UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO
DEPARTAMENTO DE ENGENHARIA DE COMPUTAÇÃO E AUTOMAÇÃO INDUSTRIAL

UM SISTEMA INTERATIVO DE ANIMAÇÃO NO CONTEXTO ProSIm

Dissertação de mestrado apresentada à Faculdade de Engenharia Elétrica e de Computação da Universidade Estadual de Campinas como parte dos requisitos para a obtenção do título de Mestre em Engenharia Elétrica.

Autor: ALBERTO BARBOSA RAPOSO
Orientador: PROF. DR. LÉO PINI MAGALHÃES

Junho de 1996

BANCA DE AVALIAÇÃO

Presidente:

Prof. Dr. Léo Pini Magalhães (UNICAMP - FEEC - DCA)

Membros:

Profa. Dra. Clarisse Sieckenius de Souza (PUC-RJ - Depto. de Informática)

Profa. Dra. Wu Shin-Ting (UNICAMP - FEEC - DCA)

Suplente:

Prof. Dr. Roberto de Alencar Lotufo (UNICAMP - FEEC - DCA)

AGRADECIMENTOS

À FAPESP, pelo apoio financeiro através da bolsa de mestrado.

Ao CNPq e CAPES, pelo apoio financeiro através da bolsa de iniciação científica (onde tudo começou...) e em outras ocasiões.

Ao meu orientador, Prof. Dr. Léo Pini Magalhães, não só pela dedicação e ajuda que sempre deu nos momentos problemáticos do trabalho, mas também pelas oportunidades que me tem aberto.

Ao Prof. Dr. José Tarcisio F. de Camargo, um grande companheiro de pesquisas desde os tempos de iniciação científica.

À Prof. Dra. Clarisse Sieckenius de Souza, pela ajuda e sugestões que sempre deu nas reuniões no Rio ou via e-mail.

A todos os professores do DCA, especialmente ao Prof. José Mário De Martino (pelas sugestões - ou "ruídos", segundo sua própria definição - nas apresentações da interface) e à Profa. Dra. Wu Shin-Ting (pela dedicação e paciência na administração da área do ProSIm).

A toda a equipe do ProSIm, em particular, ao incansável Marcelo Malheiros, ao eficiente Bruno e ao paciente Lamanna.

Aos meus pais, Manoel e Dalva, pelo incentivo e apoio que sempre deram à realização do curso de mestrado.

Aos meus irmãos, Sávio, Djane e Susane, também pelo apoio.

À minha querida noiva, Ana Beatriz, por ter suportado um namoro à distância por mais de 6 anos e por apoiar e ajudar, sempre que preciso.

Ao Sr. Orlando e a D. Zeny, pela prestatividade e hospitalidade que sempre tiveram comigo no Rio.

Ao Jô Soares, por me incentivar a ficar trabalhando até tarde.

Em memória de minha avó, Maria Antônia da Silva Raposo (1901-1996).

“O movimento é natural, surge espontaneamente. Por essa razão, a transformação do antigo torna-se fácil. O antigo é descartado e o novo é introduzido. Ambas as medidas se harmonizam com o tempo, não resultando daí, portanto, nenhum dano.”

I Ching chinês, ou “O livro das Mutações”.

RESUMO

A produção de uma animação por computador sem a presença de um especialista em programação requer um sistema que apresente facilidades tanto na criação da mesma quanto na comunicação animador-computador.

O objetivo deste trabalho é a reformulação do TOOKIMA (*TOOl Kit for scripting computer Modeled Animation*), um conjunto de ferramentas cinemáticas para a descrição algorítmica de animações modeladas por computador, de modo a torná-lo um sistema interativo para a criação de animações.

O trabalho pode ser dividido em duas etapas distintas.

Na primeira etapa, é desenvolvida uma nova linguagem para o roteiro de animação. Essa nova linguagem tem como objetivo permitir a criação de roteiros com um nível mais alto de comandos (que se aproximem mais daqueles desenvolvidos pelos animadores profissionais). Isso elimina a necessidade do conhecimento, por parte do animador, de comandos de baixo nível da linguagem do TOOKIMA, que é uma extensão da linguagem C.

A segunda etapa do trabalho é a implementação de uma interface gráfica, que permite a construção interativa de um roteiro de animação, utilizando a linguagem desenvolvida.

Dessa maneira, o sistema pretende atingir três tipos de usuários: o usuário leigo em programação (que utilizará a interface), o usuário experiente (que utilizará a interface, mas terá conhecimento da linguagem de roteiros) e o usuário “especialista” (que conhecerá a linguagem do TOOKIMA e a linguagem C, podendo dispor da maior flexibilidade que uma linguagem de mais baixo nível permite).

ABSTRACT

The production of a computer animation without the presence of a programmer requires a system that facilitates that production and the communication between the animator and the computer.

The objective of this work is to improve the TOOKIMA (TOOl Kit for scripting computer Modeled Animation), a kinematic toolkit for the algorithmic description of computer modeled animations, in order to transform it in an interactive system for the production of animations.

This work can be divided in two phases.

In the first phase, a new scripting language for the animations is developed. This new language allows the creation of scripts with high level commands (closer to that used by professional animators). This avoids the necessity of the animator to know the down level commands of the language of the TOOKIMA, which is an extension of C.

The second phase of the work is the implementation of a graphical interface that allows the interactive construction of the scripts, written in the developed language.

In this way, the system can be used by three kinds of users: the nonprogrammer user (who will use the interface), the experienced user (who will also use the interface, but will know the scripting language) and the expert (who will know the language of the TOOKIMA and C).

ÍNDICE

I - INTRODUÇÃO

"Animar é, literalmente, dar vida" [Foley, 92]. Apesar de muitas vezes ser tratada como sinônimo de "movimento", a animação engloba não somente mudanças de posição, mas também outras mudanças que implicam em efeitos visuais: mudança de forma, cor, transparência, textura, iluminação, posicionamento de câmera, etc.

A animação por computador é um tema estudado na área de computação gráfica, e tem ocupado cada vez mais espaço nas publicações e congressos da área. A indústria de entretenimento (cinema, televisão, videogames, etc) também vem obtendo constantes sucessos com a computação gráfica e a animação por computador.

A computação gráfica, embora seja um campo recente dentro da computação, tem mostrado uma evolução surpreendente, o que pode ser constatado pela qualidade e realismo do que pode ser criado com ela.

I.1 - A HISTÓRIA DA COMPUTAÇÃO GRÁFICA[1]

O primeiro uso prático da computação gráfica data de 1951, quando foi feita a primeira demonstração de um sistema computadorizado de radares, desenvolvido por um grupo de pesquisa do MIT (Massachusetts Institute of Technology). Nesse sistema, os radares eram ligados a um computador digital, que mostrava o mapa da região em seu monitor CRT (tubo de raios catódicos) e desenhava pontos, representando as aeronaves detectadas. Esses sistemas de radares foram usados até a década de 80.

O primeiro sistema de desenho por computador foi o DAC-1 (*Design Augmented by Computers*), criado pela GM e IBM em 1959. Este sistema permitia que o usuário entrasse com a descrição 3D de um automóvel e rotacionasse a imagem, para vê-la de diferentes posições.

O próximo grande avanço da computação gráfica foi a criação do Sketchpad [Sutherland, 63], no MIT. O Sketchpad era um outro programa para desenho em computadores, onde se podia usar uma caneta óptica para desenhar formas geométricas na tela de um computador, salvar estes desenhos, e recarregá-los depois.

Em 1961 foi criado, também no MIT, o primeiro videogame, que se chamava Spacewar. Em 1966 um videogame (mais tarde chamado Odyssey) veio a se tornar o primeiro produto de consumo da computação gráfica.

Na primeira metade da década de 60, vários cientistas já estavam usando a computação gráfica para ilustrar suas pesquisas (por exemplo, para mostrar controles de satélites, fluxo de fluidos viscosos, propagação de ondas de choque, etc). Entretanto, o primeiro centro de pesquisas em computação gráfica só foi criado em 1968, por Dave Evans e Ivan Sutherland (o criador do Sketchpad), na Universidade de Utah (que, no mesmo ano, fundaram sua própria empresa, a Evans & Sutherland, pioneira nos hardwares de aceleração gráfica, isto é, com rotinas de desenho embutidas nos circuitos).

[1] Adaptado de [Morrison, 94].

O primeiro grande avanço do grupo de Utah na geração de imagens 3D foi o desenvolvimento de um algoritmo eficiente para a eliminação de superfícies escondidas [Warnock, 69]. O algoritmo determina as superfícies fora do campo de visualização e as retira do *rendering*. Na verdade, a primeira solução conhecida para o problema das superfícies escondidas foi dada por [Roberts, 63]. Apesar da solução matematicamente "elegante", este algoritmo tinha complexidade proporcional ao quadrado do número de objetos em cena, não despertando, portanto, um grande interesse [Rogers, 85].

Warnock, o criador do algoritmo citado, viria a fundar mais tarde a Adobe Systems, criadora da linguagem PostScript.

À medida que os computadores passavam dos monitores vetoriais para os *rasters*, a preocupação dos pesquisadores se voltou para o desenvolvimento de modelos de iluminação e sombreamento (*shading*) que mostrassem realismo na criação de objetos coloridos. Um dos primeiros métodos de *shading* foi mostrado em [Bouknight, 70]. Hoje, melhorado, este método é conhecido como *flat shading*. Este método colore cada polígono de uma superfície da mesma cor, compromentendo o realismo de superfícies curvas. Um ano depois, surgiu o Gouraud *shading* [Gouraud, 71], que aumentou o realismo das figuras geradas, sem aumentar muito o custo computacional do *rendering*, com relação ao *flat shading*. O método de Gouraud interpola as cores ao longo dos polígonos que compõem as superfícies geradas.

No ano de 1971, graças à tecnologia dos circuitos integrados, toda a eletrônica do processador de um computador pode ser miniaturizada em um chip, o microprocessador.

Em 1973 foi realizado a primeira conferência ACM (*Association of Computing Machinery's*) SIGGRAPH (*Special Interest Group on Computer Graphics*), ainda hoje o maior evento na área. Nesse mesmo ano, a computação gráfica também foi usada pela primeira vez no cinema, no filme "Westworld".

Em 1974, o mapeamento de texturas em superfícies permitiu dar um novo nível de realismo às imagens geradas por computador [Catmull, 74]. No mesmo ano, Phong apresentou seu método de *shading* [Phong, 74], que superou algumas falhas que existiam no método de Gouraud. O método de Phong interpola os vetores normais aos polígonos da superfície, ao invés das cores, e aplica o modelo de iluminação a cada ponto. O problema é o alto custo computacional do algoritmo [Newman, 79]. Por esta razão, ainda hoje a maioria dos *renderers* permite a escolha do método de *shading* entre Gouraud e Phong (às vezes também *flat shading*).

No ano seguinte, foi dado outro grande passo em direção a um maior realismo das imagens geradas, através da publicação dos primeiros estudos do matemático francês Benoit Mandelbrot sobre a teoria dos conjuntos fractais. As teorias de mais de 20 anos de pesquisa resultaram no famoso livro "The fractal geometry of nature", [Mandelbrot, 82]. Nesse livro, Mandelbrot mostra como os seus princípios poderiam ser aplicados à computação de imagens, para criar simulações realísticas da natureza (montanhas, madeiras, etc).

Também no ano de 1975, Steve Wozniak, ajudado pelo amigo Steve Jobs, montou o primeiro Apple e Bill Gates fundou a Microsoft, junto com Paul Allen.

Nessa mesma época, a Universidade de Utah, até então o principal centro de pesquisas em computação gráfica, perdeu o posto para o Instituto de Tecnologia de Nova Iorque (NYIT), que contratou os principais pesquisadores de Utah. O motivo da "queda" de Utah foi a falta de recursos.

Em 1976, foi desenvolvida uma técnica chamada *bump mapping*, que viria a ser publicada no SIGGRAPH dois anos depois [Blinn, 78]. É uma técnica similar ao mapeamento de texturas em superfícies, mas permite a simulação de superfícies não lisas, isto é, com buracos, "dentes", etc.

Em dezembro de 1977, saiu o primeiro exemplar da revista Computer Graphics World, que hoje é importante veículo de divulgação das novidades relacionadas à computação gráfica.

No final da década de 70, o cineasta George Lucas decidiu criar uma divisão de computação gráfica para a sua empresa de efeitos especiais para cinema, a Lucasfilm. Em 1979, esta divisão foi criada, contratando os maiores nomes da computação gráfica nos EUA (a maioria do NYIT). Além da Lucasfilm, a criação de um centro de pesquisas em computação gráfica na Universidade de Cornell contribuiu para o "esvaziamento" do NYIT.

Em 1979 também foram feitos os primeiros trabalhos sobre um novo método de *rendering*, no qual o computador simula o trajeto de todos os raios luminosos, a partir do observador, até os objetos da cena, levando em consideração a reflexão e difração dos mesmos nas superfícies [Whitted, 79] e [Kay, 79]. Era o início da técnica de *ray-tracing* (na verdade, como citado em [Foley, 92], desde a década de 60 já havia trabalhos que simulavam o trajeto de raios, para a determinação de sombras por exemplo; entretanto, foram esses dois trabalhos de 1979 que estenderam o *ray-tracing*, que passou então a lidar com reflexões especulares e refração).

Entre 1974 e 1979, várias iniciativas foram tomadas na Europa para formar um grupo de especialistas em computação gráfica. As primeiras conferências do grupo foram realizadas em Bolonha, na Itália. Essas iniciativas se concretizaram em 1980, quando a Eurographics Association (*European Association for Computer Graphics*) foi instituída. Essa associação realiza anualmente a mais importante conferência européia sobre computação gráfica.

Em 1981 foi desenvolvido o primeiro *renderer* da Lucasfilm, o REYES (*Renders Everything You Ever Saw*) [Cook, 87], que mais tarde se tornaria o Renderman [Upstill, 90].

Nesse mesmo ano, a IBM lançou o primeiro computador pessoal, o IBM PC. No ano seguinte, Jim Clark, um dos pioneiros em Utah na década de 60, fundou a Silicon Graphics Inc., que passou a construir os melhores computadores gráficos existentes. Nesse mesmo ano, foi fundada a Autodesk Inc., cujo primeiro produto foi o AutoCAD 1.0, o mais conhecido programa de CAD (*Computer Aided Design*) atualmente.

Em 1982 também foi apresentado um novo modelo de iluminação, que permitia a simulação de objetos polidos, como o metal [Cook, 82]. Este modelo leva em consideração a energia das fontes de luz, ao invés da intensidade ou brilho da luz.

No SIGGRAPH do mesmo ano, foi mostrada uma sequência de vídeo na qual uma mulher se transformava em um lince. Era o surgimento das técnicas de *morphing*.

Ainda em 1982, foi lançado o filme Tron, com mais de 30 minutos de computação gráfica, e que envolveu vários estúdios no seu desenvolvimento. Apesar da qualidade do trabalho realizado em computação gráfica, o filme foi um fracasso, e isso teve uma influência negativa no cinema, com relação à computação gráfica. Mas o departamento de computação gráfica da Lucasfilm "reagiu" e, no ano seguinte, foi responsável por duas sequências de sucesso no cinema: a demonstração do Gênesis em "Jornada nas Estrelas II" e a explosão causada pela destruição do gerador da

Estrela da Morte em "O Retorno de Jedi". Ambas as sequências usaram a técnica de sistemas de partículas, apresentada em [Reeves, 83]. Esta técnica permite modelar objetos sem uma superfície bem definida (chamados objetos *fuzzy*), tais como fogo, fumaça, água e nuvens.

Até então, todos os métodos de síntese de imagens se baseavam na luz que incide diretamente sobre uma superfície. Entretanto, a maior parte da luz que vemos no mundo real é difusa, ou refletida de outras superfícies. Em 1984 foi publicado um artigo que descrevia o método da radiosidade, para o *rendering* de imagens [Goral, 84]. Este método usa fórmulas de transferência de calor, para simular a troca de energia luminosa entre os objetos de uma cena, obtendo resultados bastante realistas.

Em 1984, uma empresa chamada Wavefront lançou o primeiro produto comercial para animação 3D. Até então, cada estúdio de computação gráfica tinha que escrever seus próprios programas para a geração de animações.

Nesse mesmo ano, a Apple lançou o Macintosh, o primeiro computador pessoal a usar uma interface gráfica. No ano seguinte, o EPIC (*Electronic Photography and Imaging Center*), da AT&T, lançou o TARGA, um adaptador de vídeo para PCs. Pela primeira vez os usuários de PCs puderam trabalhar com imagens coloridas de 32 bits (armazenadas no formato TGA Targa). Foi lançado, também em 1985, o Amiga, um computador pessoal com recursos de hardware para a criação de vídeos. Este também foi o ano do surgimento da multimídia, quando a ISO (*International Standards Organization*) criou o primeiro padrão para CD-ROMs.

Em 1986, a Crystal Graphics lançou o TOPAS, um dos primeiros programas de animação de alta qualidade para PCs. Nesse mesmo ano, a computação gráfica passou a ser usada em tribunais dos EUA, para ajudar o juri a visualizar os casos. Esse tipo de computação gráfica, conhecido como *Forensic Animation*, é mais direcionado à exatidão técnica do que à estética visual.

Nesse mesmo ano, a divisão de computação gráfica da Lucasfilm se separou da mesma, se tornando a Pixar. A Pixar continuou a desenvolver seu *renderer*, criando, em 1988, o Renderman, um padrão para a descrição de cenas 3D. O padrão Renderman [Upstill, 90] descreve tudo que o computador precisa saber antes de fazer o *rendering* da cena (objetos, fontes de luz, câmera, etc). Uma vez que a cena está convertida para um arquivo Renderman, ela pode passar pelo processo de *rendering* em vários sistemas, desde PCs até ambientes UNIX.

Também em 1988, a Softimage lançou o software que se tornou o padrão para animação na Europa. No ano seguinte, foi lançado o Autodesk Animator [Autodesk, 89], [Freiwald, 92], que se tornou o padrão para animações em PCs.

Em 1989 a computação gráfica teve outro grande marco no cinema. Em "O Segredo do Abismo", uma criatura feita de água interagia com personagens do filme, um efeito que não poderia ser obtido com as técnicas tradicionais de efeitos especiais do cinema.

Em 1990, dois softwares importantes foram lançados: o Windows 3.1, pela Microsoft (seguindo uma interface gráfica similar à do Macintosh) e o 3D Studio [Autodesk, 90], [Malheiros, 95], pela Autodesk, que se tornou o líder no mercado de softwares de animação 3D para PCs.

O ano de 1991 marcou a definitiva "rendição" do cinema à computação gráfica. Nesse ano, os dois maiores sucessos de bilheteria foram "O Exterminador do Futuro 2" (onde, pela primeira vez a computação gráfica criou um personagem principal) e "A Bela e a Fera" (onde os estúdios Disney usaram o computador para

criar parte dos belos cenários). Desde então a computação gráfica passou a ser parte integrante do cinema e a realizar feitos extraordinários como a criação dos dinossauros em “Parque dos Dinossauros”, em 1993, e, em 1995, o filme “Toy Story”, o primeiro longa metragem inteiramente modelado por computador.

No tocante ao Brasil, o principal evento da comunidade de cientistas, profissionais e estudantes em computação gráfica e processamento de imagens é o SIBGRAPI (Simpósio Brasileiro de Computação Gráfica e Processamento de Imagens), uma realização anual da Sociedade Brasileira de Computação, iniciado em 1988.

A computação gráfica e, particularmente, a animação por computador, são áreas relativamente recentes de estudo, que obtiveram grandes avanços nos últimos anos. Mas as novidades continuam surgindo a cada ano e estas áreas ainda representam um vasto campo para pesquisas.

A Figura I.1 mostra um resumo com os principais fatos descritos.

I.2 - ProSIm

I.2.1 - Histórico

O ProSIm (Prototipação e Síntese de Imagens foto-realistas e animação) constitui um protótipo de ambiente gráfico interativo para a síntese de imagens foto-realistas e animação por computador. Este projeto foi criado há cerca de nove anos pelos pesquisadores atuantes em computação gráfica no Grupo de Computação de Imagens do DCA (Departamento de Engenharia de Computação e Automação Industrial), da FEEC (Faculdade de Engenharia Elétrica e de Computação), UNICAMP (Universidade Estadual de Campinas). O objetivo era aglutinar em um ambiente gráfico comum as atividades relacionadas à síntese de imagens e animação por computador.

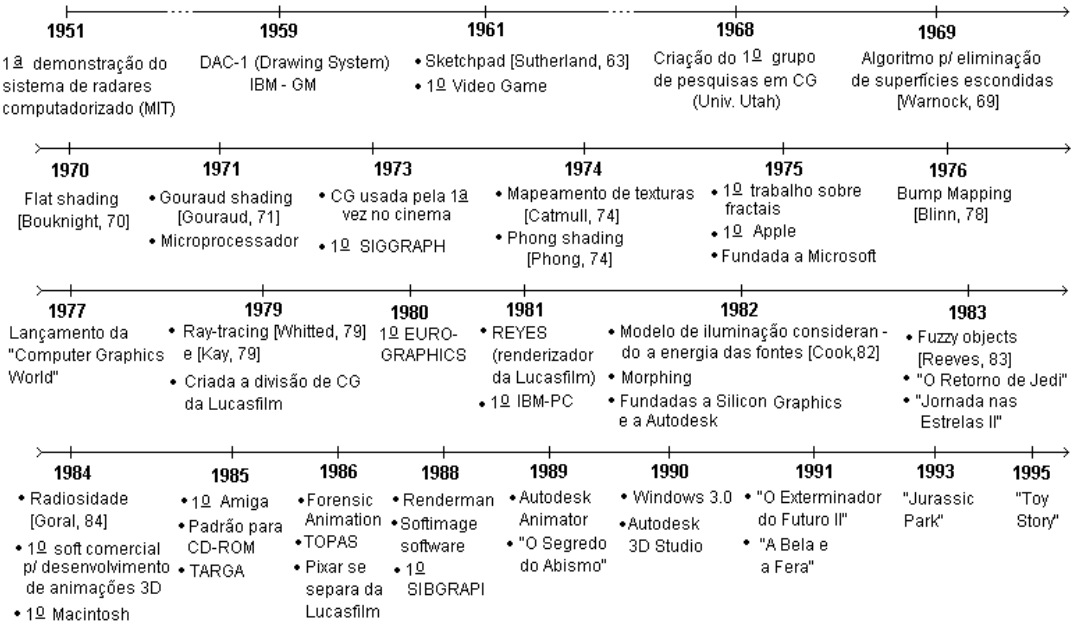

Figura I.1: Principais eventos que marcaram a história da computação gráfica.

Em seu estágio inicial, o ProSIm era um modelador de cenas baseado no paradigma CSG (*Construtive Solid Geometry*) para a interface com os usuários e no modelo B-Rep (*Boundary representation*) para a representação interna dos objetos. Além disso, permitia a visualização final da imagem através da utilização de um dos algoritmos de *rendering* existentes (um *scanline* e um *ray-tracing*), a partir da definição de parâmetros tais como fontes de luz, características das superfícies, etc. Nesse estágio, o ProSIm ainda não provia recursos para a animação de objetos em uma cena.

Num segundo estágio, o projeto migrou do ambiente PC-DOS para Sun-UNIX. Nesse estágio, começaram as atividades em animação, com o desenvolvimento do TOOKIMA (*TOOl Kit for scripting computer Modeled Animation*) [Hounsell, 92a], [Hounsell, 92b]. O TOOKIMA define um conjunto de ferramentas para a descrição algorítmica de animações de objetos modelados por computador, utilizando a filosofia procedimental. Ele provê uma linguagem para a sincronização temporal e descrição das cenas que comporão uma animação através de roteiros (*scripts*). É uma ferramenta puramente cinemática.

Para dar suporte a simulações dinâmicas, foi implementado o $ANIMA_DO$ (Animação por Dinâmica de Objetos) [Rodrigues, 93a], [Rodrigues, 93b]. O $ANIMA_DO$ fundamenta-se na utilização das leis físicas (dinâmica) para a descrição de movimentos. Com ele, o animador pode produzir movimentos bastante realistas, selecionando forças e torques de controle, diretamente relacionados às leis mecânicas que regem o movimento.

O TOOKIMA e $ANIMA_DO$ ainda eram incapazes de tratar certos tipos de modelos, como o de estruturas articuladas e objetos deformáveis.

O conjunto de técnicas apresentado em [Camargo, 93] permitiu o desenvolvimento de modelos cinemáticos e dinâmicos para estruturas articuladas, além da possibilidade de planejamento de trilhas e prevenção de colisões. Com

relação aos objetos deformáveis, resultados recentes em modelagem geométrica estão proporcionando ao ProSIm facilidades nesse sentido [Bernardes, 95], [Horta, 95].

Os objetos usados em uma cena, tanto no TOOKIMA quanto no ANIMA$_D$O, tinham suas formas definidas por funções das bibliotecas SolGen e CSG, modeladores geométricos desevolvidos para o ProSIm. O sistema de *rendering* utilizado se baseava em um algoritmo *scanline Z-buffer* [Preto, 92], sendo mais tarde substituído pelo SIPP (*Simple Polygon Processor*) [Yngvesson, 94], um *renderer* de domínio público com mais recursos.

Outros trabalhos em *rendering* também foram realizados na segunda fase do ProSIm, por exemplo: [Besuievsky, 93] (*ray-tracing*), [Queiroz, 91] (radiosidade) e [Diz, 95] (pós-tratamento da imagem sintetizada). Ainda nesta etapa do projeto, foram iniciadas atividades em texturização de imagens: [Stahlke, 93], [Alegre, 94] e [Silveira, 94].

I.2.2 - Estágio Atual

Tradicionalmente, o problema do controle dos movimentos de objetos numa animação é abordado por meio de equações diferenciais e das leis físicas que regem tais movimentos. Entretanto, a complexa natureza de vários sistemas físicos leva a sistemas de equações diferenciais cada vez mais complexos. Além disso, não existe uma solução global que possa ser aplicada à animação de qualquer objeto, isto é, uma solução conveniente para determinado sistema pode não trazer bons resultados, se aplicada a outro sistema. Em [Camargo, 94] foi apresentado um modelo para a divisão do controle dos movimentos numa animação em dois níveis: controle local e controle global. Essa abordagem simplifica a modelagem de sistemas ou objetos relativamente complexos. Pode-se atribuir a cada ator um bloco de controle local, que conterá suas “regras de comportamento” (ou seja, as leis físicas que regem seus movimentos). Para o controle da interação entre os diversos atores da animação e o animador, é sugerido um bloco de controle global, que determina, por exemplo, quais atores são afetados pela ocorrência de determinado evento e como eles reagirão.

Este paradigma é orientado a eventos, sendo o tempo não mais o único fator determinante para o desencadeamento de reações dos atores de uma animação/simulação. Os sistemas a serem simulados se enquadram nas classes dos DEDS (*Discrete Event Dynamic Systems*), que podem ser modelados por ESMs (*Extended State Machines*). Mais detalhes sobre esta proposta podem ser encontrados em [Camargo, 95a], [Camargo, 95b] e [Camargo, 95c].

A formalização da estratégia de divisão do controle de uma animação em blocos de controle local e um bloco de controle global tem sido um dos objetos de pesquisa do grupo ProSIm.

Paralelamente às atividades desenvolvidas em animação, modelagem geométrica e *rendering*, também têm sido desenvolvidos trabalhos para a integração dos programas anteriormente desenvolvidos e criação de uma interface gráfica interativa para o sistema ProSIm.

Para a implementação da interface, optou-se por uma abordagem *top-down*, em que, num primeiro momento, foram definidos seu *layout* (aparência visual) e as funcionalidades desejadas [Malheiros, 94]. A implementação deste *layout* foi feita com auxílio do XView [Heller, 90], um *toolkit* para o desenvolvimento de interfaces, seguindo o padrão OpenLook [Sun, 89].

Para complementar as funcionalidades das bibliotecas de ferramentas gráficas bidimensionais de alto nível, tais como o XView e o XMotif [Open, 89], está sendo implementada uma biblioteca chamada IQLT [Alegre, 95], com funções que permitem a manipulação e visualização de objetos tridimensionais com recursos bidimensionais.

No estágio atual de desenvolvimento, é possível acessar as seguintes funcionalidades através da interface ProSIm:

- Modelagem Geométrica: definição de objetos por instanciação ou importação de arquivos, remoção de objetos ou parte deles e exportação de objetos em arquivos.
- Modelagem de Cenas: definição e posicionamento de câmeras e posicionamento de objetos, através da entrada de valores.
- Modelagem de Propriedades Físicas: definição de cores sólidas para os objetos de cena.
- *Rendering*: SIPP, que é uma bilbioteca para o *rendering* de cenas tridimensionais, usando um algoritmo de *scanline Z-buffer* com diversos recursos, tais como, *wireframe*, mapeamento de texturas, *anti-aliasing* e sombras [Yngvesson, 94].
- Animação: desenvolvimento de um script de animação interativamente; pré-visualização em *wireframe* e MPEG; geração dos quadros em formato PPM ou TGA, para posterior gravação em vídeo.

A implementação desta última funcionalidade (Animação) foi resultado direto deste trabalho.

I.3 - MOTIVAÇÃO

A necessidade de integrar as ferramentas de animação existentes no ProSIm e, ao mesmo tempo, torná-las mais fáceis de usar se evidenciou por ocasião dos primeiros trabalhos práticos desenvolvidos com elas. As primeiras animações realizadas pela equipe de “animadores” do LCA (Laboratório do Departamento de Engenharia de Computação e Automação Industrial), FEEC, UNICAMP exigiam vários meses de trabalho e os resultados, embora razoáveis, ficavam aquém daquilo que o potencial do sistema permitia. Isso se devia principalmente à dificuldade da linguagem de composição dos roteiros de animação, que era uma extensão da linguagem C.

Quando o sistema passou a ser usado por artistas (alunos do Instituto de Artes da UNICAMP), essa necessidade tornou-se ainda mais urgente, pois eles não conheciam linguagens de programação e, por isso, ficavam bastante limitados no uso do sistema.

A produção de uma animação sem a presença de um especialista em programação requer um sistema por computador que apresente facilidades tanto na criação da mesma quanto na comunicação animador-computador.

Assim, surgiu a proposta deste trabalho, visando a integração de ferramentas de animação do ProSIm e a implementação de uma interface gráfica. A interface tem como objetivo simplificar a tarefa de criação de animações e se integrar a uma interface mais completa, que englobará todo o sistema ProSIm.

I.4 - O SISTEMA INTERATIVO DE ANIMAÇÃO

É dentro do contexto de integração de ferramentas existentes no ProSIm e desenvolvimento de interface que este trabalho se insere. O objetivo foi a reformulação do TOOKIMA (animação cinemática), de modo a torná-lo um sistema interativo para a criação de animações, inclusive com a implementação de uma interface gráfica. O intuito é tornar o processo de desenvolvimento de uma animação com o ProSIm acessível a usuários não especialistas em animação (artistas, principalmente).

A interface desenvolvida é, na verdade, um módulo da interface ProSIm, descrita anteriormente. A primeira etapa do desenvolvimento de uma animação é a modelagem dos objetos que comporão as cenas e, para isso, o animador deve usar o módulo de modelagem geométrica da interface. Por essa razão, a consistência entre os módulos teve de ser mantida e, para isso, foi seguido o *layout* definido e foram utilizadas as mesmas ferramentas de desenvolvimento (Xview, padrão OpenLook) para a implementação do módulo de animação.

