Abstrak

Dalam tugas akhir ini telah dibuat sebuah API (*Application Programming Interface*) untuk aljabar geometrik dalam bahasa Java. API dibuat portabel sehingga bisa dipakai pada multiplatform dan tidak bergantung pada jenis sistem operasi. API berisi himpunan tipe data baru bernilai multivektor pada ruang vektor 2D dan 3D yang dibuat dengan berorientasi objek sehingga mudah dipakai ulang sebagai pustaka dasar pengembangan aplikasi-aplikasi berbasis aljabar geometrik. Selain operasi dasar dan perkalian geometrik dari multivektor, disediakan juga operasi refleksi, rotasi serta proyeksi.

Pada tugas akhir ini juga diperkenalkan wavelet multivektor sebagai perluasan konsep wavelet pada ruang vektor. Wavelet sebagai fungsi yang terlokalisasi dalam ruang(waktu) dan frekuensi sangat berguna untuk merepresentasikan fungsi maupun data dan menjadi basis tak bersyarat pada pelbagai ruang fungsi. Dengan menggunakan skema lifting yang mempunyai keunggulan sebgai wavelet generasi kedua dan memanfaatkan sifat linier dari operasi lifting dan multivektor maka wavelet multivektor dapat dibentuk.

API yang dibuat telah diuji dengan fungsi-fungsi tes yang diberikan untuk meningkatkan keandalan. Fungsi-fungsi tes tersebut sekaligus menjadi contoh penggunaan API. Unjuk kerja API dalam hal rotasi pada ruang dibandingkan dengan perhitungan yang memakai matriks rotasi. Hasil yang diperoleh adalah representasi rotasi yang lebih cepat untuk dihitung dan lebih sederhana dalam memrogram dibandingkan jika dinyatakan dalam bentuk matriks.

Prakata

Alhamdulillahirabbil 'alamin. Segala puji hanyalah milik Alloh penguasa semesta alam. Penulis ingin menyampaikan rasa terima kasih kepada:

1. Mamah dan Papah yang selalu 'memaksa' penulis untuk segera menyelesaikan tugas akhirnya.

2. Bapak Ir. Hermawan K. Dipojono, MSEE. PhD. atas bimbingan, perhatian dan kepercayaan sepenuhnya dari beliau. Semoga Alloh selalu memberkahi semua ilmu dan rizkinya.

3. Bapak Ir. Ahmad Nuruddin, MS. PhD. atas dorongan moril serta apresiasi beliau terhadap karya penulis. Semoga Alloh selalu melindungi dan merahmatinya.

4. Teman sekamar penulis, Umar Khatab, atas semua bantuannya dalam pengerjaan tugas akhir ini, termasuk pengeditan draft dan dokumentasi Javadoc. Semoga bisa lulus Juni 2001.

5. Teman-teman di Lab. Bahan, khususnya Leonardus Bimo Bayuaji yang selalu siap membantu dan mendukung kapanpun penulis butuhkan. Senang bekerja dengan kalian semua.

6. Adik-adikku Adhia dan Fauzan, adik-adikku di Pondok Panglayungan serta teman-temanku di BIOTER atas perhatian dan doanya.

7. Kepada semua orang yang paper-papernya banyak memberikan khasanah pengetahuan yang berharga untuk penulis pelajari.

Semoga Alloh melindungi kita dari ilmu-ilmu yang tak bermanfaat dan semoga Alloh memudahkan kita untuk mengamalkan semua ilmu yang telah kita peroleh.

Bandung, Januari 2001

Penulis

Ya Rabbku, berilah aku ilham untuk tetap mensyukuri ni'mat-Mu yang telah Engkau anugrahkan kepadaku dan kepada dua orang ibu bapakku dan untuk mengerjakan amal saleh yang Engkau ridhoi; dan masukkanlah aku dengan rahmat-Mu ke dalam golongan hamba-hamba-Mu yang saleh. (*An-Naml 27:19*)

Pembuatan API Aljabar Geometrik yang portabel dan mudah dikembangkan berbasis Java

Ginanjar Utama
Teknik Fisika ITB
Institut Teknologi Bandung

24 Januari 2001

Daftar Isi

Daftar Gambar

Daftar Tabel

Bab 1

Pendahuluan

1.1 Latar Belakang

Telah lama diketahui orang bahwa banyak fenomena fisis yang dapat dijelaskan secara tepat dengan model-model matematik. Persamaan yang menghubungkan berbagai kuantitas fisis merumuskan ide-ide dan konsep fisika yang dapat memprediksikan keluaran eksperimen. Guna menurunkan solusi dari persamaan tersebut dan mencari tahu apa yang sebenarnya terjadi, alat bantu yang digunakan adalah berbagai macam kakas matematik yang seringkali melibatkan abstraksi dan argumentasi. Abstraksi dan argumentasi ini seringkali berurusan dengan kuantitas-kuantitas yang tidak mempunyai relevansi fisis secara langsung.

Tugas akhir ini kami mencoba menggunakan aljabar geometrik dan wavelet untuk merepresentasikan data, fungsi atau sinyal yang melukiskan sistem fisis atau bersifat fisis. Pemilihan ini berdasarkan keinginan untuk mendapatkan hasil yang dapat dibayangkan dan berhubungan langsung dengan *observabel* fisis. Penurunan solusi dengan alat-alat di atas diharapkan dapat membawa interpretasi fisis yang memberikan pemahaman yang lebih mendalam tentang fenomena yang terjadi. Banyak kuantitas fisis yang mempunyai sifat geometrik, sehingga secara alami dapat dilukiskan dalam bentuk kuantitas geometrik yang sesuai. Objek-objek fisis seperti partikel atau medan, kebanyakan terlokalisasi dalam ruang-waktu maupun frekuensi, bahkan energi pun terkuantisasi dalam satuan unit $\hbar$, sehingga wavelet meru-

geometri koordinat	kakulus spinor
analisis kompleks	aljabar Grassman
analisis vektor	kalkulus Berezin
analisis tensor	bentukan differensial
aljabar Lie	twistor
aljabar Clifford	

Tabel 1.1: Beberapa sistem aljabar yang digunakan dalam fisika modern [1].

pakan solusi yang alami untuk merepresentasikan objek-objek tersebut.

Penggunaan bahasa Java dalam tugas akhir ini adalah karena alasan-alasan portabilitas, berorientasi objek murni dan kemudahan pendokumentasian. Dan besar harapan kami agar ada orang lain yang bisa menggabungkan API yang sudah ada dengan API yang akan kami buat.

1.2 Tujuan

1. Memperkenalkan aljabar geometrik sebagai sebuah kakas matematika modern dan bahasa yang universal dalam bidang interdisiplin.

2. Membuat API (*Application Programming Interface*) aljabar geometrik yang mudah dipakai, dirawat dan dikembangkan dalam bahasa Java. API ini mudah dipakai oleh mereka yang sudah paham aljabar geometrik dasar dan mengerti bahasa pemrograman berorientasi objek. Kodenya yang mudah dibaca dan sesuai dengan pola baku (*pattern*) dalam pemrograman berorientasi objek membuat modifikasi dan pencarian kesalahan mudah. Orang-orang yang tertarik untuk menerapkan aljabar geometrik dalam bidangnya dapat menggunakan kode-kode tersebut tanpa harus menulis ulang dari awal.

3. Memperkenalkan wavelet multivektor sebagai kakas matematika baru dalam merepresentasikan medan fisis dengan cara yang mudah.

1.3 Batasan Masalah

1. API yang dibuat baru dapat menyelesaikan persoalan geometrik pada bidang (2D) dan ruang (3D) seperti refleksi, rotasi, proyeksi dan perkalian geometrik dengan asumsi pengamat berada pada ruang 3D, yang aplikasinya diilustrasikan dalam fungsi-fungsi tes. API belum dapat menyelesaikan persoalan pada dimensi yang lebih tinggi dan persoalan yang melibatkan kalkulus geometrik.

2. Skema lifting yang digunakan hanya dalam 1D dan wavelet multivektor diperkenalkan hanya sebatas kakas matematika saja dan belum dimplementasikan dalam bentuk program.

1.4 Sistematika Penulisan

Bab 1 menjelaskan latar belakang, tujuan pengerjaan, ruang lingkup dan sistematika penulisan tugas akhir ini. Bab 2 menjelaskan tentang dasar-dasar aljabar geometrik [2, 3]. Dimulai dengan perkalian geometrik yang mendefinisikan ruang vektor kemudian dilanjutkan dengan pembahasan pada bidang dan ruang. Aplikasi aljabar geometri dalam ruang waktu juga dijelaskan dibab ini. Pengenalan konsep umum diakhiri dengan mengenalkan fungsi linier dalam aljabar geometrik.

Bab 3 mendemostrasikan bagaimana skema lifting dapat digunakan untuk menyusun representasi dalam bentuk wavelet multivektor. Pertama-tama dijelaskan konsep wavelet dan analisis multiresolusi. Selanjutnya dijelaskan skema lifting dengan contoh-contoh sederhana. Contoh representasi medan fisis dalam wavelet multivektor diberikan pada bab ini.

Bab 4 menjelaskan tentang API aljabar geometrik yang dibuat. Pada bab ini dijelaskan analasis dan perancangan API aljabar geometrik dengan berorientasi objek. Kemudian bagaimana implementasi, pengujian dan hasil-hasilnya. Bab 5 berisi kesimpulan dan saran.

Bab 2

Aljabar Geometrik

Aljabar geometrik adalah sebuah bahasa matematika yang sangat ampuh untuk mempelajari objek-objek geometrik [4, 5]. Hal-hal semacam itu banyak ditemukan dalam fisika, misal kecepatan sebuah partikel dapat direpresentasikan oleh sebuah vektor. Misalnya lagi, sebuah rotasi dapat direpresentasikan dengan bidang rotasi dan besarnya rotasi, objek ini disebut dengan bivektor yang merupakan kuantitas geometri berbentuk bidang berarah tanpa terkait dengan sistem koordinat apapun.

Secara matematis aljabar geometrik berasal dari aljabar Clifford (1878) dan aljabar Grassman (1877). Pada sekitar tahun 1960-an, David Hestenes menemukan arti geometrik dari aljabar Pauli dan aljabar Dirac, kemudian ia menuliskannya dalam buku *Aljabar Ruang-Waktu* dan mengembangkannya menjadi *Aljabar Geometrik sampai Kalkulus Geometrik*. Bukunya yang lengkap (walau tidak pernah diterbitkan) mengenai Aljabar Geometrik dan Kalkulus Geometrik adalah *New Foundation of Mathematical Physics*(1998), yang bisa dilihat langsung dari situs webnya

`http:// modellingnts.la.asu.edu/GC_R&D.html.`

Sekarang ini aljabar geometrik telah diterapkan pada bidang-bidang seperti geometri komputasi [6, 7], matematika fisika [1, 8] mekanika kuantum [9], elektromagnetik [10], gravitasi [11], relativitas [12], dan *computer vision* [13]. Semuanya menggunakan bahasa yang sama yaitu aljabar geometrik sehingga pada gilirannya, para ilmuwan akan lebih mudah untuk bekerjasama dan saling berinteraksi.

2.1 Aljabar Geometrik dalam Ruang Vektor

Cara yang paling cepat untuk memahami aljabar geometrik adalah melalui konsep *ruang vektor* [2]. Pada umumnya ruang linier dianggap sama dengan ruang vektor, namun pemahaman akan lebih mudah dicapai jika keduanya dibedakan.

Ruang linier adalah sebuah himpunan banyak elemen yang tertutup di bawah operasi penjumlahan dan operasi perkalian skalar.

Definisi di atas tidak sepenuhnya melukiskan konsep geometri dari vektor karena belum mendefinisikan aturan untuk perkalian dengan vektor.

Ruang vektor adalah sebuah ruang linier dimana perkalian geometrik (*geometric product*) didefinisikan. Dari sebuah ruang vektor, banyak ruang linier yang bisa dibuat.

2.1.1 Perkalian Geometrik

Misal $\mathcal{V}_n$ adalah sebuah ruang vektor n-dimensi dari bilangan riil. Perkalian geometrik dari vektor a, b, c dalam $\mathcal{V}_n$ didefinisikan oleh tiga aturan dasar:

1. $a(bc) = (ab)c$, sifat asosiatif
2. $a(b + c) = ab + ac$
   $(b + c)a = ba + ca$, sifat distributif kiri dan kanan
3. $a^2 = a \cdot a = ||a||^2$, sifat kontraksi

Karena $a \cdot a$ hanyalah perkalian skalar biasa (menghasilkan bilangan riil), maka aturan ketiga ini mengaitkan aljabar dengan kuantitas yang dapat diukur. Dengan menggunakan aturan ini pula, dapat diturunkan formula umum perkalian skalar dari dua vektor. Jika a dan b adalah vektor maka $a+b$ juga vektor, dan dengan demikian penerapan aturan ketiga menghasilkan

$$(a + b)^2 = (a + b) \cdot (a + b)$$

yang bila diperluas lebih lanjut didapatkan

$$\begin{aligned} a^2 + b^2 + ab + ba &= a^2 + b^2 + 2\, a \cdot b \\ (ab + ba) &= 2\, a \cdot b \\ a \cdot b &= \frac{1}{2}(ab + ba) \end{aligned} \tag{2.1}$$

Persamaan (2.1) merupakan perkalian skalar atau perkalian dalam (*inner product*) dari dua buah vektor dalam aljabar geometrik. Perkalian skalar mempunyai bentuk simetrik dengan bentuk antisimetriknya adalah

$$a \wedge b = \frac{1}{2}(ab - ba) \tag{2.2}$$

yang mendefinisikan perkalian eksterior/luar (*outer product*) dari dua buah vektor. Hal ini pertama kali diperkenalkan oleh Grassman(1877).

Clifford (1878) menemukan Aljabar Geometrik dengan menggabungkan perkalian skalar dengan perkalian eksterior menjadi satu buah perkalian geometrik (*geometric product*). Dari persamaan(2.1) dan (2.2) diperoleh definisi dari perkalian geometrik.

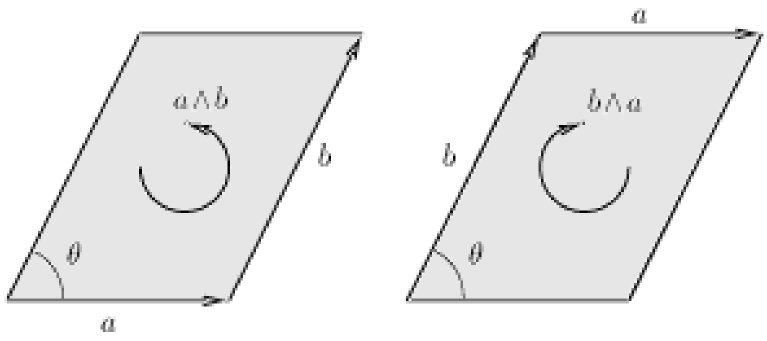
$$ab = a \cdot b + a \wedge b \tag{2.3}$$

Perkalian ini dapat-balik (*invertible*), sehingga persamaan seperti $ab = C$ mempunyai solusi $b = a^{-1}C$. Hal ini tidak mungkin dilakukan dengan perkalian skalar saja maupun perkalian eksterior saja. Karena a^2 adalah sebuah skalar, maka selama a bukan nol kita dapat menulis

$$a^{-1} = \frac{a}{a^2} \tag{2.4}$$

yang merupakan invers dari sebuah vektor.

Meskipun ruang vektor $\mathcal{V}_n$ tertutup dibawah penjumlahan vektor, ia tidak tertutup dibawah operasi perkalian seperti yang ditunjukkan oleh aturan tiga dan persamaan (2.3) di atas. Bahkan dengan perkalian dan penjumlahan vektor-vektor pada $\mathcal{V}_n$ diperoleh sebuah ruang linier $\mathcal{G}_n$=$\mathcal{G}(\mathcal{V}_n)$ yang lebih besar dan disebut dengan aljabar geometrik dari $\mathcal{V}_n$. Ruang linier ini tentunya tertutup dibawah perkalian maupun penjumlahan.



Gambar 2.1: Perkalian luar dari dua buah vektor menghasilkan sebuah bivektor yang merupakan elemen bidang berarah dengan luas $|a||b|\sin\theta$ [14].

Interpretasi geometris dari perkalian skalar dan eksterior dalam perkalian geometri dapat dilihat di bawah ini. Persamaan (2.1) mengimplikasikan bahwa

$$ab = -ba \iff a \cdot b = 0 \tag{2.5}$$

sedang persamaan (2.2) menghasilkan

$$ab = ba \iff a \wedge b = 0 \tag{2.6}$$

Ini berarti perkalian geometrik ab antikomutatif *jika dan hanya jika* a dan b orthogonal dan komutatif *jika dan hanya jika* kedua vektor kolinier. Persamaan (2.3) menyatakan bahwa ab adalah campuran dari bagian komutatif dan antikomutatif. Jadi ukuran arah a relatif terhadap b menunjukkan derajat komutativitas dari ab. Bahkan, kuantitas $a^{-1}b = ab/a^2$ menunjukkan besar dan arah relatif dari a dan b. Kuantitas ini juga bisa dianggap sebagai operator yang mentransformasi dari a ke b, seperti dalam persamaan $a(a^{-1}b) = b$. Sifat ini dimanfaatkan dalam teori rotasi.

2.1.2 Bivektor

Interpretasi geometris dapat dilukiskan dengan memperhatikan kembali persamaan (2.2). Perkalian luar dari a dan b antikomut dengan kedua vektor a dan b.

$$\begin{aligned} a(a \wedge b) &= \frac{1}{2}a(ab - ba) \\ &= \frac{1}{2}(a^2b - aba) \\ &= \frac{1}{2}(ba^2 - aba) \\ &= -(a \wedge b)a \end{aligned} \tag{2.7}$$

Karena sifat antikomutatifnya ini maka $a \wedge b$ tidak mengandung komponen skalar. Aturan-aturan di atas dapat digunakan untuk membuktikan bahwa $a \wedge b$ tidak mengandung komponen vektor c, Jika kita anggap $a \wedge b$ mengandung komponen vektor c maka $(a \wedge b) \cdot c$ akan mengandung komponen skalar.

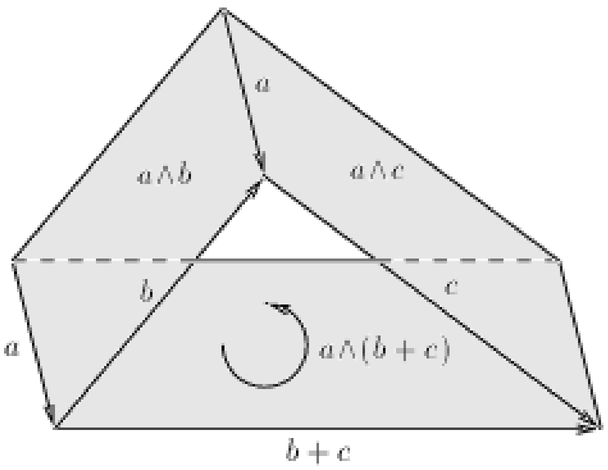
$$(a \wedge b) \cdot c = \frac{1}{2}((a \wedge b)c + c(a \wedge b))$$

tapi $(a \wedge b)c + c(a \wedge b)$ antikomut dengan vektor d yang memenuhi $d \cdot a = d \cdot b = d \cdot c = 0$, sehingga tidak mungkin mengandung komponen skalar. Hasil perkalian luar dari dua vektor ternyata melahirkan objek baru yang didefinisikan sebagai grade-2 dan disebut bivektor. Secara geometris ia merepresentasikan *segmen bidang berarah* oleh sapuan vektor a menuju b pada bidang lingkaran yang dibentang oleh vektor a dan b.

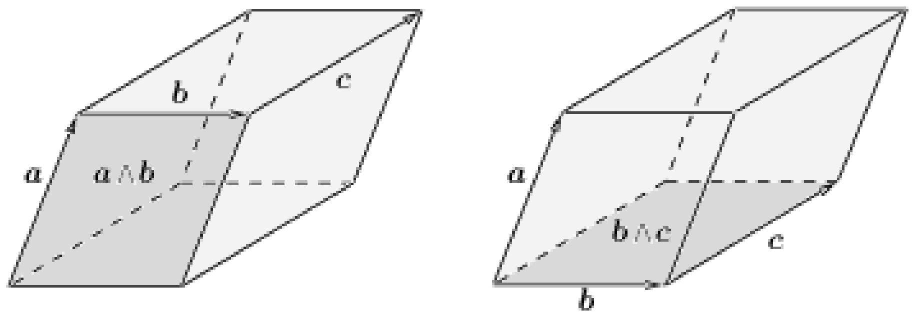
Sebuah bivektor yang berbentuk $(a \wedge b)$ disebut *blade*. Secara umum, bivektor membentuk sebuah ruang linier yang dibentuk dari penjumlahan berbagai *blade*. Hanya dalam dimensi dua dan tiga semua bivektor dapat dituliskan dalam sebuah *blade* saja.

2.1.3 Trivektor

Hasil perkalian luar $a \wedge b \wedge c$ disebut trivektor dan membentuk *segmen volume berarah* yang dapat divisualisasikan sebagai volume yang dibentuk dari sapuan area $a \wedge b$ sepanjang c.



Gambar 2.2: Penjumlahan bivektor dalam ruang 3D dapat divisualisasikan seperti penjumlahan vektor. Ini karena semua bivektor dalam 2D dan 3D dapat dituliskan dalam sebuah *blade* saja [14].



Gambar 2.3: Hasil sapuan dari $a \wedge b$ sepanjang c menghasilkan trivektor. Begitu juga dengan sapuan $b \wedge c$ sepanjang a, yang mengilustrasikan sifat asosiatif dari *blade*, $(a \wedge b) \wedge c = a \wedge (b \wedge c)$ [14].

2.1.4 Multivektor

Aljabar geometrik dari ruang n-dimensi (ruang Euclid) disebut $\mathcal{G}_n$. Elemen dari aljabar ini disebut multivektor dan biasa ditulis dalam huruf besar/kapital, sedang huruf kecil digunakan untuk vektor. Secara umum, vektor dan multivektor tidak dicetak tebal. Ruang $\mathcal{G}_n$ adalah linier terhadap bilangan riil. Jika λ, μ adalah skalar dan A, B adalah multivektor $(A, B \in \mathcal{G}_n)$, maka

$$\lambda A + \mu B \in \mathcal{G}_n, \quad \forall \lambda, \mu \tag{2.8}$$

Ruang linier $\mathcal{G}_n$ berlapis-lapis, dan setiap multivektor dapat dituliskan sebagai jumlah dari suku-suku setiap lapis.

$$A = \langle A \rangle_0 + \langle A \rangle_1 + \cdots = \sum_r \langle A \rangle_r. \tag{2.9}$$

Setiap lapis subruang dari $\mathcal{G}_n$ tertutup dibawah penjumlahan dan membentuk sebuah subruang linier. Notasi $\langle A \rangle_r$ menyatakan operasi proyeksi untuk mengambil bagian lapis-r dari A. Untuk lapis-0 yaitu bagian skalar dari A cukup dituliskan sebagai $\langle A \rangle$. Multivektor yang hanya mengandung suku-suku dalam satu lapis saja disebut *homogen* dan ditulis sebagai A_r, jadi

$$\langle A_r \rangle_r = A_r$$

Sifat asosiatif dan distributif juga berlaku bagi multivektor.

Urut-balik (*reverse*) dari multivektor A, ditulis $\widetilde{A}$, yang membalik urutan semua perkalian vektor yang membentuk multivektor sehingga

$$\begin{aligned} (\widetilde{AB}) &= \widetilde{B}\widetilde{A}, && (2.10) \\ \tilde{a}_i &= a_i, && (2.11) \\ (a_1 a_2 \ldots a_r)\tilde{} &= a_r \ldots a_2 a_1. && (2.12) \end{aligned}$$

Untuk multivektor homogen didapatkan

$$\widetilde{A}_r = (-1)^{r(r-1)/2} A_r \tag{2.13}$$

Modulus dari multivektor didefinisikan sebagai

$$|A| = \langle \widetilde{A}A \rangle^{\frac{1}{2}} \tag{2.14}$$

Perkalian skalar dari multivektor biasa ditulis

$$A * B = \langle AB \rangle. \tag{2.15}$$

Perkalian skalar selalu simetrik sehingga

$$\begin{aligned} \langle AB \rangle &= \langle BA \rangle, & (2.16)\\ \langle A \cdots BC \rangle &= \langle CA \cdots B \rangle. & (2.17) \end{aligned}$$

Sifat ini dinamakan pengurutan kembali secara siklis.

