UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Animação e Tratamento de Colisões de Corpos Rígidos Utilizando Análise Dinâmica

por

Robson R. Lemos

Dissertação submetida como requisito parcial
para a obtenção do grau de
Mestre em Ciência da Computação

Prof. Flávio Wagner
Orientador

Prof. Carla M. D. S. Freitas
Co-orientador

Porto Alegre, Junho de 1993.

CIP - CATALOGAÇÃO NA PUBLICAÇÃO

Lemos, Robson R.

Animação e Tratamento de Colisões de Corpos Rígidos Utilizando Análise Dinâmica / Robson R. Lemos.—Porto Alegre: CPGCC da UFRGS, 1993.

161 p.: il.

Dissertação (mestrado)—Universidade Federal do Rio Grande do Sul, Curso de Pós-Graduação em Ciência da Computação, Porto Alegre, 1993. Orientador: Wagner, Flávio; Co-orientador: Freitas, Carla M. D. S.

Dissertação: Computação Gráfica
Animação, Simulação, Dinâmica de Corpos Rígidos, Detecção de Colisões e Resposta

AGRADECIMENTOS

A todos que procuraram entender minha forma de expressão no desenvolvimento deste trabalho, cuja compreensão e confiança depositada permitiram a chegada ao fim desta maratona.

Aos meus orientadores, Carla Freitas e Flávio Wagner pelo apoio durante esta longa caminhada.

À energia, firmeza e participação ativa neste trabalho, por muitas vezes fundamental, do colega de mestrado e amigo Rui Bastos pelas inúmeras discussões e sugestões a respeito da física abordada, bem como, pela vibração recíproca nos momentos de superação dos diversos obstáculos (de coração mano, valeu !!!).

Ao prof. Laschuk pelas excelentes aulas ministradas e pelas viagens proporcionadas no mundo da computação gráfica.

À profa. Beatriz Gay pelos esclarecimentos sobre dinâmica.

Às pessoas que me dizem quase sempre não, poucas vezes sim e que mais amo nessa vida, meus pais, Lemos e Zenir, e meus irmãos Jeferson e Rosana.

Àqueles que ficaram na torcida e que certamente preencheram em muito minha vida aqui em Porto Alegre. Aos *brothers and sisters*, Alexandre Ribeiro (Rib), Helena (Helen), Marcelo Walter (MarceloW), Flávio de Oliveira (Flavian), Álvaro Moreira (Alvinho), Francisco de Assis (AC/DC), Yara Lemr (Ieda), Roberto Amboni (Foca, Alf), Liliana P. (Liliana Peron), Susana (Susaninha), Gladimir B. (Gladistone), Heitor S. (Heitororo), Paulo Roisenberg (Paulinho do Tombo), Luigi C. (Rolastones) e Ricardo D. (Cadinho).

Aos demais colegas, funcionários e professores do CPGCC.

SUMÁRIO

LISTA DE ABREVIATURAS

MKS	Metro K(quilograma) Segundos
CAD/CAM	Computer Aided Design/Computer Aided Manufacturing

LISTA DE FIGURAS

RESUMO

Os métodos de controle de movimento em animação baseados em Física, e utilizados em Computação Gráfica, têm como objetivo simular o comportamento de objetos de acordo com as leis físicas que governam o mundo virtual adotado.

Este trabalho utiliza a dinâmica de corpos rígidos como método de controle de movimento em animação por computador aplicada a movimentos e colisões de corpos rígidos não-articulados. O trabalho também apresenta uma metodologia para projeto e implementação de simulações gráficas com o objetivo de estabelecer relações entre modos de interação e os mecanismos de abstração necessários em ambientes de simulação.

A principal vantagem da utilização da Mecânica newtoniana está no fato de que ela garante o realismo dos movimentos e colisões. Associados a cada objeto devem estar os seguintes atributos físicos: centro de massa, massa total, momento de inércia e, eventualmente, a elasticidade do material. A partir de um estado inicial (velocidade linear, posição, velocidade angular e orientação) e de estímulos iniciais sobre os objetos (forças e torques), o sistema determina a evolução do estado dinâmico inicial ao longo de um dado intervalo de tempo. Para produzir o movimento dos corpos, são resolvidos sistemas de equações diferenciais de primeira ordem utilizando métodos numéricos.

O tratamento de colisões de corpos rígidos envolve a detecção da colisão e contato entre objetos e a determinação das forças de contato entre os mesmos. A estratégia utilizada para a colisão considera que num determinado instante de tempo existe apenas um ponto de contato entre dois objetos. As superfícies dos objetos são representadas por uma grade de pontos conectados para formar polígonos. Existem dois tipos de estratégias para se detectar o ponto de contato entre dois objetos: o ponto de contato resultante da intersecção do vértice das arestas de um objeto com a face poligonal de outro objeto e o resultante da intersecção da aresta de um objeto

com a face poligonal de um outro objeto. A análise de impacto, para resolver a dinâmica, utiliza um método analítico que preserva os momentos linear e angular durante a colisão e resulta em novas velocidades linear e angular para cada corpo rígido. Este tratamento de colisões permite ao sistema de animação realizar, em tempo de simulação, um controle automático da restrição de que dois corpos rígidos, ao colidirem, não podem se interpenetrar. Tal tratamento automático, em geral, não é realizado pelos sistemas de animação por computador atualmente existentes.

O trabalho apresenta o protótipo desenvolvido para validar as soluções dadas aos problemas de determinação do movimento e detecção de colisões, assim como sua aplicação na produção de suas seqüências animadas. São comentadas, também, as extensões do presente trabalho, decorrentes da abordagem dada ao problema da simulação do comportamento fundamental de corpos rígidos num dado mundo virtual a qual permite a incorporação de outras características aos objetos: elasticidade, para modelagem de deformações, e articulações, para produção de movimentos articulados com diferentes graus de liberdade.

PALAVRAS-CHAVE: Animação, Simulação, Dinâmica de Corpos Rígidos, Detecção de Colisões e Resposta.

TITLE: 'Animation and Treatment of Collisions of Rigid Bodies using Dynamic Analysis'

ABSTRACT

The goal of the motion control methods used in Computer Graphics for physically based animation is to simulate the behavior of objects according to physical laws that govern a certain virtual world.

This work uses rigid body dynamics as a motion control method for animation applied to motions and collisions of non-articulated rigid bodies. In addition, the work presents a methodology for the design and implementation of graphical simulation systems with the aim of providing relationships among interaction modes and abstraction mechanisms for a variety of applications.

The principal advantage in using Newtonian Mechanics is that it keeps the realism of motions and collisions. Physical attributes must be associated with objects: center of mass, mass, moment of inertia, and sometimes, elasticity of the materials. Given an initial state (linear velocity, position, angular velocity, and orientation) and initial stimuli applied to the objects (forces and torques), the system determines the evolution of the dynamic state along a determinate time interval. The motion description is obtained using numerical solutions of sets of first order differential equations.

The treatment of collisions of rigid bodies involves detecting collision and contact between objects and determining the contact forces present between contacting objects. The strategy used to treat collisions takes into account that there is just one contact point between two objects. The surfaces of objects are represented by a grid of connecting points forming polygons. There are two kinds of strategies to detect the contact point between two objects: the contact point resulting of intersecting the vertices of the edges of an object with the polygonal

face of another one and that resulting of intersecting the edges of an object with the polygonal face of another one. The analysis of impact, to resolve the dynamic, uses an analytical method that preserves the linear and angular moments during the collision, finding a new linear and angular velocity for each rigid body. This treatment of collision allows the animation system to provide, at simulation time, an automatic control of the restriction that there is no interpenetration between two rigid bodies when they colide. This automatic treatment in general is not provided by existing computer animation systems.

The work presents the prototype developed for validating the solutions given to the problems of motion control and collisions treatment, as well as its application in the production of animated sequences. The text ends with comments on extension of the present work from the approach given to the problem of simulating the behavior of objects in a certain virtual world allowing the incorporation of other characteristics to the objects: elasticity, to model deformations, and articulations, for the production of articulated movements with different degrees of freedom.

KEY-WORDS: Animation, Simulation, Dynamic of Rigid Bodies, Collision Detection and Response.

1 INTRODUÇÃO

1.1 Conceito de Animação

Tradicionalmente, a produção de uma seqüência de animação de um objeto, em computação gráfica, é realizada através da especificação explícita de posições e orientações desse objeto no espaço e no tempo. Tais posições e orientações são interpoladas produzindo uma seqüência que é, na realidade, uma alternativa para o movimento do objeto. Diferentes leis de interpolação produzem diferentes movimentos para os mesmos parâmetros inicial e final.

Tal forma de descrição de movimento, entretanto, é restrita na produção de movimentos realísticos, principalmente, quando complexos. O animador é forçado a utilizar a intuição e testar diferentes combinações de parâmetros e leis de interpolação até obter a seqüência que melhor se aproxima da realidade.

Nos últimos anos, a descrição explícita de movimentos complexos passou a ser substituída por outros métodos de controle de movimentos [FOL 90]. Os métodos de controle de movimento em animação por computador que fazem uso de análise dinâmica têm apresentado resultados interessantes. Trabalhos realizados dentro deste contexto permitem a produção de movimentos realísticos de objetos e redução da complexidade na descrição de seu movimento . Neste trabalho, investigam-se as características necessárias para modelagem com simulação física de um sistema dinâmico de corpos rígidos sob ação de forças e torques externos e, associadas a estas características, o tratamento de possíveis interações entre os corpos rígidos e dos mesmos com o ambiente.

1.2 Animação Baseada em Física

Na animação baseada em física, o movimento dos objetos é regido pelas leis físicas que governam seu comportamento em um dado ambiente, ou seja, num mundo virtual. Além de atributos geométricos para modelagem dos objetos, são utilizados, também, atributos físicos. O movimento é obtido através da resolução das equações dinâmicas de movimento associadas ao modelo dos objetos. A mecânica clássica [MAR 70] apresenta várias formulações para construção de equações dinâmicas. A resolução dessas equações pode ser realizada através da utilização de métodos numéricos [PRE 88]. Portanto, o comportamento dos objetos não é criado manualmente pelo animador, mas gerado, de uma forma intrínseca, pelo modelo de animação.

Uma analogia entre pintura e fotografia é feita por Blinn [TER 89] para esboçar as diferenças entre animação tradicional ('keyframing') e animação baseada em física: 'Um pintor deve ter bastante habilidade para representar alguma coisa realisticamente. Já um fotógrafo necessita apenas enquadrar alguma coisa em seu visor e, *CLIC*, a imagem realística é obtida'. Da mesma forma, o usuário de um sistema de animação tradicional deve conhecer física tão bem quanto um pintor deve conhecer luzes e reflexão para simular manualmente um movimento realístico. Entretanto, o usuário de um sistema de animação baseado em física obtém o movimento resultante dos objetos de acordo com as leis físicas adotadas de uma maneira automática.

Dentro desse contexto, trabalhos realizados em animação baseada em física têm investigado mecanismos para simulação de aspectos particulares do comportamento de objetos, tais como: corpos não-rígidos [PLA 88b, TER 88c], corpos rígidos articulados e não-articulados [BAR 88b, HAH 88, WIL 87b], detecção de colisões e resposta [MOO 88] e modelagem de comportamentos complexos [BRU 89]. Estes trabalhos podem ser distingüidos em duas classes principais, conforme seu objetivo [PUE 88]:

- **Produção de filmes:** onde a transmissão de uma determinada mensagem através do filme é fundamental (por exemplo, filmes para entretenimento, filmes didáticos ou propostas publicitárias).
- **Simulação:** onde os resultados numéricos gerados pela simulação são fundamentais (por exemplo, Robótica, CAD/CAM, detecção de colisões ou verificação de caminhos).

1.3 Motivação e Objetivos

A animação baseada em física é uma área recente e pouco explorada, principalmente, no Brasil. Trabalhos já realizados no CPGCC da UFRGS, abordando animação [OLA 87, OLI 92, PIN 88, SCH 92, SIL 92], utilizaram técnicas de especificação de movimento baseadas em modelos cinemáticos, mais especificamente, modelos com interpolação. Assim, como evolução do tema, faz-se mister abordar as técnicas com modelo dinâmico.

Adicionalmente a esse incentivo, a possibilidade de poder criar mundos virtuais (ambientes de simulação) através de abstrações matemáticas de objetos, modelados fisicamente, cuja representação é expressa em termos de imagens, pode auxiliar em muito a modelagem científica no sentido de validar, através da simulação, situações provadas apenas em teoria.

O objetivo principal deste trabalho é o estudo de métodos automáticos de controle de movimento, em computação gráfica, aplicados à animação baseada em física, com vistas a reduzir a complexidade da descrição do movimento e de aperfeiçoar o realismo da animação. Para tanto, parte-se de um estudo da dinâmica do movimento e tratamento de colisões de corpos rígidos não-articulados, aos quais se restringe o trabalho sem perda da generalidade. O estudo para animação de corpos rígidos não-articulados procura associar propriedades físicas utilizadas na mecânica clássica a propriedades geométricas utilizadas em computação gráfica.

Faz parte do estudo, também, a implementação de um protótipo para validação dos métodos automáticos de controle de movimento propostos. Pretende-se, com isto, introduzir conceitos de animação baseada em física como mais uma ferramenta à disposição dos potenciais usuários na instituição.

1.4 Organização da Dissertação

No capítulo 2, apresentam-se os conceitos gerais, em alto nível, para o projeto e implementação de simulações gráficas dinâmicas que são utilizados em diferentes propostas de sistemas de animação.

O capítulo 3 detalha as etapas necessárias para modelagem com simulação física de corpos rígidos assim como as características que devem ser observadas durante estas etapas. São detalhados, também, os principais métodos automáticos de controle de movimentos e apresenta-se uma análise de trabalhos realizados em métodos de controle que operam em baixo nível. A partir disto, é apresentado o modelo proposto para simulação dinâmica de corpos rígidos a ser adotado neste trabalho.

No capítulo 4, analisa-se a dinâmica do movimento translacional e rotacional aplicada à animação de corpos rígidos e apresentam-se estratégias para a implementação de um núcleo a ser acoplado ao protótipo do modelo proposto neste trabalho.

O capítulo 5 analisa a detecção de colisões quando das interações entre corpos rígidos e a resposta da dinâmica do impacto resultante dessas interações. Estratégias para implementação de um núcleo a ser acoplado ao protótipo são apresentadas.

No capítulo 6, descreve-se o protótipo implementado para geração de posições e ângulos de orientação de corpos rígidos ao longo do tempo, a partir de

estímulos iniciais. Realiza-se, também, a análise de resultados a partir de exemplos e seqüências animadas obtidas com o programa motor **CLIC**.

As conclusões e trabalhos futuros são apresentados no capítulo 7 e no anexo encontra-se a listagem do programa motor **CLIC**.

2 SIMULAÇÕES GRÁFICAS DINÂMICAS

Diversos experimentos científicos podem ser simulados através de abstrações matemáticas dos objetos em um dado ambiente, onde sua representação é expressa em termos de imagens. Com isso, é possível criar o que Zeltzer [ZEL 91] denomina de mundo virtual. As imagens deste mundo virtual correspondem aos resultados visuais da simulação de objetos de acordo com equações dinâmicas de movimento durante um determinado intervalo de tempo.

A abordagem dada às simulações gráficas dinâmicas corresponde à representação e exibição de mundos virtuais tridimensionais.

Neste trabalho, simulação dinâmica é tratada como uma técnica particular de animação, onde o controle de movimento é determinado por equações dinâmicas de movimento.

2.1 Metodologia para Projeto e Implementação de Simulações Gráficas Dinâmicas

A principal vantagem na utilização de mundos virtuais para experimentos científicos está na capacidade de permitir diferentes níveis de abstração durante a modelagem (vide seção 3.1). Assim sendo, é possível estabelecer um entendimento no sentido de representar e controlar mundos virtuais para propostas de simulações gráficas dinâmicas através da utilização de mecanismos de abstração. Trabalhos realizados na busca deste entendimento, como Barr [BAR 91b] e Zeltzer [ZEL 91], apresentam propostas de uma metodologia para construção desses mecanismos.

2.1.1 Abstração × Níveis de Representação

Barr [BAR 91b] apresenta uma abordagem para o desenvolvimento de modelos realísticos a partir da criação de abstrações e representações matemáticas de objetos modelados fisicamente e dependentes do tempo.

Basicamente, as abstrações dos objetos consistem de metas de comportamento e descrições físicas. Para permitir diferentes níveis de abstração, nessa abordagem, propriedades de restrições geométricas, propriedades mecânicas, o controle dos objetos e os parâmetros que os representam devem estar dispostos em uma mesma metodologia. Ao contrário dos sistemas de modelagem cinemáticos, onde o usuário controla o movimento a partir da especificação de posições e orientações dos objetos ao longo do tempo, nessa abordagem o controle do movimento dos objetos é representado por um conjunto de metas envolvendo quantidades dinâmicas [1] as quais restringem o comportamento dos objetos para metas específicas ao longo do tempo.

Os diferentes níveis de abstração e representação para modelagem de simulações gráficas dinâmicas envolvem quatro níveis de representação [BAR 91b] (vide figura 2.1) .

Para estes níveis de representação, que compõem uma hierarquia, (vide figura 2.1) um objeto pode ser modelado como:

- Primitivas de metas de comportamento ao longo do tempo (tais como restrições de posições e orientações, forças de restrições), onde as propriedades de restrições dos objetos ao longo do tempo são descritas através de modelos de planos (roteiros) de objetivos (trajetórias) pré-definidos,

[1] Animação baseada em física.

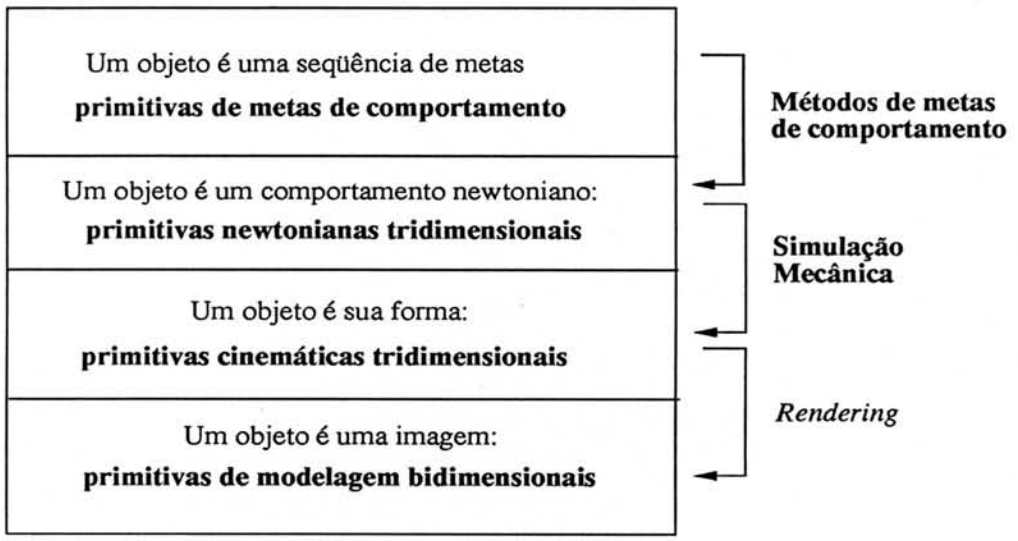

Figura 2.1: Abstração × Níveis de Representação

- Primitivas newtonianas tridimensionais (tais como corpos rígidos dependentes do tempo, corpos flexíveis), onde o comportamento físico dos objetos ao longo do tempo é descrito através de modelos dinâmicos,

- Primitivas cinemáticas tridimensionais (tais como polígonos, superfícies paramétricas), onde a forma instantânea do objeto é descrita através de modelos cinemáticos e

- Primitivas de modelagem bidimensionais (tais como valores de *pixel* e vetores de tela), onde a aparência pictórica do objeto é descrita através de modelos de imagem.

Já a conversão entre estes níveis de representação (vide figura 2.1) é realizada através da utilização de três técnicas:

- **Métodos de Metas de Comportamento:** conversão de primitivas de metas de comportamento para primitivas newtonianas tridimensionais através de métodos de metas de comportamento, tais como dinâmica in-

versa (vide seção 3.2) os quais produzem a interação dos objetos segundo as leis da física,

- **Simulação Mecânica:** redução de primitivas newtonianas tridimensionais para primitivas geométricas tridimensionais através da resolução de equações dinâmicas de movimento da simulação física, as quais produzem as posições e orientações dos objetos e
- **Rendering:** conversão de primitivas tridimensionais para primitivas bidimensionais através de técnicas de *rendering*, as quais convertem a forma de um objeto para uma imagem.

De acordo com essa abordagem com diferentes níveis de abstração e um método matemático garantido, no sentido de obter metas de comportamento específicas, é possível controlar um determinado modelo físico para realizar experimentos pré-definidos.

2.1.2 Abstração × Representação × Controle

Zeltzer [ZEL 91] apresenta uma abordagem para o desenvolvimento de sistemas de simulações gráficas dinâmicas reconfiguráveis a partir de um conjunto de categorias de abstração as quais podem ser associadas a diferentes técnicas de interação para permitir simulações de mundos virtuais em níveis de abstração apropriados. Esta abordagem procura estabelecer relações fundamentais existentes em um ambiente de simulação para que o usuário possa controlar modelos computacionais e, também, representá-los e modificá-los de acordo com o nível de abstração e complexidade que desejar.

Segundo Zeltzer [ZEL 91], para permitir manipulações na complexidade do modelo computacional, três características devem ser observadas durante o desenvolvimento de um sistema de simulação:

- **Simplificação:** no sentido de estabelecer um entendimento do processo a ser simulado, deve-se definir quais processos físicos serão simulados e quais níveis de detalhe serão levados em consideração durante a simulação,

- **Interação:** no sentido de permitir um acesso adequado ao modelo computacional, deve-se identificar quais restrições serão utilizadas durante a simulação e

- **Compreensão:** no sentido de obter uma ferramenta para aprender e resolver problemas em um ambiente virtual, deve-se fornecer transformações suaves entre os níveis de complexidade de um usuário casual até um usuário sofisticado.

Esssas características são incorporadas aos mecanismos de abstração apresentados nessa abordagem, os quais são:

- **Estrutura:** categoria de abstração estrutural a qual especifica atributos cinemáticos e dinâmicos dos objetos. Esses atributos permitem definições estruturais, tais como hierarquia de transformações para um corpo articulado ou propriedades de massa de um objeto descrito fisicamente (vide seção 3.1),

- **Procedimento:** categoria de abstração procedimental a qual controla movimentos de objetos independentemente da estrutura do objeto ou agente. Esses procedimentos de controle podem utilizar técnicas, tais como detecção de colisão, cinemática inversa, dinâmica direta ou deformação elástica (vide seção 3.2),

- **Função:** categoria de abstração funcional a qual associa a estrutura dos objetos com procedimentos de controle com o objetivo de obter um comportamento significativo (comportamento motor) para o objeto. Esse comportamento motor pode ser subdividido em módulos de classes de

movimentos particulares onde cada módulo esta sob o controle de um conjunto de programas motores, ou seja, um conjunto de unidades funcionais e

- **Agente:** categoria de abstração de agente a qual consiste numa definição estrutural composta de um conjunto de unidades funcionais que definem o repertório de comportamento de uma entidade, e de algum mecanismo que possibilite a seleção e seqüenciamento desses comportamentos.

Como visto anteriormente, estes mecanismos de abstração necessitam ser associados a técnicas de interação para permitir simulações de mundos virtuais em níveis de abstração apropriados.

São duas as técnicas de interação:

- **Guiada:** onde é realizada uma especificação explícita dos objetos e comportamento pelo usuário através de dispositivos gráficos de entrada e
- **Programada:** onde é realizada uma especificação dos objetos e comportamento através de alguma notação de programação.

De acordo com uma visão unificada dos mecanismos de abstração e interação existem dois eixos ortogonais (vide figura 2.2) para representar e controlar os objetos em um mundo virtual. Entretanto, apenas um destes eixos - o eixo de abstração - corresponde aos níveis de abstração apresentados por Barr [BAR 91b], uma vez que sua abordagem é mais simples que a de Zeltzer [ZEL 91].

As técnicas de interação (eixo de interação) podem ser utilizadas por todos os níveis de abstração (vide figura 2.3). O nível de abstração mais baixo é representado por objetos e estruturas sem nenhuma abstração procedural e é chamado nível de máquina. Já o nível de tarefa é atingido quando técnicas de interação guiada e programada são utilizadas por agentes através de seu repertório de comportamento.

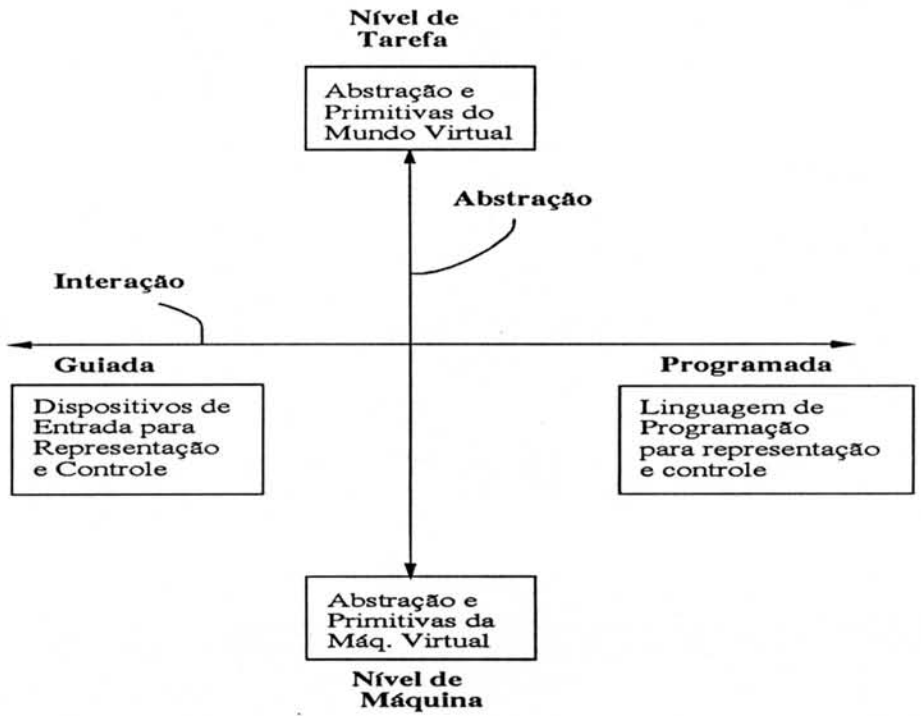

Figura 2.2: Abstração × Interação

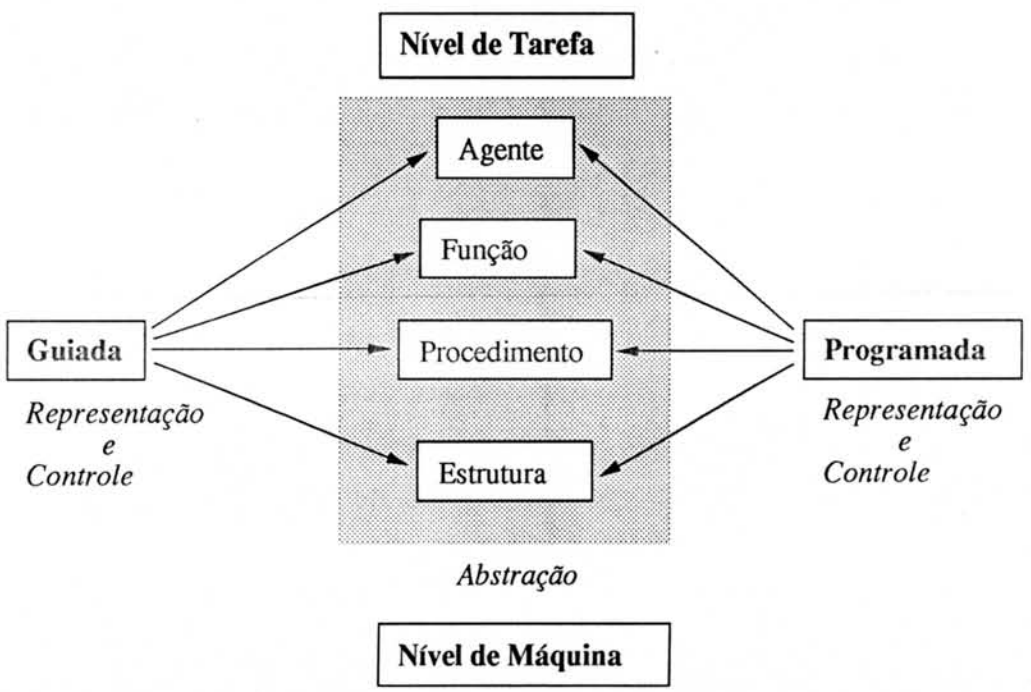

Figura 2.3: Abstração × Representação × Controle

3 SIMULAÇÃO DINÂMICA DE CORPOS RíGIDOS

Entre os trabalhos realizados em simulações gráficas dinâmicas, serão investigados os que abordam a simulação dinâmica de corpos rígidos. Esses trabalhos têm focalizado corpos rígidos articulados e não-articulados [BAR 88b, BAR 89, HAH 88, MOO 88, WIL 90]. Tais trabalhos partem da descrição física dos objetos e da definição das restrições que atuam no mundo virtual. A partir dessa descrição, é realizada a análise da dinâmica de corpos rígidos envolvida levando-se em conta as restrições e efeitos que influenciam o comportamento dos objetos no ambiente. Nessa análise, determina-se uma especificação dinâmica apropriada para formular as equações dinâmicas responsáveis pelo comportamento dos objetos durante a simulação. Definidas as equações, é necessário encontrar formas viáveis para sua resolução. As estratégias de resolução necessitam de métodos numéricos que realizem a integração das equações, para encontrar as novas posições e orientações dos objetos, ao longo do tempo. Desta forma, após a resolução destas equações, as novas posições e orientações serão utilizadas para animar os objetos graficamente.

A simulação baseada em leis físicas permite a geração de movimentos realísticos. Um sistema de simulação dinâmica deve fornecer mecanismos automáticos para controle de movimentos que permitam a realização de projetos de simulações sem grandes esforços por parte do usuário.

Para que a simulação dinâmica ocorra, devem estar associadas aos objetos (corpos rígidos), as seguintes características:

- **atributos geométricos:** posições e orientações no espaço;
- **atributos físicos:** propriedades de massa;
- **estímulos aplicados sobre o objeto:** forças e torques e

- **restrições:** vínculos que atuam no mundo virtual (ambiente).

Estas características determinam a descrição do estado do sistema dinâmico em algum instante de tempo. A partir desse estado dinâmico, é possível determinar sua evolução ao longo do tempo através de equações dinâmicas de movimento.

Após discutir os tópicos acima, este capítulo conclui apresentando o modelo para simulação dinâmica e as técnicas de controle de movimento adotados neste trabalho.

3.1 Modelagem com Simulação Física

As técnicas utilizadas na modelagem com simulação física necessitam de informações da representação geométrica dos objetos e de informações relacionadas à descrição física dos objetos a serem modelados. Neste trabalho, o sistema físico deve ser condizente com a dinâmica clássica de corpos rígidos [GOL 50, MAC 36, MAR 70, SYM 71].

Conforme citado por Wilhelms [WIL 91], a simulação física básica é produzida através das seguintes etapas: descrição física dos objetos a serem manipulados, formulação das equações dinâmicas de movimento, resolução das equações dinâmicas a partir de estímulos iniciais e integração destas equações para encontrar as novas posições e orientações dos objetos.

3.1.1 Descrição Física do Objeto

Serão apresentadas as descrições físicas básicas dos seguintes objetos: partículas, corpos rígidos e corpos rígidos articulados.

Uma partícula pode ser vista como um pequeno corpo representado por um ponto de massa [RES 80]. No espaço tridimensional, uma partícula apresenta três graus de liberdade (coordenadas independentes) podendo transladar, no sistema de coordenadas do universo, nas direções x, y e z. No entanto, o movimento rotacional para partículas é ignorado por ser insignificante.

Um corpo rígido é definido como uma coleção de partículas, restritas de tal forma que as distâncias relativas entre essas partículas permaneçam absolutamente fixas [MAR 70]. A física mostra-nos que essas partículas, observadas de acordo com sua natureza microscópica, sempre apresentam movimento umas em relação às outras. Esse movimento, entretanto, pode ser desconsiderado durante a descrição do movimento de corpos rígidos sob o ponto de vista de sua natureza macroscópica. Por outro lado, deslocamentos macroscópicos entre as partículas (deformações elásticas) podem ocorrer durante uma colisão. Todavia, é possível desconsiderar estas deformações e obter equações de movimentos válidas para expressar a dinâmica de corpos rígidos com alto grau de exatidão.

Sendo visto como uma distribuição contínua de matéria, a massa total de um corpo rígido não é concentrada em um único ponto, mas distribuída no espaço, tornando, assim, seu movimento rotacional significante. No espaço tridimensional um corpo rígido apresenta seis graus de liberdade. A posição do corpo pode transladar, no sistema de referência do universo (SRU) (vide seção 4.1), em três direções x, y e z e a orientação do corpo pode variar em três ângulos independentes, os quais fornecem a orientação do sistema de referência do objeto (SRO) (vide seção 4.1) em relação ao sistema de referência do universo.

Os objetos utilizados para representar os corpos rígidos podem ser formados por uma malha de pontos conectados para formar polígonos ou por superfícies paramétricas.

As propriedades de massa necessárias para simular um corpo rígido são: centro de massa, massa total e tensor de inércia [MAR 70]. O tensor de inércia

representa a distribuição de massa do corpo rígido. Uma vez determinado o tensor de inércia no sistema de coordenadas do corpo, ele não se modifica durante o movimento.

Já um corpo rígido articulado é constituído de segmentos de corpos rígidos conectados por articulações [WIL 87a]. Cada segmento pode ser visto fisicamente como um corpo rígido, mas suas posições e orientações necessitam ser restritas de acordo com o movimento dos outros segmentos e de acordo com sua interação com o mundo virtual. Cada articulação de um corpo articulado tem de um a três graus de liberdade de rotação. Um corpo articulado pode ser representado por uma estrutura em árvore onde as articulações são tomadas como nodos conectados por segmentos. O número total de graus de liberdade de um corpo articulado é correspondente ao somatório dos graus de liberdade de cada articulação existente no corpo adicionado aos seis graus de liberdade (posição e orientação do corpo como um todo) já citados [WIL 87b].

3.1.2 Formulação das Equações Dinâmicas

Isaac Newton apresentou, pela primeira vez (em 1686), nos *Principia mathematica philosophiae naturalis*, a sua síntese da mecânica, composta de três leis de movimento [RES 80]. Essa formulação newtoniana constitui a base da mecânica clássica. Todas as formulações dinâmicas existentes, hoje em dia, podem ser vistas como uma variação das leis de Newton.

