Aplikasi Quartenion pada Game Engine

Ali Akbar - 13514080
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
aliakbr@students.itb.ac.id

***Abstrak*—*Game engine* adalah mesin pembuat game baik untuk grafis 2D atau grafis 3D. Dalam membuat *game* 3D pastinya kita sering melihat gerakan-gerakan rotasi. Rotasi dapat dikomputasikan dengan mentransformasikan sebuah titik dengan matriks transformasi. Namun *cost* komputasi untuk menginterpolasikan gerak dengan matriks transformasi cukup berat. Sehingga Quarternion saat ini dipakai oleh *game engine* untuk membuat rotasi pada *game* 3D karena biaya komputasinya yang ramah. Quarternion diaplikasikan dalam internal *game engine* ataupun penyediaan struktur Quarternion.**

Kata Kunci*— quartenion, rotasi, interpolasi quartenion, *game engine.

I. PENDAHULUAN

Game dengan grafis 3D dan gerak 3D sudah menjadi trend untuk game masa kini. Game dengan grafis dan gerak 3D memberikan nuansa yang terlihat nyata bagi para pemain. Contoh game-game berbasis grafis 3D saat ini seperti Tomb Raider, Counter Strike, Assasin Creeds, dan lain sebagainya. Game-game tersebut dibuat dalam sebuah mesin pembuat game yang disebut *game engine*.

Game engine atau mesin pembuat game saat ini begitu banyak jenisnya, ada Unreal games, Unity, Construct, dan lain sebagainya. Selain itu game-game dibuat dalam berbagai platform mulai dari PC hingga telepon genggam pintar atau *smartphone* semua mulai bisa menjalankan *game* dengan grafis dan gerak 3D. Namun apa yang dibuat sekarang adalah hasil pemanfaatan dari teori yang lama. Jaman dulu sekitar tahun 1990an pembuatan game cukup sulit (terutama game grafis 3D) karena *game engine* dan perangkat keras jaman dulu tidak secanggih sekarang. Permasalahan beberapa *game maker* jaman dulu saat membuat game dengan grafis dan gerak 3D adalah "bagaimana caranya membuat gerak karakter dan kamera secara halus?"

Seperti yang kita tau gerak pada bidang 2 Dimensi dapat dianggap sebagai transformasi linear dari sebuah titik. Sehingga terdapat matriks transformasi yang nantinya akan memetakan titik atau komponen tersebut sesuai dengan matriks transformasinya. Dalam gerak rotasi kita tau bahwa untuk bidang datar (2 dimensi) kita dapat mengalikan vector kolom dari koordinat sumbu x dan sumbu y pada bidang kartesius dengan matriks rotasi berorde 2x2 yaitu

$$\begin{bmatrix} \cos x & -\sin x \\ \sin x & \cos x \end{bmatrix}$$

Nantinya untuk bidang tiga dimensi akan ada matriks rotasi dasar yang akan membentuk matriks rotasi tertentu. Jika kita perhatikan operasi untuk melakukan rotasi pada sebuah komponen itu cukup mudah namun ketika dikomputasikan akan memakan banyak operasi sehingga program akan menjadi berat. Apabila diimplementasikan pada game gerakan rotasi akan terkesan "patah" akibat banyaknya komputasi dan interpolasi yang tidak cukup baik.

Pada 16 Oktober 1843, Sir William Rowan Hamilton ketika sedang dalam perjalanan menuju Royal British Academy dengan istrinya dan melewati Royal Canal dari Jembatan Brougham menemukan sebuah persamaan dasar perkalian quartenion secara tiba-tiba dan mengukir persaman tersebut ke sebuah batu. Persamaannya adalah

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

Persamaan diatas adalah dasar untuk teori quartenion berkembang dan populer di matematika aljabar geometri[1].

Dengan memanfaatkan quartenion seperti rotasi, interpolasi quartenion dan sebagainya akhirnya muncullah animasi berbasis 3D yang memiliki gerakan kamera dinamis seiring dengan pergerakan karakter yang menjadi dasar game-game 3D seperti game Tomb Raider, Resident Evil, dan sebagainya. Namun bagaimana cara kerja quartenion ini dan implementasinya dalam *game engine* ?

Pada makalah ini saya akan membahas bagaimana pengaplikasian quartenion pada *game engine*. Untuk makalah ini saya menggunakan *game engine* Unity untuk mencoba gerakan rotasi dengan quartenion ini.