Perkalian komutator didefinisikan sebagai

$$A \times B = \frac{1}{2}(AB - BA). \tag{2.18}$$

Perkalian komutator ini memenuhi indentitas Jacobi

$$A \times (B \times C) + B \times (C \times A) + C \times (A \times B) = 0. \tag{2.19}$$

Konvensi urutan operator adalah perkalian dalam, luar dan skalar didahulukan dari perkalian geometrik. Jadi $a \cdot bc$ berarti $(a \cdot b)c$ bukannya $a \cdot (bc)$.

Dekomposisi Perkalian Geometrik Perkalian dalam $a \cdot b = \langle ab \rangle$ adalah sebuah skalar dan perkalian luar $a \wedge b = \langle ab \rangle_2$ adalah bivektor, jadi perkalian dalam dengan sebuah vektor merupakan operator penurun lapis dan perkalian luar dengan sebuah vektor merupakan operator penaik lapis. Perkalian geometrik dengan vektor menurunkan dan menaikkan lapis multivektor satu tingkat.

$$aA_r = a \cdot A_r + a \wedge A_r \tag{2.20}$$

di mana

$$\begin{aligned} a \cdot A_r &= \langle aA_r \rangle_{r-1} = \frac{1}{2}(aA_r - (-1)^r A_r a) = (-1)^{k+1} A_k \cdot a & (2.21)\\ a \wedge A_r &= \langle aA_r \rangle_{r+1} = \frac{1}{2}(aA_r + (-1)^r A_r a) = (-1)^k A_k \wedge a & (2.22) \end{aligned}$$

Definisi perkalian geometrik untuk vektor tidak berlaku untuk multivektor. Untuk multivektor homogen, perkalian geometriknya didekomposisikan sebagai berikut,

$$A_r B_s = \langle A_r B_s \rangle_{|r-s|} + \langle A_r B_s \rangle_{|r-s|+2} + \cdots \langle A_r B_s \rangle_{r+s}. \tag{2.23}$$

Lapis terendah dan lapis tertinggi dalam deret ini biasa ditulis dalam bentuk

$$A_r \cdot B_s = \langle A_r B_s \rangle_{|r-s|} \tag{2.24}$$

$$A_r \wedge B_s = \langle A_r B_s \rangle_{r+s} \tag{2.25}$$

Perkalian eksterior bersifat asosiatif dan memenuhi sifat simetri

$$A_r \wedge B_s = (-1)^{rs} B_s \wedge A_r \tag{2.26}$$

Basis Cara alami dalam memandang $\mathcal{G}_n$ adalah dalam suku-suku vektor basis orthonormal $\{e_i\}$, $i = 1 \dots n$. Dalam kasus ini suatu basis untuk seluruh aljabar dibangun sebagai

$$1,\ e_i,\ e_i e_j\ (i<j), e_i e_j e_k\ (i<j<k) \qquad dst.$$

Subruang lapis-r dari $\mathcal{G}_n$ disebut $\mathcal{G}_n^r$, dimana dimensi dari subruang ini adalah

$$\mathrm{Dim}[\mathcal{G}_n^r] = \binom{n}{r} = \text{kombinasi } r \text{dari } n. \tag{2.27}$$

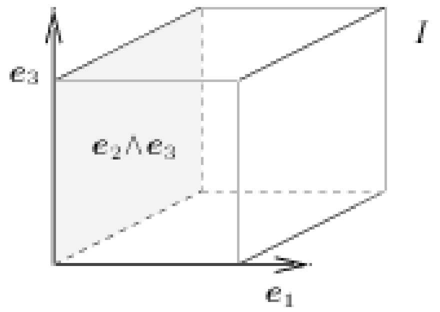
Jumlah total dimensi $\mathcal{G}_n$ adalah

$$\mathrm{Dim}[\mathcal{G}_n] = \sum_{r=0}^{n} \binom{n}{r} = 2^n \tag{2.28}$$

Pseudoscalar Perkalian eksterior dari n vektor mendefinisikan sebuah *blade* lapis n yang merupakan kelipatan dari *pseudoscalar* dalam $\mathcal{G}_n$. Pseudoscalar ini ditulis sebagai I dan dinormalisasikan menjadi 1,

$$|I|^2 = 1.$$

Tanda dari I^2 bergantung pada ukuran ruang dan cirinya. Pseudoscalar dipilih supaya mengikuti aturan tangan kanan. Dalam ruang berdimensi ganjil,



Gambar 2.4: Perkalian dari sebuah vektor dan trivektor dalam ruang 3D menghasilkan sebuah bivektor ($Ie_1 = e_1 I = e_1 e_1 e_2 e_3 = e_2 e_3$). Ini mengilustrasikan prinsip dualitas [14].

I komut dengan semua vektor, sehingga komut juga dengan semua lapisan multivektor. Sedangkan dalam ruang berdimensi genap, I antikomut dengan vektor sehingga antikomut dengan semua lapis ganjil multivektor, dan komut dengan semua multivektor lapis genap. Ini semua dapat dirangkum sebagai

$$IA_r = (-1)^{r(n-1)} A_r I \tag{2.29}$$

Dualitas Hasil kali dari pseudoscalar I lapis n dengan sebuah multivektor lapis-r A_r adalah sebuah multivektor lapis $n - r$. Operasi ini disebut dengan *transformasi dualitas*. Jika A_r adalah sebuah *blade*, IA_r akan menghasilkan *komplemen orthogonal* dari A_r berupa blade yang terbentuk dari vektor-vektor yang tidak dikandung A_r.

Dualitas ini juga berguna untuk menukar antara perkalian skalar dengan perkalian eksterior.

$$\begin{aligned}
A_r \cdot (B_s I) &= \langle A_r B_s I \rangle_{|r-(n-s)|} \\
&= \langle A_r B_s I \rangle_{n-(r+s)} \\
&= \langle A_r B_s \rangle_{r+s} I \\
&= A_r \wedge B_s I \qquad r+s \leq n \qquad (2.30) \\
A_r \wedge (B_s I) &= \langle A_r B_s I \rangle_{|r+(n-s)|} \\
&= \langle A_r B_s I \rangle_{n-(s-r)} \\
&= \langle A_r B_s \rangle_{s-r} I \\
&= A_r \cdot B_s I \qquad r \leq s \qquad (2.31)
\end{aligned}$$

2.2 Aljabar Geometrik dalam bidang (2-d)

Ruang dua dimensi dibentang oleh dua buah vektor basis orthonormal e_1 dan e_2. Vektor basis ini memenuhi

$$e_1^2 = e_2^2 = 1\ , \quad e_1 \cdot e_2 = 0 \qquad (2.32)$$

Pseudoscalarnya adalah bivektor

$$\begin{aligned}
I = e_1 e_2 = e_1 \cdot e_2 + e_1 \wedge e_2 &= e_1 \wedge e_2 \qquad (2.33) \\
&= -e_2 \wedge e_1 = -e_2 e_1 \qquad (2.34)
\end{aligned}$$

$e_1 e_2$ dipilih sehingga memenuhi aturan tangan kanan. Perkalian luar $e_1 \wedge e_2$ merepresentasikan bidang berarah. Kuantitas independen lainnya tidak bisa dibentuk karena vektor basisnya orthogonal sehingga vektor antikomut dengan pseudoscalar/bivektor.

$$Ie_1 = e_1 e_2 e_1 = -e_1 e_1 e_2 = -e_2 \qquad (2.35)$$

$$Ie_2 = e_1 e_2 e_2 = e_1 \qquad (2.36)$$

$$I^2 = (e_1 e_2)^2 = e_1 e_2 e_1 e_2 = -e_1 e_2 e_2 e_1 = -1 \qquad (2.37)$$

Jadi perkalian yang lebih tinggi dari vektor-vektor basis menghasilkan satu dari empat elemen basis yang membentang aljabar $\mathcal{G}_2$

$$\begin{array}{ccc} 1 & \{e_1, e_2\} & e_1 \wedge e_2 \\ \text{1 skalar} & \text{2 vektor} & \text{1 bivektor} \end{array} \qquad (2.38)$$

dengan multivektornya adalah kombinasi linier dari vektor-vektor basis. Misal,

$$\begin{aligned} A = a_0 + a_1e_1 + a_2e_2 + a_3e_1 \wedge e_2 \\ B = b_0 + b_1e_1 + b_2e_2 + b_3e_1 \wedge e_2 \end{aligned} \tag{2.39}$$

Hasil perkalian geometriknya dapat dituliskan

$$AB = p_0 + p_1e_1 + p_2e_2 + p_3e_1 \wedge e_2 \tag{2.40}$$

dimana

$$\begin{aligned} p_0 &= a_0b_0 + a_1b_1 + a_2b_2 - a_3b_3 \\ p_1 &= a_0b_1 + a_1b_0 + a_3b_2 - a_2b_3 \\ p_2 &= a_0b_2 + a_2b_0 + a_1b_3 - a_3b_1 \\ p_3 &= a_0b_3 + a_3b_0 + a_1b_2 - a_2b_1 \end{aligned}$$

Bilangan Kompleks dan $\mathcal{G}_2$ Kuadrat dari bivektor $e_1 \wedge e_2 = I = -1$ dan menghasilkan rotasi sebanyak 90^0. Kombinasi skalar dan bivektor yang merupakan elemen basis lapis genap membentuk subaljabar genap (subruang linier) dan dapat dipandang sebagai bilangan kompleks. Dari situ dapat dituliskan

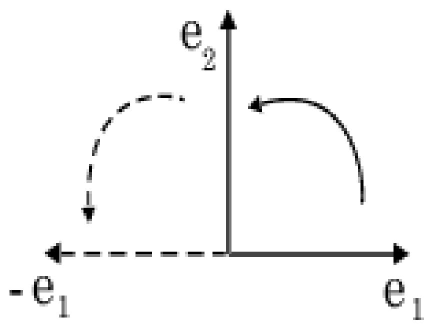
$$z = x + ye_1 \wedge e_2 = x + Iy. \tag{2.41}$$

$$\begin{aligned} z_1z_2 &= (x_1 + y_1e_1 \wedge e_2)(x_2 + y_2e_1 \wedge e_2) && (2.42) \\ &= (x_1x_2 - y_1y_2) + (x_1y_2 + y_1x_2)e_1 \wedge e_2 && (2.43) \end{aligned}$$

Ini menunjukkan bahwa interpretasi geometris dapat diperoleh secara langsung untuk persamaan yang mengandung bilangan imajiner.

Perkalian luar dari dua vektor a dan b dapat ditulis sebagai $a \wedge b_\perp$, dimana $b_\perp$ adalah proyeksi dari b yang tegak lurus a (perkalian luar dengan komponen paralel adalah nol). Komponen b yang tegak lurus terhadap a besarnya $|b| \sin\theta$ yang menunjukkan $a \wedge b = e_1e_2|a||b| \sin\theta$, sehingga $|a \wedge b| = |a||b| \sin\theta$. Untuk setiap vektor satuan m dan n,

$$mn = m \cdot n + m \wedge n = \cos\theta + m \wedge n_\perp = \cos\theta + \sin\theta e_1 \wedge e_2. \tag{2.44}$$



Gambar 2.5: Perkalian dengan pseudoscalar $I = e_1e_2$ memutar vektor pada dimensi 2 sebesar 90^0 berlawanan arah jarum jam [5].

Karena bentuknya sama (*isomorphism*) dengan bilangan kompleks, kita dapat menuliskan

$$mn = \cos\theta + \sin\theta e_1 \wedge e_2 = \mathrm{e}^{e_1\wedge e_2\theta} = \mathrm{e}^{I\theta} = \mathrm{e}^{B}, \tag{2.45}$$

di mana B adalah sebuah bivektor dengan besar θ. Perkalian $\mathrm{e}^{e_1\wedge e_2\theta}$ dengan sebuah vektor akan merotasikan vektor tersebut pada bidang e_1e_2, seperti terlihat pada persamaan di bawah ini:

$$\mathrm{e}^{I\theta}e_1 = (\cos\theta - \sin\theta I)e_1 = \cos\theta e_1 + \sin\theta e_2 \tag{2.46}$$

$$\mathrm{e}^{I\theta}e_2 = -\sin\theta e_1 + \cos\theta e_2 \tag{2.47}$$

2.3 Aljabar Geometrik dalam ruang (3-d)

Penambahan sebuah vektor othonormal lagi ke dalam set basis 2-d, membangkitkan objek-objek geometri berikut:

$$\begin{array}{cccc} 1 & \{e_1, e_2, e_3\} & \{e_1e_2, e_2e_3, e_3e_1\} & e_1e_2e_3 \\ \text{1 skalar} & \text{3 vektor} & \text{3 bivektor} & \text{1 trivector} \end{array} \tag{2.48}$$

Objek-objek ini membentang ruang linier dengan $(1 + 3 + 3 + 1) = 8 = 2^3$ dimensi. Seperti dalam 2D, pseudoscalar dipilih mengikuti aturan tangan kanan,

$$I = e_1e_2e_3 \tag{2.49}$$

Diperkenalkan perkalian geometrik,

$$(e_1e_2)e_3 = e_1e_2e_3$$
$$(e_1e_2e_3)e_k = e_k(e_1e_2e_3) \qquad (2.50)$$

dan

$$(e_1e_2e_3)^2 = e_1e_2e_3e_1e_2e_3 = e_1e_2e_1e_2e_3^2 = -1 \qquad (2.51)$$

Trivektor (pseudoscalar) $e_1e_2e_3$ juga mempunyai kuadrat negatif. Bahkan, dari delapan objek geometris tersebut, empat diantaranya mempunyai kuadrat negatif, $\{e_1e_2, e_2e_3, e_3e_1\}$ dan $e_1e_2e_3$. Dari ini semua, pseudoscalar $e_1e_2e_3$ dibedakan melalui sifat komutatifnya.

Relasi di atas membawa pemahaman geometrik baru:

- Sebuah bivektor merotasikan vektor pada bidangnya sendiri sebesar 90^0, tapi membentuk trivektor (volume) dengan vektor yang tegak lurus padanya.
- Trivektor $e_1 \wedge e_2 \wedge e_3$ komut dengan semua vektor, sehingga ia komut dengan semua multivektor.

Dari persamaan (2.49) dan (2.50) diperoleh

$$Ie_1 = e_2e_3, \qquad Ie_2 = e_3e_1, \qquad \text{dan} Ie_3 = e_1e_2 \qquad (2.52)$$

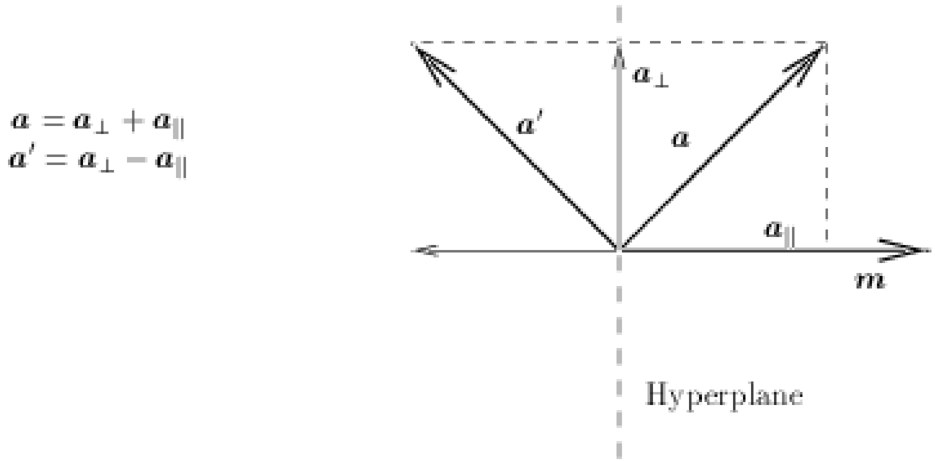
yang merupakan operasi dualitas karena memetakan elemen lapis-r menjadi elemen lapis-$(n-r)$ dengan n adalah bilangan dimensinya. Dalam 3D operasi dualitas memetakan bivektor menjadi vektor, begitu sebaliknya. Dengan menggunakan operasi dualitas dalam 3D dapat diperoleh kembali perkalian silang yang berasal dari perkalian luar,

$$a \times b = -Ia \wedge b \qquad (2.53)$$

Misal, $e_1 \times e_2 = -Ie_1e_2 = IIe_3 = e_3$ seperti yang diharapkan

Sebuah multivektor M dalam 3D dapat dituliskan sebagai berikut

$$M = \alpha + \boldsymbol{a} + \boldsymbol{b}I + \beta I \qquad (2.54)$$



Gambar 2.6: Refleksi pada sebuah bidang yang tegak lurus m [14].

2.4 Refleksi dan Rotasi

Aljabar geometrik menyediakan cara yang elegan untuk menangani refleksi dan rotasi. Apabila diberikan sebuah vektor satuan n ($n^2 = 1$), setiap sembarang vektor a dapat dipecah menjadi bagian pararel dan tegak lurus terhadap n,

$$\begin{aligned} a &= n^2 a \\ &= n(n \cdot a + n \wedge a) \\ &= a_\parallel + a_\perp \end{aligned} \tag{2.55}$$

di mana

$$a_\parallel = a \cdot nn \tag{2.56}$$

$$a_\perp = nn \wedge a \tag{2.57}$$

Hasil pencerminan a pada sebuah bidang yang orthogonal terhadap n adalah vektor $a_\parallel - a_\perp$, yang dapat dituliskan sebagai,

$$\begin{aligned} a_\parallel + a_\perp &= nn \wedge a - a \cdot nn \\ &= -n \cdot an - n \wedge an \\ &= -nan \end{aligned} \tag{2.58}$$

Rumus untuk refleksi ini berlaku untuk sembarang multivektor. Misal jika vektor a dan b keduanya direfleksikan pada bidang tegaklurus terhadap n, maka bivektor $a \wedge b$ direfleksikan menjadi

$$(-nan) \wedge (-nbn) = \frac{1}{2}(nannbn - nbnnan) = na \wedge bn$$

Rotasi dibangun dari kombinasi pasangan refleksi (dua refleksi). Refleksi pertama pada bidang orthogonal terhadap n, kemudian refleksi kedua pada bidang orthogonal terhadap m menghasilkan vektor baru

$$\begin{aligned} -m(-nan)m &= mnanm \\ &= Ra\widetilde{R} \end{aligned} \tag{2.59}$$

di mana

$$R = mn \tag{2.60}$$

Multivektor R disebut *rotor* yang hanya mengandung elemen lapis genap dan memenuhi identitas

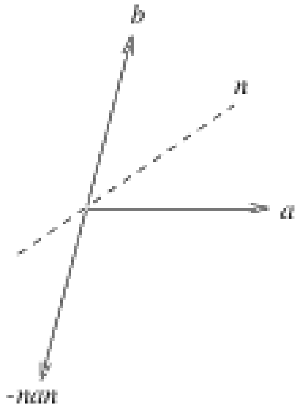
$$R\widetilde{R} = \widetilde{R}R = 1 \tag{2.61}$$

Persamaan (2.61) memastikan bahwa produk skalar dari dua vektor invarian di bawah rotor

$$\begin{aligned} (Ra\widetilde{R}) \cdot (Rb\widetilde{R}) &= \langle Ra\widetilde{R}Rb\widetilde{R} \rangle \\ &= \langle a\widetilde{R}Rb\widetilde{R}R \rangle \\ &= \langle ab \rangle \\ &= a \cdot b \end{aligned} \tag{2.62}$$

Misal diinginkan untuk merotasi vektor satuan a ke arah vektor satuan b, dengan tetap membiarkan semua vektor yang tegaklurus a dan b tak berubah. Ini dapat diselesaikan dengan sebuah refleksi pada bidang tegaklurus vektor satuan dipertengahan a dan b,

$$n = (a + b)/|a + b| \tag{2.63}$$



Gambar 2.7: Sebuah rotasi tersusun atas dua refleksi [3].

ini merefleksikan a ke $-b$. Refleksi kedua dibutuhkan untuk membawanya ke b, yang harus berlangsung pada bidang yang tegak lurus terhadap b. Semua ini memberikan rotor

$$R = bn = \frac{1+ba}{|a+b|} = \frac{1+ba}{\sqrt{2(1+b\cdot a)}}, \tag{2.64}$$

yang menghasilkan sebuah rotor sederhana pada bidang $a \wedge b$. Rotornya dituliskan

$$b = Ra\widetilde{R} \tag{2.65}$$

dan transformasi baliknya diberikan oleh

$$a = \widetilde{R}bR \tag{2.66}$$

Transformasi $a \mapsto Ra\widetilde{R} = b$ berlaku di seluruh ruang berdimensi berapapun. Lebih jauh lagi, rotor bekerja terhadap semua tipe objek geometris (multivektor). Ini karena dia bekerja terhadap semua perkalian vektor,

$$a_1a_2\ldots a_r \mapsto Ra_1\widetilde{R}Ra_2\widetilde{R}\ldots Ra_r\widetilde{R} = Ra_1a_2\ldots a_r\widetilde{R} \tag{2.67}$$

Rotor sendiri bukanlah merupakan objek geometris tapi bisa dianggap sebagai operator yang bekerja terhadap objek geometris.

Dari persamaan (2.45) sebuah rotor juga dapat dituliskan dalam bentuk

$$R = e^{-B/2} \tag{2.68}$$

dimana B adalah sebuah bivektor yang merepresentasikan bidang di mana rotasi berlangsung. Persamaan (2.46 dan (2.47) mengilustrasikannya dalam dua dimensi di mana kuantitas $\exp(-I\theta)$ merotasikan vektor berlawanan arah jarum jam sebesar sudut θ. Ini berlaku dalam 2D karena kita selalu bisa menuliskan

$$e^{-I\theta/2} r e^{I\theta/2} = e^{-I\theta} r \tag{2.69}$$

Pada ruang tiga dimensi, bivektor dapat dipetakan menjadi sebuah vektor orthogonal dengan operasi dualitas, yang memberikan

$$R = e^{-I\boldsymbol{a}/2} = \cos(|\boldsymbol{a}|/2) - I\hat{\boldsymbol{a}}\sin(|\boldsymbol{a}|/2) \tag{2.70}$$

untuk rotasi sebesar $|\boldsymbol{a}|$ radian terhadap sumbu $\boldsymbol{a}$, dimana $\hat{\boldsymbol{a}}$ adalah vektor satuan pada arah $\boldsymbol{a}$. Pada dimensi yang lebih tinggi, transformasi duasisi/bilinier seperti persamaan (2.59) diperlukan. Jika $R = e^{-B/2}$, dan B adalah sebuah bivektor (sehingga $\widetilde{B} = -B$), maka balikannya adalah $\widetilde{R} = e^{B/2}$. Sebuah rotasi sederhana dari sebuah multivektor A, sebesar $|a|$ radian terhadap sumbu a, ditulis sebagai

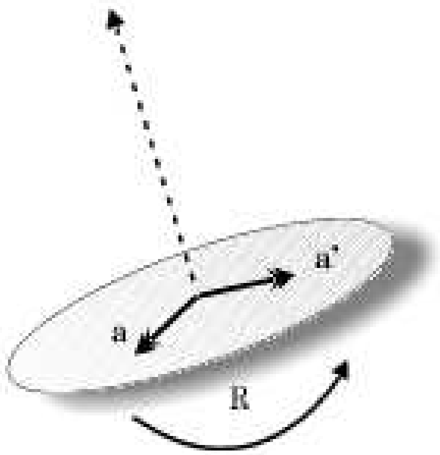
$$A \mapsto e^{-Ia/2} A e^{Ia/2} \tag{2.71}$$

2.5 Aljabar ruang dan waktu

Aljabar ruang-waktu (*spacetime algebra* biasa disingkat STA) adalah dasar dari aplikasi aljabar geometrik dalam fisika relativistik dan gravitasi. Konvensi satuan unit yang dipakai adalah $c = \epsilon_0 = \hbar = 1$ Vektor *spacelike* dan vektor *timelike* mempunyai tanda yang berlawanan untuk kuadratnya. Kuadrat sebuah vektor tidak lagi definit positif, dan sebuah vektor x disebut *timelike, lightlike*, *spacelike* bergantung apakah $x^2 > 0$, $x^2 = 0$, atau $x^2 < 0$. Ruang-waktu terdiri atas satu vektor *timelike* dan tiga vektor *spacelike* yang saling bebas. Aljabar ini disusun dari empat vektor basis orthonormal, $\{\gamma_\mu\}$, $\mu = 0 \ldots 3$, yang memenuhi

$$\gamma_\mu \cdot \gamma_\nu = \eta_{\mu\nu} = \mathrm{diag}(+\,-\,-\,-) \tag{2.72}$$

$$= \frac{1}{2}(\gamma_\mu\gamma_\nu + \gamma_\nu\gamma_\mu). \tag{2.73}$$



Gambar 2.8: Rotor R yang membawa a ke a'. Konsep dari vektor tegaklurus sebagai sumbu putar tidak diperlukan lagi, bivektor atau bidang rotasilah yang lebih penting [5].

