UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

KAO FÉLIX

Álgebra Geométrica aplicada à Simulação de Corpos Rígidos

Trabalho de Graduação

Prof. Dr. Manuel Menezes de Oliveira Neto
Orientador

Leandro Augusto Frata Fernandes
Co-orientador

Porto Alegre, dezembro de 2009

“For centuries the notion that you can only add ’like things’ has been relentlessly impressed on the mind of every schoolboy. It is a kind of mathematical taboo - its real justification unknown or forgotten. It is supposedly obvious that you cannot add apples and oranges or feet and square feet. On the contrary, it is only obvious that addition of apples and oranges is not usually a practic thing to do - unless you are making a salad.”

— DAVID HESTENES

AGRADECIMENTOS

Agradeço ao meu pai Cezar, que a seu modo, me apoiou e esteve do meu lado. Ao meu irmão Theo por me dar oportunidade de servir de exemplo e inspiração para uma pessoa. À minha madrasta Silvana por fazer parte da família.

Agradeço a minha mãe Ângela que mesmo longe faz alcançar seu carinho e apoio. Aos meus avós e tios que sempre foram uma parte importante da minha vida, mesmo com menos contato durante os anos de estudo. Ao meu padrasto Rogério pelas lições.

Agradeço à minha melhor amiga e namorada Letícia Palma Nunes por todo o carinho e apoio. À mãe dela Cenira, por me aturar por muitos dias em sua casa e ainda assim com um acolhedor carinho. Ao irmão dela Alexandre pelos momentos divertidos.

Agradeço ao meu amigo Cleo Oliveira por me ensinar a olhar o mundo de forma pacífica e humilde, e também a preocupar-me com o que realmente tem de valor na vida. Uma lição que continuo até hoje aprendendo.

Agradeço aos meus amigos Márcio Rocha Zacarias, Félix Carvalho Rodrigues e Daniel Emilio Beck com quem passei alguns dos momentos mais especiais do tempo em que estive nessa universidade.

Aos meus orientadores Manuel Menezes de Oliveira Neto e Leandro A. F. Fernandes por toda a ajuda e acompanhamento durante o desenvolvimento desse trabalho.

Por fim, agradeço a todos que fizeram parte da minha vida e que, nessa frágil sequência de eventos, de alguma forma contribuíram para que eu chegasse até aqui.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Álgebra geométrica é uma ferramenta matemática para formulação e resolução de problemas geométricos, sendo uma alternativa para a álgebra linear classicamente utilizada. Com o intuito de demonstrar a utilidade do formalismo, uma engine física para simulação de corpos rígidos foi desenvolvida.

O trabalho apresenta os conceitos de álgebra geométrica necessários para desenvolver a simulação, a base teórica de mecânica de corpos rígidos de forma tradicional e como reformular essa teoria usando álgebra geométrica. São realçadas as diferenças no uso da álgebra geométrica em comparação com as técnicas tradicionais.

Para uso na construção da engine, duas bibliotecas de álgebra geométrica são investigadas: o Geometric Algebra Implementation Generator (Gaigen 2) e a Geometric Algebra Template Library (GATL), desenvolvida na UFRGS. Uma comparação entre as duas é apresentada, incluindo um benchmark de desempenho.

Palavras-chave: Álgebra geométrica, computação gráfica, geometria, simulação física, dinâmica de corpos rígidos.

Geometric Algebra applied to Rigid Body Simulation

ABSTRACT

Geometric algebra is a mathematical tool used in the formulation and resolution of geometric problems, being an alternative to the linear algebra normally used. To demonstrate the usefulness of this formalism, a physics engine for rigid body simulation was developed.

This work presents the geometric algebra concepts needed to develop the simulation, the theoretical foundations of rigid body mechanics in the traditional way and how to reformulate this theory using geometric algebra.

To build the engine, two geometric algebra libraries are investigated: the Geometric Algebra Implementation Generator (Gaigen 2) and the Geometric Algebra Template Library (GATL), developed at UFRGS. A comparison between the two is presented, including a performance benchmark.

Keywords: geometric algebra, computer graphics, geometry, physics simulation, rigid body dynamics.

1 INTRODUÇÃO

A simulação física é essencial para se obter verossimilhança em aplicações que modelam o mundo real ou alguma aproximação dele. Jogos eletrônicos são um dos exemplos mais proeminentes de software que usa esse recurso. As técnicas tradicionais de simulação se baseiam nos fundamentos de mecânica definidos com base em *álgebra linear*. Entretanto o vocabulário da álgebra linear é bastante limitado, resultando em conceitos claramente distintos sendo representados pela mesma entidade matemática, o vetor. A representação de orientações no espaço 3-D também é particularmente problemática e tentativas de representação usando apenas recursos da álgebra linear acabam sendo desajeitadas e contra-intuitivas.

Uma parte fundamental da simulação de corpos rígidos é a detecção e tratamento da colisão entre objetos. Em alto nível, esses objetos são representados por elementos geométricos simples, como linhas, planos e esferas. Ao utilizar geometria descritiva convencional, baseada em álgebra linear, linhas, planos e esferas precisam ser definidos a partir de coleções de pontos, direções e valores escalares. Desse modo, uma linha em 3-D pode ser representada por um ponto e um vetor de direção, ou pela intersecção entre dois planos. Um plano, por outro lado, é usualmente descrito por um vetor normal e uma distância da origem. Uma esfera é representada pelo seu centro e valor do seu raio. Tal representação de elementos geométricos impede o seu uso como primitivas computacionais. Como resultado, problemas geométricos, como detecção de colisão, precisam ser resolvidos considerando o tipo particular de elemento geométrico que está sendo manipulado. Por exemplo, a intersecção entre dois planos é computada usando uma solução diferente da intersecção entre uma linha e um plano, ou qualquer outra combinação de elementos. Alem disso, cada possível resultado precisa ser tratado caso a caso (e.g., a intersecção plano-plano pode resultar em um conjunto vazio, uma linha reta, ou um plano). Alternativamente, é possível usar coordenadas de Plücker para representar pontos, linhas e planos como tipos elementares. No entanto, definir matrizes de transformação para serem aplicadas sobre elementos geométricos nessa representação não é trivial.

Além disso, a representação de transformações por meio de matrizes implica em problemas de instabilidade numérica, exigindo que a matriz seja reortogonalizada a cada frame para não haver distorções nos objetos. Nesse caso, quatérnions oferecem uma alternativa eficiente. Entretanto quatérnions não se conectam bem com outras transformações na álgebra linear, ou com coordenadas de Plücker.

Álgebra geométrica, por outro lado, é um framework matemático que naturalmente generaliza e integra formalismos úteis como números complexos, quatérnions e coordenadas de Plücker de forma consistente. Ela apresenta uma variedade maior de primitivas computacionais com interpretações geométricas claras e intuitivas, diminuindo consideravelmente a necessidade de sobrecarregar o significado de um único tipo de elemento.

Com álgebra geométrica é possível computar diretamente usando elementos geométricos (*e.g.*, direções, pontos, planos, círculos e esferas) como primitivas. Por causa da sua estrutura consistente, o equacionamento aplicado a solução de um problema se estende para dimensionalidades maiores e para todo tipo de elemento geométrico. Diversas aplicações em computação gráfica se beneficiam da intuição geométrica provida por esse formalismo (DORST; FONTIJNE; MANN, 2007a). Por conta de suas características, a álgebra geométrica é uma ótima linguagem matemática para formular fundamentos de mecânica clássica (HESTENES, 1999) e, portanto, é natural que seja um formalismo útil para implementar simulações físicas.

1.1 Objetivo

Esse trabalho tem como objetivo apresentar como a álgebra geométrica pode ser usada na solução de problemas práticos. O exemplo utilizado para fazer essa demonstração é uma *engine física* implementada usando álgebra geométrica. Essa engine realiza simulação de corpos rígidos com foco no uso em jogos e aplicações interativas que não exigem um grau de precisão muito alto na simulação.

1.2 Estrutura do Trabalho

O restante do texto encontra-se estruturado da seguinte forma:

- O capítulo 2 contém uma introdução ao formalismo da álgebra geométrica com foco nas partes utilizadas ao longo do trabalho.
- Os conceitos básicos para a simulação de corpos rígidos da maneira tradicional são descritos no capítulo 3.
- O capítulo 4 mostra como aplicar o formalismo da álgebra geométrica nos problemas envolvidos na simulação de corpos rígidos.
- O capítulo 5 descreve a implementação da engine física, passando por uma revisão de bibliotecas de álgebra geométricas.
- Por fim, o capítulo 6 apresenta as considerações finais desse trabalho.

2 ÁLGEBRA GEOMÉTRICA

Neste capítulo são introduzidos conceito básicos de álgebra geométrica, juntamente com a definição das operações utilizadas no restante do trabalho. Uma discussão mais detalhada é apresentada por Dorst, Fontijne e Mann em seu livro (DORST; FONTIJNE; MANN, 2007a).

A álgebra geométrica parte do conceito familiar de espaço vetorial $\mathbb{R}^n$, onde vetores são as primitivas disponíveis para computação, para o espaço multivetorial $\bigwedge \mathbb{R}^n$, onde é possível incorporar novas primitivas com forte significado geométrico. Introduzindo um *produto externo* (Seção 2.1) entre vetores é possível gerar em $\bigwedge \mathbb{R}^n$ subespaços de dimensionalidade k arbitrária, onde $0 \leq k \leq n$ e n é o número de dimensões do espaço vetorial. Tais subespaços de dimensionalidade arbitrária são primitivas computacionais da álgebra geométrica. Ao atribuir uma interpretação geométrica, os subespaços são convenientemente utilizados na representação de elementos geométricos como pontos, retas, planos, círculos e esferas (Seção 2.7).

Estendendo o *produto interno vetorial* para trabalhar com subespaços k-dimensionais é possível estabelecer relações métricas entre os subespaços (Seção 2.2). Permitindo, por exemplo, o cálculo de ângulos entre elementos k-dimensionais.

A partir da combinação dos produtos interno e externo de vetores é definido o *produto geométrico* (Seção 2.4). Uma das propriedades mais importantes desse produto é ser um produto inversível entre subespaços. Com essa propriedade é possível obter razões entre subespaços com significados geométricos bem definidos. Além disso, o produto geométrico pode ser utilizado na definição de qualquer um dos produtos lineares da álgebra geométrica. A capacidade de representar subespaços de dimensionalidade arbitrária e a existência de um produto tão versátil como o produto geométrico, torna álgebra geométrica um formalismo compacto definido sobre apenas um produto fundamental.

Na álgebra linear, transformações são representadas por matrizes que são aplicadas em vetores representando pontos ou direções. Na álgebra geométrica a representação de transformações se dá através de elementos chamados *versores* (Seção 2.5). Versores são aplicados a subespaços através do produto geométrico e, diferente de matrizes, podem ser usados para transformar qualquer subespaço de maneira direta, sem necessidade de decomposição desses elementos em pontos e direções.

2.1 Produto Externo de Vetores

Por definição, um *espaço vetorial* $\mathbb{R}^n$ consiste em um conjunto de elementos chamados *vetores*. Os vetores de $\mathbb{R}^n$ são combinações lineares de n vetores $\{\mathrm{e_i}\}$ escolhidos para servirem como *base* do espaço vetorial. Um vetor $\mathbf{a}$ em $\mathbb{R}^3$, por exemplo, é escrito como $\mathrm{a} = \alpha_1 \mathbf{e_1} + \alpha_2 \mathrm{e_2} + \alpha_3 \mathbf{e}_3$. A figura 2.1 mostra a representado por meio de uma seta.

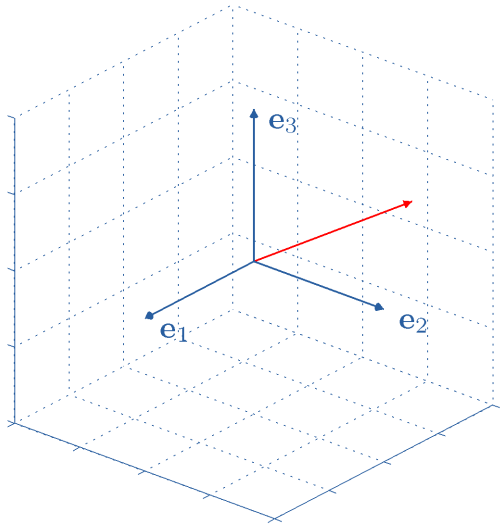

Figura 2.1: Um vetor $\mathbf{a}$ no espaço vetorial $\mathbb{R}^3$. Nessa imagem, os vetores $\mathbf{e}_i$ definem a base do espaço vetorial.