II. LANDASAN TEORI

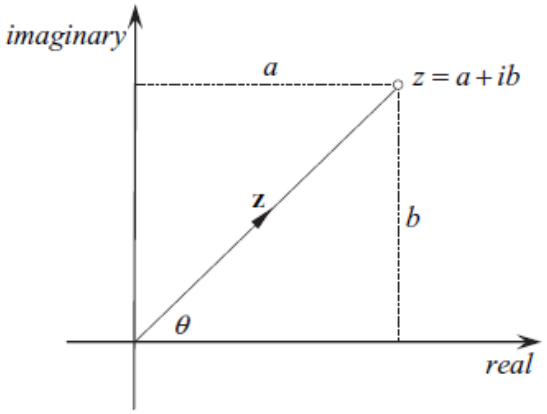
A. Bilangan Kompleks

Dalam operasi-operasi aljabar kerap kali kita menemukan kendala ketika kita menemukan adanya akar dari sebuah bilangan negatif. Akhirnya seorang matematikawan Girolamo Cardano dan Rafael Bombelli

menemukan cara untuk memanfaatkan akar negatif dalam aljabar. Caranya adalah dengan membuat sebuah bilangan imajiner yaitu simbol *i* yang merepresentasikan √-1 sehingga 2 + √-5 dapat direpresentasikan sebagai 2 + √5 i. Bentuk dasar bilangan kompleks adalah a + bi dengan a adalah bilangan real dan bi yang merupakan bilangan imajiner

Bilangan kompleks sama halnya dengan aljabar sehingga dapat dioperasikan secara aritmatik. Seperti misalnya pertambahan dan pengurangan dari 2 bilangan kompleks yaitu $(a_1 + a_2i) \pm (b_1 + b_2i) = (a_1 \pm b_1) + (a_2 \pm b_2)i$. Untuk perkalian dua bilangan kompleks juga masih mirip seperti dengan perkalian aljabar biasa. Namun harus diingat bahwa $i^2 = -1$. Untuk pembagian dua bilangan kompleks, cari akar sekawan penyebut (denumerator) sehingga penyebut dari pembagian bilangan kompleks tersebut dapat menjadi skalar setelah numerator dan denumerator pembagian keduanya dikalikan dengan akar sekawan tadi.

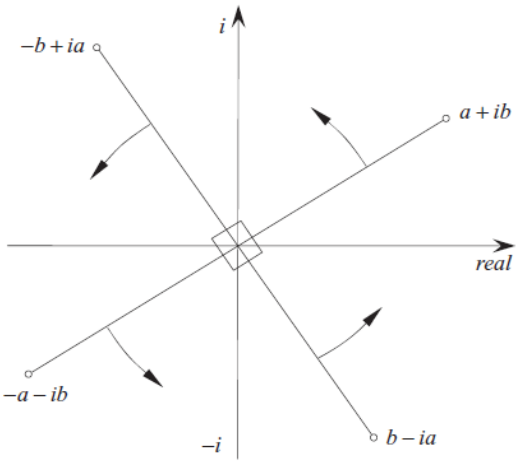
Sebuah bilangan kompleks dapat dinyatakan dalam sebuah bidang. Representasi bilangan kompleks dapat dinyatakan dengan Diagram Argand yang terdiri dari sumbu tegak (sumbu y pada bidang kartesius) yang menyatakan nilai bilangan imajiner dan sumbu datar (sumbu x pada bidang kartesius) menyatakan bilangan real dari bilangan kompleks. Dibawah ini adalah gambar Diagram Argand untuk z = a + ib



Diambil dari [3] pada halaman 16

Dengan nilai θ = arc(tan(a/b)) dan besar $|z| = \sqrt{a^2 + b^2}$.

Jika kita perhatikan dengan Diagram Argand kita dapat mengganti bentuk bilangan kompleks a dan b dengan a = |z| cos θ dan b = |z| sin θ. Selain itu bilangan kompleks dapat menjadi sebuah rotor. Misal kita bilangan kompleks a + ib maka kita dapat memutar a + ib 90 derajat hanya dengan mengalikan a + ib dengan i. Jika kita kalikan i sebanyak 4x maka kita dapatkan putaran penuh 360 derajat. Hal ini membuktikan bahwa bilangan kompleks dapat merepresentasikan rotor.



Gambar diambil dari [3] pada halaman 17

Rotor dalam bidang 2D dapat kita buat dalam fungsi q = cos(θ) + sin(θ)i yang nantinya jika dikalikan dengan bilangan kompleks lain maka bilangan tersebut akan berotasi sebesar θ dengan θ adalah sudut dari lokasi bilangan kompleks sebelumnya (anggap a) ke hasil rotasi a'.