Dengan perkalian vektor-vektor basis didapatkan keseluruh 16 elemen dari STA:

1	$\{\gamma_\mu\}$	$\{\boldsymbol{\sigma}_k, I\boldsymbol{\sigma}_k\}$	$\{I\gamma_\mu\}$	I
1 skalar	4 vektor	6 bivektor	4 pseudovektor	1 pseudoscalar

Bivektor ruangwaktu $\{\boldsymbol{\sigma}_k\},\ k = 1 \ldots 3$ didefinisikan oleh

$$\boldsymbol{\sigma}_k = \gamma_k \gamma_0 \tag{2.74}$$

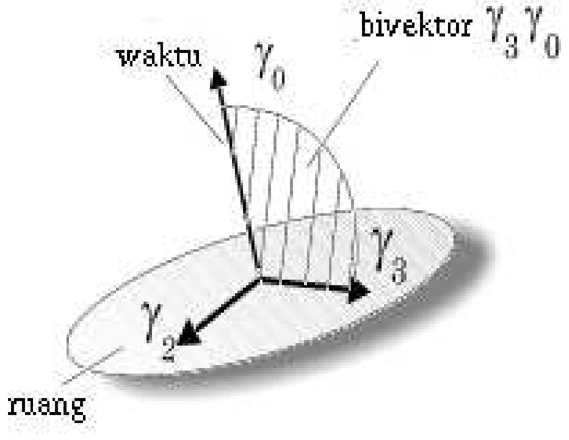
Bivektor-bivektor ini membentuk sebuah kerangka orthonormal dari vektor di dalam ruang, relatif terhadap arah γ_0. Pseudoskalar ruangwaktu I didefinisikan oleh

$$I = \gamma_0 \gamma_1 \gamma_2 \gamma_3 \tag{2.75}$$

dan, karena kita ada di ruang berdimensi genap, I antikomut dengan semua elemen lapis ganjil dan I komut dengan semua elemen lapis genap. Dari persamaan (2.74) dan (2.75) diperoleh

$$\boldsymbol{\sigma}_1 \boldsymbol{\sigma}_2 \boldsymbol{\sigma}_3 = \gamma_1 \gamma_0 \gamma_2 \gamma_0 \gamma_3 \gamma_0 = \gamma_0 \gamma_1 \gamma_2 \gamma_3 = I \tag{2.76}$$

Keenam buah bivektor ruangwaktu dapat dipisah menjadi vektor relatif dan bivektor relatif dalam ruang 3D melalui sebuah operasi yang bergantung



Gambar 2.9: Ilustrasi dari sumbu ruangwaktu 4D. Satu dari bivektor ruang-waktu diperlihatkan [5].

pada kerangka pengamat. Bivektor $\boldsymbol{\sigma}_k = \gamma_k\gamma_0$ isomorphik dengan basis vektor dalam ruang 3D. Contohnya:

$$I\boldsymbol{\sigma}_3 = \boldsymbol{\sigma}_1\boldsymbol{\sigma}_2\boldsymbol{\sigma}_3\boldsymbol{\sigma}_3 = \gamma_1\gamma_0\gamma_2\gamma_0\gamma_3\gamma_0\gamma_3\gamma_0 = \gamma_1\gamma_0\gamma_2\gamma_0 = \boldsymbol{\sigma}_1\boldsymbol{\sigma}_2 \tag{2.77}$$

$$\boldsymbol{\sigma}_k^2 = \gamma_k\gamma_0\gamma_k\gamma_0 = +1 \tag{2.78}$$

$$\boldsymbol{\sigma}_j\boldsymbol{\sigma}_k = \gamma_j\gamma_0\gamma_k\gamma_0 = \gamma_k\gamma_j\gamma_0\gamma_0 = -\boldsymbol{\sigma}_k\boldsymbol{\sigma}_j \ (j \neq k) \tag{2.79}$$

Sedang $I\boldsymbol{\sigma}_k$ isomorphik dengan basis bivektor dalam ruang 3D. Ini berarti subaljabar genap dari STA isomorphik dengan aljabar geometrik dari ruang 3D.

Jika γ_0 diambil sebagai vektor kecepatan kerangka lab/referensi, maka $\boldsymbol{\sigma}_k$ merepresentasikan sebuah kerangka dalam ruang, relatif terhadap vektor γ_0. Sekarang sebuah vektor dapat dilihat sebagai sebuah segmen garis yang ada selama jangka waktu tertentu. Oleh karena itu sebuah vektor direpresentasikan oleh sebuah bivektor ruangwaktu.

Dari relasi (2.76) diperoleh interpretasi geometris yang sangat penting, karena ruang relatif dan ruangwaktu mempunyai tepat satu pseudoscalar yang identik. Sebuah sistem inersia dapat dijelaskan secara lengkap oleh sebuah vektor (satuan) timelike yang menunjuk masa depan. Vektor ini diambil sebagai arah γ_0. Vektor pengamat ini memetakan antara vektor ruangwaktu $a = a^\mu \gamma_\mu$ dan subaljabar genap dari STA melalui

$$a\gamma_0 = a_0 + \boldsymbol{a} \tag{2.80}$$

dimana

$$a_0 = a \cdot \gamma_0 \tag{2.81}$$

$$\boldsymbol{a} = a \wedge \gamma_0 \tag{2.82}$$

Konvensinya adalah bivektor ruangwaktu (vektor relatif dalam ruang) dituliskan bercetak tebal. Pemisahan seperti ini disebut dengan pemisahan ruangwaktu (*spacetime split*). Jadi, pemisahan ruangwaktu dari sebuah vektor dalam ruangwaktu ke dalam kerangka ruang γ_0 dapat dilakukan dengan mengalikannya dengan γ_0.

Contoh, jika x adalah sebuah vektor ruang waktu yang menunjukkan posisi satu titik atau kejadian, maka pemisahan ruangwaktu kedalam kerangka γ_0 memberikan

$$x\gamma_0 = t + \boldsymbol{x} \tag{2.83}$$

yang mendefinisikan waktu pengamat

$$t = x \cdot \gamma_0 \tag{2.84}$$

dan vektor posisi relatif

$$\boldsymbol{x} = x \wedge \gamma_0 \tag{2.85}$$

Jika kita definisikan $\beta = v \cdot \gamma_0 = dt/d\tau$, maka kecepatan relatifnya adalah

$$\boldsymbol{v} = \frac{d\boldsymbol{x}}{dt} = \frac{d\boldsymbol{x}}{d\tau}\frac{d\tau}{dt} = \frac{v \wedge \gamma_0}{\beta} = \frac{v \wedge \gamma_0}{v \cdot \gamma_0}. \tag{2.86}$$

Sehingga pemisahan ruangwaktunya kedalam kerangka γ_0 menjadi

$$v\gamma_0 = v \cdot \gamma_0 + v \wedge \gamma_0 = \beta(1 + \boldsymbol{v}). \tag{2.87}$$

Karena $v^2 = 1$ adalah invarian, didapatkan

$$1 = (v\gamma_0)(\gamma_0 v) = \beta(1 + \boldsymbol{v})(1 - \boldsymbol{v}) = \beta^2(1 - \boldsymbol{v}^2), \tag{2.88}$$

untuk selanjutnya diperoleh

$$\beta = \frac{dt}{d\tau} = \frac{1}{\sqrt{(1 - \boldsymbol{v}^2)}} \tag{2.89}$$

Sedangkan skalar x^2 dapat didekomposisikan menjadi

$$x^2 = x\gamma_0\gamma_0 x = (t + \boldsymbol{x})(t - \boldsymbol{x}) = t^2 - \boldsymbol{x}^2 \tag{2.90}$$

yang juga merupakan invarian-Lorentz.

2.6 Fungsi Linier

Fungsi-fungsi linier sangat penting di dalam fisika dan biasanya dituliskan dalam bentuk tensor yang melukiskan medan. Aljabar geometrik mengijinkan manipulasi secara bebas-indeks, yang membuat penurunan solusi tampak ringkas dan lebih sederhana. Langkah pertama adalah mendefinisikan $\mathsf{f}(a)$ sebagai sebuah fungsi linier yang memetakan vektor ke vektor dalam ruang yang sama. Untuk penulisan tangan, f biasanya digarisbawahi untuk membedakannya dengan objek lain.

Sebuah fungsi linier mempunyai sifat

$$\mathsf{f}(\lambda a + \mu b) = \lambda \mathsf{f}(a) + \mu \mathsf{f}(b) \tag{2.91}$$

untuk semua skalar λ, μ dan vektor a, b. Komposisi dari dua fungsi linier f dan g (fungsi yang pertama) menghasilkan fungsi yang ketiga sehingga kita dapat menuliskan

$$\mathsf{h}(a) = \mathsf{f}[\mathsf{g}(a)] = \mathsf{fg}(a). \tag{2.92}$$

Dalam mengkomposisikan fungsi-fungsi linier berlaku sifat asosiatif.

Sebuah fungsi linier dapat diperluas untuk mendefinisikan sebuah aksi pada sembarang multivektor melalui aksinya pada *blade*. Perluasan ke multivektor ini disebut *outermorphism*. Didefinisikan

$$\mathsf{f}(a \wedge b \wedge \cdots \wedge c) = \mathsf{f}(a) \wedge \mathsf{f}(b) \wedge \cdots \wedge \mathsf{f}(c) \tag{2.93}$$

Sisi sebelah kanan merupakan sebuah blade juga dan harus memiliki lapis yang sama dengan argumen asalnya. Oleh karena itu fungsi-fungsi linier yang diperluas bersifat mempertahankan lapis (*grade preserving*)

$$\mathsf{f}(A_r) = \langle \mathsf{f}(A_r) \rangle_r. \tag{2.94}$$

Sehingga, mereka juga multilinier

$$\mathsf{f}(\lambda A + \mu B) = \lambda \mathsf{f}(A) + \mu \mathsf{f}(B), \tag{2.95}$$

dan berlaku untuk semua skalar λ, μ dan semua multivektor A, B.

Contohnya adalah fungsi rotasi oleh sebuah rotor R yang dapat ditulis

$$\mathsf{R}(a) = Ra\widetilde{R}. \tag{2.96}$$

Aksi yang diperluas dari transformasi ini pada sebuah blade lapis-r diberikan oleh

$$\begin{aligned} \mathsf{R}(a \wedge b \wedge \cdots \wedge c) &= (Ra\widetilde{R}) \wedge (Rb\widetilde{R}) \wedge \cdots \wedge (Rc\widetilde{R}) \\ &= \langle Ra\widetilde{R}Rb\widetilde{R} \cdots Rb\widetilde{R} \rangle_r \\ &= \langle Rab \cdots c\widetilde{R} \rangle_r \\ &= R(a \wedge b \wedge \cdots \wedge c)\widetilde{R} \end{aligned} \tag{2.97}$$

Ekstensinya ke dalam multivektor menjadi sederhana

$$\mathsf{R}(A) = RA\widetilde{R}. \tag{2.98}$$

Contoh yang lain lagi adalah komposisi dari dua fungsi linier. Misal $\mathsf{h}(a) = \mathsf{fg}(a)$. Kita dapatkan

$$\begin{aligned} \mathsf{h}(a \wedge b \wedge \cdots \wedge c) &= \mathsf{fg}(a) \wedge \mathsf{fg}(b) \wedge \cdots \wedge \mathsf{fg}(c) \\ &= \mathsf{f}[\mathsf{g}(a) \wedge \mathsf{g}(b) \wedge \cdots \wedge \mathsf{g}(c)] \\ &= \mathsf{f}[\mathsf{g}(a \wedge b \wedge \cdots \wedge c)]. \end{aligned} \tag{2.99}$$

Sehingga untuk multivektor diperoleh

$$\mathsf{h}(A) = \mathsf{fg}(A). \tag{2.100}$$

Versor

Sebuah *versor* adalah sebuah elemen yang dapat ditulis sebagai sebuah perkalian geometrik dari vektor-vektor $A = a_1 a_2 \ldots a_r$. Tidak semua elemen dalam aljabar mempunyai invers, bahkan seandainya adapun, mungkin sulit untuk dihitung. Invers dari versor adalah

$$A^{-1} = \frac{\widetilde{A}}{\widetilde{A}A} \tag{2.101}$$

di mana $\widetilde{A}$ didefinisikan seperti persamaan (2.12) dan (2.13). Setiap ekstensi (sebagai *outermorphism*) dari sebuah transformasi linier F dalam $\mathcal{G}_n$, dapat direpresentasikan oleh versor F melalui *persamaan versor*

$$\mathsf{F}(x) = FxF^{-1} \tag{2.102}$$

Ini berarti dalam ruang 3D translasi dan rotasi dapat dilakukan sekaligus dengan perkalian versor. Contohnya adalah refleksi (F adalah vektor) dan rotasi (F adalah perkalian geometrik dari dua vektor yaitu eksponen dari sebuah bivektor). Karena refleksi dan rotasi dapat diperluas untuk objek multivektor, maka begitu juga dengan persamaan versor

$$\mathsf{F}(A) = FAF^{-1} = A' \tag{2.103}$$

di mana A' dapat diperoleh kembali dengan

$$A' = F^{-1}AF. \tag{2.104}$$

Bab 3

Wavelet dan Multivektor

3.1 Wavelet dan Analisis Multiresolusi

Pada saat ini, hampir tidak mungkin untuk memberikan definisi baku untuk wavelet karena perkembangannya yang pesat sehingga definisi yang sekarang boleh jadi menjadi salah di masa depan. Wim Sweldens [15, 16] memberikan definisi sebagai berikut:

"Wavelet adalah *blok penyusun* yang dapat men*dekorelasi* data dengan *cepat*".

Kalimat ini setidaknya mencakup tiga fitur utama wavelet. Pertama, wavelet adalah blok penyusun untuk himpunan data, fungsi atau sinyal secara umum. Ini berarti mereka membentuk sebuah basis atau yang lebih umum lagi sebuah *frame*. Oleh karena itu himpunan data, fungsi atau sinyal dapat ditulis dengan stabil sebagai kombinasi linier dari wavelet. Jika ditunjukkan suatu wavelet dengan $\psi_{j,k}$ dan koefisiennya dengan $\gamma_{j,k}$, maka dapat dituliskan sebuah fungsi umum f sebagai

$$f = \sum_j \sum_k \gamma_{j,k} \psi_{j,k}. \tag{3.1}$$

Kedua, wavelet mempunyai daya dekorelasi yang baik. Ini berarti representasi data dalam bentuk koefisien-koefisien wavelet γ_i lebih *ringkas* jika dibandingkan dengan representasi asalnya. Dalam jargon teori informasi, entropi representasi dengan wavelet lebih kecil daripada representasi asalnya. Dalam jargon teori aproksimasi, pendekatan yang teliti dari f dapat diperoleh dengan hanya menggunakan sedikit bagian dari koefisien-koefisien wavelet.

Cara untuk mendapatkan daya dekorelasi ini adalah dengan membuat wavelet yang menyerupai data yang ingin direpresentasikan. Lebih spesifik lagi, yang diinginkan adalah wavelet yang mempunyai struktur korelasi sama dengan data tersebut. Misal, kebanyakan sinyal yang dihadapi sehari-hari mempunyai kedua korelasi ruang(waktu) dan frekuensi. Sampel yang secara ruang berdekatan lebih banyak korelasinya daripada sampel yang berjauhan, dan frekuensi sering muncul dalam bentuk pita-pita. Untuk menganalisa dan merepresentasikan sinyal seperti itu dibutuhkan wavelet yang terlokalisasi dalam ruang(waktu) dan frekuensi.

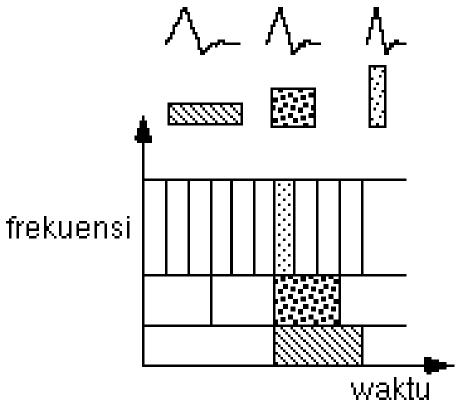
Terakhir, diinginkan agar representasi wavelet dari data tersebut dapat diperoleh secara *cepat*. Artinya, perpindahan antara representasi asal data dan representasi wavelet dari data tersebut dapat dilakukan dalam waktu yang sebanding dengan ukuran data. Dalam istilah informatik, kompleksitas penghitungan koefisien wavelet dan transformasi baliknya adalah $O(N)$. Kecepatan dan daya dekorelasi dari wavelet inilah yang menjadi kunci dari pelbagai aplikasi seperti: kompresi data, transmisi, penghilangan derau, pemulihan sinyal, dan komputasi numerik yang cepat.

3.1.1 Wavelet generasi pertama

Fungsi wavelet generasi pertama $\psi_{j,k}$ didefinisikan sebagai translasi dan dilatasi diadik[1] dari sebuah fungsi *mother wavelet* $\psi = \psi_{j,k}(x) = 2^{(j/2)}\psi(2^j x - k)$. Sifat-sifat utama dari wavelet generasi pertama [20] adalah:

1. wavelet membentuk sebuah basis untuk $L_2(\mathbf{R})$ dan sebuah basis tak bersyarat untuk pelbagai ruang fungsi. Jika suatu basis wavelet dinyatakan dengan $\{\psi_{j,k} | j, k \in \mathbf{z}\}$, sebuah fungsi f dalam $\mathcal{F}$ dapat direpresentasikan sebagai $f = \sum_j \sum_k \gamma_{j,k}\psi_{j,k}$.

[1] hasil perpangkatan dari dua



Gambar 3.1: Wavelet terlokalisasi dalam frekuensi dan waktu (*www.amara.com*).

2. Basis wavelet bisa bersifat orthogonal $\int \psi(t)\psi(t-k)dt = \delta(k)$ atau biorthogonal (mempunyai dual), $\int \psi(t)\tilde{\psi}(t-k)dt = \delta(k)$.

3. Wavelet dan dualnya terlokalisasi dalam ruangwaktu dan frekuensi, bahkan sebagian wavelet berada dalam interval yang tertutup dan terbatas (*compactly supported*).

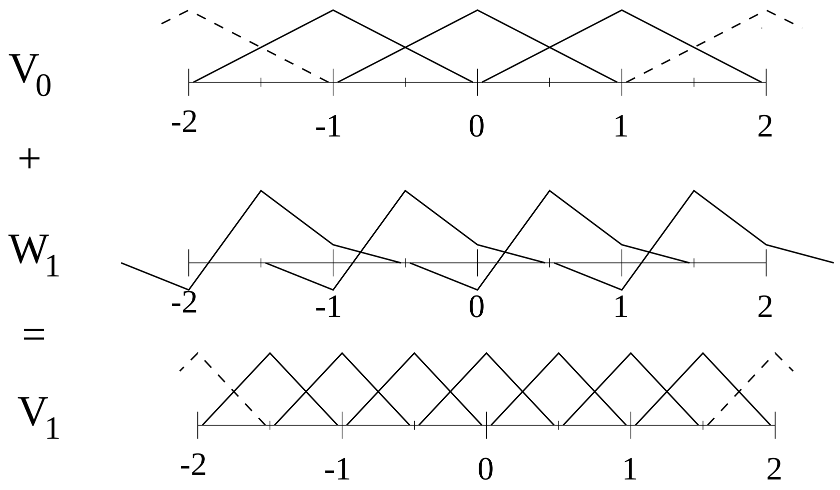
4. Wavelet memenuhi syarat analisis multiresolusi (MRA) (subbab 3.1.2) sehingga memungkinkan untuk dilakukan transformasi wavelet cepat dengan kompleksitas O(N).

   Sifat-sifat ini membuat wavelet sebagai basis yang optimal untuk analisis, representasi, kompresi, estimasi dan pemulihan fungsi dalam $\mathcal{F}$. Untuk sebagian besar jenis kelas fungsi, informasi penting yang terkandung dalam fungsi dapat diberikan hanya oleh sebagian kecil koefisien-koefisien wavelet.

3.1.2 Analisis Multiresolusi

Dekomposisi skala-2

Gambar (3.2) mengilustrasikan konsep dekomposisi skala-2 untuk fungsi dalam satu dimensi [17]. Baris pertama menunjukkan sebuah set basis yang



Gambar 3.2: Dekomposisi skala-2 untuk fungsi satu dimensi [17].

mampu melukiskan fungsi-fungsi yang berfluktuasi dalam skala panjang lebih besar dari satuan jarak. Baris ini terdiri atas sebuah fungsi tunggal $\mathsf{s}(x)$ yang diperbanyak, dimisalkan dalam bentuk fungsi segitiga yang ditranslasikan setiap satuan jarak sepanjang sumbu riil. Ruang linier dari fungsi-fungsi yang dapat dituliskan dalam basis ini disebut V_0. Cara termudah untuk menggandakan resolusi dari basis ini adalah memampatkannya dengan faktor dua. Ini akan mengurangi skala horisontal dari fungsi dan titik-titik kisi di mana fungsi berada seperti tampak dalam baris terakhir pada gambar. Ruang yang dibentang oleh basis yang lebih halus ini disebut V_1. Dekomposisi skala-2 membangun V_1 dari V_0 dengan cara menambahkan pada V_0 sebuah himpunan fungsi basis yang baru $\mathsf{d}(x)$ yang disebut fungsi detil atau *wavelet.* wavelet ini diilustrasikan dalam baris tengah pada gambar. Ruang detil yang dibentang oleh wavelet ini disebut W_1. Agar basis gabungan ini setara dengan basis yang dimampatkan, setiap fungsi dalam V_1 harus merupakan penjumlahan dari satu fungsi dalam V_0 dan satu dalam W_1, atau dalam bahasa subruang linier $V_0 \oplus W_1 = V_1$.