O presente trabalho pode ser dividido em duas etapas distintas.

Num primeiro momento foi desenvolvida uma nova linguagem para o roteiro de animação. Essa nova linguagem teve como objetivo permitir a criação de roteiros com um nível mais alto de comandos (que se aproximem mais daqueles desenvolvidos pelos animadores profissionais). Isso eliminou a necessidade do conhecimento, por parte do animador, de comandos de baixo nível, oriundos da linguagem C, que existem na linguagem do TOOKIMA (que, na verdade, é uma extensão da linguagem C) [Raposo, 95a]. Essa linguagem foi construída “sobre” a linguagem do TOOKIMA (isto é, sendo traduzida para ela depois).

A segunda etapa do trabalho foi o projeto e implementação de uma interface gráfica, que permitisse a construção interativa de um roteiro de animação, utilizando a linguagem desenvolvida.

Dessa maneira, poderá haver três tipos de usuários para o sistema de animação do ProSIm: o usuário leigo em programação (que utilizará a interface gráfica), o usuário experiente (que utilizará a interface gráfica, mas terá conhecimento da linguagem de roteiros) e o usuário “especialista” (que conhecerá a linguagem do TOOKIMA e a linguagem C, podendo dispor da maior flexibilidade que uma linguagem de mais baixo nível permite). Espera-se que, com a experiência na utilização da interface, o usuário leigo se transforme em usuário experiente. Em outras palavras, espera-se que a interface seja uma maneira de “ensinar” a utilização da linguagem de roteiros (isso dá à interface a desejada característica de direcionar o usuário à programação, defendida por [Eisenberg, 95] e discutida no próximo capítulo).

I.5 - ORGANIZAÇÃO DA DISSERTAÇÃO

Nos dois próximos capítulos serão mostrados painéis gerais sobre interfaces e animação por computador, respectivamente (visão global do assunto, trabalhos recentes na área e perspectivas futuras). Esses capítulos têm como objetivo localizar o trabalho, num contexto mais amplo, já que este é um trabalho que envolve o desenvolvimento de uma interface, na área específica de animação por computador.

No capítulo IV, retorna-se ao contexto ProSIm, onde será visto em mais detalhes o sistema de animação do ProSIm, focalizando principalmente o que foi desenvolvido neste trabalho.

No capítulo V, a linguagem de roteiros desenvolvida na primeira etapa deste trabalho será descrita detalhadamente. O capítulo VI descreve a interface gráfica, resultado da segunda etapa do trabalho.

Finalmente, no capítulo VII, a dissertação será concluída e propostas de trabalhos futuros, dando continuação a este, serão apresentadas.

A dissertação termina com três apêndices. O primeiro formaliza a linguagem de roteiros através de uma BNF. O segundo mostra o editor de texturas, um módulo à parte da interface do TOOKIMA, mas que foi desenvolvido como parte deste trabalho. O último apêndice mostra a organização do “pacote” TOOKIMA, em termos de diretórios e arquivos.

II - INTERFACES

É a interface de um programa que estabelece o diálogo entre o usuário e o programa. Hoje em dia, para os usuários de um sistema computacional, a comunicação com o mesmo é tão importante quanto a computação por ele realizada. Como citado em [Hix, 93]: "para os usuários, a interface é o sistema".

A interface com o usuário (UI - *User Interface*) é, portanto, um módulo crítico do sistema e seu desenvolvimento é parte de todo o processo de engenharia de software. O projeto de interfaces requer o conhecimento não só da tecnologia apropriada, mas também de fatores humanos[2], para garantir a eficiência da comunicação homem-máquina.

Recentemente, pesquisadores estenderam o conceito de UI, defendendo a idéia de que a linguagem da interface também seja uma linguagem de programação (EUPL - *End User Programming Language*) [Dertouzos, 92], [Eisenberg, 95], [Souza, 96]. Neste novo cenário, as interfaces devem permitir que o usuário não apenas utilize as aplicações da melhor forma, mas também customize, estenda e/ou as integre em novos ambientes de trabalho.

Seguindo este ponto de vista, [Eisenberg, 95] afirma que uma aplicação bem projetada deve ter uma interface gráfica, acessível aos usuários inexperientes, e deve ser projetada visando levar "gentilmente" o usuário à programação. Em outras palavras, as aplicações devem conter tanto uma interface gráfica acessível e extensível, quanto um interpretador para uma linguagem de programação.

II.1 - USABILIDADE

Em [Hix, 93], uma interface "amigável" é distinguida de uma interface com alta usabilidade (uma interface pode ser muito amigável, mas se não tiver eficiência para garantir produtividade ao usuário, ela não terá usabilidade). Portanto, o conceito de usabilidade, muito frequente em publicações atuais na área, está relacionado com a eficiência da UI, além da reação do usuário à mesma e do quão "natural" ela pareça ao usuário. Segundo [Shneiderman, 92], a usabilidade é uma combinação das seguintes características: facilidade de aprendizado, alta velocidade para a execução de tarefas, baixa taxa de erro no uso, satisfação subjetiva do usuário e retenção da forma de uso com o tempo.

A substituição das interfaces textuais (onde o usuário era obrigado a digitar o comando desejado) pelas interfaces gráficas (GUIs - *Grafical User Interfaces*), com janelas, menus, botões, etc (por essa razão também conhecidas como WIMPs - *Windows, Menus, Icons and Pointers*) não resolveu o problema da usabilidade das UIs. O aparecimento de ferramentas de software de apoio à programação de UIs transformou muitos programadores em "especialistas" em interfaces, sem, no

[2] Segundo [Pressman, 92], "fatores humanos" tem vários significados. Em um nível básico, englobam o conhecimento da percepção visual, da psicologia cognitiva da memória humana e do raciocínio dedutivo e indutivo. Em outro nível, os "fatores humanos" englobam o conhecimento do usuário e do seu comportamento. Finalmente, englobam também o conhecimento das tarefas que o sistema executará, e as que o usuário executará, como parte da interação homem-máquina.

entanto, saberem projetá-las com alta usabilidade. Isso tem levado ao surgimento de muitas interfaces com problemas de usabilidade. Segundo [Nielsen, 90], as modernas técnicas de interação homem-máquina aumentam o grau de liberdade no projeto de UIs, tornando ainda mais essenciais os princípios "tradicionais" de um bom projeto de diálogo (no citado artigo são definidas 9 diretrizes que devem ser levadas em consideração ao se projetar uma interface e também são mostradas as falhas de interfaces que não seguiram tais diretrizes). Existem vários outros trabalhos dedicados ao estudo de problemas em interfaces (um dos mais conhecidos é [Nielsen, 92]).

II.2 - O PROCESSO DE DESENVOLVIMENTO

O projeto de uma UI deve ir além do conceito de "interface amigável". Para um projeto de UI bem sucedido, deve haver o conhecimento completo da diversa comunidade de usuários e das tarefas a serem realizadas [Shneiderman, 92]. A interface tem o compromisso de servir ao usuário e, quando bem projetada, deve "desaparecer", permitindo que o usuário se concentre no seu trabalho.

Segundo [Hix, 93], o processo de desenvolvimento de uma UI consiste de duas partes: o desenvolvimento da componente de interação (como a interface trabalha e qual o seu comportamento em resposta à interação com o usuário) e o desenvolvimento do software de interface (implementação do código do componente de interação).

O desenvolvimento da componente de interação adota a visão do usuário sobre a interface. É desenvolvido o "modelo do usuário", que é o modelo conceitual que o usuário tem a respeito da informação que ele manipula e dos processos que se aplicarão a esta informação [Newman, 79]. São descritas as ações, percepções e tarefas do usuário. Esta etapa envolve fatores humanos, tais como: análise dos tipos de usuários que o sistema terá, suas limitações, etc. Nessa etapa, também são consideradas as diretrizes para o projeto de uma UI de alta usabilidade. Essas diretrizes visam garantir funcionalidade, consistência, simplicidade, cognição, *feedback* apropriado, prevenção de erros e acomodação das diferenças entre os usuários, na interface. Eis algumas das diretrizes citadas em [Hix, 93]:

- Faça o projeto centrado no usuário: o projetista/programador deve procurar fazer o que for melhor para o usuário, mesmo sendo mais difícil para ele (infelizmente, quase tudo o que é mais fácil para o usuário é mais difícil para o projetista) [Norman, 86].
- Conheça as classes de usuários do sistema.
- Otimize as ações do usuário, por exemplo, através de teclas aceleradoras e possibilidade de criação de macros por parte do usuário.
- Leve em consideração as limitações da memória humana: as informações na tela devem ser organizadas de modo que o usuário não precise memorizar a informação de uma tela para outra.
- Previna os erros do usuário: um bom exemplo, é desativar escolhas errôneas nos menus, ou desativar os botões cujos comandos não podem ser acessados em determinado instante.

- Deixe o usuário reconhecer, ao invés de lembrar: é melhor permitir ao usuário fazer uma escolha entre os itens de uma lista (reconhecimento) do que obrigá-lo a escrever um parâmetro ou comando para realizar uma tarefa (lembrança).
- Use a cognição: por exemplo, é mais natural para o usuário usar as teclas aceleradoras "Ctrl-c" para o comando *copy*, do que algo sem sentido, tipo "Esc-F7".
- Use *feedback* informativo, principalmente em tarefas demoradas (através do desenho do ícone em forma relógio, por exemplo), pois em poucos segundos o usuário ficaria na dúvida se o sistema está realizando a tarefa ou ficou bloqueado.
- Use mensagens explicativas e voltadas para o usuário: infelizmente, ainda são comuns mensagens "bizarras", do tipo *"505 hex 0001F9 doublewords of storage were not recovered"*, em simples processadores de textos.
- Use mensagens de erro construtivas e não violentas. As mensagens de erro constituem uma parte de sistema com grande impacto psicológico sobre o usuário. Elas devem dar ao usuário a melhor informação possível sobre o que causou o erro; mensagens do tipo *"syntax error"* ou *"incorrect data"* não ajudam em nada o usuário. Além disso, o sistema não deve "assustar" ou culpar os usuários pelos erros; é melhor dizer, por exemplo, *"unrecognized command"* do que *"illegal command"*, pois no primeiro caso, o computador está admitindo que não é capaz de reconhecer a mensagem, e não que o usuário está errado.
- Tenha sempre como desfazer as ações do usuário (comando *undo*).
- Não antropomorfize, isto é, não dê ao computador características humanas: evite mensagens do tipo "bom-dia!" ou "como está você?". Isso acaba irritando o usuário.
- Mantenha a inércia da tela: uma interface bem projetada muda o mínimo possível, entre uma tela e outra. Objetos estáticos, como botões e ícones, devem aparecer sempre nos mesmos locais, em todas as telas, para manter a consistência.

A engenharia semiótica acrescenta a estas outras diretrizes para o projeto de UIs. Ela estuda os sinais trocados nas telas de E/S da aplicação, através dos quais o usuário cria um modelo conceitual de usabilidade da aplicação (isto é, o significado assumido pelo usuário das mensagens da UI). Dentre as diretrizes da engenharia semiótica, podem ser citadas [Souza, 93b], [Souza, 96]:

- Evite usar sinais "inventados" na interface: os projetistas devem selecionar os sinais dentre os estabelecidos "culturalmente" pelos sistemas de comunicação.
- Use, sempre que possível, sinais para reforçar a "expressão" dos dispositivos da interface do computador (tais como *mouse pointers*, janelas, menus, etc).
- Sempre construa sistemas comunicativos de UI capazes de "expressar-se" de maneira clara.

O desenvolvimento da componente de interação engloba então diversas áreas de estudo: engenharia de software, semiótica, linguística, ciência comportamental, psicologia cognitiva, além da área específica para a qual o programa está sendo desenvolvido (por exemplo, uma interface para um programa médico exigirá a presença de algum médico em seu desenvolvimento). O objetivo final dessa etapa é o desenvolvimento de um protótipo da UI, que será testado e avaliado com os usuários, levando ao reprojeto, se necessário.

O desenvolvimento do software de interface, por sua vez, adota a visão do sistema sobre a interface e envolve algoritmos, bibliotecas de procedimentos, estruturas de dados, etc. São descritas as ações do sistema, em resposta às ações do usuário. Esta etapa engloba as áreas de ciência da computação e engenharia de software. O objetivo é desenvolver o software que implementa o componente de interação.

II.3 - FERRAMENTAS

As primeiras dificuldades encontradas para o desenvolvimento de UIs ocorreram devido à inadequação das linguagens de programação convencionais para esta tarefa. As linguagens convencionais não tinham mecanismos apropriados de E/S, paradigmas de orientação a objetos eficientes e nem facilidades para a programação em tempo-real e em multiprocessadores [Myers, 92a].

Na tentativa de superar estas dificuldades, foram criados métodos para a representação de UIs. Um tipo de representação possível para a interface é através de *event handlers* [Green, 85], através da qual a ação do usuário é vista como um evento pelo sistema, que a envia a uma rotina de software apropriada (o *event handler*) e que, por sua vez, pode gerar uma saída, chamar uma rotina da aplicação, ou mudar o estado do sistema. Outra representação possível para a interface é através de diagrama de estados [Jacob, 86]. Entretanto, essas representações modelam os objetos de interação, mas não modelam o comportamento do usuário. Para uma representação da UI, do ponto de vista do usuário, existem as chamadas linguagens orientadas a tarefas (*task-oriented languages*). Essas linguagens visam a identificação das tarefas a serem realizadas pelo usuário e seu "mapeamento" em especificação de ações, utilizando uma gramática característica. Algumas das mais conhecidas linguagens orientadas a tarefas são: a CLG (*Command Language Grammar*) [Moran, 81], a GOMS (*Goals, Operators, Methods, and Selection*) [Card, 83], a TAG (*Task-Action Grammar*) [Payne, 86] e a UAN (*User Action Notation*) [Hartson, 90], [Hix, 93].

Além das linguagens para a especificação de UIs (componente de interação), existem também sistemas que facilitam sua construção (software de interface). [Linton, 89] dividiu esses sistemas em duas grandes categorias: *toolkits* e UIMSs (*User Interface Management Systems*).

Um *toolkit* para o desenvolvimento de UIs é uma biblioteca de *widgets* que pode ser chamada pelo programa aplicativo. O *widget*, segundo [Myers, 96], é "uma maneira de usar um dispositivo de entrada para a entrada de um certo tipo de valor". Os *widgets* normalmente incluem menus, botões, *sliders*, barras de rolamento, etc. Três exemplos conhecidos de *toolkits* são o X Toolkit [McCormack, 88], o XView

(*X Window-system-based Visual/Integrated Environment for Workstations*) [Heller, 90] e o InterViews [Linton, 89].

O XView e o InterViews são exemplos de *toolkits* orientados a objetos. Os princípios da programação orientada a objetos têm mostrado ser um excelente mecanismo de abstração no desenvolvimento de UIs, pois é natural tratar os elementos de uma UI como objetos. Segundo [Linton, 89], "comparadas com uma implementação procedimental, UIs escritas com uma linguagem orientada a objetos são significativamente mais fáceis de desenvolver e manter".

O uso de *toolkits* dá ao programador um controle extensivo e grande flexibilidade na criação de uma UI. A construção de uma interface utilizando *toolkits*, no entanto, é restrita a programadores experientes, pois este tipo de ferramenta apresenta apenas interface procedimental.

Os UIMSs são geralmente acessíveis a não programadores, pois eles separam completamente o código que implementa a interface do código da aplicação propriamente dita, suportando um nível mais elevado de abstração na construção da UI. Esse tipo de sistema permite que o projetista crie uma UI, completa e funcional, sem ter que programar utilizando uma linguagem de programação tradicional (mas geralmente é preciso utilizar alguma linguagem especial, principalmente quando se pretende acessar banco de dados, transmitir mensagens ou realizar cálculos científicos).

Dentre os diversos tipos de UIMSs (classificados por [Myers, 96]), destacam-se os ambientes de programação visual, que usam o conceito da manipulação direta de objetos visuais (assim chamados porque cada objeto tem uma representação gráfica correspondente). Esse tipo de ambiente permite ao projetista selecionar o objeto desejado (menu, botão, etc) e posicioná-lo com o mouse. Um dos mais populares ambientes de programação visual é o Visual Basic da Microsoft [Microsoft, 92], que permite, além da construção da interface por manipulação direta dos seus componentes, o desenvolvimento de uma aplicação completa, utilizando uma variação da linguagem Basic.

No Brasil, alguns UIMSs têm sido desenvolvidos, destacando-se o EDG [Celes, 95] e o IUP/LED [Levy, 93]. O EDG não é um ambiente de programação visual, mas "permite acesso facilitado tanto a objetos de interface tradicionais quanto a objetos gráficos, possibilitando que programadores ocasionais desenvolvam rapidamente programas com sofisticado grau de interação" [Celes, 95]. Esse sistema utiliza a linguagem de programação Lua [Ierusalimschy, 95], uma linguagem de extensão que combina características tanto das linguagens estruturadas tradicionais quanto da orientação a objetos. O IUP/LED é um sistema portátil de interface com o usuário, composto por um *toolkit* virtual (IUP) e por uma linguagem de especificação de diálogos (LED), isto é, de descrição da composição visual de objetos da interface.

Os UIMSs apresentam várias vantagens. A principal delas é o fato de tornar o processo de desenvolvimento mais simples, tornando-o, portanto, compreensível a uma gama maior de pessoas e facilitando as alterações no projeto. Os UIMSs também facilitam os testes de usabilidade, pois um protótipo pode ficar disponível rapidamente. Além disso, os UIMSs permitem que o projetista dê mais atenção a textos explicativos, ajuda *online* e mensagens de erro, que são objetos importantes de uma boa interface. Entretanto, no estágio atual de desenvolvimento dos UIMSs, eles ainda apresentam algumas desvantagens com relação aos *toolkits*, o que justifica a utilização dos últimos em muitos casos. Entre estas desvantagens, podem ser citadas [Linton, 89]: limitação na criação de interfaces (devido à alta abstração), ineficiência

na criação de UIs que exigem resposta em tempo-real e dependência de linguagens especiais (normalmente desconhecidas para os programadores e sem ferramentas eficientes de *debugging*).

[Myers, 96] cita ainda um outro nível possível para os softwares de interface, que é o nível do sistema de janelas (*windowing system*), um nível mais baixo que o *toolkit*. Apesar de quase todos os sistemas de janelas possuírem *toolkits* construídos sobre eles, o programador pode precisar usar o sistema de janelas diretamente quando quiser desenhar partes específicas da interface. Isso porque os *toolkits*, de uma maneira geral, só permitem o desenho de *widgets* pré-definidos, tais como botões, menus, etc. O mais conhecido sistema de janelas é o X [Scheifler, 86].

A Figura II.1 mostra os três níveis possíveis para a programação de uma UI.

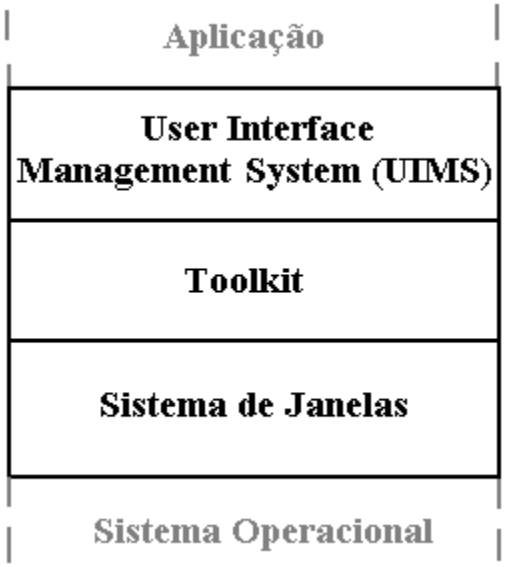

Figura II.1: Os níveis para a programação de UIs [Myers, 96].

II.4 - PERSPECTIVAS

Embora haja um grande número de ferramentas e facilidades disponíveis, a programação de UIs ainda é uma tarefa bastante complexa. Uma pesquisa [Myers, 92b] mostrou que uma média de 45% do tempo total de projeto, 50% do tempo de implementação e 37% do tempo de manutenção é gasto com a interface de um programa. Além disso, uma média de 48% do código de um programa é dedicado à sua interface. Nessa pesquisa também foram citadas as principais dificuldades encontradas para o projeto de interface: obter a informação sobre o que os usuários desejam na interface, acomodar tanto os usuários inexperientes quanto os experientes, escrever o *help* e a documentação e conseguir consistência, principalmente quando há vários desenvolvedores. (Para superar estas dificuldades, muitos trabalhos têm surgido, mostrando novas técnicas e propondo novos paradigmas de UIs.)

Em [Myers, 93], é discutido o uso de heurísticas em UIs. Interfaces que usam heurísticas podem realizar ações diferentes em dados diferentes, com o mesmo comando. Nesse caso, se diz que o sistema "adivinha" o que o usuário deseja que se faça. Um exemplo citado é o Macintosh, que move um arquivo quando ele é arrastado até um outro *folder*. Entretanto, se o novo *folder* se encontra em outro *driver*, o ato de arrastar um arquivo o copia neste novo *folder*. A utilização de heurísticas simplificaria a ação do usuário e tornaria a interface mais fácil de ser usada, já que o sistema realizaria parte do trabalho para o usuário. Entretanto, haveria o risco da ação do usuário ser interpretada incorretamente e levar ao sentimento, por parte do usuário, que ele perdeu o controle sobre o sistema. O grande desafio seria desenvolver algoritmos que interpretassem corretamente o contexto e produzissem o resultado que a maioria dos usuários esperasse. Entretanto, a "aparente facilidade" trazida pelo uso de heurísticas esconde inconsistências. Em [Souza, 93a], o exemplo citado (o Macintosh) é analisado e são mostradas incoerências relativas à transmissão do conceito de *desktop*.

A especificação e a programação por exemplos já são uma realidade, e permitem que o projetista crie um *layout* de UI por manipulação direta de objetos gráficos. As vantagens são grandes pois, além de permitirem a especificação em alto nível para não programadores, elas garantem a consistência da interface (a cada alteração, todo o *layout* é revisado automaticamente e, onde houver ambiguidades, o sistema alerta o usuário, para que ele possa fazer escolhas) [Myers, 88].

O projeto de UIs tridimensionais tem se mostrado cada vez mais importante, uma vez que as técnicas tradicionais de interfaces bidimensionais são inadequadas para as complexas aplicações 3D interativas que têm surgido. O uso de interfaces 3D em aplicações interativas tridimensionais é útil porque reduz a distância cognitiva entre o usuário e a aplicação, eliminando a abstração na representação do espaço 3D em um ambiente 2D. Além disso, as técnicas 3D aumentam a capacidade de informação do *display*, permitindo que uma quantidade maior de informação seja mostrada ao mesmo tempo, de maneira inteligível. Em [Robbins, 95] são mostrados princípios e técnicas para a criação de UIs tridimensionais.

[Robertson, 93] apresenta um sistema experimental (o *Information Visualizer*), usado para desenvolver um novo paradigma de UI para a visualização de informações, baseado nas emergentes tecnologias de 3D e animação interativa.

A realidade virtual é frequentemente citada como integrante da próxima geração de UIs. Em [Deering, 95], é apresentado o HoloSketch, uma ferramenta para a criação e manipulação de objetos tridimensionais, baseada em realidade virtual. É um sistema análogo aos sistemas de desenho 2D, que estende os conceitos para uma interface tridimensional.

M. L. Dertouzos [Dertouzos, 92] diz que é preciso haver uma grande revolução na programação, através da invenção de um novo tipo de linguagem, radicalmente diferente dos existentes. Esse novo tipo de linguagem, que o autor chamou de MVC (*My Virtual Computer*), deve visar as arquiteturas multiprocessadas e em rede, e integrar o processador, as comunicações e os periféricos de E/S. Dessa maneira, os comandos e as representações de dados, presentes na maioria das aplicações, seriam simplificados e incorporados como "interfaces padrões".

Seguindo a mesma linha de raciocínio, [Cordy, 92] e [Eisenberg, 95] concordam que o que falta à interface de manipulação direta, é justamente o que as linguagens de programação provêm: tipos de dados e estruturas, comandos de decisão, repetição e recursão, além de abstrações de procedimentos e parametrizações. Dessa maneira, é interessante unir a facilidade da manipulação direta ao poder das linguagens de programação. Um dos grandes problemas encontrados para atingir este objetivo, segundo [Souza, 96], é a passagem de uma aplicação fechada e meramente customizável para uma aplicação aberta e extensível, onde a programação do usuário final (*End User Programming*) seja possível. Em outras palavras, o problema é encontrar harmonia entre a linguagem de programação do usuário final (EUPL) e a linguagem de interface (UIL - *User Interface Language*). Esta será a tarefa da engenharia semiótica, que deve ser estendida para o projeto integrado de UIL e EUPL.

O projeto cuidadoso de uma EUPL envolve a concepção de um sistema semiótico integrado, que vá de uma maneira contínua (isto é, com o mínimo de "buracos" semânticos) da UIL ao seu ambiente de programação embutido (EUPL). Idealmente, objetos e ações específicos das tarefas devem ser acessados através do mesmo código (palavras e/ou imagens), enquanto elementos específicos da programação (variáveis, *loops*, etc) devem ser codificados em um sistema genérico, pequeno e simples, de itens léxicos e regras sintáticas que permitam ao usuário entender os fundamentos da programação [Souza, 96].

Para a integração entre UIL e EUPL, [Souza, 96] propõe alguns pontos a serem pesquisados pela engenharia semiótica (outros pontos também são propostos em [Eisenberg, 95]):

- Tentar encontrar uma orientação global de projeto que organize tanto a UIL quanto a EUPL em um conjunto comunicativo, com harmonia.
- Selecionar o conjunto mínimo de dados e estruturas de controle para a EUPL.
- Englobar a UIL no ambiente da EUPL, em uma aparente inversão do que normalmente se acredita acontecer (a EUPL englobada na UIL).
- Realizar estudos empíricos para conhecer mais profundamente a natureza semiótica da interação com computadores e as hipóteses emergentes de tal conhecimento.

Alguns autores tentam antecipar como serão as futuras UIs. Por exemplo, [Nielsen, 93] fala sobre a nova geração de UIs, que irá além dos WIMPs. Ela

incorporará realidade virtual, reconhecimento de voz e gestos, animação e multimídia, inteligência artificial, etc. Segundo o autor, a próxima geração de UIs deve conter tantas mudanças, que pode surgir um paradigma baseado em não-comandos para a interação com futuros sistemas, eliminando o diálogo explícito entre o usuário e a máquina, no qual o usuário comanda o computador, para ele fazer algo. Em outras palavras, o foco do usuário passará do controle do computador para o controle do domínio de tarefas. Mas, como o autor mesmo diz, “esse artigo tenta prever o futuro”, e ele poderá ser bem diferente do que se previu...

III - ANIMAÇÃO[3] POR COMPUTADOR

Entre as décadas de 1920 e 1930, Walt Disney popularizou a arte da animação, através da realização de filmes animados, que ainda hoje surpreendem pela qualidade técnica. Durante as décadas seguintes esta arte se desenvolveu, mas sempre envolvendo um número elevado de profissionais, pois o trabalho de desenho, pintura, edição, etc era feito manualmente, quadro a quadro.

Com o advento dos computadores, alguns cientistas começaram a usá-los para esboçar desenhos e até mesmo movimentá-los. Os animadores profissionais perceberam então que ferramentas computacionais poderiam auxiliá-los, automatizando algumas das tediosas e longas tarefas da produção de filmes animados, tais como a criação dos desenhos, dos movimentos, a pintura, a fotogravação, a edição e a sincronização. Quando o computador participa de algumas das etapas da criação de uma animação, se diz que ela é auxiliada por computador.

O crescimento do poder dos computadores permitiu o aumento da participação dos mesmos no processo de criação de uma animação. O computador pode, por exemplo, ser responsável por todo o processo de geração de uma animação, cabendo a ele a modelagem do ambiente tridimensional, dos atores e seus movimentos, e também o controle da animação como um todo. Neste caso, se diz que a animação é modelada por computador.

A grande otimização de tempo, custos, confiabilidade, etc, que a automatização proporcionou ao processo convencional de criação de uma animação, atraiu o interesse de cientistas da computação e animadores.

Hoje em dia, a animação por computador engloba uma grande quantidade de técnicas, que muitas vezes pouco têm em comum. Quase todos os sistemas que lidam com a variação temporal de elementos de imagens geradas por computador são chamados sistemas de animação por computador [Pueyo, 88]. Este capítulo objetiva fazer uma síntese dos aspectos essenciais da animação modelada por computador.

III.1 - COMPONENTES DE UM SISTEMA DE ANIMAÇÃO MODELADA POR COMPUTADOR

Animação, segundo [Thalmann, 85a], é o "processo no qual é gerada dinamicamente uma série de quadros (...), onde cada quadro é uma alteração do quadro anterior". Portanto, como já foi dito no Capítulo I, é errôneo considerar "animação" como sinônimo de "movimento", uma vez que alterações de cor, texturas, iluminação, etc, também caracterizam uma animação.

Durante o processo de criação de uma animação modelada por computador, pelo menos três ferramentas fundamentais são necessárias:

[3] Ao longo desta dissertação, a palavra "animação" será usada em seu sentido restrito, como sinônimo de "movimento".

1. Um modelador geométrico, para a definição e modelagem das características geométricas das entidades que comporão a animação.
2. Um mecanismo de controle da animação, para a definição e execução do controle dos movimentos e das interações entre atores e entre estes e o animador.
3. Um mecanismo de *rendering* e visualização, para a definição dos atributos de cor e textura dos objetos, iluminação, etc, e também para a visualização dos quadros gerados durante o processo de animação.

Algumas técnicas utilizadas em cada uma dessas ferramentas são mostradas na Figura III.1.

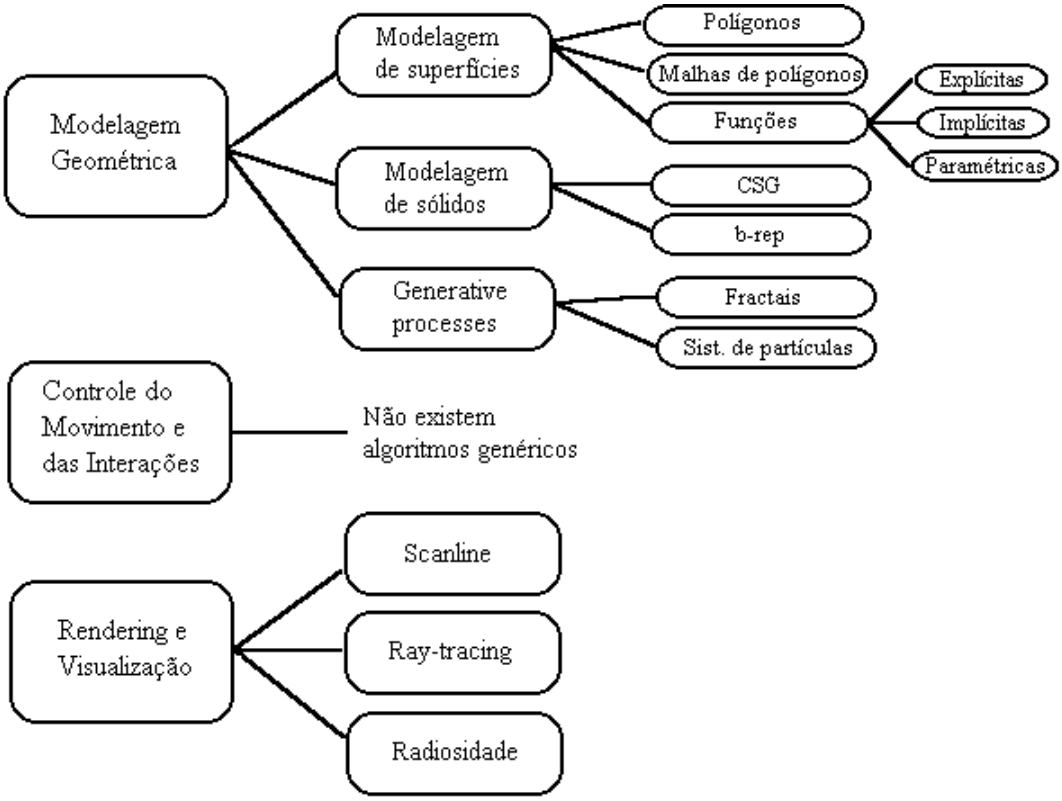

Figura III.1: Ferramentas e técnicas envolvidas na animação por computador.