Dengan dasar-dasar ini quartenion dibentuk untuk menyelesaikan permasalahan yang lebih kompleks.

B. Aljabar Quartenion

Pada konsep quartenion, terjadi pengembangan bilangan kompleks yang awalnya dapat direpresentasikan pada ruang 2 Dimensi dan sekarang akan dikembangkan pada ruang dimensi 3. Sehingga bilangan kompleks harus berbentuk z = a + bi + cj dengan i dan j adalah bilangan imajiner lainya. Namun jika 2 bilangan kompleks dikalikan akan terjadi masalah bagaimana caranya merumuskan ij. Hamilton lalu menemukan sebuah bentuk dan teorema perkalian baru. Bentuk bilangan kompleks z berubah menjadi z = s + ai + bj + ck dengan s adalah scalar dan ai + bj + ck adalah komponen imajiner. Pada prinsip perkalian 2 bilangan kompleks terdapat aturan yang dibuat Hamilton. Formula yang ia tetapkan adalah

Teorema(1) :

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

Teorema(2) :

$$ij = k \quad jk = i \quad ki = j \quad ji = -k \quad kj = -i \quad ik = -j$$

Dengan memanfaatkan dua tetapan di atas maka perkalian antara 2 bilangan kompleks dalam bentuk z = s + ai + bj + ck akan selalu memberikan bentuk yang sama pula.

Bentuk bilangan kompleks dalam 3 bilangan imajiner inilah yang disebut sebagai quartenion. Bentuk tersebut juga dapat direpresentasikan dalam bentuk z = s + **v** dimana **v** (vector v) adalah komponen yang terdapat bilangan imajiiner dari bilangan kompleks z. Komponen bilangan imajiner pada bilangan kompleks dapat direpresentasikan dalam vector dan berlaku hukum-hukum aljabar vector di dalamnya (seperti perkalian dot, perkalian cross, dan lain sebagainya). Representasi lain

dari bilangan kompleks z adalah sebuah z dapat dinyatakan sebagai pasangan terurut sebuah skalar s dan vector **v** sehingga z = [s,**v**].

Karena sifat dari komponen bilangan imajiner quartenion berlaku operasi-operasi aritmatik dan operasi pada aljabar vector. Terdapat pula beberapa property pada quartenion ini. Misal kita punya $z_1 = s_1 + a_1i + b_1i + c_1k$ dan $z_2 = s_2 + a_2i + b_2i + c_2k$ (yang akan kita pakai terus pada pembahasan di bawah) maka ada beberapa property yang berlaku

a. Pertambahan
   Pertambahan pada quartenion sama halnya dengan pertambahan pada bilangan kompleks sehingga
   $z_1 + z_2 = (s_1 \pm s_2) + (a_1 \pm a_2)i. + (b_1 \pm b_2)j + (c_1 \pm c_2)k$
b. Perkalian
   Jika kita kalikan dua bilangan quartenion di atas maka jika kita turunkan maka akan didapat :

   $z_1z_2 = a_1a_2 + ia_1b_2 + ja_1c_2 + ka_1d_2 + ib_1a_2 + i^2b_1b_2 + ijb_1c_2 + ikb_1d_2 + jc_1a_2 + jic_1b_2 + j^2c_1c_2 + jkc_1d_2 + kd_1a_2 + kid_1b_2 + kjd_1c_2 + k_2d_1d_2.$

   Dengan memanfaatkan teeorema(1) dan (2) maka didapat

   $$z_1z_2 = s_1s_2 - v_1v_2 + s_1v_2 + s_2v_1 + v_1x\,v_2$$

c. Panjang quartenion
   Panjang dari quartenion adalah akar dari seluruh bilangan riil kuadrat (termasuk koefisien bilangan imajiner i,j,k) pada sebuah quartenion. Atau dapat dirumuskan :

   $$||z_1|| = \sqrt{s_1^2 + a_1^2 + b_1^2 + c_1^2}$$

d. Quartenion unit
   Quartenion unit adalah quartenion yang dibentuk dari perkalian antara 1/ $||z_1||$ dengan quartenion $||z_1||$