Syarat ini banyak menentukan bentuk dari fungsi-fungsi basis. Pertama, ruang $V_0 \oplus W_1$ dan V_1 harus mempunyai dimensi yang sama, dan jumlah fungsi basis yang setara, sehingga harus ada satu fungsi detil untuk setiap titik pada kisi yang lebih halus yang tidak ada pada kisi yang kasar. (Pada gambar, titik-titik ini adalah titik kelipatan ganjil dari seperdua). Selanjutnya fungsi-fungsi basis untuk V_0 dan W_1 harus berada pada ruang V_1. Ini menghasilkan *persamaan penghalusan* yang menghubungkan fungsi basis yang sama pada dua skala yang berbeda. Akibatnya fungsi basis awal dapat dituliskan sebagai kombinasi linier dari translasi dan dilatasi dirinya sendiri,

$$s(x) = \sum c_n s(2x - n).$$

Koefisien c_n adalah koefisien scaling dan fungsi-fungsi yang memenuhi syarat ini disebut *fungsi scaling*. Terakhir, $V_0 \oplus W_1 = V_1$ menyebabkan fungsi detil juga merupakan kombinasi linier dari translasi dan dilatasi fungsi scaling.

$$d(x) = \sum d_n s(2x - n),$$

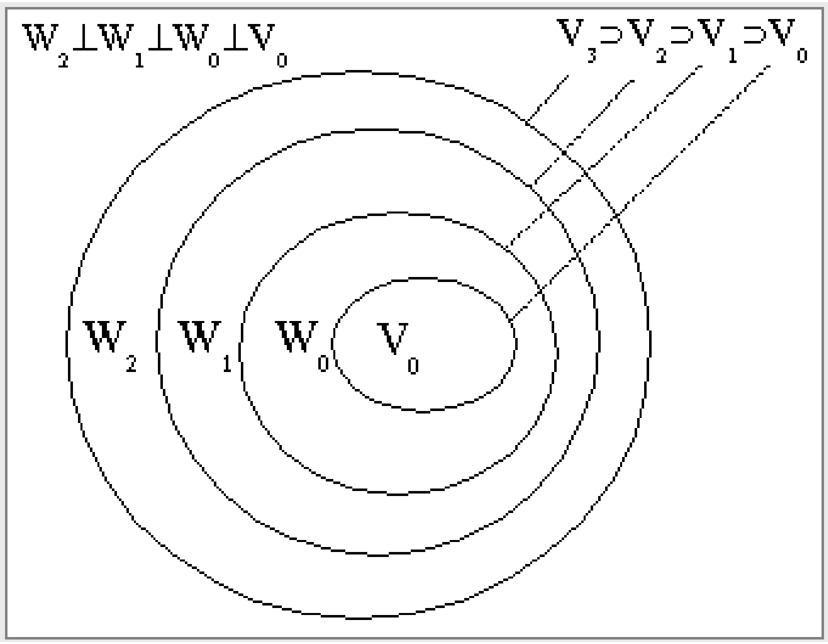
dimana d_n disebut koefisien detil atau koefisien wavelet. Hal-hal ini menunjukkan bahwa semua fungsi pada V_1 dapat dituliskan dalam suku-suku kombinasi linier dari fungsi scaling dan fungsi wavelet.

Analisis Multiresolusi

Apabila resolusi fungsi scaling terus dipertinggi dengan faktor 2, maka akan diperoleh ruang-ruang fungsi $V_0, V_1 \ldots V_N$ dengan kisi-kisinya $C_0, C_1 \ldots C_N$. Kemudian didefinisikan ruang detil W_{Q+1} yang memberikan $V_Q \oplus W_{Q+1} = V_{Q+1}$. Basis untuk W_{Q+1} terdiri atas fungsi-fungsi detil yang dimampatkan dan terletak pada titik-titik kisi $D_{Q+1} \equiv C_{Q+1} - C_Q$ yang muncul pada C_{Q+1} tapi tidak pada C_Q. Dekomposisi skala-2 yang berulang ini dapat menghasilkan ruang V_N yang beresolusi tinggi, $V_N = V_0 \oplus W_1 \ldots \oplus W_N$, yang memberikan tingkat kedetilan secara hirarkis dari sebuah fungsi basis. Ide ini dapat diperluas untuk dimensi yang lebih besar lagi.

3.2 Skema Lifting

Pada tugas akhir ini wavelet-multivektor diperkenalkan melalui wavelet generasi kedua yang merupakan perampatan dari wavelet generasi pertama. Wavelet generasi kedua ini tidak harus merupakan translasi dan dilatasi dari



Gambar 3.3: Ruang vektor dari fungsi wavelet dan fungsi scaling [18].

sebuah fungsi. Penyusunannya juga tidak bergantung pada transformasi Fourier, yang berarti kita dapat menyusun wavelet hanya dari domain ruang ataupun waktu.

Cara menyusun wavelet generasi kedua disebut dengan lifting [19, 20, 21, 22]. Lifting ini merupakan cara yang mudah, luwes dan sederhana untuk menyusun wavelet. Wavelet generasi kedua ini memiliki semua sifat utama dan keuntungan dari wavelet generasi pertama ditambah keuntungannya sendiri. Keuntungan metoda lifting ini antara lain adalah transformasi yang sangat cepat, hanya menggunakan sedikit memori akibat *in-place calculation*, transformasi dari bilangan cacah ke bilangan cacah, dan kemampuan menyusun wavelet dalam sembarang dimensi dan kisi. Kelebihan yang lain lagi, wavelet dapat diadaptasikan untuk data dalam sebuah interval, lengkungan, permukaan, dan pendekatan dengan bobot, serta pencacahan tak beraturan.

Gambar 3.4: Struktur dari transformasi wavelet: pemisahan menjadi rataan dan selisih secara rekursif [19].

3.2.1 Contoh sederhana: Wavelet Haar

Misalkan ada dua buah angka a dan b yang bersebelahan pada sebuah deret sampel. Korelasi yang ada antara a dan b dimanipulasi dengan menggunakan transformasi linier sederhana untuk menggantikan a dan b dengan *rataan* s dan *selisih* d,

$$s = \frac{a+b}{2} \tag{3.2}$$

$$d = b - a. \tag{3.3}$$

Idenya adalah jika a dan b mempunyai korelasi yang besar, maka nilai harapan mutlak dari d akan kecil. Jika $a = b$, maka $d = 0$. Tidak ada informasi yang hilang, karena dengan diberikan s dan d selalu dapat ditemukan a dan b,

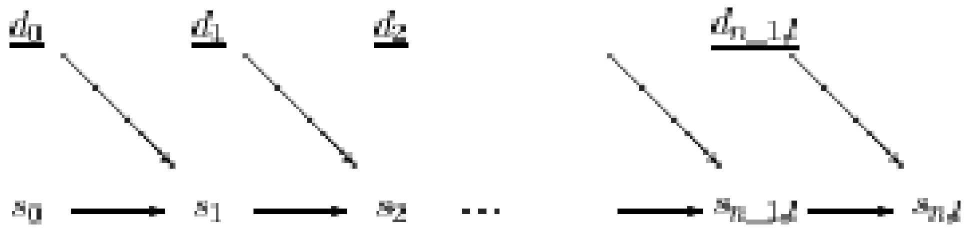
$$\begin{aligned} a &= s - d/2 \\ b &= s + d/2 \end{aligned}$$

yang dapat dilakukan dengan sebuah inversi matriks 2×2.

Transformasi sederhana ini adalah kunci dari transformasi wavelet Haar. Perhatikan sebuah himpunan s_n dengan 2^n sample bernilai $s_{n,l}$:

$$\{s_n = s_{n,l} | 0 \leq l < 2^n\}.$$

Dengan menerapkan transformasi wavelet Haar pada setiap pasang $a = s_{2l}$ dan $b = s_{2l+1}$ $(l = 0, \ldots 2^{n-1})$ yang menghasilkan 2^{n-1} pasangan $s_{n-1,l}$ dan



Gambar 3.5: Struktur dari transformasi balik wavelet: penggabungan rataan dan selisih secara rekursif [19].

$d_{n-1,l}$,

$$\begin{aligned} s_{n-1,l} &= \frac{s_{n,2l} + s_{n,2l+1}}{2} \\ d_{n-1,l} &= s_{n,2l+1} - s_{n,2l} \end{aligned}$$

Masukan s_n yang mempunyai 2^n sampel dipisah menjadi dua bagian, yaitu s_{n-1} dengan 2^{n-1} rataan $s_{n-1,l}$ dan d_{n-1} dengan 2^{n-1} selisih $d_{n-1,l}$. Dengan diberikan rataan s_{n-1} dan selisih d_{n-1} himpunan awal s_n dapat ditemukan kembali.

Rataan s_{n-1} dapat dianggap sebagai sebuah representasi kasar dari s_n dan selisih d_{n-1} sebagai informasi yang dibutuhkan untuk kembali ke himpunan asal s_n. Jika himpunan asal mempunyai koherensi lokal, misalnya sampel sebuah fungsi yang landai, maka representasi kasar akan menyerupai himpunan asal dan detil sangat kecil sehingga bisa direpresentasikan secara efisien.

Transformasi yang sama dapat dikenakan pada s_{n-1}. Dengan melakukan perataan dan pengurangan, s_{n-1} dapat dipisah menjadi dua bagian, yaitu s_{n-2} dan d_{n-2} yang masing-masing mengandung 2^{n-2} sampel. Hal ini bisa diteruskan selama n kali sampai seluruh sampel habis, lihat gambar (3.4). Dengan transformasi Haar ini diperoleh n detil d_j dengan $0 \leq j \leq n-1$, masing-masing mempunyai 2^j koefisien, dan satu koefisien pada skala yang paling kasar. Himpunan terkasar s_0 hanya mengandung satu sampel $s_{0,0}$ yang merupakan rata-rata dari keseluruhan himpunan asal, yaitu komponen arus searah atau frekuensi nol dari himpunan.

Dengan menggunakan transformasi balik yang berawal dari s_0 dan d_j untuk $0 \leq j < n$, s_n dapat diperoleh kembali. Perhatikan bahwa jumlah keseluruhan dari koefisien sesudah transformasi adalah 1 untuk s_0 ditambah 2^j untuk setiap d_j menghasilkan

$$1 + \sum_{j=0}^{n-1} d^j = 2^n,$$

yang sama dengan jumlah sampel himpunan asal. Transformasi Haar ini dapat dianggap sebagai perkalian sebuah matriks $N \times N (N = 2^n)$ terhadap sinyal s_n. Biaya komputasi dari transformasi ini hanya sebanding dengan N, berarti kompleksitasnya O(N).

Sekarang transformasi Haar ini akan dilihat dari sudut pandang baru, yang keuntungannya terletak pada cara menghitung rataan dan selisih dari a dan b. Asumsikan semua perhitungan ingin dilakukan tanpa membutuhkan memori tambahan, dengan menimpa lokasi a dan b dengan nilai s dan d hasil perhitungan. Hal ini tidak bisa dilakukan dengan rumus (3.2). Oleh karena itu, cara yang baru dimplementasikan dalam dua langkah. Pertama hanya dihitung selisih:

$$d = b - a,$$

dan disimpan pada lokasi untuk b. Karena b lenyap, selanjutnya a dan nilai d yang baru digunakan untuk menghitung rataan s sebagai

$$s = a + d/2,$$

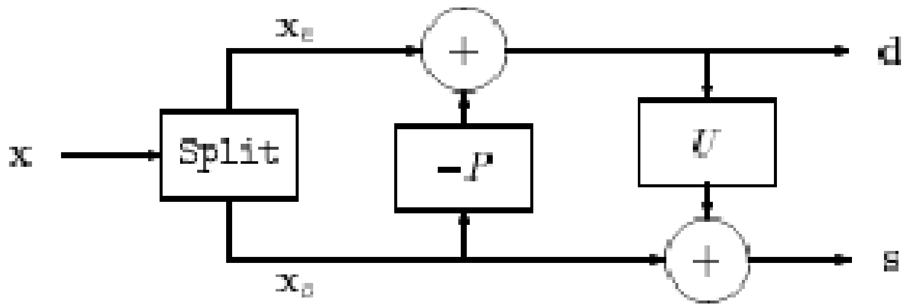
yang kemudian disimpan pada lokasi a. Ini memberikan hasil yang sama karena $a + d/2 = a + (b - a)/2 = (a + b)/2$.

Sebuah implementasi dari algoritma di atas adalah

$$\mathtt{b} - = \mathtt{a}; \qquad \mathtt{a} + = \mathtt{b}/2.$$

Keuntungannya adalah komputasi berlangsung di tempat tanpa tambahan memori sementara, dan transformasi balik dapat dilakukan tanpa perlu mecari invers matriks. Transformasi balik dilakukan dengan membalik urutan kode dan operator,

$$\mathtt{a} - = \mathtt{b}/2; \qquad \mathtt{b} + = \mathtt{a},$$



Gambar 3.6: Diagram blok untuk skema lifting [21].

akan mengembalikan semuanya pada nilai dan posisi semula. Implementasi seperti ini adalah ciri umum dari skema lifting.

3.2.2 Ide dasar

Skema lifting terdiri dari tiga langkah, yaitu *split, predict, dan update.* Misalkan sebuah sinyal $\mathsf{x} = (x_k)_{k\in\mathbf{Z}}$ dengan $x_k \in \mathbf{R}$. Kemudian sinyal dipisah menjadi dua bagian yang saling lepas, yaitu: bagian berindeks genap $\mathsf{x}_e = (x_{2k})_{k\in\mathbf{Z}}$ atau "genap" saja dan bagian berindeks ganjil $\mathsf{x}_o = (x_{2k+1})_{k\in\mathbf{Z}}$, atau "ganjil". Biasanya kedua himpunan ini cenderung berkorelasi. Jadi apabila diberikan satu himpunan, misal genap, dapat dibuat prediktor P yang ampuh untuk himpunan ganjil. Tentu saja prediktor **P** tidak harus benar, sehingga selisih atau detil d perlu direkam,

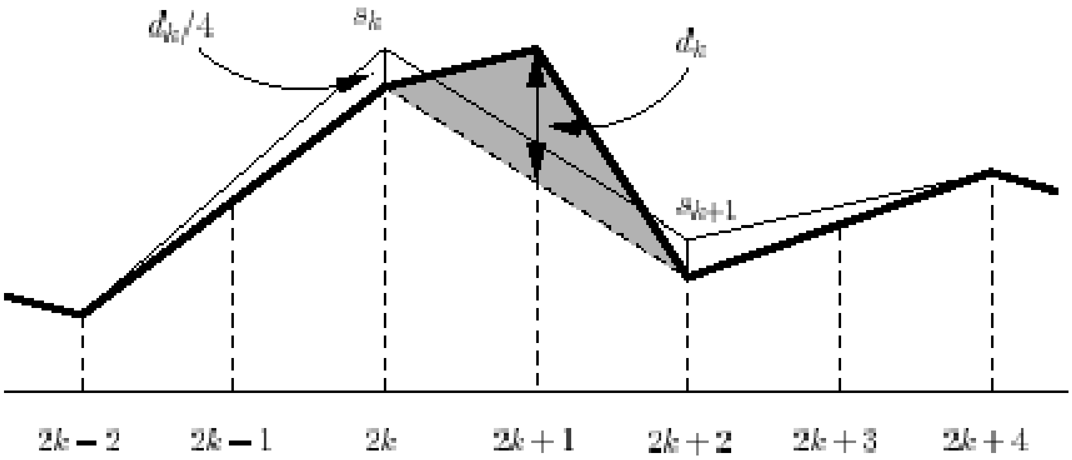
$$\mathsf{d} = \mathsf{x}_e - \mathsf{P}(\mathsf{x}_o).$$

Dengan diberikan d dan ganjil maka, genap dapat ditemukan sebagai

$$\mathsf{x}_e = \mathsf{P}(\mathsf{x}_o) + \mathsf{d}.$$

Jika P adalah prediktor yang baik, maka d akan berupa himpunan jarang dari koefisien-koefisien detil atau wavelet yang entropinya lebih kecil dibandingkan x_o. Untuk contoh yang sekarang digunakan prediktor linier. Sebuah prediktor yang mudah untuk sebuah sampel ganjil adalah rata-rata dari dua tetangga genapnya, sehingga koefisien detilnya adalah

$$d_k = x_{2k+1} - (x_{2k} + x_{2k+2})/2.$$



Gambar 3.7: Interpretasi geometrik untuk lifting dengan prediksi linier. Sinyal asal digambarkan tebal. Koefisien wavelet d_k adalah selisih dari sebuah sampel ganjil dan rata-rata dari kedua tetangga genapnya. Ini membuat hilang area abu-abu seluas $d_k/2$. Untuk menjaga rata-rata berjalan, area ini harus didistribusikan merata ke lokasi genap yang menghasilkan sinyal linier s_k yang lebih kasar digambar dengan garis tipis. Karena skala kasar dua kali skala halus dan kedua lokasi genap ikut terpengaruh maka sampel genap harus ditambah dengan $d_k/4$. (Untuk kemudahan, asumsikan koefisien wavelet d_k dan d_{k-1} adalah nol) [21].

Deskripsi di atas menunjukkan bahwa jika sinyal asal linier secara lokal, maka koefisien detil nol. Ide ini berhubungan langsung dengan wavelet sebagai berikut. Langkah prediksi dapat menangani sebagian korelasi ruang, tapi dalam wavelet juga diinginkan pemisahan dalam domain frekuensi. Sekarang ini, sudah terdapat transformasi dari $(\mathsf{x}_e, \mathsf{x}_o)$ ke $(\mathsf{x}_e, \mathsf{d})$ yang pemisahan frekuensinya sangat kurang karena x_e diperoleh hanya dengan subsampling. Secara khusus, rata-rata dari x_e tidak sama dengan sampel asal x. Untuk membetulkannya, diusulkan langkah berikutnya yang mengganti genap dengan nilai yang lebih halus $(smooth)$ s menggunakan sebuah operator *update* U yang diterapkan pada detil

$$\mathsf{s} = \mathsf{x}_e + \mathsf{U}(\mathsf{d}).$$

Kemudian x_o dapat dihitung kembali seperti telah dijelaskan sebelumnya.

Untuk contoh di atas, operator U yang menghasilkan rata-rata berjalan, sehingga mengurangi aliasing, diberikan oleh

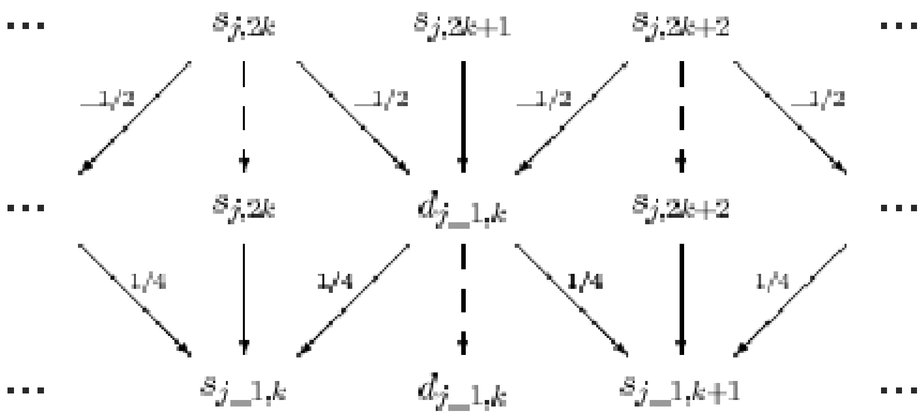
$$s_k = x_2 k + (d_{k-1} + d_k)/4,$$

seperti tampak pada gambar (3.7) Dalam kerangka wavelet, proses lifting di atas setara dengan transformasi wavelet biorthogonal(2,2) dari Cohen-Daubechies-Feauveau.

Apabila langkah *split* diterapkan lagi pada $\mathsf{x_e}$ yang telah diperoleh, kemudian dilakukan langkah *predict* dan *update*, begitu seterusnya, maka akan didapatkan tingkat-tingkat detil sinyal dari yang terhalus sampai terkasar. Jika sinyal asal (tingkat detil sinyal terhalus) disebut s_n dan sinyal terkasar adalah s_0, maka algoritma untuk skema liftingnya adalah:

$$\begin{array}{l}
\texttt{for j} = \texttt{n} - \texttt{1 down to 1} \; \{ \\
\qquad \begin{array}{rcl}
(s_{j-1}, d_{j-1}) & := & Split(s_{j+1}) \\
s_{j-1} & -= & \mathsf{P}(s_j) \\
d_{j-1} & += & \mathsf{U}(d_j) \\
\} & &
\end{array}
\end{array}$$

Transformasi baliknya diperoleh dengan membalik urutan kode dan me-



Gambar 3.8: Skema lifting tidak memerlukan memori tambahan karena melakukan perhitungan ditempat. Untuk sinyal $v[k], k = 0 \ldots 2^n - 1$ koefisien $s_{j,k} = v[2^j]$ dan $d_{j,k} = v[2^{j-1} + 2^j k]$ di mana sinyal asal adalah v_n [19].

rubah tanda saja

$$
\begin{aligned}
&\texttt{for j} = \texttt{n} - \texttt{1 down to 1} \ \{ \\
&\qquad d_{j-1} \ += \ \mathsf{U}(d_j) \\
&\qquad s_{j-1} \ -= \ \mathsf{P}(s_j) \\
&\qquad (s_{j+1}) \ := \ \mathit{Join}(s_{j-1}, d_{j-1}) \\
&\qquad \}
\end{aligned}
$$

Pada saat ini tidak disebutkan apa yang seharusnya terjadi pada kedua batas sinyal, karena sinyal diasumsikan periodik atau tak terbatas. Pembahasan mengenai wavelet dalam interval dapat dilihat di [19]. Skema lifting juga berlaku untuk menyusun keluarga wavelet orde berapapun dalam sembarang kisi dan sembarang dimensi [22]. Implementasi untuk sinyal dalam 1D dan 2D dibahas dalam [23].

3.3 Wavelet-multivektor

Selama ini kita mengenal bilangan riil dan bilangan kompleks yang merupakan penjumlahan dari bilangan riil dan imajiner. Bilangan-bilangan ini selalu digunakan dalam aplikasi-aplikasi matematika, fisika, dan rekayasa. Artinya bisa terdapat besaran skalar dan vektor yang bernilai riil ataupun kompleks meskipun keduanya tidak bisa dijumlahkan. Dalam fisika lanjut seperti mekanika kuantum, ditemukan sistem-sistem aljabar baru seperti aljabar Pauli dan aljabar Dirac, karena aljabar linier biasa tidak cukup mampu untuk menyelesaikan persoalan-persoalan yang ada. Oleh karena itu, banyak diciptakan sistem matematika baru sesuai dengan persoalan yang sedang dihadapi.

Dalam hal ini, aljabar geometrik merupakan alternatif yang sangat baik sebagai suatu sistem matematika untuk melukiskan sistem-sistem fisis [5]. Alasannya adalah:

1. Ia mempunyai notasi yang ringkas dan baik, walau belum ada pembakuan.
2. Aljabar geometrik merupakan perampatan dari aljabar-aljabar yang sudah ada.
3. Objek dan operator dalam aljabar geometrik mempunyai interpretasi geometris.
4. Banyak besaran-besaran fisis yang bisa direpresentasikan dengan baik dalam multivektor.

Sebenarnya multivektor adalah sebuah bilangan, biasa dikenal dengan bilangan Clifford (*Clifford numbers*) yang bernilai multivektor sebagaimana halnya bilangan kompleks yang bernilai kompleks.

Pada kisi 1D[2], apabila terdapat himpunan

$$\mathsf{x} = \{x_k \in \mathbf{R} | k \in \mathbf{z}\} \tag{3.4}$$

[2]istilah vektor tidak digunakan, karena sudah didefinisikan sebagai segmen garis berarah yang elemen-elemennya bilangan riil

maka dapat dicari transformasi waveletnya dengan transformasi biasa maupun skema lifting. Tapi apabila yang ada adalah himpunan bernilai

$$\mathsf{A} = \{A_k \in \mathcal{G}_n | k \in \mathbf{z}\} \tag{3.5}$$

maka cara yang alami untuk mencari representasi dalam bentuk wavelet adalah dengan skema lifting, karena langkah *predict* dan *update* adalah operasi linier dan ruang vektor juga merupakan ruang linier. Jadi yang perlu dilakukan tinggal mengganti bilangan riil dengan multivektor (bilangan Clifford). Untuk kisi-kisi pada dimensi yang lebih tinggi

$$\mathsf{A} = \{A_k \in \mathcal{G}_n | k \in \mathbf{z}^d\} \tag{3.6}$$

dimana d adalah dimensinya, lifting juga dapat dipergunakan [22].

Sekarang fokus beralih pada aplikasi wavelet multivektor dalam aljabar ruang waktu(STA), yang merepresentasikan besaran-besaran fisis dan bukan sekedar matematis. Contoh yang dipergunakan sekarang ini adalah medan elektromagnetik[3].

