# Implementasi Quaternion dalam Slerp (Spherical Linear Interpolation)

Alson Cahyadi 13514035
*Program Studi Informatika*
*Sekolah Teknik Elektro dan Informatika*
*Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia*
*alson.cahyadi@students.itb.ac.id*

***Abstract*—Animasi pada masa yang serba digital ini sudah mengalami banyak kemajuan. Bukan hanya animasi dua dimensi, tetapi juga animasi tiga dimensi sudah dipakai secara luas oleh banyak hal yang mengandung elemen animasi seperti film-film animasi, game-game tiga dimensi, serta piranti-piranti lunak berbasis tiga dimensi lainnya. Banyak orang yang tidak mengerti hal apa yang membuat objek-objek dalam animasi dapat bergerak maju, mundur, berputar, serta gerakan-gerakan lainnya. Dengan makalah ini, diharapkan banyak orang lebih mengerti tentang implementasi quaternion dalam Slerp dan kegunaannya pada piranti-piranti animasi 3D yang ada di pasaran.**

***Keywords*—3D, Animasi, Slerp, Quaternion**

## I. PENDAHULUAN

Dewasa ini teknologi animasi telah berkembang pesat. Animasi-animasi yang dulu hanya berbasis dua dimensi (2D) sekarang sudah mulai ditinggalkan, berkat piranti-piranti lunak yang membolehkan para animator untuk membuat animasi yang berbasis tiga dimensi (3D). Animasi dalam ruang tiga dimensi terkesan lebih hidup dan lebih nyata karena kemampuannya untuk lebih menyerupai dunia yang kita tinggali sekarang.

Meski terlihat mudah dan simpel, banyak sekali perhitungan-perhitngan kompleks yang harus dilakukan agar dapat menciptakan gerakan-gerakan yang dilakukan oleh objek-objek di dalam animas tiga dimensi tersebut. Salah satu cara untuk mengimplementasikan rotasi dalam animasi 3D adalah menggunakan matriks transformasi.

Masalahnya, dengan data matriks yang relatif banyak dan keterbatasan kecepatan komputasi, rotasi yang dihasilkan tidak semulus yang diinginkan. Oleh karena itu, para ilmuwan mencari cara agar rotasi dapat dilakukan dengan mulus dan dengan data yang lebih sedikit.

Maka, seorang ilmuwan jenius bernama *Ken Shoemake* menggagaskan penggunaan interpolasi quartenion untuk menganimasikan perputara tiga dimensi, yang merujuk kepada pergerakan sepanjang dengan kecepatan konstan sepanjang bujur lingkaran besar, dengan syarat parameter interpolasi yang ada di antar 0 dan 1. Gagasannya ini dinamakan Slerp (*s*pherical *l*inear int*erp*olation).

## II. BILANGAN KOMPLEKS

Bilangan kompleks adalah bilangan yang berbentuk

$$a + bi$$

dimana a dan b adalah bilangan riil, dan I adalah bilangan imajiner. Bilangan imajiner (*i*) ini memiliki sifat $i^2 = -1$. Bilangan *a* disebut dngan bilangan riil sedangkan *b* disebut sebagai bilangan imajiner. Dengan begitu, $2 + 4i$ adalah bilangan kompleks dengan 2 sebagai bilangan riil dan 4 sebagai bilangan imajinernya.

Operasi matematis (tambah, kurang, kali, bagi) dari bilangan imajiner memiliki property yang sama seperti bilangan riil, dengan beberapa sifat tambahan yang menarik.

Bilangan kompleks juga dapat didefinisikan berdasarkan bentuk polar sbb:

$$r = \sqrt{a^2 + b^2}$$

dengan

$$\theta = \arctan(\frac{b}{a})$$

maka

$$a + bi = r(\cos\theta + i\sin\theta)$$

Selain bentuk polar, bilangan kompleks juga dapat direpresentasikan dalam bentuk eksponen yaitu:

$$re^{i\theta} = r(\cos\theta + i\sin\theta)$$

## III. QUATERNION

Quaternion adalah sistem angka yang mengekstensi bilangan kompleks. Quartenion didefinisikan oleh matematikawan William Rowan Hamilton pada tahun 1843. Quartenion juga merupakan perpanjangan teori bilangan kompleks dari ruang dua dimensi kepada ruang tiga dimensi

Quartenion didefnisikan dalam bentuk:

$$z = a + ib + jc + kd.$$

dengan definisi simetri berikut:

$$z_1z_2 = s_1s_2 - v_1 \cdot v_2 + s_1 v_2 + s_2 v_1 + v_1 \times v_2$$

Objek ini disebut dengan '*quartenion*' dan bagian imajinernya dinamai '*vektor*'.