As técnicas de modelagem geométrica podem ser divididas em três categorias básicas: modelagem de superfícies, modelagem de sólidos e processos geradores (*generative processes*) [Wyvill, 90]. A modelagem de superfícies é feita através de polígonos, malhas de polígonos (*polygon meshes*) ou através de funções, que podem ser explícitas, implícitas ou paramétricas [Foley, 92]. Na modelagem de sólidos, os modelos são descritos pelo volume que eles ocupam no espaço. Pode ser utilizada a representação por fronteiras (*Boundary Representation* ou *B-Rep*), a combinação de formas primitivas por intermédio de operações (*construtive solid geometry* ou CSG), ou, ainda, uma combinação destes. Os modelos também podem ser gerados por processos algorítmicos, tais como os fractais [Mandelbrot, 82] e os sistemas de partículas [Reeves, 83] (processos geradores).

O mecanismo de *rendering* e visualização é responsável pela geração das imagens que comporão a animação. Este mecanismo deve reunir as informações sobre a forma, a cor e a textura dos objetos, com as informações sobre o posicionamento dos mesmos, fornecidas pelo módulo de controle da animação. O mecanismo de *rendering* utiliza então algum tipo de algoritmo para a geração das imagens. Os algoritmos mais comuns são: *scanline*, *ray-tracing* e radiosidade. Algoritmos de *scanline* determinam qual objeto será visível em cada pixel varrendo toda a imagem, uma linha de cada vez. Os algoritmos de *ray-tracing*, por sua vez, determinam a visibilidade das superfícies traçando raios de luz imaginários, do observador até os objetos da cena. Os métodos de radiosidade são baseados nos modelos de transmissão e reflexão da radiação térmica. Eles assumem a lei da conservação da energia em um ambiente fechado, de modo que toda a energia emitida ou refletida por uma superfície deve ser absorvida por alguma outra superfície, permitindo um tratamento mais apurado das reflexões entre os objetos.

O mecanismo de controle da animação é o responsável pela movimentação dos atores, além da coordenação das interações entre os mesmos, e entre estes e o animador. Ao contrário da modelagem geométrica e da geração de imagens, onde existem técnicas padronizadas, não existe um algoritmo genérico para o tratamento da movimentação/interação de objetos em uma animação. Na verdade, existem apenas casos de estudo, onde uma determinada técnica de controle é aplicada a um certo número de modelos. Não restam dúvidas, portanto, que esta é a tarefa mais desafiadora dentro da área de animação por computador.

A seção seguinte objetiva apresentar uma visão geral da animação modelada por computador, sob o ponto de vista do controle do movimento/interação de objetos.

III.2 - CONTROLE DO MOVIMENTO E DAS INTERAÇÕES EM UMA ANIMAÇÃO - VISÃO GERAL[4]

Com relação ao movimento, cada quadro de uma animação expressa uma parte do percurso sendo realizado pelos atores no âmbito da estória e da movimentação individual do ator enquanto realiza sua trajetória. Por exemplo, na parte superior da Figura III.2, o ator segue uma trajetória sem nenhum movimento individual, enquanto na parte inferior da mesma figura, o ator segue a mesma trajetória, mas com o movimento individual de rotação.

[4] Baseada em [Magalhães, 95].

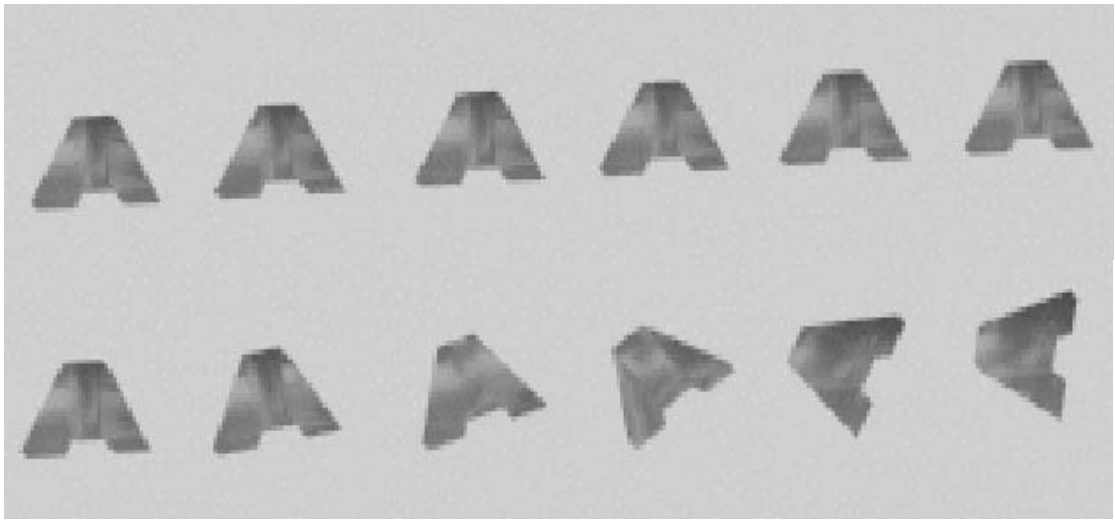

Figura III.2: Diferença entre movimento próprio do ator e movimento no ambiente.

Assim, duas questões básicas devem ser tratadas com relação aos movimentos em animação: a questão do movimento próprio (comportamento individual do ator, com referência no seu centro de massa) e a questão do movimento no ambiente (cuja referência é a origem de um sistema de coordenadas global). Esta última engloba interações com o ambiente e com os outros atores. A Figura III.3 ilustra os conceitos discutidos, no caso de atores não articulados.

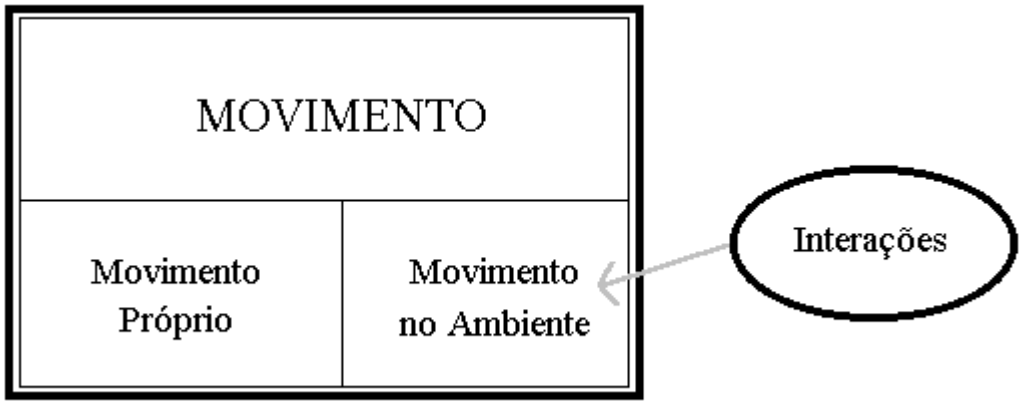

Figura III.3: Relação mov. próprio x mov. no ambiente, para atores não articulados.

Quando os atores são articulados, ocorre uma hierarquização desses conceitos, pois o movimento próprio do ator como um todo, envolve movimentos (próprios e no ambiente) de cada uma das partes que os compõem. A Figura III.4 é a adaptação da Figura III.3, para o caso de objetos articulados.

A discussão anterior foi realizada em um nível onde o controle do movimento efetivamente acontece (nível concreto). [Zeltzer, 85] ainda subdivide este nível de controle em outros dois níveis: o nível de animação guiada e o nível do animador.

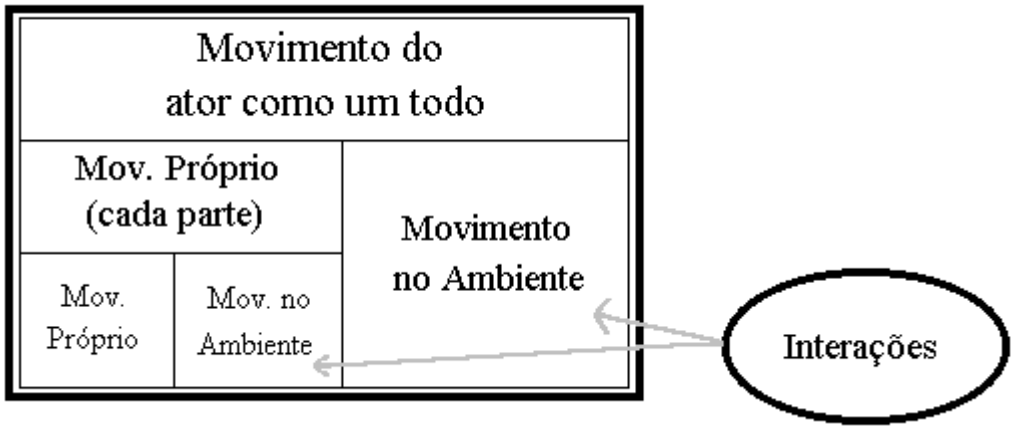

Figura III.4: Relação mov. próprio x mov. no ambiente, para atores articulados.

No nível de animação guiada, o movimento é explicitamente descrito, sem nenhum tipo de abstração. Neste nível, podem ser usados aparelhos específicos para a aquisição dos dados cinemáticos de uma figura móvel, dados estes que serão usados para o controle da figura animada. Entretanto, o tipo mais comum de animação guiada é a animação por *keyframes*, onde quadros-chave são fornecidos pelo animador, e os quadros intermediários são calculados por interpolação.

No nível do animador, o movimento é especificado algoritmicamente, provendo ao animador um controle completo sobre ele. Por esta razão, o nível do animador é apropriado para a definição de movimentos complexos (que são muito difíceis de serem criados com a animação guiada, por causa do pouco controle que o animador exerce sobre o movimento). A desvantagem do nível do animador sobre a animação guiada é a dificuldade de uso, que exige o conhecimento de uma linguagem de programação. A Figura III.5 ilustra o nível concreto de controle de movimento.

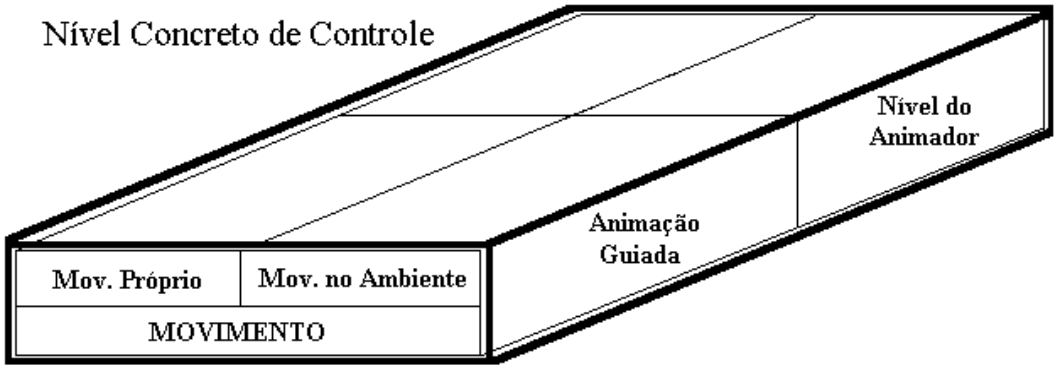

Figura III.5: O nível concreto de controle.

Com o intuito de simplificar a tarefa do animador ao especificar uma animação, é definido o nível de intenção, com maior grau de abstração que o nível concreto. Neste nível de controle, também presente em [Zeltzer, 85], o animador especifica apenas as "linhas gerais" de um movimento, deixando que o sistema resolva os detalhes do movimento. Os atores são dotados de personalidade e habilidades próprias. Este nível está mais próximo da realidade, pois o animador age

como um diretor de teatro, especificando apenas as tarefas dos atores (nesse nível, o animador diz “ande rápido naquela direção”, e não “saia do ponto A e vá até o ponto B com tal aceleração”, por exemplo). É necessária uma base de conhecimentos, para que o sistema possa realizar as tarefas desejadas (nesta base de conhecimentos estão incluídas não só as definições dos movimentos, mas também a “personalidade” de cada ator, para determinar seu comportamento em certas situações).

No nível de intenção também é possível fazer uma subdivisão em dois níveis: o nível do ator individual e o nível das interações. No nível do ator individual são definidas as tarefas individuais do mesmo, além da sua “personalidade”, isto é, suas emoções e reações diante de determinadas situações. No nível das interações são definidas as tarefas coletivas, ou seja, aquelas que envolverão mais de um ator.

A Figura III.6 ilustra os dois níveis de controle de movimento/interação discutidos nesta seção.

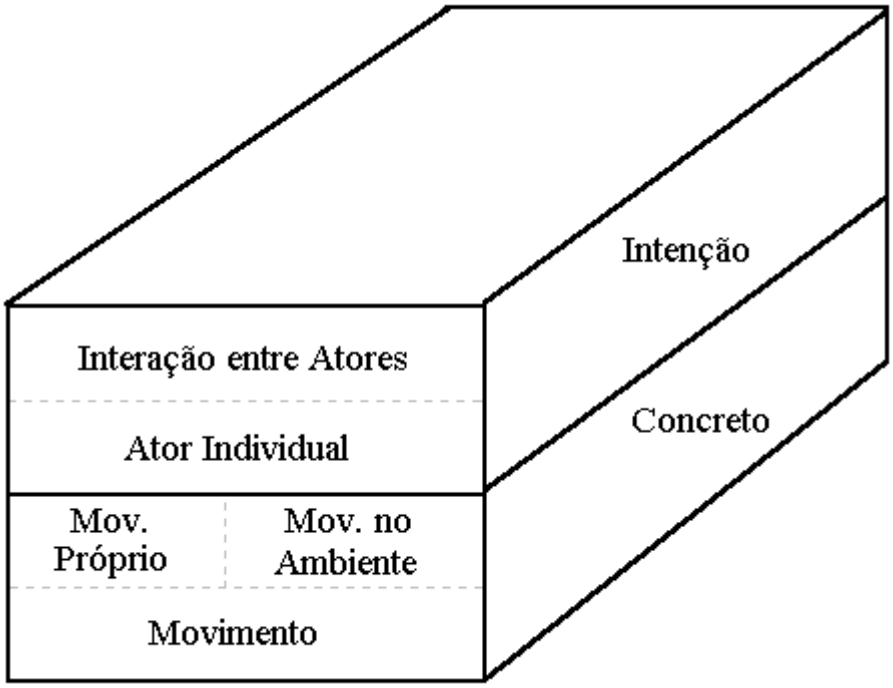

Figura III.6: Os níveis de controle do movimento em animação por computador.

Na seção seguinte o problema do movimento em animação por computador será visto mais detalhadamente e serão mostradas diversas técnicas para seu controle (no nível concreto). Na seção III.4, será tratado o problema das interações envolvendo atores.

III.3 - TÉCNICAS PARA CONTROLE DE MOVIMENTOS

O controle de uma animação, no nível concreto, é baseado na manipulação de parâmetros dos objetos da cena ou da imagem (iluminação, câmera, etc). A principal tarefa do animador é selecionar e controlar os parâmetros apropriados para a obtenção do efeito desejado [Badler, 95]. Os parâmetros a serem controlados e as técnicas utilizadas para este controle servem como método para a classificação dos tipos de animação a serem vistos nesta seção.

III.3.1 - Animação por Interpolação

Na animação tradicional, o animador desenha o ator em uma posição e depois faz outro desenho, com o ator em uma segunda posição. Estes dois desenhos são normalmente passados para outro desenhista, que então desenhará os quadros intermediários (chamados *inbetweens*). A animação por interpolação de *keyframes*, é a transposição desta técnica para os computadores.

A interpolação pode ser linear (trazendo resultados nem sempre satisfatórios) ou seguir alguma curva de interpolação. Em [Reeves, 81] é apresentado um método que permite ao animador obter maior controle sobre a interpolação. Neste método são usados os chamados *moving points*, que são curvas no espaço e no tempo, que controlam tanto a trajetória quanto a "dinâmica" de certos pontos dos quadros-chave. Dessa maneira, é possível controlar a aceleração do movimento interpolado.

Entretanto, a essência da animação por interpolação de *keyframes* é a determinação de correspondências entre os quadros-chave, isto é, quais pontos de um quadro-chave deverão ser mapeados em quais pontos do quadro-chave seguinte. É exatamente neste ponto que se encontram as maiores dificuldades na automatização deste processo, já que cada *keyframe* é uma projeção bidimensional de objetos 3D e, portanto, informações são perdidas (informações estas que podem ser essenciais ao longo da interpolação). Na animação tradicional este problema não existe, pois o artista conhece (ou pelo menos é capaz de imaginar) o modelo 3D do ambiente e dos objetos da cena.

Esse problema pode ser resolvido se for usada a interpolação paramétrica ([Badler,95] e [Wyvill, 90]), onde são dados valores-chave para certos parâmetros da imagem, valores estes que serão interpolados nos *inbetweens*. Com a imagem definida a partir de parâmetros, não há mais o problema da perda das informações tridimensionais, já que elas estão embutidas nos parâmetros a serem interpolados.

Um exemplo de interpolação paramétrica é a metamorfose de superfícies, quando estas são modeladas por algum tipo de função, que terá seus parâmetros interpolados. Em [Preston, 94], por exemplo, é apresentado um modelo para animação utilizando NURBS (*Non-Uniform Rational B-Splines*). Os objetos são modelados por NURBS, e modificações em parâmetros destas curvas (através de manipulação direta na interface) alteram a forma dos mesmos.

III.3.2 - Animação Cinemática

Ao invés de determinar posições-chave para um parâmetro e interpolá-las, o animador pode especificar uma posição (ou valor) inicial e uma função no tempo que descreva as modificações deste parâmetro. Por exemplo, é possível animar a queda livre de um objeto, colocando-o em uma altura inicial e calculando sua posição em cada quadro a partir de sua equação de queda livre.

Este método de controle é chamado cinemático porque os movimentos dos objetos são controlados por funções de posição, velocidade e aceleração.

III.3.3 - Animação Dinâmica

Quando o objetivo de uma animação é a simulação de um processo físico, um modelo mais sofisticado que o cinemático deve ser utilizado para que o resultado final apresente um aspecto mais realista. Esse modelo é o modelo dinâmico, que utiliza variáveis que foram desprezadas pelo modelo cinemático.

O modelo dinâmico usa forças e torques aplicados aos objetos, além dos respectivos momentos de inércia, para determinar os movimentos. Assim, no exemplo da queda livre, o movimento seria calculado a partir do peso do objeto, além de outras forças que poderiam estar atuando sobre o mesmo.

A grande vantagem do modelo dinâmico é criar modelos fisicamente corretos. Entretanto, é exigido o conhecimento de todas as forças que atuam sobre os objetos, o que nem sempre é simples ou possível. Assim, o modelo dinâmico apresenta um grau de complexidade maior que o cinemático, com maior número de variáveis a serem controladas.

De uma maneira geral, as simulações dinâmicas apresentam resultados mais realistas que as cinemáticas, mas o controle do movimento através das forças e torques não é tão natural ao animador. Por essa razão, o animador tem que fazer vários experimentos, até a obtenção do resultado desejado na animação dinâmica.

Portanto, a escolha entre a utilização do modelo cinemático ou dinâmico deve ser muito criteriosa. Em [Camargo, 95b] é traçado um paralelo mais detalhado entre modelos cinemáticos e dinâmicos, inclusive através do desenvolvimento de exemplos dos dois modelos.

III.3.4 - Animação por Cinemática Inversa

Recentemente, a cinemática inversa tem ganho crescente importância na simulação de estruturas articuladas, por ser um método de descrição de "nível mais alto", como citado em [Wyvill, 90]. Alguns sistemas de animação, como o Autodesk 3D Studio, têm incluído facilidades de cinemática inversa em suas funcionalidades.

A cinemática inversa trata do posicionamento de uma estrutura articulada a partir do posicionamento e orientação da extremidade desta estrutura, utilizando um algoritmo para determinar as posições e orientações das juntas intermediárias[5]. Por

[5] A cinemática inversa não é o único método para a simulação de estruturas articuladas. Elas também podem ser simuladas por cinemática direta, onde a posição e a orientação de cada junta são determinadas a cada instante, a partir de um conjunto de equações matriciais. Também é possível a simulação de estruturas articuladas através de dinâmica direta ou inversa.

exemplo, na simulação de um braço, basta determinar a posição e orientação da mão para que as posições do antebraço e do braço sejam encontradas.

Normalmente, mais de um resultado é possível para um problema de cinemática inversa, pois várias configurações das juntas podem levar a extremidade à mesma posição. Por esta razão, restrições internas são necessárias, a fim de se obter um único resultado.

III.3.5 - Animação por Dinâmica Inversa

A dinâmica inversa provê meios de determinar os torques que são necessários para a realização de determinado movimento, uma vez conhecidas todas as forças que atuam sobre o objeto que se deseja movimentar.

Na dinâmica inversa se faz a seguinte pergunta: “quais torques devem ser aplicados para que o extremo atinja determinada posição?”. Na cinemática inversa, por outro lado, a pergunta seria: “quais devem ser as posições das juntas para que o extremo esteja nesta posição?”.

A dinâmica inversa não tem sido tão utilizada quanto a cinemática inversa, pois ela apresenta um grau de complexidade muito grande.

III.4 - INTERAÇÕES ENVOLVENDO ATORES

A evolução dos modelos de controle de movimento permitiu grandes desenvolvimentos no relacionamento entre atores e ambiente. Particularmente, técnicas como a inteligência artificial, a programação orientada a objetos e a visão computacional têm permitido avanços significativos no campo das interações em animação.

Em [Thalmann, 91] as interações são divididas em três tipos: interação ator-ambiente, interação ator-ator e interação animador-ator. Ainda no mesmo artigo, os métodos de controle de movimento (MCMs) são classificados como: geométricos, físicos ou comportamentais.

Os MCMs geométricos usam informações de ordem geométrica. Dessa maneira, o movimento é definido em termos de coordenadas, ângulos, etc. Os MCMs físicos usam leis e características físicas como base para o cálculo do movimento, que é definido em termos de massa, momentos de inércia, etc. Os MCMs comportamentais especificam o movimento de um ator em termos de comportamento, em um nível mais alto de abstração.

Utilizando a divisão em dois níveis de controle apresentada na seção III.2 (Figura III.6), pode-se dizer que os MCMs geométricos e físicos se encontram no nível concreto de controle, enquanto os MCMs comportamentais se encontram no nível de intenção, como observado na Figura III.7.

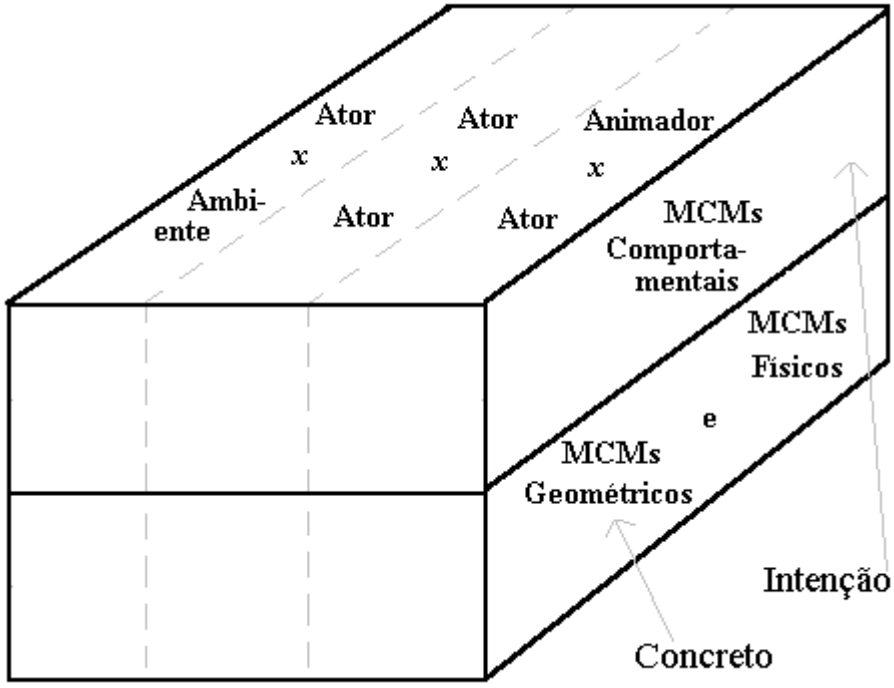

Figura III.7: Os MCMs e os níveis de controle de movimento em animação.

Em [Thalmann, 91], os MCMs não são estudados unicamente nos casos em que ocorrem interações. Eles também são vistos para atores isolados em uma animação (caso visto na seção anterior). Assim, segundo esta classificação, as técnicas cinemáticas (direta e inversa) e a animação por *keyframes* são MCMs geométricos, enquanto as técnicas dinâmicas (direta e inversa) são MCMs físicos. Na categoria de MCMs comportamentais para atores individuais são enquadrados todos os métodos que possuem diretivas para a determinação do comportamento de um ator, sem levar em consideração nenhum tipo de estímulo. Exemplos de MCMs comportamentais para atores isolados são sistemas para simulação de emoções faciais (desde que as emoções não sejam geradas em função de algum estímulo externo), como o mostrado em [Daldegan, 95].

As interações serão vistas mais detalhadamente nas subseções que se seguem.

III.4.1 - Interação Ator-Ambiente

Este tipo de interação ocorre no caso em que os atores estão se movendo em um ambiente do qual eles têm conhecimento. Isto faz com que o movimento dos atores dependa, em parte, do ambiente.

Exemplos típicos de MCMs geométricos para a interação ator-ambiente são métodos para evitar colisão com obstáculos que utilizam a interseção do ator com o obstáculo [Perez, 79], [Lengyel, 90]. Estes métodos fazem com que o ator se mova para pontos do espaço que sejam externos aos obstáculos, isto é, o ator não pode ter nenhum ponto de interseção com os obstáculos.

MCMs físicos para este tipo de interação também são usados para evitar colisões. Entretanto, eles usam métodos analíticos ou parâmetros físicos tais como forças repulsivas, campos potenciais, etc, para evitar as colisões. Modelos de objetos

deformáveis que sofram a ação de forças do ambiente (como os de [Horta, 95]) também se enquadram nesta categoria.

MCMs comportamentais para interações ator-ambiente são meios de controle automático nos quais os atores são capazes de "sensorear" o ambiente e determinar seus movimentos. Em [Wilhelms, 90], por exemplo, são usadas redes de estímulo-resposta para especificar a relação entre estímulos do ambiente e resposta do ator. Em [Costa, 95] e [Costa, 96], os atores são "agentes reativos", isto é, objetos ativos com centros sensoriais capazes de detectar eventos no ambiente associados à visão, audição e tato.

III.4.2 - Interação Ator-Ator

Ocorre quando as ações de um determinado ator são conhecidas por um segundo ator e podem afetar o comportamento deste segundo ator, sem a participação do animador. Isto implica necessariamente na existência de algum tipo de comunicação entre os atores.

MCMs para interações ator-ator são geralmente extensões da interação ator-ambiente, levando em consideração os demais atores da animação, além do ambiente.

MCMs geométricos transmitem informações geométricas entre os atores, tendo como exemplo típico a detecção de colisão geométrica entre dois atores.

Nos MCMs físicos, um ator deve aplicar uma força, que serve para mudar o movimento dos demais atores. Para evitar colisões, essa força deve ser de repulsão entre os atores.

Os MCMs comportamentais para interações ator-ator também são extensões dos utilizados para a interação ator-ambiente. Por exemplo, as redes de estímulo-resposta de [Wilhelms, 90] podem especificar a relação entre estímulos de um ator e a resposta de outro ator. Os agentes reativos de [Costa, 95] e [Costa, 96] também têm capacidade de trocar mensagens entre si.

III.4.3 - Interação Animador-Ator

Neste tipo de interação, o animador pode enviar informações ao ator e este pode responder, criando um fluxo bidirecional de informações.

No nível geométrico, podem ser usados dispositivos 3D para o envio de informações ao ator. Por exemplo, uma *spaceball* pode ser usada para a definição da trajetória a ser seguida por um ator.

MCMs físicos são usados de maneira semelhante. Dispositivos como a *spaceball* podem ser usados para a entrada de forças e torques que atuam sobre um ator.

Do ponto de vista comportamental, este tipo de interação pode ser visto como a troca de emoções entre ator e animador. As emoções do animador são transmitidas ao ator por meio da digitalização em tempo-real da imagem de uma câmera de vídeo e algum programa de processamento de imagem, que identifica a emoção. Em resposta, o ator também pode ter alguma emoção, normalmente gerada por um sistema de animação facial.

III.5 - FIGURAS ARTICULADAS / FIGURA HUMANA

Um grande número de objetos é dotado de articulações, dentre eles destaca-se a figura humana. A modelagem realista de figuras humanas é um dos problemas mais desafiadores da animação por computador, porque os modelos geométricos e matemáticos usados em computação gráfica não são muito adequados para a forma do corpo humano e, além disso, o movimento das juntas é difícil de ser modelado, principalmente por causa do papel dos músculos [Thalmann, 85a].

A animação de figuras articuladas é um campo da animação muito próximo da robótica. Os parâmetros utilizados para controlar os movimentos de objetos articulados são provenientes da robótica, como mostrado em [Camargo, 95a] e [Camargo, 95b]. Entretanto, a animação de objetos articulados por computador vai além dos modelos cinemáticos ou dinâmicos da robótica, pois se pode trabalhar no nível de intenção (animação comportamental). Em [Thalmann, 89], a animação de figuras humanas é analisada sob três aspectos: da criação das formas, do controle de movimento e da deformação de superfícies. O controle do movimento do esqueleto da figura é o aspecto que mais interessa nessa discussão e por isso será visto mais detalhadamente.

