau Y mendekati ±90 derajat, membuat rotasi sumbu lain tidak lagi bekerja secara independen. Untuk menghindari *gimbal lock*, rotasi berbasis quaternion biasanya digunakan sebagai alternatif.

Quaternion menawarkan pendekatan yang jauh lebih intuitif dan efisien untuk rotasi. Dengan quaternion, rotasi didefinisikan oleh sumbu rotasi yang direpresentasikan sebagai vektor arah, serta sudut rotasi dalam satuan derajat atau radian. Keunggulan utama dari quaternion adalah independensinya terhadap urutan rotasi, sehingga hasil akhir selalu konsisten, terlepas dari kompleksitas langkah-langkah yang dilakukan. Selain itu, quaternion mengatasi masalah yang sering muncul pada Euler angles, yaitu *gimbal lock. Gimbal lock* terjadi ketika dua sumbu rotasi menjadi sejajar, menyebabkan hilangnya satu derajat kebebasan, yang pada gilirannya membatasi fleksibilitas rotasi. Masalah ini sangat umum terjadi pada animasi atau simulasi yang melibatkan rotasi kontinu atau kompleks.



Gambar 3.4 Transformasi menggunakan Quaternion

Dalam implementasi praktis, quaternion tidak hanya menawarkan fleksibilitas yang lebih tinggi tetapi juga mendukung interpolasi rotasi yang halus melalui metode seperti *spherical linear interpolation* (SLERP). Metode SLERP memungkinkan rotasi transisi yang lebih mulus dan realistis, yang sangat penting dalam pengembangan aplikasi seperti animasi karakter atau kontrol kamera. Tidak seperti Euler angles yang mengharuskan pengembang untuk memikirkan interaksi antar sumbu, quaternion memungkinkan pengelolaan rotasi hanya dengan mendefinisikan arah rotasi dan sudutnya, tanpa perlu mempertimbangkan urutan atau efek kumulatif dari rotasi sebelumnya.

```gdscript
func update_quat():
    if raw_ijk.length() > 0:
        ijk = raw_ijk.normalized()

        quat = Quaternion(ijk, deg_to_rad(raw_w))
        var rotation = Quaternion(Vector3.RIGHT, ijk)
        quat_basis.quaternion = rotation


func _on_rotate_pressed():
    if is_rotating:
        quat_box.quaternion = final_rotation
    is_rotating = true
    final_rotation = (quat * quat_box.quaternion).normalized()


func _process(delta):
    if is_rotating:
        var current_quat: Quaternion = quat_box.quaternion
        var interpolated_quat := current_quat.slerp(final_rotation, delta * rotation_speed)

        if interpolated_quat.angle_to(quat_box.quaternion) < 0.00001:
            is_rotating = false
            interpolated_quat = final_rotation
        quat_box.quaternion = interpolated_quat
```

Gambar 3.5 Kode untuk rotasi menggunakan quaternion

Kode di atas memperlihatkan dua fungsi utama dalam pengelolaan rotasi menggunakan quaternion. Fungsi `update_quat` bertanggung jawab untuk menghasilkan quaternion baru berdasarkan sumbu rotasi (`ijk`) dan sudut (`raw_w`). Quaternion ini kemudian digunakan untuk memutakhirkan orientasi pada basis rotasi objek.

Fungsi `_on_rotate_pressed` menangani inisiasi rotasi baru. Jika objek sedang dalam proses rotasi, fungsi ini akan menyelesaikan rotasi sebelumnya sebelum memulai rotasi baru. Rotasi baru dihitung dengan mengalikan quaternion awal dengan quaternion baru, kemudian dinormalisasi untuk memastikan rotasi tetap stabil.

```cpp
real_t Quaternion::length_squared() const {
    return dot(*this);
}

real_t Quaternion::length_squared() const {
    return dot(*this);
}

real_t Quaternion::length() const {
    return Math::sqrt(length_squared());
}

void Quaternion::normalize() {
    *this /= length();
}

Quaternion Quaternion::normalized() const {
    return *this / length();
}
```