Misalkan terdapat sebuah sumber titik dari medan elektromagnetik yang dinyatakan dalam vektor ruangwaktu J dan menyebabkan dua jenis sumber

$$\rho = J \cdot \gamma_0, \qquad \boldsymbol{J} = J \wedge \gamma_0 \tag{3.7}$$

dimana γ_0 adalah kerangka pengamat. Kemudian kita membentuk

$$J\gamma_0 = \rho + \boldsymbol{J} \tag{3.8}$$

yang merupakan pemisahan ruang waktu, akibatnya J diproyeksikan pada ruang relatif 3D. Apabila ada himpunan $\{J\gamma_0\}$ dalam kisi waktu yang menyatakan perubahan nilai sumber terhadap satuan waktu

$$\{(J\gamma_0)_t \in \mathcal{G}_3 | t \in \mathbf{Z}\}, \tag{3.9}$$

dapat dibentuk wavelet yang merupakan kuantisasi dari medan sumber sekaligus mengkarakterisasi medan pada titik tersebut. Wavelet 3D yang terbentuk

[3] penurunannya tidak dituliskan karena harus memahami terlebih dahulu kalkulus geometrik

mempunyai sifat-sifat geometris seperti, kerapatan dan arah. Wavelet yang didapat tentu saja bergantung pada kerangka pengamat.

Medan elektromagnetik yang dihasilkan berasal dari empat persamaan Maxwell yang digabung menjadi satu persamaan dengan kalkulus geometrik

$$\nabla F = J = \nabla \cdot F + \nabla \wedge F. \tag{3.10}$$

Medan elektromagnetiknya dituliskan dalam bentuk bivektor Faraday

$$F = \frac{1}{2} F^{\mu\nu} \gamma_\mu \wedge \gamma_{nu} \qquad \mu, \nu = 0, 1, 2, 3. \tag{3.11}$$

Apabila F diproyeksikan ke ruang 3D dalam kerangka pengamat γ_0 maka ia dapat ditulis sebagai

$$F = \boldsymbol{E} + I\boldsymbol{B} \tag{3.12}$$

di mana $\boldsymbol{E}$ dan $\boldsymbol{B}$ adalah medan elektrik dan medan magnetik yang diberikan oleh

$$\boldsymbol{E} = E^k \boldsymbol{\sigma}_k = \frac{1}{2}(F - \gamma_0 F \gamma_0) \tag{3.13}$$

$$I\boldsymbol{B} = B^k \boldsymbol{\sigma}_k = \frac{1}{2}(F + \gamma_0 F \gamma_0) \tag{3.14}$$

Persamaan (3.12) yang berupa penjumlahan dari vektor relatif dengan bivektor relatif menjelaskan kegunaan bilangan kompleks dalam medan elektromagnetik. Apabila terdapat F pada satu titik dalam ruang yang nilainya berubah terhadap waktu, maka akan didapatkan himpunan

$$\mathsf{F} = \{F_t \in \mathcal{G}_3 | t \in \mathbf{Z}\} \tag{3.15}$$

yang representasi waveletnya mempunyai sifat arah serta direktivitas dalam ruang. Karena wavelet adalah sebuah fungsi basis dalam pelbagai ruang fungsi, maka wavelet-multivektor juga menjadi fungsi basis dalam ruang aljabar geometrik $\mathcal{G}_n$ yang bernilai multivektor. Koefisien scaling dan wavelet yang diperoleh juga bernilai multivektor.

Bab 4

Implementasi API Aljabar Geometrik

4.1 API dan Java

API (*Application Programming Interface*) adalah satu set kelas berisi tipe-tipe data baru yang membentuk antarmuka untuk digunakan oleh pemrogram klien sehingga aplikasi bisa dikembangkan dengan cepat dan mudah.

4.1.1 Fungsi dan Syarat dari API

Fungsi API yang diinginkan:

1. dapat merepresentasikan elemen-elemen geometris dalam aljabar geometrik

2. dapat melakukan operasi-operasi dasar seperti penjumlahan, perkalian dengan skalar, perkalian dalam, perkalian luar dan perkalian geometrik dari semua elemen-elemen termasuk multivektor

3. dapat melakukan operasi transformasi linier seperti refleksi dan rotasi serta operasi proyeksi

Manfaat dari API ini adalah sebagai pustaka dasar untuk pengembangan aplikasi-aplikasi yang berbasis aljabar geometrik dalam bidang fisika, rekayasa dan komputasi geometrik, sehingga operasi-operasi dasar tidak perlu lagi dibuat.

Syarat API:

1. mudah digunakan berarti dokumentasi kelas dan cara penggunaan diberikan dengan lengkap

2. portabilitas tinggi, yakni dapat digunakan dalam multiplatform sehingga tidak tergantung pada perangkat keras maupun sistem operasi

3. API difokuskan pada lapis arsitektur inti yaitu *application logic*

4. realibilitas setiap *method* yang ada harus benar sesuai dengan fungsinya, sudah diuji dan bisa diandalkan.

4.1.2 Java

Dalam pembuatan API untuk aljabar geometrik digunakan bahasa pemrograman Java [24] dengan alasan portabilitas, berorientasi objek, dan arsitektur yang kokoh dan lengkap. Portabilitas tinggi memungkinkan pemrogram untuk menulis kode sumber hanya sekali saja dan programnya dapat dijalankan di mesin manapun mengingat Java merupakan bahasa interpreter dimana setiap mesin diberikan interpreternya masing-masing yang disebut JVM (*Java Virtual Machine*). Sebagai bahasa yang berorientasi objek, program memang dirancang untuk dapat terus dikembangkan dengan mudah, artinya kemampuan pakai-ulang dari kode sumber sangatlah besar. Ini akibat dari program yang terdiri dari unit-unit kecil yang disebut kelas (*class*) yang mempunyai tanggung jawab masing-masing dan saling berinteraksi dengan serasi membentuk satu sistem. Arsitektur yang kokoh dan lengkap merupakan ciri bahasa modern yang dirancang secara *top-down*, belajar dari kelebihan dan kekurangan bahasa-bahasa pendahulunya. Java mempunyai API (*Application Programming Interface*) yang berisi perpustakaan kelas yang sangat lengkap dan selalu diperbaharui.

subruang linier	ruang vektor
pseudoscalar	rotor
versor	ruang 2D
ruang 3D	vektor
bivektor	trivektor
multivektor	pseudovektor

Tabel 4.1: Calon-calon konsep dalam aljabar geometrik.

4.2 Analisis dan perancangan berorientasi objek

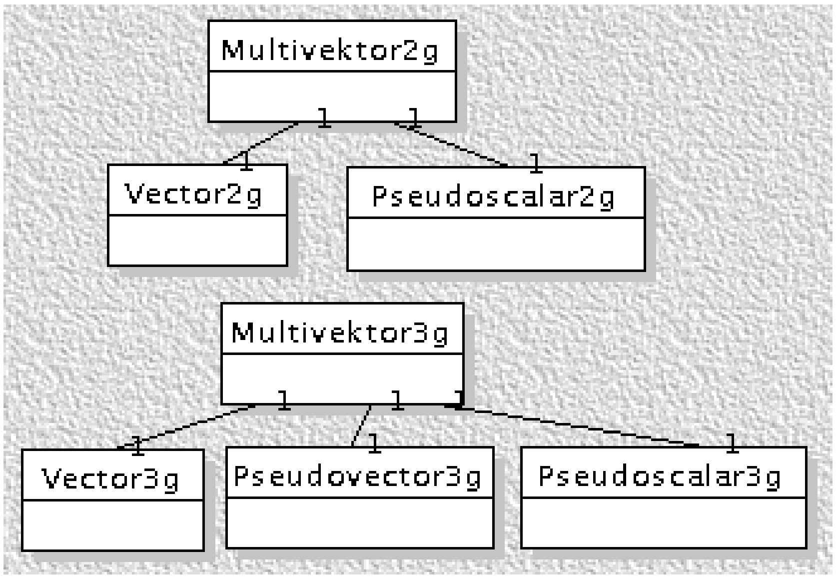
Perumusan masalah berorientasi objek dimulai dengan pembuatan model konseptual yang nantinya akan diimplementasikan dalam kelas [25]. Sebuah konsep adalah sebuah ide, sesuatu atau objek. Sebuah konsep mempunyai simbol, kata-kata atau gambar yang merepresentasikan konsep itu sendiri. Ia bisa mempunyai definisi dan juga dapat diperluas dengan contoh-contoh.

Dalam ruang 2D pseudoscalarnya merupakan bivektor sedangkan dalam ruang 3D berupa trivektor. Pseudoscalar merupakan konsep yang ada di setiap dimensi dan banyak memudahkan perhitungan, oleh karena itu ia diambil sebagai elemen tertinggi dari sebuah multivektor. Multivektor adalah objek yang berfungsi mendefinisikan ruang yang ditempati sehingga untuk saat ini. Kita mempunyai Multivektor2D untuk ruang dan Multivektor3D untuk ruang 3D.

Dalam perhitungan pada ruang 3D lebih mudah menggunakan representasi pseudovektor daripada representasi bivektor. Ini juga akibat sifat simetri dari subruang linier.

Rotor adalah sebuah versor, sedangkan versor sendiri adalah hasil perkalian dari vektor-vektor yang menghasilkan multivektor. Oleh karena itu ia belum perlu direpresentasikan sebagai suatu objek, melainkan hanya sebagai operasi yang dilakukan pada suatu objek.

Akhirnya ruang vektor direpresentasikan sebagai penjumlahan dari subruang linier sehingga konsep multivektor merupakan penjumlahan elemen-



Gambar 4.1: Model konseptual awal dari aljabar geometrik dalam UML.

elemen pada subruang linier.

Setelah model konseptual awal dibuat, kegiatan berikut adalah membuat diagram perancangan kelas dengan menambah *attribut* dan *asosiasi* pada kelas-kelas yang sudah ada. Asosiasi antarkelas ini dalam bahasa berorientasi objek disebut dengan *method*. Penambahan *method* dilakukan dengan cara membuat *fungsi tes* (Lampiran A) terlebih dahulu.

Kegunaan fungsi tes ini adalah:

1. untuk mengetahui apa yang sebenarnya ingin diperoleh dari kode yang kita buat

2. untuk mengintegrasikan tes, sehingga mempermudah pencarian kesalahan dan perbaikannya dengan cepat

3. untuk memastikan bahwa program yang dibuat bisa diandalkan dan benar-benar berjalan dengan baik

4. untuk memberikan contoh bagaimana menggunakan kelas-kelas yang sudah ada.

Soal nomor 1 membahas tentang perkalian luar vektor pada bidang. Dari fungsi tes yang dibuat harus ditambahkan konstruktor dan sejumlah *method*, seperti: wedge(), area(), add(), dot(), dan mul(). Dimulai dengan membangun objek yang paling sederhana yaitu Vector2g kemudian dengan *method overloading* kita memperluas *method* add() dan mul() sehingga dapat berinteraksi dengan objek yang lain dalam ruang 2D. Yang perlu mendapat perhatian di sini adalah perkalian geometrik dari dua buah Vektor2g menghasilkan sebuah Multivektor2g yang terdiri atas skalar dan Pseudocalar2g.

Dari soal nomor 2, *method* yang harus ditambahkan adalah invers(), length(), lengthSquared(), parallel(), perpendicular() dan perkalian dengan Pseudocalar2g. Pada tahap ini harus sudah terbentuk kelas Pseudoscalar2g(). Soal selanjutnya membahas persoalan refleksi yang menggunakan perkalian geometrik dari multivektor terhadap vektor, sehingga dilakukan *method overloading* lagi terhadap *method* mul() pada kelas Multivektor2g dan Vektor2g baru kemudian ditambahkan *method* reflect().

Soal nomor 4 adalah tentang rotasi vektor yang memakai persamaan rotor. Di sini kita menambahkan perkalian geometrik antara dua Multivektor3g dan *method* reverse(). Kemudian pada soal-soal berikutnya fokus dialihkan ke ruang dimensi tiga. Dengan menggunakan kode-kode ruang 2D, ditambahkanlah operasi perkalian dalam, luar dan geometrik. Selanjutnya baru ditambahkan operasi refleksi, rotasi dan proyeksi.

4.3 Implementasi

Setelah menambah attribut dan asosiasi maka diagram perancangan kelas awal telah selesai. Agar kelas-kelas dalam API Java yang sudah ada dapat dimanfaatkan, prinsip komposisi dan pewarisan objek (*object composition and inheritance*) harus digunakan. Komposisi objek berarti sebuah objek terdiri atas objek-objek lain yang saling bekerja sama. Sedang pewarisan bermakna spesialisasi pada satu kelas sehingga menciptakan kelas yang baru. Kelas yang baru ini mempunyai sifat yang sama dengan kelas induknya, ditambah dengan sejumlah *method* baru atau *method* lama yang diperbaharui melalui *method overriding.*

Kelas pada API Java yang digunakan kembali berasal dari API Java3D dalam *package* javax.vecmath.*, yaitu kelas Vector2d, Vector3d dan Tuple3d. Diagram perancangan yang dihasilkan diberikan pada gambar-gambar pada halaman berikutnya sesuai format UML (*Unified Modelling Language*)[1]

Perbandingan dalam perhitungan rotasi dalam 3D

Unjuk kerja API aljabar geometrik untuk kasus rotasi pada ruang 3D dibandingkan terhadap representasi rotasi dengan matriks dan memberikan hasil seperti di bawah ini. Kedua kode di bawah ini sama-sama menjalankan rotasi objek yang sama pada sumbu yang sama sebesar satu putaran penuh. Hanya saja *method* yang digunakan berbeda.

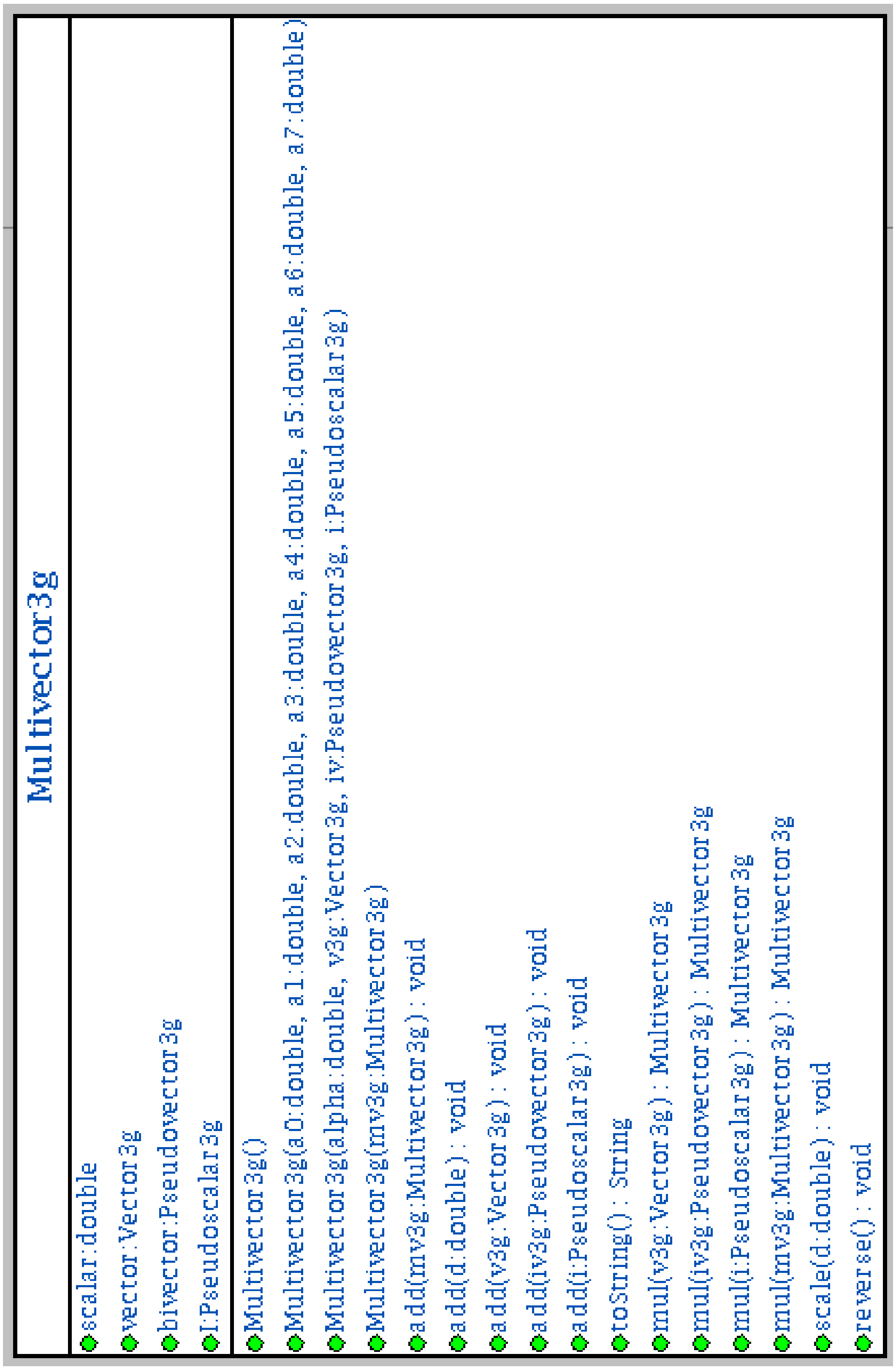
MatriksRotasi.java

```
import javax.vecmath.*;
public class MatriksRotasi {
   public static void main(String[] args) {
      Vector3g asal = new Vector3g(2,1,2);
      Vector3g sumbu = new Vector3g(1.5,2,0);
      Point3d titik = new Point3d();
      double sudut = 2*Math.PI;
      GVector hasil = asal.rotateUseMatrix(sumbu, titik, sudut);
      System.out.println("asal = " + asal);
      System.out.println("sumbu = " + sumbu);
      System.out.println("sudut = " + sudut + " rad");
      System.out.println("Hasil perputarannya adalah " + hasil);
   }
}
```

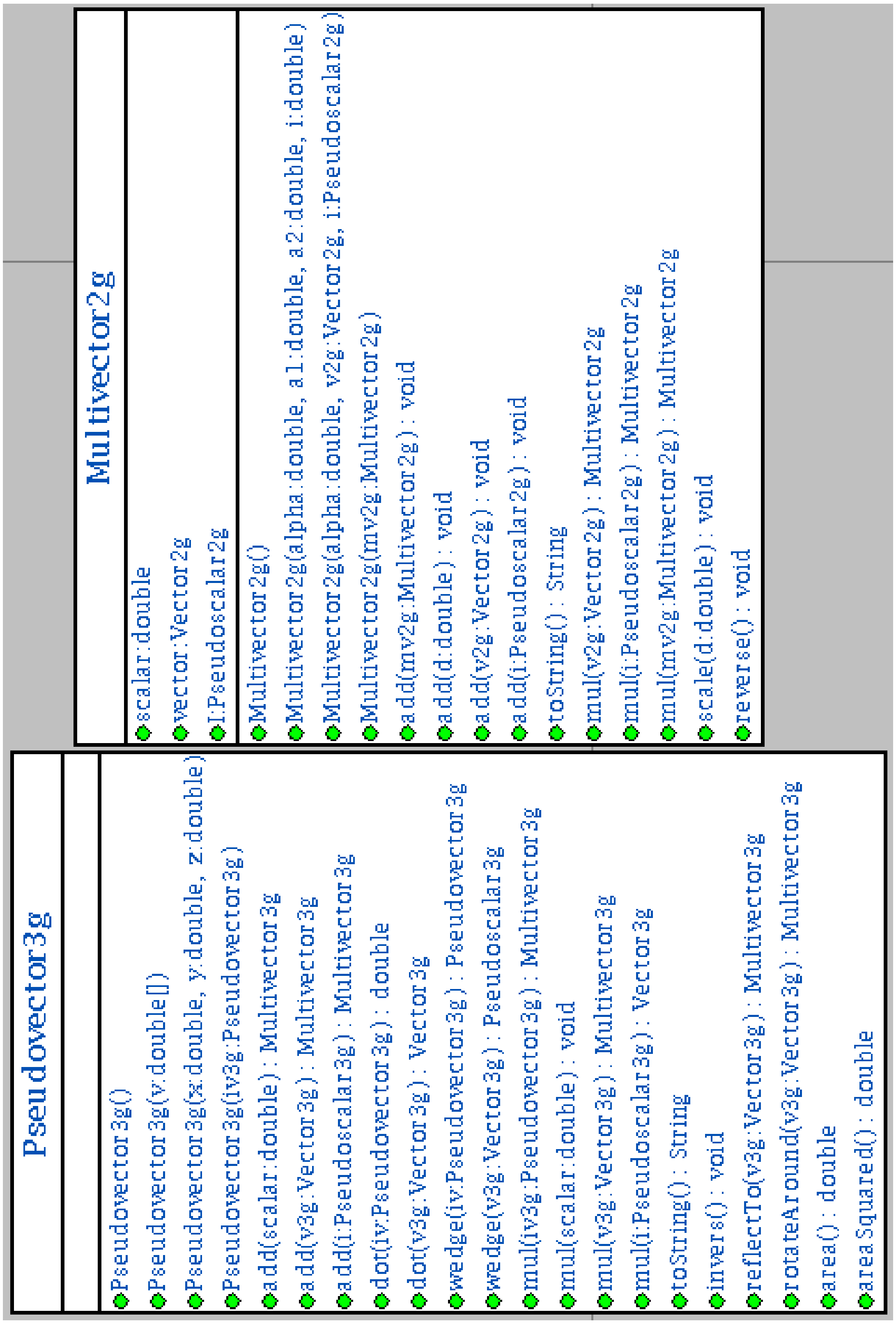
Soal5baru.java

```
public class Soal5baru {
```

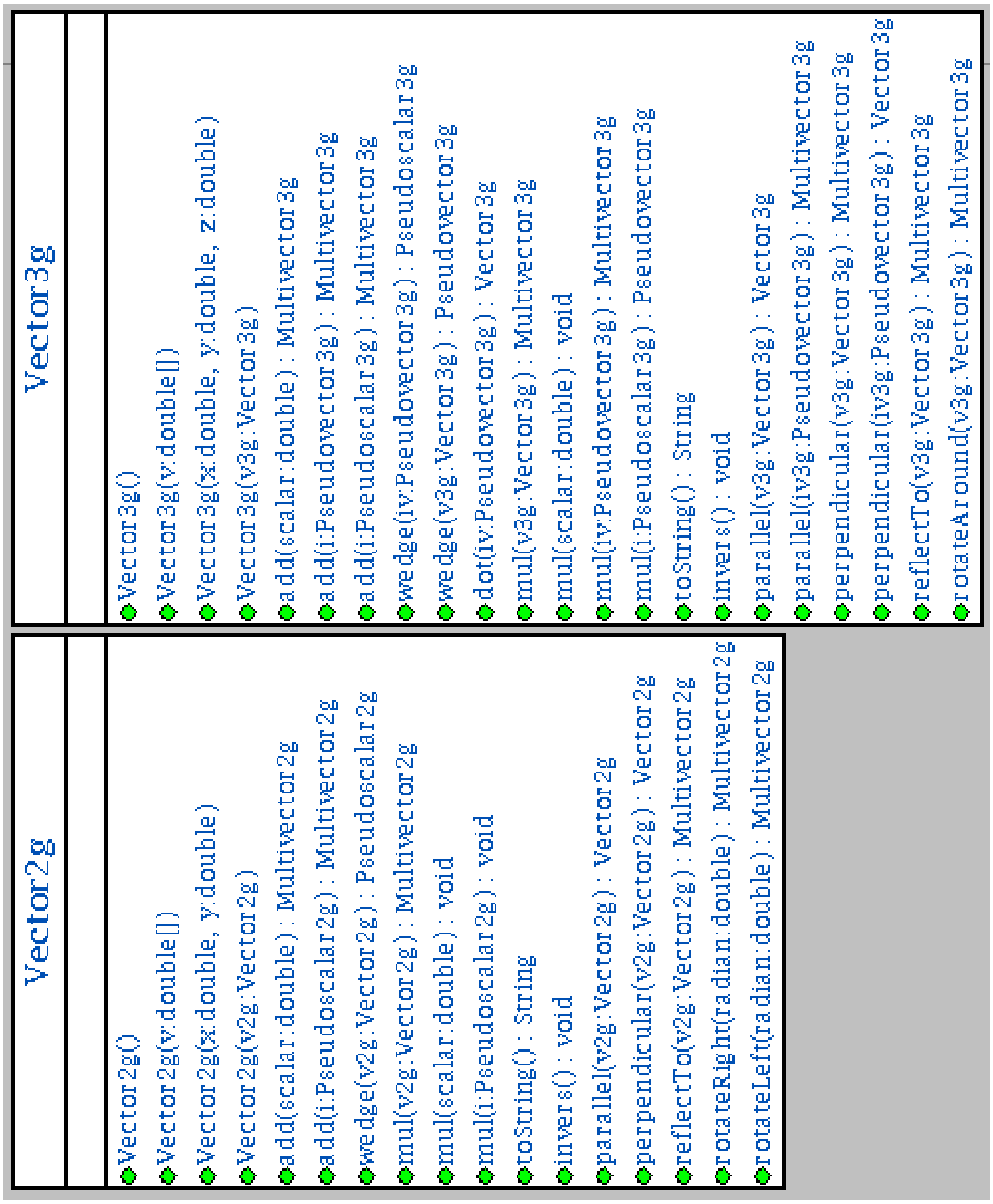
[1]UML merupakan diagram standar dari bahasa pemrograman yang berorientasi objek untuk mempermudah analisis, perancangan, dokumentasi dan komunikasi



Gambar 4.2: Diagram kelas Multivector3g.