Na álgebra geométrica a representação de subespaços é estendida para dimensões maiores através do *produto externo* de vetores. Um subespaço 2-D, por exemplo, é gerado a partir do produto externo de dois vetores:

$$\mathrm{C}_{\langle 2 \rangle} = \mathbf{a} \wedge \mathbf{b} \tag{2.1}$$

O símbolo $\wedge$ denota o produto externo entre os vetores $\mathbf{a}$ e $\mathbf{b}$. O resultado dessa operação é um 2-blade (os termos blade e subespaços são utilizados de maneira intercalada). A entidade representada por um k-blade é um elemento com uma *atitude*, um *peso* e uma *orientação*. A figura 2.2 mostra um 2-blade representado como um disco. O raio do disco indica o peso do subespaço, a seta no sentido horário indica sua orientação e a atitude é indicada pela disposição do disco no espaço. O produto externo de vetores não se limita à geração de subespaços 2-D. A dimensão dos subespaços pode ser k com $0 \leq k \leq n$.

Como na álgebra geométrica subespaços k-dimensionais são primitivas computacionais, é preciso um novo tipo de base capaz de representar esses subespaços. Note que a base adotada em $\mathbb{R}^n$ suporta apenas a representação de subespaços 1-D (*i.e.*, vetores). Um *espaço multivetorial* $\bigwedge \mathbb{R}^n$ construído a partir do espaço vetorial $\mathbb{R}^n$ usa uma base que supre essas necessidades. A base de $\bigwedge \mathbb{R}^n$ é construída a partir da combinações dos vetores da base $\mathbb{R}^n$ k a k, resultando em 2^n blades de base. Uma base para $\bigwedge \mathbb{R}^3$, por exemplo, é composta por:

$$\left\{ \underbrace{1}_{0\text{-blades (escalares)}}, \quad \underbrace{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3}_{1\text{-blades (vetores)}}, \quad \underbrace{\mathbf{e}_1 \wedge \mathbf{e}_2, \mathbf{e}_1 \wedge \mathbf{e}_3, \mathbf{e}_2 \wedge \mathbf{e}_3}_{2\text{-blades}}, \quad \underbrace{\mathbf{e}_1 \wedge \mathbf{e}_2 \wedge \mathbf{e}_3}_{3\text{-blades}} \right\} \tag{2.2}$$

Em (2.2) estão anotadas as dimensionalidades dos blades de cada porção do espaço. É interessante notar como valores escalares e vetores também são blades de dimensionalidade 0 e 1 respectivamente. A notação utilizada para representar uma porção do espaço

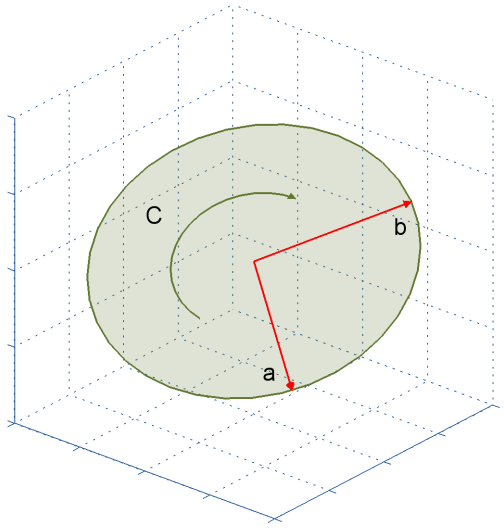

Figura 2.2: O resultado do produto externo $\mathbf{a} \wedge \mathbf{b}$ é um 2-blade, um subespaço 2-D com uma atitude, um peso e uma direção.

com dimensionalidade k é $\bigwedge^k \mathbb{R}^n$. Isso quer dizer, por exemplo, que os vetores residem em $\bigwedge^1 \mathbb{R}^n$ e os 2-blades em $\bigwedge^2 \mathbb{R}^n$.

A ordem dos vetores usada para construir os blades da base é uma questão de convenção, mas converter entre convenções depende apenas da aplicação das propriedades do produto externo (que serão apresentadas a seguir). Uma combinação linear dos blades dessa base é chamada de *multivetor*. Multivetores são utilizados na codificação tanto de subespaços quanto de transformações geométricas (Seção 2.5). Na base do $\bigwedge \mathbb{R}^3$ acima, o 3-blade da base $\mathbf{e}_1 \wedge \mathrm{e}_2 \wedge \mathbf{e_3}$ representa o espaço como um todo. Os blades que tem dimensionalidade correspondente à do espaço são chamados *pseudo escalares* e costumam ser denotados por:

$$\mathbf{I}_{\langle n \rangle} = \mathbf{e_1} \wedge \mathbf{e}_2 \wedge \ldots \wedge \mathrm{e_n} \tag{2.3}$$

Formalmente, o produto externo é um mapeamento:

$$\wedge : \bigwedge^r \mathbb{R}^n \times \bigwedge^s \mathbb{R}^n \to \bigwedge^{r+s} \mathbb{R}^n \tag{2.4}$$

onde $\bigwedge^r \mathbb{R}^n$, $\bigwedge^s \mathbb{R}^n$ e $\bigwedge^{r+s} \mathbb{R}^n$ são os espaços lineares formados pelos blades de base com dimensionalidade r, s e $r + s$ respectivamente.

O produto externo é definido pelo seguinte conjunto de propriedades:

anti simetria $\mathrm{a} \wedge \mathbf{b} = -\mathrm{b} \wedge \mathrm{a}$ e portanto $\mathrm{c} \wedge \mathbf{c} = 0$

distributividade $\mathrm{a} \wedge (\mathbf{b} + \mathbf{c}) = \mathbf{a} \wedge \mathrm{b} + \mathbf{a} \wedge \mathbf{c}$

associatividade $\mathbf{a} \wedge (\mathrm{b} \wedge \mathrm{c}) = (\mathrm{a} \wedge \mathrm{b}) \wedge \mathbf{c}$

escalares comutam $\mathrm{a} \wedge (\beta \mathrm{b}) = \beta(\mathbf{a} \wedge \mathbf{b})$

Para um conjunto de k-blades com uma mesma atitude, é possível realizar comparações através do peso de cada blade. Porém entre blades de atitude diferente não há modos de medir a relação entre eles usando apenas o espaço multivetorial com o produto externo. Para expressar essa relação métrica existem produtos métricos. Esse tipo de produto será introduzido na próxima seção.

2.2 Métrica e Produtos Internos

Para comparar o peso ou o ângulo entre dois subespaços com atitudes diferentes, é necessário um produto que dependa da métrica de $\mathbb{R}^n$. O *produto interno de vetores* multiplica dois vetores arbitrários e retorna um valor escalar que caracteriza a relação métrica entre eles

$$\mathbf{a} \cdot \mathrm{b} = \mathsf{Q}(\mathbf{a}, \mathbf{b}) \tag{2.5}$$

Ele é denotado pelo símbolo de ponto $\cdot$ e tem as seguintes propriedades

simetria $\mathbf{a} \cdot \mathrm{b} = \mathrm{b} \cdot \mathbf{a}$

distributividade $\mathrm{a} \cdot (\mathrm{b} + \mathrm{c}) = \mathbf{a} \cdot \mathrm{b} + \mathrm{a} \cdot \mathbf{c}$

escalares comutam $\mathrm{a} \cdot (\beta \mathrm{b}) = \beta(\mathrm{a} \cdot \mathrm{b})$

A função Q na equação (2.5) é uma função que retorna um escalar e caracteriza a métrica do espaço $\mathbb{R}^n$. Essa função pode ser implementada de maneira prática através de uma *matriz métrica*

$$\mathrm{M} = \begin{bmatrix} \mu_{11} & \mu_{12} & \cdots & \mu_{1n} \\ \mu_{21} & \mu_{22} & \cdots & \mu_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \mu_{n1} & \mu_{n2} & \cdots & \mu_{nn} \end{bmatrix} \tag{2.6}$$

na qual $\mu_{ij} = \mathrm{e_i} \cdot \mathbf{e_j}$ para $1 \leq i, j \leq n$ e $\mathbf{e_i}$ é um vetor da base. No caso em que M é a matriz identidade, a métrica assumida é a métrica euclidiana. Nesse caso o produto interno de vetores corresponde ao produto escalar (*dot product*) utilizado em álgebra linear, que pode ser usado para calcular o menor ângulo entre vetores (Figura 2.3, esquerda):

$$\mathbf{a} \cdot \mathrm{b} = \|a\| \, \|b\| \, cos\theta \tag{2.7}$$

O *produto escalar* da álgebra geométrica (denotado por $*$) é a generalização do produto interno de vetores para subespaços arbitrários de mesma dimensionalidade. Na métrica euclidiana ele pode ser usado para calcular o ângulo entre subespaços arbitrários (Fig. 2.3, direita) através da equação:

$$theta = \cos^{-1}\left(\frac{\mathbf{A}_{\langle 2 \rangle} * \mathrm{B}_{\langle 2 \rangle}}{\left\|\mathbf{A}_{\langle 2 \rangle}\right\| \, \left\|\mathrm{B}_{\langle 2 \rangle}\right\|}\right) \tag{2.8}$$

Com o produto escalar, o produto interno de vetores torna-se apenas um caso particular para blades com dimensionalidade 1.

A álgebra geométrica possui outros produtos métricos. Um dos mais importantes é a *contração à esquerda* definido por

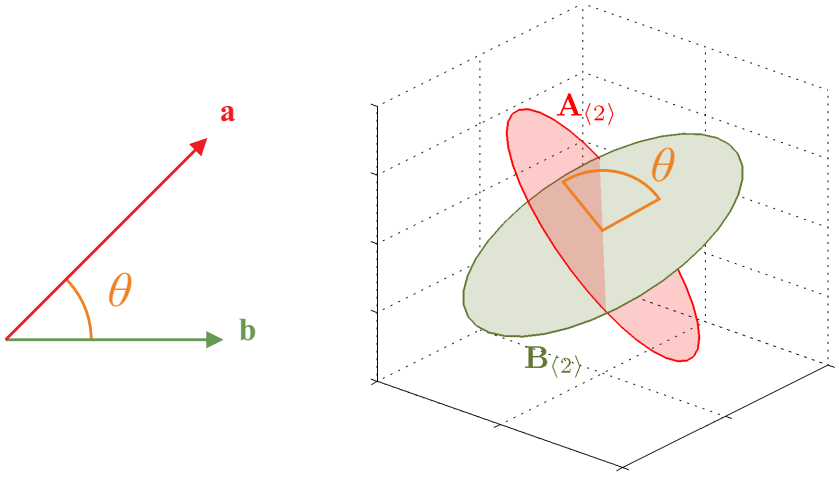

Figura 2.3: Ao assumir métrica euclidiana, o produto interno entre vetores pode ser utilizado no cálculo do menor ângulo entre dois vetores com a equação (2.7) (esquerda), do mesmo modo que o produto escalar pode ser utilizado no cálculo do menor ângulo entre subespaços de mesma dimensionalidade (direita) usando a equação (2.8).

$$\mathbf{A}_{\langle r \rangle} \rfloor \mathrm{B}_{\langle s \rangle} = \mathrm{C}_{\langle s-r \rangle} \tag{2.9}$$

A interpretação geométrica desse produto é remover de $\mathbf{B}_{\langle s \rangle}$ a parte mais "parecida" com $\mathbf{A}_{\langle r \rangle}$, retornando um blade $\mathbf{C}_{\langle s-r \rangle}$ contido em $\mathrm{B}_{\langle s \rangle}$ que é "menos parecido" com $\mathrm{A}_{\langle r \rangle}$ na métrica definida.

2.3 Intersecção e União de Subespaços

A álgebra geométrica possui duas operações muito úteis para manipular subespaços como entidades geométricas. Essas operações, o *join* e o *meet*, têm como significado geométrico a união e a intersecção entre dois subespaços, respectivamente.

Supondo dois blades $\mathbf{A}$ e $\mathbf{B}$ que contém um blade em comum $\mathbf{M}$. Então M pode ser fatorada dos blades A e B, o que pode ser escrito da seguinte forma:

$$\mathbf{A} = \mathrm{A}' \wedge \mathbf{M} \tag{2.10}$$

$$\mathrm{B} = \mathrm{M} \wedge \mathrm{B}' \tag{2.11}$$

Os blades $\mathbf{A}$ e B residem dentro de um outro blade $\mathbf{J}$ que é o menor múltiplo comum entre eles em relação ao produto externo. Esse blade $\mathbf{J}$ é o *join* entre os dois subespaços e é denotado por $\mathrm{J} = \mathbf{A} \cup \mathbf{B}$. O blade $\mathbf{M}$ em comum entre os dois subespaços é o seu *meet* denotado $\mathbf{M} = \mathbf{A} \cap \mathbf{B}$. A notação familiar da teoria de conjuntos reflete os significados de união e intersecção geométrica dessas duas operações.