Gambar 3.6 Kode untuk normalisasi yang diambil dari repository Godot

Kode tersebut merupakan implementasi fungsi dasar untuk operasi pada quaternion, yang sering digunakan dalam grafika komputer dan simulasi 3D. Fungsi `length_squared()` menghitung kuadrat panjang quaternion menggunakan dot product dirinya sendiri, sedangkan `length()` menghitung panjang quaternion dengan mengambil akar kuadrat dari `length_squared()`. Fungsi `normalize()` mengubah quaternion saat ini menjadi unit quaternion

(panjang = 1) dengan membagi setiap komponennya dengan panjangnya, sementara `normalized()` mengembalikan quaternion yang sudah dinormalisasi tanpa mengubah nilai quaternion asli.

```
Quaternion to1;
real_t omega, cosom, sinom, scale0, scale1;

// calc cosine
cosom = dot(p_to);

// adjust signs (if necessary)
if (cosom < 0.0f) {
    cosom = -cosom;
    to1 = -p_to;
} else {
    to1 = p_to;
}

// calculate coefficients

if ((1.0f - cosom) > (real_t)CMP_EPSILON) {
    // standard case (slerp)
    omega = Math::acos(cosom);
    sinom = Math::sin(omega);
    scale0 = Math::sin((1.0 - p_weight) * omega) / sinom;
    scale1 = Math::sin(p_weight * omega) / sinom;
} else {
    // "from" and "to" quaternions are very close
    //  ... so we can do a linear interpolation
    scale0 = 1.0f - p_weight;
    scale1 = p_weight;
}
// calculate final values
return Quaternion(
        scale0 * x + scale1 * to1.x,
        scale0 * y + scale1 * to1.y,
        scale0 * z + scale1 * to1.z,
        scale0 * w + scale1 * to1.w);
```

Gambar 3.7 Kode untuk SLERP yang diambil dari repositori Godot

Kode tersebut merupakan implementasi *Spherical Linear Interpolation* (SLERP) untuk quaternion, yang digunakan untuk interpolasi rotasi halus antara dua quaternion. Pertama, nilai dot product antara quaternion saat ini dan quaternion target (`p_to`) dihitung untuk menentukan seberapa mirip orientasinya. Jika hasilnya negatif, tanda dari quaternion target diubah untuk memastikan interpolasi terjadi pada jalur terpendek. Jika kedua quaternion memiliki perbedaan yang signifikan (nilai cosine kecil), interpolasi dihitung menggunakan pendekatan spherical interpolation, di mana sudut `omega` digunakan untuk menghitung koefisien berbasis sinus. Sebaliknya, jika kedua quaternion hampir identik (perbedaan sangat kecil), interpolasi linear dilakukan sebagai pendekatan sederhana untuk menghindari pembagian nol. Hasil akhirnya adalah quaternion baru yang dihasilkan dari kombinasi terinterpolasi kedua quaternion berdasarkan bobot `p_weight`.

Interpolasi rotasi dilakukan dalam fungsi `_process` menggunakan metode `slerp`. Metode ini menghasilkan transisi rotasi dari orientasi awal ke orientasi akhir. Ketika sudut antara quaternion interpolasi dan quaternion akhir berada di bawah ambang batas tertentu, rotasi dianggap selesai.

Dalam aplikasi nyata, translasi, rotasi, dan skala sering digabungkan untuk menciptakan transformasi yang kompleks. Kombinasi ini diwakili oleh matriks transformasi 3x4 yang mencakup ketiga jenis transformasi. Pada Godot Engine, properti Transform3D menyediakan alat yang efisien untuk mengelola kombinasi transformasi ini.