Entre as formulações existentes, pode-se citar: Newton [MAR 70], Euler [WIL 88a], Lagrange [WEL 69], Gibbs-Appell [WIL 87b] , D'Alembert [ISA 87] ou formulação recursiva de Armstrong [ARM 87]. Essas formulações expressam o comportamento dos objetos no mundo virtual e, como representam as mesmas leis fundamentais, resultam em movimentos idênticos. A decisão da utilização de uma determinada formulação deve levar em conta as seguintes questões de implementação

[WIL 88b]: velocidade, robustez, facilidade de implementação e a forma pela qual elas podem ser adaptadas para permitir a adição de limites nas articulações ou metas de movimentos.

Equações de movimentos resultantes da formulação newtoniana são mais apropriadas para movimentos simples e, por isto, são normalmente utilizadas na simulação de partículas [GRE 91] e corpos rígidos [BAR 88a, GRE 91, HAH 88]. Em movimentos complexos, esta formulação apresenta equações de movimento difíceis de serem manipuladas matematicamente [MAR 70].

O número de equações de movimento necessárias para especificar o sistema de um corpo rígido articulado é dependente do número de graus de liberdade do sistema [WIL 87b]. As equações de movimento para corpos rígidos articulados são mais complexas que as equações para corpos rígidos não-articulados descritos por partículas e propriedades de massa. A partir dessas características, pesquisadores em animação de corpos rígidos articulados que utilizam a análise dinâmica [ARM 87, ISA 87, WIL 87a] têm procurado formulações mais apropriadas para construir as equações de movimento.

- **Formulação de Gibbs-Appell:** formulação do tipo matriz, como a de Lagrange; utiliza o conceito de forças generalizadas, as quais podem ser representadas através de uma rede de forças e torques (dependendo dos movimentos que a articulação está realizando - deslizando ou rotacionando) atuando em um grau de liberdade particular [WIL 91]. Assim sendo, essa formulação permite tipos de articulações mais gerais e permite que equações de restrições sejam incluídas na matriz. Esta formulação apresenta, também, uma visão clara do estado dinâmico total necessário para simulação de corpos rígidos articulados. Porém, tende a apresentar um grande custo computacional ($O(n^4)$ para n graus de liberdade) por envolver a inversão de uma matriz não esparsa em cada passo de tempo da animação. E apresenta os seguintes pontos relevantes em relação às

questões de implementação: as matrizes geradas com Gibbs-Appell não são esparsas e o custo da solução para as equações é considerado proibitivo pelo enorme custo computacional [WIL 91].

- **Formulação recursiva de Armstrong:** baseia-se nas equações dinâmicas de Euler, mas evita a utilização de uma matriz. Através desta formulação, o corpo pode ser visto como uma árvore de articulações onde assume-se que todas as articulações têm três graus de liberdade de rotação e o corpo sendo especificado no universo por seis graus de liberdade (posição e orientação do corpo como um todo). Esta formulação apresenta um custo computacional linear ($O(n)$ para n graus de liberdade). Conforme citado por Green [GRE 91], esta formulação apresenta os seguintes pontos relevantes em relação às questões de implementação: as equações resultantes não são numericamente bem comportadas nem são fáceis de manipular (matematicamente) da mesma forma que na formulação de Gibbs-Appell.

- **Formulação de D'Alembert:** conforme citado por Isaacs e Cohen [ISA 87], as equações de movimento são derivadas do princípio do trabalho virtual de D'Alembert. [1] A dinâmica utilizada nesta formulação considera objetos articulados estruturados em árvores. Esta formulação apresenta um custo computacional inversamente dependente do passo de tempo necessário para obter uma solução computacional mais próxima possível da matemática e exporencialmente dependente do número de graus de liberdade. Esta formulação apresenta os seguintes pontos relevantes em relação às questões de implementação [ISA 87]: as equações de movimento permitem a imposição de restrições cinemáticas e restrições limitando os graus de liberdade desconhecidos associados a cada articulação.

[1] O princípio do trabalho virtual descreve que se um sistema está em equilíbrio dinâmico e aos corpos é permitido um pequeno movimento (deslocamento) virtual, então, a soma de trabalho das forças aplicadas, o trabalho dos torques fornecidos e o trabalho das forças internas será igual e, conseqüentemente, oposta ao trabalho das trocas de movimento [WIT 77].

3.1.3 Resolução das Equações Dinâmicas

As estratégias de resolução das equações dinâmicas de movimento necessitam de métodos numéricos que realizem a integração dessas equações, de acordo com a descrição do estado do sistema dinâmico, com o objetivo de determinar, ao fim do processo de resolução, as novas posições e orientações dos objetos ao longo do tempo.

As equações de movimento para corpos rígidos (vide capítulo 4) devem corresponder à evolução do movimento translacional e rotacional de um corpo rígido ao longo do tempo de acordo com a informação de seus estímulos iniciais (forças e torques), atributos físicos (propriedades de massa) e restrições (vínculos) que atuam no mundo virtual. Essas equações são formuladas segundo critérios pré-definidos que permitam a manipulação de valores de variáveis conhecidas com o objetivo de encontrar valores de variáveis desconhecidas. As variáveis desconhecidas das equações de movimento são definidas de acordo com as estratégias adotadas pelos métodos de controle de movimento (vide seção 3.2) aplicados à simulação dinâmica e, conseqüentemente, irão influenciar diretamente as estratégias utilizadas durante o processo de resolução das equações dinâmicas.

3.1.4 Integração das Equações Dinâmicas

Na maioria das vezes, as equações dinâmicas não apresentam soluções analíticas diretas forçando a utilização de métodos numéricos para sua resolução.

Equações direfenciais podem ser resolvidas com a utilização do computador através da técnica de análise numérica. Equações diferenciais ordinárias e equações diferenciais parciais podem surgir da modelagem baseada em física. Em análise numérica, existe a seguinte hierarquia (figura 3.1) de complexidade dos problemas [PLA 88a]:

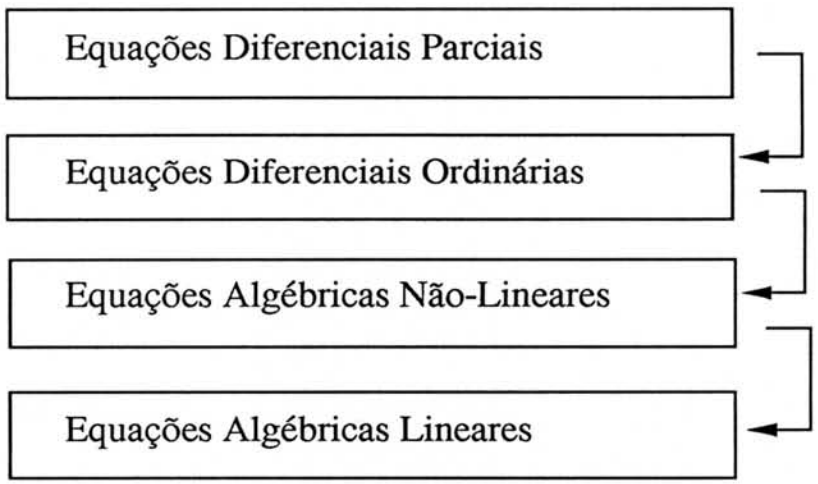

Figura 3.1: Hierarquia de complexidade dos problemas em Análise Numérica

Essa hierarquia esboça a complexidade das equações diminuindo progressivamente até o nível em que as equações são solucionáveis. Nem todos os algoritmos numéricos seguem essa hierarquia. Alguns descem mais níveis ou solucionam tipos de equações que não estão nessa hierarquia, tais como equações integrais.

Conforme citado por Platt [PLA 88a], os níveis de complexidade para esta hierarquia são:

- **Equações diferenciais parciais:** envolvem funções de mais de uma variável e derivadas parciais com relação a mais de uma variável. Existem algumas técnicas que convertem equações diferenciais parciais em equações diferenciais ordinárias,
- **Equações diferenciais ordinárias:** envolvem funções de mais de uma variável, mas derivadas com respeito a somente uma variável. Existem alguns métodos que convertem equações diferenciais ordinárias em equações algébricas não-lineares,
- **Equações algébricas não-lineares:** não podem ser escritas como uma equação matriz ou como um sistema de equações lineares simultâneas.

Entretanto, uma solução local (envolvendo matrizes e vetores) para equações algébricas não-lineares pode ser convertida para solução de equações lineares e

- **Equações lineares:** são simples o suficiente para serem resolvidas.

A literatura apresenta diversas técnicas numéricas [CON 65, PRE 88, RIC 83] que podem ser aplicadas na resolução dos tipos de equações vistas acima. Essas técnicas numéricas procuram simplificar as equações para alguma forma que possa ser interpretada numericamente e buscam uma solução aproximada para as equações.

As soluções resultantes da aplicação de técnicas numéricas nem sempre são estáveis. Situações de instabilidade nos resultados apresentados por um método numérico podem ser causadas pela equação diferencial, pelo método numérico ou pela combinação de ambos. Além disso, a estabilidade das técnicas numéricas pode ser afetada pelas técnicas de controle de movimento aplicadas na simulação dinâmica conforme discutido a seguir. As técnicas de controle são as principais contribuidoras para a rigidez das equações e para os problemas envolvidos em suas soluções numéricas [GRE 91]. Estes problemas surgem quando as equações diferenciais são não-lineares, rígidas ou têm descontinuidade.

Conforme citado por Green [GRE 91], existem três situações de instabilidade:

- **Rigidez:** quando um sistema de equações diferenciais representa uma função que varia muito ao longo do tempo, o sistema é considerado um sistema rígido [PLA 88a]. A solução adotada para tornar os algoritmos estáveis é utilizar passos de tempo muito pequenos. Porém, a redução do passo de tempo resulta em um aumento do tempo computacional necessário para o método integrador encontrar as soluções das equações, e, quando o passo de tempo é reduzido, os erros de arredondamento na com-

putação do método aumentam. Assim sendo, diante destes problemas, é necessário utilizar um método numérico apropriado para equações rígidas que permita a troca de passo de tempo adaptativamente de acordo com o comportamento das equações. Segundo Green [GRE 91], as equações de movimento para formas articuladas, não sujeitas a forças e torques, não são muito rígidas. Quando surge a necessidade de representar o efeito provocado por forças e torques, é provocada, então, a situação de instabilidade,

- **Propriedades de conservação:** a situação de instabilidade surge do fato de que os métodos numéricos aplicados a equações dinâmicas de movimento devem respeitar as leis de conservação que têm sido desenvolvidas na física e, por este motivo, durante a integração dessas equações não se deve permitir, por exemplo, o aumento ou redução de quantidades de energia do sistema e

- **Descontinuidades:** as descontinuidades nas equações de movimento surgem quando mecanismos de controle baseados em estados realizam trocas que modificam muito rapidamente as forças e torques (por exemplo, durante a colisão entre dois corpos rígidos) entre os estados. Assim sendo, é necessário detectar onde ocorre a descontinuidade (esta detecção é realizada pelo método de controle) e obter o valor resultante em torno da descontinuidade. Uma forma de resolver essa situação de instabilidade é integrar a equação de movimento até a descontinuidade e então reiniciar a integração no estado após a descontinuidade. Este método de resolução pode causar variação na velocidade de simulação, mas se esse for o problema, a descrição do movimento pode ser armazenada em quadros-chave e o processo pode ser repetido a partir desta descrição [WIL 91].

3.2 Métodos Automáticos para Controle de Movimentos

Uma vez apresentadas as etapas para modelagem com simulação física, é possível identificar as informações necessárias para o processo de geração de movimentos realísticos, de objetos interagindo com o ambiente, baseado em leis da física. Todavia, sob o ponto de vista do animador, as informações dos estímulos, os quais atuam diretamente nas equações de movimento, podem não ter uma forma intuitiva e podem causar uma extensa sessão de tentativas e erros, até que se encontre os estímulos corretos para realizar a seqüência de movimentos e interações desejáveis no mundo virtual. A forma de contornar essa estratégia singular é fornecer ao sistema de simulação métodos de controle de movimento que permitam a combinação de respostas físicas automáticas (por exemplo, objeto caindo com a força da gravidade, forças e torques resultantes do impacto entre objetos, limites nas articulações dos segmentos de um objeto articulado) com sugestões de controle especificadas pelo usuário (por exemplo, 'encontre os estímulos (forças e torques) necessários para movimentar o objeto A até à posição X') [WIL 88b].

Os métodos automáticos para controle de movimento operam em baixo nível ou em alto nível. Métodos de controle que operam em alto nível apresentam formas mais gerais para expressar o movimento, como termos da linguagem natural (por exemplo, 'caminhe até à porta'), porém necessitam ter, internamente, fundamentações sólidas de controles que operam em baixo nível. Métodos de controle de baixo nível [WIL 91] apresentam formas de controle automático que procuram explorar influências ambientais como: reação realística durante a colisão entre objetos, atrito e amortecimento durante tais interações, limites nas articulações para evitar configurações não naturais, resistência a movimentos indesejáveis, capacidade de restringir segmentos designados em posições particulares e capacidade para especificar e ter o corpo movendo-se corretamente para atingir uma meta específica.

Como não existe, atualmente na instituição, uma ferramenta com a implementação de métodos de controle de baixo nível, decidiu-se investigar métodos de controle que operam preferivelmente em baixo nível. A seguir, serão apresentadas as estratégias e métodos de controle utilizados dentro desse contexto.

3.2.1 Estratégias de Controle

Existem duas abordagens para implementar controles de movimento em baixo nível [WIL 88b], as quais podem ser referenciadas como: controle por pré-processamento e controle por matriz. As duas abordagens assumem que o estado do sistema dinâmico em um tempo ti é conhecido e que o estado em um tempo $ti + \Delta t$ deverá ser calculado.

- **Controle por Pré-processamento:** para esta abordagem as sugestões de controle e restrições são convertidas para forças e torques e incluídas nas equações dinâmicas através de uma forma natural. Esta abordagem divide-se em duas etapas. Numa primeira etapa, encontram-se as forças e torques apropriados, para satisfazer as restrições do modelo físico. Por exemplo, se no modelo físico considera-se a restrição de não interpenetração entre dois objetos durante a colisão, deve-se calcular as forças e torques que evitam esta interpenetração. E na segunda etapa, acrescenta-se às forças e torques calculados, a rede de forças e torques utilizada na análise dinâmica do objeto.

- **Controle por Matriz:** para esta abordagem as sugestões de controle e restrições são especificadas em forma de equações. Esse sistema de equações (equações dinâmicas e equações de restrições) tem como objetivo calcular a rede de forças e torques utilizados na análise dinâmica do objeto, considerando internamente em sua estratégia de resolução as influências do modelo físico. Sendo assim, são utilizadas formas matri-

ciais para representação das variáveis conhecidas e desconhecidas nesse sistema de equações.

3.2.2 Métodos de Controle de Movimento

Quanto aos métodos automáticos de controle, observa-se que para formar a especificação do movimento dos objetos, utilizam-se informações intrínsecas ao sistema de animação e informações fornecidas diretamente pelo animador. Por informações intrínsecas ao sistema de animação pode-se entender aquelas referentes a tipos de influências ambientais abordadas, descrição física dos objetos em movimento e algumas regras de comportamento. Por informações fornecidas pelo animador pode-se entender aquelas referentes aos tipos de movimentos desejados e mudanças nos parâmetros internos ao sistema de animação. Com isso, obtém-se um ganho na redução da complexidade da descrição do movimento e no aperfeiçoamento do realismo dos objetos durante a animação.

A diferença existente entre métodos de controle de baixo nível e de alto nível está no grau de detalhamento em que são fornecidas as informações pelo animador e no grau de complexidade do tratamento das informações intrínsecas ao sistema de animação.

3.2.2.1 Métodos de Controle de Movimento de Baixo Nível

Estes métodos de controle formam a especificação do movimento dos objetos a partir de informações fundamentais do movimento desejado associadas ao grau de complexidade com que são tratadas as informações intrínsecas ao sistema de animação. Normalmente, o grau de complexidade corresponde a classes específicas de movimentos.

Os principais métodos de controle de movimento de baixo nível são:

- **Cinemática -** Com a análise cinemática descreve-se o movimento dos objetos no tempo e no espaço sem levar em conta as forças e torques que causam o movimento. A produção do movimento é realizada através do cálculo de deslocamentos linear e angular, velocidades e acelerações. Dependendo dos parâmetros fornecidos, duas classes de questões surgem:

  1. Cinemática Direta: diz respeito ao cálculo da configuração de um conjunto de objetos (posições), fornecidas as transformações que estão atuando sobre eles. Exemplos são: (1) uma esfera é lançada de um avião (movimento de projéteis) de uma altura igual a $1000m$ com uma velocidade horizontal igual a $10m/s$ e com uma aceleração gravitacional constante igual a $-9,8m/s^2$, no tempo igual a zero e, deseja-se encontrar sua posição no tempo igual a $1s$; (2) fornecida a posição e orientação de uma marionete (objeto articulado) e de seus membros (segmentos do objeto articulado) no universo, no tempo igual a zero, e dado que se move com velocidade constante, deseja-se encontrar a posição da mão (segmento do objeto) no tempo igual a $1s$, a partir da informação dos ângulos de orientação das articulações do ombro, cotovelo e pulso (articulações de segmentos do objeto próximos à mão), no tempo igual a $1s$ e
  2. Cinemática Inversa: diz respeito ao cálculo de uma série de transformações de posições, as quais serão aplicadas a um conjunto de objetos para resultar em uma configuração específica. Essa série de transformações de posições deve respeitar as restrições associadas aos objetos no universo, para permitir que os objetos atinjam suas metas específicas. O mesmo tipo de problema é encontrado também na Robótica. Uma vez determinadas as transformações de posições, deve-se, então, resolver a cinemática direta para resultar no movimento dos objetos. As questões de cinemática inversa correspondem ao inverso das questões de cinemática direta: (1) de qual posição e com que velocidade deve ser lançada uma esfera de um avião (mo-

vimento de projéteis), com uma aceleração gravitacional constante igual a $-9,8m/s^2$, no tempo igual a zero, para alcançar a posição $x = 100m$, $y = 0m$ e $z = 100m$, no tempo igual a $1s$? (2) Quais os ângulos de orientação das articulações do ombro, cotovelo e pulso (articulações de segmentos do objeto próximo à mão), no tempo igual a $0s$, requeridos para produzir uma posição particular da mão (segmento do objeto), a partir da informação da posição e orientação de uma marionete (objeto articulado) e de seus membros no universo, no tempo igual a $1s$, dado que se move com velocidade constante; e

- Dinâmica - Já com a análise dinâmica descreve-se o movimento dos objetos de acordo com suas quantidades dinâmicas, sob ação de forças e torques. O comportamento (movimento) dos objetos é definido por equações dinâmicas de movimento. Da mesma forma que na cinemática, as questões podem ser colocadas de forma direta ou inversa:

  1. Dinâmica Direta: consiste em calcular o movimento de um conjunto de objetos, fornecidas as forças e torques que estão atuando sobre eles. É adotada por outras áreas, além da animação por computador, como por exemplo, na área de CAD/CAM onde a análise dinâmica é utilizada para projetar e testar máquinas. Exemplos são: (1) fornecidas as forças e torques que estão atuando em um cubo posicionado no universo em $x = 10m$, $y = 10m$, e $z = 10m$ com ângulos de orientação do corpo em relação ao universo, ângulo em x igual a 30°, ângulo em y igual a 0° e ângulo em z igual a 10° e com massa igual a $100kg$, no tempo igual a $0s$, deseja-se encontrar sua posição e orientação em relação ao sistema de referência do universo (vide seção 4.1), no tempo igual a $1s$; (2) fornecidas as forças e torques que estão atuando sobre uma alavanca, a qual contém a agulha de um toca-disco (objeto articulado) e fornecidas, também, suas posições e orientações em relação ao universo, no tempo igual a $0s$, deseja-se en-

contrar seu movimento resultante (posições e orientações), no tempo igual a $1s$ e

2. Dinâmica Inversa: consiste em calcular as forças e torques, que devem ser aplicados a um conjunto de objetos, para resultar em um movimento específico. Essas forças e torques necessitam ser ajustadas de acordo com as condições (restrições) do ambiente para corresponderem ao movimento (comportamento) específico. A dinâmica inversa também é utilizada em Robótica, onde o movimento resultante é conhecido e pretende-se descobrir as forças e torques que produzem esse movimento. Uma vez determinadas as forças e torques correspondentes às metas específicas, deve-se, então, resolver a simulação dinâmica direta para resultar no movimento dos objetos. Questões desse tipo são questões da dinâmica direta formuladas de uma forma inversa: (1) quais forças e torques devem ser fornecidos a um cubo posicionado no universo em $x = 10m$, $y = 10m$ e $z = 10m$ com ângulos de orientação do corpo em relação ao universo, ângulo em $x = 30°$, ângulo em $y = 0°$ e ângulo em $z = 10°$ e com massa igual a $100kg$, no tempo igual a $0s$, para que ele se encontre na posição, no universo, posição $x = 100m$, $y = 0m$ e $z = 100m$, no SRU, com ângulos de orientação do corpo em relação ao universo, ângulo $x = 45°$, ângulo em $y = 45°$ e ângulo em $z = 0°$, no tempo igual a $1s$? (2) Quais as forças e torques que devem ser fornecidas a uma alavanca, a qual contém a agulha de um toca-disco (objeto articulado) na posição e orientação de equilíbrio em relação ao universo, no tempo igual a $0s$, para que ela se encontre na posição e orientação de início da primeira faixa do disco em relação ao universo, no tempo igual a $1s$.

- **Satisfação Automática de Restrições -** Este método de controle procura satisfazer, de uma forma automática, restrições geométricas (posições e orientações) associadas aos objetos interagindo com o mundo virtual (por exemplo, reação realística durante a colisão com outros obje-

tos) ou associadas aos objetos interagindo com eles próprios (por exemplo, limites nas articulações de objetos para evitar configurações não naturais). Sistemas baseados em restrições podem ser subrestringidos ou superrestringidos [FOL 90]. Os primeiros ocorrem quando o sistema apresenta muitas soluções as quais satisfazem as restrições (por exemplo, a parte de trás de um quadro pendurado em uma parede deve estar em contato com a parede em um ponto). Já os superrestringidos ocorrem quando o sistema não consegue encontrar uma solução que satisfaça as restrições (por exemplo, a parte de trás e a parte da frente de um quadro pendurado em uma parede devem estar em contato com a parede em um ponto). As restrições geométricas podem ser especificadas através de igualdades ou desigualdades. Sistemas de restrições de igualdades são utilizados para classes específicas de restrições (por exemplo, o objeto A está restringido a permanecer em uma posição específica no universo) e sistemas de restrições de desigualdades são utilizados para classes mais gerais de restrições (por exemplo, faça o objeto A permanecer um pouco mais acima que o objeto B).

Quando procura-se resolver problemas da dinâmica associados a restrições, encontram-se dois conceitos de restrições [MAR 70]: restrições holonômicas, expressas através de relações algébricas entre coordenadas, e restrições não-holonômicas, expressas em termos de velocidades e não podem ser integradas para resultar em relações entre coordenadas. Conforme citado anteriormente, restrições podem estar associadas à análise cinemática e dinâmica. Portanto, restrições cinemáticas ou dinâmicas podem ser descritas para tratar classes específicas de restrições ou classes mais gerais de restrições e

- **Detecção de Colisões e Resposta -** Este método de controle aborda, especificamente, influências ambientais relacionadas com a interação entre objetos (por exemplo, objeto A interagindo com o ambiente, ou objeto A interagindo com o objeto B, ou objeto articulado A interagindo con-

sigo mesmo). Essas interações são simuladas utilizando algoritmos de detecção de colisões para modelar as restrições gerais e estratégias de respostas automáticas para modelar os efeitos da colisão (limites nas articulações podem ser tratados como um tipo de colisão). Algoritmos de detecção de colisões têm o objetivo de identificar quando uma colisão ocorre e os pontos de contato envolvidos nessa colisão. Essa detecção é realizada através de testes envolvendo o posicionamento dos objetos no universo ao longo do tempo. Esses testes são realizados de acordo com o tipo do modelo geométrico do objeto (por exemplo, objetos modelados com grade de pontos conectados para formar polígonos, ou objetos modelados por superfícies paramétricas) e de acordo com o número de pontos de contato que estão sendo considerados durante a colisão (por exemplo, um ponto de contato ou n pontos de contato). O problema da detecção de colisão entre objetos tem sido abordado, também, pela robótica e geometria computacional. Após a detecção da colisão, deve ser aplicada entre o(s) ponto(s) de contato, uma estratégia de respostas automáticas aos efeitos da colisão para determinar a evolução do comportamento dos objetos ao longo do tempo de acordo com leis físicas. Os resultados da resposta aos efeitos da colisão podem ser um novo movimento do objeto ou o objeto permanecer em repouso sobre outro objeto. Para ambos os casos, as respostas aos efeitos da colisão devem determinar as forças que estão atuando entre este(s) ponto(s) de contato, as quais não permitem a interpenetração entre os objetos (restrição para corpos rígidos) e, se houver movimento resultante deste contato, as novas forças e torques que estão atuando no movimento dos objetos. Essas respostas podem ser simuladas por métodos de penalização (inserção de forças mola, força proporcional à profundidade de interpenetração, e amortecedor, força proporcional à velocidade do movimento, entre os pontos de contato através de estratégias de controle por pré-processamento), inserção de novas equações de restrições através de estratégias de controle

por matriz e por métodos analíticos que calculam a resposta da colisão através da dinâmica de impacto.

3.2.2.2 Métodos de Controle de Movimento de Alto Nível

Estes métodos de controle formam a especificação do movimento dos objetos, basicamente, de duas maneiras: (1) através de termos mais gerais, como ordens em língua natural, permanecendo internamente como responsabilidade do sistema a transformação das instruções de movimento de alto nível para primitivas de baixo nível utilizando sistemas de controle hierárquico, os quais se concentram principalmente na locomoção de corpos articulados [BAD 87, BRU 89, CHA 89, GIR 87, McK 90, ZEL 82]; e, (2) através de comportamentos pré-definidos associados aos objetos de forma a serem utilizados para gerar seus movimentos automaticamente, permanecendo, também, internamente como responsabilidade do sistema a transformação do comportamento associado a cada objeto para primitivas de baixo nível utilizando controle comportamental ou de estímulos e resposta, os quais se concentram principalmente na definição de caminhos de objetos dentro de um ambiente [REY 87, WIL 90].

Dentre os trabalhos em controle de movimento de alto nível que utilizam análise dinâmica, pode-se citar Bruderlin [BRU 89] e Wilhelms [WIL 90].

Bruderlin [BRU 89] apresenta uma abordagem híbrida para locomoção de corpos articulados combinando controle de movimento ao nível de tarefa, o qual facilita a definição do movimento, com controle de movimento dinâmico, o qual permite a produção de movimentos realísticos. Essa abordagem permite a construção de instruções de alto nível como, por exemplo, 'Caminhe com uma velocidade = 10 m/s, comprimento do passo = 5 m e freqüência do passo = 10 passos/s' onde estes parâmetros, referentes à locomoção de corpos articulados, especificados pelo usuário, são decompostos para resultar nas forças e torques responsáveis pela geração da locomoção desejável. Esta abordagem permite a especificação do movimento de

uma maneira conveniente e animações realísticas são obtidas baseadas em equações dinâmicas de movimento sem a especificação explícita de forças e torques [BRU 89].

Wilhelms [WIL 90] apresenta uma abordagem para um sistema de animação comportamental utilizando uma rede de trabalho interativa. A descrição dos objetos é feita através de técnicas de modelagem com simulação física. A partir dessa descrição, a geração do movimento é realizada através do uso de regras que definem como os objetos reagem ao seu ambiente (mundo virtual). Essa abordagem associa aos objetos animados sensores (detecção de qualidades (cores, textura, etc) e proximidade de outros objetos) e efeitos (respostas às interações dos objetos no ambiente). Ao usuário cabe projetar interativamente um mapeamento entre eles utilizando conexões (passagens dos sinais de sensores para efeitos) e nodos (controle de mapeamento de sinais). Essa abordagem para exploração do comportamento dos objetos em um determinado ambiente apresenta-se muito útil quando vários objetos utilizando regras simples de comportamento devem ser animados [WIL 90].

3.2.3 Trabalhos Realizados em Estratégias e Métodos de Controle de Movimento de Baixo Nível

Conforme citado por Foley *et al.* [FOL 90], trabalhos realizados na síntese de movimentos realísticos apresentam uma variação contínua entre fundamentações científicas e abordagens *ad hoc*, o que não resulta numa compreensão muito clara da linha que divide estas duas formas de tratar o controle de movimentos.

Sendo assim, serão apresentados a seguir trabalhos realizados em estratégias e métodos de controle de movimentos de baixo nível os quais utilizam, eventualmente, métodos híbridos de controle para explorar influências ambientais, já citadas anteriormente, como: reação realística durante a colisão com outros objetos, atrito e amortecimento durante tais interações, limites nas articulações para evitar configurações não naturais, resistência a movimentos indesejáveis, capacidade

de restringir segmentos designados em posições particulares e capacidade para especificar e ter o corpo movendo-se corretamente para atingir uma meta específica.

Hahn [HAH 88] utiliza métodos de controle cinemáticos e dinâmicos para simular a interação entre corpos rígidos, articulados ou não-articulados, com o ambiente. Essas interações entre os objetos são simuladas utilizando a detecção da colisão para modelar as restrições (por exemplo, não interpenetração dos objetos durante as colisões, não interpenetração entre os membros pertencentes às articulações de um objeto articulado) e análise do impacto para resolver a dinâmica. A detecção da colisão é realizada para uma série de pares de objetos, modelados geometricamente com uma grade de pontos conectados para formar polígonos, os quais colidem-se em um único ponto. A resposta automática aos efeitos da colisão é simulada por um método analítico que calcula a resposta da colisão através da dinâmica do impacto. Esse método analítico resolve um conjunto de equações que descrevem como os momentos linear e angular são conservados durante a colisão envolvendo apenas velocidades e não acelerações. Essa análise do impacto permite simular interações com atrito (possibilitando a transição do contato entre objetos de escorregadio a aderente) e com elasticidade (possibilitando a transição do contato entre objetos de instantâneo a contínuo). O contato instantâneo é modelado como uma série de colisões únicas, sendo que o contato contínuo (em repouso) é aproximado por uma série de colisões únicas ocorrendo freqüentemente. Quanto aos movimentos dos objetos articulados, esses podem ser especificados diretamente através da informação das posições e orientações das trajetórias das articulações e pela utilização da análise dinâmica, como visto acima, para modelar os efeitos externos dos movimentos dos membros do objeto articulado e para ajustar as forças e torques que estão atuando no objeto como um todo. Para a análise dinâmica interpretar o movimento cinemático especificado, essas posições e orientações necessitam ser integradas numericamente para obter suas velocidades em função do tempo. O cálculo da média das velocidades obtidas e as velocidades fornecidas pela análise do impacto (se houver) podem ser utilizadas para mover o objeto. Os movimentos também podem ser especificados diretamente através da informação das forças e

torques que estão atuando no objeto, possibilitando, desta forma, a utilização direta do método de detecção de colisão e resposta.

Moore e Wilhelms [MOO 88] utilizam detecção da colisão, para modelar as restrições entre corpos rígidos articulados ou não-articulados, e resposta, para resolver a dinâmica resultante da interação entre objetos de acordo com leis físicas. Para detecção da colisão são apresentados dois algoritmos: um para tratar objetos flexíveis modelados geometricamente com uma grade de pontos conectados para formar polígonos e outro para tratar objetos rígidos modelados como poliedros. Essa detecção da colisão é realizada para uma série de pares de objetos, os quais colidem-se em um único ponto. A resposta aos efeitos da colisão permite simular interações com atrito e com elasticidade. Para resposta automática de contatos instantâneos é utilizado um método analítico, o qual resolve um conjunto de equações que descrevem como os momentos linear e angular são conservados durante a colisão envolvendo apenas velocidades e não acelerações, similar ao utilizado por Hahn [HAH 88], modelando o contato instantâneo como uma série de choques leves únicos. Para resposta automática de contatos contínuos (em repouso) é utilizado um método de penalização, o qual insere forças mola, em sentidos iguais e opostos, entre dois objetos em repouso, para evitar a interpenetração, a qual poderia ser causada com a força gravitacional. Ambas abordagens de respostas automáticas são estendidas para tratar a interação entre corpos rígidos articulados.

Barzel e Barr [BAR 88b] apresentam um método de satisfação automática de restrições utilizando análise dinâmica (denominado restrições dinâmicas) com o objetivo de simular o comportamento de corpos rígidos articulados e não-articulados, através da mecânica newtoniana, para uma classe específica de restrições geométricas associadas aos objetos em um ambiente. A técnica de restrições dinâmicas permite ao animador controlar o comportamento dos corpos a partir da especificação de restrições adotando o método de dinâmica inversa. Essa técnica implementa controle de movimentos de baixo nível utilizando equações de restrições as quais resultam nas forças que devem ser aplicadas aos objetos para permitir que sejam encontradas

as posições e orientações do movimento desejado. Logo, o movimento dos objetos é de acordo com os efeitos de inércia, forças e torques atuando no objeto, onde muitas destas forças e torques podem ser fornecidas externamente, e outras, todavia, são derivadas das restrições geométricas. Essa modelagem para o sistema suporta restrições como: fixar um ponto de um corpo para uma localização específica no espaço, fixar um ponto entre dois objetos durante seu movimento, definir o caminho que um ponto de um objeto deve seguir e restringir um objeto para uma orientação específica.

Baraff [BAR 89] simula dinamicamente movimentos e interações de corpos rígidos articulados e não-articulados, através da informação de estímulos iniciais, utilizando somente métodos analíticos no tratamento da interação entre corpos rígidos e dos mesmos com o ambiente. Na resolução da dinâmica do movimento dos objetos são utilizados sistemas de equações diferencias ordinárias similares aos utilizados por Barzel e Barr [BAR 88b]. O algoritmo de detecção da colisão utilizado para encontrar o tempo exato da colisão adota métodos 'backtracking' similares aos descritos por Moore e Wilhelms [MOO 88]. Para o tratamento das colisões é apresentado um método analítico para calcular as forças que atuam em contato contínuo (em repouso), o qual calcula as forças que surgiram naturalmente para evitar a interpenetração entre os corpos, aplicado a corpos rígidos poliédricos, os quais colidem-se em múltiplos pontos sem considerar o atrito. Esse método pode ser estendido para calcular forças que atuam em contato instantâneo e os corpos podem ser restringidos para satisfazer relações geométricas como aquelas apresentadas por Barzel e Barr [BAR 88b] e Isaacs e Cohen [ISA 87]. Essas relações são expressas através de um sistema de equações de restrições onde a formulação analítica utilizada permite o tratamento de restrições holonômicas e não-holonômicas de uma forma consistente. A solução analítica desse sistema de equações utiliza técnicas de programação linear e resolve heuristicamente um sistema de restrições de igualdades e desigualdades para satisfazer as condições apresentadas pela formulação analítica. Esse sistema de restrições evita interpenetrações e satisfaz as leis da dinâmica newtoniana [BAR 89].