Gambar 4.3: Diagram kelas Pseudovector3g dan Multivector2g



Gambar 4.4: Diagram kelas Vector2g dan Vector3g.

Gambar 4.5: Diagram kelas Pseudoscalar2g dan Pseudoscalar3g.

```
   public static void main(String[] args) {
      Vector3g r = new Vector3g(2,1,2);
      Vector3g a = new Vector3g(1.5,2,0);
      Multivector3g RrR = r.rotateAround(a,2*Math.PI);
      System.out.println("r = " + r);
      System.out.println("a = " + a);
      System.out.println("Hasil perputarannya adalah " + RrR);
   }
}
```

Perbandingan kedua *method* diberikan dibawah ini.

rotateAround

```
public Multivector3g rotateAround(Vector3g v3g, double theta) {
long awal = System.currentTimeMillis();
double length = v3g.length();
double ratio = theta /length;
Pseudovector3g temp = new Pseudovector3g(v3g.x, v3g.y, v3g.z);
temp.scale(Math.sin(ratio*length / 2) / length);
Multivector3g R = new Multivector3g(Math.cos(ratio*length / 2),
new Vector3g(), temp, new Pseudoscalar3g());
Multivector3g Rr = R.mul(this);
temp.scale(-1);
Multivector3g Rrev = new Multivector3g(Math.cos(ratio*length / 2),
new Vector3g(), temp, new Pseudoscalar3g());
Multivector3g result = Rr.mul(Rrev);
long akhir = System.currentTimeMillis();
long lama = akhir - awal;
System.out.println("Total rotation time " + lama + "ms");
return result;
}
```

rotateUseMatriks

```
public GVector rotateUseMatrix(Vector3g arah, Point3d titik,
                                    double t) {
long awal = System.currentTimeMillis();
double A,B,C,x,y,z,V,L;
A = arah.x;
B = arah.y;
C = arah.z;
x = titik.x;
y = titik.y;
z = titik.z;
V = Math.sqrt(B*B + C*C);
L = Math.sqrt(A*A + B*B + C*C);
Matrix4d T = new Matrix4d(1, 0, 0, 0,
                                    0, 1, 0, 0,
                                    0, 0, 1, 0,
                                   -x,-y,-z, 1);
          Matrix4d Tinv = new Matrix4d(1, 0, 0, 0,
                                       0, 1, 0, 0,
                                       0, 0, 1, 0,
                                       x, y, z, 1);
          Matrix4d Rx = new Matrix4d(1, 0, 0, 0,
                                    0, C/V, B/V, 0,
                                    0,-B/V, C/V, 0,
                                    0, 0, 0, 1);
          Matrix4d Rxinv = new Matrix4d(1, 0, 0, 0,
                                        0, C/V, -B/V, 0,
                                        0, B/V, C/V, 0,
                                        0, 0, 0, 1);
          Matrix4d Ry = new Matrix4d(V/L, 0, A/L, 0,
                                       0, 1,   0, 0,
                                    -A/L, 0, V/L, 0,
```

```
                                        0, 0,   0, 1);
          Matrix4d Ryinv = new Matrix4d(V/L, 0,-A/L, 0,
                                           0, 1,   0, 0,
                                         A/L, 0, V/L, 0,
                                           0, 0,   0, 1);
    Matrix4d Rz = new Matrix4d(Math.cos(t), Math.sin(t), 0, 0,
                              -Math.sin(t), Math.cos(t), 0, 0,
                                        0,             0, 1, 0,
                                        0,             0, 0, 1);
          Matrix4d TotalTransform = new Matrix4d(Tinv);
          TotalTransform.mul(Rxinv);
          TotalTransform.mul(Ryinv);
          TotalTransform.mul(Rz);
          TotalTransform.mul(Ry);
          TotalTransform.mul(Rx);
          TotalTransform.mul(T);
          GMatrix TotTrans = new GMatrix(4,4);
          TotTrans.set(TotalTransform);
          Vector4d oldvec = new Vector4d(this);
          GVector newvec = new GVector(oldvec);
          GVector result = new GVector(4);
       result.mul(TotTrans, newvec);
long akhir = System.currentTimeMillis();
long lama = akhir - awal;
System.out.println("Total rotation time " + lama + "ms");
return result;
}
```

Unjuk kerja dari keduanya dibandingkan di bawah ini:

```
Script started on Mon Jan 15 09:45:08 2001
[utama@Qolam latexdoc]$ java MatriksRotasi
Total rotation time 99ms
asal = 2.0e1 1.0e2 2.0e3
```

```
sumbu = 1.5e1 2.0e2 0.0e3
sudut = 6.283185307179586 rad
Hasil perputarannya adalah 1.9999999999999996 1.0000000000000002 2.0 0.0
[utama@Qolam latexdoc]$ java Soal5baru
Total rotation time 17ms
r = 2.0e1 1.0e2 2.0e3
a = 1.5e1 2.0e2 0.0e3
Hasil perputarannya adalah  0.0 2.0e1 0.9999999999999998e2
1.9999999999999996e3  0.0Ie1 0.0Ie2 0.0Ie3  -4.930380657631324E-32I
[utama@Qolam latexdoc]$ exit

Script done on Mon Jan 15 09:46:05 2001
```

Hasil di atas menunjukkan bahwa rotasi dalam aljabar geometrik direpresentasikan dalam bentuk yang lebih sederhana. Ini akibat representasi matriks berupa hasil perkalian dari matriks-matrik rotasi pada setiap sumbu dan koordinatnya harus dinyatakan dalam bentuk koordinat homogen sehingga dibutuhkan matriks 4×4 [26]. Aljabar geometrik bisa melakukannya karena bivektor adalah bentuk lain dari *quaternion Hamilton* yang ampuh dalam perhitungan rotasi. Pada akhirnya, perhitungan rotasi dengan menggunakan rotor menjadi lebih cepat jika dibandingkan dengan menggunakan representasi matriks.

Contoh penggunaan API dalam elektromagnetik

```
public class SoalEM {
   public static void main(String[] args) {
      Vector3g E = new Vector3g(1,2,4);
      Vector3g B = new Vector3g(3,5,7);
      Multivector3g F = new Multivector3g(0,E,
                   new Pseudovector3g(B.x,B.y,B.z),
                   new Pseudoscalar3g(0));
      System.out.println("Medan listrik E = " + E);
      System.out.println("Medan magnet B = " + B);
```

```
        System.out.println("F = E + IB = " + F);
        System.out.println("rapat Lagrangian (EE-BB)/2 adalah " +
        (E.lengthSquared()-B.lengthSquared())/2);
        System.out.println("invarian Lorentz E.B = " + E.dot(B));
        Multivector3g lorentz = F.mul(F);
        lorentz.scale(0.5);
        System.out.println("FF/2 = " + lorentz);
        System.out.println("rapat energi (EE+BB)/2 = " +
        (E.lengthSquared()+B.lengthSquared())/2);
        System.out.println("vektor Poynting (E^B)/i = " +
        E.wedge(B).mul(new Pseudoscalar3g(-1)));
        Multivector3g Frev = new Multivector3g(F);
        Frev.reverse();
        Multivector3g energi = Frev.mul(F);
        energi.scale(0.5);
        System.out.println("F'F/2 = " + energi);

    }
}
```

Yang perlu diperhatikan adalah basis vektor yang digunakan selalu relatif terhadap kerangka pengamat sehingga besaran medan F, rapat energi dan vector Poynting merupakan besaran kovarian.

```
Script started on Mon Jan 15 11:51:24 2001
[utama@Qolam latexdoc]$ java SoalEM
Medan listrik E = 1.0e1 2.0e2 4.0e3
Medan magnet B = 3.0e1 5.0e2 7.0e3
F = E + IB =  0.0 1.0e1 2.0e2 4.0e3  3.0Ie1 5.0Ie2 7.0Ie3  0.0I
rapat Lagrangian (EE-BB)/2 adalah -31.0
invarian Lorentz E.B = 41.0
FF/2 =  -31.0 0.0e1 0.0e2 0.0e3  0.0Ie1 0.0Ie2 0.0Ie3  41.0I
rapat energi (EE+BB)/2 = 52.0
vektor Poynting (E^B)/i = -6.0e1 5.0e2 -1.0e3
```

```
F'F/2 =  52.0 6.0e1 -5.0e2 1.0e3  0.0Ie1 0.0Ie2 0.0Ie3  0.0I
[utama@Qolam latexdoc]$ exit

Script done on Mon Jan 15 11:51:39 2001
```

Pengujian API dapat dilihat pada Lampiran A dan Lampiran C

Bab 5

Kesimpulan dan Saran

5.1 Kesimpulan

1. API aljabar geometrik yang portabel dan mampu melakukan perkalian geometrik dari multivektor telah berhasil dibuat.

2. Persoalan rotasi dalam ruang 3D bisa diselesaikan lebih cepat dengan aljabar geometrik dibandingkan dengan transformasi matriks.

3. Aljabar geometrik mempunyai kelebihan dalam memanipulasi persamaan matematika dengan memberikan interpretasi geometris.

4. Wavelet multivektor dapat dibuat dalam aljabar geometrik dengan menggunakan skema lifting.

5.2 Saran

Saran bagi penelitian selanjutnya adalah:

1. Membuat visualisasi dari objek-objek geometris (multivektor) dengan bantuan Java3D.

2. Membuat API skema lifting dalam bahasa Java.

3. Menyusun aplikasi sederhana berbasis aljabar geometrik dengan API yang sudah dibuat.

Daftar Pustaka

[1] Chris J.L. Doran. *Geometric Algebra and Its Applications to Mathematical Physics*, Tesis PhD, Univ. of Cambridge, 1994.

[2] David Hestenes. *New Foundation of Mathematical Physics*, 1998.

[3] S.F. Gull, A.N. Lasenby, C.J.L. Doran. Imaginary numbers are not reals - the geometric algebra of spacetime. *Found. Phys., 23(10) hal. 1295*, 1993.

[4] Anthony Lasenby dan Joan Lasenby. Applications of Geometric Algebra in Physics and Links with Engineering. *Geometric Algebra: a geometric approach to computer vision, neural and quantum computing, robotics and engineering*, Birkhauser, 2000.

[5] J. Lasenby, A.N. Lasenby, C.J.L. Doran. A Unified Mathematical Language for Physics and Engineering in 21st Century. *Phil. Trans. R. Soc Lond. A 358, 21-39*, 2000.

[6] David Hestenes. Old Wine in New Bottles: A new algebraic framework for computational geometry. *Geometric Algebra: a geometric approach to computer vision, neural and quantum computing, robotics and engineering*, Birkhauser, 2000.

[7] Leo Dorst. Honing geometric algebra for its use in the computer sciences. *Geometric Computing with Clifford Algebras*, Springer,In Press..

[8] Anthony M. Lewis. *Geometric Algebra and Covariant Methods in Physics and Cosmology*, Tesis PhD, Univ. of Cambridge, 2000.

[9] C. Doran, A. Lasenby, S.F. Gull, S. Somaroo, A. Challinor. Spacetime Algebra and Electron Physics. *Advances in Imaging and Electron Physics Vol 95 p. 271-386*, Academic Press, 1995.

[10] S. F. Gull, A. N. Lasenby and C. J. L. Doran. Electron Paths, Tunnelling and Diffraction in the Spacetime Algebra *Found. Phys. 23(10), 1329-1356*, 1993.

[11] A.N. Lasenby, C.J.L. Doran dan S.F. Gull. Gravity, Gauge Theory and Geometric Algebra *Phil. Trans. R. Soc Lo. A356 487-582*, 1998

[12] S. S. Somaroo. *Applications of the Geometric Algebra to Relativistic Quantum Theory*, Tesis PhD, Univ. of Cambridge, 1996.

[13] J. Lasenby, W. J. Fitzgerald, C. J. L. Doran dan A. N. Lasenby. New Geometric Methods for Computer Vision *Int. J. Comp. Vision 36(3), p. 191-213*, 1998.

[14] Chris Doran dan Anthony Challinor. Physical Applications of Geometric Algebra. *Lecture Notes of Physical Applications of Geometric Algebra*, Univ. of Cambridge, 2000

[15] Wim Sweldens. The Lifting Scheme: A New Philosophy in Biorthogonal Wavelet Constructions, 1996.

[16] Wim Sweldens. Wavelets: what next? *Proceedings of the IEEE, 84(4):680-685*, 1996.

[17] T.A. Arias dan T.D. Engenes. Beyond Wavelets: Exactness theorems for physical calculations. *Computer Simulation Studies in Condensed Matter Physics XII*, 1999.

[18] C.S. Burrus, R.A. Gopinath, H. Guo. Introduction to wavelets an wavelet transforms: A primer. Prentice-Hall, 1998.

[19] W. Sweldens dan P. Schroder. Building your own wavelet at home. *Wavelets in Computer Graphics page 15-87, ACM SIGGRAPH Course notes*, 1996.

[20] Wim Sweldens. The lifting scheme: A construction of a second generation wavelets. *SIAM J. Math. Anal.,29(2):511-546*, 1997.

[21] I. Dubechies dan W. Sweldens. Factoring wavelet transforms into lifting steps. *J. Fourier Anal. Appl., 4(3):247-269*, 1998.

[22] Jelena Kovacevic dan Wim Sweldens. Wavelet Families of Increasing Order in Arbitrary Dimensions. *IEEE Transactions on Image Processing*, 1999.

[23] G. Fernandez, S. Periaswamy dan Wim Sweldens LIFTPACK: A Software Package for Wavelet Transforms using Lifting, 1997.

[24] Bruce Eckel *Thinking in Java 2nd Ed.*, Prentice Hall, 2000.

[25] Craig Larman *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design*, Prentice Hall ,1998

[26] Steven Harrington *Computer Graphics: A Programming Approach*, McGraw-Hill, 1983.

[27] Gerald Kaiser Complex-Distance Potential Theory and Hyperbolic Equations *J. Math Physics*, 2000.

Lampiran A

Fungsi-Fungsi Tes untuk API Aljabar Geometrik

Diambil dari demo program CLICAL(1992)

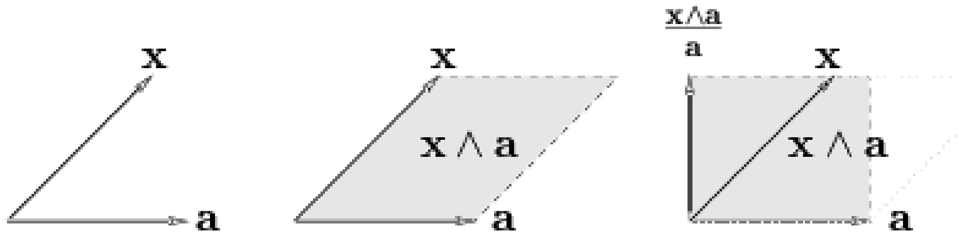
Soal nomor 1

Diketahui sebuah segitiga OPQ dimana P=(5,-1), Q=(3,1) dan O adalah titik asal. Berapakah luas segitiga OPQ?
$\overrightarrow{OP} = p = 5e_1 - e_2$
$\overrightarrow{OQ} = q = 3e_1 + e_2$
Luas segitiga OPQ adalah luas bidang yang dibentuk oleh p dan q dibagi dengan dua, yaitu $|p \wedge q|/2$

$$\begin{aligned} p \wedge q &= (5e_1 - e_2) \wedge (3e_1 + e_2) \\ &= 5e_1 \wedge e_2 - 3e_2 \wedge e_1 \\ &= 8e_1 \wedge e_2 \\ |p \wedge q| &= 8 \end{aligned}$$

Jadi luas segitiga OPQ $= 8/2 = 4$.

```
public class Soal1 {
   public static void main(String[] args) {
      Vector2g p = new Vector2g(5,-1);
```



Gambar A.1: Ilustrasi soal nomor dua untuk proyeksi vektor yang tegak lurus garis.

```
        Vector2g q = new Vector2g(3,1);
        Pseudoscalar2g I=p.wedge(q);
        double luas = I.area/2;
        System.out.println("p = " + p);
        System.out.println("q = " + q);
        System.out.println("I = p^q = " + I);
        System.out.println("Luas segitiga OPQ = " + luas);
    }
}
```

Soal nomor 2

Cari proyeksi $a = 3e_1 + e_2$ pada arah $b = e_1 + e_2$ dan proyeksi a yang tegak lurus dengan b!

Vektor a dapat dipecah menjadi bagian yang sejajar dengan b dan yang tegak lurus dengan b.

$$\begin{aligned} a &= (ab)b^{-1} \\ &= (a \cdot b + a \wedge b)b^{-1} \\ &= (a \cdot b)b^{-1} + (a \wedge b)b^{-1} \\ &= a_{\|} + a_{\perp} \end{aligned}$$

dimana $a_{\|} = a \cdot bb/b^2$ dan $a_{\perp} = a \wedge bb/b^2$.

$b^2 = b \cdot b = 1 + 1 = 2$

$a \cdot b = (3e_1 + e_2) \cdot (e_1 + e_2) = 3 + 1 = 4,$

$a \wedge b = (3e_1 + e_2) \wedge (e_1 + e_2) = (3 - 1)e_1 \wedge e_2 = 2e_1 \wedge e_2.$

$$\begin{aligned} \therefore \quad a_{\|} &= \frac{4(e_1 + e_2)}{2} = 2e_1 + 2e_2 \\ a_{\perp} &= \frac{2e_1 \wedge e_2(e_1 + e_2)}{2} = e_1 - e_2 \end{aligned}$$

Perhatikan kalau penjumlahannya benar menghasilkan a.

```
public class Soal2 {
   public static void main(String[] args) {
      Vector2g a = new Vector2g(3,1);
      Vector2g b = new Vector2g(1,1);
      Vector2g sejajar = a.parallel(b);
      Vector2g tegaklurus = a.perpendicular(b);
      System.out.println("a = " + a);
      System.out.println("b = " + b);
      System.out.println("proyeksi a sejajar b = " + sejajar);
      System.out.println("proyeksi a tegaklurus b = " + tegaklurus);
   }
}
```

Soal nomor 3

Cari bayangan dari $a = 3e_1 + 4e_2$ terhadap garis $b = 2e_1 + e_2$!

Ini adalah persoalan refleksi,

$$\begin{aligned} a' &= bab^{-1} \\ &= \frac{b(ab)}{b^2} \\ &= (2e_1 + e_2)(3e_1 + 4e_2)(2e_1 + e_2)/5 \\ &= (2e_1 + e_2)(10 - 5e_1 \wedge e_2)/5 \\ &= (20e_1 + 10e_2 - 10e_2 + 5e_1)/5 \\ &= 25e_1/5 \\ &= 5e_1 \end{aligned}$$

```
public class Soal3 {
   public static void main(String[] args) {
      Vector2g a = new Vector2g(3,4);
      Vector2g b = new Vector2g(2,1);
      Multivector2g bayangan = a.reflectTo(b);
```

```
      System.out.println("a = " + a);
      System.out.println("b = " + b);
      System.out.println("bayangan a terhadap garis b adalah " + bayangan);
   }
}
```

Soal nomor 4

Cari hasil perputaran vektor $a = 3e_1 + 5e_2$ ke kanan sebesar $\pi/3$ radian !
Karena rotasinya ke kanan maka rotornya adalah

$$\begin{aligned} R &= e^{I\frac{\theta}{2}} \\ &= e^{I\frac{\pi}{6}} \\ &= \cos\pi/6 + I\sin\pi/6 \\ &= 0.866 + 0.5I \\ \widetilde{R} &= 0.866 - 0.5I \end{aligned}$$

Hasil rotasinya

$$\begin{aligned} a' &= Ra\widetilde{R} \\ &= (0.866 + 0.5I)(3e_1 + 5e_2)(0.866 - 0.5I) \\ &= (2.598e_1 + 4.33e_2 - 1.5e_2 + 2.5e_1)(0.866 - 0.5I) \\ &= (5.098e_1 + 2.83e_2)(0.866 - 0.5I) \\ &= 4.415e_1 + 2.451e_2 - 2.549e_2 + 1.415e_1 \\ &= 5.83e_1 - 0.098e_2 \end{aligned}$$

Kita bisa periksa dengan $|a| = |a'| = 34$.

```
public class Soal4 {
   public static void main(String[] args) {
      Vector2g a = new Vector2g(3,5);
      Multivector2g RaR = a.rotateRight(Math.PI/3);
      System.out.println("a = " + a);
      System.out.println("a' = " + RaR);
   }
}
```

Soal nomor 5

Cari hasi perputaran rotasi vektor $r = 2e_1 + e_2 + 2e_3$ terhadap sumbu $\boldsymbol{a} = 1.5e_1 + 2e_2$ sebesar sudut $|a| = 2.5$ radian. Arah perputarannya tidak diberitahu sehingga kita ambil saja rotornya

$$\begin{aligned}
R &= e^{I\frac{\boldsymbol{a}}{2}} \\
&= \cos(|\boldsymbol{a}|/2) + I\hat{\boldsymbol{a}}\sin(|\boldsymbol{a}|/2) \\
&= 0.3153 + I\frac{1.5e_1 + 2e_2}{2.5}0.949 \\
&= 0.3153 + I(0.56939e_1 + 0.75919e_2) \\
&= 0.3153 + 0.56939e_2e_3 + 0.75919e_3e_1 \\
\widetilde{R} &= 0.3153 - 0.56939e_2e_3 - 0.75919e_3e_1
\end{aligned}$$

Transformasinya

$$\begin{aligned}
r' &= Rr\widetilde{R} \\
&= (0.3153 + 0.56939e_2e_3 + 0.75919e_3e_1)(2e_1 + e_2 + 2e_3) \\
&= (0.3153 - 0.56939e_2e_3 - 0.75919e_3e_1) \\
&= ((0.6306 - 1.5184)e_1 + (0.3153 + 1.13878)e_2 + (0.6306 + 1.5184 - 0.5694)e_3 \\
&\quad + (0.7592 + 1.1388)I)(0.3153 - 0.56939e_2e_3 - 0.75919e_3e_1) \\
&= (-0.8878e_1 + 1.4541e_2 + 1.5796e_3 + 1.898I)(0.3153 - 0.56939e_2e_3 - 0.75919e_3e_1) \\
&= (-0.2799 - 1.1992 + 1.08068)e_1 + (0.4585 + 1.4409 + 0.8994)e_2 \\
&\quad + (0.4980 - 0.6740 - 0.8279)e_3 + (0.5984 - 1.1039 + 0.5055)I \\
&= -0.3983e_1 + 2.7988e_2 - 1.0039e_3
\end{aligned}$$

```
public class Soal5 {
   public static void main(String[] args) {
      Vector3g r = new Vector3g(2,1,2);
      Vector3g a = new Vector3g(1.5,2,0);
      Multivector3g RrR = r.rotateAround(a);
      System.out.println("r = " + r);
      System.out.println("a = " + a);
      System.out.println("Hasil perputarannya adalah " + RrR);
   }
}
```

Soal nomor 6

Hitung luas parallelogram dengan sisi $u = 2e_1 + 3e_2 + 4e_3$ dan $v = 4e_1 + e_2 + 3e_3$!
Luasnya adalah $|u \wedge v|$

$$\begin{aligned} u \wedge v &= (2e_1 + 3e_2 + 4e_3) \wedge (4e_1 + e_2 + 3e_3) \\ &= 5e_{12} + 10e_{23} - 10e_{31} \\ |u \wedge v| &= 15 \end{aligned}$$

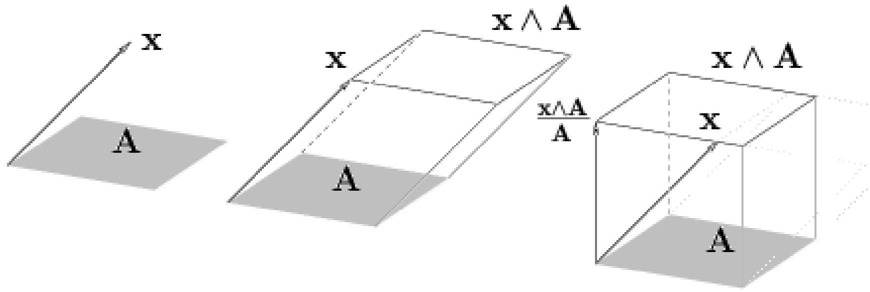
Perhatikan luasnya sama dengan $|u \times v|$, tentu saja perkalian silang hanya berlaku untuk 3D, sedangkan perkalian luar berlaku untuk sembarang dimensi yang lebih besar dari satu.

```
public class Soal6 {
   public static void main(String[] args) {
      Vector3g u = new Vector3g(2,3,4);
      Vector3g v = new Vector3g(4,1,3);
      Pseudovector3g outer = u.wedge(v);
      double area = outer.area();
      System.out.println("u = " + u);
      System.out.println("v = " + v);
      System.out.println("u^v = " + outer);
      System.out.println("Luas parallelogram adalah " + area);
   }
}
```

Soal nomor 7

Hitung komponen dari $q = -5e_1 + 7e_2$ yang sejajar dan tegak lurus dengan bidang $F = (4e_1 + e_3) \wedge (3e_1 + e_2)$!
Kita akan memecah q dengan operator proyeksi F^{-1}.

$$\begin{aligned} q &= (qF)F^{-1} \\ &= (q \cdot F + q \wedge F)F^{-1} \\ &= (q \cdot F)F^{-1} + (q \wedge F)F^{-1} \\ &= q_{\|} + q_{\perp} \end{aligned}$$



Gambar A.2: Ilustrasi soal nomor lima untuk proyeksi vektor yang tegak lurus bidang.