   $$\hat{z}_1 = \frac{1}{||z_1||} z_1$$

e. Quartenion Murni
   Quartenion dengan skalar 0 disebut sebagai quartenion murni.
f. Konjugasi Quartenion
   Konjugasi quartenion adalah quartenion yang bentuk nya diubah dari z = [s,**v**] menjadi z' = [s,-**v**], z' disini adalah quartenion konjugasi z. (bentuk formalnya adalah z diikuti dengan *upperbar* diatasnya).
g. Invers Quartenion
   Misal ada quartenion z dengan z = [s,**v**] maka rumus invers quartenion z dapat dinyatakan dengan

   $$z^{-1} = \frac{\bar{z}}{||z||^2}$$

   $\bar{z}$ *adalah quartenion konjugasi*

h. Dot Product pada quartenion
   Jika kita anggap pasangan z = [s,**v**] sebagai sebuah larik dari vector maka
   $z_1.z_2 = s_1s_2 + a_1\,a_2\,b_1b_2 + c_1c_2.$
   Selain itu juga berlaku
   $z_1.z_2 = ||z_1||.||z_2||\cos\theta$

Berikutnya kita akan membahas bagaimana rotasi dapat dibentuk pada quartenion dengan memanfaatkan property property diatas. Untuk membuat rotasi maka kita harus memilki sebuah rotor. Jika pada bidang 2D rotor didefinisikan sebagai q = cos(θ) + sin(θ)i maka pada bidang kompleks 3D kita menggunakan rotor

$$q = \cos\left(\frac{1}{2}\theta\right) + \sin(\frac{1}{2}\theta)\hat{u}$$

Dengan u adalah unit quartenion yang memiliki panjang satu dan θ adalah sudut rotasi yang akan dibentuk. Formula untuk membentuk sebuah quartenion terotasi dengan rotor q normalnya adalah

$$p' = qpq^{-1}$$

dengan p adalah quartenion murni dari titik yang ingin dirotasi, q adalah rotor, p' adalah quartenion hasil rotasi dan q^{-1} adalah inversti rotor q. Invers dari rotor q adalah

$$q^{-1} = \cos\left(\frac{1}{2}\theta\right) - \sin(\frac{1}{2}\theta)\hat{u}$$

p adalah quartenion murni dari titik P. Misal P memiliki titik P(x,y,z) maka nantinya p = xi + yj +zk. Hasil rotasi adalah quartenion murni yang mengandung koordinat x,y,z setelah dirotasi dengan rotor q.

Selain sifat-sifat di atas ada beberapa axioma pada quatenion, jika kita tau $q_1, q_2, q_3, q_4 \ni C$ diantaranya :

a. Closure
   $q_1 + q_2 \ni C$, $q_1q_2 \ni C$
b. Identitas
   Untuk q quartenion maka q + 0 = 0 + q = q dan (1)q = (1)q = 1.
c. Invers
   $q^{-1}q = qq^{-1} = 1$
d. Asosiatifitas
   Hukum asosiatif quartenion berlaku pada pertamabahan dan perkalian.
   $q_1 + (q_2 + q_3) = (q_1 + q_2) + q_3$
   $q_1(q_2\,q_3 = q_1(q_2\,q_3)$
e. Komutatifitas
   Sifat komutatif hanya berlaku untuk pertambahan namun tidak untuk perkalian.
   $q_1 + q_2 = q_2 + q_1$
   $q_2\,q_3 \neq q_3\,q_2$
f. Distributifitas
   $q_1(q_2 + q_3) = q_1q_2 + q_1q_3$
   $(q_2 + q_3)\,q_1 = q_2\,q_1 + q_3\,q_1$

C. Interpolasi Quartenion

Hal mendasar mengapa quartenion dipakai untuk grafis 3D adalah kita memiliki beberapa cara untuk melakukan interpolasi rotasi pada quartenion. Interpolasi rotasi digunakan agar rotasi berjalan seiring berjalannya waktu sesuai dengan yang kita inginkan. Dengan interpolasi quartenion rotasi pada grafis computer akan semakin halus. Ada beberapa cara yang digunakan untuk menginterpolasikan quartenion. Diantaranya SQAD (Spherical and Quadrangle) dan SLERP (Spherical Linear Interpolation).

a. SLERP

SLERP menggunakan penurunan dari interpolasi standar menjadi bentuk interpolasi umum untuk quartenion. Bentuk umum dari hasil penurunan dari interpolasi linearnya adalah :

$$q' = q_1(q_1^{-1}q_2)^t$$

q_1 adalah orientasi pertama (kuartenion titik awal) dan q_2 adalah orientasi kedua (kuartenion titik akhir) dan q' adalah titik hasil interpolasi tiap waktu dengan t adalah satuan waktu. Namun bentuk umum diatas pada praktiknya jarang diimplementasikan karena menggunakan bentuk eksponensial. Bentuk lain yang digunakan adalah dengan menggunakan interpolasi speris dengan rumus :

$$q_t = \frac{\sin(1-t)\theta}{\sin(\theta)} q_1 + \frac{\sin t\theta}{\sin\theta} q_2$$

θ dapat dicari dengan prinsip dot product pada q_1 dan q_2.