Baraff [BAR 90] estende a formulação dinâmica apresentada em seu trabalho anterior [BAR 89], referente ao cálculo de forças de contato contínuo (em repouso) entre pares de corpos, para permitir o tratamento de corpos rígidos construídos através de polígonos ou superfícies curvas. O método analítico apresentado determina a força de contato (sem considerar o atrito) entre corpos rígidos, sem a necessidade de restringir o movimento dos objetos para classes específicas de movimentos tangenciais (por exemplo, não permitir o contato deslizante entre pares de objetos). A análise realizada para o problema de calcular analiticamente as forças de contato entre superfícies curvas é restringido para um número finito de pontos de contato. As condições estabelecidas na verificação de interpenetração entre pontos de objetos distintos são construídos através de conceitos obtidos na formulação de multiplicadores de Lagrange. É apresentado, também, um algoritmo para detecção de colisão entre corpos rígidos poliédricos ou construídos por superfícies curvas convexas que explora a coerência geométrica ao longo do tempo relacionada a problemas de detecção de colisões sucessivas utilizando informações de intervalos de tempos anteriores para resolver o problema de detecção no intervalo de tempo corrente com o objetivo de obter uma maior eficiência durante sua execução.

E por fim, Baraff [BAR 91a] apresenta uma solução analítica para calcular as forças de atrito que atuam durante a interação de corpos rígidos poliédricos para um número finito de pontos de contato. As regiões de contato consideradas são os vértices do segmento de uma linha e as faces poligonais. As forças de contato são classificadas por forças normais (forças que atuam perpendicularmente à superfície de contato) e forças de atrito (forças que atuam tangencialmente à superfície de contato e com movimento oposto ao sentido do deslizar do objeto). As forças de atrito podem ser dinâmicas (dois corpos deslizando em um ponto de contato) ou estáticas (dois corpos parados em um ponto de contato). As forças de contato entre corpos devem satisfazer o modelo de Coulomb de atrito [BAR 91a]. A partir de um princípio tradicional da mecânica, o qual diz que as forças de contato são impulsivas se e somente se forças de contato não-impulsivas são insuficientes para manter restrições de interpenetração entre os corpos, seu trabalho mostra que este

princípio requer um algoritmo de tempo exponencial. É apresentada, então, uma formulação natural deste princípio onde obtém-se um algoritmo com comportamento de tempo polinomial para calcular as forças de contato.

3.3 Modelo Proposto para Simulação Dinâmica de Corpos Rígidos

Conforme mencionado anteriormente, este trabalho tem como objetivo abordar a animação de corpos rígidos não-articulados e o tratamento de eventuais interações (colisões) entre os corpos ou dos mesmos com o ambiente através de métodos de controle de movimento utilizando a análise dinâmica. Segundo este enfoque, após a análise da dinâmica de corpos rígidos envolvida, o modelo de simulação é definido de forma a atender estes objetivos de projeto.

As etapas que descrevem este modelo são: modelagem com simulação física, métodos automáticos para controle de movimentos e a arquitetura geral para o sistema computacional.

3.3.1 Modelagem com Simulação Física

São descritas as informações utilizadas para representação geométrica e atributos dos objetos a serem simulados.

- **Descrição física do objeto:** são considerados objetos (corpos rígidos) simétricos com geometria simples modelados geometricamente por uma malha de pontos conectados para formar polígonos (por exemplo, esferas, cubos, paralelepípedos, pirâmides, cilindros e cones) e é assumida uma distribuição homogênea de massa. Cada tipo de corpo rígido tem definidos os atributos necessários para simulação física (centro de massa, eixos

principais, momento de inércia sobre o centro de massa, massa total e elasticidade do material).

- **Formulação das equações dinâmicas:** para formular as equações dinâmicas utiliza-se a formulação newtoniana [MAR 70] por ser mais apropriada à dinâmica do movimento de corpos rígidos (vide capítulo 4). E, para o tratamento das colisões, utiliza-se um método de penalização e um método analítico (baseado nas leis da dinâmica newtoniana) (vide capítulo 5).

- Resolução das Equações Dinâmicas: como visto anteriormente, as estratégias de resolução das equações dinâmicas são definidas segundo critérios relacionados aos métodos de controle de movimento utilizados na simulação dinâmica (vide seção 3.3.2). Esses critérios definem quais variáveis são conhecidas e quais variáveis são desconhecidas ao longo do tempo. Atributos físicos, tais como, centro de massa, massa total, tensor de inércia e elasticidade do material são considerados variáveis conhecidas das equações de movimento e são assumidos constantes. As variáveis desconhecidas, após a informação dos estímulos iniciais (forças e torques externos iniciais, velocidades linear e angular iniciais, posições e orientações iniciais) são as novas velocidades linear e angular e as novas posições e orientações dos objetos ao longo do tempo.

- Integração das equações dinâmicas: para produzir o movimento dos objetos, são resolvidos sistemas de equações diferenciais ordinárias (vide capitulo 4), acopladas ou não, utilizando-se o método de Runge-Kutta de quarta ordem [PRE 88]. Para produzir o tratamento de colisões, são resolvidos sistemas de equações lineares (vide capítulo 5), utilizando-se o método de decomposição LU [PRE 88].

3.3.2 Métodos Automáticos para Controle de Movimentos

Neste trabalho procura-se utilizar métodos de controle de movimento de baixo nível os quais exploram influências ambientais como: movimentos realísticos dos objetos sob ação de forças e torques externos incluindo a força gravitacional, reação realística durante a colisão com outros objetos e amortecimento durante tais interações. Os métodos de controle abordados são:

- **Análise dinâmica:** utilizada para descrever o movimento dos objetos através da informação de forças externas atuando no corpo e
- **Detecção de colisões e resposta:** utilizada com o objetivo de simular interações entre objetos ou dos mesmos com o ambiente.

A abordagem utilizada para implementar esses controles de movimento em baixo nível é o controle por pré-processamento.

3.3.3 Arquitetura Geral para o Sistema Computacional

O sistema computacional procura resolver o processo de simulação a partir da descrição geométrica dos objetos, da descrição dos atributos físicos e das entradas para simulação durante um determinado intervalo de tempo de simulação através da resolução da dinâmica do movimento para corpos rígidos e tratamento das interações dos objetos ao longo do tempo utilizando o fluxograma geral para simulação dinâmica mostrada na figura 3.2.

Estas etapas serão descritas, em detalhe, nos capítulos subseqüentes.

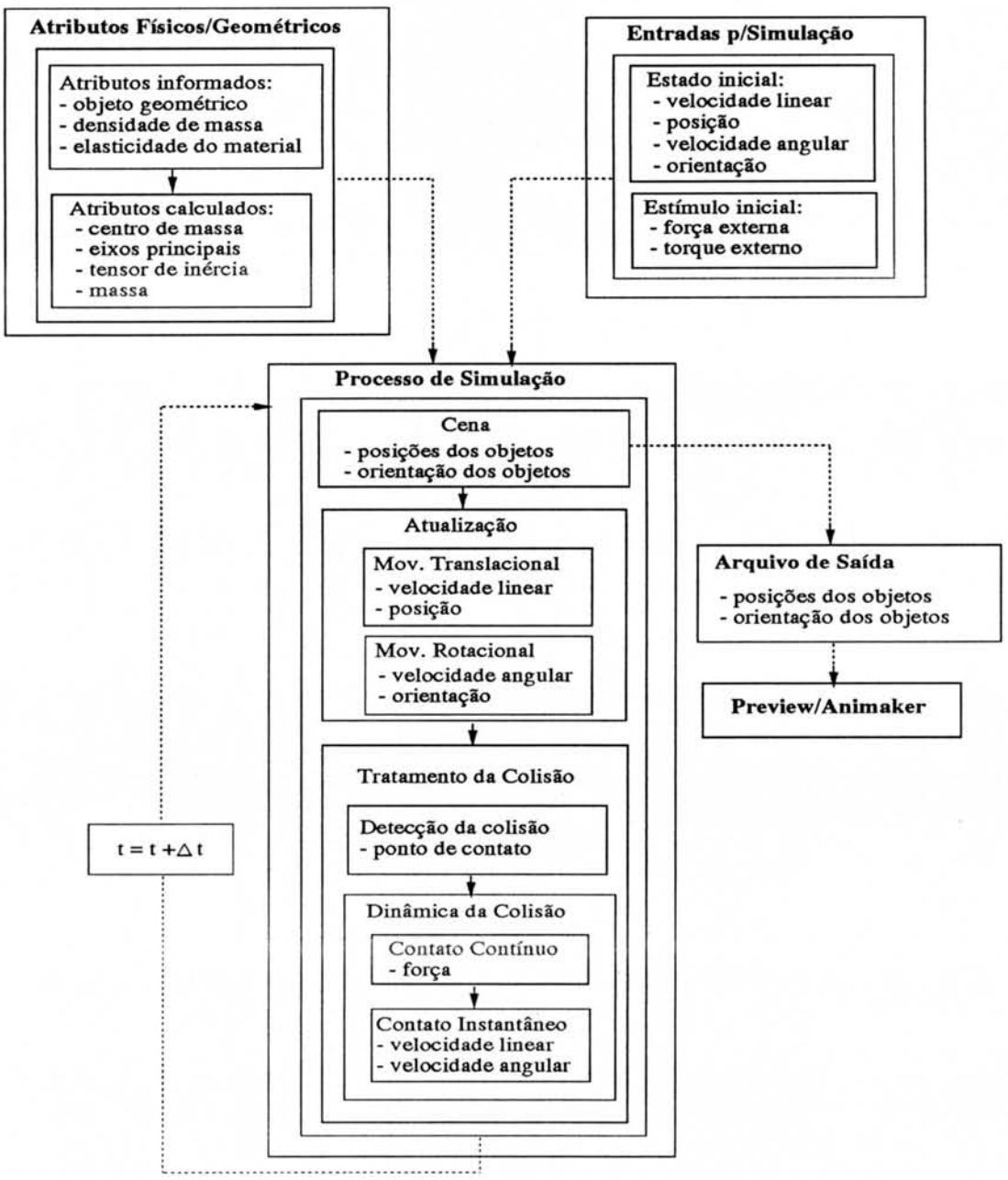

Figura 3.2: Fluxograma geral para simulação dinâmica

4 DINÂMICA DO MOVIMENTO DE CORPOS RíGIDOS

A análise dinâmica utilizada em animação por computador como um método de controle de movimento considera os objetos, pertencentes a uma seqüência de cenas animadas, compostos por atributos geométricos e propriedades de massa sob a influência de forças e torques.

Abordagens para modelagem com simulação física que fazem uso de análise dinâmica permitem a modelagem de mundos virtuais onde o movimento dos objetos, nesse mundo virtual, dá-se de acordo com restrições estabelecidas previamente.

A análise dinâmica torna-se problemática quando se deseja controlar o movimento produzido pela simulação dinâmica. Movimentos simples sob o ponto de vista físico (por exemplo, especificar a trajetória de um corpo rígido, sob a ação da gravidade, para uma determinada posição específica) são fáceis de produzir, mas movimentos complexos (por exemplo, especificar a trajetória de vários corpos rígidos, sob a ação da gravidade, para um conjunto de posições específicas evitando possíveis interações com o ambiente) podem ser mais difíceis de produzir. Outro problema associado com a utilização da análise dinâmica está no desenvolvimento dos métodos numéricos para resolução das equações diferenciais ao longo do tempo, os quais podem apresentar problemas, tais como, rigidez, propriedades de conservação e descontinuidades (vide seção 3.1.4). Estas duas classes de problemas estão intrinsecamente associadas. Normalmente, quando se deseja controlar movimentos mais complexos surgem problemas de instabilidade na resolução das equações dinâmicas de movimento. No entanto, uma vez encontradas as formas de como solucionar esses problemas, a análise dinâmica como um método de controle de movimento apresenta resultados consideravelmente realísticos e está se tornando uma técnica padrão em animação por computador [GRE 91].

A seguir, são apresentadas estratégias para dinâmica do movimento de corpos rígidos utilizadas em animação por computador que exploram a geração de movimentos fundamentais (translacionais e rotacionais) de corpos rígidos sob ação de forças e torques (vide figura 3.2).

4.1 Movimento de Corpos Rígidos

Uma estratégia geral para representação da dinâmica do movimento de corpos rígidos em seqüências de cenas animadas por computador é apresentada. A estratégia associa conceitos utilizados em técnicas de visualização de objetos tridimensionais [LEM 90] com conceitos utilizados na dinâmica de corpos rígidos [MAR 70].

Conforme foi mencionado anteriormente, um corpo rígido apresenta seis graus de liberdade, correspondendo às posições e orientações que ele pode assumir no espaço tridimensional. Sob o ponto de vista dinâmico, essas posições e orientações ao longo do tempo podem ser decompostas em dois tipos de movimento, respectivamente: movimento translacional do centro de massa do corpo sob ação de forças externas e movimento rotacional ao redor do centro de massa do corpo sob a ação de torques externos. Sob o ponto de vista da visualização tridimensional, para descrição desses movimentos são utilizados dois sistemas de referência (vide figura 4.1): sistema de referência do universo (SRU), denominação do sistema de referência inercial, fixo no espaço, para o qual assume-se a regra de sistema de referência mão-direita, e sistema de referência do objeto (SRO), fixo no corpo, com origem no centro de massa do corpo, para o qual assume-se a regra do sistema de referência mão-esquerda.

Na geração do movimento dos corpos rígidos para animação por computador, a escala utilizada nos sistemas de referência tem sentido físico e deve ser associada a ela uma unidade de medida. Desta forma, optou-se pelo sistema MKS de unidades [RES 80].

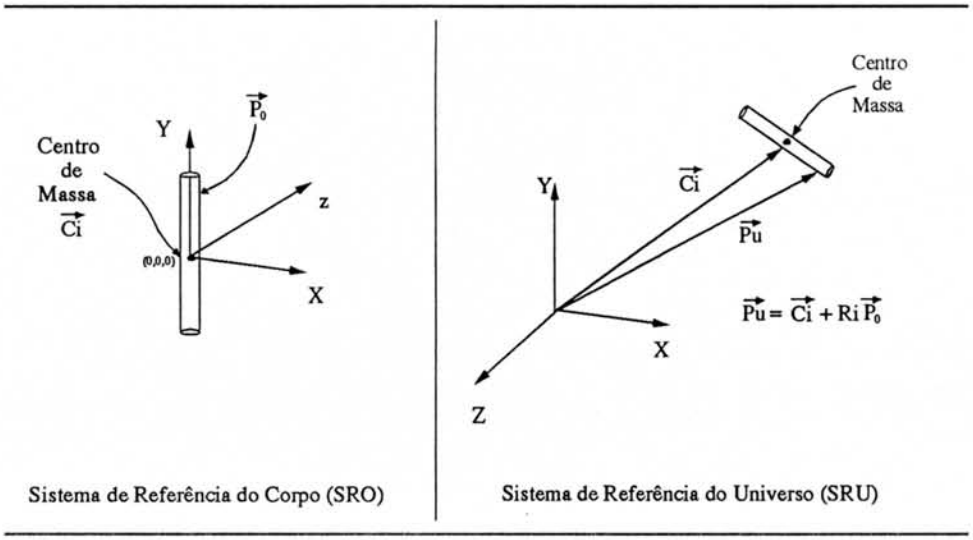

Figura 4.1: Transformações de um corpo rígido do SRO para o SRU através de rotações e translações

Como se necessita da representação dos objetos de uma cena animada em um sistema referencial inercial, ou seja, no SRU, para posterior visualização em um dispositivo de saída gráfica, as coordenadas de um corpo rígido representadas no SRO podem ser transformadas para o SRU através de rotações e translações:

$$\vec{P}_u = \vec{c}_i + R_i \vec{P}_o \tag{4.1}$$

onde, para um corpo i, $\vec{P}_u$ é um ponto P do objeto representado no SRU, $\vec{c}_i$ é o vetor posição do centro de massa do corpo (posição representada no SRU), R_i é a matriz de rotação (orientação) e $\vec{P}_o$ é o ponto P do objeto representado no SRO.

A matriz de rotação R_i, a qual especifica as propriedades de transformação das coordenadas de um ponto, é definida como:

$$R_i = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

Os componentes da matriz R_i são:

$$a_{11} = \cos\beta\cos\alpha \tag{4.2}$$

$$a_{12} = -\sin\beta\cos\alpha \tag{4.3}$$

$$a_{13} = \sin\alpha \tag{4.4}$$

$$a_{21} = \sin\beta\cos\theta + \cos\beta\sin\alpha\sin\theta \tag{4.5}$$

$$a_{22} = \cos\beta\cos\theta - \sin\beta\sin\alpha\sin\theta \tag{4.6}$$

$$a_{23} = -\cos\alpha\sin\theta \tag{4.7}$$

$$a_{31} = \sin\beta\sin\theta - \cos\beta\sin\alpha\cos\theta \tag{4.8}$$

$$a_{32} = \cos\beta\sin\theta + \sin\beta\sin\alpha\cos\theta \tag{4.9}$$

$$a_{33} = \cos\alpha\cos\theta \tag{4.10}$$

onde θ, α e β são os ângulos de rotação no sentido anti-horário sobre os eixos x, y e z, respectivamente. Em relação à ordem dessas rotações assume-se que, primeiramente, é realizada uma rotação do eixo x com relação ao eixo original x, depois, uma rotação do eixo y com relação ao eixo y criado pela rotação em x e, por fim, uma rotação do eixo z com relação ao eixo z criado pelas duas rotações anteriores.

Como se vê na equação 4.1, conhecida a posição de um ponto do objeto ($\vec{P}_o$), com relação ao centro de massa ($\vec{c}_i$), aplica-se a matriz de rotação R_i do objeto i, e realiza-se a rotação desse ponto do objeto com relação ao centro de massa. A adição ao vetor $\vec{c}_i$ realiza a translação do objeto, já que o movimento translacional de um objeto pode ser descrito pela translação do seu centro de massa (vide figura 4.1). Desta forma, verifica-se que a dinâmica do movimento de um objeto é realizada tratando, isoladamente, a translação e a rotação desse objeto.

Sendo assim, a cada incremento de tempo Δt são fornecidos (vide equação 4.1) os elementos do vetor de deslocamento $\vec{c}_i$ e os elementos da matriz de orientação R_i dos objetos em uma cena a serem utilizados no processo de simulação (vide figura 3.2) com o objetivo de gerar a próxima cena de animação.

4.2 Dinâmica do Movimento Translacional de Corpos Rígidos sob Ação de Forças Externas

Como foi visto anteriormente, o movimento translacional de um objeto pode ser descrito pela translação do seu centro de massa. Ainda mais o movimento do centro de massa $\vec{c}_i$ de um objeto pode ser descrito como o movimento de um ponto de massa, ou seja, uma partícula (vide seção 3.1.1).

A equação da dinâmica do movimento de uma partícula é representada pela segunda lei de Newton [MAR 70]:

$$\vec{F} = m\vec{a} = m\frac{d^2\vec{c}}{dt^2} = m\frac{d\vec{v}}{dt} \tag{4.11}$$

onde $\vec{F}$ é o vetor de força externa atuando na partícula, m é a massa da partícula e $\vec{a}$ é o vetor de aceleração da partícula.

A equação 4.11 é uma equação diferencial, pois a aceleração é uma função do tempo. Como se necessita das posições do centro de massa de um objeto ao longo do tempo e não, exatamente, de sua aceleração ao longo do tempo pode-se representar a equação 4.11 através de um conjunto de equações diferencias ordinárias acopladas [GRE 91, HAH 88]:

$$\begin{aligned} m\frac{d\vec{v}_i}{dt} &= \sum_j \vec{F}_j \\ \frac{d\vec{c}_i}{dt} &= \vec{v}_i \end{aligned} \tag{4.12}$$

onde, para um corpo i, $\vec{F}_j$ são os vetores de forças externas atuando no corpo, m é a massa total do corpo, $\vec{v}_i$ é o vetor velocidade linear do corpo e $\vec{c}_i$ é o vetor posição do centro de massa do corpo (posição em função do tempo). A representação destes vetores é realizada no SRU.

A massa total do corpo, m, é definida como:

$$m = \rho v \tag{4.13}$$

onde ρ é a densidade de massa e v é o volume do corpo rígido.

Desta forma, é possível através da resolução do conjunto de equações 4.12 obter as posições do centro de massa $\vec{c}_i$ (vide equação 4.1) de um objeto ao longo do tempo.

4.3 Dinâmica do Movimento Rotacional de Corpos Rígidos sob Ação de Torques Externos

O movimento rotacional sobre o centro de massa de um corpo rígido é determinado por torques externos atuando no corpo.

A representação da dinâmica rotacional de corpos rígidos pode ser realizada de diversas maneiras. As principais estratégias investigadas serão apresentadas em duas etapas: movimento rotacional do corpo e representação da orientação do corpo.

4.3.1 Movimento Rotacional do Corpo

Conforme mencionado anteriormente, a massa total de um corpo rígido não é concentrada em um único ponto, mas distribuída no espaço. Sendo assim, deve-se conhecer a distribuição de massa do corpo para poder representar seu movimento rotacional.

A distribuição de massa do corpo é representada por uma matriz tensor de inércia I definida em um dado sistema de referência [GRE 91, HAH 88, MAR 70, WIL 88a].

$$I = \begin{bmatrix} I_x & -I_{xy} & -I_{xz} \\ -I_{xy} & I_y & -I_{yz} \\ -I_{xz} & -I_{yz} & I_z \end{bmatrix}$$

Os elementos da diagonal da matriz I são:

$$I_x = \int \rho(y^2 + z^2)dv \tag{4.14}$$

$$I_y = \int \rho(x^2 + z^2)dv \tag{4.15}$$

$$I_z = \int \rho(x^2 + y^2)dv \tag{4.16}$$

e os elementos não-pertencentes à diagonal são definidos por

$$I_{xy} = \int \rho(xy)dv \tag{4.17}$$

$$I_{xz} = \int \rho(xz)dv \tag{4.18}$$

$$I_{yz} = \int \rho(yz)dv \tag{4.19}$$

onde ρ é a densidade do objeto e todas as integrações acima são realizadas sobre o volume do objeto (x, y, e z são as direções do sistema de referência em questão).

Os elementos da diagonal da matriz I (I_x, I_y e I_z) representam os momentos de inércia sobre os eixos x, y e z, respectivamente, e os elementos não-pertencentes à diagonal representam os produtos de inércia. Para objetos simétricos dispostos em torno do centro de massa, os elementos do produto de inércia relativos a esse centro de massa são nulos.

Para objetos simétricos com geometria simples existem formas diretas de calcular esses momentos de inércia [CHI 78]. Por exemplo, para um paralelepípedo regular de lados a, b e c com origem no centro de massa e com dimensões c em x, b

em y, e a em z, os momentos de inércia ao redor do centro de massa são:

$$I_x = \frac{1}{12}m(a^2 + b^2) \tag{4.20}$$

$$I_y = \frac{1}{12}m(a^2 + c^2) \tag{4.21}$$

$$I_z = \frac{1}{12}m(b^2 + c^2) \tag{4.22}$$

Se for escolhido representar os momentos de inércia em um sistema de referência inercial (SRU), à medida que o corpo se movimenta, a matriz I mudará ao longo do tempo, assumindo que o corpo está rotacionando. Com isso, é necessário transformar os momentos de inércia através de rotações e translações para o SRO, ao longo do tempo.

Por outro lado, existe um sistema de referência em que a matriz I é uma matriz diagonal. Nesse sistema, os vetores base são chamados eixos principais de inércia, e os elementos da diagonal de I são chamados momentos principais de inércia [MAR 70]. Na hierarquia de sistemas de referência adotada os eixos principais de inércia correspondem aos vetores base do SRO, e torna-se conveniente calcular os momentos de inércia no SRO. E, uma vez calculados os momentos de inércia no SRO, estes permanecem invariantes à descrição geométrica do objeto, ao longo do tempo.

Desta forma, o movimento rotacional de um corpo pode ser representado pela seguinte equação dinâmica [MAR 70]:

$$\frac{d\vec{L}_i}{dt} = \vec{T}_i \tag{4.23}$$

onde, para um corpo i, $\vec{L}_i$ são os vetores momento angulares do corpo e $\vec{T}_i$ são os vetores resultantes de torques externos atuando no corpo.

O vetor momento angular do corpo, $\vec{L}_i$, é definido como:

$$\vec{L}_i = I\vec{w}_i \tag{4.24}$$

onde I é a matriz tensor de inércia do corpo e $\vec{w}_i$ é o vetor velocidade angular do corpo que se deseja resolver para determinar o movimento rotacional. Para este caso, a matriz tensor de inércia, I, e o vetor velocidade angular, $\vec{w}_i$, devem ser representados no SRU.

Já os vetores resultantes de torques externos atuando no corpo, $\vec{T}_i$, são obtidos através de dois tipos de torques: torque puro atuando no corpo e torque resultante de forças externas atuando em um ponto de massa do corpo.

Para o segundo tipo de torque, se existirem vetores de forças externas do corpo, $\vec{F}_j$, atuando em pontos de massa do corpo, $\vec{P}_j$, relativos ao centro de massa do objeto, o vetor torque, $\vec{T}_j$, é definido pelo produto vetorial:

$$\vec{T}_j = \vec{P}_j \times \vec{F}_j \tag{4.25}$$

Segundo o princípio de sobreposição de Euler [BAR 88a], estas forças $\vec{F}_j$, atuando em pontos de massa do corpo, contribuem, também, para translação do corpo e devem ser adicionadas à rede de forças externas apresentada na equação (4.12).

De acordo com as equações (4.23) e (4.24), a matriz tensor de inércia, I, deve ser representada no SRU. Esta característica, como visto anteriormente, dificulta a resolução da dinâmica rotacional pela necessidade de calcular a cada intervalo de tempo a matriz tensor de inércia do objeto quando o corpo rotaciona. Contudo, isso pode ser evitado calculando a dinâmica rotacional no SRO.

A forma utilizada para resolver a dinâmica rotacional do corpo no SRO pode ser realizada através de um conjunto de equações diferenciais ordinárias aco-

pladas conhecidas como equações de Euler [MAR 70]:

$$
\begin{aligned}
I_x \frac{d\omega_x}{dt} + (I_z - I_y)\omega_y\omega_z &= T_x \\
I_y \frac{d\omega_y}{dt} + (I_x - I_z)\omega_x\omega_z &= T_y \\
I_z \frac{d\omega_z}{dt} + (I_y - I_x)\omega_x\omega_y &= T_z
\end{aligned}
\tag{4.26}
$$

onde, para um corpo i, $\vec{T}$ são os vetores de torques externos atuando no corpo, I_x, I_y e I_z são os momentos de inércia sobre o centro de massa e $\vec{\omega}$ são os vetores de velocidades angulares do corpo. Os x, y, e z são as direções dos eixos principais de inércia, ou seja, os vetores base do SRO.

Desta forma, é possível através da resolução do conjunto de equações (4.26) obter as velocidades angulares (vide figura 4.2) ($\vec{\omega_i}$) de um objeto ao longo do tempo.

4.3.2 Representação da Orientação do Corpo

A partir da informação das velocidades angulares ($\vec{\omega_i}$) de um objeto no SRO é possível representar sua orientação ao longo do tempo em relação a um sistema de referência inercial (SRU) através da matriz de rotação R_i, que pode ser construída de diversas maneiras.

Uma das formas de construção da matriz R_i é a equação de movimento apresentada por Barzel e Barr [BAR 88a]:

$$
\frac{dR_i}{dt} = \omega^* R_i \tag{4.27}
$$

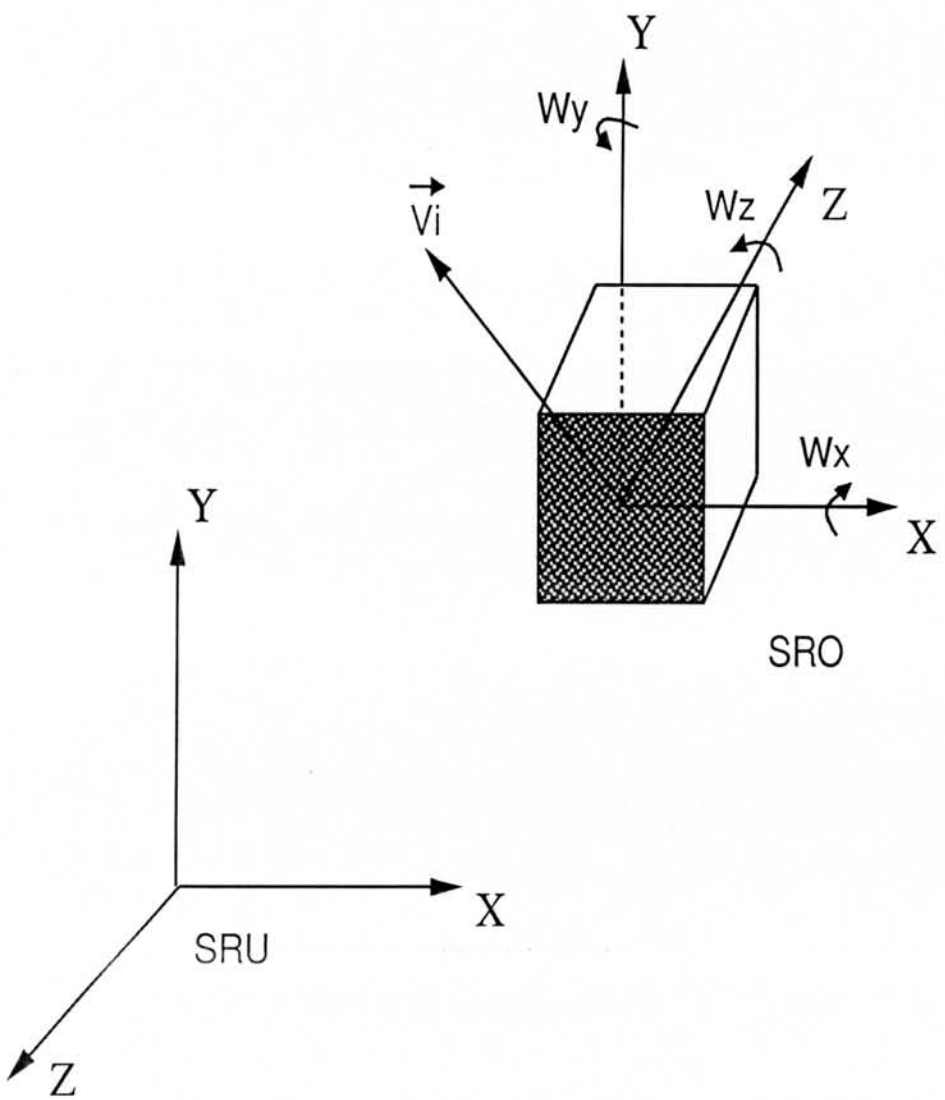

Figura 4.2: Velocidades linear e angular atuando sobre um corpo rígido

onde ω^* é o dual de $\vec{\omega}$, definido por:

$$\omega^* = \begin{bmatrix} 0 & \omega_z & -\omega_y \\ -\omega_z & 0 & \omega_x \\ \omega_y & -\omega_x & 0 \end{bmatrix}$$

De acordo com a equação (4.27), a variação de R_i em função do tempo, $\frac{dR_i}{dt}$, é definida como:

$$\frac{dR_i}{dt} = \begin{bmatrix} \frac{da_{11}}{dt} & \frac{da_{12}}{dt} & \frac{da_{13}}{dt} \\ \frac{da_{21}}{dt} & \frac{da_{22}}{dt} & \frac{da_{23}}{dt} \\ \frac{da_{31}}{dt} & \frac{da_{32}}{dt} & \frac{da_{33}}{dt} \end{bmatrix}$$

A partir disto, substituindo as matrizes ω^* e R_i na equação (4.27) obtém-se um conjunto de nove equações diferenciais ordinárias acopladas definidas por:

$$\begin{aligned}
\frac{da_{11}}{dt} &= a_{21}\omega_z - a_{31}\omega_y \\
\frac{da_{12}}{dt} &= a_{22}\omega_z - a_{32}\omega_y \\
\frac{da_{13}}{dt} &= a_{23}\omega_z - a_{33}\omega_y \\
\frac{da_{21}}{dt} &= -a_{11}\omega_z + a_{31}\omega_x \\
\frac{da_{22}}{dt} &= -a_{12}\omega_z + a_{32}\omega_x \\
\frac{da_{23}}{dt} &= -a_{13}\omega_z + a_{33}\omega_x \\
\frac{da_{31}}{dt} &= a_{11}\omega_y - a_{21}\omega_x \\
\frac{da_{32}}{dt} &= a_{12}\omega_y - a_{22}\omega_x \\
\frac{da_{33}}{dt} &= a_{13}\omega_y - a_{23}\omega_x
\end{aligned} \tag{4.28}$$

Como se necessita da informação da velocidade angular, $\vec{\omega}$, ao longo do tempo, no conjunto de nove equações (4.28) acima, deve-se acoplar, também, o conjunto das três equações de Euler (4.26), responsáveis pelo cálculo da velocidade

angular. Sendo assim, a partir destas doze equações diferenciais acopladas é possível obter de uma forma direta as nove componentes da matriz de rotação, R_i (vide equação 4.1), ao longo do tempo.

Assumindo que é possível associar um vetor com uma rotação infinitesimal, outra forma de construir a matriz de rotação, R_i, é associar as derivadas no tempo dos ângulos de rotação θ, α e β com as componentes dos vetores da velocidade angular ω_x, ω_y e ω_z, conforme apresentado em Marion [MAR 70] e Hahn [HAH 88]:

$$\begin{aligned} \frac{d\theta}{dt} &= \omega_x \\ \frac{d\alpha}{dt} &= \omega_y \\ \frac{d\beta}{dt} &= \omega_z \end{aligned} \tag{4.29}$$

Novamente, como se necessita da informação da velocidade angular, $\vec{\omega}$, ao longo do tempo, deve-se acoplar ao conjunto de três equações (4.29) acima, o conjunto das três equações de Euler (4.26) responsáveis pelo cálculo da velocidade angular. Portanto, a partir da resolução destas seis equações diferenciais ordinárias acopladas é possível obter os ângulos θ, α e β, ao longo do tempo. Uma vez obtidos estes ângulos, eles devem ser substituídos nos componentes da matriz R_i (vide equação 4.1).

5 TRATAMENTO DE COLISÕES DE CORPOS RíGIDOS

A filosofia de funcionamento dos métodos automáticos para controle de movimento está na capacidade desses métodos combinarem respostas físicas automáticas com sugestões de controle especificadas pelo usuário. Em um mundo virtual, respostas físicas automáticas são características intrínsecas do sistema de animação. Como exemplo de uma característica intrínseca à simulação dinâmica de corpos rígidos, pode-se citar a influência da força gravitacional durante o movimento dos objetos.