Tinggal bagaimana cara mencari invers dari bivektor F. Kita tahu bahwa kuadrat dari bivektor bukan skalar, sehingga kita tidak mendapatkan bentuk yang sama dengan invers vektor. Dengan menggunakan prinsip dualitas maka bivektor dirubah representasinya menjadi pseudovektor.

$$\begin{aligned} F &= (4e_1 + e_3) \wedge (3e_1 + e_2) \\ &= 4e_{12} + 3e_{31} - e_{23} \\ &= I(-e_1 + 3e_2 + 4e_3) \end{aligned}$$

Sedangkan inversnya $F^{-1} = I^{-1}(-e_1 + 3e_2 + 4e_3)/26$ karena $I^2 = -1$ maka $I^{-1} = -I$

$q \wedge F = ((-5e_1 + 7e_2) \cdot (-e_1 + 3e_2 + 4e_3))I = (5 + 21)I = 26I$

$q \cdot F = ((-5e_1 + 7e_2) \wedge (-e_1 + 3e_2 + 4e_3))I = (28e_2e_3 + 20e_3e_1 - 8e_1e_2)I$

Akhirnya didapatkan

$$\begin{aligned} q_\perp &= (q \wedge F)F^{-1} = 26II^{-1}(-e_1 + 3e_2 + 4e_3)/26 = -e_1 + 3e_2 + 4e_3 \\ q_{\|} &= (q \cdot F)F^{-1} = (28e_2e_3 + 20e_3e_1 - 8e_1e_2)II^{-1}(-e_1 + 3e_2 + 4e_3)/26 \\ &= -4e_1 + 4e_2 - 4e_3 \end{aligned}$$

Perhatikan bahwa kita bisa mencari $q_{\|}$ secara mudah dengan cara $q_{\|} = q - q_\perp$

```
public class Soal7 {
   public static void main(String[] args) {
      Vector3g q = new Vector3g(-5,7,0);
      Vector3g f1 = new Vector3g(4,0,1);
      Vector3g f2 = new Vector3g(3,1,0);
```

```
      Pseudovector3g F = f1.wedge(f2);
      Vector3g tegaklurus = q.perpendicular(F);
      Multivector3g sejajar = q.parallel(F);
      System.out.println("q = " + q);
      System.out.println("f1 = " + f1);
      System.out.println("f2 = " + f2);
      System.out.println("F = f1^f2 = " + F);
      System.out.println("komponen q yang tegak-lurus F adalah "
                         + tegaklurus);
      System.out.println("komponen q yang sejajar F adalah "
                         + sejajar);
   }
}
```

Lampiran B

Kode Sumber

Vector2g.java

```
import javax.vecmath.Vector2d;
/**
 *  Vector in 2D space with two elements
 *  (a1e1 + a2e2) with an extra ability to calculate geometric product
 *@author     Ginanjar Utama
 *@created    January 11, 2001
 */
public class Vector2g extends Vector2d {
/**
 *  Construct a Vector2g object and initialize it into (0,0)
 */
public Vector2g() {
super();
}


/**
 *  Construct a Vector2g object and initialize it with the value \
 *  of the input array v
 *
 *@param  v  1D array with 2 elements with double precision floating
 *           point type
 */
public Vector2g(double[] v) {
super(v);
}


/**
 *  Construct a Vector2g object and initialize it with (x,y)
 *
 *@param  x  the x coordinate with reference to basis vector e1
 *@param  y  the y coordinate with reference to basis vector e2
 */
public Vector2g(double x, double y) {
super(x, y);
}


/**
 *  Construct a Vector2g object and initialize with the value of
```

```
 *  another Vector2g object.
 *
 *
 *@param  v2g  another vector which values are copied
 */
public Vector2g(Vector2g v2g) {
x = v2g.x;
y = v2g.y;
}


/**
 *  Add a scalar to the vector2g object itself to create a multivector
 *
 *@param  scalar  scalar value
 *@return         multivector object
 */
public Multivector2g add(double scalar) {
Multivector2g mv2g = new Multivector2g(scalar,
new Vector2g(this), new Pseudoscalar2g(0));
return mv2g;
}


/**
 *  Add a pseudoscalar i to the vector2g object itself and create
 *  a multivector
 *
 *@param  i  pseudoscalar
 *@return    multivector object
 */
public Multivector2g add(Pseudoscalar2g i) {
Multivector2g mv2g = new Multivector2g(0,
new Vector2g(this), i);
return mv2g;
}


/**
 *  Perform the outer product between thr vector2g and another vector2g
 *  and create a multivector
 *@param  v2g  Description of Parameter
 *@return      Description of the Returned Value
 */
public Pseudoscalar2g wedge(Vector2g v2g) {
double i = x * v2g.y - y * v2g.x;
Pseudoscalar2g I = new Pseudoscalar2g(i);
return I;
}


/**
 *  Perform the geometric product between the vector2g and another vector2g
 *  and create a multivector
 *
 *@param  v2g  vector in 2D space
 *@return      multivector mv2g
 */
public Multivector2g mul(Vector2g v2g) {
double scalar = this.dot(v2g);
Pseudoscalar2g I = this.wedge(v2g);
Multivector2g mv2g = new Multivector2g(scalar,
new Vector2g(), I);
return mv2g;
}
```

```
/**
 *  Scale the vector2g object with a scalar
 *
 *@param  scalar  Description of Parameter
 */
public void mul(double scalar) {
this.scale(scalar);
}


/**
 *  Perform a duality operation or 90 degree-rotation anticlockwise
 *  on the vector
 *
 *@param  i  Description of Parameter
 */
public void mul(Pseudoscalar2g i) {
double temp = x;
x = i.area * -y;
y = i.area * temp;
}


/**
 *  Compose the value of the vector in a string
 *
 *@return    a string which describes the value of the vector
 */
public String toString() {
return "" + x + "e1 " + y + "e2";
}


/**
 *  Invert the vector
 */
public void invers() {
double abs2 = this.lengthSquared();
x /= abs2;
y /= abs2;
}


/**
 *  Calculate the parallel component of the vector with reference to
 *  another vector v2g
 *
 *@param  v2g  referenced vector
 *@return      parallel component
 */
public Vector2g parallel(Vector2g v2g) {
Vector2g result = new Vector2g(v2g);
double temp = this.dot(result);
result.invers();
result.scale(temp);
return result;
}


/**
 *  Calculate the perpendicular component of the vector with reference to
 *  another vector v2g
 *
```

```
 *@param  v2g  referenced vector
 *@return      perpendicular element
 */
public Vector2g perpendicular(Vector2g v2g) {
Vector2g result = new Vector2g(v2g);
Pseudoscalar2g I = this.wedge(result);
result.invers();
return I.mul(result);
}


/**
 *  Perform the reflection of the vector into another vector v2g
 *
 *@param  v2g  referenced vector
 *@return      multivector as the result of the reflection
 */
public Multivector2g reflectTo(Vector2g v2g) {
Vector2g newv2g = new Vector2g(v2g);
newv2g.invers();
return v2g.mul(this).mul(newv2g);
}


/**
 *  Rotate the vector to the right
 *
 *@param  radian  How far the rotation is performed
 *@return         multivector as the result of the rotation
 */
public Multivector2g rotateRight(double radian) {
Multivector2g R = new Multivector2g(Math.cos(radian / 2),
0, 0, Math.sin(radian / 2));
Multivector2g Rreverse = new Multivector2g(Math.cos(radian / 2),
0, 0, -Math.sin(radian / 2));
return R.mul(this).mul(Rreverse);
}


/**
 *  Rotate the vector to the left
 *
 *@param  radian  How far the rotation is performed
 *@return         multivector as the result of the rotation
 */
public Multivector2g rotateLeft(double radian) {
return rotateRight(-radian);
}
}
```

Pseudoscalar2g.java

```
/**
 *  An area in 2D space taking the form of a bivector e1e2 or I
 *
 *@author     utama
 *@created    January 11, 2001
 */
public class Pseudoscalar2g {
/**
 *  a value describing the area of the bivector
 */
public double area;
```

```
/**
 *  Construct a pseudoscalar2g object and initialize it with zero
 */
public Pseudoscalar2g() {
this(0);
}


/**
 *  Construct a pseudoscalar2g object and initialize it with
 *  a specified  area value
 *
 *@param  a  the value initializing the pseudoscalar object
 */
public Pseudoscalar2g(double a) {
area = a;
}


/**
 *  Construct a pseudoscalar2g object and initialize it with
 *  the value of another pseudoscalar i
 *
 *@param  i  the other pseudoscalar
 */
public Pseudoscalar2g(Pseudoscalar2g i) {
area = i.area;
}


/**
 *  Add another pseudoscalar i to the pseudoscalar
 *
 *@param  i  the other pseudoscalar
 */
public void add(Pseudoscalar2g i) {
area += i.area;
}


/**
 *  Perform a geometric product between the pseudoscalar and a vector2g v2g
 *
 *@param  v2g  vector2g value
 *@return      vector2g containing the result
 */
public Vector2g mul(Vector2g v2g) {
Vector2g v = new Vector2g();
v.x = area * v2g.y;
v.y = area * -v2g.x;
return v;
}


/**
 *  Scale the area of the bivector with a scalar d
 *
 *@param  d  scalar value
 */
public void scale(double d) {
area *= d;
}
```

```
/**
 *  Compose a string which describes the value of the pseudoscalar
 *
 *@return   the string whose content describes the value of the pseudoscalar
 */
public String toString() {
return "" + area + "I";
}
}
```

Multivector2g.java

```
/**
 *  A multivector in 2D space represented by scalar + vector + pseudoscalar.
 *  (a0 + a1e1 + a2e2 + a3I)
 *@author     Ginanjar Utama
 *@created    January 11, 2001
 */
public class Multivector2g {
/**
 *  The subspace scalar.
 */
public double scalar;
/**
 *  The subspace vector in 2D (e1,e2).
 */
public Vector2g vector;
/**
 *  The subspace bivector in 2D (e12).
 */
public Pseudoscalar2g I;


/**
 *  Construct and intializes a Multivector2g to 0 + 0e1 + 0e2 + 0e12.
 */
public Multivector2g() {
this(0, 0, 0, 0);
}


/**
 *  Constructor and initializes Multivector2g object to
 *  alpha + a1e1 + a2e2 + iI
 *
 *@param  alpha  scalar
 *@param  a1     coefficient of e1
 *@param  a2     coefficient of e2
 *@param  i      area of bivector
 */
public Multivector2g(double alpha, double a1, double a2,
double i) {
this(alpha, new Vector2g(a1, a2), new Pseudoscalar2g(i));
}


/**
 *  Constructor and initialize a Multivector2g from the specified
 *  Vector2g
 *@param  alpha  scalar
 *@param  v2g    a Vector2g
 *@param  i      area of bivector
 */
```

```
public Multivector2g(double alpha, Vector2g v2g, Pseudoscalar2g i) {
scalar = alpha;
vector = v2g;
I = i;
}


/**
 *  Constructor and intializes Multivector2g from the specified Multivector2g
 *
 *@param  mv2g  The Multivector2d containing the initialization data
 */
public Multivector2g(Multivector2g mv2g) {
scalar = mv2g.scalar;
vector = new Vector2g(mv2g.vector);
I = new Pseudoscalar2g(mv2g.I);
}


/**
 *  Add a  specified multivector with the multivector
 *
 *@param  mv2g  specified multivector
 */
public void add(Multivector2g mv2g) {
add(mv2g.scalar);
add(mv2g.vector);
add(mv2g.I);
}


/**
 *  Add the scalar component of the multivector with a scalar d
 *
 *@param  d  scalar to be added to the multivector
 */
public void add(double d) {
scalar += d;
}


/**
 *  Add the vector component of the multivector with a vector v2g
 *
 *@param  v2g  vector to be added to the multivector
 */
public void add(Vector2g v2g) {
vector.add(v2g);
}


/**
 *  Add the pseudoscalar component of the multivector with a pseudoscalar i
 *
 *@param  i  pseudoscalar to be added to the multivector
 */
public void add(Pseudoscalar2g i) {
I.add(i);
}


/**
 *  Compose a string which describes the value of the multivector
 *
 *@return   the string whose content describes the value of the multivector
```

```
 */
public String toString() {
return " " + scalar + " " + vector + " " + I;
}


/**
 *  Perform the geometric product between the multivector and a vector v2g
 *
 *@param  v2g  vector value
 *@return      multivector containing the result
 */
public Multivector2g mul(Vector2g v2g) {
Vector2g newv2g = new Vector2g(v2g);
newv2g.scale(scalar);
Multivector2g mv2g = new Multivector2g(0, newv2g,
new Pseudoscalar2g(0));
mv2g.add(vector.mul(v2g));
mv2g.add(I.mul(v2g));
return mv2g;
}


/**
 *  Perform the geometric product between the multivector and a pseudoscalar i
 *
 *@param  i  pseudoscalar value
 *@return    multivector containing the result
 */
public Multivector2g mul(Pseudoscalar2g i) {
Multivector2g result = new Multivector2g(this);
result.scalar = -I.area * i.area;
result.vector.mul(i);
result.I.area = scalar * i.area;
return result;
}


/**
 *  Perform the geometric product between the multivector and
 *  another multivector mv2g
 *
 *@param  mv2g  specified multivector
 *@return       multivector containing the result
 */
public Multivector2g mul(Multivector2g mv2g) {
Multivector2g result = new Multivector2g(this);
result.scale(mv2g.scalar);
result.add(this.mul(mv2g.vector));
result.add(this.mul(mv2g.I));
return result;
}


/**
 *  Scale the scalar component of the multivector with a scalar
 *
 *@param  d  scalar value
 */
public void scale(double d) {
scalar *= d;
vector.scale(d);
I.scale(d);
}
```

```
/**
 *  Reverse the multivector
 */
public void reverse() {
I.area = -I.area;
}
}
```

Vector3g.java

```
import javax.vecmath.Vector3d;
import javax.vecmath.Matrix4d;
import javax.vecmath.Point3d;
import javax.vecmath.GMatrix;
import javax.vecmath.GVector;
import javax.vecmath.Vector4d;

/**
 *  Vector in 3D space with three elements
 *  (a1e1 + a2e2 +a3e3) with an extra ability to calculate geometric product
 *
 *@author     Ginanjar Utama
 *@created    January 11, 2001
 */
public class Vector3g extends Vector3d {
/**
 *  Construct a Vector3g object and initialize it into (0,0,0)
 */
public Vector3g() {
super();
}


/**
 *  Construct a Vector3g object and initialize it with the value
 *  of the input array v
 *
 *@param  v  1D array with 3 elements with double precision floating
 *           point type
 */
public Vector3g(double[] v) {
super(v);
}


/**
 *  Construct a Vector3g object and initialize it with (x,y,z)
 *
 *@param  x  the x coordinate with reference to basis vector e1
 *@param  y  the y coordinate with reference to basis vector e2
 *@param  z  the z coordinate with reference to basis vector e3
 */
public Vector3g(double x, double y, double z) {
super(x, y, z);
}


/**
 *  Construct a Vector3g object and initialize with the value of
 *  another Vector3g object.
 *
 *
 *@param  v3g  another vector which values are copied
```

```
 */
public Vector3g(Vector3g v3g) {
x = v3g.x;
y = v3g.y;
z = v3g.z;
}


/**
 *  Add a scalar to the vector3g object itself to create a multivector
 *
 *@param  scalar  scalar value
 *@return         multivector object
 */
public Multivector3g add(double scalar) {
Multivector3g mv3g = new Multivector3g(scalar,
new Vector3g(this), new Pseudovector3g(),
new Pseudoscalar3g());
return mv3g;
}


/**
 *  Add a pseudoscalar i to the vector3g object itself and create
 *  a multivector
 *
 *@param  i  pseudovector
 *@return    multivector object
 */
public Multivector3g add(Pseudovector3g i) {
Multivector3g mv3g = new Multivector3g(0,
new Vector3g(this), i, new Pseudoscalar3g());
return mv3g;
}


/**
 *  Add a pseudoscalar i to the vector3g object itself and create
 *  a multivector
 *
 *@param  i  pseudoscalar
 *@return    multivector object
 */
public Multivector3g add(Pseudoscalar3g i) {
Multivector3g mv3g = new Multivector3g(0,
new Vector3g(this), new Pseudovector3g(), i);
return mv3g;
}


/**
 *  Perform an outer product operation between the vector and
 *  a pseudovector iv
 *
 *@param  iv  pseudovector
 *@return     pseudoscalar as the result of the operation
 */
public Pseudoscalar3g wedge(Pseudovector3g iv) {
return new Pseudoscalar3g(this.x * iv.x +
this.y * iv.y + this.z * iv.z);
}


/**
 *  Perform the outer product between the vector3g and another vector3g
```

```
 *  and create a pseudovector
 *@param  v3g  vector3g
 *@return      pseudovector as the result
 */
public Pseudovector3g wedge(Vector3g v3g) {
Vector3g temp = new Vector3g(this);
temp.cross(this, v3g);
Pseudovector3g ans = new Pseudovector3g(temp.x, temp.y, temp.z);
return ans;
}


/**
 *  Perform the inner product between the vector3g and a pseudovector iv
 *  and create a new vector3g object
 *@param  iv  vector3g
 *@return      new vector3g object as the result
 */
public Vector3g dot(Pseudovector3g iv) {
Vector3g temp = new Vector3g(iv.x, iv.y, iv.z);
temp.cross(temp, this);
return temp;
}


/**
 *  Perform the geometric product between the vector and another vector v3g
 *
 *@param  v3g  another vector3g object
 *@return      multivector
 */
public Multivector3g mul(Vector3g v3g) {
double scalar = this.dot(v3g);
Pseudovector3g Iv = this.wedge(v3g);
Multivector3g mv3g = new Multivector3g(scalar,
new Vector3g(), Iv, new Pseudoscalar3g());
return mv3g;
}


/**
 *  Scale the vector
 *
 *@param  scalar  scalar value
 */
public Vector3g mul(double scalar) {
this.scale(scalar);
     return this;
}


/**
 *  Perform the geometric product between the vector and a pseudovector iv
 *
 *@param  iv  pseudovector value
 *@return     multivector value
 */
public Multivector3g mul(Pseudovector3g iv) {
Vector3g v = new Vector3g(this.dot(iv));
Pseudoscalar3g newI = new Pseudoscalar3g(this.wedge(iv));
Multivector3g mv3g = new Multivector3g(0, v,
new Pseudovector3g(), newI);
return mv3g;
}
```

```
/**
 *  Perform the geometric product between the vector and a pseudoscalar i
 *
 *@param  i  pseudoscalar value
 *@return     pseudovector value
 */
public Pseudovector3g mul(Pseudoscalar3g i) {
return new Pseudovector3g(i.volume * x, i.volume * y, i.volume * z);
}


/**
 *  Compose the value of the vector in a string
 *
 *@return    a string which describes the value of the vector
 */
public String toString() {
return "" + x + "e1 " + y + "e2 " + z + "e3 ";
}


/**
 *  Invert the vector
 */
public void invers() {
double abs2 = this.lengthSquared();
x /= abs2;
y /= abs2;
z /= abs2;
}


/**
     *  Calculate the parallel component of the vector with reference to
 *  another vector v3g
 *
 *@param  v3g  referenced vector
 *@return      parallel component
 */
public Vector3g parallel(Vector3g v3g) {
Vector3g result = new Vector3g(v3g);
double temp = this.dot(result);
result.invers();
result.scale(temp);
return result;
}


/**
     *  Calculate the parallel component of the vector with reference to
 *  a pseudovector iv3g
 *
 *@param  iv3g  pseudovector value
 *@return       multivector
 */
public Multivector3g parallel(Pseudovector3g iv3g) {
Pseudovector3g temp = new Pseudovector3g(iv3g);
temp.invers();
return this.dot(iv3g).mul(temp);
}


/**
     *  Calculate the perpendicular component of the vector with reference to
```

```
 *  another vector v3g
 *
 *@param  v3g  referenced vector
 *@return      multivector
 */
public Multivector3g perpendicular(Vector3g v3g) {
Vector3g result = new Vector3g(v3g);
Pseudovector3g Iv = this.wedge(result);
result.invers();
return Iv.mul(result);
}


/**
     *  Calculate the perpendicular component of the vector with reference to
 *  a pseudovector iv3g
 *
 *@param  iv3g  pseudovector value
 *@return       vector value
 */
public Vector3g perpendicular(Pseudovector3g iv3g) {
Pseudovector3g temp = new Pseudovector3g(iv3g);
temp.invers();
return this.wedge(iv3g).mul(temp);
}


/**
 *  Perform a reflection of the vector to another vector v3g
 *
 *@param  v3g  referenced vector
 *@return      multivector value
 */
public Multivector3g reflectTo(Vector3g v3g) {
Vector3g newv3g = new Vector3g(v3g);
newv3g.invers();
return v3g.mul(this).mul(newv3g);
}


/**
 *  Perform a rotation of the vector around another vector v3g
 *
 *@param  v3g  referenced vector
 *@return      multivector value
 */
public Multivector3g rotateAround(Vector3g v3g) {
long awal = System.currentTimeMillis();
double length = v3g.length();
Pseudovector3g temp = new Pseudovector3g(v3g.x, v3g.y, v3g.z);
temp.scale(Math.sin(length / 2) / length);
Multivector3g R = new Multivector3g(Math.cos(length / 2),
new Vector3g(), temp, new Pseudoscalar3g());
Multivector3g Rr = R.mul(this);
temp.scale(-1);
Multivector3g Rrev = new Multivector3g(Math.cos(length / 2),
new Vector3g(), temp, new Pseudoscalar3g());
Multivector3g result = Rr.mul(Rrev);
long akhir = System.currentTimeMillis();
long lama = akhir - awal;
System.out.println("Total rotation time " + lama + "ms");
return result;
}
public Multivector3g rotateAround(Vector3g v3g, double theta) {
long awal = System.currentTimeMillis();
```

```
double length = v3g.length();
double ratio = theta /length;
Pseudovector3g temp = new Pseudovector3g(v3g.x, v3g.y, v3g.z);
temp.scale(Math.sin(ratio*length / 2) / length);
Multivector3g R = new Multivector3g(Math.cos(ratio*length / 2),
new Vector3g(), temp, new Pseudoscalar3g());
Multivector3g Rr = R.mul(this);
temp.scale(-1);
Multivector3g Rrev = new Multivector3g(Math.cos(ratio*length / 2),
new Vector3g(), temp, new Pseudoscalar3g());
Multivector3g result = Rr.mul(Rrev);
long akhir = System.currentTimeMillis();
long lama = akhir - awal;
System.out.println("Total rotation time " + lama + "ms");
return result;
}
public GVector rotateUseMatrix(Vector3g arah, Point3d titik,
                                    double t) {
long awal = System.currentTimeMillis();
double A,B,C,x,y,z,V,L;
A = arah.x;
B = arah.y;
C = arah.z;
x = titik.x;
y = titik.y;
z = titik.z;
V = Math.sqrt(B*B + C*C);
L = Math.sqrt(A*A + B*B + C*C);
Matrix4d T = new Matrix4d(1, 0, 0, 0,
                                    0, 1, 0, 0,
                                    0, 0, 1, 0,
                                   -x,-y,-z, 1);
          Matrix4d Tinv = new Matrix4d(1, 0, 0, 0,
                                       0, 1, 0, 0,
                                       0, 0, 1, 0,
                                       x, y, z, 1);
          Matrix4d Rx = new Matrix4d(1, 0, 0, 0,
                                    0, C/V, B/V, 0,
                                    0,-B/V, C/V, 0,
                                    0, 0, 0, 1);
          Matrix4d Rxinv = new Matrix4d(1, 0, 0, 0,
                                        0, C/V, -B/V, 0,
                                        0, B/V, C/V, 0,
                                        0, 0, 0, 1);
          Matrix4d Ry = new Matrix4d(V/L, 0, A/L, 0,
                                       0, 1,   0, 0,
                                    -A/L, 0, V/L, 0,
                                       0, 0,   0, 1);
          Matrix4d Ryinv = new Matrix4d(V/L, 0,-A/L, 0,
                                          0, 1,   0, 0,
                                        A/L, 0, V/L, 0,
                                          0, 0,   0, 1);
     Matrix4d Rz = new Matrix4d(Math.cos(t), Math.sin(t), 0, 0,
                                    -Math.sin(t), Math.cos(t), 0, 0,
                                               0,            0, 1, 0,
                                               0,            0, 0, 1);
          Matrix4d TotalTransform = new Matrix4d(Tinv);
          TotalTransform.mul(Rxinv);
          TotalTransform.mul(Ryinv);
          TotalTransform.mul(Rz);
          TotalTransform.mul(Ry);
          TotalTransform.mul(Rx);
          TotalTransform.mul(T);
          GMatrix TotTrans = new GMatrix(4,4);
          TotTrans.set(TotalTransform);
```

```
          Vector4d oldvec = new Vector4d(this);
          GVector newvec = new GVector(oldvec);
          GVector result = new GVector(4);
       result.mul(TotTrans, newvec);
long akhir = System.currentTimeMillis();
long lama = akhir - awal;
System.out.println("Total rotation time " + lama + "ms");
return result;
}

 public static void main(String[] args) {
  Vector3g t = new Vector3g(1,2,3);
  Pseudovector3g k = new Pseudovector3g(3,2,1);
  Pseudoscalar3g j = new Pseudoscalar3g(2);
  t.scale(2);
  System.out.println("t = " + t);
  System.out.println("k = " + k);
  System.out.println("j = " + j);
  System.out.println("tk = " + t.mul(k) );
  System.out.println("tj = " + t.mul(j) );
  }
}
```