Os métodos para o controle de movimento de objetos articulados dividem a animação deste tipo de objeto em:

- Animação por gravação de movimentos: utilizado nas primeiras tentativas de modelar o movimento humano. Uma das técnicas utilizadas era a rotoscopia, na qual as coordenadas das juntas eram digitalizadas a partir de pelo menos duas vistas ortogonais de uma cena previamente gravada em video. Este tipo de técnica é muito tedioso e muito específico para resolver o amplo problema da movimentação humana.
- Animação por *keyframes*: interpolação dos quadros intermediários a partir de quadros-chave (já vista na seção III.3.1).
- Animação por cinemática direta: as posições das juntas são calculadas por meio de uma equação cinemática e passadas ao sistema de animação (ver seção III.3.2).
- Animação por cinemática inversa com restrições de posição: as posições das juntas são calculadas pelo sistema, a partir da posição do extremo da cadeia articulada (seção III.3.4).
- Animação por dinâmica direta: determina a trajetória de todas as juntas a partir de forças e torques aplicados sobre elas (seção III.3.3).
- Animação por dinâmica inversa: conhecidas as forças que atuam sobre o objeto, são determinados os torques que devem ser aplicados para a realização de determinado movimento (seção III.3.5).
- Animação comportamental: o "comportamento" do ator é modelado, variando desde o simples planejamento de trajetórias livres de colisão até complexas interações emocionais entre atores (seção III.4). O movimento dos atores será uma consequência de instruções dadas pelo animador, levando em consideração o "comportamento" modelado para o ator. Um exemplo deste tipo de animação é visto em [Koga, 94], onde os braços do

personagem não movem seguindo uma trajetória dada por uma lei física, mas movem com a intenção de realizar determinada tarefa (geração automática de movimentos complexos de manipulação).

III.6 - SISTEMAS DE ANIMAÇÃO POR COMPUTADOR

Os sistemas de animação por computador são comumente divididos em duas categorias: sistemas gráficos (ou interativos) e sistemas de animação procedimental (ou *scripting systems*). Nos sistemas gráficos, a animação é desenvolvida através de manipulação direta na interface, podendo ser usadas técnicas de *keyframes* ou a especificação de curvas para a geração do movimento. A animação procedimental utiliza roteiros escritos em linguagens próprias para o controle da animação.

III.6.1 - Sistemas Gráficos

Sistemas gráficos criam animações guiadas, segundo a classificação de [Zeltzer, 85] (seção III.2). Não é exigido do animador o conhecimento prévio de nenhuma linguagem de programação específica. Todo o controle da animação é feito graficamente na interface.

O Autodesk 3D Studio [Autodesk, 90], [Malheiros, 95] é um exemplo bastante conhecido de um sistema gráfico de animação que utiliza a técnica de *keyframes*.

Outra técnica bastante utilizada para o controle de movimentos em sistemas gráficos é a manipulação de curvas. Através da edição de curvas, o animador pode criar e alterar o movimento de uma sequência da animação. Esta técnica também permite a separação do controle da posição do controle da velocidade, pois eles podem ser manipulados por curvas distintas [Watt, 93].

Um exemplo de sistema gráfico de animação que utiliza a manipulação de curvas é o Controller [John, 89]. Nesse sistema, o controle do movimento é feito em três etapas. Numa primeira etapa, é dada a informação sobre a posição do objeto, através do desenho de uma curva no plano. Esta curva será desenhada como segmentos de reta, cujos extremos podem ser manipulados. Esses extremos servirão como pontos de controle para a definição de uma B-spline, que definirá a trajetória do ator. Numa segunda etapa, são dadas as informações sobre a aceleração com a qual a trajetória será percorrida. O animador pode optar por qualquer combinação entre o movimento acelerado, desacelerado e com velocidade constante. O sistema também permite que sejam feitas considerações sobre o peso do objeto móvel, de modo que um objeto pesado leve mais tempo para acelerar (e desacelerar) do que um objeto leve. Na terceira etapa do controle do movimento, o animador pode estabelecer outros parâmetros do ator que serão alterados dinâmicamente na animação. Os valores desses parâmetros são calculados por interpolação a partir valores-chave em determinados quadros. Um exemplo de parâmetro que pode ser alterado é a altura em que o ator se encontra, caracterizando a animação tridimensional, pois a curva da primeira etapa era desenhada no plano.

Mais recentemente, em [Snibbe, 95], foi apresentado um sistema gráfico que caracteriza bem a separação entre controle de posição e de velocidade. Esse sistema apresenta ferramentas de manipulação direta para atender os seguintes objetivos do animador: translação temporal ("atinja determinado ponto em determinado instante"; afeta a velocidade com que o objeto percorre uma trajetória), translação espacial (afeta a trajetória, mantendo o intervalo de tempo e a velocidade com que cada segmento é percorrido), escala temporal (altera a duração do segmento de animação, tornando-o mais longo ou mais curto) e modificação de velocidade (altera a velocidade do ator, sem alterar a trajetória e nem a duração do segmento de animação).

A grande vantagem dos sistemas gráficos, como já foi dito, é o fato de não exigirem um conhecimento de programação por parte do animador. Entretanto, eles apresentam algumas desvantagens com relação aos sistemas de animação procedimental, tais como: o movimento espacial só pode ser definido em gráficos bidimensionais; as curvas são limitadas a um parâmetro unidimensional vs. tempo (exemplo, rotação em y vs. tempo, componente vermelho da cor vs. tempo, etc) e a manipulação direta só é possível nos pontos de controle das curvas, o que dificulta a alteração de posições intermediárias.

III.6.2 - Sistemas de Animação Procedimental

Sistemas de animação procedimental utilizam roteiros, que determinam como os parâmetros da animação variarão ao longo do tempo. A vantagem deste tipo de sistema sobre os sistemas gráficos é permitir um maior controle dos movimentos por parte do animador, facilitando a criação de movimentos complexos.

Os roteiros são escritos em uma linguagem própria do sistema, que pode ser uma extensão de uma linguagem de programação já existente, à qual são adicionadas rotinas gráficas e de controle de movimento.

O sistema ASAS (*Actor/Scriptor Animation System*) [Reynolds, 82] é um exemplo "clássico" de sistema de animação procedimental. O ASAS é um sistema orientado a objetos, implementado em Lisp. Os objetos básicos de um script em ASAS são os atores, que são funções especiais dotadas de recursos para a troca de mensagens. Cada ator pode controlar um ou mais aspectos da animação. Enquanto ativo, cada ator será executado uma vez a cada quadro da animação. Um ator pode ser ativado e desativado pelo roteiro (que pode ser visto como a função *main* de um programa) ou por um outro ator. O ator também pode desativar a si mesmo. A linguagem do ASAS provê grande flexibilidade ao animador. O conceito de ator pode levar inclusive ao desenvolvimento de animações comportamentais, pois os atores podem ser definidos para saberem responder a determinados estímulos. O preço dessa flexibilidade é uma sintaxe muito complexa, um forte obstáculo ao programador inexperiente.

Outro sistema de animação procedimental é o CORY [McLachlan, 85], criado com o objetivo de permitir ao animador mudar qualquer parâmetro da cena como uma função do tempo. Foi desenvolvido em linguagem C, utilizando o conceito de objetos. Nesse sistema, os principais objetos do roteiro são as *cues*, que controlam as ações da animação, determinando quando elas começam e terminam. No CORY, o roteiro indica uma sequência de cenas, cada uma representando uma ação e associada a uma *cue*. Ações simultâneas são determinadas associando a mesma *cue* a mais de

uma cena. Embora mais simples que um roteiro em ASAS, um roteiro no CORY ainda é bastante complexo para o leigo em programação.

Tentando criar um sistema acessível a artistas não programadores, [Thalmann, 85b] apresentou o MIRANIM, composto de um sistema orientado ao animador (o ANIMEDIT) e uma sublinguagem de roteiros (a CINEMIRA-2). Com o ANIMEDIT, o animador pode especificar um roteiro de animação completo, sem nenhum tipo de programação. O animador, segundo [Thalmann, 85a], pode "criar atores com seus movimentos e transformações, bem como câmeras virtuais, com seus movimentos e características". Também há recursos para a definição interativa de fontes de luz, bem como para sua movimentação. A CINEMIRA-2 é chamada sublinguagem porque é uma versão menos complexa da linguagem CINEMIRA e está limitada à programação de entidades usadas pelo ANIMEDIT, não sendo possível escrever um roteiro de animação com esta sublinguagem. Uma entidade programada em CINEMIRA-2 pode ser diretamente acessada pelo ANIMEDIT. Esse sistema, no entanto, ainda exige programação procedimental, no caso de movimentos muito complexos.

O Clockworks [Breen, 87] é um sistema de animação orientado a objetos, que engloba a modelagem geométrica de objetos (CSG), o controle de movimentos, o *rendering* (que pode ser feito por meio de várias técnicas, dentre elas, *ray-tracing* e *scanline*) e o pós-processamento de imagens (através da superposição de imagens, útil quando a ação ocorre em frente a um fundo fixo). Pode ser usado como ferramenta de projeto, através da criação e visualização de formas complexas. Pode também ser utilizado como ferramenta de animação. Para o controle do movimento, o Clockworks pode utilizar a interpolação de *keyframes* ou uma linguagem de roteiros. A metodologia de *keyframes* é utilizada através de ferramentas interativas, mas sempre é gerado um roteiro para ser executado. Por esta razão este sistema é classificado como um sistema de animação procedimental.

Mais recentemente, [Zeleznik, 91] desenvolveu outro sistema de modelagem e animação orientado a objetos, que facilitou a integração de vários paradigmas de animação. Os objetos podem ser geométricos ou não-geométricos (câmeras, luzes, etc) e trocam mensagens entre si. A lista de mensagens de um objeto determina seus parâmetros variantes no tempo e seu comportamento. Esta lista pode ser alterada pelo animador e por outros objetos (interação ator-ator e ator-animador). Este sistema também explora as noções de controle global e controle local [Camargo, 95a], porque as mensagens enviadas aos objetos são relativamente abstratas, cabendo ao próprio objeto definir como ela o afetará. A mensagem determina o *que* será feito, e o objeto determina *como* fazê-lo. O roteiro de animação deve ser escrito numa linguagem própria, capaz de descrever objetos e suas mensagens.

A análise dos sistemas de animação procedimental existentes permite concluir que este tipo de sistema é mais poderoso que os sistemas gráficos, pois permite a criação de movimentos complexos. O grande desafio, portanto, é criar sistemas com roteiros simples, utilizando ferramentas interativas da melhor maneira possível, de modo a torná-los acessíveis a artistas não programadores.

III.7 - OUTRAS CLASSIFICAÇÕES

Até aqui, já foram vistas várias classificações possíveis para animação por computador: auxiliada ou modelada por computador; por interpolação, por cinemática (direta e inversa) ou por dinâmica (também direta e inversa). Também foi visto como são classificados os métodos de controle do movimento em animação: nível concreto ou nível de intenção, MCMs geométricos, físicos ou comportamentais. Com relação aos sistemas de animação, eles foram classificados em gráficos ou de animação procedimental.

Além dessas, ainda há outras classificações relacionadas à animação por computador, que serão vistas a seguir.

III.7.1 - Animação x Simulação

É usual classificar como animação os trabalhos em que uma grande exatidão física dos movimentos não é exigida. Nesta categoria enquadram-se as animações voltadas para o entretenimento.

Um exemplo de sistema para o desenvolvimento de animações (e não de simulações) é o TicTacToon [Fekete, 95], um sistema profissional para a criação de animações bidimensionais. Este sistema foi desenvolvido com a preocupação de permitir ao animador trabalhar de maneira semelhante à da animação tradicional, baseada em desenhos no papel. O objetivo é automatizar o processo tradicional de criação de animações, sem grande impacto para o animador. O TicTacToon está sendo usado para uma grande variedade de produções, incluindo comerciais de TV, vinhetas, etc.

A simulação normalmente exige a exatidão física dos movimentos, pois ela está associada à representação de algum processo do mundo real (processo físico, movimento de robôs, visualização científica, etc). Simulações são comumente usadas para o estudo de problemas científicos e para a representação e interpretação de experimentos. A simulação, como citado em [Palamidese, 94], é “vantajosa para a aplicação que está sendo considerada, uma vez que ela permite um entendimento global e sintético dos resultados principais ou revela aspectos que não poderiam ser observados a partir de uma imagem estática”.

O ANIMA [Moltedo, 93] é uma ferramenta interativa desenvolvida para a produção de simulações a partir de resultados numéricos de experimentos científicos. Ele permite o desenvolvimento da simulação a partir de uma interface gráfica, sem a necessidade de conhecimento de linguagem de programação por parte do animador (no caso, o cientista).

III.7.2 - Tempo x Eventos

A animação por computador normalmente é construída discretizando o movimento em instantes específicos de tempo. A variável tempo é utilizada como um relógio de quadros, cujos passos determinam os instantes em que o movimento deve ser "fotografado", para o *rendering*. Esse método funciona muito bem para sistemas com comportamento contínuo.

Entretanto, existem situações, como nas simulações de fenômenos físicos, em que eventos podem acontecer assincronamente em relação ao relógio (isto é, entre um quadro e outro) e causar descontinuidades no comportamento do sistema simulado. Nessas situações, o tratamento puramente temporal da animação pode não ser adequado.

Em [Kalra, 92], o conceito de evento em animação é formalizado e é criada uma primitiva de tempo chamada unidade de evento (*event unit*). A unidade de evento é composta de três elementos dependentes do estado atual do sistema: um conjunto de regras para o comportamento do sistema antes da detecção do evento, uma variável lógica para indicar a ocorrência do evento e um conjunto de regras de comportamento para depois da detecção do evento. Através da composição de unidades de evento, é possível simular o comportamento de um sistema.

O conceito de eventos está intimamente ligado à animação comportamental. Por isso, tende a se tornar um aspecto cada vez mais importante na animação por computador.

III.7.3 - *Playback* x Tempo-Real

Para produzir a sensação de continuidade para o olho humano, as imagens de uma animação devem ser mostradas a uma taxa de 24 quadros por segundo, no mínimo. Entretanto, o processo de *rendering* de cada um destes quadros é demorado e, dependendo da complexidade e do grau de realismo, pode demorar vários minutos ou mesmo horas. A visualização da animação na taxa de quadros por segundo desejada só é possível depois que o *rendering* de todos os quadros estiver terminado e eles estiverem armazenados. É o chamado *real-time playback* (a animação é visualizada, mas não produzida em tempo-real).

Hoje em dia, com a evolução do hardware gráfico, do multiprocessamento e da velocidade dos computadores, já existem recursos para a produção de animações em tempo-real (isto é, são gerados 24 quadros por segundo, ou mais). Um exemplo de *toolkit* para o desenvolvimento de aplicações gráficas tridimensionais (simulação visual e realidade virtual, por exemplo) em tempo-real é o IRIS [Rohlf, 94]. O IRIS utiliza o multiprocessamento, possuindo recursos para resolver o problema do particionamento de tarefas entre os processadores e da sincronização entre os processos, obtendo assim o desempenho máximo das estações gráficas.

III.8 - RESUMO

Neste capítulo, a animação e os sistemas de animação por computador foram classificados e analisados segundo vários aspectos. A seguir será mostrado um resumo com as principais classificações feitas nesse capítulo. Classificações semelhantes podem ser encontradas em [Pueyo, 88], [Thalmann, 85a] e [Zeltzer, 85].

- Classificações gerais de animação:
    - Histórica: animação auxiliada ou animação modelada por computador.
    - Quanto ao grau de realismo: animação ou simulação.
    - Quanto ao elemento que altera o estado dos objetos: animação temporal ou orientada a eventos.
- Classificações relacionadas ao controle de movimentos:
    - Quanto ao grau de abstração do controle: nível concreto de controle ou nível de intenção. O nível concreto ainda pode ser subdividido em animação guiada e no nível do animador.
    - Quanto à técnica de controle: animação por interpolação, cinemática, dinâmica, cinemática inversa ou por dinâmica inversa.
    - Quanto às informações usadas para o controle: MCM geométrico, físico ou comportamental.
- Classificações de sistemas de animação:
    - Quanto à interação com o animador: sistema gráfico ou de animação procedimental.
    - Quanto ao desempenho no *rendering* da animação: sistema de *playback* ou de tempo-real.

IV - O SISTEMA DE ANIMAÇÃO DO ProSIm

O ProSIm, como visto no capítulo I, possui ferramentas para o desenvolvimento de animações cinemáticas, dinâmicas, envolvento objetos rígidos simples e articulados, além de objetos deformáveis. O presente trabalho está centrado no TOOKIMA que, dentro do contexto apresentado no capítulo anterior, pode ser classificado como um sistema para o desenvolvimento de animações ou simulações cinemáticas (MCMs geométricos), temporais[6], modeladas por computador, que trabalha no nível concreto de controle. É um sistema de animação procedimental, que pode apenas visualizar, e não gerar, as animações em tempo-real (sistema de *playback*).

A linguagem original do TOOKIMA para a composição de roteiros de animação é uma extensão da linguagem C, provendo tipos de dados e uma biblioteca de funções que permitem o desenvolvimento de animações. Entretanto, a dificuldade que esta linguagem apresenta para usuários leigos em programação levou ao desenvolvimento de uma linguagem de mais alto nível para a composição de roteiros. Esta nova linguagem, por sua vez, serviu de base para o desenvolvimento de uma interface, onde o roteiro pode ser construído interativamente. A Figura IV.1 ilustra os possíveis níveis para o desenvolvimento de uma animação utilizando o TOOKIMA, e os usuários característicos de cada um dos níveis.

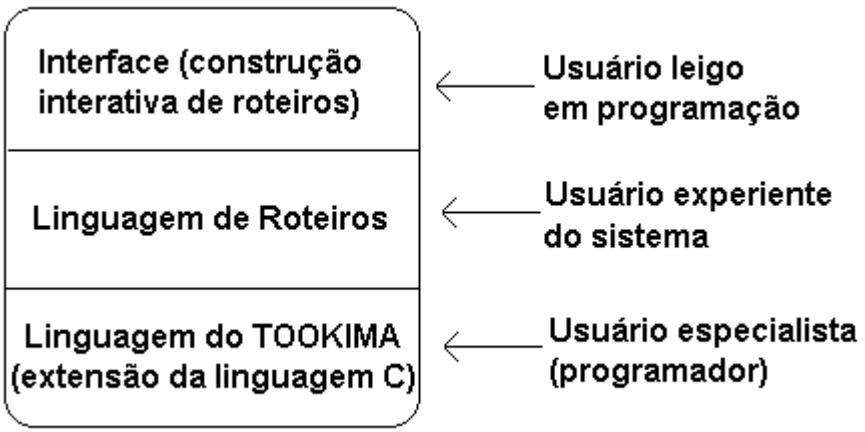

Figura IV.1: Níveis de roteiros de animação.

As seções seguintes mostrarão mais detalhadamente cada um desses níveis. Na seção IV.1 será apresentada a linguagem do TOOKIMA, a única que existia antes do início do presente trabalho. A seção IV.2 mostrará uma visão geral da linguagem de roteiros, que foi a primeira etapa do presente trabalho (no capítulo V esta linguagem será vista em detalhe). Para concluir, a seção IV.3 apresentará uma visão

[6] O TOOKIMA não dispõe de recursos específicos para o tratamento de eventos, mas pode ser utilizado como ferramenta de visualização de animações orientadas a eventos, cujo tratamento é feito por algum programa externo, e o resultado passado ao TOOKIMA, como a animação mostrada em [Camargo, 94].

global da construção interativa de roteiros, através da interface gráfica desenvolvida na segunda parte deste trabalho (esta interface será detalhada no capítulo VI).

IV.1 - LINGUAGEM DO TOOKIMA

A linguagem do TOOKIMA é uma extensão da linguagem C desenvolvida por [Hounsell, 92a]. O roteiro de uma animação, nesse nível, é um programa em C, que utiliza funções e tipos de dados da biblioteca definida. Por ser um programa em C, o roteiro é compilado e então é criado um programa executável, que utilizará o *renderer* para gerar os quadros da animação.

Na criação de uma animação com esta linguagem, são utilizados alguns conceitos importantes:

Script (roteiro) - é um programa principal que gerencia as cenas, sendo responsável pelo macro controle de animação. Utilizando uma hierarquia (árvore) para representar o processo de descrição de uma animação, o roteiro seria a raiz (contém uma visão geral da animação, de forma sucinta).

Cena - desdobramento do roteiro, que descreve a animação com maior detalhamento. Os desdobramentos das cenas serão chamados sub-cenas, aumentando cada vez mais o nível de detalhes.

Frames - são as folhas da árvore, onde não são mencionadas as mudanças mas, mostradas quadro a quadro.

Cue - outro conceito muito importante, utilizado no sistema CORY [McLachlan, 85]. *Cue* é um intervalo de tempo associado a uma cena (intervalo de tempo em que a cena acontecerá). Cada cena está associada a uma *cue* e a cada *cue* estão associadas uma ou mais cenas. Cada *cue* possui dois valores que identificam o instante em que a cena começa e o instante em que ela termina.

Relógio Absoluto - varia de 0 a TotalTime (variável definida no roteiro; é o tempo total da animação), com passo de 1/Rate (Rate é a taxa de quadros por segundo de animação, também é definida no roteiro).

Relógio Local - relógio determinado internamente à cena.

Camera Sintética - permite efeitos semelhantes aos obtidos com uma câmera de cinema (*zoom*, *travelling*, etc.).

Iluminação - permite a criação de fontes luminosas de três tipos: pontual (emitindo luz em todas as direções), *spot* (situada em um ponto e emitindo luz em uma direção especificada) e solar (emite raios paralelos).

Atores x Decoração - Atores são objetos que podem se mover, e não estão restritos a estruturas provenientes do modelador geométrico; podem também ser parâmetros de câmera (oservador, centro de interesse, etc), posição ou direção de fonte de luz, etc. Decoração, por sua vez, deve ser uma estrutura proveniente do modelador geométrico e não tem a capacidade de se mover durante o transcorrer da animação.

Com o conhecimento desses conceitos, é possível criar um quadro com a estrutura de um roteiro de animação com a linguagem do TOOKIMA (Figura IV.2).

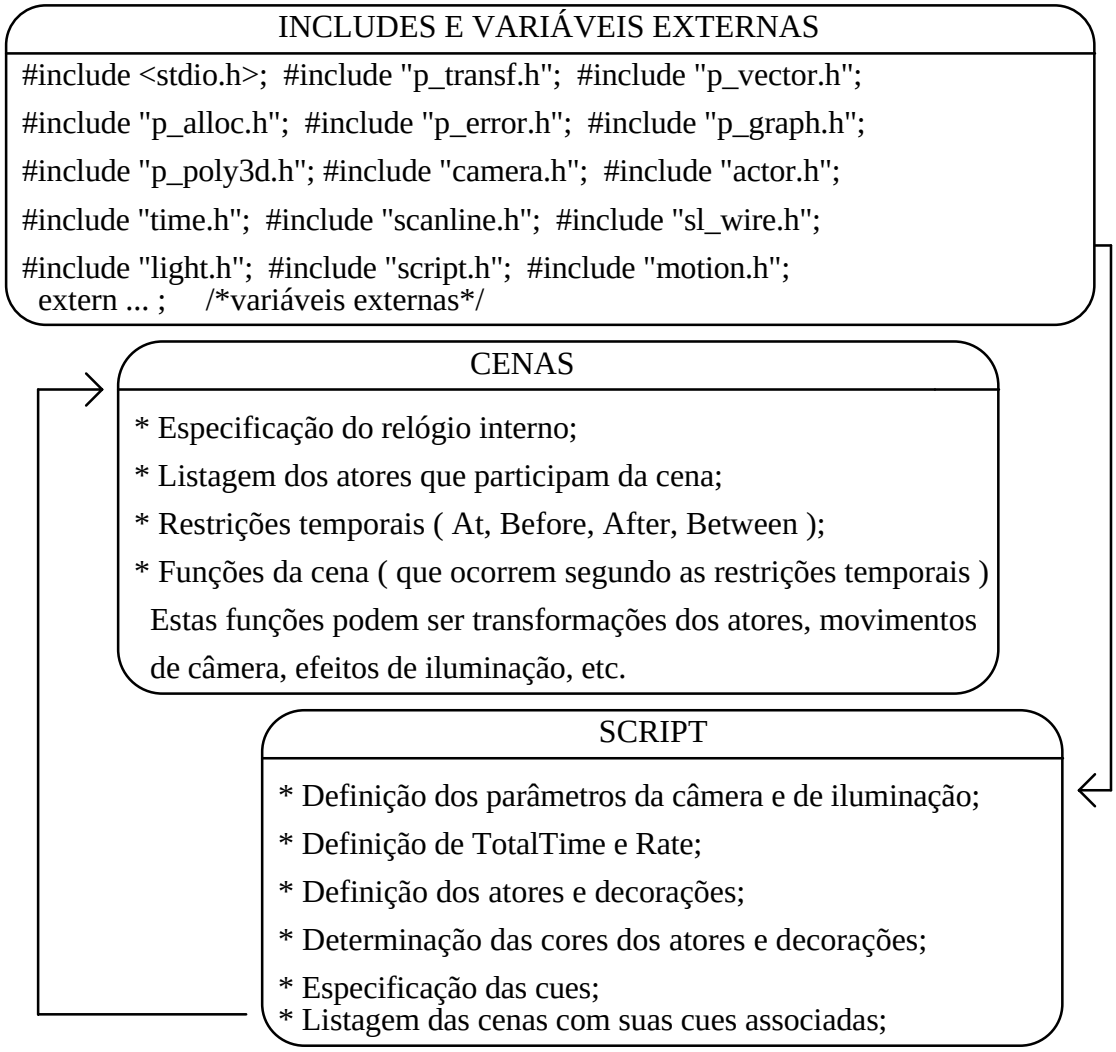

Figura IV.2: Estrutura de um roteiro utilizando a linguagem do TOOKIMA.

Uma descrição mais detalhada desta linguagem foge do escopo desta dissertação, uma vez que ela já foi apresentada em [Hounsell, 92a] e [Raposo, 95a]. A Figura IV.2 já cumpre o objetivo desta seção, que é mostrar o grau de dificuldade de construção de um roteiro com esta linguagem, uma vez que estão envolvidos conceitos nem sempre óbvios e um complexo sistema de inter-relacionamento entre as estruturas.

IV.2 - LINGUAGEM PARA COMPOSIÇÃO DE ROTEIROS

A experiência com a utilização de sistemas de desenvolvimento de animação, levou à conclusão de que é mais natural para o animador imaginar os movimentos de cada elemento da cena (seja ator, câmera ou iluminação) no decorrer do tempo do que imaginar todos os movimentos que estão ocorrendo num determinado instante de tempo. Ou seja, é melhor tratar cada elemento independentemente.

A Figura IV.3 ilustra uma situação exemplo. Nela, o ator A se movimenta num intervalo de tempo, o ator B se movimenta na primeira metade deste intervalo, e o ator C na segunda metade do mesmo. É muito mais natural para o animador especificar que o ator A se movimentará de t0 a t2, B se movimentará de t0 a t1 e C, de t1 a t2, do que tentar imaginar a linha do tempo e especificar que entre t0 e t1 A e B estão se movimentando e entre t1 e t2, A e C.

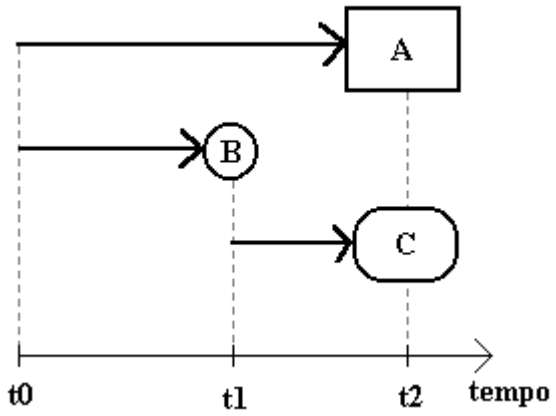

Figura IV.3: Três atores se movendo, ao longo do tempo.

Para atingir a simplificação desejada, alguns dos conceitos utilizados pelo TOOKIMA foram eliminados. Por exemplo, não há mais a hierarquia (*script*, cena, *cue* e *frame*) para a descrição da animação e nem a diferenciação de ator e decoração (todos são tratados como objetos, que podem ter ou não movimentos). Na nova linguagem, a descrição da animação é feita por módulos, onde os diferentes elementos da animação são tratados. Dessa maneira, tudo o que diz respeito à câmera, por exemplo, fica no módulo `CAMERA` do roteiro; tudo que diz respeito à iluminação fica no módulo `LIGHTS`, e assim por diante. Pode-se dizer que o roteiro é orientado aos elementos da cena, e não mais ao relógio temporal da cena, como na linguagem do TOOKIMA.

São possíveis oito módulos num roteiro:

- `GENERAL:` onde são colocados parâmetros gerais da animação (tempo total e taxa de quadros/s), além de variáveis globais.
- `ACTORS:` define os objetos que participarão da cena (suas características e movimentos).
- `GROUPS:` cria grupos de atores, que podem ser movimentados em conjunto.

- `CAMERA:` especifica parâmetros de câmera e seus movimentos.
- `LIGHTS:` são definidas as fontes de luz do ambiente e a cor de fundo.
- `RENDER:` determina o intervalo da animação que será passado ao *renderer*.
- `OUTPUT:` determina a qualidade das imagens geradas e o nome das mesmas.
- `TRACKS:` define trajetórias pela interpolação de curvas, a partir de pontos de controle.

Para garantir a reusabilidade de componentes do roteiro, tais como os atores, a câmera e a iluminação, é possível defini-los em arquivos externos, que poderão compor bibliotecas reutilizáveis. Os atores, a câmera e a iluminação não precisam estar definidos explicitamente no roteiro. No roteiro, basta fazer uma referência ao arquivo de uma das bibliotecas. Além disso, cada arquivo de definição de ator deve referenciar um arquivo com o modelo geométrico do mesmo e outro com a definição da sua textura. É possível também definir trajetórias (*tracks*), que são usadas para a movimentação de atores, de câmera (observador ou centro de interesse) e de fontes de luz. A Figura IV.4 mostra o possível relacionamento entre estas bibliotecas e o roteiro de uma animação.

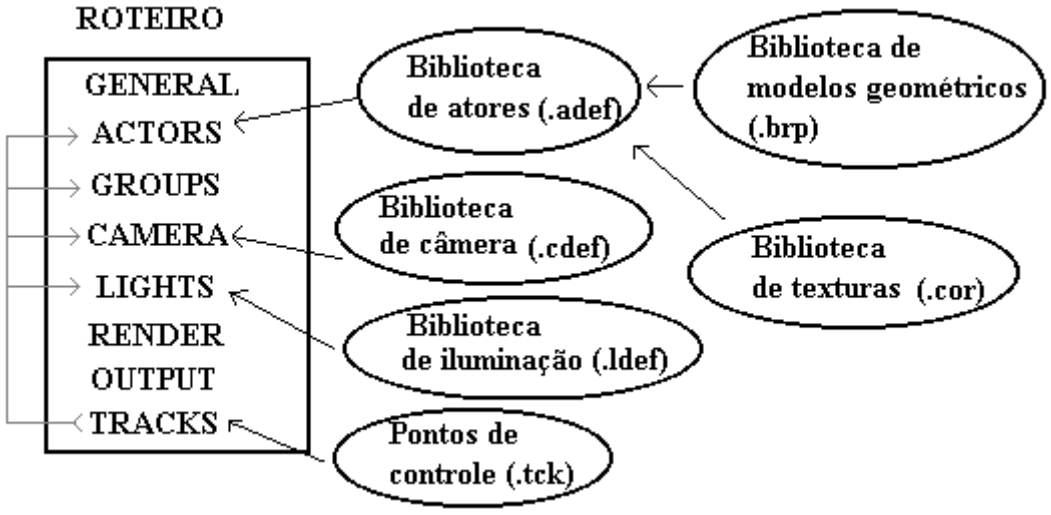

Figura IV.4: Bibliotecas reutilizáveis em roteiros de animação.