A seção 4.2.1 irá mostrar uma aplicação do *meet* para a detecção de colisão entre formas geométricas. A figura 4.2 mostra o significado geométrico dessa operação aplicada sobre esferas.

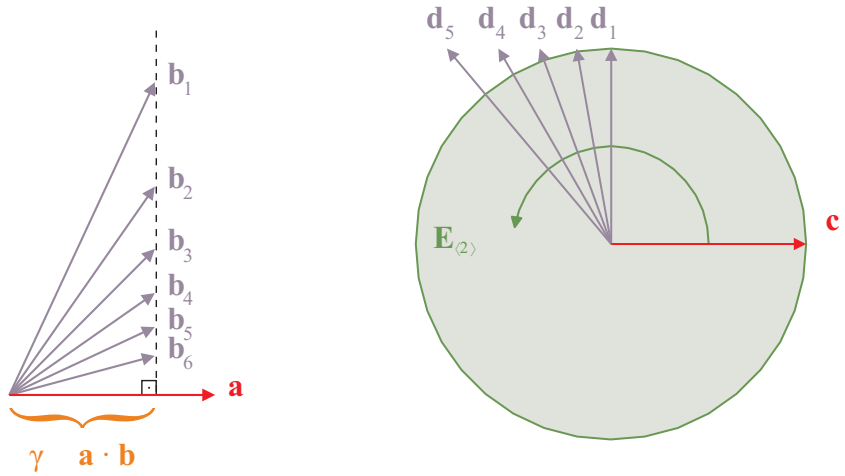

Figura 2.4: Os produtos internos (esquerda) e externo (direita) não inversíveis se combinam para criar um produto inversível.

2.4 Produto Geométrico

O *produto geométrico* é o produto fundamental da álgebra geométrica por duas razões: a primeira porque é um produto inversível e a segunda porque todos os outros produtos lineares e alguns não lineares da álgebra geométrica podem ser expressados em termos dele.

A motivação de se ter um produto inversível para vetores pode ser vista ao se pensar no problema de encontrar o valor de um vetor desconhecido $\mathbf{b}$ quando tudo que se conhece é o valor γ de seu produto interno com um vetor $\mathbf{a}$, ou seja

$$\gamma = \mathbf{a} \cdot \mathbf{b} \tag{2.12}$$

O produto interno não é inversível (Fig. 2.4 esquerda), isso quer dizer que existem inúmeros valores para b que satisfazem (2.12). Da mesma forma, se quisermos determinar um vetor $\mathbf{d}$ sabendo apenas que seu produto com o vetor $\mathbf{c}$ é igual a um $\mathbf{E}_{\langle 2 \rangle}$, ou seja

$$\mathbf{E}_{\langle 2 \rangle} = \mathbf{c} \wedge \mathbf{d} \tag{2.13}$$

temos outro caso onde não há uma solução única (Fig. 2.4 direita). No entanto, ao avaliar o significado geométrico de (2.12) e (2.13) se percebe que os dois produtos são complementares. O produto interno $\mathbf{a} \cdot \mathrm{b}$ fixa um valor para o tamanho da projeção de b sobre a. Os inúmeros valores possíveis para $\mathbf{b}$ "andam" por cima de uma linha fixa (Fig. 2.4, esquerda). Enquanto isso, o produto externo fixa uma linha ortogonal à linha fixada pelo produto interno. Como $\mathbf{E}_{\langle 2 \rangle}$ é conhecido, seu peso é fixo e o vetor $\mathbf{d}$ precisa estar a uma distância ortogonal fixa de $\mathbf{c}$ (Fig. 2.4 direita). Combinando as duas informações é possível se determinar um único vetor.

Esse raciocínio é exatamente o que leva a definição do produto geométrico a ser

$$\mathrm{a}\,\mathbf{b} = \mathrm{a} \cdot \mathbf{b} + \mathbf{a} \wedge \mathrm{b} \tag{2.14}$$

Por sua importância na álgebra geométrica o produto geométrico é denotado por um espaço entre os operandos. Usando o produto geométrico sob métrica euclidiana, é possível determinar o vetor x tal que $\mathbf{a}\,\mathbf{x} = B$. A solução é simplesmente $B \,/\, \mathrm{a}$, onde $/$ denota

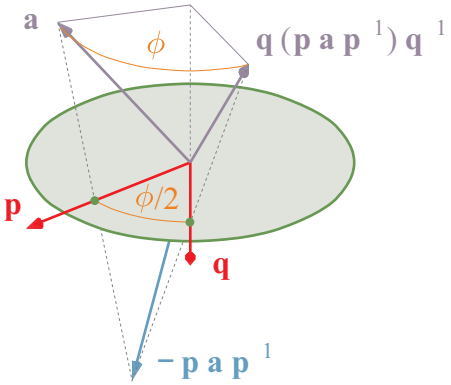

Figura 2.5: Duas reflexões formando uma rotação. A reflexão de a em p e logo em seguida em q é equivalente a uma rotação do dobro do ângulo entre p e q.

o produto geométrico inverso. A definição do produto geométrico entre multivetores arbitrários pode ser vista em (FERNANDES; OLIVEIRA, 2007).

2.5 Versores

O produto geométrico é o produto utilizado para aplicar transformações na álgebra geométrica. Sendo $\mathbf{a}$ e $\mathbf{b}$ dois vetores, a operação

$$\mathrm{a}' = \mathrm{b}\,\mathbf{a}\,/\,\mathbf{b} \tag{2.15}$$

resulta na reflexão de $\mathbf{a}$ em $\mathbf{b}$. Na equação (2.15), $\mathbf{b}$ é dito ser um refletor. Geometricamente, qualquer transformação ortogonal pode ser representada por um número de reflexões. Por exemplo, um número par de reflexões em vetores unitários e inversíveis resulta em uma rotação (Figura 2.5).

A aplicação de transformações sempre tem a forma desse "sanduíche" com o produto geométrico. Multivetores que representam transformações são chamados de versores. Um versor $\boldsymbol{V}$ pode ser aplicado a qualquer subespaço $\mathrm{X}_{\langle k \rangle}$:

$$\mathrm{X}'_{\langle k \rangle} = V\,\mathbf{X}_{\langle k \rangle}\,/\,V \tag{2.16}$$

O versor que realiza uma rotação se chama rotor e é denotado por $\boldsymbol{R}$. Um rotor que faz uma rotação com ângulo ϕ no plano definido pelo 2-blade unitário $\mathrm{B}_{\langle 2 \rangle}$ pode ser obtido através de uma operação de exponenciação:

$$\boldsymbol{R_\phi} = exp(-\frac{\phi}{2}\mathrm{B}_{\langle 2 \rangle}) = cos(\frac{\phi}{2}) - \mathrm{B}_{\langle 2 \rangle}\, sin(\frac{\phi}{2}) \tag{2.17}$$

Rotações podem ser compostas simplesmente aplicando um rotor sobre o resultado de outra aplicação de rotor. Com base na propriedade de associatividade do produto geométrico

$$\boldsymbol{R_2}\,(R_1 \mathrm{X}_{\langle k \rangle}\,/\,\boldsymbol{R}_1)\,/\,R_2 = (R_2\,\boldsymbol{R}_1)\,\mathbf{X}_{\langle k \rangle}\,/\,(\boldsymbol{R}_1\,R_2) \tag{2.18}$$

conclui-se que o rotor que realiza a rotação definida por $\boldsymbol{R}_1$ composta com a rotação definida por R_2 é simplesmente

$$\boldsymbol{R} = R_2\,\boldsymbol{R_1} \tag{2.19}$$

2.6 Forma Direta e Forma Dual

Em alguns modelos de álgebra geométrica (*e.g.*, o modelo conforme que será introduzido na seção 2.7) existem duas formas de representar objetos: a forma *direta* e a forma *dual*. A forma dual de um objeto é seu complemento ortogonal em relação ao espaço como um todo. A motivação do uso das duas formas é que cada uma delas codifica certas propriedades dos objetos representados de forma mais conveniente.

Sendo A um blade representando um objeto na forma direta, seu dual é obtido através do operador de dual definido pela seguinte equação:

$$\mathrm{B} = \mathrm{A}^* = \mathbf{A}\,\mathrm{I}_{\langle n\rangle}^{-1} \tag{2.20}$$

na qual $\mathrm{I}_{\langle n\rangle}$ é o pseudoescalar no espaço e $\mathbf{I}_{\langle n\rangle}^{-1}$ é seu *inverso*. O inverso de um k-blade $\mathrm{X}_{\langle k\rangle}$ é definido por:

$$\mathbf{X}_{\langle k\rangle}^{-1} = \frac{\widetilde{\mathbf{X}_{\langle k\rangle}}}{\left\|\mathbf{X}_{\langle k\rangle}\right\|^2} \tag{2.21}$$

onde $\widetilde{\mathrm{X}_{\langle k\rangle}}$ é o *reverso* do blade $\mathbf{X}_{\langle k\rangle}$ que por sua vez é definido por:

$$\widetilde{\mathbf{X}_{\langle k\rangle}} = (-1)^{k(k-1)\,/\,2}\mathbf{X}_{\langle k\rangle} \tag{2.22}$$

O reverso decompõe o blade $\mathrm{X}_{\langle k\rangle}$ em fatores vetoriais $\mathbf{x_1} \wedge \mathbf{x_2} \wedge \ldots \wedge \mathrm{x_k}$ e reverte a ordem desses fatores, resultando em $\widetilde{\mathrm{X}_{\langle k\rangle}} = \mathbf{x_k} \wedge \ldots \wedge \mathrm{x}_2 \wedge \mathrm{x}_1$.

A forma direta pode ser obtida novamente a partir da forma dual usando o operador de *undual*:

$$\mathrm{A} = \mathbf{B}^{-*} = \mathrm{B}\,\mathrm{I}_{\langle n\rangle} \tag{2.23}$$

2.7 Modelo Conforme da Álgebra Geométrica

É possível adotar uma convenção que atribui uma interpretação geométrica aos blades e versores do espaço multivetorial considerado, assim como a escolha de diferentes métricas a fim de aplicar o formalismo da álgebra geométrica a problemas práticos. Essas convenções são os *modelos de geometria* da álgebra geométrica. O modelo de geometria mais simples é chamado *modelo vetorial com métrica euclidiana*. Esse modelo foi utilizado na explicação e nas ilustrações das subseções anteriores.

O modelo mais poderoso descrito na literatura se chama *modelo conforme*. O modelo conforme usa um espaço representacional com duas dimensões a mais do que o espaço representado e uma métrica especial. As duas dimensões adicionais e a métrica diferenciada providenciam k-blades com interpretações geométricas mais variadas.

Os vetores de base para o modelo conforme n-D são $\{e_i\}$ com $1 \leq i \leq n$ e mais dois vetores com interpretações geométricas especiais: o (pronunciado *no*), interpretado como o ponto na origem e $\boldsymbol{\infty}$ (pronunciado *ni*) que é interpretado como o ponto no infinito. A existência de dois vetores representando pontos não surge por acaso. É apenas um exemplo de uma capacidade do modelo conforme não presente em espaços vetoriais: a representação de pontos euclidianos através de uma entidade própria para esse fim. De fato, $\mathbf{o}$ e ∞ são vetores apenas no espaço representacional, na interpretação euclidiana eles são vistos como pontos. Diferentes pontos também são representados por vetores com certas características especiais definidas pela métrica do espaço.