Formula diatas memiliki beberapa kelemahan. Diantaranya adalah hasil negatif dari dot product antara 2 quartenion akan menyebabkan fungsi menjadi mengambil jalan terpanjang untuk mencapai tujuan. Hal ini dapat diatasi dengan dalam komputasinya mutlakkan nilai dari dot product sehingga jalan yang diambil adalah jalan yang paling dekat. Selain itu apabila jarak antara orientasi pertama dan kedua terlalu dekat maka nilai dari θ sangat kecil dan mengakibatkan nilai sin θ menjadi 0 (fungsi menjadi tak terdefinisi). Apabila ini terjadi maka harus digunakan interpolasi linear bentuk umum untuk menyelesaikannya[1].

b. SQAD

SQAD memanfaatkan SLERP untuk membuat jalan pada rotasi menjadi halus dengan membuat sebuah quartenion bantuan (s_i) dari suatu quartenion ke i dari deret quartenion n dan selanjutnya dibuat fungsi *nested slerp* pada parameter fungsi SQAD. Jika misalnya kita punya deret quartenion $q_1, q_{2,\ldots}, q_{n-1}, q_n$ maka quartenion bantuannya dapat dirumus kan sebagai berikut

$$s_i = \exp\left(-\frac{\log\left(q_{i+1}q_i^{-1}\right) + \log\left(q_{i-1}q_i^{-1}\right)}{4}\right)q_i$$

Orientasi dari sub kurva dari deret di atas dapat dinyatakan sebagai $q_{i-1}, q_i, q_{i+1}, q_{i+2}$. Fungsi interpolasi dengan sqad adalah sebagai berikut

$$sqad(q_i, q_{i+1}, s_i, s_{i+1}, t) = slerp(slerp(q_i, q_{i+1}, t), slerp(s_i, s_{i+1}, t), 2t(1-t))$$

III. IMPLEMENTASI QUARTENION PADA BEBERAPA GAME ENGINE

Dalam implementasinya quartenion digunakan oleh *game engine* untuk membuat rotasi. Misalnya gerakan planet memutari orbitnya, atau gerakan sebuah objek mengikuti gerak input dari user. Banyak dari *game engine* saat ini telah memberikan class Quartenion yang di dalamnya telah terdapat method-method untuk membentuk quartenion (konstruktor dan operator dasar) dan fungsi SLERP untuk interpolasi quartenion. Salah satu *game engine* tersebut adalah Unity.

Unity sebenarnya secara internal telah menggunakan quartenion sebagai dasar untuk rotasi. Namun karena quartenion cukup sulit dipahami karena ada 3 bilangan kompleks akhirnya Unity membuat class Quartenion dimana walaupun kita tidak perlu tau koordinat xyz nya kita tetap dapat menginterpolasikan 2 rotasi (rotasi pada struktur transform unity strukturnya sama dengan class Quartenion yang berarti rotasi adalah quartenion juga pada Unity). Unity diprogram dengan bahasa C# atau Javascript. Script dimasukkan ke dalam sebuah objek sehingga objek tersebut bergerak sesuai dengan script yang ada. Kelas Quartenion yang telah disediakan oleh Unity dan di dalamnya terdapat beberapa method-method seperti lerp (linear interpolation), slerp, dan lain sebagainya. Berikut adalah contoh source code yang diberikan oleh Unity untuk tutorial penggunaan class Quartenion :

```
public class LookAtScript : MonoBehaviour
{
    public Transform target;


    void Update ()
    {
        Vector3 relativePos = target.position - transform.position;
        transform.rotation = Quaternion.LookRotation(relativePos);
    }
}
```

(source code ini diambil dari [2] ditulis dalam C#)

LookRotation akan membuat sebuah rotasi dari sebuah objek dengan vector gerakan relative terhadap target. Anggap misalnya A (bola) adalah target dan B (orang) adalah objek yang diisi dengan script di atas maka objek A akan berputar mengikuti gerak objek B (target), apabila

objek B bergerak maka ia akan selalu melihat kearah A. Hal ini terjadi karena Quartenion.LookRotation akan membuat quartenion rotation dengan vector posisi relative terhadap jarak objek A dan B lalu quartenion hasil rotasi dimasukkan kedalam rotasi objek tersebut sehingga objek berotasi. Screen shot contoh gerakan dari script di atas adalah sebagai berikut