Outra característica intrínseca desejável, durante o movimento de corpos rígidos, é o tratamento automático de detecção de colisões e resposta. Esta característica vem do fato que o movimento dos objetos tem sentido físico e, sob esse ponto de vista, um corpo rígido não pode interpenetrar outro corpo rígido. Todavia, se houver uma colisão, ou seja, uma situação de provável interpenetração, é desejável, então, detectar os pontos de contatos dessa colisão e, posteriormente, responder fisicamente ao movimento resultante da dinâmica de impacto entre esses pontos.

Conforme visto anteriormente, existem diversas formas de abordar o método de controle de movimento de detecção e resposta (vide seção 3.2.2.1). Trabalhos realizados, dentro desse contexto, apresentam algoritmos para detecção de colisões os quais abordam objetos modelados por diferentes técnicas de modelagem geométrica e apresentam, também, abordagens para o tratamento de respostas à dinâmica de colisão as quais apresentam diferentes aproximações em relação às respostas que satisfazem a dinâmica newtoniana (vide seção 3.2.3).

Assim sendo, escolheu-se investigar aspectos gerais de algoritmos de detecção de colisões para corpos rígidos modelados geometricamente como uma grade de pontos conectados para formar polígonos (poliedros rígidos convexos) por ser o

tipo de modelagem geométrica adotada na simulação dinâmica de corpos rígidos. E, como abordagens para respostas da dinâmica de colisão, escolheu-se investigar abordagens que possam ser utilizadas em estratégias de controle por pré-processamento (vide seção 3.2.1) por serem facilmente associadas à formulação da dinâmica do movimento de corpos rígidos abordada neste trabalho (vide capítulo 4).

5.1 Detecção da Colisão

A detecção da colisão tem sido abordada extensivamente pelas áreas de robótica, animação e geometria computacional [BAR 90, BOY 79, HAH 88, LIN 92, MOO 88, UCH 83].

Algoritmos relacionados com esse tipo de detecção têm sido desenvolvidos em ambientes estáticos e dinâmicos (objetos em movimento). Para ambos os casos, a ênfase adotada é desenvolver algoritmos o mais eficientes possíveis, ou seja, com menores ordens de complexidade de tempo computacional. Algoritmos de detecção que resolvem problemas estáticos podem ser utilizados em simulações dinâmicas para resolverem seqüências de problemas estáticos ao longo do tempo. Porém, algoritmos de detecção projetados especificamente para resolver seqüências de problemas dinâmicos ao longo do tempo são mais eficientes por permitirem a implementação de estratégias que fazem uso de características de proximidade de outros objetos durante o movimento, eliminando-se, desta forma, a necessidade de testes estáticos exaustivos de detecção a cada intervalo de tempo.

Trabalhos realizados neste contexto têm modelado restrições de contato entre pares de poliedros rígidos convexos em seqüências de problemas em ambientes dinâmicos: Moore e Wilhelms [MOO 88] apresentam um método baseado no algoritmo de 'clipping' de Cyrus-Beck [ROG 88], o qual fornece uma alternativa simples e robusta com tempo de execução $O(n^2m^2)$ para n poliedros e m vértices por poliedro; Hahn [HAH 88] utiliza um método hierárquico envolvendo envelopes

tridimensionais para testes de intersecção com outros objetos com tempo de execução $O(m^2)$ para cada par de poliedros; e Lin e Canny [LIN 92] apresentam um método que utiliza a geometria do poliedro para estabelecer critérios de aplicabilidade local na detecção de colisões, onde através de um procedimento de pré-processamento reduz-se o número de testes necessários para determinação das características de proximidade entre pares de poliedros convexos, onde o tempo de execução tende a permanecer constante e não sofre degradação com o aumento do número de vértices na representação geométrica dos objetos.

Aqui, esses algoritmos não serão descritos em detalhes. A análise destes algoritmos e a implementação do algoritmo adotado por este trabalho (algoritmo apresentado por Moore e Wilhelms [MOO 88]) pode ser encontrado em Lemos [LEM 93]. Quer-se, entretanto, salientar um problema que deverá ser enfrentado independentemente da escolha de qualquer um deles.

Assim sendo, assumindo que o algoritmo utilizado para detecção de colisão considera que num determinado instante de tempo existe apenas um ponto de contato entre dois objetos, onde esse ponto de contato pode ser o ponto resultante da intersecção do vértice das arestas de um objeto com a face poligonal de outro objeto ou o resultante da intersecção da aresta de um objeto com a face poligonal de outro objeto [BOY 79], de acordo com a natureza de execução dos algoritmos para simulação dinâmica em intervalos de tempos discretos (vide figura 5.1) o instante de tempo da detecção de uma colisão pode ocorrer no instante de interpenetração entre esses dois objetos.

Quando esta situação de interpenetração ocorre (vide figura 5.1), deve-se retornar ao intervalo de tempo imediatamente antes da interpenetração ter ocorrido, reduzir o intervalo de tempo e repetir o procedimento de teste de interpenetração até encontrar o ponto em que os dois objetos se tocam mas não se interpenetram. Este processo, em busca do tempo exato do ponto de contato entre dois objetos, é similar à pesquisa binária. Assim sendo, encontrado o ponto de contato, deve-se

resolver, a colisão entre os objetos e incrementar o intervalo de tempo Δt para o processo de simulação (vide figura 3.2).

5.2 Dinâmica da Colisão de Corpos Rígidos

A dinâmica da colisão de corpos rígidos é responsável pelo cálculo das forças que surgiriam durante a colisão entre corpos rígidos para evitar que esses se interpenetrem.

Durante essa colisão dois tipos de forças podem ser encontradas: forças de contato contínuo; e forças de contato instantâneo.

As forças de contato contínuo surgem quando dois corpos estão em repouso. Essas forças são contínuas durante um intervalo de tempo diferente de zero.

Já as forças de contato instantâneo surgem quando dois corpos estão em movimento. E, como essas forças existem em um único instante de tempo, elas são descontínuas.

Técnicas tradicionais da Engenharia e da Física não são aplicáveis ao problema de calcular forças entre corpos que estão em repouso. Essas técnicas assumem que os corpos sob análise estão em equilíbrio [BAR 89].

No entanto, simulações gráficas dinâmicas, normalmente, envolvem sistemas de corpos rígidos que não estão em equilíbrio. Como visto anteriormente, existem três métodos para calcular as respostas aos efeitos da colisão entre corpos em movimento ou em repouso (vide seção 3.2.2.1): métodos de penalização, métodos analíticos e métodos de inserção de novas equações de restrições.

Como pretende-se investigar abordagens que possam ser utilizadas em estratégias de controle por pré-processamento (vide seção 3.2.1), serão apresentadas as características principais dos métodos de penalização e dos métodos analíticos:

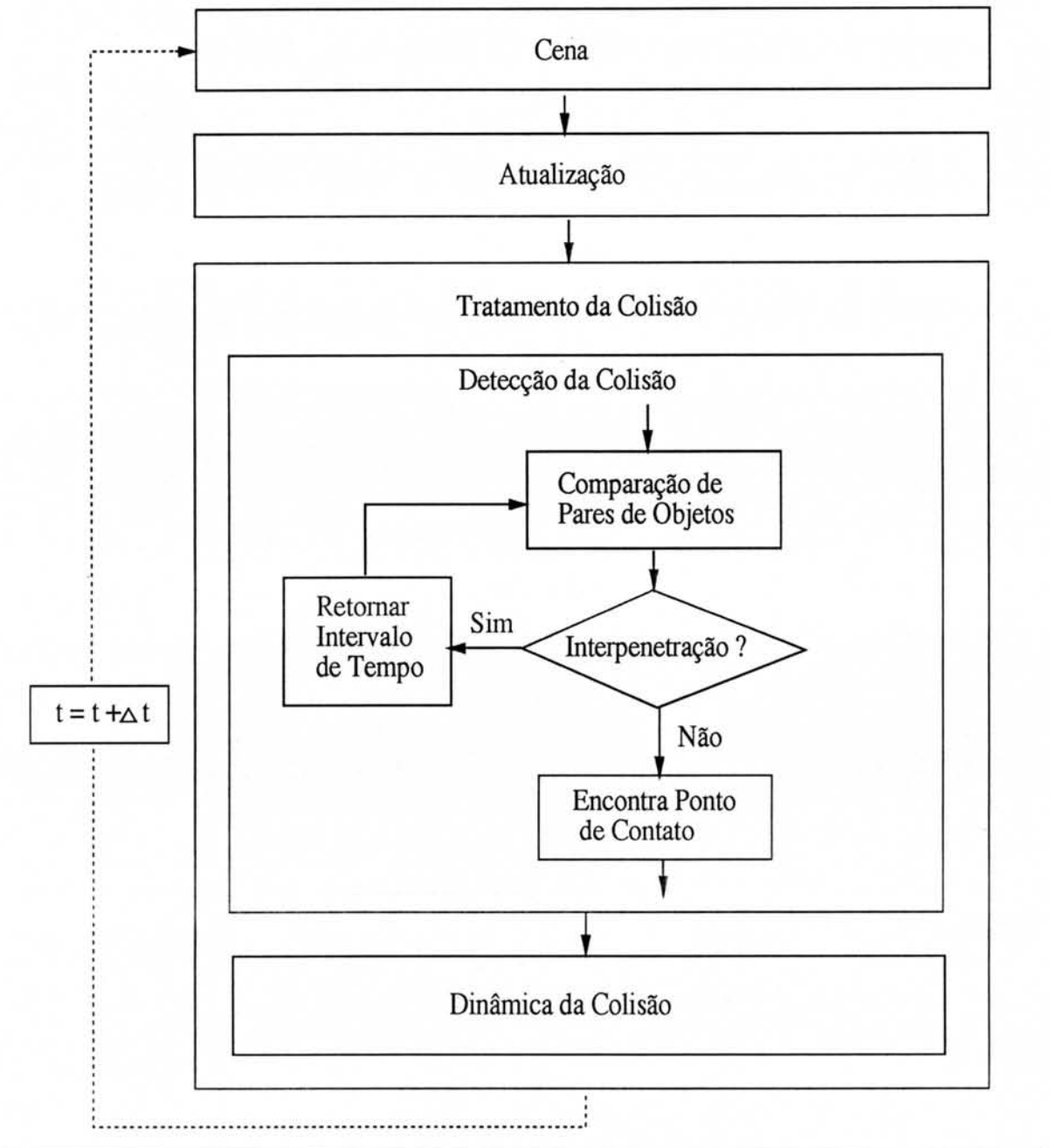

Figura 5.1: Fluxo de controle para detecção da colisão

- **Métodos de penalização:** estes métodos apresentam só resultados aproximados às respostas que satisfazem a dinâmica newtoniana, requerem ajustes para simulações sob condições diferentes e as equações diferenciais utilizadas podem tornar-se rígidas. Por outro lado, estes métodos são simples de serem implementados e são facilmente estendidos para corpos não-rígidos e

- **Métodos analíticos:** apresentam soluções exatas, pois são baseados diretamente nas leis da dinâmica newtoniana e as equações diferenciais requerem menos passos durante a simulação. No entanto, são mais complexos de serem derivados e implementados.

Métodos analíticos resolvem equações que descrevem como os momentos linear e angular são conservados durante a colisão. Esses métodos são rápidos para colisões fortes (contato instantâneo), porque a solução analítica necessita ser encontrada apenas uma vez [MOO 88]. Porém, para colisões leves (contato contínuo), a força da gravidade pode ser a causa para dois objetos interpenetrarem-se e a solução analítica teria que ser fornecida, novamente, de tempos em tempos. Por outro lado, um método de penalização, fornecendo uma força mola a um objeto em repouso, pode neutralizar a força da gravidade resultando um cálculo para força de contato mais rápido, mais estável e com um efeito realístico para animação. Desta forma, é interessante combinar ambos os métodos de penalização (resolvendo forças de contato contínuo) e analítico (resolvendo forças de contato instantâneo) para simular os efeitos da colisão entre corpos rígidos.

5.2.1 Dinâmica de Contato Contínuo

Durante o contato contínuo (em repouso) entre dois corpos rígidos, será adotado, então, o método de penalização para solucionar a dinâmica da colisão.

Uma vez encontrado o ponto de contato entre dois objetos (vide seção 5.2) em repouso, de acordo com o método de penalização, são inseridas forças mola (força proporcional à profundidade de interpenetração) temporárias entre os pontos de contato, que evitarão a interpenetração durante a colisão entre os dois objetos. Essas forças mola são fornecidas igualmente em direções opostas à colisão entre os dois objetos.

A lei mola, normalmente, é

$$k/d$$

onde k é uma constante controlando a rigidez da mola e d é a distância que separa os dois objetos. A lei mola pode ser expressa, também, através de outra forma funcional que tende ao infinito, tendo em vista que a separação d dos dois objetos aproxima-se de zero [MOO 88].

Portanto, de acordo com a estratégia de controle por pré-processamento após o cálculo das forças mola, estas são incluídas na rede de forças utilizadas na dinâmica do movimento de corpos rígidos permitindo, assim, a atualização do movimento dos objetos durante o processo de simulação (vide figura 3.2).

5.2.2 Dinâmica de Contato Instantâneo

Para dinâmica de contato instantâneo (em movimento) entre dois corpos rígidos, será adotado o método analítico apresentado por MacMillan [MAC 36] e proposto para simulações gráficas dinâmicas por Moore e Wilhelms [MOO 88].

Esse método analítico resolve um sistema de equações lineares descrevendo os momentos linear e angular de cada corpo e a natureza da força de contato instantâneo entre eles. Para tal, são utilizadas técnicas de soluções analíticas e numéricas envolvendo somente estados antes e depois da colisão. Sua solução for-

nece as velocidades linear e angular para cada corpo imediatamente após a colisão e estas podem ser, então, inseridas como uma ocorrência instantânea nas equações dinâmicas do movimento de corpos rígidos no tempo em que a colisão ocorreu.

Assumindo que o método calcula a dinâmica da colisão entre pares de objetos que colidem em um único ponto, considera-se para o problema da colisão os seguintes atributos físicos (vide figura 5.2): $\vec{v}_i$, vetor velocidade linear; $\vec{\omega}_i$, vetor velocidade angular; m, massa; $\vec{c}_i$, vetor posição do centro de massa; I, matriz tensor de inércia; $\vec{p}_i$, vetor centro de massa até o ponto de colisão e $\vec{R}$, vetor de impulso. Todos estes vetores devem ser representados em um mesmo sistema de referência.

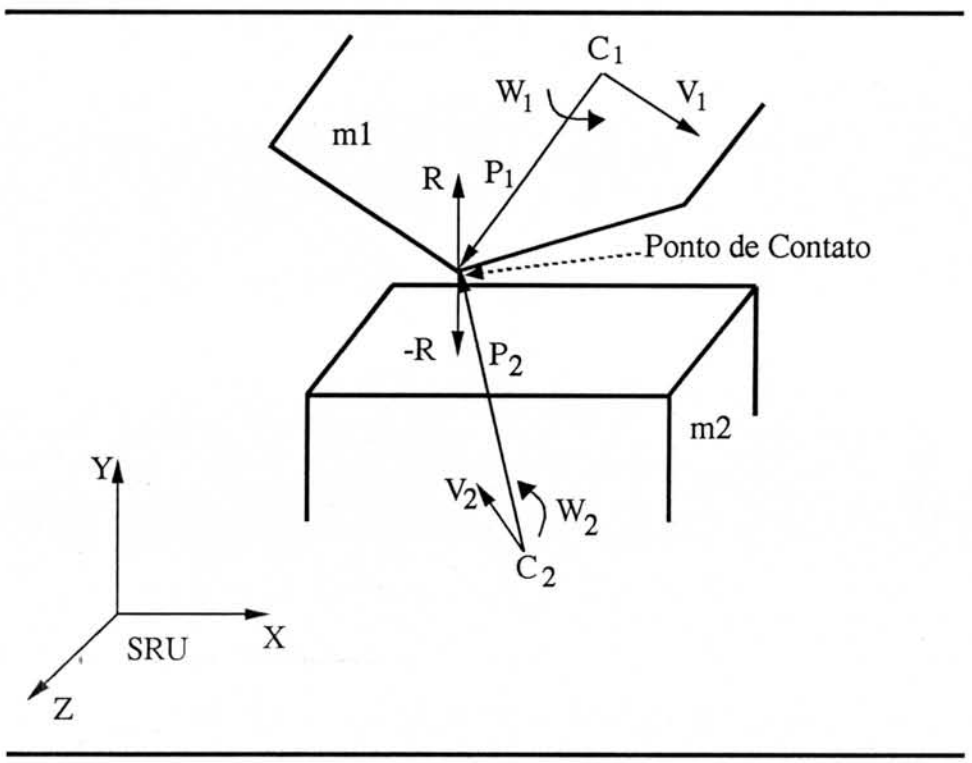

Figura 5.2: Problema da colisão entre pares de corpos rígidos

A solução deste problema de colisão, segundo Moore e Wilhelms [MOO 88], requer um sistema de referência de colisão. Assumindo que os vetores base desse sistema são $\vec{i}$, $\vec{j}$ e $\vec{k}$, $\vec{k}$ deverá ser perpendicular ao plano de colisão e $\vec{i}$ e $\vec{j}$ deverão formar o plano de colisão. Sendo assim, como esse plano de colisão torna-se algo arbitrário, ele é definido de acordo com o tipo de ocorrência de colisão:

- Vértice de um objeto colidindo com a face poligonal de outro objeto: a face poligonal define o plano de colisão,
- Aresta de um objeto colidindo com a aresta de outro objeto: as duas arestas definem o plano de colisão e
- Vértice de um objeto colidindo com o vértice de outro objeto: a direção do vetor base $\vec{k}$ passa ao longo da linha que une dois vértices.

Uma vez definido o problema de colisão, MacMillan [MAC 36] apresenta um conjunto de equações gerais para colisão entre dois corpos, as quais conservam os momentos linear e angular durante a colisão.

De acordo com o princípio da conservação do momento linear e desde que outras forças que estão atuando durante a colisão não são importantes, a troca do momento linear resultante do impacto pode ser expressa como:

$$\begin{aligned} m_1\bar{v_1} &= m_1\vec{v_1} + \vec{R} \\ m_2\bar{v_2} &= m_2\vec{v_2} - \vec{R} \end{aligned} \tag{5.1}$$

onde, respectivamente para cada objeto, $\bar{v_1}$ e $\bar{v_2}$ são os vetores velocidades lineares depois da colisão, $\vec{v_1}$ e $\vec{v_2}$ são os vetores velocidades lineares antes da colisão, m_1 e m_2 são as massas, e $\vec{R}$ é o vetor de impulso.

Da mesma forma, de acordo com o princípio da conservação do momento angular, a troca do momento angular resultante do impacto pode ser expressa como:

$$\begin{aligned} I_1\bar{\omega_1} &= I_1\vec{\omega_1} + \vec{p_1} \times \vec{R} \\ I_2\bar{\omega_2} &= I_2\vec{\omega_2} - \vec{p_2} \times \vec{R} \end{aligned} \tag{5.2}$$

onde, respectivamente, para cada objeto, $\bar{\omega_1}$ e $\bar{\omega_2}$ são os vetores velocidades angulares depois da colisão, $\vec{\omega_1}$ e $\vec{\omega_2}$ são os vetores velocidades angulares antes da colisão, I_1 e

I_2 são os momentos tensores de inércia, $\vec{p_1}$ e $\vec{p_2}$ são os vetores posição do centro de massa até o ponto de colisão, e $\vec{R}$ é o vetor de impulso.

Considerando o conjunto de equações (5.1) e (5.2) no espaço tridimensional, obtem-se, na verdade, um conjunto de doze equações lineares com quinze incógnitas. As quinze incógnitas são: os vetores velocidades lineares depois da colisão para cada objeto ($\bar{v_1}$,$\bar{v_2}$); os vetores velocidades angulares depois da colisão para cada objeto ($\bar{\omega_1}$,$\bar{\omega_2}$); e o vetor de impulso $\vec{R}$.

Como para solução das quinze incógnitas são necessárias quinze equações lineares, as três equações lineares restantes surgem das condições assumidas durante a colisão [MOO 88].

Assumindo que a elasticidade, ε, é zero (os dois objetos ao colidirem páram, um em relação ao outro, no ponto de colisão) e que as superfícies do objetos não possuem atrito (o impulso deve ser perpendicular ao plano de colisão), tem-se:

$$R\vec{i} = 0$$

e

$$R\vec{j} = 0$$

Logo, a diferença da velocidade no ponto de colisão, como visto acima para os dois objetos, será zero na direção de $\vec{k}$ e as últimas três equações podem ser definidas como:

$$(\bar{v_2} + \bar{\omega_2} \times \vec{p_2} - \bar{v_1} - \bar{\omega_1} \times \vec{p_1})\vec{k} = 0 \qquad (5.3)$$

Porém, se se desejar considerar colisões elásticas, a partir deste ponto do algoritmo, também é possível. Visto que a elasticidade efetiva da colisão pode ser tomada como a elasticidade mais baixa da colisão de duas superfícies, um novo impulso da colisão $\vec{R}_{efetivo}$ pode ser calculado como

$$\vec{R}_{efetivo} = (1 + \varepsilon_{efetiva})\vec{R}$$

E o novo impulso deve, então, ser substituído no conjunto das quinze equações lineares (5.1), (5.2) e (5.3) anteriores, para calcular os novos vetores $\bar{v}_i$ e $\bar{\omega}_i$. Desta forma, a partir da resolução do conjunto de quinze equações lineares (5.1), (5.2) e (5.3) é possível obter as novas velocidades linear e angular resultantes da dinâmica de contato instantâneo, permitindo, assim, a atualização do movimento dos objetos durante o processo de simulação (vide figura 3.2).

6 ANÁLISE E VALIDAÇÃO DE RESULTADOS

A animação e tratamento de colisões de corpos rígidos envolve diversas etapas, dentre as quais a fundamental é a simulação dinâmica do movimento. Devido às considerações adotadas para a modelagem com simulação física, o conjunto de objetos utilizados e suas interações com o ambiente são limitados.

A simulação do movimento de objetos é obtida com alto grau de realismo através do uso de equações dinâmicas baseadas diretamente nas leis da dinâmica newtoniana. O tratamento de eventuais colisões durante o movimento dos objetos adota um método de penalização, que apresenta soluções aproximadas durante o contato contínuo (em repouso) e um método analítico baseado diretamente nas leis da dinâmica newtoniana, que apresenta soluções exatas durante o contato instantâneo (em movimento).

No escopo de validação das técnicas abordadas ao longo deste capítulo, será apresentada a realização das etapas envolvidas na simulação da dinâmica do movimento de corpos rígidos e avaliações realizadas no decorrer destas etapas. A listagem do programa que faz a simulação do movimento encontra-se no anexo A-1.

6.1 Protótipo

O protótipo implementa as técnicas descritas para dinâmica do movimento de corpos rígidos e para detecção de colisões entre objetos. Utilizou-se a linguagem C padrão. de forma a facilitar a portabilidade do código. Realizaram-se testes em rede de estações de trabalho SUN.

O protótipo é composto de:

- **Primitivas de corpos rígidos:** coleção de corpos rígidos poliédricos convexos, modelados geometricamente por uma malha de pontos conectados para formar polígonos e é assumida distribuição de massa homogênea, tais como esferas, cubos, paralelepípedos, pirâmides, cilindros e cones. Cada tipo de corpo tem definidos os atributos necessários para simulação física (centro de massa, eixos principais, momentos de inércia sobre o centro de massa, massa total e elasticidade do material),

- **Pontos de controle:** pontos pertencentes ao corpo representados no SRO e

- **Forças externas:** forças externas atuando no corpo. As forças externas, atuando em um ponto de massa no corpo com localização diferente do centro de massa e as forças externas, atuando no centro de massa, são representadas no SRO. Cada força é um vetor que aponta para um ponto de controle.

As estratégias adotadas para resolução da dinâmica do movimento consideram as seguintes informações necessárias para o movimento:

- **Informações invariáveis:** informações tais como centro de massa, eixos principais, momentos de inércia sobre o centro de massa, massa total e elasticidade do material são consideradas constantes durante o movimento e

- **Informações variáveis:** informações tais como forças e torques externos, velocidades linear e angular e posições e orientações dos objetos são consideradas variáveis durante o movimento.

Sob o ponto de vista de controle do movimento dos objetos, pode-se representar as forças que atuam nos pontos de massa dos objetos de duas formas: através de forças constantes (forças atuando durante todo o intervalo de tempo

da simulação, por exemplo, a força gravitacional) e através de forças instantâneas (forças atuando somente no instante inicial de tempo, por exemplo uma força inicial informada para arremessar um objeto).

A interface do programa é realizada através de um arquivo de entrada que contém informações sobre o estado inicial do sistema dinâmico. O formato desse arquivo é:

$SIMULACAO

$TEMPO t_0 t_n

$NRO-OBJETOS n

$OBJETO

$NOME *objeto.obt*

$DENSIDADE d

$POSICAO-SRU p_x p_y p_z

$VEL-LINEAR-SRO v_x v_y v_z

$NRO-FORCA-EXT n

$FORCA-EXT-SRO f_x f_y f_z

$ORIENTACAO-SRO-SRU a_x a_y a_z

$VEL-ANG-SRO w_x w_y w_z

$NRO-TORQUE-EXT n

$PONTO-OBJ-SRO c_x c_y c_z

$PONTO-FORCA-SRO r_x r_y r_z

$ENDOBJ

$ENDSIMU

onde

$SIMULACAO indica o início do arquivo;

$TEMPO indica os tempos inicial (t_0) e final (t_n) para simulação;

$NRO-OBJETOS indica o número de objetos (n);

$OBJETO indica o início da descrição do estado inicial de um objeto;

$NOME indica o nome do objeto (*objeto.obt*);

$DENSIDADE indica a densidade (d) de massa do objeto;

$POSICAO-SRU indica a posição (p_x, p_y, p_z) do centro de massa do objeto no SRU;

$VEL-LINEAR-SRO indica a velocidade linear (v_x, v_y, v_z) do objeto no SRU;

$NRO-FORCA-EXT indica o número de forças externas (n) atuando no objeto;

$FORCA-EXT-SRO indica a força externa instantânea (f_x, f_y, f_z) atuando no centro de massa do objeto no SRU;

$ORIENTACAO-SRO-SRU indica os ângulos de orientação (a_x, a_y, a_z) do objeto no SRO em relação ao SRU;

$VEL-ANG-SRO indica a velocidade angular (w_x, w_y, w_z) do objeto no SRO;

$NRO-TORQUE-EXT indica o número de torques externos (n) atuando no objeto;

$PONTO-OBJ-SRO indica o ponto de controle (c_x, c_y, c_z) do objeto em relação ao centro de massa no SRO;

$PONTO-FORCA-SRO indica o ponto (r_x, r_y, r_z) no SRO que formará juntamente com o ponto de controle (c_x, c_y, c_z) o vetor força externa instantânea atuando no objeto no SRO; assume-se o sentido do vetor força como ponto (r_x, r_y, r_z) $\longrightarrow$ ponto (c_x, c_y, c_z);

$ENDOBJ indica o final da descrição do estado inicial de um objeto e

$ENDSIMU indica o fim do arquivo.

Sendo assim, a simulação é realizada a partir da descrição do estado inicial do sistema dinâmico, conforme o formato do arquivo de entrada visto acima. O estado dinâmico em um tempo Δt posterior é obtido através da resolução das equações dinâmicas de movimento, onde Δt corresponde ao incremento de tempo entre os quadros de animação (vide figura 3.2). Portanto, cada estado subseqüente é obtido a partir do estado anterior.

Durante a etapa da resolução da estratégia geral para representação da dinâmica do movimento de corpos rígidos (vide seção 4.1) observaram-se os problemas descritos a seguir.

Na resolução da matriz de rotação Ri (vide equação 4.1) utilizou-se, inicialmente, a estratégia apresentada por Barzel e Barr [BAR 88a] para representação da orientação do corpo (vide seção 4.3.2). Observou-se que a integração independente das nove componentes de R_i numericamente obtidas, pela formação dos conjuntos de equações diferenciais ordinárias (4.26) e (4.28); ao longo do tempo, quando aplicadas aos pontos do objeto (para resultar na rotação destes pontos em relação ao centro de massa) apresenta problema, pois não é mantida a descrição geométrica original do objeto ao longo do tempo (provoca deformações na forma do objeto devido a imprecisões na definição da matriz de rotação). Através da utilização de um método numérico para resolução de equações diferenciais ordinárias com controle de passo de tempo adaptativo (Runge-Kutta de 5^a ordem [PRE 88]), observou-se que, de acordo com o tipo de matriz de rotação R_i adotado, o conjunto de equações (4.28) torna-se um sistema rígido (vide seção 3.1.4) ao longo do tempo.

Desta forma, utilizou-se a estratégia apresentada em Marion [MAR 70] e Hahn [HAH 88] (vide seção 4.3.2) para representação da orientação do corpo, onde o número de integrações numéricas (sujeitas a erro) foi reduzido e não apresenta os problemas de deformações dos objetos detectados com o método anterior.

Esta etapa de resolução da dinâmica de movimento forma a base para a construção de uma próxima etapa para o tratamento de colisões de corpos rígidos (vide capítulo 5).

As saídas do programa são armazenadas em arquivos, os quais contêm as posições dos objetos no SRU e seus ângulos de orientação em relação ao SRU.

Os arquivos de saída são compatíveis com os arquivos utilizados na descrição do roteiro para movimento de objetos definidos nos sistemas de animação PREVIEW [SCH 92] e ANIMAKER [SIL 92]. Desta forma, os resultados gerados pelo programa motor CLIC podem ser visualizados através destes sistemas de animação.

6.2 Resultado da simulação dinâmica do movimento de um objeto

De acordo com o objeto apresentado na figura 6.2, um paralelepípedo de lados $0,10m$ em X, $0,18m$ em Y e $0,02m$ em Z de massa homogênea com lados paralelos aos eixos do SRU e centro de massa na origem do SRU inicialmente parado linear e angularmente com duas forças externas instantâneas agindo sobre ele, uma agindo apenas translacionalmente, $\vec{F}_1$, e outra gerando também torque, $\vec{F}_2$, e sob a ação da gravidade (força externa constante) é representada na figura 6.1 a descrição do estado dinâmico inicial para sua simulação. Os resultados de 12 segundos de animação [1] são apresentados na figura 6.3. O comportamento realístico deste paralelepípedo deve-se ao fato da utilização de equações dinâmicas de movimento obtidas da mecânica newtoniana.

[1] 14 quadros gerados durante 12 segundos de animação, através do sistema de animação ANIMAKER [SIL 92].

```
$SIMULACAO
$TEMPO 0.0 12.0
$NRO_OBJETOS 1
$OBJETO
$NOME paral.obt
$DENSIDADE 1000
$POSICAO_SRU 0.0 0.0 0.0
$VEL_LIN_SRO 0.0 0.0 0.0
$NRO_FORCA_EXT 1
$FORCA_EXT_SRO -1000.0 -800.0 0.0
$ORIENTACAO_SRO_SRU 0.0 0.0 0.0
$VEL_ANG_SRO 0.0 0.0 0.0
$NRO_TORQUE_EXT 1
$PONTO_OBJ_SRO -0.05 0.09 0.0
$PONTO_FORCA_SRO -5.0 0.09 0.0
$ENDOBJ
$ENDSIMU
```

Figura 6.1: Estado do sistema dinâmico inicial para simulação

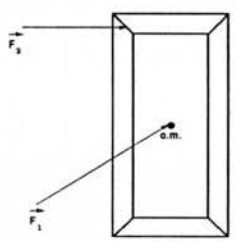

Figura 6.2: Paralelepípedo com direção e sentido das forças externas instantâneas $\vec{F}_1$ e $\vec{F}_2$ atuando sobre o objeto

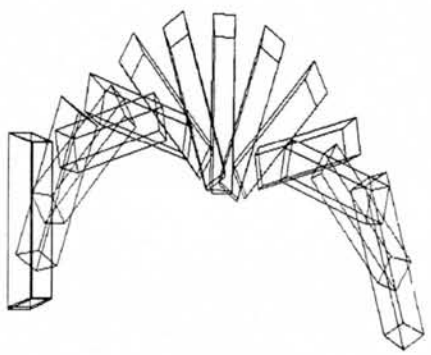

Figura 6.3: Arremesso de um paralelepípedo

6.3 Resultados da detecção de colisões entre pares de objetos poliédricos

São apresentados a seguir os resultados correspondentes às três etapas do algoritmo para detecção de colisões apresentado por Lemos [LEM 93].

6.3.1 Resultados do teste de presença de vértices do objeto B dentro das faces do objeto A

6.3.1.1 Exemplo 1

A partir do estado inicial do sistema dinâmico representado na figura 6.4, é detectada, no instante de tempo igual à 1.2 segundos, a colisão de um vértice contra um volume poliédrico. Os resultados de 1.2 segundos de animação [2] são apresentados nas figuras 6.5 e 6.6. Os resultados numéricos correspondentes ao movimento dos objetos (paral1.obt e paral2.obt) encontram-se respectivamente nas figuras 6.7 e 6.8 (translação e orientação do objeto paral1.obt) e 6.9 e 6.10 (translação e orientação do objeto paral2.obt).

6.3.1.2 Exemplo 2

A partir do estado inicial do sistema dinâmico representado na figura 6.11, é detectada, no instante de tempo igual à 1.633 segundos, a colisão de um vértice contra um volume poliédrico. Os resultados de 1.633 segundos de animação [3] são apresentados nas figuras 6.12 e 6.13 .