A métrica do modelo conforme estabelece que o produto interno de dois vetores $\mathbf{p}$ e $\mathbf{q}$ representando pontos finitos p e q (a convenção adotada aqui é denotar um ponto conceitual como p, enquanto o vetor que representa o ponto é denotado por p, em negrito) é proporcional ao quadrado da distância euclidiana entre esses pontos, quando os pontos estão normalizados. Um ponto está normalizado quando o coeficiente associado a $\mathbf{o}$ é igual a 1. O peso de um ponto é obtido através do seguinte produto interno:

$$-\boldsymbol{\infty} \cdot \mathrm{p} \tag{2.24}$$

Dessa forma, o produto interno entre dois vetores $\mathbf{p}$ e $\mathbf{q}$ representando pontos fica definido como

$$\frac{\mathrm{p}}{-\boldsymbol{\infty} \cdot \mathrm{p}} \cdot \frac{\mathbf{q}}{-\infty \cdot \mathbf{q}} = \frac{1}{2} d_E^2(p, q) \tag{2.25}$$

onde $d_E^2(p, q)$ corresponde ao quadrado da distância euclidiana entre os pontos p e q. A divisão de cada ponto pelo seu peso em (2.25) ocorre para garantir que um ponto está normalizado. Uma consequência de (2.25) é que pontos são representados por *vetores nulos* (ou seja, um vetor $\mathbf{p}$ tal que $\mathrm{p} \cdot \mathrm{p} = 0$), pois um ponto está a uma distância 0 de si mesmo. Para o caso dos vetores o e ∞ temos que $\mathrm{o} \cdot \boldsymbol{\infty} = -1$. A matriz de métrica do modelo conforme incorpora esses resultados e tem a seguinte forma:

	o	$\mathbf{e_1}$	$\mathrm{e_2}$	$\cdots$	$\mathrm{e_n}$	∞
$\mathbf{o}$	0	0	0	$\cdots$	0	-1
$\mathbf{e_1}$	0	1	0	$\cdots$	0	0
$\mathbf{e_2}$	0	0	1	$\cdots$	0	0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
$\mathbf{e_n}$	0	0	0	$\cdots$	1	0
$\boldsymbol{\infty}$	-1	0	0	$\cdots$	0	0

(2.26)

O conjunto de vetores $\{\mathbf{e}_1, \mathrm{e}_2, \ldots, \mathrm{e}_\mathrm{n}\}$ representam a porção euclidiana do espaço. É notável que a porção da matriz de métrica correspondente a esses vetores é como uma matriz identidade. Os elementos que se diferenciam são os correspondentes aos vetores especiais o e $\boldsymbol{\infty}$ que possuem uma relação métrica especial. Para representar um ponto a partir de um vetor puramente euclidiano $\vec{p}$, usamos a seguinte fórmula:

$$\mathbf{p} = \mathrm{o} + \vec{p} + \frac{1}{2}\vec{\boldsymbol{p}}^2 \boldsymbol{\infty} \tag{2.27}$$

Uma propriedade útil de (2.27) é que, apesar de $\boldsymbol{\infty}$ ser um ponto com propriedades especiais, $\mathbf{o}$ pode ser substituído por outro ponto qualquer que se deseje usar como origem.

2.7.1 Objetos do Modelo Conforme

A seção anterior introduziu o modelo conforme e como pontos euclidianos são representados através de vetores nulos. Entretanto não existem apenas vetores nulos no espaço representativo do modelo conforme. Investigando a propriedade de vetores não nulos conclui-se que eles podem também representar os duais de planos e esferas. Vetores sem o componente o na forma

$$\pi = \vec{\boldsymbol{n}} + \delta\boldsymbol{\infty} \tag{2.28}$$

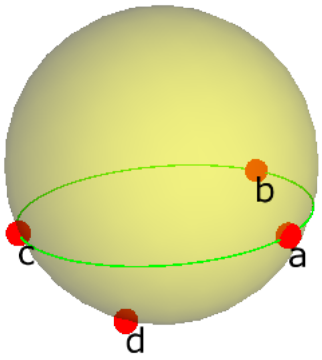

Figura 2.6: Exemplos de *rounds*: um círculo formado por $\mathbf{a} \wedge \mathrm{b} \wedge \mathrm{c}$ e uma esfera formada por $\mathbf{a} \wedge \mathrm{b} \wedge \mathbf{c} \wedge \mathrm{d}$.

representam planos duais sendo que $\vec{\boldsymbol{n}}$ é o vetor puramente euclidiano representando a normal do plano dual π e $\delta \,/\, \|\vec{\boldsymbol{n}}\|$ é a sua distância para a origem.

Já vetores não-nulos no caso geral representam esferas duais. Usando

$$\sigma = \mathbf{c} - \frac{1}{2} r^2 \boldsymbol{\infty} \tag{2.29}$$

obtemos a esfera dual σ com centro no ponto representado pelo vetor nulo $\mathbf{c}$ e raio r.

Esferas e planos na forma direta (*i.e.*, o oposto da forma dual) fazem parte de classes de objetos do modelo conforme chamadas de *rounds* e *flats*. Os *rounds* são formados a partir do produto externo de pontos finitos.

$$\mathbf{R} = \alpha \mathrm{p}_0 \wedge \mathrm{p}_1 \wedge \cdots \wedge \mathbf{p}_\mathrm{k} \tag{2.30}$$

Dependendo do número de pontos usados na combinação um tipo diferente de entidade geométrica é representada. Por exemplo:

$$\mathrm{C} = \mathbf{a} \wedge \mathbf{b} \wedge \mathbf{c} \tag{2.31}$$

é o círculo passando pelos pontos a, b e c. Se adicionarmos mais um ponto d teremos a esfera:

$$\boldsymbol{\Sigma} = \mathbf{a} \wedge \mathbf{b} \wedge \mathbf{c} \wedge \mathrm{d} \tag{2.32}$$

cuja superfície contém os pontos a, b, c e d. A figura 2.6 ilustra rounds formados a partir de pontos. A adição de mais pontos resulta em hiperesferas e outros análogos de esferas com maior dimensionalidade.

Um k-*flat* é um subespaço planar de dimensão k com um deslocamento da origem. Linhas e planos são exemplo de 1-*flats* e 2-*flats* respectivamente. No modelo conforme *flats* são representados através do produto externo de k pontos com o ponto no infinito $\boldsymbol{\infty}$, assumindo a forma

$$\mathbf{X} = \alpha(\mathrm{p_0} \wedge \mathrm{p}_1 \wedge \cdots \wedge \mathrm{p}_\mathbf{k}) \wedge \boldsymbol{\infty} \tag{2.33}$$

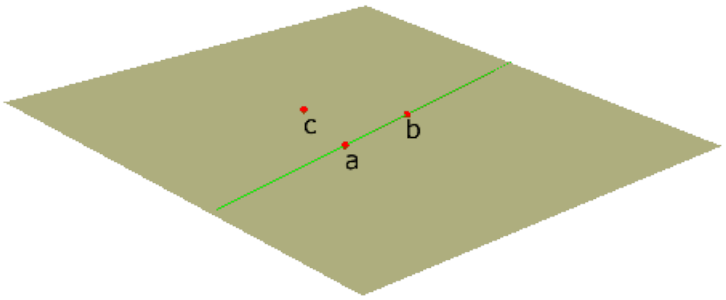

Figura 2.7: Exemplos de *flat*: uma definida por $\mathrm{a} \wedge \mathrm{b} \wedge \infty$ e um plano definido por $\mathrm{a} \wedge \mathrm{b} \wedge \mathbf{c} \wedge \infty$. O ponto no infinito ∞ não está representado na imagem.

O *flat* definido em (2.33) contém os pontos $\{p_0, p_1, \ldots, p_k\}$, logo a equação

$$\mathbf{L} = \mathbf{a} \wedge \mathbf{b} \wedge \infty \tag{2.34}$$

define a linha L que passa pelos pontos a e b e a equação

$$\mathbf{\Pi} = \mathbf{a} \wedge \mathbf{b} \wedge \mathbf{c} \wedge \infty \tag{2.35}$$

define o plano Π que contém os pontos a, b e c. A figura 2.7 ilustra esses dois exemplos. Como no caso dos *rounds*, adicionar mais pontos resulta em formas planares de maior dimensionalidade.

Um *flat* criado com apenas um ponto, ou seja, da forma $\mathrm{p} \wedge \infty$, é chamado ponto *flat*.

2.8 Discussão

Esse capítulo introduziu o básico necessário para o entendimento do restante do trabalho. Foi visto como os produtos da álgebra geométrica permitem o uso de subespaços como primitivas computacionais e fornecem significados geométricos bem definidos e consistentes para a manipulação de subespaços.

A representação de transformações da álgebra geométrica foi apresentada na forma de versores, elementos de dimensionalidade mista que são aplicados sobre subespaços através do produto geométrico. Os rotores, versores que representam rotações, são especialmente importantes para esse trabalho. Em 3-D os rotores estão fortemente relacionados com os quatérnions e por isso são úteis como uma representação para orientação de objetos. Diferente dos quatérnions, eles estão fortemente integrados na álgebra e não é necessário fazer conversões entre formalismos.

Com o modelo conforme a interpretação geométrica da álgebra se eleva a um novo patamar, com representação direta de objetos geométricos (*e.g.*, pontos, retas, círculos, planos e esferas) como primitivas computacionais e uma clara distinção entre vetores e pontos. Além disso, a representação de transformações pode ser interpretada de forma geométrica de maneira mais intuitiva do que a codificação tradicional por matrizes.

3 SIMULAÇÃO DA MECÂNICA DE CORPOS RÍGIDOS

Simulações físicas são usadas em diversas aplicações de computação visual, tanto para observar fenômenos reais em aplicações científicas, como para obter um grau desejável de realismo e semelhança com o mundo real dentro de jogos ou aplicações interativas. Uma simulação extensa de todos os aspectos da realidade não é viável, por isso as aplicações se concentram em simular algum aspecto específico, como dinâmica de fluidos, aerodinâmica, mecânica de corpos rígidos ou até mesmo um conjunto desses aspectos.

O realismo desejado em uma simulação varia conforme seu foco. Em aplicações científicas se deseja que a simulação demonstre com a maior fidelidade possível o modo com que um fenômeno real se comportaria. Erros devem ser controlados e mantidos dentro de uma tolerância aceitável, porque a obtenção de um resultado incorreto pode causar um desastre quando se desejar obter o mesmo resultado na realidade. Já em uma aplicação interativa, como um jogo, se almeja um realismo apenas suficiente para convencer o jogador do que se passa no mundo modelado da simulação. A engine apresentada nesse trabalho será apropriada apenas para uso nesse último caso.

O tipo de simulação mais usada em jogos e aplicações interativas é a de mecânica de corpos rígidos. A mecânica de corpos rígidos descreve como corpos sólidos se movem no espaço de acordo com a ação de forças. Uma biblioteca implementando uma simulação desse comportamento, na verdade, equipa o programador de uma aplicação com um framework que permite simular uma grande quantidade de fenômenos físicos observados no mundo real. Além disso, há uma grande flexibilidade para se manipular os corpos de acordo com forças que não ocorrem no mundo real de modo geral, o que em jogos é algo desejável, já que é comum os mundos virtuais representados possuírem leis físicas muito diferentes da realidade.

As leis fundamentais definidas na mecânica clássica para descrever o comportamento de corpos rígidos são descritas aqui de maneira informal, comparada com a literatura específica da área (GOLDSTEIN; POOLE; SAFKO, 2000). Entretanto, essa postura simplista costuma ser assumida no caso de simulações físicas para jogos, conforme pode ser visto em (MILLINGTON, 2007) e (PALMER, 2005).

A seção 3.1 apresenta as leis físicas que governam o movimento de partículas, pois elas dão a fundamentação para a simulação de corpos rígidos. As diferenças na simulação de partículas e corpos rígidos aparece na seção 3.2, onde são apresentadas as equações usadas na simulação do movimento angular. Na seção 3.4 será descrito o comportamento de corpos que colidem uns contra os outros e como mudar as variáveis cinemáticas de um corpo para responder a essa colisão. No presente capítulo, os conceitos aplicados à simulação da mecânica dos corpos rígidos são apresentados por meios convencionais. No capítulo 4 esses conceitos são reformulados utilizando álgebra geométrica.

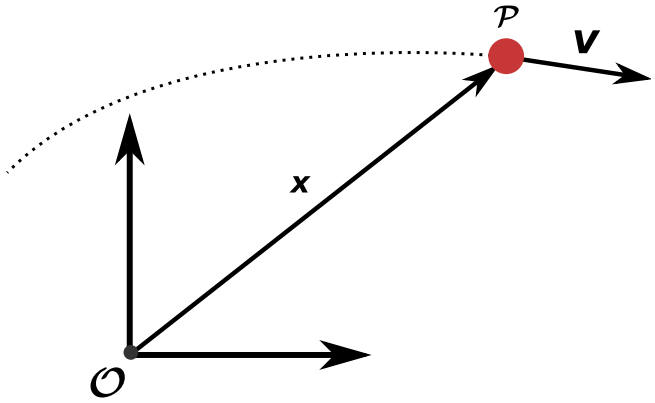

Figura 3.1: Uma partícula P com vetor de posição $\mathbf{x}$ se movendo com velocidade $\mathbf{v}$. A linha pontilhada indica a trajetória da partícula num dado período de tempo.