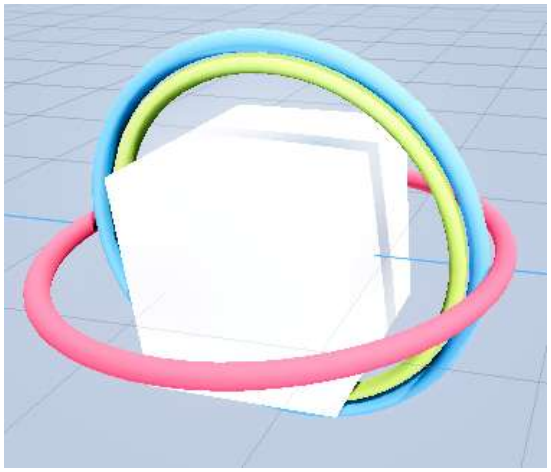
```
var input := Vector3.ZERO
input.x = Input.get_axis("move_left", "move_right")
input.z = Input.get_axis("move_forward", "move_back")

apply_central_force(twist_pivot.basis * input * 1200.0 * delta)

twist_pivot.rotate_y(twist_input)
pitch_pivot.rotate_x(pitch_input)
pitch_pivot.rotation.x = clamp(pitch_pivot.rotation.x, -0.5, 0.5)
twist_input = 0.0
pitch_input = 0.0
```

Gambar 3.8 Kode implementasi menggerakkan benda

Kode tersebut mengimplementasikan mekanisme kontrol gerakan dan rotasi dalam ruang 3D. Variabel input adalah vektor 3D yang digunakan untuk menangkap masukan gerakan horizontal (sumbu X) dan vertikal (sumbu Z) melalui tombol yang didefinisikan, seperti "move_left", "move_right", "move_forward", dan "move_back". Vektor ini kemudian dikalikan dengan basis transformasi twist_pivot, konstanta gaya (1200.0), dan delta waktu untuk memastikan gaya diterapkan secara proporsional terhadap waktu frame, lalu diteruskan ke fungsi `apply_central_force` untuk mendorong objek secara dinamis. Selanjutnya, rotasi diterapkan pada sumbu Y (rotate_y) untuk objek `twist_pivot` dan pada sumbu X (rotate_x) untuk objek `pitch_pivot`, di mana rotasi pada sumbu X dibatasi menggunakan fungsi clamp untuk menjaga sudut dalam rentang -0.5 hingga 0.5 radian, mencegah rotasi berlebih. Akhirnya, nilai `twist_input` dan `pitch_input` di-*reset* ke 0 untuk memastikan rotasi hanya terjadi berdasarkan input terbaru.

IV. HASIL DAN PEMBAHASAN

Setelah implementasi metode rotasi menggunakan Euler angles dan quaternion dalam aplikasi berbasis Godot, hasil eksperimen menunjukkan perbedaan signifikan antara kedua pendekatan tersebut dalam hal efisiensi, presisi, dan kemudahan penggunaan. Pada implementasi berbasis Euler angles, rotasi objek sering kali memerlukan perhitungan yang kompleks karena urutan rotasi yang berpengaruh terhadap hasil akhir. Selain itu, masalah *gimbal lock* menjadi hambatan besar, terutama ketika rotasi melibatkan sudut yang signifikan di beberapa sumbu secara bersamaan. Contoh kasus menunjukkan bahwa untuk mencapai rotasi tertentu, pengembang harus memahami dengan baik interaksi antar sumbu, yang sering kali mengarah pada kesalahan interpretasi dan perhitungan tambahan.



Gambar 4.1 Terjadi *gimbal lock* pada sumbu x ketika menggunakan Euler

Sebaliknya, implementasi quaternion menghasilkan hasil yang lebih konsisten dan mudah diatur. Dengan mendefinisikan sumbu rotasi sebagai vektor arah dan sudut rotasi, pengembang dapat dengan mudah mengontrol orientasi objek tanpa memikirkan urutan rotasi. Hasil eksperimen menunjukkan bahwa quaternion tidak hanya mengatasi masalah *gimbal lock* tetapi juga memungkinkan interpolasi rotasi yang sangat halus melalui metode seperti `slerp`. Dalam animasi karakter, misalnya, rotasi yang kompleks dapat dilakukan dengan mulus, memberikan pengalaman visual yang lebih realistis kepada pengguna.

Dari segi performa, quaternion juga menunjukkan efisiensi yang lebih tinggi. Perhitungan quaternion menunjukkan penggunaan sumber daya yang lebih ringan dibandingkan dengan *Euler angles*, terutama ketika rotasi melibatkan banyak langkah atau interpolasi yang sering dilakukan dalam aplikasi berbasis animasi atau simulasi. Selain itu, kemampuan quaternion untuk dinormalisasi memastikan stabilitas rotasi bahkan dalam skenario yang kompleks.