---

[2] 23 quadros gerados durante 1.2 segundos de animação, através do sistema de animação ANIMAKER [SIL 92]

[3] 25 quadros gerados durante 1.633 segundos de animação, através do sistema de animação ANIMAKER [SIL 92]

```
$SIMULACAO
$TEMPO 0.0 5.0
$NRO_OBJETOS 2
$OBJETO 0
$NOME paral1.obt
$DENSIDADE 0.01
$POSICAO_SRU 0.0 0.0 0.0
$VEL_LINEAR_SRO 0.0 0.0 0.0
$NRO_FORCA_EXT 1
$FORCA_EXT_SRO -10000.0 0.0 0.0
$ORIENT_SRO_SRU 0.0 0.0 0.0
$VEL_ANG_SRO 0.0 0.0 0.0
$NRO_TORQUE_EXT 1
$PONTO_OBJ_SRO 0.0 0.0 0.0
$PONTO_FORCA_SRO 0.0 0.0 0.0
$ENDOBJ
$OBJETO 0
$NOME paral2.obt
$DENSIDADE 0.01
$POSICAO_SRU -350.0 0.0 0.0
$VEL_LINEAR_SRO 0.0 0.0 0.0
$NRO_FORCA_EXT 1
$FORCA_EXT_SRO 0.0 0.0 0.0
$ORIENT_SRO_SRU 30.0 0.0 30.0
$VEL_ANG_SRO 0.0 0.0 0.0
$NRO_TORQUE_EXT 1
$PONTO_OBJ_SRO 0.0 0.0 0.0
$PONTO_FORCA_SRO 0.0 0.0 0.0
$ENDOBJ
$ENDSIMU
```

Figura 6.4: Exemplo 1 - Estado do sistema dinâmico inicial

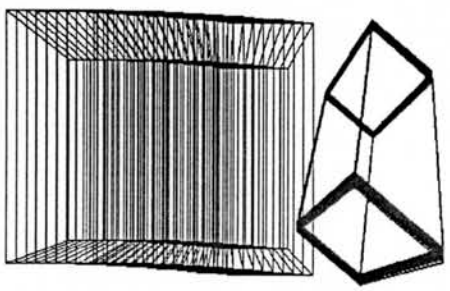

Figura 6.5: Exemplo 1 - Vista frontal da colisão

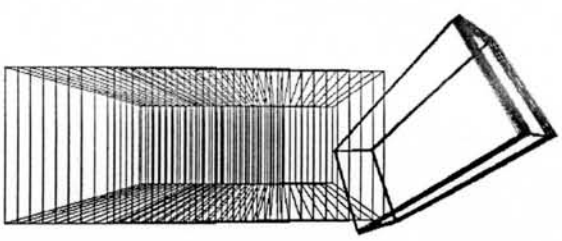

Figura 6.6: Exemplo 1 - Vista superior da colisão

tempo	vel_lin x	vel_lin y	vel_lin z	trans x	trans y	trans z
0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
0.033333	-173.611111	-0.326667	0.000000	-5.787037	-0.005444	0.000000
0.066667	-173.611111	-0.653333	0.000000	-11.574074	-0.021778	0.000000
0.100000	-173.611111	-0.980000	0.000000	-17.361111	-0.049000	0.000000
0.133333	-173.611111	-1.306667	0.000000	-23.148148	-0.087111	0.000000
0.166667	-173.611111	-1.633333	0.000000	-28.935185	-0.136111	0.000000
0.200000	-173.611111	-1.960000	0.000000	-34.722222	-0.196000	0.000000
0.233333	-173.611111	-2.286667	0.000000	-40.509259	-0.266778	0.000000
0.266667	-173.611111	-2.613333	0.000000	-46.296296	-0.348444	0.000000
0.300000	-173.611111	-2.940000	0.000000	-52.083333	-0.441000	0.000000
0.333333	-173.611111	-3.266667	0.000000	-57.870370	-0.544444	0.000000
0.366667	-173.611111	-3.593333	0.000000	-63.657407	-0.658778	0.000000
0.400000	-173.611111	-3.920000	0.000000	-69.444444	-0.784000	0.000000
0.433333	-173.611111	-4.246667	0.000000	-75.231481	-0.920111	0.000000
0.466667	-173.611111	-4.573333	0.000000	-81.018519	-1.067111	0.000000
0.500000	-173.611111	-4.900000	0.000000	-86.805556	-1.225000	0.000000
0.533333	-173.611111	-5.226667	0.000000	-92.592593	-1.393778	0.000000
0.566667	-173.611111	-5.553333	0.000000	-98.379630	-1.573444	0.000000
0.600000	-173.611111	-5.880000	0.000000	-104.166667	-1.764000	0.000000
0.633333	-173.611111	-6.206667	0.000000	-109.953704	-1.965444	0.000000
0.666667	-173.611111	-6.533333	0.000000	-115.740741	-2.177778	0.000000
0.700000	-173.611111	-6.860000	0.000000	-121.527778	-2.401000	0.000000
0.733333	-173.611111	-7.186667	0.000000	-127.314815	-2.635111	0.000000
0.766667	-173.611111	-7.513333	0.000000	-133.101852	-2.880111	0.000000
0.800000	-173.611111	-7.840000	0.000000	-138.888889	-3.136000	0.000000
0.833333	-173.611111	-8.166667	0.000000	-144.675926	-3.402778	0.000000
0.866667	-173.611111	-8.493333	0.000000	-150.462963	-3.680444	0.000000
0.900000	-173.611111	-8.820000	0.000000	-156.250000	-3.969000	0.000000
0.933333	-173.611111	-9.146667	0.000000	-162.037037	-4.268444	0.000000
0.966667	-173.611111	-9.473333	0.000000	-167.824074	-4.578778	0.000000
1.000000	-173.611111	-9.800000	0.000000	-173.611111	-4.900000	0.000000
1.033333	-173.611111	-10.126667	0.000000	-179.398148	-5.232111	0.000000
1.066667	-173.611111	-10.453333	0.000000	-185.185185	-5.575111	0.000000
1.100000	-173.611111	-10.780000	0.000000	-190.972222	-5.929000	0.000000
1.133333	-173.611111	-11.106667	0.000000	-196.759259	-6.293778	0.000000
1.166667	-173.611111	-11.433333	0.000000	-202.546296	-6.669444	0.000000
1.200000	-173.611111	-11.760000	0.000000	-208.333333	-7.056000	0.000000

Figura 6.7: Exemplo 1 - Resultados numéricos para translação do objeto para11.obt

tempo	vel_ang x	vel_ang y	vel_ang z	orien x	orien y	orien z
0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
0.033333	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
0.066667	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
0.100000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
0.133333	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
0.166667	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
0.200000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
0.233333	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
0.266667	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
0.300000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
0.333333	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
0.366667	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
0.400000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
0.433333	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
0.466667	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
0.500000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
0.533333	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
0.566667	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
0.600000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
0.633333	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
0.666667	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
0.700000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
0.733333	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
0.766667	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
0.800000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
0.833333	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
0.866667	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
0.900000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
0.933333	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
0.966667	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
1.033333	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
1.066667	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
1.100000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
1.133333	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
1.166667	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
1.200000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000

Figura 6.8: Exemplo 1 - Resultados numéricos para orientação do objeto paral1.obt

tempo	vel_lin x	vel_lin y	vel_lin z	trans x	trans y	trans z
0.000000	0.000000	0.000000	0.000000	-350.000000	0.000000	0.000000
0.033333	0.000000	-0.326667	0.000000	-350.000000	-0.005444	0.000000
0.066667	0.000000	-0.653333	0.000000	-350.000000	-0.021778	0.000000
0.100000	0.000000	-0.980000	0.000000	-350.000000	-0.049000	0.000000
0.133333	0.000000	-1.306667	0.000000	-350.000000	-0.087111	0.000000
0.166667	0.000000	-1.633333	0.000000	-350.000000	-0.136111	0.000000
0.200000	0.000000	-1.960000	0.000000	-350.000000	-0.196000	0.000000
0.233333	0.000000	-2.286667	0.000000	-350.000000	-0.266778	0.000000
0.266667	0.000000	-2.613333	0.000000	-350.000000	-0.348444	0.000000
0.300000	0.000000	-2.940000	0.000000	-350.000000	-0.441000	0.000000
0.333333	0.000000	-3.266667	0.000000	-350.000000	-0.544444	0.000000
0.366667	0.000000	-3.593333	0.000000	-350.000000	-0.658778	0.000000
0.400000	0.000000	-3.920000	0.000000	-350.000000	-0.784000	0.000000
0.433333	0.000000	-4.246667	0.000000	-350.000000	-0.920111	0.000000
0.466667	0.000000	-4.573333	0.000000	-350.000000	-1.067111	0.000000
0.500000	0.000000	-4.900000	0.000000	-350.000000	-1.225000	0.000000
0.533333	0.000000	-5.226667	0.000000	-350.000000	-1.393778	0.000000
0.566667	0.000000	-5.553333	0.000000	-350.000000	-1.573444	0.000000
0.600000	0.000000	-5.880000	0.000000	-350.000000	-1.764000	0.000000
0.633333	0.000000	-6.206667	0.000000	-350.000000	-1.965444	0.000000
0.666667	0.000000	-6.533333	0.000000	-350.000000	-2.177778	0.000000
0.700000	0.000000	-6.860000	0.000000	-350.000000	-2.401000	0.000000
0.733333	0.000000	-7.186667	0.000000	-350.000000	-2.635111	0.000000
0.766667	0.000000	-7.513333	0.000000	-350.000000	-2.880111	0.000000
0.800000	0.000000	-7.840000	0.000000	-350.000000	-3.136000	0.000000
0.833333	0.000000	-8.166667	0.000000	-350.000000	-3.402778	0.000000
0.866667	0.000000	-8.493333	0.000000	-350.000000	-3.680444	0.000000
0.900000	0.000000	-8.820000	0.000000	-350.000000	-3.969000	0.000000
0.933333	0.000000	-9.146667	0.000000	-350.000000	-4.268444	0.000000
0.966667	0.000000	-9.473333	0.000000	-350.000000	-4.578778	0.000000
1.000000	0.000000	-9.800000	0.000000	-350.000000	-4.900000	0.000000
1.033333	0.000000	-10.126667	0.000000	-350.000000	-5.232111	0.000000
1.066667	0.000000	-10.453333	0.000000	-350.000000	-5.575111	0.000000
1.100000	0.000000	-10.780000	0.000000	-350.000000	-5.929000	0.000000
1.133333	0.000000	-11.106667	0.000000	-350.000000	-6.293778	0.000000
1.166667	0.000000	-11.433333	0.000000	-350.000000	-6.669444	0.000000
1.200000	0.000000	-11.760000	0.000000	-350.000000	-7.056000	0.000000

Figura 6.9: Exemplo 1 - Resultados numéricos para translação do objeto paral2.obt

tempo	vel_ang x	vel_ang y	vel_ang z	orien x	orien y	orien z
0.000000	0.000000	0.000000	0.000000	30.000000	0.000000	30.000000
0.033333	0.000000	0.000000	0.000000	30.000000	0.000000	30.000000
0.066667	0.000000	0.000000	0.000000	30.000000	0.000000	30.000000
0.100000	0.000000	0.000000	0.000000	30.000000	0.000000	30.000000
0.133333	0.000000	0.000000	0.000000	30.000000	0.000000	30.000000
0.166667	0.000000	0.000000	0.000000	30.000000	0.000000	30.000000
0.200000	0.000000	0.000000	0.000000	30.000000	0.000000	30.000000
0.233333	0.000000	0.000000	0.000000	30.000000	0.000000	30.000000
0.266667	0.000000	0.000000	0.000000	30.000000	0.000000	30.000000
0.300000	0.000000	0.000000	0.000000	30.000000	0.000000	30.000000
0.333333	0.000000	0.000000	0.000000	30.000000	0.000000	30.000000
0.366667	0.000000	0.000000	0.000000	30.000000	0.000000	30.000000
0.400000	0.000000	0.000000	0.000000	30.000000	0.000000	30.000000
0.433333	0.000000	0.000000	0.000000	30.000000	0.000000	30.000000
0.466667	0.000000	0.000000	0.000000	30.000000	0.000000	30.000000
0.500000	0.000000	0.000000	0.000000	30.000000	0.000000	30.000000
0.533333	0.000000	0.000000	0.000000	30.000000	0.000000	30.000000
0.566667	0.000000	0.000000	0.000000	30.000000	0.000000	30.000000
0.600000	0.000000	0.000000	0.000000	30.000000	0.000000	30.000000
0.633333	0.000000	0.000000	0.000000	30.000000	0.000000	30.000000
0.666667	0.000000	0.000000	0.000000	30.000000	0.000000	30.000000
0.700000	0.000000	0.000000	0.000000	30.000000	0.000000	30.000000
0.733333	0.000000	0.000000	0.000000	30.000000	0.000000	30.000000
0.766667	0.000000	0.000000	0.000000	30.000000	0.000000	30.000000
0.800000	0.000000	0.000000	0.000000	30.000000	0.000000	30.000000
0.833333	0.000000	0.000000	0.000000	30.000000	0.000000	30.000000
0.866667	0.000000	0.000000	0.000000	30.000000	0.000000	30.000000
0.900000	0.000000	0.000000	0.000000	30.000000	0.000000	30.000000
0.933333	0.000000	0.000000	0.000000	30.000000	0.000000	30.000000
0.966667	0.000000	0.000000	0.000000	30.000000	0.000000	30.000000
1.000000	0.000000	0.000000	0.000000	30.000000	0.000000	30.000000
1.033333	0.000000	0.000000	0.000000	30.000000	0.000000	30.000000
1.066667	0.000000	0.000000	0.000000	30.000000	0.000000	30.000000
1.100000	0.000000	0.000000	0.000000	30.000000	0.000000	30.000000
1.133333	0.000000	0.000000	0.000000	30.000000	0.000000	30.000000
1.166667	0.000000	0.000000	0.000000	30.000000	0.000000	30.000000
1.200000	0.000000	0.000000	0.000000	30.000000	0.000000	30.000000

Figura 6.10: Exemplo 1 - Resultados numéricos para orientação do objeto paral2.obt

```
$SIMULACAO
$TEMPO 0.0 5.0
$NRO_OBJETOS 2
$OBJETO 0
$NOME paral1.obt
$DENSIDADE 0.01
$POSICAO_SRU 0.0 0.0 0.0
$VEL_LINEAR_SRO 0.0 0.0 0.0
$NRO_FORCA_EXT 1
$FORCA_EXT_SRO -10000.0 -20.0 0.0
$ORIENT_SRO_SRU 0.0 0.0 0.0
$VEL_ANG_SRO 0.0 0.0 0.0
$NRO_TORQUE_EXT 1
$PONTO_OBJ_SRO 0.0 0.0 0.0
$PONTO_FORCA_SRO 0.0 0.0 0.0
$ENDOBJ
$OBJETO 0
$NOME paral2.obt
$DENSIDADE 0.01
$POSICAO_SRU -400.0 0.0 0.0
$VEL_LINEAR_SRO 0.0 0.0 0.0
$NRO_FORCA_EXT 1
$FORCA_EXT_SRO 0.0 0.0 0.0
$ORIENT_SRO_SRU 0.0 -30.0 0.0
$VEL_ANG_SRO 0.0 0.0 0.0
$NRO_TORQUE_EXT 1
$PONTO_OBJ_SRO 0.0 0.0 0.0
$PONTO_FORCA_SRO 0.0 0.0 0.0
$ENDOBJ
$ENDSIMU
```

Figura 6.11: Exemplo 2 - Estado do sistema dinâmico inicial

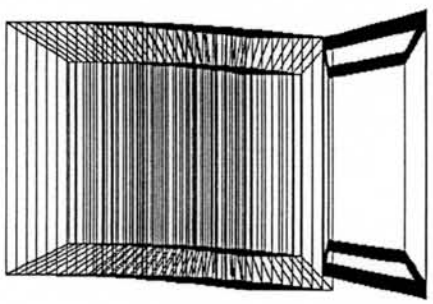

Figura 6.12: Exemplo 2 - Vista frontal da colisão

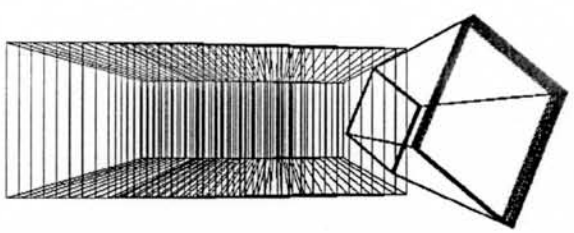

Figura 6.13: Exemplo 2 - Vista superior da colisão

6.3.1.3 Exemplo 3

A partir do estado inicial do sistema dinâmico representado na figura 6.14, é detectada, no instante de tempo igual à 1.66 segundos, a colisão de um vértice contra um volume poliédrico. Os resultados de 1.66 segundos de animação [4] são apresentados nas figuras 6.15 e 6.16 .

```
$SIMULACAO
$TEMPO 0.0 5.0
$NRO_OBJETOS 2
$OBJETO 0
$NOME paral1.obt
$DENSIDADE 0.01
$POSICAO_SRU 0.0 0.0 0.0
$VEL_LINEAR_SRO 0.0 0.0 0.0
$NRO_FORCA_EXT 1
$FORCA_EXT_SRO -10000.0 -20.0 0.0
$ORIENT_SRO_SRU 30.0 0.0 30.0
$VEL_ANG_SRO 0.0 0.0 0.0
$NRO_TORQUE_EXT 1
$PONTO_OBJ_SRO 0.0 0.0 0.0
$PONTO_FORCA_SRO 0.0 0.0 0.0
$ENDOBJ
$OBJETO 0
$NOME paral2.obt
$DENSIDADE 0.01
$POSICAO_SRU -400.0 0.0 0.0
$VEL_LINEAR_SRO 0.0 0.0 0.0
$NRO_FORCA_EXT 1
$FORCA_EXT_SRO 0.0 0.0 0.0
$ORIENT_SRO_SRU 0.0 -30.0 0.0
$VEL_ANG_SRO 0.0 0.0 0.0
$NRO_TORQUE_EXT 1
$PONTO_OBJ_SRO 0.0 0.0 0.0
$PONTO_FORCA_SRO 0.0 0.0 0.0
$ENDOBJ
$ENDSIMU
```

Figura 6.14: Exemplo 3 - Estado do sistema dinâmico inicial

[4] 26 quadros gerados durante 1.66 segundos de animação, através do sistema de animação ANIMAKER [SIL 92]

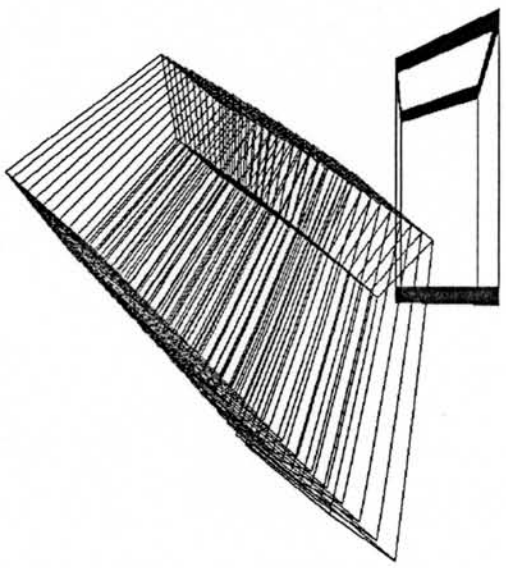

Figura 6.15: Exemplo 3 - Vista frontal da colisão

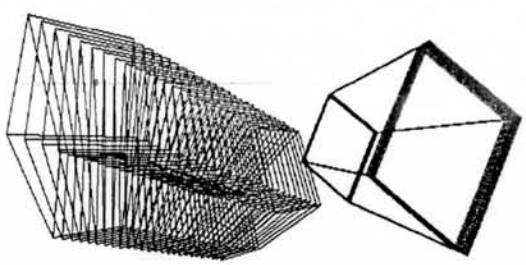

Figura 6.16: Exemplo 3 - Vista superior da colisão

6.3.2 Resultados do teste de presença de arestas do objeto B dentro das faces do objeto A

6.3.2.1 Exemplo 4

A partir do estado inicial do sistema dinâmico representado na figura 6.17, é detectada, no instante de tempo igual à 1.8 segundos, a colisão de um vértice contra um volume poliédrico. Os resultados de 1.8 segundos de animação [5] são apresentados nas figuras 6.18 e 6.19 .

6.3.3 Resultados do teste de presença dos centróides das faces do objeto B dentro das faces do objeto A

6.3.3.1 Exemplo 5

A partir do estado inicial do sistema dinâmico representado na figura 6.20, é detectada, no instante de tempo igual à 1.8 segundos, a colisão de um centróide de uma face contra um volume poliédrico. Os resultados de 1.8 segundos de animação [6] são apresentados nas figuras 6.21 e 6.22.

[5] 27 quadros gerados durante 1.8 segundos de animação, através do sistema de animação ANIMAKER [SIL 92]

[6] 27 quadros gerados durante 1.8 segundos de animação, através do sistema de animação ANIMAKER [SIL 92]

```
$SIMULACAO
$TEMPO 0.0 5.0
$NRO_OBJETOS 2
$OBJETO 0
$NOME paral1.obt
$DENSIDADE 0.01
$POSICAO_SRU 0.0 0.0 0.0
$VEL_LINEAR_SRO 0.0 0.0 0.0
$NRO_FORCA_EXT 1
$FORCA_EXT_SRO -10000.0 -20.0 0.0
$ORIENT_SRO_SRU 90.0 0.0 0.0
$VEL_ANG_SRO 0.0 0.0 0.0
$NRO_TORQUE_EXT 1
$PONTO_OBJ_SRO 0.0 0.0 0.0
$PONTO_FORCA_SRO 0.0 0.0 0.0
$ENDOBJ
$OBJETO 0
$NOME paral2.obt
$DENSIDADE 0.01
$POSICAO_SRU -400.0 0.0 0.0
$VEL_LINEAR_SRO 0.0 0.0 0.0
$NRO_FORCA_EXT 1
$FORCA_EXT_SRO 0.0 0.0 0.0
$ORIENT_SRO_SRU 0.0 0.0 0.0
$VEL_ANG_SRO 0.0 0.0 0.0
$NRO_TORQUE_EXT 1
$PONTO_OBJ_SRO 0.0 0.0 0.0
$PONTO_FORCA_SRO 0.0 0.0 0.0
$ENDOBJ
$ENDSIMU
```

Figura 6.17: Exemplo 4 - Estado do sistema dinâmico inicial

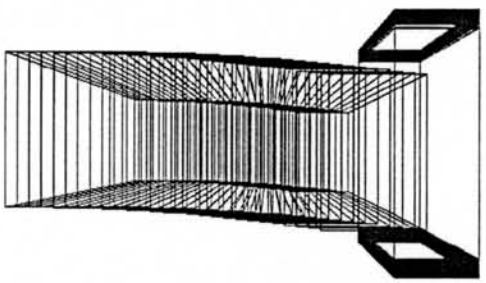

Figura 6.18: Exemplo 4 - Vista frontal da colisão

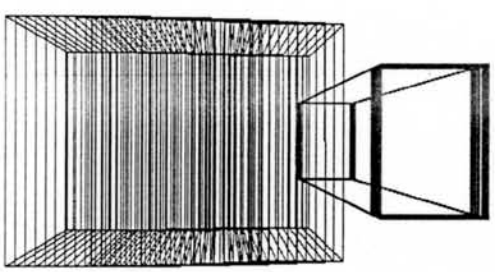

Figura 6.19: Exemplo 4 - Vista superior da colisão

```
$SIMULACAO
$TEMPO 0.0 5.0
$NRO_OBJETOS 2
$OBJETO 0
$NOME paral1.obt
$DENSIDADE 0.01
$POSICAO_SRU 0.0 0.0 0.0
$VEL_LINEAR_SRO 0.0 0.0 0.0
$NRO_FORCA_EXT 1
$FORCA_EXT_SRO -10000.0 0.0 0.0
$ORIENT_SRO_SRU 0.0 0.0 0.0
$VEL_ANG_SRO 0.0 0.0 0.0
$NRO_TORQUE_EXT 1
$PONTO_OBJ_SRO 0.0 0.0 0.0
$PONTO_FORCA_SRO 0.0 0.0 0.0
$ENDOBJ
$OBJETO 0
$NOME paral2.obt
$DENSIDADE 0.01
$POSICAO_SRU -400.0 0.0 0.0
$VEL_LINEAR_SRO 0.0 0.0 0.0
$NRO_FORCA_EXT 1
$FORCA_EXT_SRO 0.0 0.0 0.0
$ORIENT_SRO_SRU 0.0 0.0 0.0
$VEL_ANG_SRO 0.0 0.0 0.0
$NRO_TORQUE_EXT 1
$PONTO_OBJ_SRO 0.0 0.0 0.0
$PONTO_FORCA_SRO 0.0 0.0 0.0
$ENDOBJ
$ENDSIMU
```

Figura 6.20: Exemplo 5 - Estado do sistema dinâmico inicial

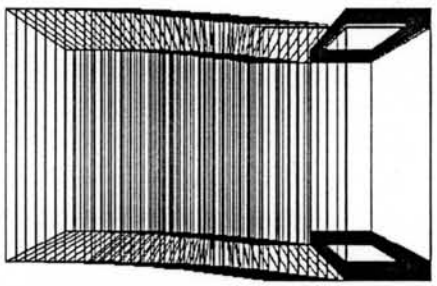

Figura 6.21: Exemplo 5 - Vista frontal da colisão

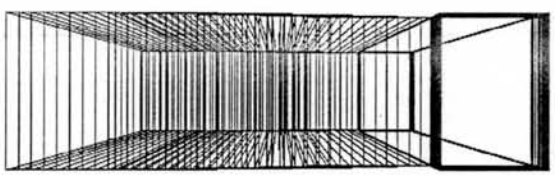

Figura 6.22: Exemplo 5 - Vista superior da colisão

7 CONCLUSÃO

A idéia deste trabalho surgiu da possibilidade de uma máquina, através do uso de recursos gráficos, poder simular visualmente mundos virtuais.

Desde logo se pôde perceber as dimensões desse problema envolvendo, principalmente, áreas como engenharia, física, matemática computacional e ciência da computação.

Definiu-se então que o estudo abordaria a simulação e tratamento de colisões de corpos rígidos não-articulados baseados em física utilizando análise dinâmica.

Sendo uma área recente em animação por computador adicionado ao fato da não existência de ferramentas que implementem métodos de controle de movimentos de baixo nível na instituição, decidiu-se, além desse estudo, implementar um protótipo que abordasse os referidos métodos. Espera-se que este trabalho sirva de base a outros trabalhos em áreas afins, por exemplo, como ferramenta básica para implementação de métodos de controle de níveis mais altos.

Durante o decorrer deste trabalho, verificou-se que a estratégia adotada na dinâmica do movimento é suficientemente genérica para ser estendida com o objetivo de permitir a simulação de corpos rígidos articulados.

A geração do movimento através de equações dinâmicas baseadas em leis físicas fornece um alto grau de realismo. Pode-se, ainda, utilizar métodos de controle que permitam adição de mais níveis de realismo e controle, tais como respostas as colisões.

Apesar das limitações no controle, permitindo apenas o fornecimento da informação de estímulos iniciais (de uma forma não intuitiva para o usuário), este trabalho está atingindo seus objetivos como uma plataforma de testes para validação

de um ambiente básico onde poderão ser simuladas interações entre objetos e dos mesmos com o ambiente.

De acordo com a abordagem para projeto e implementação de simulações gráficas dinâmicas apresentada em Zeltzer [ZEL 91], este trabalho surge no nível de abstração 'procedimento' e na técnica de interação 'programada'.

Como trabalho futuro, num primeiro momento, propõe-se a implementação das estratégias apresentadas na dinâmica da colisão de corpos rígidos.

Uma outra possibilidade inclui a extensão deste trabalho para o tratamento de corpos rígidos articulados.

Pode-se explorar, também, algoritmos de cálculo de momentos de inércia para objetos com geometria complexa permitindo, assim, um acréscimo ao universo de objetos a serem simulados.

Formas de controle mais intuitivas para o usuário podem ser obtidas a partir de mudanças na forma de resolução das equações dinâmicas de movimento, permitindo, por exemplo, a utilização de estratégias de controle de movimento como a dinâmica inversa.

E, por fim, as ferramentas aqui apresentadas e desenvolvidas podem ser integradas a um ambiente de projeto onde prioriza-se o realismo do comportamento dinâmico dos objetos como, por exemplo, nas áreas de animação, simulação e robótica.

ANEXO A-1 LISTAGEM DO PROGRAMA MOTOR CLIC

A-1.1 Estrutura de Dados

```
#define T_TRANS_X 0     /* Define tipos p/ ODE's translacional */
#define T_TRANS_Y 1
#define T_TRANS_Z 2
#define T_ORIENTACAO 3  /* Define tipos p/ ODE's orientacao */
#define N_ODES_TRANS 2  /* Define nro. de ODE's translacional */
#define N_ODES_ORIEN 6     /* Define nro. de ODE's orientacao */
#define NRO_FRAMES_SEGS 30 /* Define nro. de quadros / seg. */
#define N_COMPONENTES 9  /* Define nro. de componentes da m. de orient. */
#define T_SHAPE_CUBO 0   /* Define tipo de shape -> cubo */
#define MAXSTP 10000  /* Nro. max. de interacoes */
#define TINY 1.0E-30
#define PGROW  0.20
#define PSHRNK 0.25
#define FCOR 0.06666666  /* 1.0 / 15.0 */
#define SAFETY 0.9
#define ERRCON 6.0E-4 /* =~ a 4/SAFETY aumentando o poder de (1/PGROW) */
#define TRUE 1
#define FALSE 0

typedef int bool; /* Define tipo bool como variavel booleana */

typedef struct c_xyz{ /* Estrutura do tipo vetor */
        double x,
               y,
               z;
        }C_XYZ;

typedef struct c_faces{   /* Estrutura do tipo faces */
        int n_vert,
            *p_ind;       /* Aponta p/ uma tabela de vertices */
        C_XYZ normal,
              centroide;
        }C_FACES;

typedef struct c_objeto{  /* Estrutura do tipo objeto */
```

```
int   n_faces,    /* Descricao geometrica */
      n_vertices;
C_FACES *p_faces;
C_XYZ *p_vert;
C_XYZ all_forca, /* Vetor c/ resultado de todas forcas (SRU) */
      all_torque, /* Vetor c/ resultado de todos torques (SRO) */
      tensor,  /* Vetor tensor de inercia (SRO) */
double massa,  /* Massa do corpo rigido */
        **fun_trans_all_x,  /* Array p/ resultados de funcoes de translacao*/
        **fun_trans_all_y,  /* funcoes...[0][0...nstep_simu-1] (velocid.)  */
        **fun_trans_all_z,  /* funcoes...[1][0...nstep_simu-1] (posicao)   */
        **fun_ang_ori_all;  /* Array p/ resultados de funcoes de rotacao   */
                            /* funcoes...[0][0...nstep_simu-1] (vel_ang_x) */
                            /* funcoes...[1][0...nstep_simu-1] (vel_ang_y) */
                            /* funcoes...[2][0...nstep_simu-1] (vel_ang_z) */
                            /* funcao[3][4][5] angulos */
        int shape;          /* Tipo de formato geometrico */
        char nome[30];      /* Nome do objeto */
}C_OBJETO;


typedef struct c_atributos {  /* Atributos fisicos (estado inicial) */
        C_XYZ vel_linear,     /* vel. linear (SRO) */
              posicao,        /* Posicao (SRU) */
              *p_forca_ext,   /* Vetor c/ forcas ext. atuando (SRO) */
              vel_angular,    /* (SRO) */
              ang_orientacao, /* (SRUXSRO) */
              *p_forca_out_cm, /* Vetor c/ forcas atuando fora do c.m. (SRO) */
              *p_radius;      /* Vetor c/ o ponto do corpo o qual eh
                                 fornecida a forca (SRO) */
        int   nro_forca_ext,   /* Nro. de forcas ext. atuando */
              nro_torques_ext; /* Nro. de torques ext. atuando */
        double densidade;      /* Densidade de massa */
        }C_ATRIBUTOS;

typedef struct c_pts_inter{ /* Estrutura do tipo pontos intermediarios */
        double t;
        C_XYZ ponto;
        }C_PTS_INTER;

                        /* Objetos organizados como uma lista */
C_OBJETO *obj_corr,     /* Pointer p/ objeto corrente */
         *objetos;      /* Pointer p/ primeiro objeto */

char nome_modelo[30]="";  /* Nome do modelo fisico */
```

```
int  nstep_simu,          /* Nro. de passos p/ simulacao */

     nro_obj_simu,        /* Fixo temporariamente */
                          /* (odeint)  */
     kmax,                /* Nro. maximo de valores armazenados */

     nok_trans_x,         /* Nro. de aproximacoes ok */
     nok_trans_y,
     nok_trans_z,
     nok_ori,
     nbad_trans_x,        /* Nro. de aproximacoes nao ok */
     nbad_trans_y,
     nbad_trans_z,
     nbad_ori;


 double a_gravidade = -9.8, /* Aceleracao da gravidade do mundo virtual */

       *tempo_sincro,  /* Array p/ os tempos de simulacao */

       time_start,     /* Tempo de Inicio p/ Simulacao */
       time_finish,    /* Tempo de Termino da simulacao */

       h = 1.0/30.0,   /* Intervalo de Tempo utilizado */
       /* (odeint) */
       *trans_x_xp,    /* Vetores p/ valores de x */
       *trans_y_xp,
       *trans_z_xp,
       *ori_xp,
       exat = 0.2,         /* Exatidao requerida */
       hinic = 1.0/30.0,   /* Stepsize inicial */
       hminimo = 0.0;      /* Stepsize minimo */
```