Pseudovector3g.java

```
import javax.vecmath.Tuple3d;
/**
 *  a pseudovector in 3D space taking the form of a4Ie1 + a5Ie2 + a6Ie3
 *@author     Ginanjar Utama
 *@created    January 11, 2001
 */
public class Pseudovector3g extends Tuple3d {
/**
 *  Constructor for the Pseudovector3g object
 */
public Pseudovector3g() {
super();
}


/**
 *  Construct a Pseudovector3g object and initialize it with the value of
 *  an array v
 *
 *@param  v  the array whose values initialize the new pseudovector3g object
 */
public Pseudovector3g(double[] v) {
super(v);
}


/**
 *  Construct a Pseudovector3g object and initialize it with the value of
 *  x, y, and z
 *
 *@param  x  initial value for the new pseudovector3g object
 *@param  y  initial value for the new pseudovector3g object
 *@param  z  initial value for the new pseudovector3g object
 */
public Pseudovector3g(double x, double y, double z) {
super(x, y, z);
}
```

```
/**
 *  Construct a Pseudovector3g object and initliaze it with the value of
 *  another pseudovector3g
 *
 *@param  iv3g  the value of another pseudovector3g
 */
public Pseudovector3g(Pseudovector3g iv3g) {
x = iv3g.x;
y = iv3g.y;
z = iv3g.z;
}


/**
 *  Add the scalar component of the pseudovector3g with
 *
 *@param  scalar  a scalar to be added to the scalar component of
 *                the pseudovector
 *@return         multivector3g value
 */
public Multivector3g add(double scalar) {
Multivector3g mv3g = new Multivector3g(scalar,
new Vector3g(), new Pseudovector3g(this),
new Pseudoscalar3g());
return mv3g;
}


/**
 *  Add the vector3g part of the pseudovector with a vector v3g
 *
 *@param  v3g  a vector3g to be added to the pseudovector
 *@return      a multivector3g value
 */
public Multivector3g add(Vector3g v3g) {
Multivector3g mv3g = new Multivector3g(0, v3g,
new Pseudovector3g(this), new Pseudoscalar3g());
return mv3g;
}


/**
 *  Add the pseudovector with a pseudoscalr3g i
 *
 *@param  i  the pseudoscalar to be added to the pseudovector
 *@return    a multivector3g value
 */
public Multivector3g add(Pseudoscalar3g i) {
Multivector3g mv3g = new Multivector3g(0,
new Vector3g(), new Pseudovector3g(this), i);
return mv3g;
}


/**
 *  Perform an inner product operation between the pseudovector and another
 *  pseudovector3g iv
 *
 *@param  iv  the value of another pseudovector
 *@return     a scalar value
 */
public double dot(Pseudovector3g iv) {
return -(this.x * iv.x + this.y * iv.y + this.z * iv.z);
}
```

```
/**
 *  Perform an inner product operation between the pseudovector
 *  and a vector3g v3g
 *
 *@param  v3g  the value of the vector3g
 *@return      a vector3g value
 */
public Vector3g dot(Vector3g v3g) {
Vector3g result = v3g.dot(this);
result.scale(-1);
return result;
}


/**
 *  Perform an outer product operation between the pseudovector
 *  and another pseudovector3g iv
 *
 *@param  iv  another pseudovector3g
 *@return     a pseudovector value
 */
public Pseudovector3g wedge(Pseudovector3g iv) {
return new Pseudovector3g(this.z * iv.y - this.y * iv.z,
this.x * iv.z - this.z * iv.x, this.y * iv.x - this.x * iv.y);
}


/**
 *  Perform an outer product operation between the pseudovector
 *  and a vector3g v3g
 *
 *@param  v3g  a vector3g
 *@return      a pseudoscalar value
 */
public Pseudoscalar3g wedge(Vector3g v3g) {
return v3g.wedge(this);
/*Pseudoscalar3g ans = new Pseudoscalar3g(this.x * v3g.x +
this.y * v3g.y + this.z * v3g.z);
ans.scale(-1);
return ans;
*/
}


/**
 *  Perform a geometric product operation between the pseudovector
 *  and another pseudovector3g iv3g
 *
 *@param  iv3g  another pseudovector3g
 *@return       a multivector value
 */
public Multivector3g mul(Pseudovector3g iv3g) {
double scalar = this.dot(iv3g);
Pseudovector3g Iv = this.wedge(iv3g);
Multivector3g mv3g = new Multivector3g(scalar,
new Vector3g(), Iv, new Pseudoscalar3g());
return mv3g;
}


/**
 *  Perform a geometric product operation between the pseudovector
 *  and a scalar
 *
```

```
 *@param  scalar  a scalar value
 */
public void mul(double scalar) {
this.scale(scalar);
}


/**
 *  Perform a geometric product operation between the pseudovector
 *  and a vector3g
 *
 *@param  v3g  a vector3g value
 *@return      a mutivector3g value
 */
public Multivector3g mul(Vector3g v3g) {
Vector3g v = new Vector3g(this.dot(v3g));
Pseudoscalar3g I = new Pseudoscalar3g(this.wedge(v3g));
Multivector3g mv3g = new Multivector3g(0, v,
new Pseudovector3g(), I);
return mv3g;
}


/**
 *  Perform a geometric product operation between the pseudovector
 *  and a pseudoscalar3g i
 *
 *@param  i  a pseudoscalar value
 *@return    a vector3g value
 */
public Vector3g mul(Pseudoscalar3g i) {
return new Vector3g(-i.volume * x, -i.volume * y, -i.volume * z);
}


/**
 *  Compose a string which describes the value of the pseudovector
 *
 *@return   the string whose content describes the value of the pseudovector
 */
public String toString() {
return "" + x + "Ie1 " + y + "Ie2 " + z + "Ie3 ";
}


/**
 *  Invert the pseudovector
 */
public void invers() {
double abs2 = this.areaSquared();
x = (-x / abs2);
y = (-y / abs2);
z = (-z / abs2);
}


/**
 *  Reflect the pseudovector to a vector3g v3g
 *
 *@param  v3g  the referenced vector3g
 *@return      a multivector3g value as the result of the reflection
 */
public Multivector3g reflectTo(Vector3g v3g) {
Vector3g newv3g = new Vector3g(v3g);
newv3g.invers();
```

```
return v3g.mul(this).mul(newv3g);
}


/**
 *  Rotate the pseudovector around a vector3g
 *
 *@param  v3g  the referenced vector3g
 *@return      a multivector3g value as the result of the rotation
 */
public Multivector3g rotateAround(Vector3g v3g) {
double length = v3g.length();
Pseudovector3g temp = new Pseudovector3g(v3g.x, v3g.y, v3g.z);
temp.scale(Math.sin(length / 2));
Multivector3g R = new Multivector3g(Math.cos(length / 2),
new Vector3g(), temp, new Pseudoscalar3g());
Multivector3g Rr = R.mul(this);
temp.scale(-1);
Multivector3g Rrev = new Multivector3g(Math.cos(length / 2),
new Vector3g(), temp, new Pseudoscalar3g());
return Rr.mul(Rrev);
}


/**
 *  Calculate the area given by the pseudovector3g
 *
 *@return    the area
 */
public double area() {
return Math.sqrt(areaSquared());
}


/**
 *  Calculate the squared area given by the pseudovector3g
 *
 *@return    the squared area
 */
public double areaSquared() {
return (x * x + y * y + z * z);
}
public static void main(String[] args) {
      Pseudovector3g E = new Pseudovector3g(1,2,4);
      Vector3g B = new Vector3g(3,5,7);
      System.out.println("EB = " + E +"mul"+ B + "="+ E.mul(B));
      Pseudovector3g C = new Pseudovector3g(2,1,2);
      System.out.println("EC = " + E +"mul"+ C + "="+ E.mul(C));
      Pseudoscalar3g J = new Pseudoscalar3g(2);
      System.out.println("" + E.mul(J));
   }
}
```

Pseudoscalar3g.java

```
/**
 *  A volume in 3D space taking the form of a trivector e1e2e3 or I
 *
 *@author     Ginanjar Utama
 *@created    January 11, 2001
 */
public class Pseudoscalar3g {
/**
 *  a value describing the volume of the trivector
```

```
 */
public double volume;


/**
 *  Construct a pseudoscalar3g object and initialize it with zero
 */
public Pseudoscalar3g() {
this(0);
}


/**
 *  Construct a pseudoscalar3g object and initialize it with
 *  a specified  volume value
 *
 *@param  a  the value initializing the pseudoscalar object
 */
public Pseudoscalar3g(double a) {
volume = a;
}


/**
 *  Construct a pseudoscalar3g object and initialize it with
 *  the value of another pseudoscalar3g i
 *
 *@param  i  the other pseudoscalar3g
 */
public Pseudoscalar3g(Pseudoscalar3g i) {
volume = i.volume;
}


/**
 *  Add another pseudoscalar3g i to the pseudoscalar3g
 *
 *@param  i  the other pseudoscalar3g
 */
public void add(Pseudoscalar3g i) {
volume += i.volume;
}


/**
 *  Perform a geometric product between the pseudoscalar and a vector3g v3g
 *
 *@param  v3g  vector3g value
 *@return      vector3g containing the result
 */
public Pseudovector3g mul(Vector3g v3g) {
return new Pseudovector3g(v3g.x * volume,
v3g.y * volume, v3g.z * volume);
}


/**
 *  Description of the Method
 *
 *@param  iv3g  Description of Parameter
 *@return       Description of the Returned Value
 */
public Vector3g mul(Pseudovector3g iv3g) {
return new Vector3g(-iv3g.x * volume, -iv3g.y * volume,
-iv3g.z * volume);
```

```
}


/**
 *  Description of the Method
 *
 *@param  d  Description of Parameter
 */
public void scale(double d) {
volume *= d;
}


/**
 *  Description of the Method
 *
 *@return    Description of the Returned Value
 */
public String toString() {
return "" + volume + "I";
}
}
```

Multivector3g.java

```
/**
 *  A Multivector in 3D space
 *   (a0 + a1e1 + a2e2 + a3e3 + a4Ie1 + a5Ie2 + a6Ie3 + a7I)
 *@author     Ginanjar Utama
 *@created    January 11, 2001
 */
public class Multivector3g {
/**
 *  the scalar component of the multivector
 */
public double scalar;
/**
 *  the vector component of the multivector
 */
public Vector3g vector;
/**
 *  the pseudovector component of the multivector
 */
public Pseudovector3g bivector;
/**
 *  the pseudoscalar component of the multivector
 */
public Pseudoscalar3g I;


/**
 *  Construct a Multivector3g object and initialize with zeros
 */
public Multivector3g() {
this(0, 0, 0, 0, 0, 0, 0, 0);
}


/**
 *  Construct a Multivector3g object and initialize with specified values
 *  a0, a1, a2, a3, a4, a5, a6, a7
 *
 *@param  a0  value to initialize the scalar component
 *@param  a1  value to initialize the vector3g component e1
```

```
 *@param  a2  value to initialize the vector3g component e2
 *@param  a3  value to initialize the vector3g component e3
 *@param  a4  value to initialize the bivector3g component Ie1
 *@param  a5  value to initialize the bivector3g component Ie2
 *@param  a6  value to initialize the bivector3g component Ie3
 *@param  a7  value to initialize the trivector3g
 */
public Multivector3g(double a0, double a1, double a2,
double a3, double a4, double a5,
double a6, double a7) {
this(a0, new Vector3g(a1, a2, a3), new Pseudovector3g(a4, a5, a6),
new Pseudoscalar3g(a7));
}


/**
 *  Construct a Multivector3g object and initialize with specified values
 *  alpha, v3g, iv, and i
 *
 *@param  alpha  a value which initialize the scalar component
 *@param  v3g    a value which initialize the vector component
 *@param  iv     a value which initialize the pseudovector component
 *@param  i      a value which initialize the pseudoscalar component
 */
public Multivector3g(double alpha, Vector3g v3g,
Pseudovector3g iv, Pseudoscalar3g i) {
scalar = alpha;
vector = v3g;
bivector = iv;
I = i;
}


/**
 *  Construct a Multivector3g object and initialize with the value
 *  of another multivector 3g
 *
 *@param  mv3g  another multivector to initialize the multivector
 */
public Multivector3g(Multivector3g mv3g) {
scalar = mv3g.scalar;
vector = new Vector3g(mv3g.vector);
bivector = new Pseudovector3g(mv3g.bivector);
I = new Pseudoscalar3g(mv3g.I);
}


/**
 *  Add the value of another multivector3g to the multivector
 *
 *@param  mv3g  another multivector whose values are added
 *              to the multivector
 */
public void add(Multivector3g mv3g) {
add(mv3g.scalar);
add(mv3g.vector);
add(mv3g.bivector);
add(mv3g.I);
}


/**
 *  Add a scalar d to the scalar component of the multivector3g
 *
 *@param  d    a scalar to be added to the multivector3g
```

```
 */
public void add(double d) {
scalar += d;
}


/**
 *  Add a vector v3g to the vector component of the multivector3g
 *
 *@param  v3g  a vector to be added to the multivector3g
 */
public void add(Vector3g v3g) {
vector.add(v3g);
}


/**
 *  Add a pseudovector iv3g to the pseudovector component of
 *  the multivector3g
 *
 *@param  iv3g  a pseudovector to be added to the multivector3g
 */
public void add(Pseudovector3g iv3g) {
bivector.add(iv3g);
}


/**
 *  Add a pseudoscalar i to the pseudoscalar component of
 *  the multivector3g
 *
 *@param  i  a pseudoscalar to be added to the multivector3g
 */
public void add(Pseudoscalar3g i) {
I.add(i);
}


/**
 *  Compose a string which describes the value of the multivector
 *
 *@return   the string whose content describes the value of the multivector
 */
public String toString() {
return " " + scalar + " " + vector + " " + bivector + " " + I;
}


/**
 *  Perform the geometric product between the multivector and a vector v3g
 *
 *@param  v3g  vector value
 *@return      multivector containing the result
 */
public Multivector3g mul(Vector3g v3g) {
Vector3g newv3g = new Vector3g(v3g);
newv3g.scale(scalar);
Multivector3g mv3g = new Multivector3g(0, newv3g,
new Pseudovector3g(), new Pseudoscalar3g());
mv3g.add(vector.mul(v3g));
mv3g.add(bivector.mul(v3g));
mv3g.add(I.mul(v3g));
return mv3g;
}
```

```
/**
 *  Perform the geometric product between the multivector and
 *  a pseudovector iv3g
 *
 *@param  iv3g  pseudovector value
 *@return       multivector containing the result
 */
public Multivector3g mul(Pseudovector3g iv3g) {
Pseudovector3g newiv3g = new Pseudovector3g(iv3g);
newiv3g.scale(scalar);
Multivector3g mv3g = new Multivector3g(0, new Vector3g(),
newiv3g, new Pseudoscalar3g());
mv3g.add(vector.mul(iv3g));
mv3g.add(bivector.mul(iv3g));
mv3g.add(I.mul(iv3g));
return mv3g;
}


/**
 *  Perform the geometric product between the multivector and
 *  a pseudoscalar i
 *
 *@param  i  pseudoscalar value
 *@return       multivector containing the result
 */
public Multivector3g mul(Pseudoscalar3g i) {
Multivector3g result = new Multivector3g();
result.scalar = -I.volume * i.volume;
result.vector = bivector.mul(i);
result.bivector = vector.mul(i);
result.I.volume = scalar * i.volume;
return result;
}


/**
 *  Perform the geometric product between the multivector and
 *  another multivector3g
 *
 *@param  mv3g  another multivector3g
 *@return       multivector containing the result
 */
public Multivector3g mul(Multivector3g mv3g) {
Multivector3g result = new Multivector3g(this);
result.scale(mv3g.scalar);
result.add(this.mul(mv3g.vector));
result.add(this.mul(mv3g.bivector));
result.add(this.mul(mv3g.I));
return result;
}


/**
 *  Scale the scalar part of the multivector with a scalar value
 *
 *@param  d  the scalar value
 */
public void scale(double d) {
scalar *= d;
vector.scale(d);
bivector.scale(d);
I.scale(d);
}
```

```
/**
 *  Reverse the multivector
 */
public void reverse() {
bivector.scale(-1);
I.volume = -I.volume;
}
     public Multivector3g rotateAround(Vector3g v3g) {
long awal = System.currentTimeMillis();
double length = v3g.length();
Pseudovector3g temp = new Pseudovector3g(v3g.x, v3g.y, v3g.z);
temp.scale(Math.sin(length / 2) / length);
Multivector3g R = new Multivector3g(Math.cos(length / 2),
new Vector3g(), temp, new Pseudoscalar3g());
Multivector3g Rr = R.mul(this);
temp.scale(-1);
Multivector3g Rrev = new Multivector3g(Math.cos(length / 2),
new Vector3g(), temp, new Pseudoscalar3g());
Multivector3g result = Rr.mul(Rrev);
long akhir = System.currentTimeMillis();
long lama = akhir - awal;
System.out.println("Total rotation time " + lama + "ms");
return result;
}
public Multivector3g rotateAround(Vector3g v3g, double theta) {
long awal = System.currentTimeMillis();
double length = v3g.length();
double ratio = theta /length;
Pseudovector3g temp = new Pseudovector3g(v3g.x, v3g.y, v3g.z);
temp.scale(Math.sin(ratio*length / 2) / length);
Multivector3g R = new Multivector3g(Math.cos(ratio*length / 2),
new Vector3g(), temp, new Pseudoscalar3g());
Multivector3g Rr = R.mul(this);
temp.scale(-1);
Multivector3g Rrev = new Multivector3g(Math.cos(ratio*length / 2),
new Vector3g(), temp, new Pseudoscalar3g());
Multivector3g result = Rr.mul(Rrev);
long akhir = System.currentTimeMillis();
long lama = akhir - awal;
System.out.println("Total rotation time " + lama + "ms");
return result;
}
}
```

Lampiran C

Keluaran

```
Script started on Sat Jan  6 05:41:08 2001

[utama@Qolam latexdoc]$ java Soal1
p = 5.0e1 -1.0e2
q = 3.0e1 1.0e2
I = p^q = 8.0I
Luas segitiga OPQ = 4.0

[utama@Qolam latexdoc]$ java Soal2
a = 3.0e1 1.0e2
b = 1.0e1 1.0e2
proyeksi a sejajar b = 2.0e1 2.0e2
proyeksi a tegaklurus b = 1.0e1 -1.0e2

[utama@Qolam latexdoc]$ java Soal3
a = 3.0e1 4.0e2
b = 2.0e1 1.0e2
bayangan a terhadap garis b adalah  0.0 5.0e1 0.0e2 0.0I

[utama@Qolam latexdoc]$ java Soal4
a = 3.0e1 5.0e2
a' =  0.0 5.830127018922194e1 -0.09807621135331512e2 0.0I

[utama@Qolam latexdoc]$ java Soal5
R  0.3153223623952687 0.0e1 0.0e2 0.0e3  0.5693907716133517Ie1
   0.759187695484469Ie2 0.0Ie3  0.0I
Rrev  0.3153223623952687 0.0e1 0.0e2 0.0e3  -0.5693907716133517Ie1
   -0.759187695484469Ie2 -0.0Ie3  0.0I
r = 2.0e1 1.0e2 2.0e3
a = 1.5e1 2.0e2 0.0e3
a + r = 3.5e1 3.0e2 2.0e3
Hasil perputarannya adalah  0.0 -0.39847032300387747e1 2.7988527422529077e2
                        -1.0038150869899107e3  0.0Ie1 0.0Ie2 0.0Ie3  0.0I

[utama@Qolam latexdoc]$ java Soal6
u = 2.0e1 3.0e2 4.0e3
v = 4.0e1 1.0e2 3.0e3
u^v = 5.0Ie1 10.0Ie2 -10.0Ie3
Luas parallelogram adalah 15.0

[utama@Qolam latexdoc]$ java Soal7
q = -5.0e1 7.0e2 0.0e3
f1 = 4.0e1 0.0e2 1.0e3
f2 = 3.0e1 1.0e2 0.0e3
```

```
F = f1^f2 = -1.0Ie1 3.0Ie2 4.0Ie3
komponen q yang tegak-lurus F adalah -1.0e1 3.0e2 4.0e3
komponen q yang sejajar F adalah  0.0 -4.0e1 4.000000000000001e2 -4.0e3
                                  0.0Ie1 0.0Ie2 0.0Ie3  0.0I

Script done on Sat Jan  6 05:42:45 2001
```