A linguagem para a composição de roteiros será detalhada no próximo capítulo.

Para tornar sua implementação menos custosa, a linguagem de roteiros foi implementada "sobre" a linguagem do TOOKIMA (isto é, sendo traduzida para ela depois), uma vez que o TOOKIMA já oferecia grande parte das funções necessárias. Essa tradução pode parecer levar a um significativo aumento no tempo de geração de uma animação, mas o atraso que realmente ocorre não é significativo (a etapa de *rendering* é o verdadeiro "gargalo" de todo o processo).

Após a tradução para a linguagem do TOOKIMA, o roteiro é compilado e serão gerados *n* roteiros para o *renderer* (onde *n* é o número de quadros da animação), que terá então condições de produzir os quadros. Entretanto, o processo de tradução do roteiro para a linguagem do TOOKIMA, sua compilação e geração

dos *n* roteiros para o *renderer* (escritos numa linguagem própria do *renderer*) é transparente para o usuário, que só obterá o resultado, na forma dos *n* quadros ou de uma animação MPEG [Le Gall, 91]. A Figura IV.5 ilustra esta sequência de passos para a geração de uma animação.

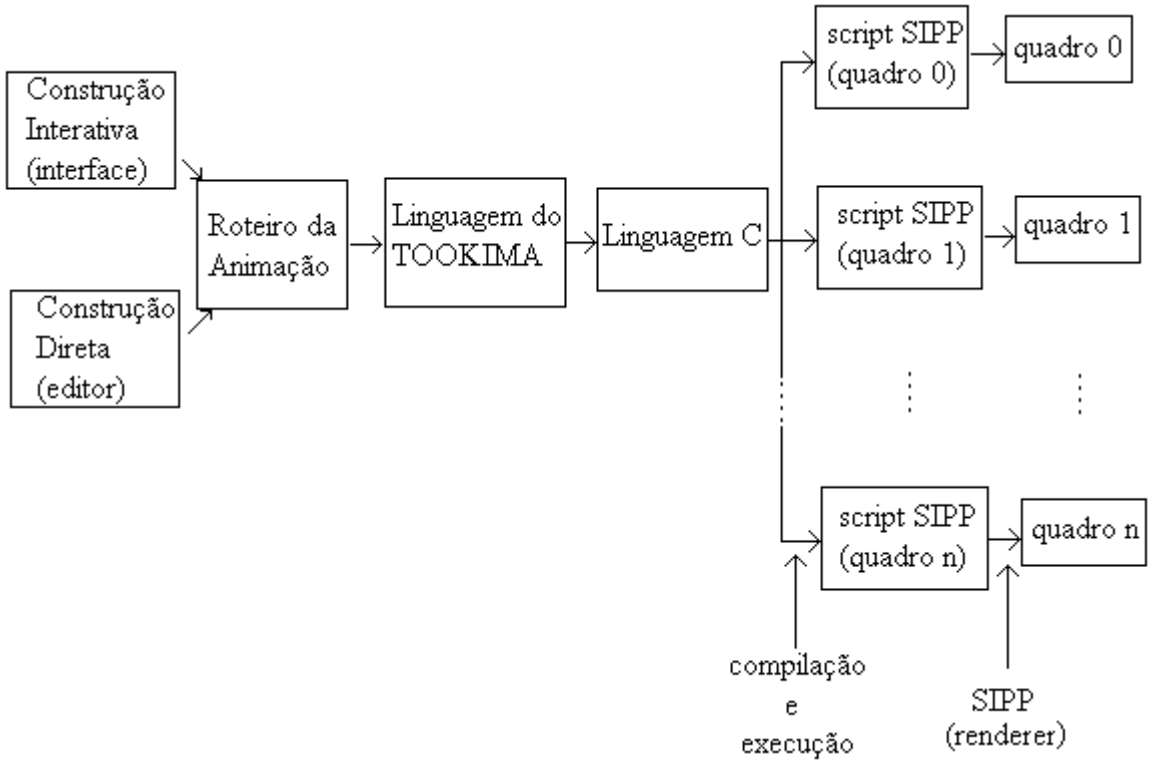

Figura IV.5: Fluxo de informações durante a geração de uma animação.

IV.3 - CONSTRUÇÃO INTERATIVA DE ROTEIROS

O nível mais abstrato para o desenvolvimento de animações com o ProSIm é o nível interativo, através da interface gráfica. Esse nível, no entanto, não pretende tornar o nível inferior (isto é, o nível da linguagem de roteiros) transparente para o usuário. O objetivo da interface é tornar o usuário apto a lidar diretamente com o roteiro da animação, de acordo com os princípios de *End User Programming Language*, discutidos no capítulo II.

A interface possui os elementos gráficos tradicionais (menus, botões, caixas de diálogo, etc), oferecendo todas as funções necessárias para a construção do roteiro, e também uma área de texto, onde o roteiro pode ser manipulado diretamente (essa área funciona como um editor de textos). À medida que o roteiro vai sendo montado pelos elementos gráficos, ele vai sendo atualizado na área de texto, permitindo com que o usuário vá aos poucos compreendendo a sintaxe da linguagem de roteiros (como proposto por [Eisenberg, 95], a interface "leva" o usuário à programação). Portanto, como mostrado na Figura IV.5, o roteiro pode ser construído, através da interface, de maneira interativa ou direta (através do editor de textos).

O controle dos movimentos é feito em dois níveis pela interface do TOOKIMA: em um nível mais alto, onde aos atores, câmera ou fontes de luz são

associadas trajetórias previamente criadas[7] e em um nível mais baixo, onde os movimentos são definidos diretamente através de comandos do tipo *translate*, *rotate*, *zoom* (para câmera), etc.

Os quadros da animação podem ser gerados pelo *renderer* (SIPP, [Yngvesson, 94]) no formato PPM (*Portable PixMap*) ou TGA não comprimido. É possível também uma pré-visualização em *wireframe* (isto é, só as bordas dos objetos são desenhadas, como se eles fossem de arame, sem a preocupação com texturas, iluminação e eliminação de superfícies escondidas). Neste tipo de pré-visualização serão gerados quadros no formato PBM (*Portable BitMap*). Para a visualização, o sistema permite a geração de animações no formato MPEG (*Motion Picture Expert Group*) [Le Gall, 91]. Este é um formato bastante compacto para a representação de animações, embora apresente perdas na qualidade das imagens. Entretanto, além da animação MPEG, o sistema continua gerando os quadros PPM, para o caso de se desejar editá-los numa animação sem perdas. A codificação para o formato MPEG é feita de maneira transparente ao usuário, utilizando o Berkeley MPEG-1 Video Encoder [Gong, 94], software de domínio público.

A cooperatividade com outras interfaces já existentes foi buscada, sempre que possível, a fim de reduzir o custo de desenvolvimento. Além do citado codificador de MPEG, a visualização da animação MPEG usa o Berkeley MPEG-Player e a manipulação de imagens é feita através do XV [Bradley, 93], ambos de domínio público.

Para manter a consistência com os outros módulos de interface do ProSIm, a interface do TOOKIMA foi desenvolvida com o XView [Heller, 90], para executar no Sistema Unix das máquinas Sun SparcStations (sistemas operacionais SunOS ou Solaris). Esta interface será descrita detalhadamente no capítulo VI.

Também como parte deste trabalho, foram desenvolvidos módulos para a criação dos arquivos de definição de iluminação e texturas. Estes módulos não fazem parte da interface do TOOKIMA, mas criam arquivos que deverão ser utilizados na elaboração de uma animação. Estes módulos também serão descritos no capítulo VI, com exceção do editor de texturas, que será visto no Apêndice B.

[7] A interface também permite a criação destas trajetórias, através de um “editor de trajetórias”.

V - A LINGUAGEM DE ROTEIROS

O uso de roteiros constitui um método prático e usual para automatizar o processo de animação. Um roteiro é uma sequência ordenada de comandos, interpretada de maneira não ambígua pelo sistema de animação, e que deve ser passada ao computador para que ele opere sobre os elementos que compõem a cena.

A linguagem desenvolvida para a criação de roteiros pretende ser intermediária entre a complexa linguagem do TOOKIMA [Raposo, 95a] e a interface.

O roteiro proposto é composto de 8 módulos, que especificarão os parâmetros e funções referentes ao objeto do módulo. Os módulos podem aparecer em qualquer ordem no roteiro (desde que o módulo GENERAL seja o primeiro), ou até mesmo não aparecer (alguns módulos não são obrigatórios). O sinal " ; " indica que o resto da linha do roteiro será considerada como comentário.

Nas seções seguintes, será visto um resumo das funcionalidades e sintaxe de cada um dos módulos da linguagem de roteiros. Uma descrição completa da mesma é encontrada em [Raposo, 95b].

V.1 - MÓDULO "GENERAL"

Neste módulo são colocados os parâmetros gerais da animação: TotalTime (tempo total da animação, em segundos) e Rate (taxa de quadros por segundo). Se houver a necessidade de variáveis globais para a animação, elas devem também ser colocadas neste módulo.

É obrigatória a presença deste módulo (e dos parâmetros Rate e TotalTime) no início do roteiro de animação.

O fragmento de roteiro a seguir mostra o módulo `GENERAL`:

```
GENERAL
     TOTALTIME = 10
     FR_RATE = 20
     double x1 = 7.5
     double x2 = read_file("/usr/input.in")
END
```

A animação do exemplo anterior terá 200 quadros (10 segundos de animação, a uma taxa de 20 quadros/s). A variável x1 é do tipo double e assume o valor 7.5. Já a variável x2 será lida do arquivo /usr/input.in (a cada frame será lido um valor do arquivo, que será armazenado nesta variável). A utilidade da variável lida em arquivo, é que este pode, por exemplo, conter o resultado numérico de uma simulação, cujos valores podem ser usados como parâmetros de movimentos de atores.

Uma variável global pode assumir ainda o valor dado pela função "following_track", usada quando se deseja seguir uma trilha de pontos. A explicação da utilização de trilhas será dada no módulo `TRACKS`.

V.2 - MÓDULO "ACTORS"

Este módulo define os objetos que participarão da cena e suas características. Deve ser dado o nome do arquivo com suas características geométricas e sua posição inicial, além de sua cor (superfície) e de seu método de tonalização (*shading*).

Este módulo também trata dos movimentos dos atores. Cada movimento (translação, rotação e mudança de escala) deve vir acompanhado dos parâmetros que o caracterizam (quantos graus o ator gira, qual a porcentagem da mudança de escala, etc), além do intervalo de tempo em que ocorrerão. A medida de tempo pode ser feita em segundos ou em quadros (*frames*).

A definição das características "físicas" de cada ator e sua posição inicial podem ser dadas num arquivo em separado, que tem a seguinte forma:

```
BRP = "/proj/prosim/objects/esf.brp"
SHADE_TYPE = WOOD
SHADOW = FALSE
COLOR = "/home/usr/color/madeira1.cor"
INITIAL_POSITION = 0., 3., -5.
```

`BRP` define o arquivo B-Rep, com as características geométricas do ator (este arquivo é gerado pelo modelador geométrico). `SHADE_TYPE` determina o tipo da textura do ator (pode ser `WOOD`, `MARBLE`, `GRANITE`, `STRAUSS` - superfícies metálicas, `PHONGS` ou `BASIC`). Esse parâmetro deve ser coerente com o arquivo de cor, definido por `COLOR`. `SHADOW` indica se o ator terá ou não sombra (na verdade, o SIPP - *renderer* - não permite que certos atores tenham sombra e outros não; ou todos têm ou nenhum tem). Esse parâmetro foi colocado para facilitar a compatibilidade com outros *renderers*. O parâmetro `INITIAL_POSITION` determina a posição inicial do objeto; se este parâmetro não estiver presente no arquivo, o programa assumirá que a posição inicial do ator será aquela na qual ele foi modelado (a que existe no arquivo .brp).

Uma alternativa à definição do ator como um arquivo B-Rep é defini-lo como uma sequência numerada de arquivos .brp, sendo que, a cada quadro, apenas um será mostrado. Para isso, ao invés de `BRP`, deve-se usar `GEO_MORPH` da seguinte maneira, no arquivo de definição do ator.

```
GEO_MORPH = "/prosim/brp/esf", 10, FRAME(8)
```

O primeiro parâmetro indica o nome dos B-Reps numerados (no exemplo, a sequência será `/prosim/brp/esf000.brp`, `/prosim/brp/esf001.brp`, ...). O segundo parâmetro indica o tamanho da sequência numerada (no exemplo, a sequência irá de 0 a 9). O terceiro parâmetro determina em que quadro a sequência começará a ser vista na animação (no exemplo, no quadro 8 será visto a `esf000.brp`, no quadro 9, a `esf001.brp`, e assim por diante, até o final da sequência).

De maneira semelhante, a textura de um ator pode ser definida como uma sequência numerada de texturas, de modo que a cada quadro o ator tenha uma textura diferente. Para isso, ao invés de `COLOR`, deve-se usar `CLR_MORPH`, de maneira semelhante a `GEO_MORPH`.

```
CLR_MORPH = "/tmp/cor/texture", 20, FRAME(5)
```

O arquivo de definição será referenciado para cada ator da seguinte maneira no roteiro, dentro do módulo `ACTORS`:

```
ACTORS
     A0: DEFINITION("def/actor0.adef")
     A1: DEFINITION("def/actor1.adef")
END
```

Qualquer um dos parâmetros do arquivo de definição pode ser redefinido no roteiro, se colocado depois do `DEFINITION` do ator. Por exemplo, é possível colocar dois atores iguais em posições iniciais diferentes e dar ao segundo uma outra cor:

```
ACTORS
     A0: DEFINITION("def/actor1.adef")
     A1: DEFINITION("def/actor1.adef")
         INITIAL_POSITION = 5., 5., 5.
         COLOR = "/usr/color/madeira2.cor"
END
```

A Figura V.1 ilustra o relacionamento entre os arquivos usados neste módulo:

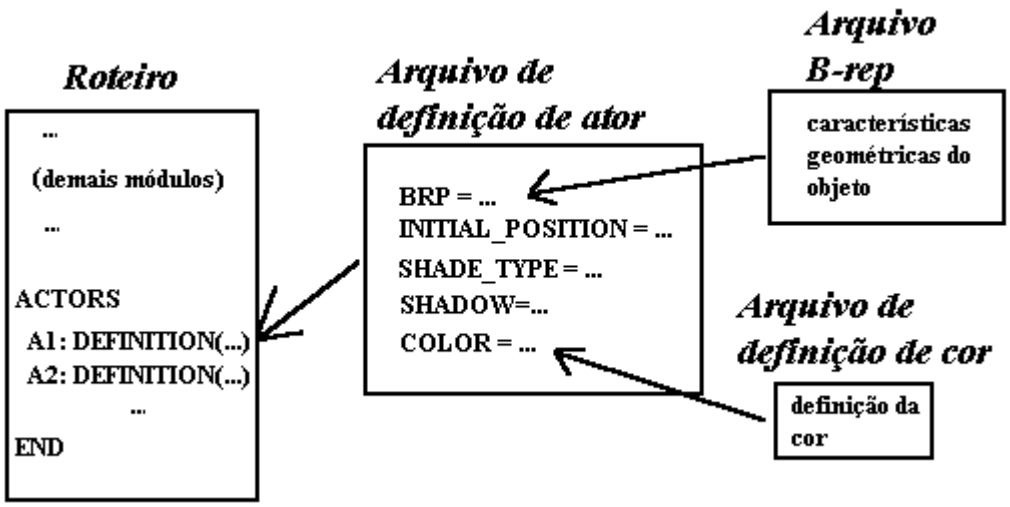

Figura V.1: Módulo `ACTORS`.

Os movimentos dos atores também são especificados neste módulo; eles devem ser colocados após a definição de cada ator. O movimento deve ser sempre precedido pelo intervalo de tempo em que ocorrerá. Os delimitadores de intervalo usados são:

- `At:` determina que o movimento que se segue ocorrerá exatamente naquele instante (ou naquele quadro). Exemplos.: `At(2.0), At(FRAME(60)).`

- `After:` indica que o movimento ocorrerá após determinado instante (ou determinado quadro). Exemplos: `After(FRAME(95)), After(3.0).`
- `Before:` o movimento ocorrerá do início até determinado instante (ou quadro). Exemplo: `Before(FRAME(22)).`
- `Between:` indica que o movimento ocorrerá entre dois intervalos de tempo (incluindo estes dois instantes). Exemplo: `Between (FRAME(4),FRAME(50))` indica que o movimento começa no quadro 4 e termina no 50.

Quando o intervalo se prolonga por mais de um quadro (em todos os casos, exceto quando se usa `At`), é necessário especificar se o valor dos parâmetros do movimento se referem ao movimento de um quadro para outro ou se referem ao movimento total do intervalo (por exemplo, se é dito que um ator deve girar 10º, deve-se saber se ele vai girar 10º por quadro ou se no final do intervalo ele deverá ter girado 10º). Para isso existem as palavras-chave `EACH_FRAME` (indicando que o movimento desejado se dará a cada quadro) e `TOTAL` (indicando que o movimento desejado se completará no final do intervalo especificado).

As linhas do roteiro que indicam os movimentos dos atores devem ter a seguinte forma:

delimitador de intervalo(...) movimento(...) `EACH_FRAME` (ou `TOTAL`)

Os possíveis movimentos de atores, com seus respectivos parâmetros são listados a seguir:

- `part:` indica que o ator está ativo na cena a partir de certo instante. Por default, todos os atores definidos participam de toda a cena.

- `unpart:` indica que o ator desaparecerá da cena, no instante especificado. Para que o ator reapareça, é preciso usar part. Veja o exemplo a seguir, onde o ator não participará da animação até o quadro 100, só aparecendo a partir do 101º.

```
ACTORS
     A0: DEFINITION("/usr/def/actor0.adef")
         Between(FRAME(0),FRAME(100)) unpart
         After(FRAME(100)) part
END
```

- `translate_actor(A_R, x, y, z):` realiza a translação do ator. O primeiro parâmetro é a condição `ABSOLUTE` ou `RELATIVE`. *Translação absoluta significa levar o ator exatamente para o ponto desejado. Translação relativa significa deslocar o ator de "um valor na(s) direção(ões) especificada(s) em relação à sua posição anterior (seu centro de gravidade)"* [Hounsell, 92a]. Os parâmetros `x`, `y` e `z` representam as coordenadas do ponto para onde o ator se deslocará (se a

translação for absoluta) ou o valor que o ator se deslocará em cada direção, a partir de seu centro de gravidade (se a translação for relativa). Exemplo:

```
ACTORS
  A0:DEFINITION("actor0.adef")
     At(2.5) translate_actor(ABSOLUTE,0,3,2)
     After(3.) translate_actor(RELATIVE,0,2,1)
                                          TOTAL
END
```

- `translate_actor_x(A_R, n)`, `translate_actor_y(A_R, n)`, `translate_actor_z(A_R, n)`: são casos particulares da função anterior, em que o deslocamento se faz de `n` unidades na direção especificada. Por exemplo, `translate_actor_x(ABSOLUTE,3.)` é a mesma coisa que `translate_actor(ABSOLUTE, 3.,0.,0.)`.

- `rotate_actor(A_R, x, y, z, n)`: realiza a rotação do ator. O primeiro parâmetro é a condição `ABSOLUTE` ou `RELATIVE`. *Rotação absoluta "rotaciona o ator em relação ao eixo especificado em torno da origem, modificando também o posicionamento do ator no espaço". Rotação relativa "rotaciona o ator em relação ao eixo especificado em torno do seu centro de gravidade, só mudando seu direcionamento no espaço, não a sua posição"* [Hounsell, 92a]. A rotação positiva é determinada pela "regra da mão direita". Os parâmetros `x`, `y` e `z` determinam o eixo de rotação e `n` é o valor em graus da rotação. Exemplo:

```
ACTORS
 A0:DEFINITION("actor0.adef")
    After(3) rotate_actor(ABSOLUTE,1,0,2.5,60.)
                                       EACH_FRAME
END
```

- `rotate_actor_x(A_R, n)`, `rotate_actor_y(A_R, n)`, `rotate_actor_z(A_R, n)`: casos particulares da função anterior, em que o eixo da rotação é um dos eixos de coordenadas. Assim, por exemplo: `rotate_actor_z(RELATIVE, 30.)` é o mesmo que `rotate_actor(RELATIVE, 0., 0., 1., 30.)`.

- `free_rotate_actor(px, py, pz, vx, vy, vz, n)`: esta função permite novas alternativas para ponto de referência na rotação (na absoluta, era usada a origem como referência; na relativa, o centro de gravidade do ator). Os três primeiros parâmetros desta função são as coordenadas do ponto de referência desejado (em torno do qual o ator girará). Depois, vêm as coordenadas do vetor que determina o eixo de rotação. O último parâmetro é o valor em graus que o ator girará.

- `scale_actor(A_R, x, y, z)`: realiza a mudança de escala, que significa o aumento ou diminuição do tamanho do objeto em uma ou mais direções. O primeiro parâmetro é a condição `ABSOLUTE` ou `RELATIVE`.

Mudança de escala absoluta *"aumenta ou diminui o tamanho do objeto em uma ou três direções em relação à origem, causando a movimentação do ator, além da mudança de escala"*. Mudança de escala relativa *"aumenta ou diminui o tamanho do objeto em uma ou três direções em relação ao centro de gravidade do próprio ator, alterando somente seu tamanho"* [Hounsell, 92a]. Os parâmetros `x`, `y` e `z` são coordenadas que representam o quanto o ator vai crescer ou diminuir em cada direção. Estas coordenadas devem ser dadas em valores percentuais. Valores negativos nas coordenadas significa diminuição de tamanho. No exemplo seguinte, o ator diminuirá 50% na direção `x` e aumentará 60% na direção `z`:

```
ACTORS
   A0:DEFINITION("actor.adef")
      Between(FRAME(30), FRAME(85))
       scale_actor(RELATIVE,-50.,0.,60.) TOTAL
END
```

- `scale_actor_x(A_R, n)`, `scale_actor_y(A_R, n)`, `scale_actor_z(A_R, n)`: casos particulares da função anterior, em que a mudança de escala só se dá numa direção. Por exemplo, `scale_actor_y (RELATIVE, 35.)` é o mesmo que `scale_actor(RELATIVE, 0., 35., 0.)`.

- `growth(A_R, n)`, `shrink(A_R, n)`: funções que permitem um crescimento/encolhimento igual nas três dimensões. O parâmetro `n` é o valor, em porcentagem, deste crescimento ou encolhimento. Por exemplo, `growth(ABSOLUTE,65.)` é o mesmo que `scale_actor (ABSOLUTE, 65., 65., 65.)`.

- `free_scale_actor(px, py, pz, vx, vy, vz)`: permite novas alternativas para ponto de referência na mudança de escala. Os três primeiros parâmetros são as coordenadas do ponto de referência desejado. Os três parâmetros seguintes são as coordenadas que determinam o valor percentual da mudança de escala em cada direção.

- `repaint(texture_type, color)`: permite que o ator tenha sua textura alterada para a textura definida pelo arquivo cujo nome é o segundo parâmetro da função (`color`). O parâmetro `texture_type` deve ser coerente com o arquivo especificado e pode assumir os mesmos valores que `SHADE_TYPE` (ou seja, `BASIC`, `PHONGS`, `STRAUSS`, `MARBLE`, `GRANITE` ou `WOOD`). No exemplo a seguir, o ator terá sua textura alterada no quadro 27. A nova textura será dada pelo arquivo `new_texture.cor`, que deve, necessariamente definir uma textura do tipo `GRANITE`.

```
ACTORS
     A0: DEFINITION("actor3.def")
         At(FRAME(27))
            repaint(GRANITE,"new_texture.cor")
```

```
END
```

Variáveis globais, declaradas no módulo GENERAL também podem ser usadas como parâmetros para os movimentos.

V.3 - MÓDULO "GROUPS"

Neste módulo, os atores são agrupados, de modo que os movimentos possam ser realizados não só com os atores individualmente, mas também com grupos deles (podem ser criadas hierarquias de atores).

Num grupo de atores, devem ser especificados os atores que o compõem (devidamente definidos no módulo ACTORS) e o centro de gravidade do grupo (para que os movimentos do mesmo possam ser realizados relativamente a este ponto).

A definição dos movimentos para grupos de atores é idêntica à dos atores individuais.

Este módulo é muito parecido com o módulo ACTORS, se diferenciando dele apenas porque:

1. Ao invés de usar a função DEFINITION, com o nome do arquivo com as características do ator, o grupo é definido com a função COMPONENTS, que tem como parâmetros os nomes dos atores que compõem o grupo.
2. Após a definição do grupo, é preciso definir a variável GC, que indica o centro de gravidade do grupo (pode ser um ponto qualquer ou o nome de um ator, indicando que o centro de gravidade dele será o centro do grupo).

O fragmento de roteiro a seguir, mostra os módulos ACTORS e GROUPS:

```
ACTORS
     A0: DEFINITION("def/actor0.adef")
     A1: DEFINITION("def/actor1.adef")
     A2: DEFINITION("def/actor2.adef")
END

GROUPS
   G0: COMPONENTS(A0,A1,A2)
       GC = 0., 0., 0.
       After(2.) rotate_actor_x(RELATIVE,9.)
                                          TOTAL
   G1: COMPONENTS(A0,A2)
       GC = A2
       At(6.) translate_actor_y(RELATIVE,3.)
END
```

No exemplo anterior, são definidos três atores e dois grupos. O primeiro grupo é composto pelos três atores e o segundo por A0 e A2. O grupo G0 tem seu centro de gravidade na origem, e G1, no centro de gravidade do ator A2. O grupo G0 (portanto, os três atores) vai ter girado 9^o até o final da animação (com o

movimento iniciando depois de t = 2 s). Os atores do grupo `G1` vão se mover 3 unidades na direção y, no instante t = 6s.

V.4 - MÓDULO "CAMERA"

Neste módulo são especificados os parâmetros de câmera, como vistos na Figura V.2, onde:

- **obs** é a posição do observador.
- **coi** - *center of interest* - é o centro da atenção do observador (um ponto da cena para o qual ele está olhando).
- **vup** - *view up vector* - direção que identifica o que é "para cima".
- **d** - *focal distance* - é a distância entre o observador e o plano de visualização.
- **aperture** - ângulo (θ) de abertura da pirâmide de visualização.

Os parâmetros acima permitem o cálculo de outros parâmetros importantes:

- **vpn** - identifica a direção de visualização, dado por (vpn = coi - obs). Este vetor será perpendicular ao *view-plane* (plano de visualização, onde os objetos são projetados).
- **window** - "é uma porção retangular do plano de visualização que delimita a imagem de interesse e a pirâmide de visualização da mesma" [Hounsell, 92a]. Ela é determinada pelo ângulo de abertura e pela distância focal.

Assim como no caso de atores, a câmera pode ter um arquivo de definições, onde seus parâmetros são definidos. O exemplo a seguir mostra um arquivo de definição de câmera:

```
OBS = 8., 2., 0.
COI = 0., 0.5, 0.
VUP = 0., 0., 1.
D = 1.
APERTURE = 32., 32.
```

O arquivo anterior define o observador no ponto `(8, 2, 0)`, olhando para o ponto `(0, 0.5, 0)`, sendo `z` o eixo apontando para cima (vup). Além disso, a distância focal é `1` e o ângulo de abertura da pirâmide de visualização é `32`° nas duas direções (este ângulo, juntamente com a distância `d`, define a `WINDOW`, que pode ser dada neste arquivo, alternativamente).

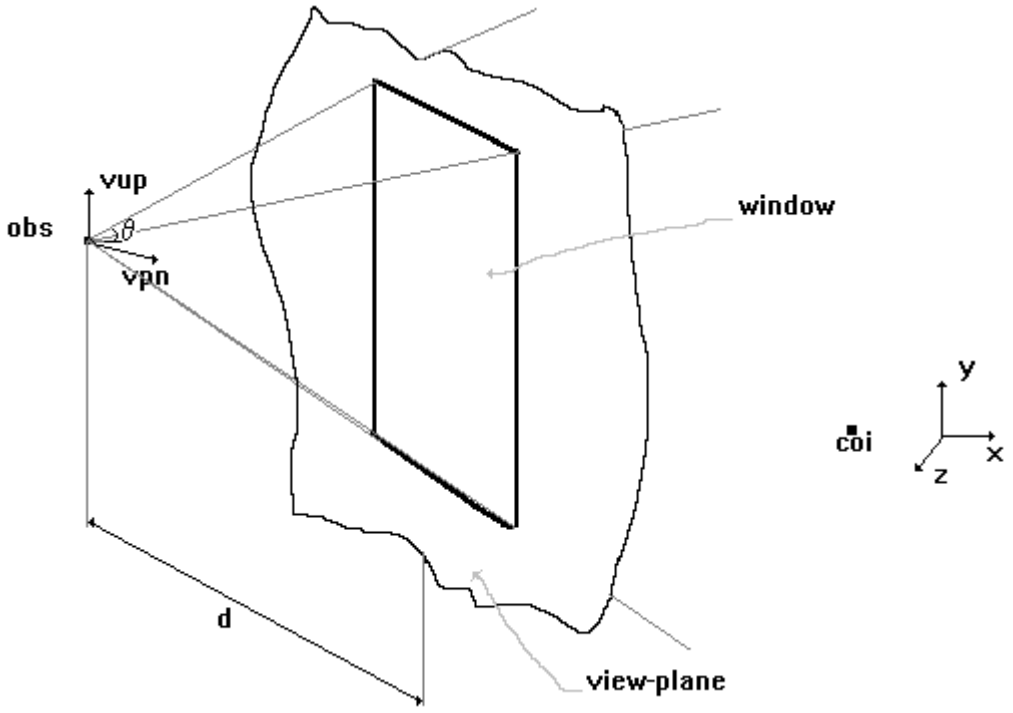

Figura V.2: Modelo de câmera.

Assim como no caso dos atores, o arquivo de definição de câmera virá referenciado no módulo `CAMERA`, através da função `DEFINITION`:

```
CAMERA
     DEFINITION("def/camera.cdef")
END
```

Com relação aos movimentos de câmera, eles seguem a mesma estruturação dos movimentos de atores ou grupos de atores:

delimitador de intervalo(...) movimento(...) `EACH_FRAME` (ou `TOTAL`)

A seguinte convenção é usada para os movimentos de câmera: movimentos começados com `go_` lidam com o observador, os que começam com `aim_` movem o centro de interesse e os que começam com `set_` redefinem os parâmetros de câmera. A lista de movimentos de câmera é dada a seguir:

- `go_forward(n)`, `go_backward(n)`: aproxima e afasta o observador do centro de interesse, respectivamente. O parâmetro `n` fornece o valor, em unidades, deste movimento. Exemplo:

```
CAMERA
   DEFINITION("camera1.cdef")
   After(FRAME(30)) go_forward(1.) EACH_FRAME
END
```

Uma maneira de se considerar o movimento do observador em volta do centro de interesse é considerar o último como o centro de um cubo, no qual o

observador anda paralelo aos seus lados. Para representar as direções nas quais ele se movimenta, são definidas as funções `go_up`, `go_down`, `go_right` e `go_left`, como mostra a Figura V.3.