*Dot product* dari v1 dan v2 (v1 . v2) didefinisikan sbb:

$$b_1b_2 + c_1c_2 + d_1d_2$$

Dan menghasilkan bilangan scalar, yang menghasilkan definisi berikut:

$$\mathbf{v}_1 \cdot \mathbf{v}_2 = \|\mathbf{v}_1\|\|\mathbf{v}_2\| \cos\theta$$

Sedangkan *cross product* (v1 x v2) didefinisikan sbb:

$$i(c_1d_2 - d_1c_2) + j(d_1b_2 - b_1d_2) + k(b_1c_2 - c_1b_2)$$

Dimana hasil dari cross product adalah sebuah vektor v3, dan

$$\|\mathbf{v}_3\| = \|\mathbf{v}_1\|\|\mathbf{v}_2\| \sin\theta$$

### A. Penambahan Quartenion

Seperti vektor, quartenion dapat ditambah atau dikurangi seperti berikut:

$$q_1 = s_1 + ix_1 + jy_1 + kz_1$$

$$q_2 = s_2 + ix_2 + jy_2 + kz_2$$

$$q_1 \pm q_2 = [(s_1 \pm s_2) + i(x_1 \pm x_2) + j(y_1 \pm y_2) + k(z_1 \pm z_2)].$$

### B. Perkalian Quartenion

Jika ada dua quartenion:

$$q_1 = s_1 + \mathbf{v}_1 = s_1 + ix_1 + jy_1 + kz_1$$

$$q_2 = s_2 + \mathbf{v}_2 = s_2 + ix_2 + jy_2 + kz_2$$

maka perkalian kedua quartenion tsb adalah sbb:

$$q_1q_2 = s_1s_2 - \mathbf{v}_1 \cdot \mathbf{v}_2 + s_1\mathbf{v}_2 + s_2\mathbf{v}_1 + \mathbf{v}_1 \times \mathbf{v}_2$$

Perlu dicatat bahwa perkalian dua quartenion anti komutatif sehingga q1q2 tidak sama dengan q2q1.

### C. Besar Quartenion

Besar quartenion adalah sbb:

$$\|q\| = \sqrt{s^2 + x^2 + y^2 + z^2}.$$

### D. Unit Quartenion

Untuk quartenion q

$$q = 1 + i2 + j3 + k4$$

besarnya adalah

$$\|q\| = \sqrt{1^2 + 2^2 + 3^2 + 4^2} = \sqrt{30}$$

yang berarti unit quartenionnya adalah

$$\hat{q} = \frac{1}{30}(1 + i2 + j3 + k4).$$

### E. Pure Quartenion

Quartenion dengan besar scalar nol adalah *pure quartenion* seperti:

$$q_1 = ix_1 + jy_1 + kz_1 \text{ and } q_2 = ix_2 + jy_2 + kz_2$$

### F. Konjugasi Quartenion

Untuk quartenion

$$q = s + \mathbf{v}$$

$$q = s + ix + jy + kz$$

konjugasinya adalah

$$\overline{q} = s - \mathbf{v} = s - (ix + jy + kz).$$

### G. Quartenion Inverse

Inverse dari suatu quartenion didefinisikan sebagai persamaan

$$q^{-1} = \frac{\overline{q}}{\|q\|^2}.$$

dimana

$$qq^{-1} = 1$$

dan bersifat komutatif

$$qq^{-1} = q^{-1}q$$

### H. Properti Algebra Quartenion

1. Closure

   Untuk semua $q_1$ dan $q_2$,

$$\text{addition} \quad q_1 + q_2 \in \mathbb{C}$$
$$\text{multiplication} \quad q_1 q_2 \in \mathbb{C}.$$

2. Identitas
   Untuk setiap $q$ ada elemen identitas 0 dan 1 sebagaimana sehingga:
$$\text{addition} \quad q + 0 = 0 + q = q \ (0 = 0 + i0 + j0 + k0)$$
$$\text{multiplication} \quad q(1) = (1)q = q \ (1 = 1 + i0 + j0 + k0).$$

3. Inverse
   Untuk setiap $q$ ada elemen inerse $-q$ dan $q^{-1}$ sebagaimana sehingga:
$$\text{addition} \quad q + (-q) = -q + q = 0$$
$$\text{multiplication} \quad qq^{-1} = q^{-1}q = 1 \ (q \neq 0).$$