Selain itu, translasi dan skala juga menunjukkan hasil yang efisien dalam eksperimen. Translasi memberikan kemampuan untuk memindahkan objek secara presisi ke lokasi yang diinginkan, sementara skala memungkinkan manipulasi ukuran objek untuk menciptakan efek visual yang diinginkan. Kombinasi ketiga transformasi ini, ketika diimplementasikan dengan baik, memberikan fleksibilitas yang besar dalam pengelolaan adegan 3D.

V. KESIMPULAN

Hasil penelitian ini menunjukkan bahwa quaternion adalah pendekatan yang lebih unggul untuk pengelolaan rotasi dalam aplikasi berbasis tiga dimensi, terutama dalam kasus yang melibatkan kompleksitas tinggi dan kebutuhan akan transisi visual yang halus. Dibandingkan dengan Euler angles, quaternion menawarkan solusi yang lebih intuitif, efisien, dan bebas dari masalah seperti *gimbal lock*. Dengan mendukung interpolasi rotasi yang mulus, quaternion memungkinkan pengembang untuk menciptakan pengalaman pengguna yang lebih realistis dan imersif.

Dalam pembahasan, dapat disimpulkan bahwa quaternion lebih unggul dibandingkan *Euler angles* dalam berbagai aspek, terutama pada aplikasi yang membutuhkan rotasi kontinu, kompleksitas rendah, dan transisi visual yang halus. Meskipun Euler angles masih relevan untuk kasus sederhana, seperti animasi berbasis sumbu tunggal, penggunaan quaternion memberikan keuntungan yang signifikan dalam hal fleksibilitas, presisi, dan efisiensi.

Selain itu, translasi dan skala juga berperan penting dalam pengelolaan transformasi objek 3D. Translasi memberikan fleksibilitas dalam memindahkan objek ke posisi yang diinginkan, sementara skala memungkinkan perubahan ukuran objek secara dinamis untuk berbagai kebutuhan visual. Kombinasi ketiga transformasi ini memberikan dasar yang kuat untuk pengembangan aplikasi berbasis tiga dimensi yang efisien dan fleksibel.

VI. UCAPAN TERIMA KASIH

Penulis mengucapkan syukur kepada Tuhan Ynag Maha Esa atas rahmat dan karunia-Nya sehingga makalah ini dapat diselesaikan dengan baik. Ucapan terima kasih juga penulis sampaikan kepada dosen mata kuliah Aljabar Linear & Geometri IF2123. yaitu Bapak Rinaldi Munir, Bapak Rila Mandala, dan Bapak Arrival Dwi Sentosa, atas ilmu dan materi yang telah diberikan selama ini. Selain itu penulis juga mengucapkan terima kasih kepada seluruh pembaca makalah ini, dengan harapan semoga isi makalah ini dapat memberikan manfaat sesuai dengan tujuan penulis.

REFERENSI

[1] Munir, Rinaldi. Aljabar Quaternion (Bagian I), Institut Teknologi Bandung (ITB), 2023 [Online], https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2023-2024/Algeo-25-Aljabar-Quaternion-Bagian1-2023.pdf, diakses pada 27 Desember 2024.

[2] Munir, Rinaldi. Aljabar Quaternion (Bagian II). Institut Teknologi Bandung (ITB). 2023 [Online], https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2023-2024/Algeo-26-Aljabar-Quaternion-Bagian2-2023.pdf, diakses pada 27 Desember 2024.

[3] https://www.youtube.com/watch?v=Ri2xIhcii8I diakses pada 27 Desember 2024

[4] https://github.com/godotengine/godot diakses pada 27 Desember 2024

[5] https://docs.godotengine.org/en/stable/classes/class_transform3d.html diakses pada 27 Desember 2024

[6] https://docs.godotengine.org/en/stable/classes/class_quaternion.html diakses pada 27 Desember 2024

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 27 Desember 2024

ttd

Ahmad Ibrahim 13523089