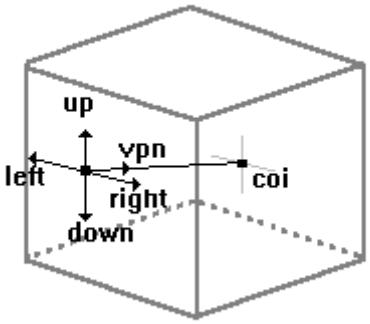

Figura V.3: Coi considerado como centro de um cubo.

O plano que contém as direções *up*, *down*, *left* e *right* é o plano que contém o vetor vup, ou seja, o plano perpendicular a vpn. A direção *up* é determinada por vup, e as outras, por consequência. Este plano não depende dos eixos cartesianos, mas do sistema de coordenadas de visualização.

- `go_up(n)`, `go_down(n)`, `go_left(n)`, `go_right(n)`: deslocam o observador nas direções acima descritas. O valor deste deslocamento é dado pelo parâmetro `n`.

Existe uma outra maneira de considerar o movimento do observador em torno do centro de interesse. Esta maneira consiste em considerar o coi como o centro de uma esfera sobre a qual desloca-se o observador nas direções cardeais. Este método garante que o movimento mantém o raio da esfera constante. As funções são: `go_north`, `go_south`, `go_east` e `go_west`, como indica a Figura V.4.

Por convenção, o Norte é indicado pelo vup e as outras direções são avaliadas considerando o observador olhando para o coi, tendo a sua direita o Leste.

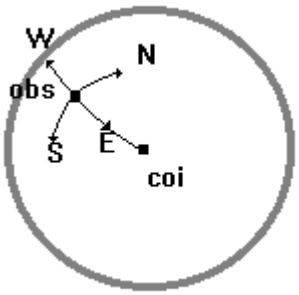

Figura V.4: Coi como centro de uma esfera.

- `go_north(n)`, `go_south(n)`, `go_east(n)`, `go_west(n)`: deslocam o observador nas direções acima descritas. O valor em graus deste deslocamento é dado pelo parâmetro `n`. Devem ser usadas para ângulos "pequenos" (entre -90^o e $+90^o$).

- `go_flying(x, y, z)`: permite o deslocamento do observador para uma posição arbitrária, mas garantindo que a direção de visualização seja recalculada, para se ajustar à direção deste movimento. Os parâmetros da função são as coordenadas da posição para a qual o observador se deslocará.

- `go_crow(x, y, z)`: "arrasta" o observador para a posição especificada, mantendo vpn constante, isto é, "arrastando" também o coi. Os valores dos deslocamentos (translação) do observador e do coi são dados pelos parâmetros da função.

- `aim_in(n)`, `aim_out(n)`: aproxima e afasta o coi do observador em n unidades, respectivamente.

- `aim_up(n)`, `aim_down(n)`, `aim_right(n)`, `aim_left(n)`: deslocam o centro de interesse em `n` unidades, nas respectivas direções.

- `aim_north(n)`, `aim_south(n)`, `aim_east(n)`, `aim_west(n)`: deslocam o centro de interesse em `n` graus, nas respectivas direções.

- `aim_actor(nome)`: especifica um ator, cujo centro de gravidade será o centro de interesse. O parâmetro é o nome deste ator.

- `aim_scene`: determina que o centro de interesse vai ser o "centro de gravidade geral" da cena, que é calculado como sendo a média dos centros de gravidade dos atores.

- `set_obs(x, y, z)`: muda a posição do observador. Esta função ainda verifica se o observador coincide com o centro de interesse do objeto (coi); se isto acontecer, aparece uma mensagem de erro.

- `set_coi(x, y, z)`: altera a posição do centro de interesse. Também verifica se o centro de interesse coincide com o observador

- `set_vup(x, y, z)`: altera a posição de vup. Os três parâmetros são as coordenadas do novo vetor vup. A função retornará mensagens de erro se os três parâmetros forem nulos ou se vup for colinear a vpn.

- `set_d(n)`: altera a distância focal. O valor do parâmetro não pode ser zero.

- `set_viewport(xcorner, xlength, ycorner, ylength)`: redefine a *viewport* (ou seja, as dimensões da imagem gerada). Os parâmetros são inteiros, dados em coordenadas de tela do dispositivo (DC - device coordinates).

- `set_window(u_min, u_max, v_min, v_max)`: altera a *window*. A dimensão destes parâmetros deve ser compatível com o valor da distância focal, para garantir um cone de projeção perspectiva coerente.

Além disso `u_min` e `v_min` devem ser iguais a `-u_max` e `-v_max`, respectivamente.

- `set_aperture (alfa_u, alfa_v)`: redefine os ângulos de abertura da pirâmide de visualização. Assim, se pode definir a janela (*window*), sem a preocupação com o valor da distância focal.

Existem ainda funções com nomes dos movimentos de câmera usados em cinema. Estas funções são, na maioria, idênticas a funções já apresentadas e têm como objetivo uma abstração no sentido de torná-las mais "próximas" do animador.

- `zoom_in(n)`, `zoom_out(n)`: efetuam aumento e diminuição dos detalhes da imagem, respectivamente. O tamanho da janela é modificado na porcentagem desejada (parâmetro `n`).

- `spin_clock(n)`, `spin_Cclock(n)`: "rodam" a direção de vup de `n` graus sobre o eixo vpn. A rotação é feita no sentido horário e anti-horário, respectivamente, visto pelo observador que olha na direção vpn.

- `tilt_up(n)`, `tilt_down(n)`: o mesmo que `aim_north(n)` e `aim_south(n)`, respectivamente .

- `pan_right(n)`, `pan_left(n)`: o mesmo que `aim_east(n)` e `aim_west(n)`, respectivamente.

- `traveling(x, y, z)`: o mesmo que `go_crow(x, y, z)`.

- `see_track(trilha)`: permite a visualização da trilha dada como parâmetro. É necessário que a trilha seja definida no módulo `TRACKS`.

V.5 - MÓDULO "LIGHTS"

Trata da iluminação das cenas. Neste módulo são definidas as fontes de luz do ambiente e a cor do fundo, com possibilidade de *fade-in* e *fade-out*, em intervalos de tempo especificados.

Há três tipos de fontes de luz possíveis:

- `POINT` (lâmpada) - é uma fonte pontual que irradia energia luminosa em todas as direções. É modelada pela sua cor e posição no espaço.
- `SUN` (sol) - é uma fonte extensa e que produz raios de energia luminosa paralelos entre si. É modelada pela cor e pela direção dos raios luminosos paralelos.
- `SPOT` - é uma fonte pontual direcional (isto é, tem uma direção principal na qual ocorre a máxima concentração de energia luminosa; fora desta direção, ocorre uma atenuação desta energia). O *spot* é modelado através de sua cor, posição, direção de atuação e ângulo de abertura. Ainda é permitido escolher entre o *spot* sem atenuação da energia luminosa, ou seja, com uma terminação abrupta (`SHARP`), ou o *spot* com atenuação da luz (`SOFT`), que é mais realista.

Assim como os atores e a câmera, a iluminação pode ser definida num arquivo externo ao roteiro. Neste arquivo deve estar definida a cor de fundo, através de suas componentes rgb, da seguinte maneira:

```
BACK = 65000., 55000., 10000.
```

Pela convenção do ProSIm, os valores de cada componente das cores variam de 0 a 65535.

Além disso, o arquivo de definições deve listar as fontes de luz da animação, com suas características. O exemplo a seguir mostra um arquivo de definição de iluminação, com três fontes de luz: uma lâmpada (`POINT`), um sol (`SUN`) e um *spot*. Cada tipo de fonte exige a definição de parâmetros diferentes:

```
BACK = 65000., 65000., 65500.

L0:  TYPE = POINT
     POSITION = 25., 4., -1.
     COLOR = 60000., 60000., 60000.

L1:  TYPE = SUN
     DIRECTION = 0., -1., 1.
     COLOR = 0., 63000., 63000.

L2:  TYPE = SPOT
     POSITION = -3., 3., 1.5
     DIRECTION = 3., -4., -2.
     COLOR = 65535., 0., 0.
     SOLID_ANGLE = 35.
     SPOT_TYPE = SOFT
```

O arquivo de definição de iluminação é referenciado no roteiro da mesma maneira que o dos atores e o de câmera, através de DEFINITION:

```
LIGHTS
     DEFINITION("def/luzes1.ldef")
END
```

Por default, todas as fontes definidas estão acesas. Para apagá-las, deve-se usar a função TURN_OFF, que tem como parâmetro o nome da fonte a ser apagada. Para reacendê-las, deve-se usar TURN_ON. Veja o exemplo a seguir, onde a fonte L0 será apagada no início da animação e acesa apenas no quadro 150:

```
LIGHTS
     DEFINITION("def/luzes1.ldef")
     At(FRME(0)) TURN_OFF(L0)
     At(FRAME(150)) TURN_ON(L0)
END
```

Também é possível mudar a posição, a direção e/ou a cor de qualquer uma das fontes de luz, num determinado instante de tempo. Para isso, são usadas as mesmas palavras-chave do arquivo de definição, mas tendo como parâmetro o nome da fonte de luz que será modificada. O exemplo a seguir altera a cor de L0 e a posição de L1:

```
LIGHTS
   DEFINITION("luzes.ldef")
   At(1.5) COLOR(L0) = 0., 65500., 10000.
   At(4.5) POSITION(L1) = 2., 3.6, -2.
END
```

A cor de fundo também pode ser mudada de maneira semelhante, como mostrado no exemplo a seguir.

```
LIGHTS
     DEFINITION("luzes.ldef")
     At(3.6) BACK = 0., 0., 65535.
END
```

O módulo LIGHTS ainda possui funções que realizam os *fades* de iluminação (*fade-in* significa ir acendendo a luz, do preto até a sua cor e *fade-out*, o contrário). É possível realizar *fades* de qualquer uma das fontes definidas no arquivo através das funções FADE_IN e FADE_OUT, que têm como parâmetro o nome da fonte que sofrerá o *fade*. Como os *fades* ocorrem em intervalos de tempo (e não em um único instante), o delimitador de tempo usado para estas funções é sempre Between. Exemplo:

```
LIGHTS
     DEFINITION("luzes.ldef")
     Between(0.,3.) FADE_IN(L1)
     Between(20.,23.) FADE_OUT(L1)
END
```

Os *fades* não são restritos às fontes de luz; a cor de fundo também pode sofrê-los. Para isso, existem as funções `FADE_IN_BACK` e `FADE_OUT_BACK` (sem parâmetros). Também é possível fazer *fades* generalizados, englobando todas as fontes definidas e a cor de fundo, através das funções `FADE_IN_ALL` e `FADE_OUT_ALL`. Estas duas funções também não possuem parâmetros.

V.6 - MÓDULO "RENDER"

Determina os intervalos da animação que serão passados ao subsistema de *rendering*. A determinação destes intervalos é importante porque a etapa de *rendering* é a mais demorada do processo de criação de uma animação por computador e, portanto, a limitação dos quadros a serem gerados pode dividir o processo em partes menores, além de agilizar a etapa de testes, quando só é preciso visualizar determinados trechos da animação.

Este módulo se organiza como no exemplo a seguir:

```
RENDER
      FROM 2.0
      TO 10.0
END
```

No exemplo acima, o *rendering* começará no instante t = 2 e terminará no instante t = 10..

Outro fator que pode agilizar a etapa de testes, é gerar apenas um quadro em cada grupo de `n` quadros consecutivos (por exemplo, gerar um a cada cinco quadros), dentro do intervalo de *rendering* determinado. Para isso, usa-se `ONE_FR_IN`. Apenas os quadros de números múltiplos de `ONE_FR_IN` serão gerados. Exemplo:

```
RENDER
      FROM FRAME(0)
      TO FRAME(80)
      ONE_FR_IN 4
END
```

No exemplo anterior, só serão gerados os quadros 0, 4, 8, 12, ..., 76 e 80.

V.7 - MÓDULO "OUTPUT"

Neste módulo são dados o nome dos arquivos que conterão as imagens e a qualidade das imagens geradas. O sistema de animação, integrado ao SIPP [Yngvesson, 94], possibilita como saída a visualização da animação em MPEG, a partir dos quadros gerados. É também possível ter como saída apenas os quadros que comporão a animação (com o *rendering* completo ou na forma de *wireframe*).

O fragmento de roteiro a seguir ilustra o módulo `OUTPUT`:

```
OUTPUT
     OUTPUT_PATH = "/tmp/quadros"
     QUALITY = SIPPPPM
     SIPP_SHADE = FLAT_SIPP
     SIPP_SHADOW = TRUE
     SIPP_SAMPLE = 2
     VIEWPORT = 320, 200
END
```

No exemplo anterior, os quadros da animação serão gerados como `/tmp/quadros0.ppm`, `/tmp/quadros1.ppm`, e assim por diante (indicado pelo parâmetro `OUTPUT_PATH`).

O parâmetro `QUALITY` indica a qualidade do *rendering* e pode assumir os seguintes valores:

- `DEBUG`: nenhum quadro é gerado; apenas o programa é testado (*debugged*). Útil apenas para o programador mais experiente, que conhece a linguagem do TOOKIMA.
- `SIPPTGA`: as imagens serão geradas com o SIPP. O tipo de *rendering* a ser realizado por ele é determinado por parâmetros adicionais (`SIPP_SHADE`, `SIPP_SHADOW` e `SIPP_SAMPLE`), que serão vistos a seguir. A saída será no formato TGA não comprimido.
- `SIPPPPM`: utiliza o SIPP, mas a saída terá o formato PPM (ou PBM, se for usada a opção `LINE` no parâmetro `SIPP_SHADE` - ver parágrafo seguinte).
- `SIPPMPEG`: também utiliza o SIPP, e o resultado será a animação MPEG e seus quadros PPM.

O parâmetro `SIPP_SHADE` determina o tipo de *shading* da cena. Pode assumir os seguintes valores: `PHONG_SIPP` (usa o método de Phong para o *shading*), `GOUR_SIPP` (usa Gouraud *shading*), `FLAT_SIPP` (usa Flat *shading*), `LINE` (visualização em *wireframe*) ou `LINE_ENVELOPE` (também *wireframe*, mas só é desenhado o envoltório convexo tridimensional - *bounding box* - de cada ator). As opções `LINE` e `LINE_ENVELOPE` gerarão quadros PBM, se a qualidade for `SIPPPPM` ou `SIPPMPEG`; elas não são compatíveis com a qualidade `SIPPTGA`.

O parâmetro `SIPP_SHADOW` pode assumir os valores `TRUE` ou `FALSE`, indicando se é desejada ou não a visualização de sombras nas imagens. Apenas fontes de luz do tipo `SPOT` podem gerar sombras.

O parâmetro SIPP_SAMPLE é um inteiro que indica o tamanho da superamostragem (*oversampling)* que o SIPP fará para evitar *alias*. Cada pixel será gerado internamente como uma matriz de (SIPP_SAMPLE x SIPP_SAMPLE) subpixels, cuja cor média representará o pixel na imagem final.

A imagem gerada terá dimensões 320 x 200, definidas por VIEWPORT.

V.8 - MÓDULO "TRACKS"

Este módulo permite a utilização de um recurso poderoso do TOOKIMA, que é a definição de trajetórias e a geração de trilhas (*tracks*). A trajetória é definida através da interpolação de uma B-Spline cúbica, a partir de pontos de controle. Em seguida, são geradas as trilhas, sobre as quais o ator poderá se movimentar. O objetivo das trilhas é tornar os movimentos mais suaves e naturais.

Neste módulo é definido o tipo de trajetória desejada, ou seja, se é uma trajetória "livre" (cujos pontos de controle são dados num arquivo) ou se é uma trajetória "conhecida" (cujos pontos de controle são obtidos de uma função matemática conhecida). Se for uma trajetória "livre", deve ser dado o nome do arquivo onde são definidos os pontos de controle. Se for uma trajetória "conhecida", devem ser dados os pontos iniciais e finais da mesma, a partir dos quais a função escolhida determina os pontos de controle que servirão para o cálculo da B-Spline cúbica.

No roteiro, após o nome da trilha definida, deve ser dado o tipo da trilha (TYPE) no roteiro. O parâmetro TYPE assume o nome do arquivo de pontos de controle, no caso de se desejar uma trajetória "livre". No caso de trajetórias conhecidas, TYPE indica a função geradora dos pontos de controle. Pode assumir os seguintes valores: LINEAR, QUADR, CUBIC ou EXP.

Outro parâmetro que deve ser definido para cada trilha é o seu valor (VALUE), que indica o tipo de elemento que compõe a trilha. Esse valor pode ser: POINT (para as posições sucessivas dos atores, ou da câmera, por exemplo), DOUBLE (para ângulos de rotação, por exemplo), VECTOR (para eixos de rotação) ou COLOR.

Quando se tratar de trilhas baseadas em funções matemáticas (ou seja, quando o parâmetro TYPE não for nome de arquivo), é ainda necessário definir os parâmetros START e STOP (que será um ponto, um real, um vetor, ou uma cor, de acordo com o valor do parâmetro VALUE).

O exemplo a seguir ilustra a definição de trilhas:

```
TRACKS
     T0:  TYPE = "pontos_trilha.in"
          VALUE = POINT

     T1:  TYPE = QUADR
          VALUE = DOUBLE
          START = 1.5
          STOP = 30.
END
```

Neste exemplo, a trilha `T0` é uma trilha de pontos (`VALUE = POINT`), lida a partir do arquivo `pontos_trilha.in`. Este arquivo deve se iniciar com o número de pontos de controle que ele contém e dar as três coordenadas de cada um desses pontos (se fosse uma trilha de reais, por exemplo, o arquivo deveria conter os valores destes números reais). A trilha `T1`, por sua vez, é uma trilha de reais, que começa em 1.5 e vai até 30., e os valores serão interpolados segundo uma função quadrática.

Para usar os valores de uma trilha como parâmetros para movimentos de atores, câmera, etc, é necessário definir uma variável global (no módulo `GENERAL`) com a função `following_track`. Esta função deve ser definida da seguinte maneira:

- `following_track(A_R, trilha, acc, A_R, begin, end)`: o primeiro parâmetro é `ABSOLUTE` ou `RELATIVE`, e indica o modo de caminhamento pela trilha. *"Caminhar por uma trilha de maneira absoluta (relativa à origem) significa ir para um determinado ponto na curva da trajetória, pois é usada uma translação absoluta para esta tarefa. Caminhar de maneira relativa (relativa à última posição) significa perguntar ao procedimento (...) quanto variou o parâmetro do ator entre a última 'posição' e a atual. (...) Caso escolha-se `ABSOLUTE`, o ator segue rigorosamente a curva definida. (...) Caso escolha-se `RELATIVE` (...), o ator segue a curva, mas a partir da posição em que o elemento está inicializado (como se a curva tivesse sido definida a partir da posição do ator)"* [Hounsell, 92a]. O segundo parâmetro desta função é o nome da trilha que se deseja seguir. O terceiro parâmetro indica a "aceleração" com que a trilha será percorrida; ele pode ter os seguintes valores:
  - `linear`: movimento uniforme.
  - `slow_in`: movimento acelerado.
  - `slow_out`: movimento desacelerado.
  - `acc_dec`: movimento acelerado até a metade do percurso e desacelerado na segunda metade.
  - `dec_acc`: desacelerado até a metade e depois acelerado.

  O quarto parâmetro é mais uma vez a condição `ABSOLUTE` ou `RELATIVE`, indicando se *"o valor de retorno é para ser calculado a partir do começo ou se é para obter apenas a variação no último instante, respectivamente"* [Hounsell, 92a]. Quando se deseja que o ator se desloque a cada instante para o ponto definido pela trilha, deve-se usar `ABSOLUTE`. A condição `RELATIVE` leva a resultados dificilmente previsíveis. Os dois últimos parâmetros indicam o intervalo de tempo em que a trilha será percorrida. É aconselhável que o intervalo do movimento (`translate`, `rotate`, etc) que usará um parâmetro proveniente de `following_track` seja o mesmo especificado nesta função. Caso contrário, o resultado poderá ser inesperado.

O fragmento de roteiro a seguir mostra os módulos `GENERAL`, `ACTORS` e `TRACKS`:

```
GENERAL
     TOTALTIME = 10.
```

```
      FR_RATE = 10.
      Point trackp = following_track(ABSOLUTE,
                T0, slow_in, ABSOLUTE, 0., 10.)
                              ; definida de 0 a 10 s
      double trackd = folowing_track(ABSOLUTE,
                T1, linear, ABSOLUTE, 0., 5.)
                              ; definida entre 0. e 5 s
END

ACTORS
  A0: DEFINITION("def/actor0.adef")
      Between(0.,10.) translate_actor(ABOLUTE,
        trackp.x,trackp.y,trackp.z) EACH_FRAME
                              ; o mesmo intervalo de definição
                              ; de T0
  A1: DEFINITION("def/actor1.adef")
      Between(0., 5.)
              scale_actor_y(RELATIVE,*trackd)
                                        EACH_FRAME
                              ; o mesmo intervalo de definição
                              ; de T1
END

TRACKS
     T0:  TYPE = LINEAR
          VALUE = POINT
          START = 10., 0., -1.
          STOP = 1.5, 13., -6.5
     T1:  TYPE = "trilha.in"
          VALUE = DOUBLE
END
```

No exemplo anterior, são definidas duas variáveis globais. A primeira delas (`trackp`) é do tipo `Point` (tipo definido pelo ProSIm, que é uma estrutura com três valores reais, representando as coordenadas). Esta variável é usada para a translação absoluta do ator `A0` e assumirá os valores da trilha `T0` (trilha de pontos, intercalados linearmente, entre os valores de `START` e `STOP`), percorrendo-a de maneira acelerada (`slow_in`), no intervalo entre 0 e 10 segundos de animação. A segunda variável global (`trackd`) é real e é usada para a mudança de escala de `A1`. Ela assumirá os valores da trilha `T1`, que é uma trilha de reais, controlada pelos valores dados no arquivo `trilha.in`. `T1` será percorrida com movimento uniforme (parâmetro `linear` em `following_track`), na primeira metade da animação (de 0 a 5 s).

V.9 - EXEMPLO

Para completar a exposição da linguagem para a composição do roteiro de animação, será mostrado a seguir um exemplo comentado:

```
GENERAL
     TOTALTIME = 10.          ; Animação de 10 s
     FR_RATE = 15             ; Taxa de 15 quadros/s
     double x1 = read_file("valores.in")
                              ; x1 terá valores lidos em arquivo
     Point trackp = following_track(ABSOLUTE, T0,
                   slow_out, ABSOLUTE, 0., 7.5)
                              ; Seguirá trilha T0,
                              ; desaceleradamente, até o
                              ; instante 7.5 seg
     Color trackc = following_track(RELATIVE, T1,
                   linear,ABSOLUTE, 4.5, 10.)
                              ; Seguirá T1, uniformemente, a
                              ; partir do instante 4.5 seg

END

ACTORS
     A0:  DEFINITION("def/actor1.adef")
          BRP = "/tmp/new_obj.brp"
          At(0.) shrink(RELATIVE, 50.)
                              ; A0 terá seu arquivo .brp
                              ; modificado em relação ao do
                              ; seu arquivo de definição e
                              ; reduzirá seu tamanho em 50%,
                              ; no início da animação
     A1:  DEFINITION("def/actor2.adef")
          Before(7.5) translate_actor(ABSOLUTE,
                        trackp.x, trackp.y, trackp.z)
                                              EACH_FRAME
                              ; A1 se moverá para os pontos
                              ; dados pela trilha  T0
     A2:  DEFINITION("def/actor3.adef")
          After(2.) unpart
                              ; A2 não participará mais da
                              ; cena, depois de 2 segundos
END

GROUPS
     G0:  COMPONENTS(A0, A1, A2)
                              ; Grupo composto pelos 3 atores
                              ; da animação
          GC = 0., -1. ,2.
          After(3.) free_rotate_actor(1.,1.,1.,
                           0.,-1.,-1.,20.) TOTAL
     G1:  COMPONENTS(A0, A1)
```

```
          GC = A0
          At(3.6) scale_actor_z(RELATIVE, -5.)
                                ; Este grupo terá o centro de
                                ; gravidade no centro de A0 e se
                                ; reduzirá 5% na direção z, no
                                ; instante 3.6 segundos.
                                ; Atenção porque o scale
                                ; será relativo ao centro de A0,
                                ; podendo causar efeito
                                ; inesperado em A1
END

CAMERA
     DEFINITION("def/cam1.cdef")
     After(7.): go_flying(1., 1., 1.) EACH_FRAME
                                ; A partir de 7 segundos, será
                                ; executado o go_flying, a cada
                                ; quadro
     Between(1.0, 10.) aim_down(0.5)
                                          EACH_FRAME
                                ; O coi se deslocará 0.5 unidades
                                ; para baixo, a partir do frame 15
                                ; (t = 1 s)
END

LIGHTS
     DEFINITION("def/light1.ldef")
     Between(6.6, 9.) FADE_OUT_BACK
                                ; O fundo sofre fade-out
     At(0.) TURN_OFF(L1)
     At(5.) TURN_ON(L1)         ; Fonte 1, definida em
                                ; def/light1.ldef só se
                                ; acenderá em t = 5 s
     At(6.) POSITION(L0) = 5., 6., -3.6
                                ; Em 6 s, a posição da fonte 0
                                ; é alterada
     After(4.5) COLOR(L0) = trackc.r, trackc.g,
                                          trackc.b
                                ; A cor de L0 passa a mudar, de
                                ; acordo com os valores da trilha
                                ; T1 (trilha de cores)
END

OUTPUT
     OUTPUT_PATH = "/tmp/teste"
     QUALITY = SIPPTGA          ; Saída serão os quadros TGA
     SIPP_SHADE = GOUR_SIPP
     SIPP_SHADOW = FALSE
     SIPP_SAMPLE = 1
     VIEWPORT = 320, 200        ; de dimensões 320 x 200
END
```

```
TRACKS
     T0:  TYPE = CUBIC
          VALUE = POINT
          START = 10., 10., 10.
          STOP = =10., -10., 10
                                   ; T0 é trilha de pontos, com
                                   ; interpolação dada por função
                                   ; cúbica
     T1:  TYPE = "/tmp/trilha.tr"
          VALUE = COLOR
                                   ; T1 é trilha de cores, lida em
                                   ; arquivo
END
```

A descrição formal completa da linguagem de roteiros apresentada (através de uma BNF) é mostrada no Apêndice A.

A primeira fase do presente trabalho, que culminou com o desenvolvimento da linguagem de roteiros, serviu para aumentar a usabilidade do sistema, tornando-o mais modular, consistente, robusto e de mais fácil utilização. Além disso, esta linguagem criou um modelo de roteiro de animação que serviu como base para a criação da interface com o usuário, que será vista no próximo capítulo.

VI - A INTERFACE

A animação por computador é muito mais do que uma combinação de hardware e software; é uma arte. John Lasseter, em [Lasseter, 87], mostra como os princípios fundamentais da animação tradicional devem ser aplicados à animação modelada por computador. Segundo ele, muitas animações por computador saem ruins porque os novos "animadores" não conhecem tais princípios, que têm sido usados nas últimas seis décadas pelos animadores tradicionais. Complementando, [Booth, 83] alega que os cientistas da computação são capazes de criar bons sistemas de animação, mas raramente são capazes de criar boas animações, que exigem uma habilidade especial, adquirida por meio de treinamento artístico e experiência. Por esta razão, os sistemas de animação bem sucedidos foram criados levando em consideração as necessidades dos artistas.

Baseado nestas constatações, após a reestruturação do sistema e a criação da linguagem de roteiros, ainda se fez necessário mais um passo para elevar o nível de abstração na criação de uma animação, visando atender as necessidades dos artistas não programadores. Este passo foi o desenvolvimento de uma interface para a construção interativa de roteiros.

A interface desenvolvida apresenta uma janela principal, que está dividida em quatro áreas, como mostra a Figura VI.1.

Na parte superior da janela, há uma linha de botões, responsáveis pela construção do roteiro e pelo controle geral do sistema (manipulação de arquivos, parâmetros gerais, execução e finalização do programa).

Alguns botões desta linha geram uma coluna de botões, à esquerda, que funciona como um "submenu" do comando (na Figura VI.1, por exemplo, esta área está ocupada por botões do módulo ACTORS). Esta coluna de botões está dividida em três áreas: edição, criação de movimentos e cancelamento de movimentos. Na área de edição, o elemento da animação pode ser criado e alterado. Na área de movimentos, tanto a posição do elemento quanto algumas de suas características físicas podem ser alteradas (metamorfose é vista como movimento, por exemplo). Na última área, os movimentos atribuídos a um elemento da cena podem ser cancelados. O módulo ACTORS da Figura VI.1, em particular, ainda apresenta uma quarta área, que permite chamar recursos externos para a modelagem de objetos e texturas.

Ocupando a maior parte da tela, existe a área de edição de textos, onde o roteiro da animação é escrito. Esta área, além de espelhar as alterações no roteiro feitas interativamente, também pode ser usada como um editor de textos comum, pois possui todos os recursos necessários para tal (*select*, *copy*, *paste*, *find*, *replace*, etc). Dessa maneira, o roteiro pode ser manipulado tanto de modo interativo, através dos botões, caixas de diálogo, etc, como também de modo direto, editando seu texto.

A quarta área da janela principal é a área de mensagens, na parte inferior da mesma. Nesta área são mostradas, à esquerda, mensagens relativas aos comandos que estão sendo realizados (uma espécie de ajuda *online*). À direita, na área de mensagens, é mostrado o nome do roteiro editado.

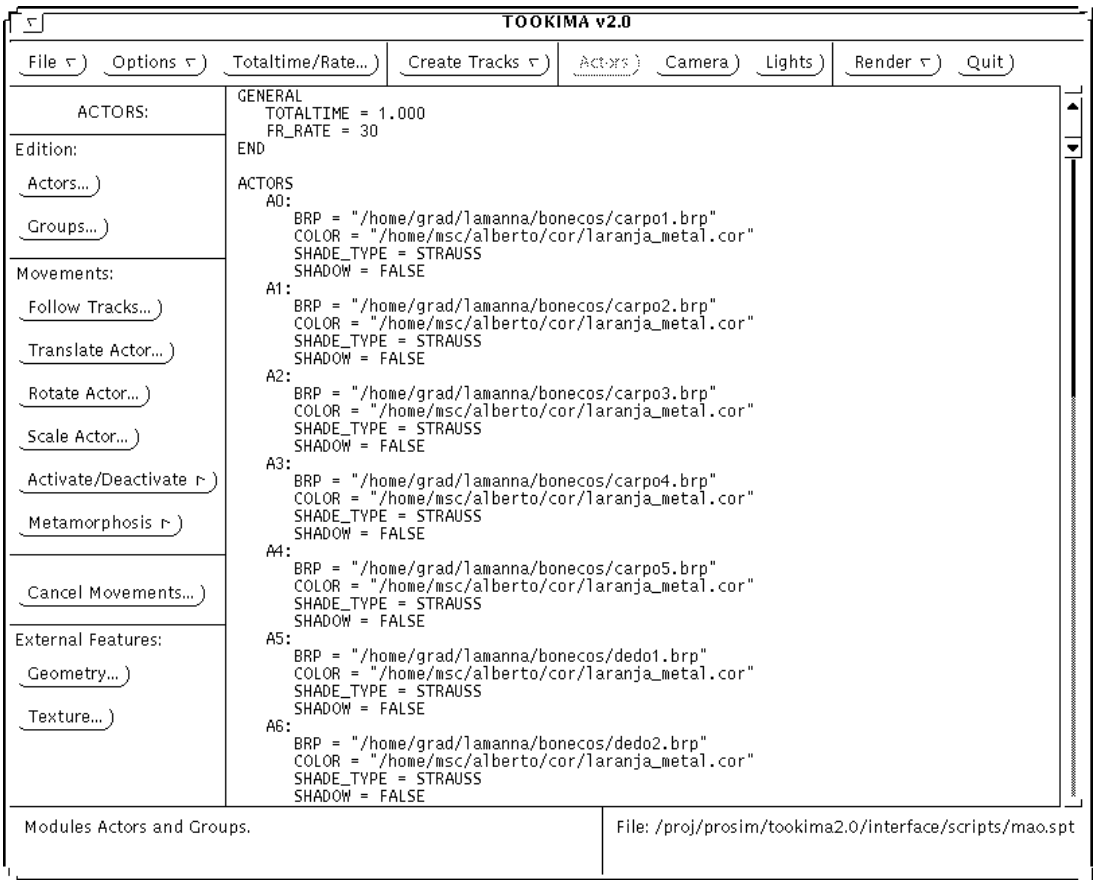

Figura VI.1: Visão geral da interface.