4. Asisiatif
   Untuk setiap $q_1, q_2$ dan $q_3$
$$\text{addition} \quad q_1 + (q_2 + q_3) = (q_1 + q_2) + q_3$$
$$\text{multiplication} \quad q_1(q_2 q_3) = (q_1 q_2) q_3.$$

5. Komutatif
   Untuk setiap $q_1$ dan $q_2$
$$\text{addition} \quad q_1 + q_2 = q_2 + q_1$$
$$\text{multiplication} \quad q_1 q_2 \neq q_2 q_1.$$

6. Distributif
   Untuk setiap $q_1, q_2$ dan $q_3$
$$q_1(q_2 + q_3) = q_1 q_2 + q_1 q_3$$
$$(q_1 + q_2) q_3 = q_1 q_3 + q_2 q_3.$$

## IV. SLERP

Slerp adalah singkatan untuk *S*pherical *L*inear Int*er*polation. Slerp menyediakan metode untuk meninterpolasi sebuah titik antara dua orientasi dengan lembut. [1]

Misalkan orientasi pertama adalah $q1$ dan orientasi kedua adalah $q2$. Titik yang diinterpolasi direpresentasikan sebagai $P$ dan titik yang diinterpolasi direpresentasikan dengan $p'$. Parameter interpolasi $t$ akan menginterpolasi $P$ dari $q1$ saat $t = 0$ sampai $q2$ saat $t = 1$. Maka, rumu standar interpolasi lanjar adalah:

$$\mathbf{p}' = \mathbf{p_1} + t(\mathbf{p_2} - \mathbf{p_1})$$

Langkah-langkah untuk mengaplikasikan persamaannya adalah:

- Hitung selisih antara $p1$ dan $p2$
- Ambil bilangan tak bulat dari selisih tsb
- Atur nilai asli oleh beda bilangan tak bulat antara dua titik

### *A. Selisih Quartenion*

Langkah pertama mengharuskan kita untuk menghitung selisih antara $q1$ dan $q2$. Hal ini ekuivalen dengan menghitung selisih sudut antara dua quartenion

$$= q_1^{-1} q_2$$

### *B. Eksponensial pada Quartenion*

Langkah selanjutnya adalah mengambil bagian fraksional dari selisih itu. Kita dapat menghitung bagian fraksional suatu quartenion dengan meningkatkan pangkatnya sampai nilainya ada di antara 0… 1.

Rumus umum untuk eksponensasi quartenion adalah:

$$q^t = \exp(t \log q)$$

Di mana fungsi eksponensial dari quartenion adalah:

$$\begin{aligned} \exp(q) &= \exp([0, \theta\hat{\mathbf{v}}]) \\ &= [\cos\theta, \sin\theta\hat{\mathbf{v}}] \end{aligned}$$

Dan logaritma quartenion adalah:

$$\begin{aligned} \log q &= \log(\cos\theta + \sin\theta\hat{\mathbf{v}}) \\ &= \log(\exp(\theta\hat{\mathbf{v}})) \\ &= \theta\hat{\mathbf{v}} \\ &= [0, \theta\hat{\mathbf{v}}] \end{aligned}$$

Untuk *t=0*, kita dapat:

$$\begin{aligned} q^0 &= \exp(0 \log q) \\ &= \exp([\cos(0), \sin(0)\hat{\mathbf{v}}]) \\ &= \exp([1, \mathbf{0}]) \\ &= [1, \mathbf{0}] \end{aligned}$$

Dan untuk *t=1*, kita dapat:

$$\begin{aligned} q^1 &= \exp(\log q) \\ &= q \end{aligned}$$

### *C. Selisih Fraksional pada Quartenion*

Untuk menghitung rotasi angular interpolasi, kita atur orientasi awal $q1$ dan mengaturnya berdasarkan bagian fraksional dari selisih antara $q1$ dan $q2$.