A-1.2 Rotinas em Linguagem C

```
/*-------------------------------------------------------------------+
|                              alocacao.c                           |
|-------------------------------------------------------------------|
|    Modulo: alocacao                                               |
```

```
|    Descricao: Realiza a alocacao/liberacao das areas de trabalho      |
|               utilizadas no prototipo                                 |
|    Autor: Robson R. Lemos                                             |
+-----------------------------------------------------------------------*/

#include "estruda.h"
extern C_OBJETO *objetos;
extern int nro_obj_simu;

/*-----------------------------------------------------------------------+
|    Aloca_Vector                                                       |
|    Funcao: Aloca um array do tipo vetor de nvar posicoes              |
|    Parametros:  int nvar (Nro. de posicoes)                           |
|                 double **p_vector_out (Endereco p/ array de vetores)  |
|    Globais: ---*---                                                   |
|    Vlr. Retorno:  double **p_vector_out (Endereco alocado)            |
+-----------------------------------------------------------------------*/

void Aloca_Vector(nvar, p_vector_out)
int nvar;
double **p_vector_out;
{
 if ((*p_vector_out = (double *) calloc(nvar,sizeof(double))) == NULL)
    {
     printf("Memoria Insuficiente p/ p_vector_out \n");
     exit(0);
    }
}

/*-----------------------------------------------------------------------+
|    Free_Vector                                                        |
|    Funcao:  Libera um array do tipo vetor de nvar posicoes            |
|    Parametros:   double *p_vector (Endereco p/ array de vetores)      |
|    Globais:  ---*---                                                  |
|    Vlr. Retorno:  double *p_vector (Endereco liberado)                |
+-----------------------------------------------------------------------*/

void Free_Vector(p_vector)
double *p_vector;
{
 free(p_vector);
}

/*-----------------------------------------------------------------------+
|    Aloca_Vetor_XYZ                                                    |
```

```
|    Funcao: Aloca areas do tipo vetor                                    |
|    Parametros: int n (Nro. de areas)                                    |
|    Globais: ---*---                                                     |
|    Vlr. Retorno: C_XYZ *p_out (Endereco alocado)                        |
+-------------------------------------------------------------------------*/

C_XYZ *Aloca_Vetor_XYZ(n)
int n;
{
 C_XYZ *p_out;
 if ((p_out = (C_XYZ *) calloc(n,sizeof(C_XYZ))) == NULL)
         {
          printf("Memoria insuficiente p/ C_XYZ \n");
          exit(0);
         }
 return p_out;
}

/*-------------------------------------------------------------------------+
|    Free_Vetor_XYZ                                                       |
|    Funcao: Libera areas do tipo vetor                                   |
|    Parametros: C_XYZ *p_vet_xyz (Endereco p/ area de vetor)             |
|    Globais: ---*---                                                     |
|    Vlr. Retorno: C_XYZ *p_vet_xyz (Endereco liberado)                   |
+-------------------------------------------------------------------------*/

void Free_Vetor_XYZ(p_vet_xyz)
C_XYZ *p_vet_xyz;
{
 free(p_vet_xyz);
}

/*-------------------------------------------------------------------------+
|    Aloca_Faces                                                          |
|    Funcao:  Aloca areas do tipo faces                                   |
|    Parametros: int n_faces (Nro. de areas)                              |
|                C_FACES **p_faces_out (Endereco p/ area de faces)        |
|    Globais: ---*---                                                     |
|    Vlr. Retorno: C_FACES **p_faces_out (Endereco alocado)               |
+-------------------------------------------------------------------------*/

void Aloca_Faces(n_faces,p_faces_out)
int n_faces;
C_FACES **p_faces_out;
{
```

```
 if ((*p_faces_out = (C_FACES *) calloc(n_faces,sizeof(C_FACES))) == NULL)
    {
     printf("Memoria insuficiente p/ C_FACES \n");
     exit(0);
    }
}

/*-------------------------------------------------------------------+
|    Free_Faces                                                      |
|    Funcao: Libera areas do tipo faces                              |
|    Parametros: C_FACES *p_faces_out (Endereco p/ area de faces)    |
|    Globais: ---*---                                                |
|    Vlr. Retorno: C_FACES *p_faces_out (Endereco liberado)          |
+-------------------------------------------------------------------*/

void Free_Faces(p_faces_out)
C_FACES *p_faces_out;
{
 free(p_faces_out);
}

/*-------------------------------------------------------------------+
|    Aloca_PTS_INTER                                                 |
|    Funcao: Aloca areas do tipo pontos intermediarios               |
|    Parametros: int nvar (Nro. de areas)                            |
|    Globais: ---*---                                                |
|    Vlr. Retorno: C_PTS_INTER *p_out (Endereco alocado)             |
+-------------------------------------------------------------------*/

C_PTS_INTER *Aloca_PTS_INTER(nvar)
int nvar;
{
 C_PTS_INTER *p_out;
 if ((p_out = (C_PTS_INTER *) calloc(nvar,sizeof(C_PTS_INTER))) == NULL)
    {
     printf("Memoria insuficiente p/ C_PTS_INTER \n");
     exit(0);
    }
 return p_out;
}

/*-------------------------------------------------------------------+
|    Free_PTS_INTER                                                  |
|    Funcao: Libera areas do tipo pontos intermediarios              |
|    Parametros: C_PTS_INTER *p_pts_inter (Ender. p/ areas de pontos |
```

```
|                                                  intermediarios)             |
|    Globais: ---*---                                                          |
|    Vlr. Retorno:  C_PTS_INTER *p_pts_inter (Endereco liberado)               |
+-----------------------------------------------------------------------------*/

void  Free_PTS_INTER(p_pts_inter)
C_PTS_INTER *p_pts_inter;
{
 free(p_pts_inter);
}

/*-----------------------------------------------------------------------------+
|    Aloca_Matrix                                                              |
|    Funcao : Aloca um array do tipo matriz de nvar X nstep posicoes           |
|    Parametros:  int nvar (Nro. de linhas)                                    |
|                 int nstep (Nro. de colunas)                                  |
|                 double ***p_matrix_out (Endereco p/ array de matrizes)       |
|    Globais: ---*---                                                          |
|    Vlr. Retorno: double ***p_matrix_out (Endereco alocado)                   |
+-----------------------------------------------------------------------------*/

void Aloca_Matrix(nvar, nstep, p_matrix_out)
int nvar,
    nstep;
double ***p_matrix_out;
{
 int i;
 if ((*p_matrix_out = (double **) calloc(nvar,sizeof(double *))) == NULL)
    {
     printf("Memoria Insuficiente p/ p_matrix_out \n");
     return;
    }
 for (i=0; i<nvar; i++)
    {
     if (((*p_matrix_out)[i] = (double *) calloc(nstep,sizeof(double))) == NULL)
        {
         printf("Memoria Insuficiente p/ p_matrix_out[i] \n");
         while (--i)
           free((*p_matrix_out)[i]);
         free(*p_matrix_out);
         return;
        }
    }
}
```

```
/*-------------------------------------------------------------------+
|    Free_Matrix                                                     |
|    Funcao: Libera um array do tipo matriz                          |
|    Parametros:  int nvar (Nro. de linhas)                          |
|                 int nstep (Nro. de colunas)                        |
|                 double **p_matrix (Endereco p/ array de matrizes)  |
|    Globais:  ---*---                                               |
|    Vlr. Retorno:  double **p_matrix (Endereco liberado)            |
+-------------------------------------------------------------------*/

void Free_Matrix(nvar,p_matrix)
int nvar;
double **p_matrix;
{
 int i;

 for (i=0; i<nvar; i++)
     free(p_matrix[i]);
 free(p_matrix);
}

/*-------------------------------------------------------------------+
|    Aloca_Objetos                                                   |
|    Funcao: Aloca areas do tipo objeto                              |
|    Parametros: ---*---                                             |
|    Globais: C_OBJETO *objetos (Pointer p/ objeto inicial)          |
|             int nro_obj_simu (Nro. de objetos simulados)           |
|    Vlr. Retorno: C_OBJETO *objetos (Endereco alocado)              |
+-------------------------------------------------------------------*/

void Aloca_Objetos()
{
 if ((objetos = (C_OBJETO *) calloc(nro_obj_simu,sizeof(C_OBJETO))) == NULL)
    {
     printf("Memoria insuficiente p/ C_OBJETO \n");
     exit(0);
    }
}

/*-------------------------------------------------------------------+
|    Free_Objetos                                                    |
|    Funcao: Libera areas do tipo objeto                             |
|    Parametros: ---*---                                             |
|    Globais: C_OBJETO *objetos (Endereco p/ area de objetos)        |
|    Vlr. Retorno:  C_OBJETO *objetos (Endereco liberado)            |
```

```
+-------------------------------------------------------------------*/

void Free_Objetos()
{
 free(objetos);
}

/*-------------------------------------------------------------------+
|    Aloca_Atributos                                                 |
|    Funcao: Aloca areas do tipo atributos                           |
|    Parametros: ---*---                                             |
|    Globais: ---*---                                                |
|    Vlr. Retorno: C_ATRIBUTOS *p_out (Endereco alocado)             |
+-------------------------------------------------------------------*/

C_ATRIBUTOS *Aloca_Atributos()
{
 C_ATRIBUTOS *p_out;

 if (( p_out = (C_ATRIBUTOS *) malloc(sizeof(C_ATRIBUTOS))) == NULL)
    {
     printf("Memoria insuficiente p/ C_ATRIBUTOS \n");
     exit(0);
    }
 return p_out;
}

/*-------------------------------------------------------------------+
|    Free_Atributos                                                  |
|    Funcao: Libera areas do tipo atributos                          |
|    Parametros:  C_ATRIBUTOS *p_atributos (End. p/ area de atributos)|
|    Globais: ---*---                                                |
|    Vlr. Retorno: C_ATRIBUTOS *p_atributos (Endereco liberado)      |
+-------------------------------------------------------------------*/

void Free_Atributos(p_atributos)
C_ATRIBUTOS *p_atributos;
{
 free(p_atributos);
}



/*-------------------------------------------------------------------+
|                              matematica.c                          |
```

```
|-------------------------------------------------------------------|
|    Modulo: matematica                                             |
|    Descricao: Realiza os procedimentos matematicos utilizados nas |
|               rotinas de dinamica do movimento e deteccao de colisoes |
|    Autor: Robson R. Lemos                                         |
+-------------------------------------------------------------------*/

#include "estruda.h"
extern int nro_obj_simu;
extern C_OBJETO *objetos;
extern C_OBJETO *obj_corr;

/*-------------------------------------------------------------------+
|    Atualiza_Componentes                                           |
|    Funcao: Atualiza as componentes da matriz de orientacao        |
|    Parametros: double *fun_comp_ori (Areas de comp. da matriz orien.) |
|                C_XYZ *angulos (Angulos de orientacao)             |
|    Globais: ---*---                                               |
|    Vlr. Retorno: double *fun_comp_ori (Comp. da matriz orien.)    |
+-------------------------------------------------------------------*/

void Atualiza_Componentes(fun_comp_ori,angulos)
double *fun_comp_ori;
C_XYZ *angulos;
{
 double ang_rot_x,
        ang_rot_y,
        ang_rot_z,
sin_ang_x,
sin_ang_y,
sin_ang_z,
cos_ang_x,
cos_ang_y,
cos_ang_z;
 /* Inicializa funcoes p/ compon. da matriz orientacao */
 /* Matriz mudanca de base (cossenos diretores) */
 ang_rot_x  = (angulos->x * M_PI / 180.0);
 ang_rot_y  = (angulos->y * M_PI / 180.0);
 ang_rot_z  = (angulos->z * M_PI / 180.0);
 sin_ang_x  = sin(ang_rot_x);
 sin_ang_y  = sin(ang_rot_y);
 sin_ang_z  = sin(ang_rot_z);
 cos_ang_x  = cos(ang_rot_x);
 cos_ang_y  = cos(ang_rot_y);
 cos_ang_z  = cos(ang_rot_z);
```

```
 /* r11 matriz orientacao */
 fun_comp_ori[0] = cos_ang_y * cos_ang_z;
 /* r12 matriz orientacao */
 fun_comp_ori[1] = -sin_ang_z * cos_ang_y;
 /* r13 matriz orientacao */
 fun_comp_ori[2] = sin_ang_y;
 /* r21 matriz orientacao */
 fun_comp_ori[3] = (sin_ang_z * cos_ang_x) + (cos_ang_z * sin_ang_y * sin_ang_x);
 /* r22 matriz orientacao */
 fun_comp_ori[4] = (cos_ang_x * cos_ang_z) - (sin_ang_x * sin_ang_y * sin_ang_z);
 /* r23 matriz orientacao */
 fun_comp_ori[5] = - cos_ang_y * sin_ang_x;
 /* r31 matriz orientacao */
 fun_comp_ori[6] = (sin_ang_z * sin_ang_x) - (cos_ang_z * sin_ang_y * cos_ang_x);
 /* r32 matriz orientacao */
 fun_comp_ori[7] = (cos_ang_z * sin_ang_x) + (sin_ang_z * sin_ang_y * cos_ang_x)
 /* r33 matriz orientacao */
 fun_comp_ori[8] = cos_ang_y * cos_ang_x;
}

/*-------------------------------------------------------------------+
|    Mudanca_Base_SRO_2_SRU                                          |
|    Funcao: Realiza a mudanca de base de pontos do SRO para o SRU de |
|            um unico vetor                                          |
|    Parametros: C_XYZ *vetor_out (Area de vetor)                    |
|                C_XYZ *vetor_in (Vetor no SRO)                      |
|                double *fun_comp_ori (Comp. da matriz de orientacao) |
|                double *fun_trans (Posicao do centro de massa)      |
|    Globais : ---*---                                               |
|    Vlr. Retorno:  C_XYZ *vetor_out (Vetor no SRU)                  |
+-------------------------------------------------------------------*/

void Mudanca_Base_SRO_2_SRU(vetor_out,vetor_in,fun_comp_ori,fun_trans)
C_XYZ *vetor_out,
      *vetor_in;
double *fun_comp_ori,
       *fun_trans;
{
 C_XYZ antes;
 antes.x =  vetor_in->x;
 antes.y =  vetor_in->y;
 antes.z =  vetor_in->z;
 /* Transformacao c/ Matriz de Cossenos Diretores */
 vetor_out->x = antes.x * fun_comp_ori[0] +
                antes.y * fun_comp_ori[1] +
```

```
                antes.z * fun_comp_ori[2];
 vetor_out->y = antes.x * fun_comp_ori[3] +
                antes.y * fun_comp_ori[4] +
                antes.z * fun_comp_ori[5];
 vetor_out->z = antes.x * fun_comp_ori[6] +
                antes.y * fun_comp_ori[7] +
                antes.z * fun_comp_ori[8];
/* Translacao p/ a posicao do c.m. no sru */
 vetor_out->x += fun_trans[0];
 vetor_out->y += fun_trans[1];
 vetor_out->z += fun_trans[2];
}

/*-------------------------------------------------------------------+
|    Mudanca_Base_N_SRO_2_SRU                                        |
|    Funcao: Realiza a mudanca de base de pontos do SRO para o SRU de |
|            n vetores                                               |
|    Parametros: C_XYZ *vetor_out (Area de vetores)                  |
|                C_XYZ *vetor_in (Vetores no SRO)                    |
|                double *fun_comp_ori (Comp. da matriz de orientacao) |
|                double *fun_trans (Posicao do centro de massa)      |
|                int nro_vet (Nro. de vetores)                       |
|    Globais: ---*---                                                |
|    Vlr. Retorno:  C_XYZ *vetor_out (Vetores no SRU)                |
+-------------------------------------------------------------------*/

void Mudanca_Base_N_SRO_2_SRU(vetor_out,vetor_in,fun_comp_ori,fun_trans,nro_vet)
C_XYZ *vetor_out,
      *vetor_in;
double *fun_comp_ori,
       *fun_trans;
int nro_vet;
{
 int i;
 for (i=0;i<nro_vet;i++)
     {
      Mudanca_Base_SRO_2_SRU(&vetor_out[i],&vetor_in[i],fun_comp_ori,fun_trans);
     }
}

/*-------------------------------------------------------------------+
|    Produto_Vetorial                                                |
|    Funcao: Realizar o produto vetorial                             |
|    Parametros: C_XYZ *vetor1 (Primeiro vetor)                      |
|                C_XYZ *vetor2 (Segundo vetor)                       |
```

```
|                C_XYZ *vetor_out (Area de vetor)                          |
|    Globais: ---*---                                                      |
|    Vlr. Retorno:  C_XYZ *vetor_out (Vetor resultante)                    |
+-------------------------------------------------------------------------*/

void Produto_Vetorial(vetor1,vetor2,vetor_out)
C_XYZ *vetor1,
      *vetor2,
      *vetor_out;
{
 vetor_out->x = (vetor1->y * vetor2->z) - (vetor1->z * vetor2->y);
 vetor_out->y = (vetor1->z * vetor2->x) - (vetor1->x * vetor2->z);
 vetor_out->z = (vetor1->x * vetor2->y) - (vetor1->y * vetor2->x);
}

/*-------------------------------------------------------------------------+
|    Calcula_V_Normais_Faces                                               |
|    Funcao: Calcular os vetores normais as faces dos objetos              |
|    Parametros: ---*---                                                   |
|    Globais: ---*---                                                      |
|    Vlr. Retorno:    C_XYZ *normal (Vetores normais as faces)             |
+-------------------------------------------------------------------------*/

void  Calcula_V_Normais_Faces()
{
 int i,j,ind0,ind1,ind2;
 C_XYZ vetor1,vetor2;
 for(i=0;i < nro_obj_simu;i++)
    {
     obj_corr = objetos + i;
     for (j=0; j < obj_corr->n_faces; j++)
         {
          ind0 = obj_corr->p_faces[j].p_ind[0];
          ind1 = obj_corr->p_faces[j].p_ind[1];
          ind2 = obj_corr->p_faces[j].p_ind[2];
          vetor1.x = obj_corr->p_vert[ind2].x - obj_corr->p_vert[ind1].x;
          vetor1.y = obj_corr->p_vert[ind2].y - obj_corr->p_vert[ind1].y;
          vetor1.z = obj_corr->p_vert[ind2].z - obj_corr->p_vert[ind1].z;
          vetor2.x = obj_corr->p_vert[ind0].x - obj_corr->p_vert[ind1].x;
          vetor2.y = obj_corr->p_vert[ind0].y - obj_corr->p_vert[ind1].y;
          vetor2.z = obj_corr->p_vert[ind0].z - obj_corr->p_vert[ind1].z;
          Produto_Vetorial(&vetor1,&vetor2,&obj_corr->p_faces[j].normal);
         }
    }
}
```

```
/*-------------------------------------------------------------------+
|    Produto_Escalar_2V                                             |
|    Funcao: Calcular o produto escalar de dois vetores             |
|    Parametros: C_XYZ *vet1 (Primeiro vetor)                       |
|                C_XYZ *vet2 (Segundo vetor)                        |
|                double *valor (Pointer p/ double)                  |
|    Globais: ---*---                                               |
|    Vlr. Retorno: double *valor (Valor resultante)                 |
+-------------------------------------------------------------------*/

void  Produto_Escalar_2V(vet1,vet2,valor)
C_XYZ *vet1,*vet2;
double *valor;
{
 valor[0] = vet1->x * vet2->x + vet1->y * vet2->y + vet1->z * vet2->z;
}

/*-------------------------------------------------------------------+
|    Mudanca_Base_SRU_2_SRO                                         |
|    Funcao: Realiza a mudanca de base de pontos do SRU para o SRO  |
|    Parametros: C_XYZ *vetor_out (Area de vetor)                   |
|                C_XYZ *vetor_in (Vetor no SRU)                     |
|                double *fun_comp_ori (Comp. da matriz de orientacao)|
|                double *fun_trans (Posicao do centro de massa)     |
|    Globais: ---*---                                               |
|    Vlr. Retorno:  C_XYZ *vetor_out (Vetor no SRO)                 |
+-------------------------------------------------------------------*/

void Mudanca_Base_SRU_2_SRO(vetor_out,vetor_in,fun_comp_ori,fun_trans)
C_XYZ *vetor_out,
      *vetor_in;
double *fun_comp_ori,
       *fun_trans;
{
 C_XYZ antes;
 antes.x =  vetor_in->x - fun_trans[0];
 antes.y =  vetor_in->y - fun_trans[1];
 antes.z =  vetor_in->z - fun_trans[2];
 /* Transformacao c/ Matriz de Cossenos Diretores */
 vetor_out->x = antes.x * fun_comp_ori[0] +
                antes.y * fun_comp_ori[1] +
                antes.z * fun_comp_ori[2];
 vetor_out->y = antes.x * fun_comp_ori[3] +
                antes.y * fun_comp_ori[4] +
```

```
                antes.z * fun_comp_ori[5];
 vetor_out->z = antes.x * fun_comp_ori[6] +
                antes.y * fun_comp_ori[7] +
                antes.z * fun_comp_ori[8];
}

/*---------------------------------------------------------------------+
|    Calcula_Centroides_F_4V                                           |
|    Funcao:  Calcular os centroides p/ faces de objetos compostas por |
|             quatro vertices                                          |
|    Parametros:   ---*---                                             |
|    Globais: C_OBJETO *obj_corr (Pointer p/ objeto corrente)          |
|             C_OBJETO *objetos (Pointer p/ objeto inicial)            |
|    Vlr. Retorno:  C_XYZ centroide (Vetor posicao do centro da face)  |
+---------------------------------------------------------------------*/

void  Calcula_Centroides_F_4V()
{
 int i,j,ind0,ind1,ind2,ind3;
 C_XYZ pt1,pt2;
 for (i=0;i < nro_obj_simu;i++)
     {
      obj_corr = objetos + i;
      for (j=0;j < obj_corr->n_faces;j++)
          {
           ind0 = obj_corr->p_faces[j].p_ind[0];
           ind1 = obj_corr->p_faces[j].p_ind[1];
           ind2 = obj_corr->p_faces[j].p_ind[2];
           ind3 = obj_corr->p_faces[j].p_ind[3];
           pt1.x = (obj_corr->p_vert[ind0].x + obj_corr->p_vert[ind1].x) / 2.0;
           pt1.y = (obj_corr->p_vert[ind0].y + obj_corr->p_vert[ind1].y) / 2.0;
           pt1.z = (obj_corr->p_vert[ind0].z + obj_corr->p_vert[ind1].z) / 2.0;
           pt2.x = (obj_corr->p_vert[ind2].x + obj_corr->p_vert[ind3].x) / 2.0;
           pt2.y = (obj_corr->p_vert[ind2].y + obj_corr->p_vert[ind3].y) / 2.0;
           pt2.z = (obj_corr->p_vert[ind2].z + obj_corr->p_vert[ind3].z) / 2.0;
           obj_corr->p_faces[j].centroide.x = (pt1.x + pt2.x) / 2.0;
           obj_corr->p_faces[j].centroide.y = (pt1.y + pt2.y) / 2.0;
           obj_corr->p_faces[j].centroide.z = (pt1.z + pt2.z) / 2.0;
          }
     }
}


/*---------------------------------------------------------------------+
|                              deteccao.c                              |
```

```
|-------------------------------------------------------------------|
|    Modulo: deteccao                                               |
|    Descricao: Realiza as etapas para o algoritmo de deteccao de   |
|               colisoes entre pares de objetos poliedricos         |
|    Autor: Robson R. Lemos                                         |
+-------------------------------------------------------------------*/

#include "estruda.h"
extern int nro_obj_simu;
extern C_OBJETO *obj_corr;
extern C_OBJETO *objetos;

/*-------------------------------------------------------------------+
|    Monta_VertB_2_SRA                                              |
|    Funcao: Transforma um vertice do objeto B (definido no SRO do obj. |
|            B) para o SRO do objeto A                              |
|    Parametros: C_XYZ *V_B_SRA (Area p/ vetor)                     |
|                C_XYZ *V_B (Vetor no SRO do obj. B)                |
|                int step_atual (Passo atual)                       |
|                C_OBJETO *obj_B (Pointer p/ obj. B)                |
|                C_OBJETO *obj_A (Pointer p/ obj. A)                |
|    Globais : ---*---                                              |
|    Vlr. Retorno :  C_XYZ *V_B_SRA (vetor no SRO do obj. A)        |
+-------------------------------------------------------------------*/

void Monta_VertB_2_SRA(V_B_SRA,V_B,obj_B,obj_A,step_atual)
C_XYZ *V_B_SRA,*V_B;
int step_atual;
C_OBJETO *obj_B,*obj_A;
{
 C_XYZ angulos,vet_SRU,vet_SRO;
 double *fun_comp,*fun_trans;
 Aloca_Vector(N_COMPONENTES,&fun_comp);
 Aloca_Vector(3,&fun_trans);
 angulos.x = obj_B->fun_ang_ori_all[3][step_atual];
 angulos.y = obj_B->fun_ang_ori_all[4][step_atual];
 angulos.z = obj_B->fun_ang_ori_all[5][step_atual];
 fun_trans[0] = obj_B->fun_trans_all_x[1][step_atual];
 fun_trans[1] = obj_B->fun_trans_all_y[1][step_atual];
 fun_trans[2] = obj_B->fun_trans_all_z[1][step_atual];
 Atualiza_Componentes(fun_comp,&angulos);
 /* OBJ. B NO SRU */
 Mudanca_Base_SRO_2_SRU(&vet_SRU,V_B,fun_comp,fun_trans);
 angulos.x = obj_A->fun_ang_ori_all[3][step_atual];
 angulos.y = obj_A->fun_ang_ori_all[4][step_atual];
```

```
 angulos.z = obj_A->fun_ang_ori_all[5][step_atual];
 fun_trans[0] = obj_A->fun_trans_all_x[1][step_atual];
 fun_trans[1] = obj_A->fun_trans_all_y[1][step_atual];
 fun_trans[2] = obj_A->fun_trans_all_z[1][step_atual];
 Atualiza_Componentes(fun_comp,&angulos);
 /* OBJ. B NO SRO DE A */
 Mudanca_Base_SRU_2_SRO(&vet_SRO,&vet_SRU,fun_comp,fun_trans);
 V_B_SRA->x = vet_SRO.x;
 V_B_SRA->y = vet_SRO.y;
 V_B_SRA->z = vet_SRO.z;
 Free_Vector(fun_comp);
 Free_Vector(fun_trans);
}

/*-------------------------------------------------------------------+
|    Classifica_Ts                                                   |
|    Funcao: Classifica em ordem crescente os pontos intermediarios  |
|            de uma aresta                                           |
|    Parametros: C_PTS_INTER *pts (Area de pontos intermediarios)    |
|                int nro_pontos (Nro. de pontos intermediarios)      |
|    Globais: ---*---                                                |
|    Vlr. Retorno:   C_PTS_INTER *pts (Pontos inter. classificados)  |
+-------------------------------------------------------------------*/

void  Classifica_Ts(pts,nro_pts)
C_PTS_INTER *pts;
int nro_pts;
{
 C_PTS_INTER *pts_aux;
 C_XYZ ponto_aux;
 double t_aux;
 int i,j;
 for (i=0;i < nro_pts;i++)
     {
      pts_aux = pts + i;
      if ((pts_aux->t <= 0.0) || (pts_aux->t >= 1.0))
         pts_aux->t = 1.0;
     }
 /* CLASSIFICA pts_aux->t EM ORDEM CRESCENTE P/ OS VAL. ENTRE [0,1] */
 for (i=0;i < nro_pts;i++)
     {
      for (j=0;j < nro_pts - 1;j++)
          {
           pts_aux = pts + j;
           if (pts_aux->t > (pts_aux + 1)->t)
```

```
                {
                 t_aux = pts_aux->t;
                 pts_aux->t = (pts_aux + 1)->t;
                 (pts_aux + 1)->t = t_aux;
                 ponto_aux.x = pts_aux->ponto.x;
                 ponto_aux.y = pts_aux->ponto.y;
                 ponto_aux.z = pts_aux->ponto.z;
                 pts_aux->ponto.x = (pts_aux + 1)->ponto.x;
                 pts_aux->ponto.y = (pts_aux + 1)->ponto.y;
                 pts_aux->ponto.z = (pts_aux + 1)->ponto.z;
                 (pts_aux + 1)->ponto.x = pts_aux->ponto.x;
                 (pts_aux + 1)->ponto.y = pts_aux->ponto.y;
                 (pts_aux + 1)->ponto.z = pts_aux->ponto.z;
                }
            }
        }
}

/*-------------------------------------------------------------------+
|    Calcula_PTS_Medios                                              |
|    Funcao: Calcula pontos medios de segmentos de linha             |
|    Parametros: C_XYZ *pts_medios (Area p/ vetores)                 |
|                C_XYZ *v_i (Vetor i da aresta)                      |
|                C_XYZ *v_j (Vetor j da aresta)                      |
|    Globais: ---*---                                                |
|    Vlr. Retorno:  C_XYZ *pts_medios (Vetores posicao de pontos med.)  |
+-------------------------------------------------------------------*/

void Calcula_PTS_Medios(pts_medios,v_i,v_j,pts,nvar)
C_XYZ *pts_medios,
      *v_i, *v_j;
C_PTS_INTER *pts;
int nvar;
{
 int i;
 C_XYZ *pts_medios_aux;
 C_PTS_INTER *pts_aux;
 pts_aux = pts;
 for (i=0; i < nvar;i++)
     {
      if (i < nvar - 2)
         pts_aux = pts + i;
      pts_medios_aux = pts_medios + i;
      if ((i == 0) || (i == nvar - 1))
```

```
        {
         if (i == 0)
            {
             pts_medios_aux->x = (v_i->x + pts_aux->ponto.x) / 2.0;
             pts_medios_aux->y = (v_i->y + pts_aux->ponto.y) / 2.0;
             pts_medios_aux->z = (v_i->z + pts_aux->ponto.z) / 2.0;
            }
         else
            {
             if (i == 1)
                {
                 pts_medios_aux->x = (v_j->x + pts_aux->ponto.x) / 2.0;
                 pts_medios_aux->y = (v_j->y + pts_aux->ponto.y) / 2.0;
                 pts_medios_aux->z = (v_j->z + pts_aux->ponto.z) / 2.0;
                }
             else
                {
                 pts_medios_aux->x = (v_j->x + (pts_aux + 1)->ponto.x) / 2.0;
                 pts_medios_aux->y = (v_j->y + (pts_aux + 1)->ponto.y) / 2.0;
                 pts_medios_aux->z = (v_j->z + (pts_aux + 1)->ponto.z) / 2.0;
                }
            }
        }
       else
          {
           pts_medios_aux->x = (pts_aux->ponto.x + (pts_aux + 1)->ponto.x) / 2.0;
           pts_medios_aux->y = (pts_aux->ponto.y + (pts_aux + 1)->ponto.y) / 2.0;
           pts_medios_aux->z = (pts_aux->ponto.z + (pts_aux + 1)->ponto.z) / 2.0;
          }
      }
}

/*-------------------------------------------------------------------+
|    Ponto_B_Inside_Obj_A                                            |
|    Funcao: Verificar se um ponto do obj. B encontra-se dentro do   |
|            volume geometrico do obj. A                             |
|    Parametros: C_XYZ *p_xyz (Pointer p/ area de vetor)             |
|                int n_int (Nro. de pontos)                          |
|                int ind (Indice do ponto)                           |
|                int step_atu (Passo atual)                          |
|    Globais: ---*---                                                |
|    Vlr. Retorno:  bool TRUE (Condicao p/ ponto dentro)             |
|                   bool FALSE (Condicao p/ ponto fora)              |
+-------------------------------------------------------------------*/
```

```
bool Ponto_B_Inside_Obj_A(p_xyz,n_int,ind,obj_A,obj_B,step_atu)
C_XYZ *p_xyz;
int n_int,ind,step_atu;
C_OBJETO *obj_A,*obj_B;
{
 int i,j,indi,cont;
 double resul;
 bool SINAL_POS = FALSE,
      SINAL_NEG = FALSE;
 C_XYZ vet_B_SRA,*p_vet,vetor1,vetor2;
 p_vet = p_xyz;
 for (cont=0;cont < n_int;cont++)
     {
      /* TESTE P/ First... + Third_Step_3D_Cyrus_Beck */
      if (n_int == 1) p_vet = p_xyz + ind;
      for (i=0;i < obj_A->n_faces;i++)  /* FACE A */
          {
           for (j=0;j < obj_A->p_faces[i].n_vert;j++)   /* NRO. VERT./FACE A */
               {
                indi = obj_A->p_faces[i].p_ind[j];
                if (n_int == 1)
                   {
                    /* TESTE P/ First... + Third_Step_3D_Cyrus_Beck */
                    Monta_VertB_2_SRA(&vet_B_SRA,&p_vet[cont],obj_B,
                                      obj_A,step_atu);
                   }
                else
                   {
                    /* TESTE P/ Second_Step_3D_Cyrus_Beck */
                    vet_B_SRA.x = p_vet[cont].x;
                    vet_B_SRA.y = p_vet[cont].y;
                    vet_B_SRA.z = p_vet[cont].z;
                   }
                vetor1.x = vet_B_SRA.x - obj_A->p_vert[indi].x;
                vetor2.x = obj_A->p_faces[i].normal.x;
                vetor1.y = vet_B_SRA.y - obj_A->p_vert[indi].y;
                vetor2.y = obj_A->p_faces[i].normal.y;
                vetor1.z = vet_B_SRA.z - obj_A->p_vert[indi].z;
                vetor2.z = obj_A->p_faces[i].normal.z;
                Produto_Escalar_2V(&vetor1,&vetor2,&resul);
                if (resul >= 0.0)
                   SINAL_POS = TRUE;
                else
                   SINAL_NEG = TRUE;
               }
```

```
          }
       /* NAO HOUVE NENHUM POSITIVO P/ TODAS AS FACES */
       if (!SINAL_POS) return TRUE;
       SINAL_POS = FALSE;
       SINAL_NEG = FALSE;
      }
 return FALSE;
}

/*-------------------------------------------------------------------------+
|    Le_Objeto_Geometrico                                                  |
|    Funcao:  Realiza a leitura de um arquivo .obt                         |
|    Parametros:  ---*---                                                  |
|    Globais: C_OBJETO *obj_corr (Pointer p/ objeto corrente)              |
|    Vlr. Retorno: ---*---                                                 |
+-------------------------------------------------------------------------*/

void Le_Objeto_Geometrico()
{
 char token[30];
 double outros1;
 int i,
     j,
     outros2;
 FILE *fp1;
 if ((fp1 = fopen(obj_corr->nome,"rt")) == NULL)
    {
     printf("Erro na abertura de arquivo (.obt) \n");
     exit(0);
    }
 /* $OBJETO */
 fscanf(fp1,"%s",token);
 fscanf(fp1,"%s",token);
 fscanf(fp1,"%lf",&outros1);
 fscanf(fp1,"%lf",&outros1);
 fscanf(fp1,"%lf",&outros1);
 fscanf(fp1,"%lf",&outros1);
 /* $FACES */
 fscanf(fp1,"%s",token);
 fscanf(fp1,"%d",&obj_corr->n_faces);
 Aloca_Faces(obj_corr->n_faces,&obj_corr->p_faces);
 for (i=0;i < obj_corr->n_faces;i++)
     {
      fscanf(fp1,"%d",&obj_corr->p_faces[i].n_vert);
      if ((obj_corr->p_faces[i].p_ind = (int *)
```

```
          calloc(obj_corr->p_faces[i].n_vert,sizeof(int))) == NULL)
        {
         printf("Memoria insuficiente p/ int \n");
         exit(0);
        }
      for (j=0;j < obj_corr->p_faces[i].n_vert;j++)
          {
           fscanf(fp1,"%d",&obj_corr->p_faces[i].p_ind[j]);
          }
      fscanf(fp1,"%d",&outros2);
     }
 /* $VERTICE */
 fscanf(fp1,"%s",token);
 fscanf(fp1,"%d",&obj_corr->n_vertices);
 obj_corr->p_vert = Aloca_Vetor_XYZ(obj_corr->n_vertices);
 for (i=0;i < obj_corr->n_vertices;i++)
     {
      fscanf(fp1,"%lf",&obj_corr->p_vert[i].x);
      fscanf(fp1,"%lf",&obj_corr->p_vert[i].y);
      fscanf(fp1,"%lf",&obj_corr->p_vert[i].z);
     }
 fscanf(fp1,"%s",token);
 fclose(fp1);
}


/*-------------------------------------------------------------------+
|    First_Step_3D_Cyrus_Beck                                        |
|    Funcao: Testar a presenca de vertices do obj. B dentro das faces|
|            do obj. A                                               |
|    Parametros: int step_atu  (Passo de tempo)                      |
|    Globais : ---*---                                               |
|    Vlr. Retorno:  bool TRUE (Condicao p/ ponto dentro)             |
|                   bool FALSE (Condicao p/ ponto fora)              |
+-------------------------------------------------------------------*/

bool First_Step_3D_Cyrus_Beck(step_atu)
int step_atu;
{
 int i,
     j,
     k;

 C_OBJETO *obj_tes, *obj_grao;
 bool Houve_Colisao = FALSE;
```

```
 for (i=0;i < nro_obj_simu; i++)  /* OBJETOS B */
     {
      obj_tes = objetos + i;
      for (j=0; j < nro_obj_simu;j++)  /* OBJETOS A */
          {
           if (i != j)
              {
               obj_grao = objetos + j;  /* OBJETO A */
               for (k=0;k < obj_tes->n_vertices;k++)  /* VERT. B */
                   {
                    Houve_Colisao = Ponto_B_Inside_Obj_A(obj_tes->p_vert,
                              1,k,obj_grao,obj_tes,step_atu);
                    if (Houve_Colisao)  return TRUE;
                   }
              }
          }
     }
}


/*-------------------------------------------------------------------+
|    Second_Step_3D_Cyrus_Beck                                       |
|    Funcao: Testar a presenca de arestas do obj. B dentro das faces |
|            do obj. A                                               |
|    Parametros: int step_atu  (Passo de tempo)                      |
|    Globais : ---*---                                               |
|    Vlr. Retorno:  bool TRUE (Condicao p/ aresta dentro)            |
|                   bool FALSE (Condicao p/ aresta fora)             |
+-------------------------------------------------------------------*/

bool Second_Step_3D_Cyrus_Beck(step_atu)
int step_atu;
{
 int i,j,nro_visitas,k,l,m,n,ind,ind_i,ind_j,o,nro_pares,cont;
 C_OBJETO *obj_tes,*obj_grao;
 C_PTS_INTER *p_pts,*p_pts_corr;
 C_XYZ vert_i,vert_j,vet1,vet2,*p_pts_medios;
 double dis_i,dis_j;
 bool Houve_Colisao = FALSE;

 /* NRO. DE TESTES PARA 1 ARESTA */

 nro_visitas = objetos->n_faces * objetos->p_faces[0].n_vert;
 p_pts = Aloca_PTS_INTER(nro_visitas);
```

```
for (i=0;i < nro_obj_simu;i++)   /* OBJETO B */
    {
     obj_tes = objetos + i;
     for (j=0;j < nro_obj_simu;j++)   /* OBJETO A  */
         {
          if (i != j)
             {
              obj_grao = objetos + j;
              for (k=0;k < obj_tes->n_faces;k++)   /* OBJ. B  */
                  {
                   for (l=0;l < obj_tes->p_faces[k].n_vert;l++) /* OBJ. B */
                       {
                        ind_i = obj_tes->p_faces[k].p_ind[l];
                        Monta_VertB_2_SRA(&vert_i,&obj_tes->p_vert[ind_i],
                                          obj_tes,obj_grao,step_atu);
                        if (l + 1 == obj_tes->p_faces->n_vert)
                           {
                            ind_j = obj_tes->p_faces[k].p_ind[0];
                            Monta_VertB_2_SRA(&vert_j,&obj_tes->p_vert[ind_j],
                                      obj_tes,obj_grao,step_atu);
                           }
                        else
                           {
                            ind_j = obj_tes->p_faces[k].p_ind[l + 1];
                            Monta_VertB_2_SRA(&vert_j,&obj_tes->p_vert[ind_j],
                                              obj_tes,obj_grao,step_atu);
                           }
                        p_pts_corr = p_pts;
                        cont = 0;
                        for (m=0;m < obj_grao->n_faces;m++)
                            {
                             for (n=0;n < obj_grao->p_faces[m].n_vert;n++)
                                 {
                                  ind = obj_grao->p_faces[m].p_ind[n];
                                  vet1.x = vert_i.x -
                                           obj_grao->p_vert[ind].x;
                                  vet1.y = vert_i.y -
                                           obj_grao->p_vert[ind].y;
                                  vet1.z = vert_i.z -
                                           obj_grao->p_vert[ind].z;
                                  vet2.x = vert_j.x -
                                           obj_grao->p_vert[ind].x;
                                  vet2.y = vert_j.y -
                                           obj_grao->p_vert[ind].y;
                                  vet2.z = vert_j.z -
```

```
                        obj_grao->p_vert[ind].z;
                Produto_Escalar_2V(&vet1,
                      &obj_grao->p_faces[m].normal,&dis_i);
                Produto_Escalar_2V(&vet2,
                      &obj_grao->p_faces[m].normal,&dis_j);
                if ((dis_i < 0.0 && dis_j < 0.0) ||
                    (dis_i >= 0.0 && dis_j >= 0.0))
                    {
                     p_pts_corr->t = 1.0;
                     p_pts_corr->ponto.x = 1.0;
                     p_pts_corr->ponto.y = 1.0;
                     p_pts_corr->ponto.z = 1.0;
                    }
                else
                    {
                     if (dis_i < 0.0) dis_i = dis_i * -1.0;
                     if (dis_j < 0.0) dis_j = dis_j * -1.0;
                     p_pts_corr->t = dis_i /
                                     (dis_i + dis_j);
                     p_pts_corr->ponto.x = vert_i.x +
                     (p_pts_corr->t * (vert_j.x-vert_i.x));
                     p_pts_corr->ponto.y = vert_i.y +
                     (p_pts_corr->t * (vert_j.y-vert_i.y));
                     p_pts_corr->ponto.z = vert_i.z +
                     (p_pts_corr->t * (vert_j.z-vert_i.z));
                    }
                cont++;
                if (cont < nro_visitas)
                      p_pts_corr = p_pts + cont;
               }
          }
Classifica_Ts(p_pts,nro_visitas);
/* Descartar pontos que nao pertencem a aresta  */
o = 0;
p_pts_corr = p_pts;
while (p_pts_corr->t != 1.0 && o < nro_visitas)
      {
       o++;
       if (o < nro_visitas)
          p_pts_corr = p_pts + o;
      }
if (o != 0)
   {
    nro_pares = o + 1;
    p_pts_medios =  Aloca_Vetor_XYZ(nro_pares);
```

```
                                Calcula_PTS_Medios(p_pts_medios,&vert_i,
                                            &vert_j,p_pts,nro_pares);
                                Houve_Colisao = Ponto_B_Inside_Obj_A(p_pts_medios,
                                          nro_pares,0,obj_grao,obj_tes,step_atu);
                                if (Houve_Colisao)  return TRUE;
                               }
                            }

                        }
                    }
                }
            }
 Free_PTS_INTER(p_pts);
 return FALSE;
}


/*-------------------------------------------------------------------------+
|    Third_Step_3D_Cyrus_Beck                                              |
|    Funcao: Testar a presenca do centroide das faces do obj. B dentro     |
|            das faces  do obj. A                                          |
|    Parametros: int step_atu  (Passo de tempo)                            |
|    Globais: ---*---                                                      |
|    Vlr. Retorno:  bool TRUE (Condicao p/ centroide dentro)               |
|                   bool FALSE (Condicao p/ centroide fora)                |
+-------------------------------------------------------------------------*/

bool Third_Step_3D_Cyrus_Beck(step_atu)
int step_atu;
{
 int i,j,k;
 C_OBJETO *obj_tes,*obj_grao;
 bool Houve_Colisao = FALSE;

 for (i=0;i < nro_obj_simu;i++)   /* OBJETOS B */
     {
      obj_tes = objetos + i;
      for (j=0;j < nro_obj_simu;j++)
          {
           if (i != j)
              {
               obj_grao = objetos + j;      /* OBJETO A */
               for (k=0;k < obj_tes->n_faces;k++)
                   {
                    Houve_Colisao = Ponto_B_Inside_Obj_A(obj_tes->
```

```
                                        p_faces[k].centroide,
                                        1,0,obj_grao,obj_tes,step_atu);
                    if (Houve_Colisao)
                       return TRUE;
                   }
               }
           }
      }
 return FALSE;
}