VI.1 - COMANDOS GERAIS

Os primeiros botões da linha de botões tratam de funções do sistema ou de parâmetros gerais do roteiro.

O botão *File* apresenta um submenu com comandos referentes à manipulação de arquivos (roteiros de animação). As opções deste submenu são: importar arquivo, exportar arquivo (salvar) e criar novo arquivo (o conteúdo da área de textos é apagado e começa-se a trabalhar em um novo roteiro).

Antes de todas as ações destrutivas (por exemplo, criar um novo roteiro sem que o anterior esteja salvo) são mostrados avisos, nos quais é pedida a confirmação da ação. Entretanto, estas situações são pouco comuns, pois a cada vez que o roteiro é lido ou reescrito pelo sistema (e isto acontece a cada vez que uma caixa de diálogo com parâmetros do roteiro é aberta ou fechada), ele é automaticamente salvo. Por esta razão, logo da primeira vez que se escreve algo (de maneira interativa) no roteiro, aparece a caixa de diálogo *Save File*, para que seja dado o nome do arquivo no qual o roteiro será armazenado.

O botão *Options* apresenta duas opções gerais a serem escolhidas pelo usuário durante a construção do roteiro. A primeira delas diz respeito à unidade de tempo utilizada. Em animação, pode-se trabalhar com as unidades de tempo em *Frames* ou em segundos. A opção *Frames* é a default. Uma vez escolhida a opção de unidade de tempo, todas as caixas de diálogo e o roteiro passam a trabalhar com ela (o roteiro construído com a interface não “mistura” as duas unidades de tempo,

embora isto seja permitido pela linguagem de roteiros). O objetivo é permitir que o usuário trabalhe com a unidade de tempo que lhe pareça mais natural, mantendo a consistência ao longo de toda a interface e do roteiro. A segunda opção geral do sistema está relacionada aos arquivos de definição (no capítulo anterior foi mostrado que atores, câmera e iluminação podem ser representados por arquivos externos .adef, .cdef e .ldef, respectivamente). Esta opção permite escolher entre a atualização dos mesmos ou a atualização apenas do roteiro, em caso de modificações realizadas interativamente. Isto funciona como a vinculação (*linkage*) ou incorporação (*embedding*) de objetos OLE do Windows [Neibauer, 92]. No primeiro caso, qualquer alteração realizada nos parâmetros dos atores, da câmera ou da iluminação será refletida nos respectivos arquivos de definição. No segundo caso, o arquivo é incorporado ao roteiro, e modificações futuras não refletirão no arquivo original, mas apenas no roteiro (esta é a opção default).

O terceiro botão da parte superior da janela (*Totaltime/Rate*) abre uma caixa de diálogo para a entrada do tempo total da animação e sua taxa de quadros por segundo. O número total de quadros da animação será o produto destes dois parâmetros. Como visto no capítulo anterior, estes são parâmetros do primeiro módulo do roteiro (GENERAL - seção V.1). Este módulo deve estar definido antes de se começar a escrever os outros módulos do roteiro. Caso contrário, uma mensagem de erro aparecerá.

VI.2 - CRIAÇÃO DE TRAJETÓRIAS

Como já visto no capítulo anterior, as trajetórias a serem seguidas por atores, câmera ou fontes de luz em uma animação podem ser definidas por arquivos de pontos de controle. O botão *Create Tracks* na linha de botões da janela principal funciona como um editor destes arquivos de pontos de controle. Há três tipos de trajetórias possíveis de serem editadas: lineares, circulares e livres.

Trajetórias lineares são retas traçadas entre um ponto inicial e um ponto final dados.

Trajetórias circulares exigem um ponto inicial (onde o círculo começará a ser desenhado), um eixo de rotação (para determinar o plano em que será calculado o círculo), um ponto de referência (origem do eixo de rotação, que determina o raio do círculo) e o número de graus girados (se este número for menor que 360, a trajetória não será uma circunferência fechada, mas um arco).

Trajetórias livres permitem a entrada de um número qualquer de pontos de controle, que serão interpolados em uma B-Spline cúbica, para a geração da trajetória.

A saída do editor de trajetórias é um arquivo de pontos de controle (.tck), que é apenas um arquivo texto iniciado com o número de pontos de controle, seguido de suas coordenadas. Este arquivo de pontos de controle pode ser associado ao movimento dos atores, da câmera ou das fontes de luz, como será visto nas seções seguintes.

VI.3 - ATORES

O botão *Actors* da linha superior da janela principal da interface ativa a coluna de botões à esquerda da mesma janela. Nesta coluna, ficam os botões relacionados à edição e movimentação dos atores que compõem a animação. Enquanto esta coluna estiver ativa, o botão *Actors* estará inativo, como na Figura VI.1. Na abertura da coluna de botões, aqueles cujas funções não podem ser realizadas ficam inativos (por exemplo, se não há nenhum ator definido na animação, botões de movimentos de atores ficam inativos).

A coluna de botões do módulo ACTORS se inicia com dois botões de edição. O primeiro é o botão *Actors*, que abre a caixa de diálogo mostrada na Figura VI.2.

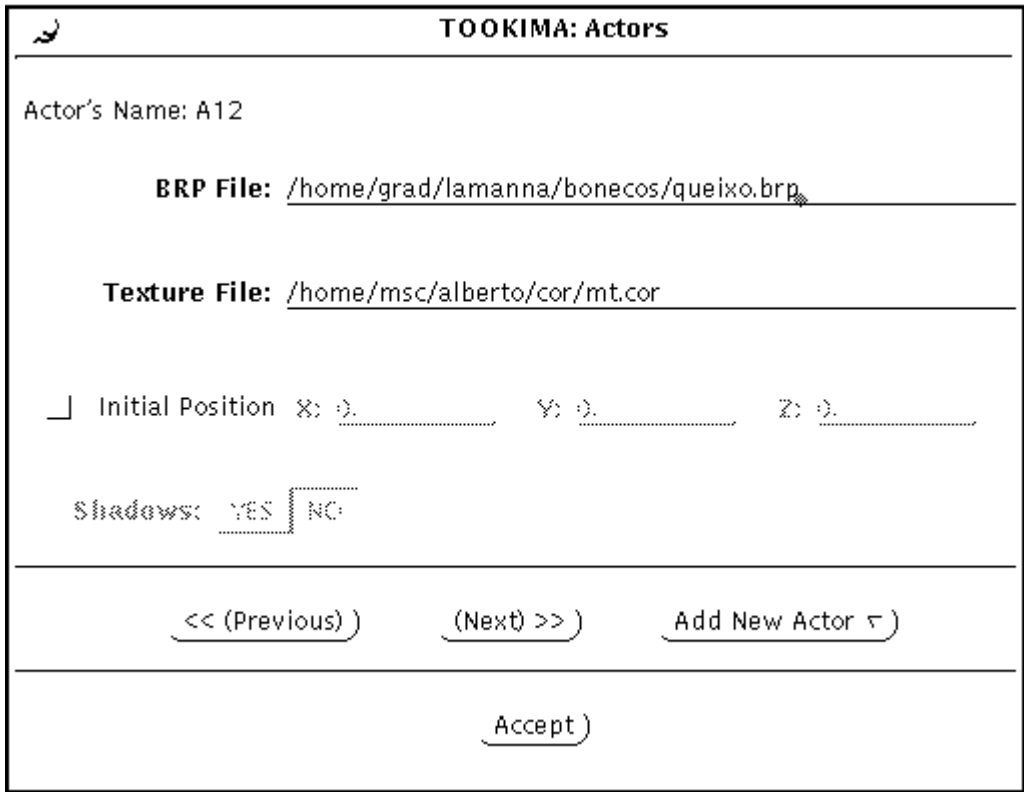

Figura VI.2: Caixa de diálogo para a associação de B-Rep e textura aos atores.

A primeira linha dessa caixa mostra o nome do ator que está sendo editado. A seguir, são dados os arquivos B-Rep (.brp) e de textura (.cor) associados ao ator. Logo depois, há a opção "posição inicial", que permite ao ator começar a animação numa posição diferente daquela em que ele foi modelado (na Figura VI.2 esta opção não está sendo utilizada, levando o ator a começar a animação na posição em que foi modelado). Nesta caixa ainda existe a opção *Shadows*, que permite definir se o ator terá ou não sombras. Entretanto, esta opção está desativada porque o SIPP (*renderer*) não permite que certos atores tenham sombras e outros não (a definição da existência ou não de sombras para todos os atores da animação é feita na caixa de *rendering*, a ser vista na seção VI.6). Os botões *Previous* e *Next* levam aos outros atores, para que edições semelhantes sejam realizadas. O botão *Add New Actor* acrescenta um novo ator à animação. Duas opções são possíveis para a criação de um novo ator: ler em um arquivo de definição (.adef), cujo nome será pedido, ou definir

suas características diretamente na caixa de diálogo mostrada. Após a edição dos atores existentes e a criação de novos, o botão *Accept* se encarrega de fechar a caixa de diálogo e atualizar o roteiro na área de edição de textos da interface.

Voltando à coluna lateral de botões do módulo ACTORS, há o botão *Groups*, logo após o *Actors*, que serve para criar grupos de atores. Este botão abre uma caixa de diálogo onde se pode selecionar, dentre os atores definidos, aqueles que comporão um grupo. Na definição do grupo de atores, também deve ser dado o centro de gravidade do mesmo, que pode ser um ponto qualquer no espaço ou o centro de gravidade de um ator. Esta caixa de diálogo cria o módulo GROUPS do roteiro (descrito na seção V.3).

Depois do botão *Groups*, entra-se na área de definição de movimentos dos atores ou grupos criados. A primeira forma de associar movimentos é através de trajetórias previamente definidas (seção VI.2). Para isso, deve-se clicar o botão *Follow Tracks*, que abre a caixa de diálogo mostrada na Figura VI.3.

Nesta caixa, a primeira coisa a ser feita é escolher o ator ou grupo que deve seguir a trajetória. Depois, deve ser dado o arquivo de pontos de controle (.tck) que define a trajetória. A "aceleração" com que a trajetória será percorrida também deve ser dada nesta caixa de diálogo (o movimento pode ser em velocidade constante, acelerado, desacelerado, acelerado até a metade e depois desacelerado, e vice-versa). Para completar a definição do movimento, deve ser dado o intervalo da animação em que a trajetória será percorrida (este valor é dado em *Frames* na Figura VI.3, indicando que se trabalha com esta unidade de tempo; isto poderia ser mudado com o botão *Options* da linha superior de botões - ver seção VI.1).

Antes de colocar a trajetória escolhida na animação é possível pré-visualizá-la de duas maneiras: *See Track* (pré-visualização "estática", onde é mostrado o primeiro quadro da animação e a trajetória é mostrada em forma de pontos no desenho) e *Preview* (pré-visualização em *wireframe* da animação, no intervalo em que a trajetória é percorrida). Finalmente, após as pré-visualizações é possível aceitar a trajetória e escrevê-la no roteiro (botão *Accept*) ou simplesmente não aceitá-la (botão *Don't Accept*), que fecha a caixa de diálogo sem alterar o roteiro.

A translação (botão *Translate Actor*) abre a caixa de diálogo mostrada na Figura VI.4. Nesta caixa deve ser escolhido o ator ou grupo que sofrerá a translação, seus parâmetros em cada direção, o intervalo em que ocorrerá , a condição absoluta ou relativa da mesma, além da condição *at each frame* ou *total*, indicando que a translação ocorrerá a cada quadro ou se completará no final do intervalo (todos estes parâmetros da translação foram explicados na seção V.2).

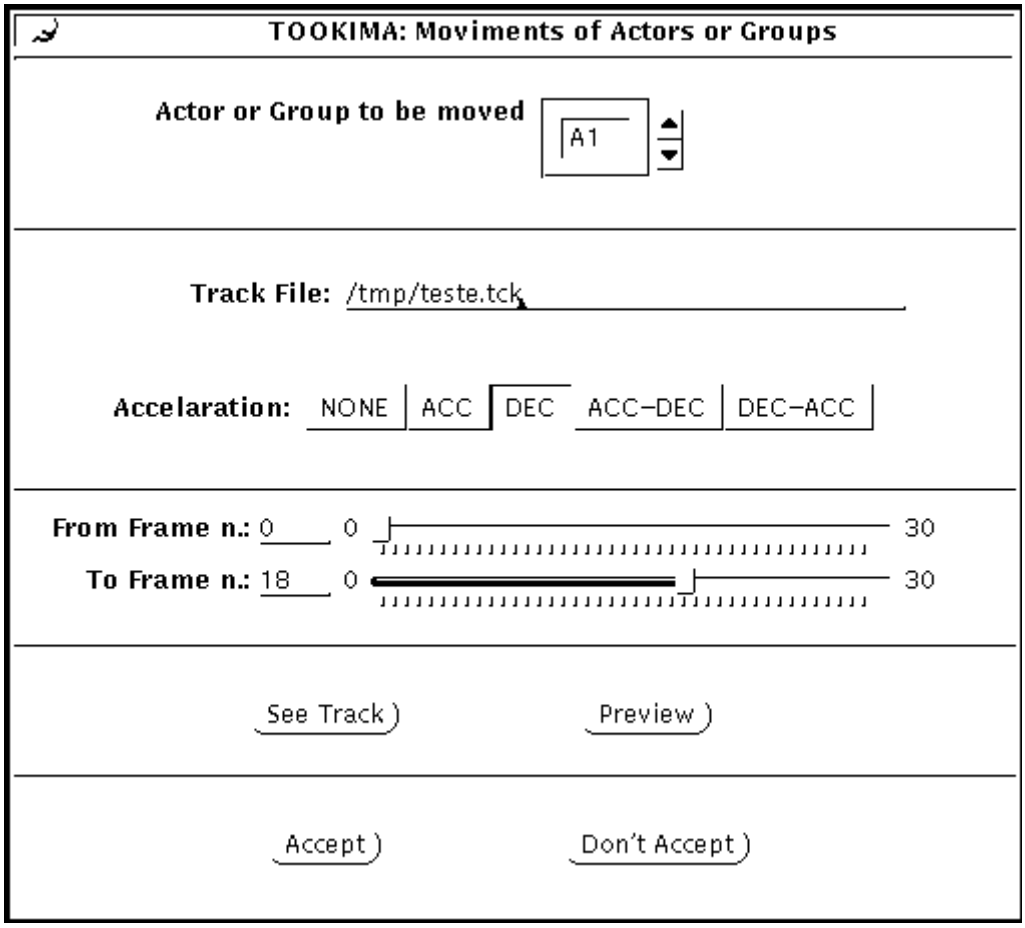

Figura VI.3: Caixa de diálogo para asociar uma trajetória ao movimento de um ator.

A rotação (*Rotate Actor*) e a mudança de escala (*Scale Actor*) abrem caixas de diálogo parecidas com a da translação (ver seção V.2).

O botão *Activate/Deactivate* permite a ativação ou desativação de determinado ator, ou seja, o ator pode aparecer ou desaparecer em determinado instante da animação. Por default, todos os atores definidos estão ativos em todo o intervalo da animação. Portanto, a ativação de um ator só tem sentido se ele tiver sido desativado antes.

O sistema permite também a visualização de metamorfoses[8] para os atores. As metamorfoses podem ser de dois tipos: de geometria (sequência numerada de arquivos .brp) ou de textura (sequência numerada de arquivos .cor). Para a definição da metamorfose, deve ser escolhido o ator que a sofrerá e também o intervalo de tempo em que ela acontecerá (lembrando que o intervalo de tempo determinará o número de arquivos .brp ou .cor que deverão existir). É válido ressaltar que a interface não oferece nenhum tipo de ferramenta para a criação, propriamente dita, da metamorfose; ela apenas associa ao ator um arquivo de uma sequência, a cada quadro da animação. Para a criação das metamorfoses devem ser usadas ferramentas externas, como por exemplo a de [Horta, 95], para a deformação de superfícies.

[8] "Metamorfose" no contexto deste trabalho é utilizada em seu sentido restrito, apenas como sendo a alteração no arquivo B-Rep da descrição geométrica de um ator da animação.

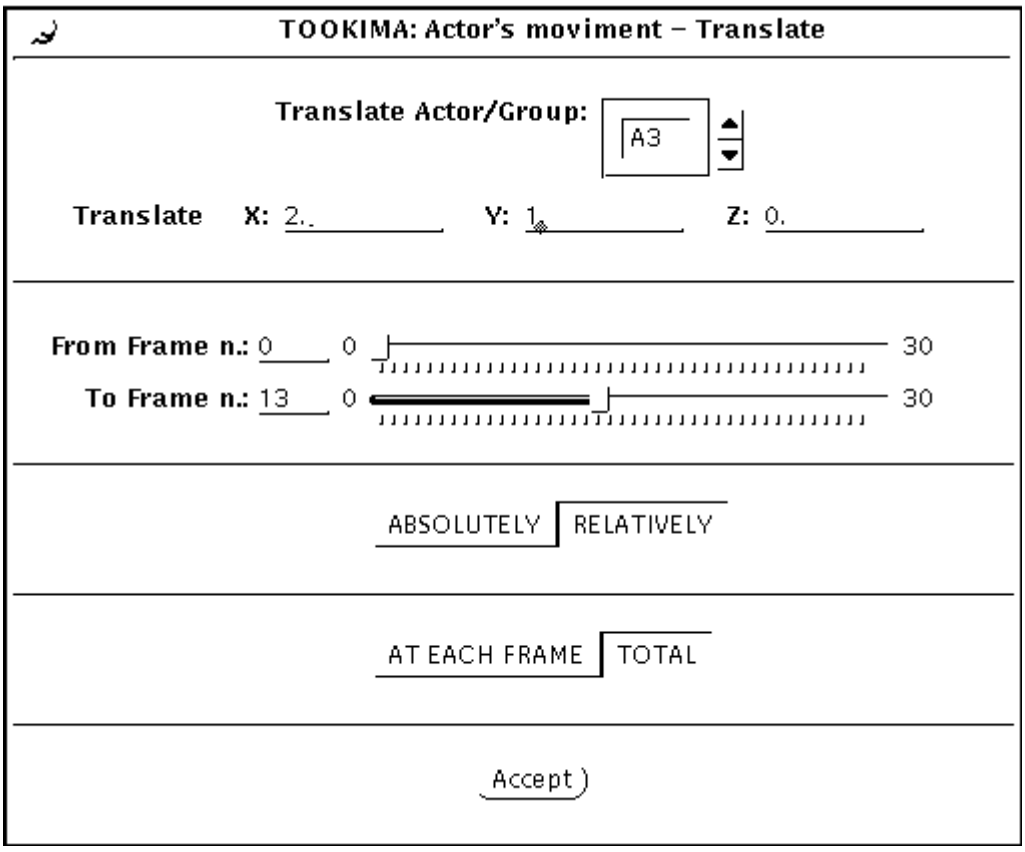

Figura VI.4: Caixa de diálogo para a translação de atores ou grupos de atores.

O próximo botão da coluna do módulo ACTORS é o *Cancel Movements*, que permite cancelar todos os movimentos de um ator ou grupo (inclusive as metamorfoses). Uma mensagem para a confirmação do comando é mostrada antes da atualização do roteiro.

A última parte da coluna de botões do módulo ACTORS permite a conexão com ferramentas auxiliares. O botão *Geometry* aciona o modelador geométrico do ProSIm, para a criação dos modelos B-Reps dos atores. O botão *Texture* ativa o editor de texturas, um executável para a criação das texturas dos atores, também desenvolvido como parte deste trabalho (ver Apêndice B). Estas ferramentas auxiliares não atuam diretamente na construção do roteiro, mas permitem a criação de arquivos que devem ser associados aos atores da animação.

VI.4 - CÂMERA

Assim como o botão *Actors*, o botão *Camera* da linha superior abre uma coluna lateral de botões, com comandos específicos para o tratamento da câmera. A Figura VI.5 mostra esta coluna.

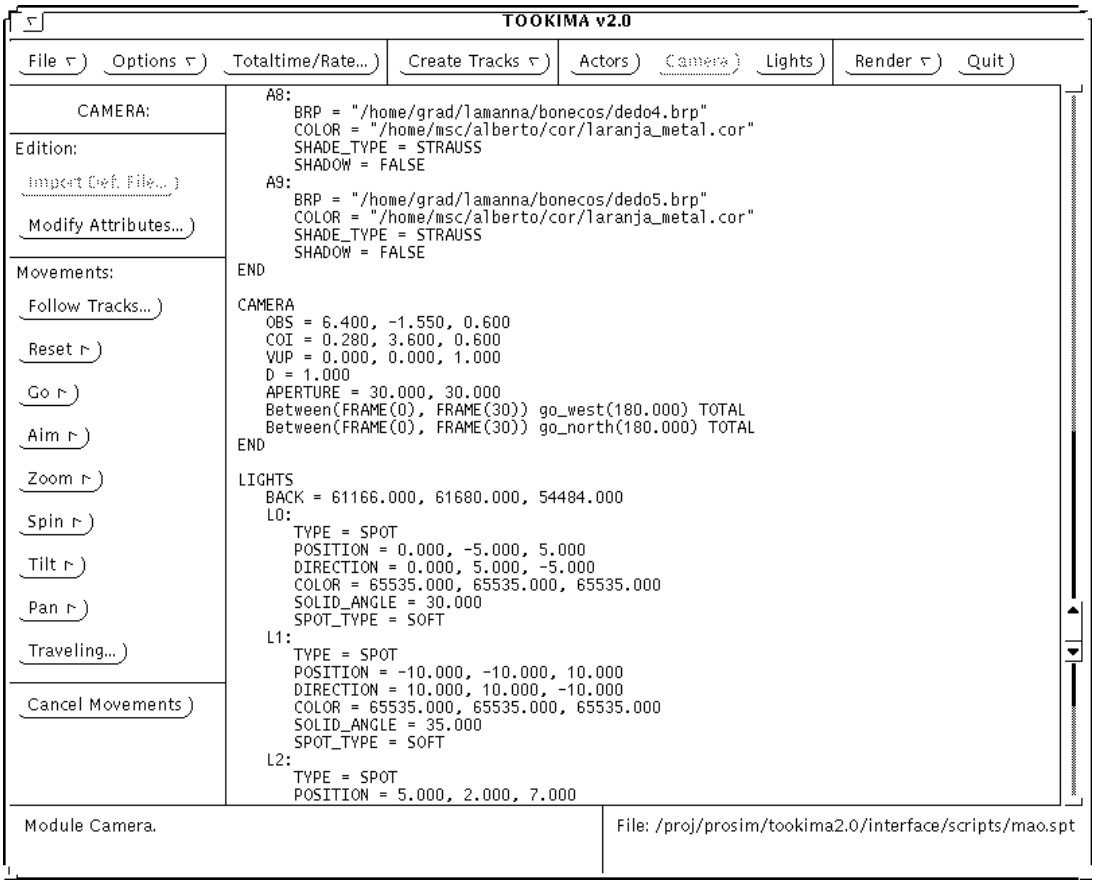

Figura VI.5: Interface com a coluna de botões do módulo CAMERA.

O primeiro botão desta coluna é o *Import Def. File*, que permite carregar um arquivo de definição de câmera (.cdef). O arquivo de definição de câmera segue o padrão visto na seção V.4 e pode ser construído diretamente no modelador do ProSIm. O botão *Import Def. File* só estará ativo se o módulo CAMERA ainda não tiver sido definido no roteiro. Se já houver uma definição de câmera no roteiro, não é possível carregar um arquivo de definição de câmera, pois isto implicaria em alterar todos os parâmetros de uma só vez.

O botão seguinte na coluna do módulo CAMERA é o botão de edição de câmera, que exibe a caixa de diálogo da Figura VI.6. Nesta caixa, os parâmetros da câmera podem ser modificados. O significado destes parâmetros foi visto na seção V.4. O botão *Accept* fecha a caixa de diálogo e atualiza o roteiro, com os novos parâmetros de câmera.

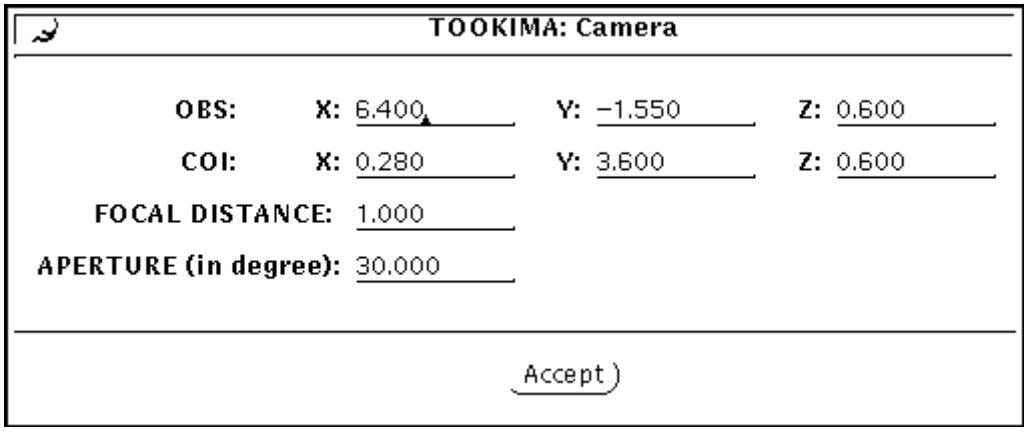

Figura VI.6: Caixa de Diálogo para a edição da câmera.

Continuando na coluna de botões do módulo CAMERA, chega-se aos movimentos. O primeiro deles, assim como para os atores, é dado por trajetórias pré-definidas. O botão *Follow Tracks* abre uma caixa de diálogo parecida com a da Figura VI.3, onde a única diferença é que agora se escolhe o observador ou o centro de interesse para seguir a trajetória, e não mais um ator ou grupo.

Os botões seguintes representam os movimentos básicos de câmera, que já foram definidos na seção V.4. Movimentos do tipo *Reset* alteram os valores de determinado parâmetro da câmera em um certo instante da animação. Movimentos do tipo *Go* alteram a posição do observador, enquanto os do tipo *Aim* alteram o centro de interesse. O *Zoom* aumenta ou diminui a visualização de detalhes da cena. *Spin* gira o plano da imagem, no sentido horário ou anti-horário. *Tilt* corresponde ao movimento de abaixar ou levantar o observador na vertical. *Pan* faz o mesmo que *Tilt*, porém na horizontal. O *Traveling*, por sua vez, corresponde ao movimento do observador sem mudar a direção de visualização, ou seja, é a translação do observador juntamente com a do centro de interesse.

Para ilustrar os movimentos básicos de câmera, a Figura VI.7 mostra a caixa de diálogo ativada quando se escolhe o movimento de *Go North* para a câmera. Nesta caixa é preciso dar o número de graus do movimento, o intervalo do mesmo e a condição *at each frame* ou *total*.

O último botão da coluna do módulo CAMERA é o *Cancel Movements*, que permite cancelar todos os movimentos definidos para a câmera (comando que exige confirmação).

VI.5 - ILUMINAÇÃO

O botão *Lights* da linha de botões na parte superior da janela também abre uma coluna lateral de botões, que pode ser vista na Figura VI.8. Esta coluna apresenta botões com funções relacionadas à manipulação da cor de fundo e das fontes de luz.

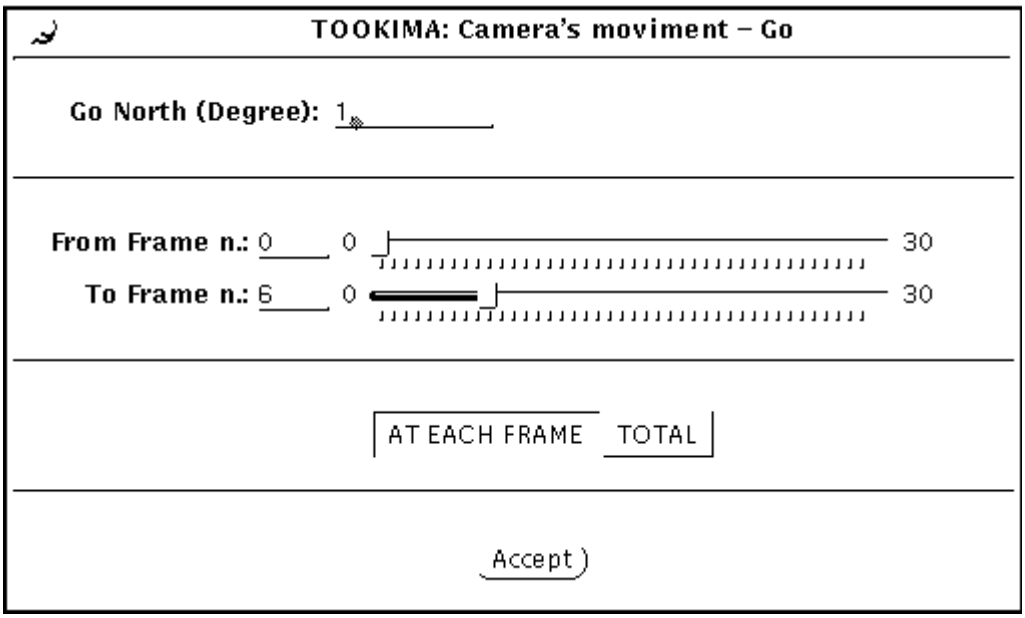

Figura VI.7: Caixa de diálogo para o movimento "Go North" da câmera.

O primeiro botão desta coluna é o *Import Def. File*. Este botão funciona de maneira semelhante ao botão com o mesmo nome, visto na seção anterior. Ele está ativo enquanto o módulo LIGHTS ainda não está definido no roteiro, e seu objetivo é carregar um arquivo de definição de iluminação, criado em um executável externo à interface[9] .

O próximo botão da coluna é o *Background Color*, usado para a definição da cor de fundo da animação. Este botão abre a caixa de diálogo vista na Figura VI.9, onde a cor é definida por meio de suas componentes rgb (o desenho em "V" à esquerda varia à medida que os valores rgb são alterados, mostrando a cor definida).