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

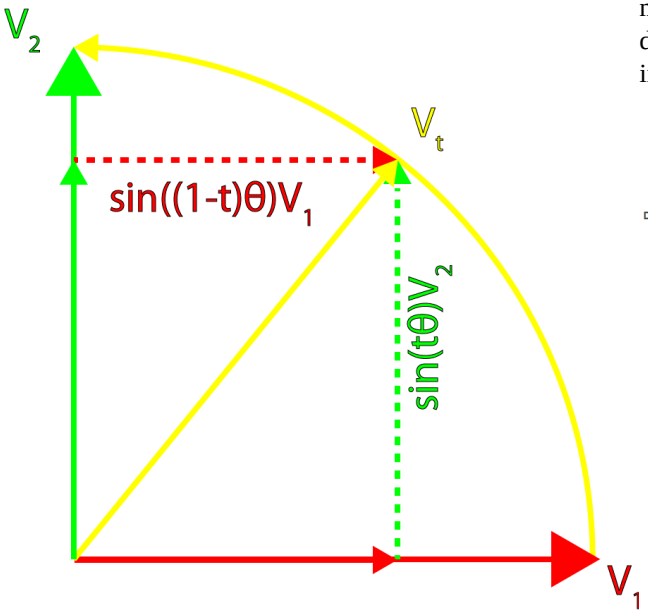
Dimana ini merupakan bentuk umum dari Slerp dengan menggunakan quaternion. Meskipun begitu, rumus tsb bukanlah bentuk dari rumus *slerp* yang umum dipakai pada prakteknya.

Kita dapat mengaplikasikan rumus yang serupa untuk

melakukan *spherical interpolation* dari vektor ke quaternion. Bentuk umum dari *spherical interpolation* untuk vektor didefinisikan sebagai:

$$\mathbf{v}_t = \frac{\sin(1-t)\theta}{\sin\theta}\mathbf{v}_1 + \frac{\sin t\theta}{\sin\theta}\mathbf{v}_2$$

Di mana divisualisasikan ke dalama gambar berikut:




*Gambar 4.3.1*, visualisasi Slerp
Sumber: http://www.3dgep.com/understanding-quaternions/#SLERP

Rumus ini dapat diaplikasikan secara langsung untuk quartenion:

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

Maka kita bisa mendapatkan sudut $\theta$ dengan menghitung dot product antara $q1$ dan $q2$

$$\cos\theta = \frac{q_1 \cdot q_2}{|q_1||q_2|}$$

$$= \frac{s_1 s_2 + x_1 x_2 + y_1 y_2 + z_1 z_2}{|q_1||q_2|}$$

$$\theta = \cos^{-1}\left(\frac{s_1 s_2 + x_1 x_2 + y_1 y_2 + z_1 z_2}{|q_1||q_2|}\right)$$

Meskipun begitu, ada dua masalah dengan implementasi ini yang harus diperhatikan saat diimplementasikan.

Pertama, bila *dot product* dari quartenion menghasilkan angka negative, maka hasil interpolasi akan bergerak mengelilingi bola (*sphere*), yang tentu saja bukan apa yang kita mau. Untuk menyelesaikan masalah ini, kita harus memeriksa hasil dari *dot product*, dan bila hasilnya negative, maka kita dapat menegasikan salah satu orientasi. Menegasikan bagian scalar dan vektor tidak mengubah orientasi yang direpresentasikan, tetapi dengan begitu kita menjamin rotasi dilakukan dengan jalan terpendek.

Masalah lain timbul ketika selisih angular antara $q1$ dan $q2$ sangat kecil sehingga $\sin\theta$ menjadi 0. Bila ini terjadi, maka akan didapatkan hasil yang tak terdefinisi saat dibagi dengan $\sin\theta$. Dalam kasus ini, dapat digunakan interpolasi lanjar antara $q1$ dan $q2$ saja.

### *D. Aplikasi Slerp dalam Pemrograman*

**Definition**

| | |
|---|---|
| Visual Basic | Public Shared Function **Slerp**( _<br>ByVal *q1* As Quaternion, _<br>ByVal *q2* As Quaternion, _<br>ByVal *t* As Single _<br>) As Quaternion |
| C# | public static Quaternion **Slerp**(<br>Quaternion *q1*,<br>Quaternion *q2*,<br>float *t*<br>); |
| C++ | public:<br>static Quaternion **Slerp**(<br>Quaternion *q1*,<br>Quaternion *q2*,<br>float *t*<br>); |
| JScript | public static function **Slerp**(<br>*q1* : Quaternion,<br>*q2* : Quaternion,<br>*t* : float<br>) : Quaternion; |

**Parameters**

| | |
|---|---|
| *q1* | Microsoft.DirectX.Quaternion<br>Source Quaternion structure. |
| *q2* | Microsoft.DirectX.Quaternion<br>Source Quaternion structure. |
| *t* | System.Single<br>A Single value that indicates how far to interpolate between the quaternions. |