3.1 Movimento de Partículas

Uma partícula é um ponto no espaço com uma *massa* representada pelo valor escalar m e uma *posição* representada pelo vetor de deslocamento $\mathbf{x}$ em respeito a uma origem O. A posição de uma partícula pode variar no tempo, por isso é útil escrever x como uma função do tempo $\mathbf{x}(t)$. A linha formada pelos valores de $\mathbf{x}(t)$ num período de tempo é a *trajetória* da partícula. A derivada primeira dessa função é chamada de *velocidade* e é igual a

$$\mathbf{v} = \dot{\mathbf{x}} = \frac{\delta \mathbf{x}}{\delta t} \tag{3.1}$$

O *momento linear* de uma partícula é definido como sendo o produto de sua massa e sua velocidade:

$$\mathrm{p} = m\mathbf{v} \tag{3.2}$$

A forma com que a velocidade de uma partícula varia se chama *aceleração*. A aceleração então é a derivada primeira da velocidade e a derivada segunda da posição:

$$a = \dot{\mathrm{v}} = \frac{\delta \mathrm{v}}{\delta t} = \ddot{\mathbf{x}} \tag{3.3}$$

A notação com pontos $\dot{\mathrm{x}}$ e $\ddot{\mathbf{x}}$ é usual da física para denotar derivadas em relação ao tempo. O número de pontos corresponde a ordem da derivada.

A *segunda lei do movimento de Newton* estabelece que o movimento de uma partícula no espaço depende apenas de sua interação com outras partículas. A interação da partícula com objetos externos é descrita através de forças. A força atuante em um corpo é definida como a derivada do seu momento linear

$$\mathbf{f} = \dot{\mathbf{p}} = m\mathbf{a} \tag{3.4}$$

A força total F que atua em uma partícula num instante de tempo t é dada pela soma vetorial de todas as forças individuais $\mathbf{f}_\mathrm{i}$ atuantes no mesmo instante de tempo

$$\mathrm{F} = \sum_{i=1}^{n} \mathbf{f}_\mathrm{i} \tag{3.5}$$

O teorema da conservação para o momento linear de uma partícula garante que se a força total $\mathrm{F} = 0$ então $\dot{\mathbf{p}} = 0$ e o momento linear se conserva. Isso quer dizer que a única forma de alterar o momento linear de uma partícula é através de forças.

3.2 Corpos Rígidos

Um corpo rígido possui uma posição e uma massa, assim como uma partícula, e também possui outras propriedades. O *centro de massa* C de um corpo é o ponto onde toda a massa pode ser considerada acumulada para propósito dos cálculos. Sua posição é definida pela média das posições de cada partícula do corpo, ponderada pela massa de cada partícula. O movimento do centro de massa é descrito da mesma forma que o movimento linear de uma partícula. Ou seja, a aplicação de forças em um objeto faz com que o seu centro de massa se mova linearmente de forma equivalente ao que uma partícula se moveria ao ter a mesma força aplicada sobre ela.

Além da posição, corpos rígidos possuem uma *atitude* (chamada *orientação* em textos de física tradicionais, mas aqui o nome de atitude é usado para manter correspondência com o mesmo conceito para k-blades), que pode ser vista como uma rotação do objeto em relação a uma base de referência. O *momento angular* de um objeto é descrito pela equação:

$$\mathbf{L} = I\boldsymbol{\Omega} \tag{3.6}$$

na qual $\boldsymbol{\Omega}$ corresponde a *velocidade angular* do corpo rígido e I é o seu *momento de inércia*. A derivada da velocidade angular é a *aceleração angular* denotada por $\boldsymbol{\omega}$.

A versão angular da segunda lei de Newton define o *torque* de um corpo rígido como sendo

$$\gamma = \dot{\mathrm{L}} = I\boldsymbol{\omega} \tag{3.7}$$

O torque é um análogo angular da força e o momento de inércia é o análogo da massa. Assim como a massa de um corpo descreve a dificuldade de fazer um corpo mudar seu momento linear através da aplicação de uma força, o momento de inércia descreve a dificuldade de fazer um corpo girar ao redor de seu centro de massa. Em 3-D, o momento de inércia é um tensor na forma de uma matriz 3×3. O produto usado entre I e ω ou $\boldsymbol{\Omega}$ é um produto de matriz por vetor coluna.

Quando forças são aplicadas em um corpo rígido elas geram um torque

$$\gamma = \mathrm{r} \times \mathrm{f} \tag{3.8}$$

O vetor r é o deslocamento do ponto de aplicação da força $\mathbf{f}$ em relação ao centro de massa do objeto. A símbolo $\times$ denota o *cross-product* entre vetores, definido no espaço 3-D. A partir dessa equação se deduz que uma força aplicada ao centro de massa do objeto irá gerar um torque resultante nulo.

$$\gamma = \mathbf{0} \times \mathbf{f} = \mathbf{0} \tag{3.9}$$

Na seção anterior foi dito como todas as forças $\mathrm{f_i}$ agindo sobre um corpo podem ser somadas gerando uma única força F equivalente, conforme descrito na equação (3.5). O mesmo acontece para torques γ_i, que podem ser somados formando um único torque total Γ. Entretanto, um cuidado especial deve ser tomado. Enquanto as forças F é equivalente

a aplicação de cada força f_i individualmente, o torque que resulta de cada força f_i *não* é o mesmo que o torque gerado pela aplicação da força total F. Portanto o torque γ_i precisa ser computado para cada força $\mathrm{f_i}$ individualmente antes de ser acumulado em Γ. Uma maneira intuitiva de verificar isso é pensar em um botão de volume giratório onde os dedos aplicam forças opostas que somadas se tornam próximas de zero, mas o torque resultante faz com que o botão gire.

3.3 Integração Numérica

A segunda lei de Newton relaciona as forças atuantes sobre um objeto com a derivada segunda de sua posição a aceleração. Isso significa que sabendo as forças que atuam em um objeto em um dado instante de tempo é possível obter sua posição através de integração. Em uma simulação física para jogos são geralmente usados métodos de integração numérica passo-a-passo. O método de integração mais simples desse tipo é o método de Euler. Esse método se baseia no fato de que uma aproximação de uma função pode ser obtida "andando-se" pequenos passos na direção de sua derivada.

Tendo $f'(t)$, é possível obter uma aproximação de $f(t)$ usando:

$$f(t+h) = f(t) + hf'(t) \tag{3.10}$$

Quanto menor h for, melhor será a aproximação obtida. A simplicidade do método de Euler tem um custo em precisão. Os valores de h para se obter uma simulação com uma boa precisão precisam ser muito pequenos. Existem métodos que melhoram esse aspecto, mas eles não serão discutidos aqui.

Aplicando esse método, a integração da posição de um objeto pode ser feita em alguns passos simples. Primeiro se obtém a aceleração resultante das forças aplicadas na fatia de tempo considerada através da equação (3.4):

$$\mathbf{a} = \frac{\mathrm{F}}{m} \tag{3.11}$$

Depois usamos o resultado para integrar a velocidade com

$$\mathrm{v}(t+h) = \mathbf{v}(t) + h\mathbf{a}(t) \tag{3.12}$$

e a posição logo depois com

$$\mathrm{x}(t+h) = \mathbf{x}(t) + h\mathrm{v}(t) \tag{3.13}$$

A velocidade depende também da aceleração e portanto (3.13) deveria ter um termo envolvendo a aceleração. No entanto esse termo é tão pequeno para os valores de h normalmente utilizados em simulações que ele pode ser desprezado.

3.4 Colisões entre Corpos Rígidos

A modelagem adequada da colisão entre corpos rígidos é uma parte essencial para se obter o realismo desejado em uma simulação. Para modelar esse comportamento certas propriedades devem ser observadas no momento em que ocorre um contato entre os corpos. O ponto onde os corpos se tocam e a normal do contato são as propriedades do contato propriamente dito. Elas terão papel fundamental para determinar como os momentos lineares e angulares de um corpo rígido irão afetar o outro.

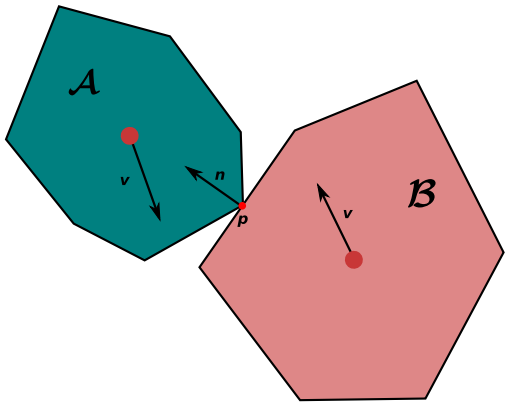

Figura 3.2: Uma colisão entre dois corpos rígidos em 2-D

O primeiro problema que deveria ser resolvido é o de encontrar o momento exato - ou uma aproximação razoável - em que ocorre um contato entre dois corpos, para assim determinar o ponto de contato entre eles. Como a simulação usa métodos numéricos com passos discretos, é natural que ocorra interpenetração entre as formas dos corpos. Entretanto, a descrição mostrada assume que o ponto de contato entre dois corpos já é conhecido.

Quando dois corpos A e B estão em contato é preciso determinar se eles estão realmente colidindo. É possível que os corpos estejam se separando ou ainda que estejam em contato, mas em repouso, como um livro sobre uma mesa. Para que estejam colidindo é preciso que os pontos de contato em cada objeto estejam indo de encontro um ao outro. Para determinar quais das situação acontece é preciso saber a velocidade do ponto de colisão em cada um dos objetos. Sendo p o ponto de colisão entre os dois objetos, o ponto p_A é o ponto p expressado nas coordenadas locais do objeto A

$$\mathrm{v}_{\mathbf{pA}} = \mathrm{p} - \mathrm{x}_{\mathbf{A}} \tag{3.14}$$

e sua velocidade é dada pela fórmula

$$\mathbf{v}_{\mathrm{pA}} = \Omega \times \mathbf{p_A} + \mathbf{v}_{\mathrm{A}} \tag{3.15}$$

A velocidade do ponto p_B é obtida da mesma maneira. Usando as duas velocidades é possível obter a velocidade relativa entre os dois pontos usando

$$\mathbf{v}_{\mathbf{p}_{\mathrm{A}}\mathbf{p}_{\mathrm{B}}} = \mathbf{v}_{\mathbf{p_A}} - \mathbf{v}_{\mathbf{p_B}} \tag{3.16}$$

A velocidade relativa é usada em conjunto com a normal $\mathbf{n}$ do contato para determinar se os corpos estão de fato colidindo. Para que se possa fazer essa determinação de forma não ambígua, algumas convenções são necessárias. Usando a velocidade relativa como foi definida acima e convencionando que a normal do contato sempre aponta do corpo B para o corpo A, os corpos estarão colidindo se a projeção de velocidade relativa sobre a normal for maior que zero, ou seja, se

$$\mathbf{v}_{\mathbf{p}_{\mathrm{A}}\mathbf{p_B}} \cdot \mathbf{n} > 0 \tag{3.17}$$

Corpos rígidos, por definição, não podem interpenetrar uns aos outros, mas caso a condição em (3.17) seja verdadeira, é o que está para acontecer. Se t_c é o instante de tempo exato em que A e B estão em contato com velocidades normais opostas, a menos

que as velocidades mudem, no instante $t_c + dt$ ocorrerá interpenetração. Para impedir que isso aconteça é preciso que uma força mude os momentos lineares de A e B de forma praticamente instantânea. Tal força deve ter uma magnitude muito grande relativamente a quaisquer outras forças que normalmente são aplicadas sobre o objeto para conseguir essa mudança em um período de tempo tão pequeno. Para modelar essa força existe o conceito de *impulso*.

Um impulso, na prática, é uma mudança direta nos momentos linear e angular de um objeto, sem passar pela integração de uma força. Forças precisam de tempo para ser integradas, mas essa mudança acontece em um momento infinitesimal entre t_c e $t_c + dt$. Como em uma simulação o tempo passa discretamente o que acontece é literalmente uma mudança direta na velocidade dos objetos envolvidos. Essa mudança repentina pode ser interpretada como uma força muito grande integrada sobre um período de tempo muito pequeno.