/*-------------------------------------------------------------------+
|    Deteccao_Colisao                                                |
|    Funcao: Realizar a deteccao de colisoes entre pares de objetos  |
|            poliedricos                                             |
|    Parametros: int step_corr (Passo atual)                         |
|    Globais: ---*---                                                |
|    Globais: ---*---                                                |
|    Vlr. Retorno:  bool TRUE (Condicao p/ ocorrencia de colisao)    |
|                   bool FALSE (Condicao p/ nao ocorrencia de colisao)|
+-------------------------------------------------------------------*/

bool Deteccao_Colisao(step_corr)
int step_corr;
{
 bool First_Step_Check, /* 1 Houve Colisao 0 Nao Houve Colisao */
      Second_Step_Check,
      Third_Step_Check;
if ((First_Step_Check = First_Step_3D_Cyrus_Beck(step_corr)) == TRUE)
   return TRUE;
if ((Second_Step_Check = Second_Step_3D_Cyrus_Beck(step_corr)) == TRUE)
   return TRUE;
if ((Third_Step_Check = Third_Step_3D_Cyrus_Beck(step_corr)) == TRUE)
   return TRUE;
return FALSE;
}



/*-------------------------------------------------------------------+
|                              dinamica.c                            |
|--------------------------------------------------------------------|
|    Modulo: dinamica                                                |
|    Descricao:  Este e o modulo principal do prototipo p/ Animacao e |
```

```
|                Tratamento de Colisoes de Corpos Rigidos Utilizando    |
|                Analise Dinamica                                       |
|    Autor: Robson R. Lemos                                             |
+-----------------------------------------------------------------------*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "estruda.h"
#include "varglo.h"

/*-----------------------------------------------------------------------+
|    Aloca_Areas_Trabalho                                               |
|    Funcao:  Aloca areas de trabalho  utilizadas no metodo numerico    |
|             e as areas de resultados destas funcoes durante a         |
|             simulacao ao longo do tempo                               |
|    Parametros: ---*---                                                |
|    Globais: int nstep_simu                                            |
|    Vlr. Retorno: ---*---                                              |
+-----------------------------------------------------------------------*/

void Aloca_Areas_Trabalho()
{
 /* Resultados de funcoes de tempo (odeint) */
 /*Aloca_Vector(nstep_simu,&trans_x_xp);
 Aloca_Vector(nstep_simu,&trans_y_xp);
 Aloca_Vector(nstep_simu,&trans_z_xp);
 Aloca_Vector(nstep_simu,&ori_xp);*/
 /* Resultados de funcoes de translacao */
 Aloca_Matrix(N_ODES_TRANS,nstep_simu,&obj_corr->fun_trans_all_x);
 Aloca_Matrix(N_ODES_TRANS,nstep_simu,&obj_corr->fun_trans_all_y);
 Aloca_Matrix(N_ODES_TRANS,nstep_simu,&obj_corr->fun_trans_all_z);
 /* Resultados de funcoes de orientacao matriz de orientacao */
 Aloca_Matrix(N_ODES_ORIEN,nstep_simu,&obj_corr->fun_ang_ori_all);
}

/*-----------------------------------------------------------------------+
|    Free_Areas_Trabalho                                                |
|    Funcao:  Libera areas de trabalho p/ as funcoes iniciais utilizadas|
|             no metodo numerico e as areas de resultados destas        |
|             funcoes durante a simulacao ao longo do tempo             |
|    Parametros:  ---*---                                               |
|    Globais:  ---*---                                                  |
|    Vlr. Retorno:  ---*---                                             |
```

```
+-------------------------------------------------------------------*/

void Free_Areas_Trabalho()
{
 int i;
 Free_Vector(tempo_sincro);
 /* Libera funcoes de tempo (odeint) */
 /*Free_Vector(trans_x_xp);
 Free_Vector(trans_y_xp);
 Free_Vector(trans_z_xp);
 Free_Vector(ori_xp);*/
 for (i=0;i<nro_obj_simu;i++)
     {
      obj_corr = objetos + i;
      /* Libera  resultados de funcoes de translacao */
      Free_Matrix(N_ODES_TRANS,obj_corr->fun_trans_all_x);
      Free_Matrix(N_ODES_TRANS,obj_corr->fun_trans_all_y);
      Free_Matrix(N_ODES_TRANS,obj_corr->fun_trans_all_z);
      /* Libera  resultados de funcoes de orientacao */
      Free_Matrix(N_ODES_ORIEN,obj_corr->fun_ang_ori_all);
     }
 Free_Objetos();
}

/*-------------------------------------------------------------------+
|    Derivs                                                          |
|    Funcao:   Calcula derivada de funcoes p/ o movimento translacional,|
|              movimento rotacional e orientacao do corpo rigido no es- |
|              paco (x,y,z)                                          |
|    Parametros:  int tipo (Tipo da funcao)                          |
|                 double  t (Tempo da funcao)                        |
|                 double *funcoes (Valores iniciais da funcao)       |
|                 double *deriv_out (Valores da derivada resultante) |
|    Globais:  ---*---                                               |
+-------------------------------------------------------------------*/

void Derivs(tipo, t, funcoes, deriv_out)
int tipo;
double t,
       *funcoes,
       *deriv_out;
{
 switch(tipo)
       {
        case T_TRANS_X :    /* dvx/dt */
```

```
                        if (t == 0.0)
                           deriv_out[0] = obj_corr->all_forca.x /
                                          obj_corr->massa;
                        else
                           deriv_out[0] = 0.0;
                        /* dcx/dt */
                        deriv_out[1] = funcoes[0];
                        break;
case T_TRANS_Y :        /* dvy/dt */
                        if (t == 0.0)
                           deriv_out[0] = obj_corr->all_forca.y /
                                          obj_corr->massa;
                        else
                           deriv_out[0] = a_gravidade
                        /* dcy/dt */
                        deriv_out[1] = funcoes[0];
                        break;
case T_TRANS_Z :        /* dvz/dt */
                        if (t == 0.0)
                           deriv_out[0] = obj_corr->all_forca.z /
                                          obj_corr->massa;
                        else
                           deriv_out[0] = 0.0;
                        /* dcz/dt */
                        deriv_out[1] = funcoes[0];
                        break;
case T_ORIENTACAO :
                        /* dwx/dt */
                        if (t == 0.0)
                           deriv_out[0] = (obj_corr->all_torque.x -
                           (obj_corr->tensor.z * funcoes[1] * funcoes[2] -
                           obj_corr->tensor.y * funcoes[1] * funcoes[2] )) /
                           obj_corr->tensor.x;
                        else
                           deriv_out[0] = 0.0;
                        /* dwy/dt */
                        if (t == 0.0)
                           deriv_out[1] = (obj_corr->all_torque.y -
                           (obj_corr->tensor.x * funcoes[0] * funcoes[2] -
                           obj_corr->tensor.z * funcoes[0] * funcoes[2] )) /
                           obj_corr->tensor.y;
                        else
                           deriv_out[1] = 0.0;
                        /* dwz/dt */
                        if (t == 0.0)
```

```
                                deriv_out[2] = (obj_corr->all_torque.z -
                                (obj_corr->tensor.y * funcoes[0] * funcoes[1] -
                                obj_corr->tensor.x * funcoes[0] * funcoes[1] )) /
                                obj_corr->tensor.z;
                             else
                                deriv_out[2] = 0.0;
                             /* Estrategia de James  k. Hahn */
                             /*d_ang_x/dt */
                             deriv_out[3] = funcoes[0];
                             /*d_ang_y/dt */
                             deriv_out[4] = funcoes[1];
                             /*d_ang_x/dt */
                             deriv_out[5] = funcoes[2];
    break;
       }
}

/*-------------------------------------------------------------------+
|    Runge_Kutta_Fourth                                              |
|    Funcao:   Metodo de Runge_Kutta de Quarta Ordem p/ o calculo de |
|              Equacoes diferenciais Ordinarias a partir da Informacao|
|              x0, y0 e dy0/dx0 p/ funcoes do tipo y' = f(x,y)       |
|              onde atraves da aplicacao do metodo e possivel encontrar|
|              um novo yn em xn                                      |
|    Parametros:  double *y  (Vetor contendo valores de y's iniciais)|
|                 double *dydx  (Vetor contendo derivadas iniciais)  |
|                 double x  ( Instante de Tempo)                     |
|                 double *yout (Vetor contendo valores yn's resultantes)|
|                 int n  ( Nro de ODE's)                             |
|                 int tipo_equa (Tipo de ODE's)                      |
|    Globais:   ---*---                                              |
+-------------------------------------------------------------------*/

void Runge_Kutta_Fourth(y,dydx,n,x,h1,yout,tipo_equa)
double *y,
       *dydx,
       x,
       h1,
       *yout;
int n,
    tipo_equa;
{
 int i;
 double xh,        /* Proximo instante de tempo */
        hh,        /* 1/2 * h */
```

```
        h6,         /* 1/6 * h */
        *dym,       /* Vetor Derivs third step e depois second step
                       + third step */
        *dyt,       /* Vetor Derivs second step e depois fourth step */
        *yt;        /* Vetor Y's temporarios  */
 Aloca_Vector(n,&dym);
 Aloca_Vector(n,&dyt);
 Aloca_Vector(n,&yt);
 hh = 0.5 * h1;
 h6 = h1 / 6.0;
 xh = x + hh;
 for (i=0; i<n; i++)
     yt[i] = y[i] + hh * dydx[i];                    /* first step */
 Derivs(tipo_equa,xh,yt,dyt);                        /* second step */
 for (i=0; i<n; i++)
     yt[i] = y[i] + hh * dyt[i];
 Derivs(tipo_equa,xh,yt,dym);                        /* third step */
 for (i=0; i<n; i++)
     {
      yt[i] = y[i] + h1 * dym[i];
      dym[i] += dyt[i];
     }
 Derivs(tipo_equa,x + h1,yt,dyt);                    /* fourth step */
 for (i=0; i<n; i++)                                 /* Yn's Resultantes  */
     yout[i] = y[i] + h6 * (dydx[i] + dyt[i] + 2.0 * dym[i]);
 Free_Vector(dym);
 Free_Vector(dyt);
 Free_Vector(yt);
}

/*-------------------------------------------------------------------------+
|    Rk_Drivers                                                            |
|    Funcao:  Rotina p/ realizar a integracao de Equacoes                  |
|             Diferenciais Ordinarias, armazenar resultados intermedia-    |
|             rios e atualizar informacoes a partir dos valores iniciais   |
|             vstart[0...nvar] p/ nvar funcoes conhecidas em um tempo      |
|             t0                                                           |
|    Parametros:  double *vstart  (Vetor de valores Iniciais de funcoes)   |
|                 double **p_matrix (Matriz de Valores resultantes)        |
|                 int nvar  (Nro. de ODE's)                                |
|                 int tipo_equa (Tipo de ODE's)                            |
|                 int step_corr (Nro. Passo de Tempo Corrente)             |
|                 double n_corr (Passo de Tempo Corrente)                  |
|                 double tempo_corr (Tempo Corrente)                       |
|    Globais:   ---*---                                                    |
```

```
|    Vlr. Retorno:  double **p_matrix (Pointer p/ matriz c/ resultados) |
+-------------------------------------------------------------------*/

void Rk_Drivers(vstart,nvar,tipo_equa,step_corr,h_corr,tempo_corr,p_matrix)
double *vstart,
       **p_matrix,
       h_corr,
       tempo_corr;
int nvar,
    step_corr,
    tipo_equa;
{
 int i;
 double x,          /* Variavel tempor. p/ tempo  */
        *v,         /* Vetor tempor. p/ funcoes   */
        *vout,      /* Vetor tempor. p/ funcoes   */
        *dv;        /* Vetor tempor. p/ derivadas */
 Aloca_Vector(nvar,&v);
 Aloca_Vector(nvar,&vout);
 Aloca_Vector(nvar,&dv);
 for (i=0; i<nvar; i++)           /* Inicializa Vetores  */
     {
      v[i] = vstart[i];
     }
 x = tempo_corr;
 Derivs(tipo_equa,tempo_corr,v,dv);
 Runge_Kutta_Fourth(v,dv,nvar,tempo_corr,h_corr,vout,tipo_equa);
 if ((double) (tempo_corr + h_corr) == x)
     printf("Step size too small in routine Rk_Drivers \n");
 for (i=0; i<nvar; i++)
     {
      p_matrix[i][step_corr] = vout[i];
     }
 Free_Vector(v);
 Free_Vector(vout);
 Free_Vector(dv);
}

/*-------------------------------------------------------------------+
|    rkqc                                                            |
|    Funcao:   Metodo de runge_Kutta de Quinta Ordem p/ o calculo de |
|              Equacoes diferenciais Ordinarias a partir da Informacao |
|              x0, y0 e dy0/dx0 p/ funcoes do tipo y' = f(x,y)       |
|              onde atraves da aplicacao do metodo e possivel encontrar |
|              um novo yn em xn com controle de erro de trunc. local |
```

```
|    Parametros:  double *y  (Vetor contendo valores de y's iniciais)   |
|                 double *dydx  (Vetor contendo derivadas iniciais)     |
|                 int n  ( Nro de ODE's)                                |
|                 double *x  ( Instante de Tempo)                       |
|                 double htry                                           |
|                 double eps                                            |
|                 double *yscal                                         |
|                 double *hdid                                          |
|                 double *hnext                                         |
|                 int tipo_equa (Tipo de ODE's)                         |
|    Globais:   ---*---                                                 |
|    Vlr. Retorno:  double *yout (Pointer p/ vetor c/ resultados)       |
+-----------------------------------------------------------------------*/

void rkqc(y,dydx,n,x,htry,eps,yscal,hdid,hnext,tipo_equa)
double *y, /* eh substituido por um novo valor */
       *dydx,
       *x, /* eh substituido por um novo valor */
       htry,
       *yscal,
       *hdid, /* sao calculados e retornados */
       *hnext,
       eps;
int n;
{
 int i;
 double xsav,
        hh,
        h1,
        temp,
        errmax,
        delta_exat,
        deltas_div,
        *dysav,
        *ysav,
        *ytemp;
 Aloca_Vector(n,&dysav);
 Aloca_Vector(n,&ysav);
 Aloca_Vector(n,&ytemp);
 xsav = (*x);     /* salva valores iniciais */
 for (i=0;i<n;i++)
     {
      ysav[i] = y[i];
      dysav[i] = dydx[i];
     }
```

```
h1 = htry;  /* seta passo inicial p/ prim. passagem */
for (;;)
    {
     hh = 0.5 * h1;    /* ter dois meios passos */
     Runge_Kutta_Fourth(ysav,dysav,n,xsav,hh,ytemp,tipo_equa);
     *x = xsav + hh;
     Derivs(tipo_equa,*x,ytemp,dydx);
     Runge_Kutta_Fourth(ytemp,dydx,n,*x,hh,y,tipo_equa);
     *x = xsav + h1;
     if (*x == xsav)
        printf("Step size too small in routine rkqc\n");
     /* ter um passo maior */
     Runge_Kutta_Fourth(ysav,dysav,n,xsav,h1,ytemp,tipo_equa);
     errmax = 1.0E-8;
     for (i=0;i<n;i++)
         {
          ytemp[i] = y[i] - ytemp[i]; /* ytemp agora contem o erro estimado */
          /*temp = fabs(ytemp[i] / yscal[i]);*/
          temp = fabs(ytemp[i]);
          if (errmax < temp)
             errmax = temp;
         }
     delta_exat = 1.0E-8;
     for (i=0;i<n;i++)
         {
          temp = fabs(eps * yscal[i]);
          if (delta_exat < temp)
             delta_exat = temp;
         }
      deltas_div = delta_exat / errmax;
      /*errmax /= eps;*/  /* escala relativa p/ tolerancia requerida */
      if (delta_exat >= errmax)
        {
         *hdid = h1;
         if (errmax > ERRCON)
            {
             *hnext = SAFETY * h1 * exp(PGROW * log(deltas_div));
             printf("Aumentando um pouco o h\n");
            }
         else
            *hnext = 4.0 * h1;
         /**hnext = (errmax > ERRCON) ?
           (SAFETY * h1 * exp(PGROW * log(errmax))) : (4.0 * h1);*/
         break;
        }
```

```
      h1 = SAFETY * h1 * exp(PSHRNK * log(deltas_div));
      printf("Diminuindo um pouco o h\n");
      /* erro de truncamento muito largo, reduzir o tamanho do passo */
     }
 for (i=0;i<n;i++)
     y[i] += ytemp[i] * FCOR;
     /* MOP UP FIFTH-ORDER TRUNCATION ERROR */
 Free_Vector(ytemp);
 Free_Vector(dysav);
 Free_Vector(ysav);
}

/*-------------------------------------------------------------------+
|    odeint                                                          |
|    Funcao:  Rotina p/ iniciar e parar a integracao de equacoes     |
|             Diferenciais Ordinarias, armazena resultados intermedia-|
|             rios e atualiza informacoes a partir dos valores iniciais|
|             vstart[0...nvar] p/ nvar funcoes conhecidas em um tempo |
|             t0. Inicia  o processo de integracao p/ o metodo       |
|             Runge Kutta de Quinta Ordem com controle de passo de tempo|
|             adaptativo                                             |
|    Parametros:  double *ystart  (Vetor de valores Iniciais de funcoes)|
|                 int nvar  (Nro. de ODE's)                          |
|                 double eps (Exatidao requerida)                    |
|                 double h1 (Stepsize inicial)                       |
|                 double hmin (Stepsize minimo)                      |
|                 int tipo_equa (Tipo de ODE's)                      |
|                 int nok (Nro. de passos ok)                        |
|                 int nbad (Nro. de passos nao ok)                   |
|                 double *p_vector (Vetor de valor. de tempos resul. |
|                 double **p_matrix (Matriz de Valores resultantes)  |
|    Globais:   int kmax                                             |
|               int nstep_simu                                       |
|               double time_start                                    |
|               double time_finish                                   |
|               double *p_vector                                     |
|               double **p_matrix                                    |
|    Vlr. Retorno:  double **p_matrix (Pointer p/ matriz c/ resultados)|
|                   double *p_vector (Pointe p/ vetor c/ resul. tempos |
+-------------------------------------------------------------------*/

void odeint(ystart,nvar,eps,h1,hmin,tipo_equa,nok,nbad,p_vector,p_matrix)
double *ystart, /* y's iniciais */
       *p_vector, /* vetor com tempos armazenados */
       **p_matrix, /* matriz c/ resultados */
```

```
        eps,    /* exatidao requerida */
        h1,     /* Stepsize desejado  */
        hmin;   /* Stepsize minimo    */
int     nvar,
        *nok,
        *nbad,
        tipo_equa;
{
 int nstep,    /* nro. do passo  */
     i,        /* indices  */
     kount;
 double xsav,
        x,     /* passo de tempo corrente  */
        hnext, /* calculado por rkqc */
        hdid,
        ha,    /* h corrente  */
        *yscal, /* y utilizado p/ avaliar o erro */
        *y,     /* y */
        *dydx,  /* derivadas ao longo do tempo */
        dxsav = 0.0;
 Aloca_Vector(nvar,&yscal);
 Aloca_Vector(nvar,&y);
 Aloca_Vector(nvar,&dydx);
 x = time_start;
 ha = (time_finish > time_start) ? fabs(h1) : -fabs(h1);
 (*nok) = (*nbad) = kount = 0;
 for (i=0;i<nvar;i++)
     {
      y[i] = ystart[i];
      p_matrix[i][kount] = ystart[i];
     }
 p_vector[kount++] = x;
 if (kmax > 0)
    xsav = x - dxsav * 2.0;      /* Assegura arm. no prim. passo */
 for (nstep=1; nstep<MAXSTP; nstep++)
     {
      Derivs(tipo_equa,x,y,dydx);
      for (i=0;i<nvar;i++)
          yscal[i] = fabs(y[i]) + fabs(dydx[i] * ha) + TINY;
      if (kmax > 0)
         {
          if (fabs(x - xsav) > fabs(dxsav)) /* Armaz. Resultados
                                               Intermediarios */
             {
              if (kount < kmax)
```

```
                {
                 p_vector[kount] = x;
                 for (i=0;i<nvar;i++)
                     p_matrix[i][kount] = y[i];
                 kount++;
                 xsav = x;
                }
            }
         }
      if ((x + ha -time_finish) * (x + ha - time_start) > 0.0)
         ha = time_finish - x;
      rkqc(y,dydx,nvar,&x,ha,eps,yscal,&hdid,&hnext,tipo_equa);
      if (hdid == ha)
         ++(*nok);
      else
         ++(*nbad);
      if ((x - time_finish) * (time_finish - time_start) >= 0.0)
         {                                    /* Are we done? */
          for (i=0;i<nvar;i++)
              ystart[i] = y[i];
          if (kmax)                           /* Save final step */
             {
              p_vector[kount] = x;
              for (i=0;i<nvar;i++)
                  p_matrix[i][kount] = y[i];
             }
          Free_Vector(dydx);
          Free_Vector(y);
          Free_Vector(yscal);
          return;                            /* Normal exit */
         }
      if (fabs(hnext) <= hmin)
         printf("Step size too small in odeint \n");
      ha = hnext;
     }
 printf("Too many steps in routine odeint \n");
}

/*-------------------------------------------------------------------+
|    Inicializa_Modelo                                               |
|    Funcao:  Inicializa os valores p/ funcoes  utilizadas no metodo |
|             numerico durante a simulacao ao longo do tempo         |
|    Parametros:  C_ATRIBUTOS *p_in                                  |
|    Globais:  int nstep_simu                                        |
|              C_OBJETO obj_corr                                     |
```

```
|              double time_start                                          |
|              double h                                                   |
|    Vlr. Retorno:  ---*---                                               |
+-------------------------------------------------------------------------*/
void Inicializa_Modelo(p_in)
C_ATRIBUTOS *p_in;
{
 double t,
        *fun_comp;
 int i;
 C_XYZ *p_forca_out_sru,
       *p_torque_aux;
 /* Inicializa funcoes p/ veloc. angul. */
 obj_corr->fun_ang_ori_all[0][0] = p_in->vel_angular.x;
 obj_corr->fun_ang_ori_all[1][0] = p_in->vel_angular.y;
 obj_corr->fun_ang_ori_all[2][0] = p_in->vel_angular.z;
 /* Inicializa funcoes p/ ang. de orient. */
 obj_corr->fun_ang_ori_all[3][0] = p_in->ang_orientacao.x;
 obj_corr->fun_ang_ori_all[4][0] = p_in->ang_orientacao.y;
 obj_corr->fun_ang_ori_all[5][0] = p_in->ang_orientacao.z;
 Aloca_Vector(N_COMPONENTES,&fun_comp);
 Atualiza_Componentes(fun_comp,&p_in->ang_orientacao);
 Mudanca_Base_SRO_2_SRU(&p_in->vel_linear,&p_in->vel_linear,fun_comp,
                        &p_in->posicao);
 /* Inicializa funcoes de translacao */
 obj_corr->fun_trans_all_x[0][0] = (p_in->vel_linear.x - p_in->posicao.x);
 obj_corr->fun_trans_all_x[1][0] = p_in->posicao.x;
 obj_corr->fun_trans_all_y[0][0] = (p_in->vel_linear.y - p_in->posicao.y);
 obj_corr->fun_trans_all_y[1][0] = p_in->posicao.y;
 obj_corr->fun_trans_all_z[0][0] = (p_in->vel_linear.z - p_in->posicao.z);
 obj_corr->fun_trans_all_z[1][0] = p_in->posicao.z;
 /* Sentido do torque PONTO_FORCA_SRO -> PONTO_OBJ_SRO */
 for (i=0;i<p_in->nro_torques_ext;i++)
     {
      p_in->p_forca_out_cm[i].x =  p_in->p_radius[i].x -
                 p_in->p_forca_out_cm[i].x;
      p_in->p_forca_out_cm[i].y =  p_in->p_radius[i].y -
                 p_in->p_forca_out_cm[i].y;
      p_in->p_forca_out_cm[i].z = p_in->p_radius[i].z -
                 p_in->p_forca_out_cm[i].z;
     }
 p_forca_out_sru = Aloca_Vetor_XYZ(p_in->nro_torques_ext);
 /* Mudanca de Sist. Refer. P_FORCA_OUT_SRU (SRU) <- P_FORCA_OUT_CM (SRO) */
 Mudanca_Base_N_SRO_2_SRU(p_forca_out_sru,p_in->p_forca_out_cm,fun_comp,
                          p_in->posicao,p_in->nro_torques_ext);
```

```
/* Calcula as forcas externas resultantes totais p/ o objeto corrente no SRU */
/* de acordo com o Principio de Euler de Sobreposicao     */
for (i=0;i<p_in->nro_torques_ext;i++)
    {
     obj_corr->all_forca.x += (p_forca_out_sru[i].x -  p_in->posicao.x);
     obj_corr->all_forca.y += (p_forca_out_sru[i].y -  p_in->posicao.y);
     obj_corr->all_forca.z += (p_forca_out_sru[i].z -  p_in->posicao.z);
    }
Free_Vetor_XYZ(p_forca_out_sru);
/* Mudanca de Sist. Refer. P_FORCA_EXT (SRU) <- P_FORCA_EXT (SRO)*/
Mudanca_Base_N_SRO_2_SRU(p_in->p_forca_ext,p_in->p_forca_ext,fun_comp,
                         p_in->posicao,p_in->nro_forca_ext);
/* Calcula as forcas externas totais p/ o objeto corrente */
/* Sentido da forca externa PONTO_FORCA (SRU) -> CENTRO DE MASSA */
for (i=0;i<p_in->nro_forca_ext;i++)
    {
     obj_corr->all_forca.x += (p_in->p_forca_ext[i].x -  p_in->posicao.x);
     obj_corr->all_forca.y += (p_in->p_forca_ext[i].y -  p_in->posicao.y);
     obj_corr->all_forca.z += (p_in->p_forca_ext[i].z -  p_in->posicao.z);
    }
/* Calcula a forca externa total em y considerando a acel. gravidade */
obj_corr->all_forca.y += (obj_corr->massa *
                          a_gravidade);
/* Calcula os torques externos totais p/ o objeto corrente SRO */
p_torque_aux = Aloca_Vetor_XYZ(1);
for (i=0;i<p_in->nro_torques_ext;i++)
    {
     Produto_Vetorial(&p_in->p_radius[i],&p_in->p_forca_out_cm[i],
                      &p_torque_aux[0]);
     obj_corr->all_torque.x += p_torque_aux[0].x;
     obj_corr->all_torque.y += p_torque_aux[0].y;
     obj_corr->all_torque.z += p_torque_aux[0].z;
    }
 Free_Vetor_XYZ(p_torque_aux);
 Free_Vector(fun_comp);
}

/*-------------------------------------------------------------------+
|    Le_Modelo_Fisico                                                |
|    Funcao:  Realiza a leitura  de arquivo de formato padrao (.mod) |
|             de um modelo fisico                                    |
|    Parametros:  ---*---                                            |
|    Globais:   double time_start                                    |
|               double time_finish                                   |
|               double a_gravidade                                   |
```

```
|               C_OBJETO objetos                                          |
|               C_OBJETO obj_corr                                         |
|               int nro_obj_simu                                          |
|               int nstep_simu                                            |
|               int kmax                                                  |
|               char nome_modelo                                          |
|    Vlr. Retorno:  ---*---                                               |
+-------------------------------------------------------------------------*/

void Le_Modelo_Fisico()
{
 char token[30];
 int i,j;
 FILE *fp1;
 C_ATRIBUTOS *p_atr;
 if ((fp1 = fopen(nome_modelo,"rt")) == NULL)
    {
     printf("Erro na abertura de arquivo (.mod) \n");
     exit(0);
    }
 /* $SIMULACAO */
 fscanf(fp1,"%s",token);
 /* $TEMPO  <tempo inicial>  <tempo final> */
 fscanf(fp1,"%s %lf %lf",token,&time_start,&time_finish);
 /* Calcula nro. de passos para Simulacao */
 nstep_simu = (time_finish - time_start) * NRO_FRAMES_SEGS;
 kmax = nstep_simu;
 Aloca_Vector(nstep_simu,&tempo_sincro);
 tempo_sincro[0] = time_start;
 /* $NRO_OBJETOS  <nro. objetos> */
 fscanf(fp1,"%s %d",token,&nro_obj_simu);
 Aloca_Objetos();
 /* Realiza leitura de $OBJETO ... $ENDOBJ */
 for (i=0;i<nro_obj_simu;i++)
     {
      p_atr = Aloca_Atributos();
      /* Atualiza pointer p/ objeto corrente  */
      obj_corr = objetos+i;
      /* $OBJETO  <tipo de objeto> */
      fscanf(fp1,"%s %d",token,&obj_corr->shape);
      /* $NOME <nome do objeto> */
      fscanf(fp1,"%s %s",token,obj_corr->nome);
      Le_Objeto_Geometrico();
      /* $DENSIDADE  <densidade de massa>  */
      fscanf(fp1,"%s %lf",token,&p_atr->densidade);
```

```
/*  Calcula a massa do objeto corrente */
/* masso.obt  */
/*obj_corr->massa = p_atr->densidade *8.99999999999999921E-05;*/
/* cig.obt (paralelepipedo) */
obj_corr->massa = p_atr->densidade * 32.0;
/* (paralelepipedo)  */
/*obj_corr->massa = p_atr->densidade * 250000.0;*/
/* cigarro.obt */
/*obj_corr->massa =  p_atr->densidade * 8.5 * M_PI * 0.25;*/
/* Calcula os tensores de inercia do objeto corrente */
/* masso.obt  */
/*obj_corr->tensor.x = (1.0/12.0) * obj_corr->massa *
        (0.00040000000000000019 + 0.00809999999999999956);
obj_corr->tensor.y = (1.0/12.0) * obj_corr->massa *
        (0.00040000000000000019 +  0.00250000000000000049);
obj_corr->tensor.z = (1.0/12.0) * obj_corr->massa *
        (0.00809999999999999956 + 0.00250000000000000049);*/
/* cig.obt (paralelepipedo) */
obj_corr->tensor.x = (1.0/12.0) * obj_corr->massa *
        (4.0 + 64.0);
obj_corr->tensor.y = (1.0/12.0) * obj_corr->massa *
        (4.0 + 4.0);
obj_corr->tensor.z = (1.0/12.0) * obj_corr->massa *
        (64.0 + 4.0);
/* (paralelepipedo) */
/*obj_corr->tensor.x = (1.0/12.0) * obj_corr->massa *
        (2500.0 + 10000.0);
obj_corr->tensor.y = (1.0/12.0) * obj_corr->massa *
        (2500.0 + 2500.0);
obj_corr->tensor.z = (1.0/12.0) * obj_corr->massa *
        (1000.0 + 2500.0);*/
/* cigarro.obt (cilindro) */
/*obj_corr->tensor.x = (1.0/12.0) * obj_corr->massa *
        (64.0 + 3.0);
obj_corr->tensor.y = (1.0/12.0) * obj_corr->massa *
        (64.0 + 3.0);
obj_corr->tensor.z = (1.0/12.0) * obj_corr->massa *
        (64.0 + 3.0);*/
/* $POSICAO_SRU  <x>  <y>  <z>  */
fscanf(fp1,"%s %lf %lf %lf",token,&p_atr->posicao.x,
  &p_atr->posicao.y,
  &p_atr->posicao.z);
/* $VEL_LINEAR_SRO  <x>  <y>  <z>  */
fscanf(fp1,"%s %lf %lf %lf",token,&p_atr->vel_linear.x,
  &p_atr->vel_linear.y,
```

```
  &p_atr->vel_linear.z);
/* $NRO_FORCA_EXT  <nro. forcas externas> */
fscanf(fp1,"%s %d",token,&p_atr->nro_forca_ext);
/* Aloca areas p/ forcas externas utilizadas no objeto corrente */
p_atr->p_forca_ext = Aloca_Vetor_XYZ(p_atr->nro_forca_ext);
/* $FORCA_EXT_SRO  <x> <y> <z>  */
for (j=0;j<p_atr->nro_forca_ext;j++)
    fscanf(fp1,"%s %lf %lf %lf",token,&p_atr->p_forca_ext[j].x,
      &p_atr->p_forca_ext[j].y,
      &p_atr->p_forca_ext[j].z);
/* $ORIENT_SRU_SRO <angulo rot. x> <angulo rot. y> <angulo rot. z>   */
fscanf(fp1,"%s %lf %lf %lf",token,&p_atr->ang_orientacao.x,
  &p_atr->ang_orientacao.y,
  &p_atr->ang_orientacao.z);
/* $VEL_ANG_SRO  <x> <y> <z>  */
fscanf(fp1,"%s %lf %lf %lf",token,&p_atr->vel_angular.x,
  &p_atr->vel_angular.y,
  &p_atr->vel_angular.z);
/* $NRO_TORQUE_EXT  <nro. torques externos> */
fscanf(fp1,"%s %d",token,&p_atr->nro_torques_ext);
/* Aloca areas p/ pontos do objeto corrente (SRO) no qual sao fornecidas
   as forcas externas */
p_atr->p_radius = Aloca_Vetor_XYZ(p_atr->nro_torques_ext);
/* Aloca areas p/ pontos de forcas externas (SRO) aplicadas fora do centro
   de massa do objeto corrente(SRO) */
p_atr->p_forca_out_cm = Aloca_Vetor_XYZ(p_atr->nro_torques_ext);
for (j=0;j<p_atr->nro_torques_ext;j++)
    {
    /* $PONTO_OBJ_SRO  <x> <y> <z>  */
    fscanf(fp1,"%s %lf %lf %lf",token,&p_atr->p_radius[j].x,
       &p_atr->p_radius[j].y,
                &p_atr->p_radius[j].z);
   /* $PONTO_FORCA_SRO  <x> <y> <z>  */
   fscanf(fp1,"%s %lf %lf %lf",token,&p_atr->p_forca_out_cm[j].x,
     &p_atr->p_forca_out_cm[j].y,
     &p_atr->p_forca_out_cm[j].z);
    }
/* Aloca areas de Trabalho p/ utilizacao do CliC */
Aloca_Areas_Trabalho();
/* Inicializa funcoes p/ utilizacao do CliC */
Inicializa_Modelo(p_atr);
/* $ENDOBJ  */
fscanf(fp1,"%s",token);
Free_Vetor_XYZ(p_atr->p_forca_ext);
Free_Vetor_XYZ(p_atr->p_radius);
```

```
      Free_Vetor_XYZ(p_atr->p_forca_out_cm);
      Free_Atributos(p_atr);
     }
 /* $ENDSIMU  */
 fscanf(fp1,"%s",token);
 fclose(fp1);
 Calcula_V_Normais_Faces();
 Calcula_Centroides_F_4V();
}

/*-------------------------------------------------------------------+
|    Atualiza_Funcoes                                                |
|    Funcao: Atualizar funcoes de translacao e orientacao utilizadas na |
|            rotina de sincronizacao                                 |
|    Parametros: double *f_tr_x (Area p/ translacao em x)            |
|                double *f_tr_y (Area p/ translacao em y)            |
|                double *f_tr_z (Area p/ translacao em z)            |
|                double *f_an_or (Area p/ angulos de orientacao)     |
|                int step (Passo de tempo atual)                     |
|    Globais:  C_OBJETO *obj_corr  (Pointer p/ objeto corrente)      |
|    Vlr. Retorno: double *f_tr_x (Translacao em x)                  |
|                  double *f_tr_y (Translacao em y)                  |
|                  double *f_tr_z (Translacao em z)                  |
|                  double *f_an_or (Angulos de orientacao)           |
+-------------------------------------------------------------------*/

void Atualiza_Funcoes(f_tr_x,f_tr_y,f_tr_z,f_an_or,step)
double *f_tr_x,
       *f_tr_y,
       *f_tr_z,
       *f_an_or;
int step;
{
 int i;
 for (i=0;i<N_ODES_TRANS;i++)
     {
      f_tr_x[i] = obj_corr->fun_trans_all_x[i][step];
      f_tr_y[i] = obj_corr->fun_trans_all_y[i][step];
      f_tr_z[i] = obj_corr->fun_trans_all_z[i][step];
     }
 for (i=0;i<N_ODES_ORIEN;i++)
     f_an_or[i] = obj_corr->fun_ang_ori_all[i][step];
}

/*-------------------------------------------------------------------+
```

```
|    Sincroniza_Simulacao                                                  |
|    Funcao : Realiza a sicronizacao das operacoes a serem executadas      |
|             durante a simulacao ao longo do tempo                        |
|    Parametros : ---*---                                                  |
|    Globais : C_OBJETO *objetos (Pointer p/ area inicial de objetos)      |
|              C_OBJETO *obj_corr (Pointer p/ objeto corrente)             |
|              int nro_obj_simu (Nro. de objetos p/ simulacao)             |
|    Vlr. Retorno : ---*---                                                |
+-------------------------------------------------------------------------*/

void Sincroniza_Simulacao()
{
 double t_atual = time_start + h,
        *fun_trans_x,
        *fun_trans_y,
        *fun_trans_z,
        *fun_ang_ori,
        h_atual = h;
 int i,
     j,
     step_atual = 1;
 bool Check_Collision,   /* 1 houve colisao 0 nao houve colisao */
      Houve_Colisao = FALSE;
 Aloca_Vector(N_ODES_TRANS,&fun_trans_x);
 Aloca_Vector(N_ODES_TRANS,&fun_trans_y);
 Aloca_Vector(N_ODES_TRANS,&fun_trans_z);
 Aloca_Vector(N_ODES_ORIEN,&fun_ang_ori);
 while (t_atual <= (time_finish - time_start))
       {
       for (i=0;i<nro_obj_simu;i++)
           {
            obj_corr = objetos + i;
            if (! Houve_Colisao)
               Atualiza_Funcoes(fun_trans_x,fun_trans_y,fun_trans_z,
                                fun_ang_ori,step_atual - 1);
           /* (odeint)  */
           /*odeint(fun_trans_x,N_ODES_TRANS,exat,hinic,hminimo,
                    T_TRANS_X,&nok_trans_x,&nbad_trans_x,trans_x_xp,
                    obj_corr->fun_trans_all_x);
           odeint(fun_trans_y,N_ODES_TRANS,exat,hinic,hminimo,
                   T_TRANS_Y,&nok_trans_y,&nbad_trans_y,trans_y_xp,
                   obj_corr->fun_trans_all_y);
           odeint(fun_trans_z,N_ODES_TRANS,exat,hinic,hminimo,
                  T_TRANS_Z,&nok_trans_z,&nbad_trans_z,trans_z_xp,
                  obj_corr->fun_trans_all_z);*/
```

```
   Rk_Drivers(fun_trans_x,N_ODES_TRANS,T_TRANS_X,step_atual,h_atual,
              tempo_sincro[step_atual - 1],obj_corr->fun_trans_all_x);
   Rk_Drivers(fun_trans_y,N_ODES_TRANS,T_TRANS_Y,step_atual,h_atual,
              tempo_sincro[step_atual - 1],obj_corr->fun_trans_all_y);
   Rk_Drivers(fun_trans_z,N_ODES_TRANS,T_TRANS_Z,step_atual,h_atual,
              tempo_sincro[step_atual - 1],obj_corr->fun_trans_all_z);
   /* (odeint)  */
   /*odeint(fun_ang_ori,N_ODES_ORIEN,exat,hinic,hminimo,
            T_ORIENTACAO,&nok_ori,&nbad_ori,ori_xp,
            obj_corr->fun_ang_ori_all);*/
   Rk_Drivers(fun_ang_ori,N_ODES_ORIEN,T_ORIENTACAO,step_atual,h_atual,
              tempo_sincro[step_atual - 1],obj_corr->fun_ang_ori_all);
   }
Check_Collision = Deteccao_Colisao(step_atual);
if (Check_Collision)
   {
    Houve_Colisao = TRUE;
    h_atual = h_atual / 2.0;
    /* CONSIDERAR ATE UMA PRECISAO POSSIVEL */
    t_atual = tempo_sincro[step_atual - 1] + h_atual;
    t_atual = (time_finish - time_start) + 1.0;
    nstep_simu = step_atual + 1;
   }
else
   {
    if (!Houve_Colisao)
       {
        tempo_sincro[step_atual] = tempo_sincro[step_atual - 1] + h;
        t_atual += h;
        step_atual++;
       }
    else
       {
        tempo_sincro[step_atual] = t_atual;
        Houve_Colisao = FALSE;
       /* --- Resposta_Colisao(...); */
        step_atual++;
        t_atual += h;
        h_atual = h;
        /*t_atual = (time_finish - time_start) + 1.0;
        nstep_simu = step_atual; */
       }
    }
if (step_atual >= nstep_simu)
   t_atual = (time_finish - time_start) + 1.0;
```

```
      }
 Free_Vector(fun_trans_x);
 Free_Vector(fun_trans_y);
 Free_Vector(fun_trans_z);
 Free_Vector(fun_ang_ori);
}

/*-------------------------------------------------------------------+
|    Trata_Saidas_Simulacao                                          |
|    Funcao : Realiza a gravacao dos valores das funcoes simuladas ao|
|             longo do tempo nos arquivos de saida :                 |
|             ...trans.out                                           |
|             ...orien.out                                           |
|             ...pos.out                                             |
|             ...ang.out                                             |
|    Parametros : ---*---                                            |
|    Globais :   char nome_modelo (nome do modelo a ser simulado)    |
|                C_OBJETO *objeto (Pointer p/ area inicial de objetos)|
|                C_OBJETO *obj_corr (Pointer p/ objeto corrente)     |
|                int nro_obj_simu  (Nro. de objetos a serem simulados)|
|    Vlr. Retorno : ---*---                                          |
+-------------------------------------------------------------------*/

void Trata_Saidas_Simulacao()
{
 char trans[30],
      orien[30],
      pos[30],
      ang[30];
 FILE *p_trans,
      *p_orien,
      *p_pos,
      *p_ang;
 int i,
     j;
 /* Mensagens de saida p/ Simulacao */
 printf("\n\n");
 printf("Saidas da Simulacao \n\n");
 for (i=0;i<nro_obj_simu;i++)
     {
      obj_corr = objetos + i;
      /* Geracao simplificada de arq. de saida p/ apenas dois objetos */
      if (i==0)
         {
         strcpy(trans,nome_modelo);
```

```
    strcat(trans,"paral1");
    strcat(trans,"trans.out");
    strcpy(orien,nome_modelo);
    strcat(orien,"paral1");
    strcat(orien,"orien.out");
    strcpy(pos,nome_modelo);
    strcat(pos,"paral1");
    strcat(pos,"pos.out");
    strcpy(ang,nome_modelo);
    strcat(ang,"paral1");
    strcat(ang,"ang.out");
   }
else
   {
    strcpy(trans,nome_modelo);
    strcat(trans,"paral2");
    strcat(trans,"trans.out");
    strcpy(orien,nome_modelo);
    strcat(orien,"paral2");
    strcat(orien,"orien.out");
    strcpy(pos,nome_modelo);
    strcat(pos,"paral2");
    strcat(pos,"pos.out");
    strcpy(ang,nome_modelo);
    strcat(ang,"paral2");
    strcat(ang,"ang.out");
   }
if ((p_trans = fopen(trans,"w")) == NULL)
    {
     printf("Nao foi possivel abrir arquivo (trans) \n");
     exit(0);
    }
if ((p_orien = fopen(orien,"w")) == NULL)
    {
     printf("Nao foi possivel abrir arquivo (orien) \n");
     exit(0);
    }
if ((p_pos = fopen(pos,"w")) == NULL)
    {
     printf("Nao foi possivel abrir arquivo (pos) \n");
     exit(0);
    }
if ((p_ang = fopen(ang,"w")) == NULL)
    {
     printf("Nao foi possivel abrir arquivo (ang) \n");
```

```
            exit(0);
           }
        for (j=0; j<nstep_simu; j++)
            {
            /* Arquivo ..._trans.out  */
            fprintf(p_trans,"%f %f %f %f %f %f %f\n",tempo_sincro[j],
                            obj_corr->fun_trans_all_x[0][j],
                            obj_corr->fun_trans_all_y[0][j],
                            obj_corr->fun_trans_all_z[0][j],
                            obj_corr->fun_trans_all_x[1][j],
                            obj_corr->fun_trans_all_y[1][j],
                            obj_corr->fun_trans_all_z[1][j]);
            /* Arquivo ..._orien.out  */
            fprintf(p_orien,"%f %f %f %f %f %f %f\n",tempo_sincro[j],
                            obj_corr->fun_ang_ori_all[0][j],
                            obj_corr->fun_ang_ori_all[1][j],
                            obj_corr->fun_ang_ori_all[2][j],
                            obj_corr->fun_ang_ori_all[3][j],
                            obj_corr->fun_ang_ori_all[4][j],
                            obj_corr->fun_ang_ori_all[5][j]);
           /* Arquivo ..._pos.out  */
           fprintf(p_pos,"%f %f %f\n",obj_corr->fun_trans_all_x[1][j],
                                      obj_corr->fun_trans_all_y[1][j],
                                      obj_corr->fun_trans_all_z[1][j]);
           /* Arquivo ..._ang.out  */
           fprintf(p_ang,"%f %f %f\n",obj_corr->fun_ang_ori_all[3][j],
                                      obj_corr->fun_ang_ori_all[4][j],
                                      obj_corr->fun_ang_ori_all[5][j]);
           }
        printf("%s \n",trans);
        printf("%s \n",orien);
        printf("%s \n",pos);
        printf("%s \n",ang);
        fclose(p_trans);
        fclose(p_orien);
        fclose(p_pos);
        fclose(p_ang);
        }
     Free_Areas_Trabalho();
}

/*-------------------------------------------------------------------------+
|    main                                                                  |
|    Funcao : Gerenciar o prototipo (CLIC)                                 |
|    Parametros : ---*---                                                  |
```

```
|    Globais : ---*---                                                    |
|    Vlr. Retorno : ---*---                                               |
+-------------------------------------------------------------------------*/

void main()
{
 /* Mensagens de entrada p/ Simulacao */
 printf("\n\nMotor Program - CLIC \n");
 printf("Entradas da Simulacao \n\n");
 printf("Entre c/ o nome do arquivo (.mod) modelo fisico \n");
 printf(">");
 scanf("%s",nome_modelo);
 Le_Modelo_Fisico();
 Sincroniza_Simulacao();
 Trata_Saidas_Simulacao();
}
```