Gambar diambil dari potongan video [2]

Contoh script lain yang diberikan oleh unity sebagai contoh adalah sebagai berikut :

```
using UnityEngine;
using System.Collections;

public class GravityScript : MonoBehaviour
{
    public Transform target;


    void Update ()
    {
        Vector3 relativePos = (target.position + new
Vector3(0, 1.5f, 0)) - transform.position;
        Quaternion rotation =
Quaternion.LookRotation(relativePos);

        Quaternion current = transform.localRotation;

        transform.localRotation =
Quaternion.Slerp(current, rotation, Time.deltaTime);
        transform.Translate(0, 0, 3 * Time.deltaTime);
    }
}
```

(source code ini diambil dari [2] ditulis dalam C#)

Script di atas jika di attach ke sebuah objek misalnya B dan target A maka akan membuat B berputar dengan poros adalah A. Poros dibuat dengan LookRotation function pada Quartenion dan di simpan di rotation (karena kita ingin rotasi berporos pada A). Sementara gerakan B dibuat dari fungsi Slerp. Slerp menerima 3 buah parameter seperti yang kita tau yaitu quartenion awal (di script ditulis sebagai lokasi awal objek atau current) lalu quartenion akhir (disini adalah target rotasi dari B) dan waktu interpolasi. Setelah dikomputasikan dengan menggunakan metode SLERP gerakan/rotasi ditranslasi pada objek dalam bentuk arah dan jarak dengan fungsi Translate (rotasi pada Unity juga merupakan quartenion).

Game engine lain yang menyediakan kelas Quartenion adalah Unreal Engine. Hampir sama dengan Unity struktur Quartenionnya (di Unreal adalah FQuat) juga memiliki fungsi slerp dan operator-operator quartenion dasar. Perbedaannya dalah Unreal Engine ditulis dalam C++ dan banyak fungsi dari struktur ini yang masih memerlukan pengetahuan tentang quartenion sementara pada Unity fungsi-fungsi dibuat agar user tetap bisa menggunakannya walaupun belum mengerti Quartenion sekalipun. Secara internal Unreal juga menggunakan Quartenion dalam membuat rotasi.

IV. KESIMPULAN

Quartenion dapat membantu kita dalam membuat sebuah gerak rotasi pada grafis computer dengan sangat mulus. Pada pemafaatannya dalam membuat *game* 3D, banyak dari *game engine* secara internal telah memasukkan quartenion dalam rotasi, namun *game engine* masih menyediakan struktur kelas quartenion yang di dalamnya telah terdapat beberapa method seperti SLERP dan method-method operasi dasar pada quartenion untuk kebutuhan *user*. Kita sebenarnya dapat membentuk quartenion ini dengan membuat struktur data baru dan nantinya memiliki operasi-operasi dasar quartenion lalu memanfaatkan interpolasi quartenion agar simulasi gerakan menjadi lebih halus. Namun kita jangan lupa untuk menghindari kelemahan-kelemahan dalam interpolasi quartenion seperti menghindari kasus dot product negatif dan jarak antar quartenion yang kecil sehingga hasil interpolasi menjadi tak terdefinisi menggunakan interpolasi sperikal.

V. UCAPAN TERIMAKASIH

Penulis mengucapkan puji dan syukur kepada Allah SWT karena berkat rahmatnya penulis dapat menyelesaikan tulisan ini. Penulis juga berterimakasih kepada dosen pengajar Aljabar Geometri Bapak Ir.Rinaldi Munir dan Drs.Judhi Santoso, M.Sc karena berkat pengajarannya penulis memiliki pengetahuan tentang quartenion, aljabar vector, dan bilangan kompleks yang menjadi dasar pokok bahasan disini.

Penulis juga berterima kasih atas dukungan teman-teman dalam penyelesaian tulisan ini.

REFERENCES

[1] http://www.3dgep.com/understanding-quaternions/ (diakses pada 15 Desember 2015)

[2] https://unity3d.com/learn/tutorials/modules/intermediate/scripting/quaternions (diakses pada 15 Desember 2015)

[3] Vince, John. 2008. *Geometric Algebra for Computer Graphics.* Springer : London.

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, 8 Desember 2015

Ali Akbar, 13514080