No modelo utilizado aqui, o impulso é um valor escalar j que multiplica a normal da colisão. Esse valor é então usado para calcular as novas velocidades linear e angular do objeto:

$$\mathbf{v'_A} = \mathrm{v_A} + \frac{j}{m_A}\mathrm{n} \tag{3.18}$$

$$\omega'_{\mathbf{A}} = \omega_{\mathbf{A}} + \frac{\mathbf{p_A} \times j\mathbf{n}}{I_A} \tag{3.19}$$

O impulso é dado por

$$j = -(1+e)\frac{\mathrm{v}_{\mathbf{p_A p_B}} \cdot \mathbf{n} + (\mathrm{p_A} \times \mathrm{n}) \cdot \boldsymbol{\omega}_{\mathbf{A}} - (\mathrm{p_B} \times \mathbf{n}) \cdot \omega_{\mathbf{B}}}{\frac{1}{m_A} + \frac{1}{m_B} + (\mathrm{p_A} \times \mathrm{n}) \cdot \frac{(\mathbf{p_A} \times \mathbf{n})}{I_A} + (\mathbf{p}_\mathrm{B} \times \mathrm{n}) \cdot \frac{(\mathbf{p_B} \times \mathbf{n})}{I_B}} \tag{3.20}$$

onde e é o coeficiente de restituição, cujo valor varia entre 0 e 1. O valor igual a 1 resulta em uma colisão totalmente elástica, enquanto o valor 0 resulta em uma colisão não elástica (*i.e.*, colisão sem restituição).

3.5 Discussão

Com os conceitos desenvolvidos aqui já se tem a base necessária para a construção de uma simulação de mecânica de corpos rígidos tradicional. O próximo capítulo explica mudanças que devem ser feitas para usar álgebra geométrica dentro de uma simulação.

4 ÁLGEBRA GEOMÉTRICA PARA SIMULAÇÃO FÍSICA

Conforme pode ser observado no capítulo 3, na abordagem convencional vetores são usados na representação de diversas gradezas físicas. Dentre elas força, torque, velocidades linear e angular e posição. É importante notar que ocorre uma sobrecarga semântica ao se representar diversos conceitos claramente distintos usando um mesmo objeto matemático. Por exemplo, a velocidade linear e a velocidade angular possuem interpretações completamente diferentes. Enquanto o vetor representando a velocidade linear indica uma direção a magnitude do vetor codifica a rapidez com que um corpo se move nessa direção, o vetor representando a velocidade angular indica um eixo de rotação e sua magnitude a variação do ângulo ao redor desse eixo num dado instante.

A representação de atitude de um corpo rígido no espaço 3-D é vista como complicada na literatura direcionada a construção de engines físicas para jogos. Em (MILLINGTON, 2007) algumas seções são dedicadas para uma revisão das técnicas mais intuitivas, porém inapropriadas. Por exemplo, o uso de ângulos de Euler na codificação da atitude para simulações físicas é intratável, por ser suscetível a *guimbal lock*. Matrizes de rotação são instáveis numericamente durante uma simulação e tendem a deixar de representar puramente rotações (deixam de ser ortogonais) (BARAFF, 1994). Por fim, a solução mais popular costuma ser a utilização de quatérnions. Entretanto, enquanto quatérnions fornecem uma descrição apropriada de rotações com diversas propriedades desejáveis, os quatérnions são descritos em um formalismo matemático diferente do restante dos elementos e transformações representadas por algebra linear. Para amarrar os dois formalismos diferentes, é preciso fazer conversões entre representações. Além disso, quatérnions são particularmente assustadores para não-matemáticos sendo caracterizados como "generalizações 4-D de números complexos". Tudo isso acaba os tornando entidades altamente abstratas que muitas vezes acabam sendo usadas como uma solução caixa-preta para resolver um problema sem um real entendimento do que está acontecendo.

A detecção de colisões (não a física das colisões vista na seção 3.4) é um problema puramente geométrico. Num dado instante de tempo é necessário determinar pontos de intersecção entre as formas geométricas dos corpos rígidos, assim como a normal da colisão. Existem métodos para determinar essas informações entre malhas de triângulo convexas. Entretanto é comum que engines físicas forneçam formas geométricas pré-definidas (*e.g.*, esferas, planos, cilindros, caixas) por questão de performance. É possível realizar testes mais rápidos entre essas formas do que com malhas de triângulo aproximando a forma correspondente. Infelizmente, suportar tais formas significa um aumento no código para gerenciar casos especiais.

Nas seções que seguem é descrito como o formalismo da álgebra geométrica pode ser utilizado na solução dos problemas enfrentados na simulação da mecânica de corpos rígidos por meios convencionais baseados em álgebra linear. As formulações são baseadas

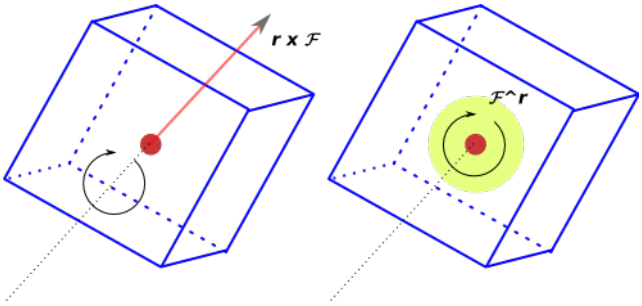

Figura 4.1: Torque como um vetor (esquerda) e como um bivetor (direita).

nas observações feitas por Hestenes em seu livro *New Foundations for Classic Mechanics* (HESTENES, 1999)

4.1 Dinâmica de Corpos Rígidos

A álgebra geométrica fornece uma linguagem matemática mais rica e consistente para a formulação da dinâmica de corpos rígidos. Enquanto grandezas físicas como a posição do centro de massa, forças, velocidade e aceleração continuam usando vetores para sua representação, mudanças na notação aparecem na hora de descrever as variáveis envolvidas no movimento angular.

Na mecânica clássica tradicional o torque em um espaço 3-D é representado por um produto vetorial (*cross-product*) entre dois vetores como descrito na equação (3.8). O vetor resultante é interpretado como o eixo de rotação e sua magnitude é a intensidade do torque aplicado. A interpretação da direção da rotação resultante depende de estabelecer uma convenção de acordo com o sentido em que o vetor aponta em relação à base do espaço de representação (*i.e.*, a "mão" do sistema de coordenadas). Na formulação com álgebra geométrica o torque é representado como um 2-blade:

$$\gamma_{\langle 2 \rangle} = \mathrm{f} \wedge \mathrm{r} \tag{4.1}$$

A interpretação geométrica dessa estrutura expõe mais claramente os conceitos que o torque representa. A visualização de disco usada na figura 2.2 é particularmente interessante, pois a orientação indica claramente a rotação induzida pela velocidade angular sobre o plano definido pelo 2-blade, que nesse contexto também pode ser chamado de *bivetor*. Note que o 2-blade é a representação dual da representação vetorial e daí o fato de ele não codificar o eixo de rotação em que o torque atua, mas sim um plano de rotação. A figura 4.1 ilustra melhor a relação entre a representação tradicional do torque como vetor e a representação como bivetor.

É possível demonstrar que os quatérnions unitários são equivalentes aos rotores (Seção 2.5) no espaço 3-D (DORST; FONTIJNE; MANN, 2007b). Por isso nada é mais natural para a escolha da representação da atitude de um objeto do que um rotor. Apesar de matemática e representacionalmente serem equivalentes, rotores estão completamente integrados a álgebra geométrica e não é necessário realizar conversões entre formalismos.

Para representar a atitude de um corpo rígido utiliza-se um rotor $\boldsymbol{R}$. Esse rotor codifica a atitude como a rotação em relação a uma base de referência $\{\mathbf{e_k}\}$ com $k = 3$ vetores ortogonais. Logo

$$\mathrm{e_x}' = \boldsymbol{R}\mathbf{e}_\mathrm{k} \,/\, \boldsymbol{R} \tag{4.2}$$

corresponde a atitude do corpo rígido no instante de tempo atual.

Seguindo a representação do torque é natural usar 2-blades para representar a velocidade e a aceleração angulares, agora denotados por $\Omega_{\langle 2\rangle}$ e $\omega_{\langle 2\rangle}$ respectivamente. O momento angular com álgebra geométrica é definido pela seguinte equação

$$\mathrm{L}_{\langle 2\rangle} = \left(F(\mathbf{\Omega}^{*}_{\langle 2\rangle})\right)^{-*} \tag{4.3}$$

na qual F é o tensor de inércia, que na formulação com álgebra geométrica é uma função linear sobre vetores que é aplicada sobre o dual do bivetor da velocidade angular.

A integração de $\Omega_{\langle 2\rangle}$ pelo método de Euler ocorre da mesma forma que com os vetores das componentes lineares

$$\mathbf{\Omega}'_{\langle 2\rangle} = \Omega_{\langle 2\rangle} + h\,\boldsymbol{\omega}_{\langle 2\rangle} \tag{4.4}$$

Ao final da integração temos um bivetor com a velocidade angular atual do corpo rígido. Para integrar o rotor de orientação $\boldsymbol{R}$ obtemos um rotor $\boldsymbol{\delta R}$, correspondente a variação na orientação durante o tempo h. Esse rotor é obtido através da seguinte equação:

$$\boldsymbol{\delta R} = exp(h\,\boldsymbol{\omega}_{\langle 2\rangle}) \tag{4.5}$$

na qual exp é a operação de exponenciação definida em (2.17).

Fazer variar o rotor $\boldsymbol{R}$ por $\boldsymbol{\delta R}$ é só questão de acumular as rotações representadas pelos dois rotores através do produto geométrico:

$$R' = \boldsymbol{\delta} R\, R \tag{4.6}$$

Com isso se tem o necessário para implementar uma engine de física sem colisões.

A resposta a colisões é obtida através da adaptação da equação (3.20) que descreve o impulso para ser compatível com a formulação de álgebra geométrica.

$$j = -(1+e)\frac{\mathrm{v}_{\mathrm{p}_{\mathbf{AB}}} * \mathbf{n} + (\mathrm{n} \wedge \mathrm{p}_{\mathbf{A}}) * \omega_{\mathbf{A}\langle 2\rangle} - (\mathbf{n} \wedge \mathbf{p}_{\mathrm{B}}) * \boldsymbol{\omega}_{\mathbf{B}\langle 2\rangle}}{\frac{1}{m_A} + \frac{1}{m_B} + \frac{1}{I_A} + \frac{1}{I_b}(\mathrm{p}_{\mathbf{A}} \wedge \mathrm{n}) * (\mathbf{n} \wedge \mathbf{p}_{\mathrm{A}}) + (\mathbf{n} \wedge \mathrm{p}_{\mathbf{B}}) * (\mathbf{p}_{\mathrm{B}} \wedge \mathrm{n})} \tag{4.7}$$

4.2 Detecção de Colisões

O modelo conforme da álgebra geométrica descrito na seção 2.7 é especialmente útil para a detecção de colisões. Sua capacidade de representar diversos elementos geométricos como primitivas unido a universalidade do produto *meet* (Seção 2.3) usado para calcular intersecções, faz desse modelo uma ótima ferramenta matemática no cálculo de colisões entre objetos. Em um caso especial onde o *join* de dois subespaços resulta no espaço como um todo, o *meet* pode se definido pela equação

$$\mathrm{A} \cap \mathbf{B} = (\mathrm{A}^{*} \wedge \mathbf{B}^{*})^{-*} \tag{4.8}$$

Como esse é o caso no uso feito aqui essa será a definição utilizada.

A intersecção de *flats* e *rounds* produz resultados intuitivos e imediatamente usáveis para tratar com colisões entre objetos. Para o desenvolvimento desse trabalho serão analisados os resultados do *meet* para os pares esfera - esfera, plano - esfera, ponto *flat* - esfera (Seção 4.2.1). Além disso, uma técnica baseada em Álgebra Geométrica Conforme Orientada (CAMERON; LASENBY, 2005) será descrita para o tratamento de colisão entre objetos convexos (Seção 4.2.2).

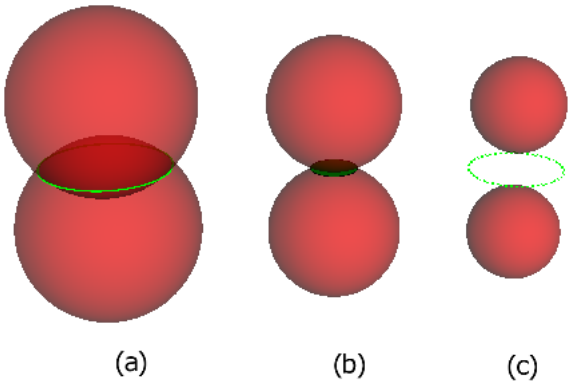

Figura 4.2: Os três casos de intersecção de esferas relevantes: (a) um círculo, (b) um espaço tangente e (c) um círculo imaginário.