BIBLIOGRAFIA

[ARN 89a] ARNALDI, B. et al. Animation control with dynamics. In: COMPUTER ANIMATION'89, 1989, Geneva, Switzerland. **Proceedings...** Tokio: Springer-Verlag, 1989. 226p. p.113-123.

[ARN 89b] ARNALDI, B.; DUMONT, G.; HEGRON, G. Dynamics and unification of animation control. **The Visual Computer**, Montreal, v.5, n.1/2, p.22-31, Mar. 1989.

[ARM 87] ARMSTRONG, W.; GREEN, M. Near real-time of human figure models. **IEEE Computer Graphics and Applications**, Los Alamitos, v.7, n.6, p.52-61, June 1987.

[BAD 87] BADLER, N. I.; MANOUCHEHRI, K. H.; WALTERS G. Articulated figure positioning by multiple constraints. **IEEE Computer Graphics and Applications**, Los Alamitos, v.7, n.6, p.28-51, June 1987.

[BAR 88a] BARZEL, R.; BARR, A. H. A Modeling system based on dynamic constraints. **Computer Graphics**, New York, v.22, n.4, p.179-188, Aug. 1988.

[BAR 88b] BARZEL, R.; BARR, A.H. Controlling rigid bodies with dynamic constraints. In: ANNUAL CONFERENCE ON COMPUTER GRAPHICS AND INTERACTIVE TECHNIQUES, Aug. 1-5, 1988, New York. **Developments in Physically-Based Modeling**. New York: [*s.n*]. Aug., 1988. p.$E1 - E82$. (Course Notes) SIGGRAPH' 88.

[BAR 89] BARAFF, D. Analytical methods for dynamic simulation of non-penetrating rigid bodies. **Computer Graphics**, New York, v.23, n.3, p.223-232, July 1989.

[BAR 90] BARAFF, D. Curved surfaces and coherence for non-penetrating rigid body simulation. **Computer Graphics**, New York, v.24, n.4, p.19-28, Aug. 1990.

[BAR 91a] BARAFF, D. Coping with friction for non-penetrating rigid body simulation. **Computer Graphics**, New York, v.25, n.4, p.31-40, July 1991.

[BAR 91b] BARR, A. H. Teleological modeling. In: BADLER, N. I. et al. (eds.) **Making them move**: mechanics, control, and animation of articulated figures. San Mateo: Morgan Kaufmann, 1991. 348p. p.315-321.

[BOY 79] BOYSE, J. W. Interference detection among solids and surfaces. **Communications of the ACM**, New York, v.22, n.1, p.3-9, Jan. 1979.

[BRO 88] BROTMAN, L. S.; NETRAVALI, A. N. Motion interpolation by optimal control. **Computer Graphics**, New York, v.22, n.4, p.309-315, Aug. 1988.

[BRU 89] BRUDERLIN, A.; CALVERT, T. Goal-directed dynamic animation of human walking. **Computer Graphics**, New York, v.23, n.3, p.233-242, July 1989.

[BRU 90] BRUDERLIN, A. Towards realistic human figure animation. In: ANNUAL CONFERENCE ON COMPUTER GRAPHICS AND INTERACTIVE TECHNIQUES, 17., Aug. 6-10, 1990, Dallas. **Human Figure Animation**: approaches and applications. Dallas: [s.n], 1990. p.11-20. (Course Notes) SIGGRAPH' 90.

[CAL 88] CALVERT, T. W. The Challenge of human figure animation. In: ANNUAL CONFERENCE ON COMPUTER GRAPHICS AND INTERACTIVE TECHNIQUES, 17., Aug. 6-10, 1990, Dallas. **Human Figure Animation**: approaches and applications. Dallas: [s.n], 1990. p.3-10. (Course Notes) SIGGRAPH' 90.

[CHA 89] CHADWICK, J. E.; HAUMANN, D. R.; PARENT R. E. Layered construction for deformable animated characteres. **Computer Graphics**, New York, v.23, n.3, p.243-252, July 1989.

[CHI 73] CHIOCCARELLO, R. **Manual de fórmulas e tabelas matemáticas.** São Paulo: McGraw-Hill, 1973. 270p.

[CON 65] CONTE, S. D. **Elementos de análise numérica.** Porto Alegre: Globo, 1965. 331p.

[FOL 90] FOLEY, J. D. et al. **Computer graphics**: principles and practice. Reading: Addison-Wesley, 1990. 1174p.

[FRE 90] FREITAS, C. M. D. S. **Técnicas de visualização em simulação.** Porto Alegre: CPGCC da UFRGS, 1990. 47p. (Trabalho Individual, 187).

[GIR 87] GIRARD, M. Interactive design of 3D computer-animated legged animal motion. **IEEE Computer Graphics and Applications**, Los Alamitos, v.7, n.6, p.39-51, June 1987.

[GOL 50] GOLDSTEIN, H. **Classical mechanics.** Reading: Addison-Wesley, 1950. 672p.

[GOL 60] GOLDSMITH, W. **Impact**: the theory and physical behaviour of colliding solids. London: Edward Arnold, 1960. 379p.

[GRE 91] GREEN, M. Using dynamics in computer animation: control and solution issues. In: BADLER, N. I. et al. (eds.) **Making them move**: mechanics, control, and animation of articulated figures. San Mateo: Morgan Kaufmann, 1991. 348p. p.281-314.

[HAH 88] HAHN, J. K. Realistic animation of rigid bodies. **Computer Graphics**, New York, v.22, n.4, p.299-308, Aug. 1988.

[HER 90] HERZEN, B. Von et al. Geometric collisions for time-dependent parametric surfaces. **Computer Graphics**, New York, v.24, n.4, p.39-48, Aug. 1990.

[ISA 87] ISAACS, P. M.; COHEN, M. F. Controlling dynamic simulation with kinematic constraints, behaviour functions and inverse dynamics. **Computer Graphics**, New York, v.21, n.4, p.215-224, July 1987.

[ISA 88] ISAACS, P. M.; COHEN, M. F. Mixed methods for complex kinematic constraints in dynamic figure animation. **The Visual Computer**, Massachusetts, v.4, n.6, p.296-305, Dec. 1988.

[LEM 90] LEMOS, R. R. **Um Ambiente de visualização 3D**. Porto Alegre: CPGCC da UFRGS, 1990. 91p. (Trabalho Individual, 165).

[LEM 92a] LEMOS, R. R.; BASTOS, R. M.; FREITAS, C. M. D. S.; WAGNER, F. R. Animação da dinâmica do movimento e colisões de corpos rígidos. In: CLEI: Conferencia Latinoamericana de Informatica, 18, 31 ago. 1992, Las Palmas de Gran Canaria, España. **Anais**... Las Palmas de Gran Canaria: Actas, 1992, 1266p. p.636-634.

[LEM 92b] LEMOS, R. R.; BASTOS, R. M.; FREITAS, C. M. D. S. Animação de corpos rígidos e tratamento de colisões utilizando análise dinâmica In: SIMPÓSIO BRASILEIRO DE COMPUTAÇÃO GRÁFICA E PROCESSAMENTO DE IMAGENS, 5, nov., 1992. **Anais**... Águas de Lindóia: [s.n.], 86p. p.79-82. SIBGRAPI' 92.

[LEM 93] LEMOS, R. R. **Detecção de colisões entre pares de poliedros rígidos**. Porto Alegre: CPGCC da UFRGS, 1993. 58p. (Trabalho Individual, 354).

[LUC 91] LUCIANI, A. *et al.* Computational a modeler - simulator for animated objects. In: EUROPEAN COMPUTER GRAPHICS CONFERENCE AND EXHIBITION, Sept., 1991, Cambridge, **Procee-**

dings... Vienna: North Holland, 1991. 554p. p.425-436. EUROGRAPHICS' 91.

[LIN 92] LIN, M.; CANNY, J. F. Efficient collision detection for animation. In: EUROGRAPHICS WORKSHOP ON ANIMATION AND SIMULATION, 3, Sept., 1992, Cambridge. **Proceedings** ... 1992, EUROGRAPHICS' 92.

[MAC 36] MacMILLAN, W. D. **Dynamics of rigid bodies**. New York: McGraw-Hill, 1936. 478p.

[MAR 70] MARION, J. B. **Classical dynamics of particles and systems.** New York: Academic Press, 1970. 407p.

[McK 90] McKENNA, M.; ZELTZER, D. Dynamic simulation of autonomous legged locomotion. **Computer Graphics**, New York, v.24, n.4, p.29-38, Aug. 1990.

[MOO 88] MOORE, M.; WILHELMS, J. Collision detection an response for computer animation. **Computer Graphics**, New York, v.22, n.4, p.289-298, Aug. 1988.

[NET 91] NETO, M. M. ; BASTOS, R. M. BEZIER4D: ambiente para modelagem, visualização e animação tridimensional. In: SIMPÓSIO BRASILEIRO DE COMPUTAÇÃO GRÁFICA E PROCESSAMENTO DE IMAGENS, 4, jul. 1991. **Anais**... São Paulo: EPUSP, 1991. 266p. SIBGRAPI' 91.

[OLA 87] OLABARRIAGA, S. D. **SPA**: um sistema computacional para produção de animação. Porto Alegre: CPGCC da UFRGS, 1987. 139p. (Dissertação de Mestrado).

[OLA 92] OLABARRIAGA, S. D. *et al.* Integrating Graphic Software - The Making of 'M.A.T.E.' In: CLEI: Conferencia Latinoamericana de Informa-

tica, 18, 31 ago. 1992, Las Palmas de Gran Canaria, España. **Anais** ... Las Palmas de Gran Canaria: Actas, 1992, 1266p. p.353-361.

[OLI 92] OLIVEIRA NETO, M. M. **Um Algoritmo para interpolação de formas entre objetos modelados por superfície Spline**. Porto Alegre: CPGCC da UFRGS, 1992. 136p. (Dissertação de Mestrado).

[PIN 88] PINHO, M. S. **Diretor:** um sistema interativo para animação. Porto Alegre: UFRGS, Instituto de Informática, 1988. 81p. (Trabalho de Diplomação).

[PLA 88a] PLATT, J. Another introduction to numerical analysis. In: ANNUAL CONFERENCE ON COMPUTER GRAPHICS AND INTERACTIVE TECHNIQUES, Aug. 1-5, 1988, New York. **Developments in Physically-Based Modeling**. New York: [*s.n*], Aug., 1988. p.$C1 - C9$. (Course Notes) SIGGRAPH' 88.

[PLA 88b] PLATT, J. C.; BARR, A. H. Constraint methods for flexible models. **Computer Graphics**, New York, v.22, n.4, p.279-288, Aug. 1988.

[PRE 88] PRESS, W. H. et al. **Numerical recipes in C**: the art of scientific computing. Cambridge: Cambridge University Press, 1988. 735p.

[PUE 88] PUEYO, X.; TOST, D. A survey of computer animation. **Computer Graphics Forum**, Amsterdam, v.7, p.281-300, 1988.

[RES 80] RESNICK, R.; HALLIDAY, D. **Física.** Rio de Janeiro: Livros Técnicos e Científicos, 1980. 554p.

[REY 87] REYNOLDS, C. W. Flocks, herds, and schools: a distributed behavioral model. **Computer Graphics**, New York, v.21, n.4, p.25-34, Jul. 1987.

[RIC 83] RICE, J. R. **Numerical methods, software and analysis**. Auckland: McGraw-Hill, 1983. 483p.

[ROG 88] ROGERS, D. F. **Procedural elements for computer graphics**. New York: McGraw-Hill, 1988. 433p.

[SCH 92] SCHMIDT, A. E.; MUSSE, S. R. **PREVIEW:** sistema de animação. Porto Alegre: CPGCC da UFRGS, 1992. 47p. (Relatório de Pesquisa, 184.)

[SEL 89] SELBIE, S. An Introduction to the use of dynamic simulation for the animation of human movement. In: COMPUTER ANIMATION'89, 1989, Geneva, Switzerland. **Proceedings...** Tokio: Springer-Verlag, 1989. 226p. p.33-45.

[SIL 92] SILVA, R. **ANIMAKER:** Sistema de Animação. Porto Alegre: UFRGS, Instituto de Informática, 1992. 96p. (Trabalho de Diplomação).

[SYM 71] SYMON, K. R. **Mechanics**. Reading: Addison-Wesley, 1971. 557p.

[TER 87] TERZOPOULOS, D. et al. Elastically deformable models. **Computer Graphics**, New York, v.21, n.4, p.205-214, Jul. 1987.

[TER 88a] TERZOPOULOS, D.; WITKIN, A. Physically based models with rigid and deformable components. **IEEE Computer Graphics and Applications**, Los Alamitos, v.8, n.6, p.41-51, Nov. 1988.

[TER 88b] TERZOPOULOS, D.; FLEISCHER, K. Deformable models. **The Visual Computer**, Massachusets, v.4, n.6, p.306-331, Dec. 1988.

[TER 88c] TERZOPOULOS, D.; FLEISCHER, K. Modeling inelastic deformation: viscoelasticity, plasticity, fracture. **Computer Graphics**, New York, v.22, n.4, p.269-278, Aug. 1988.

[TER 89] TERZOPOULOS, D. Physically-based modeling: past, present, and future. **Computer Graphics**, New York, v.23, n.5, p. 191-209, Dec. 1989.

[THA 89] THALMANN, D. Motion control: from key frame to task-level animation. In: COMPUTER ANIMATION, 1989, Geneva, Switzerland. **Proceedings...** Tokio: Springer-Verlag, 1989. 226p. p.3-17.

[UCH 83] UCHIKI, T.; OHASHI, T. TOKORO, M. Collision detection in motion simulation. **Computers & Graphics**, Great Britain, v.7, n.3-4, p.285-293, 1983.

[WEL 69] WELLS, D. A. **Lagragian dynamics**. New York: McGraw-Hill, 1969. 353p. (Shaum's outline series).

[WIL 87a] WILHELMS, J. Toward automatic motion control. **IEEE Computer Graphics and Applications**, Los Alamitos , v.7, n.4, p.11-22, Apr. 1987.

[WIL 87b] WILHELMS, J. Using dynamics analysis for animation of articulated bodies. **IEEE Computer Graphics and Applications**, v.7, n.6, p.12-27, Jun. 1987.

[WIL 88a] WILHELMS, J. Dynamics for computer graphics: a tutorial. **Computing Systems**, v.1, n.1, p.63-93, Winter 1988.

[WIL 88b] WILHELMS, J.; MOORE, M; SKINNER, R. Dynamic animation: interaction and control. **The Visual Computer**, Massachusets, v.4, n.6, p.283-295, Dec. 1988.

[WIL 90] WILHELMS, J. Behavioral animation using an interactive network. In: COMPUTER ANIMATION, 1990, Geneva, Switzerland. **Proceedings...** Tokio: Springer-Verlag, 1990. 237p. p.95-106.

[WIL 91] WILHELMS, J. Dynamic experiences. In: BADLER, N. I. et al. (eds.) **Making them move:** mechanics, control, and animation of articuled figures. San Mateo: Morgan Kaufmann, 1991. 348p. p.265-279.

[WIT 77] WITTENBURG, J. **Dynamics of systems of rigid bodies**, B. G. Teubner, Stuttgart, 1977.

[WIT 87] WITKIN, A.; FLEISCHER, K.; BARR, A. Energy constraints on parameterized models. **Computer Graphics**, New York, v.21, n.4, p.225-232, July 1987.

[WIT 88] WITKIN, A.; KASS, M. Spacetime constraints. **Computer Graphics**, New York, v.22, n.4, p.159-168, Aug. 1988.

[WIT 90] WITKIN, A.; WELCH, W. Fast animation and control of nonrigid structures. **Computer Graphics**, New York, v.24, n.4, p.243-252, Aug. 1990.

[ZEL 82] ZELTZER, D. Motor Control technique for figure animation. **IEEE Computer Graphics and Applications**, Los Alamitos, v.2, n.9, p.53-59, Nov. 1982.

[ZEL 91] ZELTZER, D. Task-level graphical simulation: abstraction, representation, and control. In: BADLER, N. I. et al. (eds.) **Making them move:** mechanics, control, and animation of articuled figures. San Mateo: Morgan Kaufmann, 1991. 348p. p.3-33.

Informática
UFRGS

"Animação e tratamento de colisões de corpos rígidos utilizando análise dinâmica".

Dissertação apresentada aos Srs.:

Prof. Anatólio Laschuk

Profa. Carla Maria Dal Sasso Freitas

Prof. Dr. Dalcídio Moraes Claudio

Prof. Dr. Léo Pini Magalhães (UNICAMP)

Vista e permitida a impressão.
Porto Alegre, 27/01/93.

Prof. Dr. Flávio Rech Wagner,
Orientador.

Prof. Dr. Ricardo A. da L. Reis,
Coordenador do Curso de Pós-Graduação
em Ciência da Computação.