*Gambar 4.4.1*, implementasi slerp pada beberapa kode pemrograman
Sumber: https://msdn.microsoft.com/en-us/library/windows/desktop/bb281655(v=vs.85).aspx

Slerp sudah diimplementasikan sebagai prosedur atau fungsi di dalam bahasa-bahasa pemrograman. Gambar 4.4.1 menunjukkan implementasi slerp dalam bentuk fungsi pada bahasa pemrograma Visual Basic, C++, C# dan JavaScript.

```
quat slerp(quat qa, quat qb, double t) {
          // quaternion to return
          quat qm = new quat();
          // Calculate angle between them.
          double cosHalfTheta = qa.w * qb.w + qa.x *
qb.x + qa.y * qb.y + qa.z * qb.z;
          // if qa=qb or qa=-qb then theta = 0 and we
can return qa
          if (abs(cosHalfTheta) >= 1.0){
                    qm.w = qa.w;qm.x = qa.x;qm.y =
qa.y;qm.z = qa.z;
                    return qm;
          }
          // Calculate temporary values.
          double halfTheta = acos(cosHalfTheta);
          double sinHalfTheta = sqrt(1.0 -
cosHalfTheta*cosHalfTheta);
          // if theta = 180 degrees then result is not
fully defined
          // we could rotate around any axis normal to
qa or qb
          if (fabs(sinHalfTheta) < 0.001){ // fabs is
floating point absolute
                    qm.w = (qa.w * 0.5 + qb.w * 0.5);
                    qm.x = (qa.x * 0.5 + qb.x * 0.5);
                    qm.y = (qa.y * 0.5 + qb.y * 0.5);
                    qm.z = (qa.z * 0.5 + qb.z * 0.5);
                    return qm;
          }
          double ratioA = sin((1 - t) * halfTheta) /
sinHalfTheta;
          double ratioB = sin(t * halfTheta) /
sinHalfTheta;
          //calculate Quaternion.
          qm.w = (qa.w * ratioA + qb.w * ratioB);
          qm.x = (qa.x * ratioA + qb.x * ratioB);
          qm.y = (qa.y * ratioA + qb.y * ratioB);
          qm.z = (qa.z * ratioA + qb.z * ratioB);
          return qm;
}
//Persamaan 4.4.1
//Sumber:
http://www.euclideanspace.com/maths/algebra/realNormedA
lgebra/quaternions/slerp/
```

Contoh implementasi fungsi Slerp pada pemrograman dapat dilihat pada contoh kode C++ yang menghasilkan quaternion antara dua quaternion yang bergantung pada variable t, seperti yang terlihat di persamaan 4.4.1. Jika t=0 maka qm = qa, dan jika t=1 maka qm=qb. Jika t ada di antara 0 dan 1 maka qm akan diinterpolasi di antaranya.

## V. KESIMPULAN

Penemuan metode Slerp ini menjadi dasar rotasi di animasi tiga dimensi yang sangat penting. Slerp dapat diimplementasikan dalam rotasi objek pada animasi tiga dimensi, dan sudah banyak dipakai di banyak bahas pemrograman seperti C++, C#, JavaScript, Visual Basic, OpenGL, dan lain-lain.

## VI. UCAPAN TERIMAKASIH

Penulis mengucapkan terimakasih kepada Tuhan YME yang dengan berkat dan anugerah-Nya penulis dapat meneyelesaikan makalah Aljabar Geometri ini. Penulis juga mengucapkan terimakasih kepada Drs. Judhi Santoso, M.Sc. selaku dosen mata kuliah Aljabar Geometri untuk semua pelajaran yang telah diberikan, terutama pelajaran quartenion yang menjadi dasar penulisan makalah ini.

## REFERENCES

[1] J. Vince, *Geometric Algebra for Computer Graphics.* London: Springer, Februari 2008

[2] J. Vince, *Quartenions for Computer Graphics*. 1st ed. London: Springer, 2011

[3] Dunn, F. and Parberry, I, *3D Math Primer for Graphics and Game Development.* 1st ed. Plano, Texas: Wordware Publishing, Inc, 2002

[4] http://www.3dgep.com/understanding-quaternions/#SLERP Diakses 14 Desember 2015 pada 15.00 WIB

[5] http://www.euclideanspace.com/maths/algebra/realNormedAlgebra/quaternions/slerp/ Diakses 14 Desember 2015 pada 15.07 WIB

## 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, 16 Desember 2015

Alson Cahyadi
13514035