4.2.1 Intersecções Esfera - Esfera e Esfera - *Flat*

Para a aplicação do `meet` na detecção de colisão é importante observar os resultados de intersecções entre objetos da álgebra geométrica conforme. Considerando a intersecção entre duas esferas Σ_1 e Σ_2

$$C = \Sigma_1 \cap \Sigma_2 \tag{4.9}$$

o resultado C da intersecção pode ser um círculo, um único ponto ou pode não haver intersecção. Por fins práticos, o caso em que a interseção resulta em uma esfera (i.e., $\Sigma_1 = \Sigma_2$) não precisa ser tratado. Na álgebra geométrica conforme os três casos relevantes são representados nos resultados como um círculo, um espaço tangente, ou um círculo imaginário respectivamente. A figura 4.2 ilustra esses casos. Um teste para verificar se há intersecção entre Σ_1 e Σ_2 pode ser feito observando o valor de $t = C^2$, com os possíveis resultados:

- Se $t < 0$, então as esferas não estão se tocando;
- se $t = 0$, as esferas se tocam em exatamente um ponto; e
- se $t > 0$, existe interpenetração entre as esferas.

Uma vez determinado que há intersecção, o ponto onde o ocorre a intersecção pode ser encontrado usando a seguinte equação:

$$p = C \infty C \tag{4.10}$$

A interpretação geométrica dessa equação, no caso do resultado C ser um círculo, se dá imaginando o ponto no infinito sendo refletido pelo círculo. Como o ponto no infinito está a uma distância infinita do círculo, independente de onde o círculo está localizado, essa reflexão sempre acabará no centro de qualquer círculo. A figura 4.3 ilustra essa intuição em duas dimensões. Para os outros tipos de resultado a intuição é a mesma.

A normal do contato pode ser obtida com a equação

$$\mathrm{n} = (-\infty \rfloor C) \wedge \infty^* \tag{4.11}$$

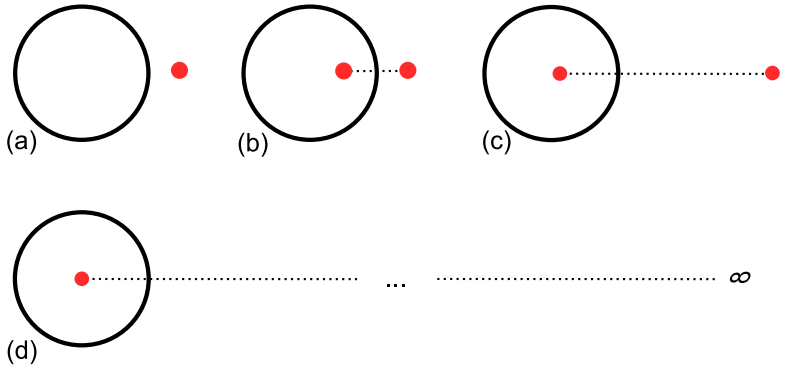

Figura 4.3: Em (a) um ponto e um círculo. Em (b) o ponto foi refletido pelo círculo, resultando em um ponto dentro do círculo. Conforme a distância aumenta o ponto refletido se aproxima ao centro do círculo (c). No caso limite (o ponto no infinito) o ponto refletido corresponderá ao centro do círculo (d).

A princípio esse método parece apenas uma maneira de complicar um problema simples. Calcular intersecções entre esferas é muito simples usando a distância entre seus centros e tamanhos dos raios. Porém o teste acima pode ser usado sem nenhuma alteração trocando uma das esferas por um plano ou por um ponto *flat*. Isso resulta em mais generalidade e menos tratamento de casos especiais.

4.2.2 Intersecção de Objetos Convexos

Para a intersecção de objetos convexos foi estudado o artigo (CAMERON; LASENBY, 2005). A técnica descrita no artigo consiste em dar interpretação para o sinal do peso das primitivas geométricas do modelo conforme. No modelo conforme apresentado aqui os blades $\mathrm{a} \wedge \mathbf{a}$ e $\mathbf{b} \wedge \mathbf{a}$ representam a mesma entidade geométrica. Entretanto, algebricamente, $\mathbf{a} \wedge \mathbf{b} = -\mathrm{b} \wedge \mathbf{a}$. Os sinais diferentes podem ser interpretados como uma direção para as primitivas geométricas representadas.

O artigo descreve uma técnica para verificar a inclusão de pontos dentro de facetas triangulares. Essa técnica pode ser usada para realizar detecção de colisão para objetos convexos definidos por malhas de polígonos. Infelizmente, por falta de tempo, um algoritmo baseado nessa técnica não foi desenvolvido.

4.3 Discussão

As extensões providas pela álgebra geométrica apresentam maneiras de representar conceitos da física de maneira integrada e com uma forte intuição geométrica. 2-blades são usados na representação da velocidade angular, dando assim uma visualização geométrica diferenciada para esse conceito. Para a representação de orientação os rotores são uma excelente alternativa aos quatérnions, por terem a vantagem de ser elementos computacionais da álgebra da mesma forma que os k-blades, eliminando assim, a necessidade de conversões entre diferentes formalismos.

Outro ponto onde a álgebra geométrica se sobressai é na geração de pontos de contato entre corpos rígidos. Usando os elementos geométricos que podem ser representados por subespaços no modelo conforme e o produto *meet* é possível diminuir consideravelmente o número de casos especiais no cálculo de intersecção entre objetos.

5 IMPLEMENTAÇÃO DE UMA ENGINE FÍSICA

É uma prática comum iniciar o desenvolvimento de uma engine física implementando uma série de tipos representando conceitos matemáticos básicos necessários como vetores, matrizes e quatérnions. Alternativamente é possível usar bibliotecas que implementam cada um desses objetos. Hoje em dia é possível encontrar bibliotecas que reúnem todos esses conceitos, graças a frequência com que eles são usados em conjunto em aplicações gráficas. Entretanto, mesmo nessas bibliotecas, grande parte dos métodos é dedicada a funções de conversão entre diferentes formalismos.

O capítulo anterior demonstrou como a álgebra geométrica pode ser usada para descrever a matemática da mecânica de corpos rígidos de maneira mais compacta e consistente. A existência dessa formulação não é apenas um benefício para os físicos em seu estudo sobre tais fenômenos. Ela também beneficia desenvolvedores que podem utilizar uma biblioteca que implementa álgebra geométrica para por em prática a reformulação.

A álgebra geométrica reúne toda a matemática necessária para representar posições, transformações e orientações em um único formalismo e, portanto, em uma única biblioteca. Não há necessidade de conversões *ad-hoc*, tudo acontece naturalmente a partir das definições fundamentais da álgebra. Isso resulta em fórmulas mais compactas e menos dependências, demonstrando que a álgebra geométrica não é só mais elegante matematicamente, mas também benéfica do ponto de vista da engenharia de software, gerando código mais enxuto e portanto mais fácil de manter.

Esse capítulo descreve a implementação de uma engine física desenvolvida como prova de conceito de que a álgebra geométrica é benéfica tanto do ponto de vista teórico quanto do ponto de vista prático. A seção 5.1 apresenta e discute duas implementações de álgebra geométrica. A seção 5.2 descreve a arquitetura da biblioteca. A seção 5.3 explica como a parte interna da simulação de corpos rígidos foi implementada, enquanto que a seção 5.4 descreve a implementação da detecção e tratamento de colisão.

5.1 Implementações de Álgebra Geométrica

No contexto da ciência da computação, um framework matemático de nada serve sem uma implementação de software apropriada para construir uma aplicação. Não seria prático exigir que cada novo usuário de um formalismo construísse sua própria biblioteca de software. Felizmente já existem disponíveis implementações de álgebra geométrica que praticantes de computação visual podem usar na solução de seus problemas.

Entretanto, é importante levar em consideração o desempenho desejado de uma aplicação na hora de escolher uma biblioteca, pois não é trivial construir uma implementação eficiente de álgebra geométrica. Uma implementação ingênua pode facilmente se tornar inutilizável, exceto para os menores dos problemas. Esse impacto é devido, principal-

mente, ao crescimento exponencial no número de blades de base conforme o aumento da dimensionalidade do espaço de trabalho. Mesmo no caso 3-D, por exemplo, o espaço representacional do modelo conforme tem 5 dimensões e portanto $2^5 = 32$ coeficientes para cada multivetor. Essa explosão na quantidade de coeficientes exige considerações cuidadosas no gerenciamento da memória alocada para representar os multivetores e também na implementação das operações sobre eles. Também é preciso levar em consideração que normalmente poucos desses coeficientes estarão sendo efetivamente usados, já que apenas blades e versores são de interesse na álgebra geométrica.

Técnicas para a implementação eficiente da álgebra geométrica são descritas em (FONTIJNE, 2007). Elas se baseiam em computação generativa, ou seja, programas que geram outros programas. Essa geração pode ocorrer em diversas partes do fluxo de trabalho que ocorre para a geração de um programa executável a partir de um código fonte. O Gaigen 2, ou *Geometric Algebra Implementation Generator 2* usa como abordagem a geração de uma implementação de álgebra geométrica antes da compilação do programa propriamente dito. Para que a implementação gerada se torne eficiente é necessário primeiro gerar uma informação de profiling específica para a aplicação desejada e depois fazer a geração de uma nova implementação otimizada. A ferramenta e o processo de uso são descritos com mais detalhes na seção 5.1.1. A GATL, *Geometric Algebra Template Library*, é uma biblioteca em C++ usando uma abordagem baseada em meta-programação para gerar em tempo de compilação implementações eficientes das operações da álgebra geométrica. Ela foi desenvolvida pelo coorientador desse trabalho, Leandro A. F. Fernandes. A seção 5.1.2 apresenta uma descrição mais detalhada dessa outra biblioteca.

As implementações geradas pelo Gaigen 2 já foram comparadas com outras implementações de álgebra geométrica (FONTIJNE, 2006). Os resultados obtidos indicam o Gaigen 2 como sendo o mais eficiente. Por ser relativamente nova, a GATL nunca foi comparada com outras implementações de álgebra geométrica. Por isso, uma comparação entre ela e o Gaigen 2 é apresentada na seção 5.1.3.

5.1.1 Gaigen 2

O Gaigen 2 não é uma implementação de álgebra geométrica propriamente dita e sim um gerador de implementações de álgebra geométrica. A geração se dá a partir de uma especificação escrita em uma linguagem de domínio específico que descreve como operações na implementação da álgebra geométrica são mapeadas para a linguagem alvo. Esse arquivo serve de entrada para o gerador em si, que vai criar o código fonte necessário para implementar o modelo de álgebra geométrica desejado. A partir daí é possível usar os tipos, funções e operadores definidos no código gerado para construir uma aplicação.

Para gerar código eficiente, o Gaigen 2 depende de um ciclo de realimentação, onde o programa a ser otimizado deve rodar algumas vezes para gerar informação de *profiling*. Essa informação é utilizada para gerar uma nova implementação da álgebra com otimizações específicas para o programa em questão. Basicamente, o Gaigen 2 identifica os coeficientes dos multivetores que são sempre iguais a zero e os elimina de computações realizadas pela biblioteca final. Além disso, laços são desenrolados de maneira implícita e chamadas internas a pequenas funções são evitadas pela substituição *inline* das mesmas.

Uma das vantagens do Gaigen 2 é gerar implementações de álgebra geométrica para várias linguagens. Até o presente momento apenas geração para Java e C++ é suportada, porém suporte a novas linguagens pode ser adicionado. A otimização via profiling só é implementada para código gerado em C++.

	GATL	Gaigen 2 s/ profiling	Gaigen 2 c/ profiling
`plus / add`	1.14	13.96	13.76
`gp`	1.63	19.60	0.14
`lcont`	1.10	13.86	13.96
`op`	1.20	15.74	14.70
`scp`	1.08	2.62	2.14

Tabela 5.1: Resultado dos benchmarks por função. Cada função foi executada 10.000.000 vezes e medida com o comando time do linux. O tempo exibido corresponde ao "user time" que é medido em segundos.

5.1.2 GATL

A GATL é uma biblioteca de álgebra geométrica implementada através de técnicas de meta-programação usando *templates* em C++. A vantagem dessa abordagem é a possibilidade de otimização do código em tempo de compilação. A otimização ocorre de maneira implícita a partir da escolha dos tipos utilizados nas computações e dos produtos utilizados. Deste modo, a biblioteca sabe quais os coeficientes dos multivetores são garantidamente iguais a zero, eliminando-os do processamento. Além de deixar para o compilador a decisão de quais rotinas devem ser definidas inline e quais não devem. Note que esta é uma decisão que depende tanto da plataforma onde o programa é compilado quanto da computação que será realizada.

Na GATL não há a necessidade do passo extra de profiling e recompilação que o Gaigen 2 exige. Um dos pontos fracos da GATL é a possibilidade de ocorrer um aumento no tempo de compilação por conta na complexidade dos templates definidos. Mas, por outro lado, o fluxo de trabalho do desenvolvedor se mantém dentro do normal, sem a necessidade de recorrer a aplicações externas.

5.1.3 Benchmark Comparativo

Afim de comparar as duas ferramentas descritas nas seções 5.1.1 e 5.1.2, foi realizado um *benchmark*. O benchmark consiste em executar operações da álgebra geométrica repetidas vezes sobre multivetores e blades inicializados com coeficientes aleatórios. Cada operação então foi executada 10.000.000 vezes e teve seu tempo medido pelo comando `time` do Linux. O tempo considerado foi o `user time`.

A tabela 5.1 descreve os tempos resultantes da execução do benchmark. O resultado sugere que a otimização executada pelo profiling do Gaigen 2 pode ser seletiva em alguns aspectos. É possível que simplicidade do benchmark seja um dos fatores que ocasione uma dificuldade para o profiler obter dados suficientes para realizar uma otimização com melhores resultados.

Para a implementação da engine física foi escolhida a GATL não apenas por apresentar um desempenho mais consistente no benchmark, mas também por simplificar o fluxo de trabalho no desenvolvimento.

As ferramentas apresentadas nessa seção representam o estado da arte das implementações de álgebra geométrica. Com essas bibliotecas as computações geométricas intuitivas podem ser usadas diretamente na implementação de algoritmos e solução de problemas geométricos. Diversos exemplos de aplicação podem ser encontrados em (DORST; FONTIJNE; MANN, 2007a).

5.2 Arquitetura da Engine

A engine foi implementada em C++ usando o paradigma de Orientação a Objetos e a GATL (Seção 5.1.2). A partir do que foi descrito nos capítulos 3 e 4, é possível separar a simulação de corpos rígidos em dois problemas bastante distintos. Um deles é a simulação da dinâmica dos corpos e como eles reagem a aplicação de forças e impulsos, o outro é um problema puramente geométrico de achar intersecções entre as formas dos corpos rígidos para determinar os pontos e normais de seus contatos. Isso reflete na arquitetura da implementação através da existência de duas classes, cada uma com uma responsabilidade distinta dentro da simulação:

- A classe `World` tem como responsabilidade a parte dinâmica da simulação. Ela mantém propriedades globais do mundo virtual sendo simulado (*e.g.*, gravidade, *damping*) e uma lista de corpos rígidos presentes na simulação.

- A classe `Space` se ocupa do problema geométrico de determinar contatos entre objetos. Ela armazena as formas geométricas associadas aos corpos da simulação e sob demanda gera uma lista de contatos entre objetos assim como responde se existem interpenetrações entre as formas dos corpos simulados.

A classe `World` representa o mundo físico da simulação onde os corpos representados por `Body` existem como corpos rígidos. A classe `World` funciona como um contêiner para armazenar e gerenciar todas as instâncias de `Body` existentes na simulação e também como um local para definir aspectos globais da simulação.

Cada corpo rígido é representado por uma classe `Body`. Essa classe apenas guarda informações sobre a dinâmica do corpo simulado, como sua massa, momento de inércia, posição e velocidade. As variáveis cinemáticas do corpo são representadas pelos seguintes atributos:

- `position`: é um vetor representando a posição do corpo rígido no momento atual

- `velocity`: outro vetor representando a velocidade com que o corpo se move, ou seja, a taxa com que a posição do objeto está variando no momento.

- `orientation`: um rotor descrevendo a orientação do objeto. Assume-se que a base do espaço é o referencial a partir do qual o rotor descreve a rotação necessária para chegar na orientação atual.

- `angular_velocity`: um bivetor descrevendo a taxa com que o rotor de orientação está variando no momento atual.

As propriedades intrínsecas do objeto como a massa e o tensor de inércia são representadas por um valor escalar e por uma função linear respectivamente. Nessa implementação, é assumido que o centro de massa está localizado na origem do sistema de coordenadas local do objeto e, portanto, não é representado explicitamente por um atributo.

A descrição geométrica de um corpo é representada com uma instância da classe `Shape`. Essa classe mantém uma descrição de uma forma geométrica de maneira apropriada para o cálculo de colisões, assim como uma referência do corpo ao qual a forma está associada. A posição da forma no espaço é derivada diretamente do corpo rígido associado, então não é armazenada explicitamente na classe `Shape`.

A interação com os corpos rígidos da simulação se dá principalmente através do método `apply_force(vector_t force, vector_t offset)`. Esse método aplica uma força com uma intensidade igual ao vetor `force` e com um deslocamento do centro de massa do objeto igual ao vetor `offset`. A força aplicada é representada no sistema de coordenadas do mundo, enquanto o deslocamento é representado localmente em relação ao centro de massa. Como conveniência, o valor padrão de `offset` é o vetor nulo facilitando a aplicação de forças que afetam apenas o movimento linear do objeto.

O objeto também oferece uma interface para aplicar um torque diretamente através do método `apply_torque(bivector_t torque)`. Esse método é uma conveniência para simulações onde o resultado da ação de forças aplicadas resulta em apenas torque.

5.3 Simulação

Uma simulação modela fenômenos ocorrendo durante um período de tempo. O software que realiza a simulação deve prover uma função para avançar o tempo da simulação. Cada avanço de tempo simulado é um passo discreto e, por consequência, uma aproximação. O tamanho desse passo é definido de acordo com a necessidade da aplicação e corresponderá ao h utilizado na integração numérica.

As equações (3.11), (3.13) e (3.12) da seção 3.3 fornecem a informação necessária para implementar o movimento linear na simulação. O movimento angular é baseado nas equações (4.4) e (4.5). Com isso é possível criar uma função step para a simulação que terá uma forma semelhante ao seguinte pseudo-código:

```
function step(h) begin
  for each body do
    vector_t total_force = get_total_force(body)
    body.acceleration = total_force / body.mass
    body.velocity += body.acceleration * h
    body.position += body.velocity * h

    bivector_t total_torque = get_total_torque(body)
    body.angular_acceleration = total_torque / body.inertia_tensor
    body.angular_velocity += body.angular_acceleration * h

    body.attitude = exp(-2 * body.angular_velocity * h) *
                    body.attitude
  end

  space.handle_collisions()
end
```

A função `get_total_force` obtém a força total aplicada em um corpo no instante atual de tempo.

5.4 Detecção de Colisões

A detecção de colisões é uma das partes mais custosas da simulação. Durante essa fase da simulação é necessário encontrar o ponto onde a geometria dos objetos está em

contato, a normal da colisão e por fim usar essa informação para calcular os impulsos que devem ser aplicados para cada objeto.

A detecção de colisão se divide e ocorre em duas fases. A primeira fase é chamada de fase ampla e usa métodos mais simples para determinar se dois objetos podem estar colidindo num dado momento. O objetivo dessa fase é usar algoritmos rápidos para determinar pares de objetos "candidatos" a estarem colidindo. A segunda fase trata de examinar os pares de colisão detalhadamente para calcular pontos de contato entre suas formas geométricas. Essa divisão ocorre por motivos de eficiência.

Como a motivação da existência da fase ampla é puramente de otimização, foi escolhido omitir da engine a implementação de um algoritmo para essa fase. O principal motivo dessa escolha é que no tempo dado para a pesquisa não foi encontrado algum benefício aparente no uso da álgebra geométrica na implementação de tal algoritmo.

A álgebra geométrica não tem aplicações aparentes na fase ampla da detecção de colisão e implementar um bom algoritmo demanda tempo. Por causa disso optou-se por manter essa parte o mais simples possível. A única consequência dessa escolha é impor uma limitação prática no número de corpos que podem ser simulados simultaneamente.

O funcionamento da detecção de colisão pode ser resumida pelo seguinte pseudocódigo:

```
function handle_collisions() begin
  pairs = collision_pairs()

  contacts = new list
  for each pair in pairs do
    contact = generate_contact(pair)
    if contact is not null do
      contacts.append(contact)
    end
  end

  for each contact in contacts do
    do_collision(contact)
  end
end
```

O método `collision_pairs` gera pares resultantes da detecção de colisão ampla. No caso, como não há detecção, os pares são todas os pares não ordenados de `shapes` na classe `space`. Depois contatos são gerados para cada par de colisão e inseridos na lista de contatos `contacts`.

Depois para cada contato existente na lista a colisão é resolvida por uma chamada a `do_collision` que é responsável por aplicar os impulsos resultantes. A aplicação de impulsos é uma implementação direta da equação

6 CONSIDERAÇÕES FINAIS

A álgebra geométrica é uma alternativa muito atraente na resolução dos problemas encontrados em computação visual. Seu uso não é tão difundido quanto o da álgebra linear, mas isso eventualmente pode mudar. Esse trabalho tenta realçar as vantagens do uso da álgebra geométrica através de um exemplo prático já bastante difundido. A simplicidade alcançada em algumas formulações diminuem o custo de manutenção do programa, tornando a leitura do código mais simples, claro, para os que estão familiares com a linguagem matemática. Com a divulgação do formalismo é possível esperar que mais cientistas e programadores da área de computação visual compreendam as formulações, assim como acontece com as formulações baseadas em álgebra linear.

Para produzir a implementação apresentada ao longo desse trabalho, uma série de simplificações foi feita. O modelo mais simples de resposta a colisão foi utilizado, a geração de pares de colisão foi feita da forma mais simples, entre outras. Enquanto o resultado obtido não é apropriado para uso em produção, serve como um ponto de partida para quem deseja explorar mais a fundo os benefícios da álgebra geométrica na implementação de engines físicas.

Parte das simplificações feitas decorreram do fato de ser um assunto completamente novo para o autor, o que apresentou um grande desafio de aprendizado.

6.1 Melhorias e Trabalhos Futuros

Como foi necessário o estudo do formalismo juntamente com o desenvolvimento de uma aplicação prática, os objetivos de implementação da engine foram bastante modestos. Engines físicas possuem outros recursos que não foram cobertos nessa implementação. O modelo de colisão usado, por exemplo, não usa fricção. Para a fase ampla da detecção de colisão existem técnicas avançadas, mas não é claro se a álgebra geométrica iria gerar algum ganho na implementação.

A técnica de implementação descrita aqui se baseia em dois modelos da álgebra geométrica. O modelo vetorial euclidiano é usado na descrição da mecânica dos objetos enquanto a detecção de colisões usa o modelo conforme. Uma integração total entre as duas partes poderia ser feita investigando como usar um único modelo para todas as partes simulação. Um possível caminho seria usar a forma de descrever uma transformação de corpo rígido (uma translação e uma rotação) na forma de um único versor no modelo conforme que é apresentada em (DORST; FONTIJNE; MANN, 2007a).

Uma outra vantagem da álgebra geométrica é a facilidade com que se pode fazer interpolação de versores. Detalhes sobre isso podem ser vistos também em (DORST; FONTIJNE; MANN, 2007a).

A engine apresentada aqui foi apenas testada com um número relativamente pequeno

de objetos. Uma avaliação da escalabilidade da biblioteca pode ser feita, assim como comparações de desempenho com engines tradicionais baseadas em álgebra linear.

REFERÊNCIAS

BARAFF, D. **An Introduction to Physically Based Modelling**: rigid body simulation 1 - unconstrained rigid body dynamics. 1994.

CAMERON, J.; LASENBY, J. Oriented conformal geometric algebra. In: ICCA7, 2005. **Proceedings...** [S.l.: s.n.], 2005.

DORST, L.; FONTIJNE, D.; MANN, S. **Geometric Algebra for Computer Science - An Object Oriented Approach to Geometry**. [S.l.]: Morgan Kauffmann, 2007.

DORST, L.; FONTIJNE, D.; MANN, S. **Geometric Algebra for Computer Science - An Object Oriented Approach to Geometry**. [S.l.]: Morgan Kauffmann, 2007. p.181, 182.

FERNANDES, L. A. F.; OLIVEIRA, M. M. **Geometric Algebra**: a powerful tool for solving geometric problems in visual computing. 2007.

FONTIJNE, D. **Gaigen 2**: a geometric algebra implementation generator. [S.l.]: University of Amsterdam, 2006.

FONTIJNE, D. **Efficient Implementation of Geometric Algebra**. 2007. Tese (Doutorado em Ciência da Computação) — University of Amsterdam.

GOLDSTEIN, H.; POOLE, C.; SAFKO, J. **Classical Mechanics**. Terceira.ed. [S.l.]: Addison Wesley, 2000.

HESTENES, D. **New Foundations for Classical Mechanics**. Segunda.ed. [S.l.]: Kluwer Academic Publishers, 1999.

MILLINGTON, I. **Game Physics Engine Development**. [S.l.]: Morgan Kaufmann, 2007.

PALMER, G. **Physics for Game Programmers**. [S.l.]: Apress, 2005.