O botão seguinte, ainda na parte de edição da coluna de botões, é o *Light Sources*. Este botão abre a caixa de diálogo mostrada na Figura VI.10. Nesta caixa, a primeira linha indica a fonte que está sendo editada. Abaixo do nome da fonte, é possível escolher o tipo da fonte (*spot*, pontual ou do tipo *sun*). Em função do tipo de fonte escolhida, os parâmetros dados abaixo podem estar ou não ativos. Por exemplo, a posição da fonte é ativa para o caso de *spots* e fontes pontuais. Depois da posição, deve ser dado o alvo do *spot* (isto é, o ponto para o qual ele aponta). Se a fonte for do tipo *sun*, o alvo é substituído pela direção da mesma. Se a fonte for do tipo *spot*, os parâmetros ângulo de abertura e tipo de *spot* também devem ser dados. Abaixo, a cor da luz emitida pela fonte é definida de maneira semelhante à cor de fundo. Finalmente, a caixa de diálogo apresenta os botões *Previous* e *Next*, para levar o usuário às outras fontes de luz, além do botão *Add New Source*, para criar uma nova fonte de luz. O botão *Accept* atualiza o roteiro e fecha a caixa de diálogo.

[9] Este executável também foi desenvolvido como parte deste trabalho, mas não será descrito nesta dissertação, pois tem uma interface muito semelhante à caixa de diálogo para a edição de fontes de luz vista na Figura VI.10.

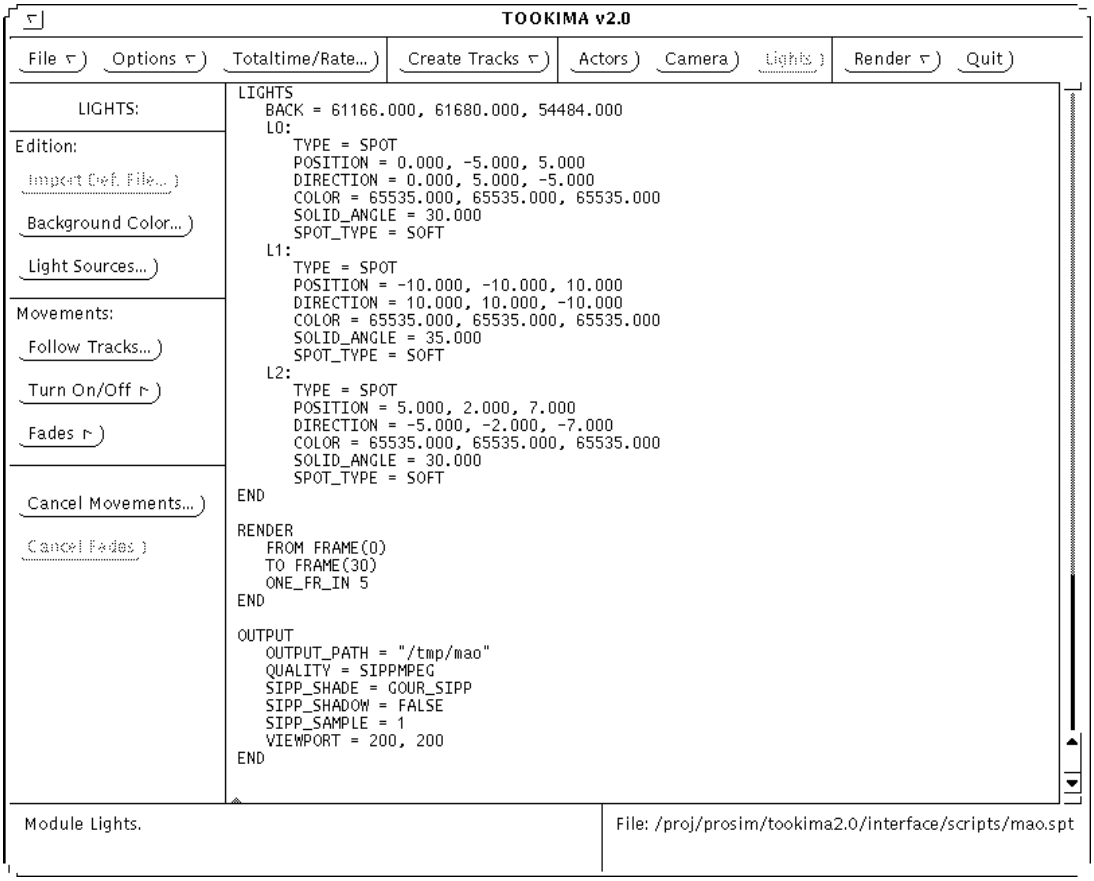

Figura VI.8: Interface com a coluna de botões do módulo LIGHTS.

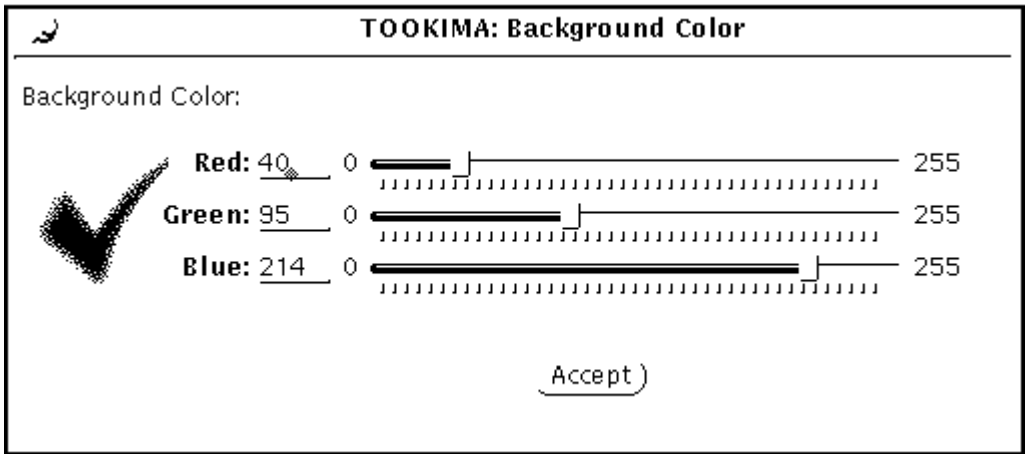

Figura VI.9: Caixa de diálogo para a edição da cor de fundo.

Na parte de movimentos da coluna de botões, aparece o botão *Follow Tracks*, que abre uma caixa de diálogo parecida com a da Figura VI.3, com a diferença de que aqui deve-se escolher a fonte de luz que seguirá a trajetória, e se a trajetória alterará a posição ou a cor da fonte. Seguir uma "trajetória" de cores significa que a cor da luz da fonte terá os valores rgb dados pelas coordenadas dos pontos da trajetória (portanto, os pontos desta trajetória devem ter valores entre 0 e 65535, os limites para as componentes rgb).

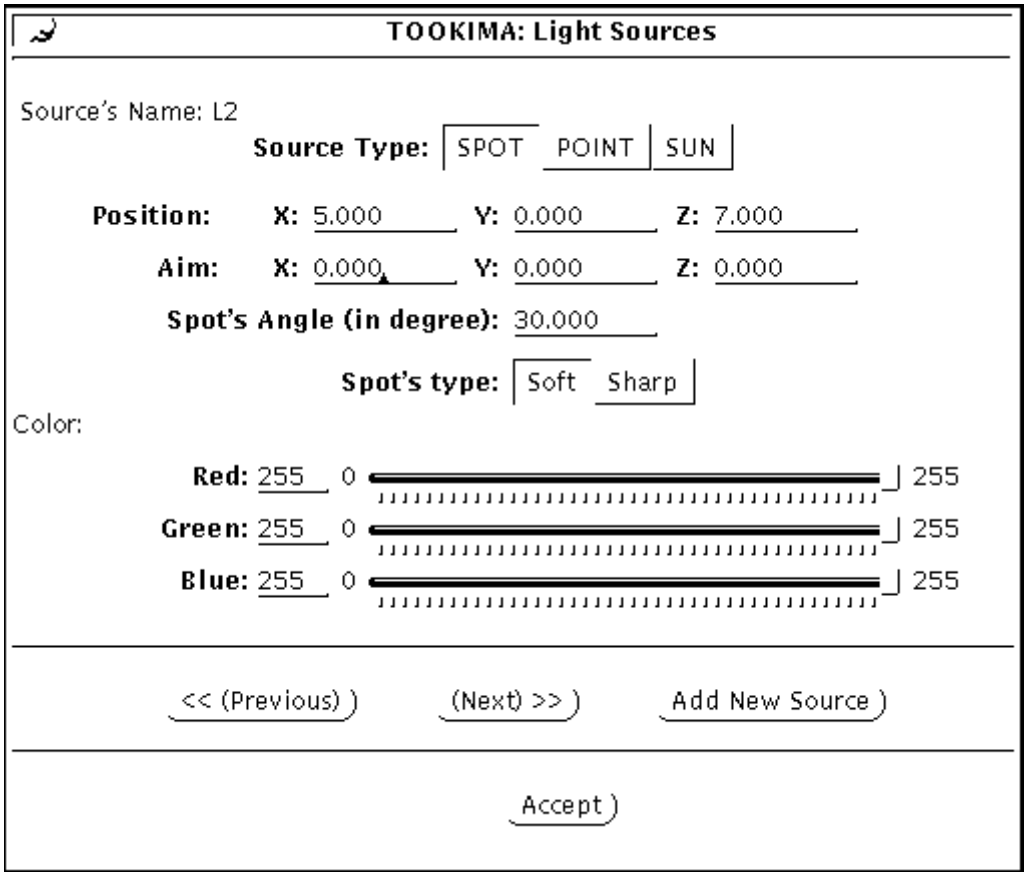

Figura VI.10: Caixa de diálogo para a edição de fontes de luz

Depois das trajetórias, ainda há duas alterações que uma fonte de luz pode sofrer: liga/desliga e *fade* (estas alterações foram vistas com mais detalhes na seção V.5). O botão *Turn On/Off* permite ligar ou desligar uma fonte em determinado instante. É válido lembrar que, por default, todas as fontes definidas estão ligadas. Portanto, ligar uma fonte só tem sentido se ela tiver sido desligada anteriormente. O botão *Fades* permite a criação de *fades-in* (acender uma fonte gradativamente, do preto até a sua cor) e de *fades-out* (apagar uma fonte gradativamente).

Os dois últimos botões da coluna servem para cancelar os movimentos das fontes. *Cancel Movements* elimina as trajetórias e as ações de ligar ou desligar uma fonte. *Cancel Fades* elimina os *fades* da animação. Ambos os botões exigem a confirmação dos comandos.

.

VI.6 - *RENDERING*

Depois dos botões da parte superior da janela principal que geram a coluna lateral (botões *Actors*, *Camera* e *Lights*), aparece o botão *Render*. Este botão apresenta um menu onde se deve escolher o *renderer* utilizado e, em função do *renderer* escolhido, uma caixa de diálogo é aberta. Por enquanto, o sistema funciona apenas com o SIPP, não existindo, na prática, a escolha do *renderer*.

A caixa de diálogo aberta ao clicar o botão *Render* é vista na Figura VI.11.

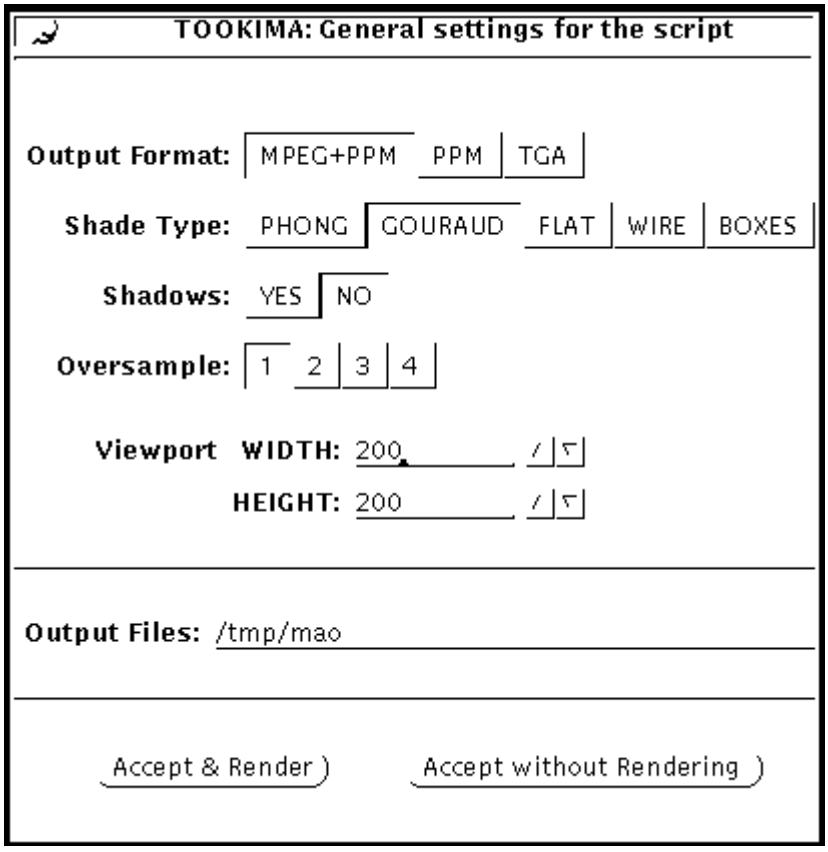

Figura VI.11: Caixa de diálogo para os parâmetros de rendering.

Nesta caixa, estão os parâmetros de *rendering* (módulo OUTPUT do roteiro - seção V.7). A primeira opção desta caixa indica se a saída será uma animação no formato MPEG, juntamente com os quadros PPM da mesma, ou apenas os quadros no formato PPM ou TGA. A opção *Shade Type* indica o tipo de sombreamento dos quadros (Phong, Gouraud, *flat shading*, quadros em *wireframe*, ou *wireframe* apenas com os envoltórios convexos - *bounding boxes* - dos objetos). A opção *Shadows* indica se haverá ou não a representação das sombras dos objetos e a opção *Oversample* indica a dimensão da superamostragem. *Viewport* indica as dimensões das imagens a serem geradas. Nesta caixa também deve ser dado o *path* e o nome dos arquivos resultantes (quadros da animação).

Depois de escolhidos os parâmetros de *rendering*, o usuário tem duas opções: gerar ou não gerar os quadros. Se ele optar por não gerá-los (*Accept without Rendering*), o roteiro será atualizado com os parâmetros de *rendering*, mas ele não será efetuado. Se ele optar pela geração dos quadros (*Accept & Render*), o roteiro também será atualizado, e uma nova caixa de diálogo aparecerá, onde devem ser dados os parâmetros do módulo RENDER (seção V.6), ou seja, o intervalo de *rendering* e em cada quantos quadros um será gerado (se será gerado um a cada dois quadros, um a cada dez, etc). O botão *Accept* desta caixa de diálogo inicia efetivamente o processo de *rendering* dos quadros da animação.

O *rendering* é realizado como um processo filho, executado em *background*. Por esta razão, o usuário pode continuar trabalhando com a interface enquanto os quadros estão sendo gerados. Enquanto o *rendering* está sendo efetuado, aparece um botão chamado *Abort Render* na área de mensagens da interface. Este botão serve para abortar o processo de *rendering*.

O último botão da linha de botões da janela principal é o *Quit*, que finaliza a execução do TOOKIMA (exigindo uma confirmação para tal).

VII - CONCLUSÕES

O presente trabalho pode ser dividido em duas etapas sequenciais: uma etapa de reestruturação do conjunto de ferramentas para a descrição de animações cinemáticas existente no ProSIm (o TOOKIMA) e uma etapa de construção de uma interface gráfica para o mesmo. A etapa de reestruturação do TOOKIMA teve como objetivo aumentar sua usabilidade e robusteza, através de testes, correção de falhas e documentação, culminando com a criação de uma nova linguagem de roteiros mais simples e modular. A segunda etapa teve como objetivo elevar o grau de abstração na criação de animações, com o desenvolvimento de uma interface interativa, visando tornar o sistema acessível a artistas não programadores.

VII.1 - RESULTADOS

O trabalho foi realizado com o compromisso de manter a consistência e a compatibilidade com outros módulos do ProSIm. A consistência foi garantida seguindo o mesmo *layout* das interfaces já existentes. Para manter a compatibilidade, foi exigido que o sistema de animação desenvolvido trabalhasse com os mesmos tipos de dados produzidos pelos outros módulos (no caso, foi exigido que os objetos da animação tivessem o formato B-Rep).

Aliado a estes compromissos, o sistema foi desenvolvido de modo a ser modular e relativamente independente da configuração particular do modelador geométrico e do *renderer*. Embora trabalhe apenas com o modelador do ProSIm e com o *renderer* SIPP, o sistema foi desenvolvido de modo a facilitar a adaptação a novos modeladores ou *renderers*.

Três contribuições básicas no contexto ProSIm podem ser vistas como consequências deste trabalho.

- Manutenção do TOOKIMA. Na verdade, esta foi uma etapa iniciada ainda como trabalho de Iniciação Científica, que serviu para conhecer a fundo o sistema já desenvolvido, testá-lo e fazer as alterações necessárias. Nesta fase o sistema também foi documentado, tanto através do desenvolvimento de um manual de utilização [Raposo, 95a] como através da revisão do código, deixando-o mais comentado.

- Criação de uma nova linguagem para a descrição de roteiros de animação. Esta linguagem não apenas simplificou os roteiros de animação, mas também incorporou facilidades que se mostraram necessárias à medida que animações iam sendo desenvolvidas com o sistema (facilidades que não existiam na linguagem do TOOKIMA, tais como a possibilidade de leitura em arquivos, etc). Foi implementado um sistema “tradutor” da nova linguagem, que durante a execução traduz o roteiro para a antiga linguagem do TOOKIMA. A linguagem de roteiros está documentada em [Raposo, 95b].

- Implementação de uma interface para a construção interativa de roteiros. Após o passo intermediário, que foi a elaboração de uma linguagem mais simples para a composição de roteiros de animação, tornou-se possível o desenvolvimento de uma interface que facilitasse a construção dos mesmos. O objetivo foi permitir o trabalho em um nível ainda mais alto de abstração durante a criação de uma animação. Assim como na transposição da linguagem do TOOKIMA para a linguagem de roteiros, ao longo do processo de desenvolvimento da interface foram incorporadas novas facilidades (tais como a possibilidade de utilização de uma sequência de B-Reps para um ator, etc).

Além destas três contribuições básicas, foram criadas algumas ferramentas auxiliares (por exemplo o editor de texturas, visto no Apêndice B) que, embora não tratem diretamente de animações, são bastante úteis durante a criação das mesmas.

O sistema apresenta, no total, cerca de 34900 linhas de código, sendo que aproximadamente 70% deste total envolve a interface com o usuário e 10%, o tradutor da linguagem de roteiros para a linguagem do TOOKIMA. O restante (cerca de 20% do total de linhas de código) faz parte do TOOKIMA propriamente dito (é o código desenvolvido por [Hounsell, 92a], um pouco modificado para se adaptar à reestruturação feita).

O TOOKIMA 2.0 apresenta uma interface gráfica, relacionada à construção de um roteiro de animação. Este tipo de implementação difere radicalmente da maioria dos sistemas de animação comercializados (Animator [Autodesk, 89] e 3D Studio [Autodesk, 90], por exemplo). Estes sistemas, normalmente apresentam interfaces exclusivamente de manipulação direta, desvinculadas de qualquer roteiro, o que em muitos casos dificulta o uso, pois o usuário fica sem ter uma orientação de como utilizar os inúmeros recursos dispersos na interface (a linguagem de interface não se relaciona com a linguagem de desenvolvimento de uma animação). No TOOKIMA 2.0, a linguagem de interface se relaciona intimamente com a linguagem de roteiros, criando um vínculo que facilita sua utilização.

O sistema de animação criado não encerra os trabalhos voltados para a criação de animações no ProSIm. Ainda há a necessidade de novos esforços relacionados à adaptação de novos paradigmas de animação, de novos *renderers* e, sobretudo, relacionados à integração dos módulos do ProSIm em um sistema único, que englobe desde a modelagem até o *rendering*.

A Figura VII.1 ilustra o sistema ProSIm ao final deste trabalho. Hoje, a modelagem de objetos e de câmera já estão integradas em uma única interface, cujas saídas são os arquivos B-Reps e de definição de câmera. Dois módulos auxiliares definem os arquivos de textura (Apêndice B) e de iluminação. Todos estes arquivos são utilizados pelo TOOKIMA para a definição de uma animação.

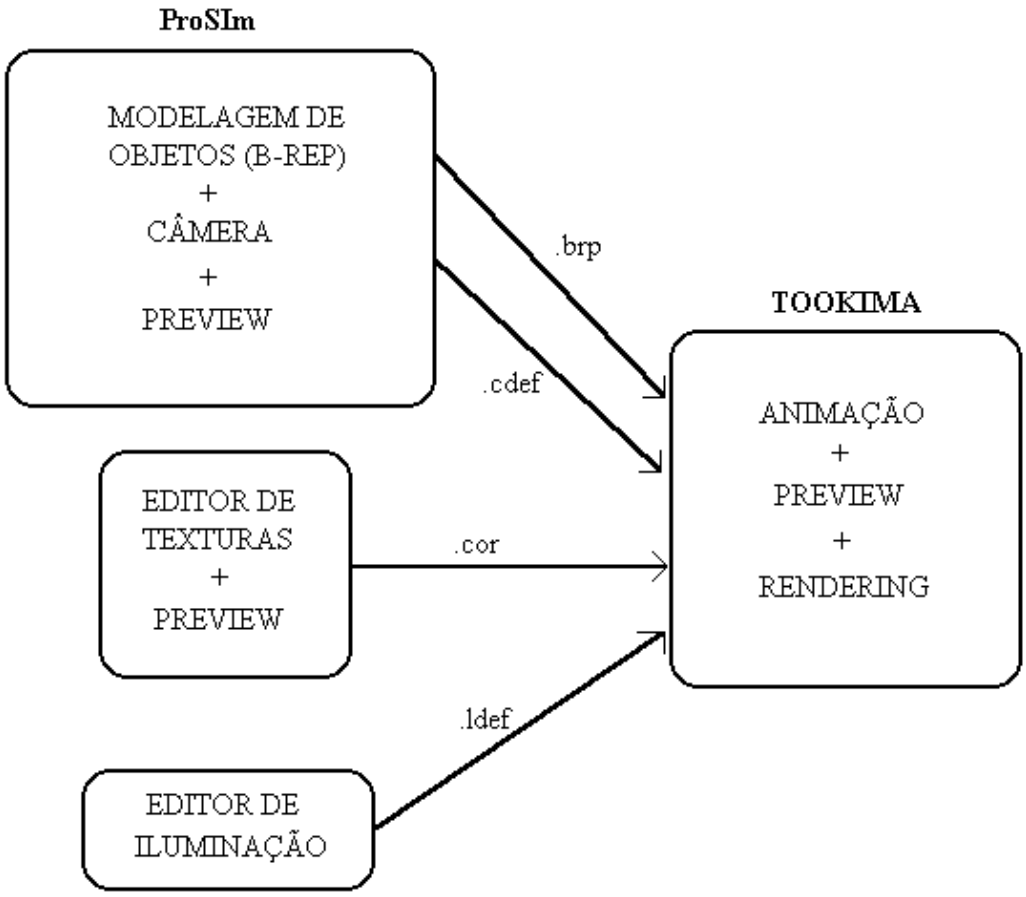

Figura VII.1: Sistema ProSIm hoje em dia.

VII.2 - PERSPECTIVAS

O objetivo do ProSIm, a longo prazo, é ter todos estes recursos integrados em uma ferramenta gráfica única. Para isso, será necessário que o modelador existente englobe a modelagem de fontes de luz e de textura. Numa etapa posterior, os módulos de modelagem e de animação migrariam para uma interface comum, que permita a criação de animações com mais recursos gráficos.

Além da integração de ferramentas, outras sugestões de trabalhos dando continuidade a este podem ser feitas.

- Desenvolvimento de ferramentas adicionais para a utilização da interface na criação de outros tipos de animação além da cinemática (criação de ferramentas para animação dinâmica, por exemplo).

- Tratamento gráfico das trajetórias, isto é, permitir que as trajetórias a serem seguidas pelos atores de uma animação sejam modeladas como curvas no modelador geométrico.

- Adaptação do sistema a novos formatos de representação de objetos modelados, além do B-Rep.

- Adaptação a novos *renderers*, em particular, o POV-Ray [Young, 94], um *ray-tracer* com o qual já foram iniciados trabalhos no LCA.

- Remodelagem das interfaces utilizando bibliotecas gráficas mais atualizadas (Motif, por exemplo).

- Adaptação do sistema para a computação distribuída/paralela, visando melhorar o desempenho do *rendering* dos quadros de uma animação.

- Extensão da linguagem de roteiros para oferecer um maior nível de abstração, através da incorporação de "comportamentos" (por exemplo, através da criação de macros).

- Extensão/alteração da linguagem de roteiros, de modo que ela possa se tornar efetivamente uma EUPL, pois ainda há uma "distância semântica" entre ela e a linguagem da interface.

- Criação de ferramentas gráficas para a construção de animações por manipulação direta, embora tentando não se distanciar da EUPL.

Apêndice A

Descrição Formal da Linguagem de Roteiros

A definição formal da sintaxe de uma linguagem de programação é importante tanto para o usuário quanto para o desenvolvedor. Para o usuário, ela serve como referência, pois é uma descrição clara da linguagem. Para o desenvolvedor, ela facilita a manutenção da linguagem.

A BNF ("Backus Normal Form") é uma notação para escrever "gramáticas" [Gear, 74] e será utilizada para descrever formalmente a sintaxe da linguagem de roteiros do ProSIm, informalmente descrita no capítulo V. A BNF consiste de sentenças que definem a maneira em que a linguagem de programação deve ser escrita. Ela é chamada de *meta-linguagem*, e usa caracteres diferentes daqueles que são usados na linguagem por ela descrita. Na BNF, os símbolos não-terminais (isto é os símbolos da meta-linguagem) são delimitados por **< >** e representam estágios intermediários no processo de descrição da linguagem.

O sinal **::=** significa: "é substituído por". Assim, a sentença: *<digito_0>* ::= 0 é lida: "o meta-símbolo *<digito_0>* é substituído por 0".

O símbolo | também é usado na BNF, indicando alternativa ("ou"). Assim, para representar um digito qualquer, tem-se: *<digito>* ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9.

Para facilitar a descrição da linguagem de roteiros do ProSIm, foram acrescentados novos símbolos à meta-linguagem. Esses símbolos se baseiam na meta-linguagem usada por [Pressman, 92] para a definição de um dicionário de dados. Os símbolos da meta-linguagem a ser usada e seus respectivos significados são descritos na tabela a seguir.

Símbolo	Significado
< >	delimitam meta-símbolo
::=	"é substituído por"
|	"ou"
[ ]	opcional
{ }^{n}	"n repetições de"
⇒	continuação de linha
/* */	delimitam comentários

A linguagem de roteiros do ProSIm é assim descrita:

/* Definições gerais */

```
<digito> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<inteiro> ::= <digito> | <inteiro> <digito>
                                /* representa qualquer número inteiro */
<v_inteiro> ::= <inteiro> | <string>
                                /* variável inteira: valor ou nome de variável (que
                                pode ser usada como parâmetro de movimento),
                                com a restrição de que a variável assuma
                                valores do tipo desejado */
<real_pos> ::= <inteiro> | <inteiro>. | .<inteiro> |
             ⇒ <inteiro>.<inteiro>
<v_real_pos> ::= <real_pos> | <string>
<real_neg> ::= -<real_pos>
<v_real_neg> ::= <real_neg> | <string>
<real> ::= <real_pos> | <real_neg>
<v_real> ::= <real> | <string>
<letra> ::= a | b | c | ... | z | A | B | C | ... | Z
                                /* qualquer letra minúscula ou
                                maiúscula */
<c_especial> ::= - | _ | . | # | ...
                                /* qualquer caracter não alfa-numérico que possa
                                fazer parte do nome de um arquivo (com as
                                restrições do sistema; isto é, número de caracteres,
                                etc) */
<barra> ::= /
                                /* faz parte do "path" */
<string> ::= <letra> | <string> <letra> | <string> <inteiro>
           ⇒ | <string> <c_especial>
                                /* qualquer combinação de caracteres
                                começada com letra */
<path> ::= <barra> | <string> | <c_especial> |
         ⇒ <path> <barra> | <path> <string> |
         ⇒ <path> <c_especial>
                                /* o "path" de um arquivo, com as restrições do
                                sistema (isto é, os tipos de caracteres permitidos,
                                etc) */
<a_r> ::= ABSOLUTE | RELATIVE
<t_f> ::= TRUE | FALSE
<e_t> ::= EACH_FRAME | TOTAL
<tempo> ::= <real_pos> | FRAME(<inteiro>)
                                /* indica tempo em segundos ou frames */
<tempo_2> ::= <tempo> | END
<t_mov_1> ::= At(<tempo_2>)
                                /* nao precisa <e_t> */
<t_mov_2> ::= Before(<tempo_2>) | After(<tempo>) |
            ⇒ Between(<tempo>, <tempo_2>)
                                /* exigem <e_t> */
```

/* Módulo GENERAL */

```
<mod_general> ::= GENERAL <totaltime_def> <rate_def>
                  ⇒ { [<ext_var>] }^n END
                                /* n é um número qualquer */
<totaltime_def> ::= TOTALTIME = <real_pos>
<rate_def> ::= FR_RATE = <inteiro>
<ext_var> ::= <var_type> <string> = <var_value> |
              ⇒ <var_type_pr> <string> = <var_value_pr>
                                /* variável externa: <tipo> <nome> = <valor> */
<var_type> ::= int | float | double | ...
                                /* qualquer tipo de variável da linguagem C */
<var_type_pr> ::= Point | Vector | Color
                                /* tipos de variáveis do ProSIm */
<var_value> ::= <inteiro> | <real> | <fcao_read_file> |
                ⇒ <fcao_foll_track>
                                /* a definição aqui não é livre de contexto; é preciso
                                   saber o tipo da variável para saber se ela assumirá
                                   um valor inteiro ou real */
<var_value_pr> ::= <real>, <real>, <real> | <fcao_read_file>
                   ⇒ | <fcao_foll_track>
<fcao_read_file> ::= read_file("<path>")
                                /* <path> deve indicar um arquivo com valores
                                   para a variável */
<fcao_foll_track> ::= following_track(<a_r>, <tr_name>,
                      ⇒ <acc>, <a_r>, <tempo>, <tempo_2> )
                                /* <tr_name> será definido no módulo TRACKS */
<acc> ::= linear | slow_in | slow_out | acc_dec | dec_acc
```

/* Módulo ACTORS */

```
<mod_actors> ::= ACTORS {<actor_name>: [<actor_def>]
                 ⇒ {[<actor_param>]}^s {[<actor_mov>]}^t}^n END
                                /* n atores (n, s e t quaisquer) */
<actor_name> ::= A<inteiro>
                                /* os atores devem ser numerados em ordem
                                   crescente, a partir do 0 */
<actor_def> ::= DEFINITION("<path>")
                                /* <path> deve indicar um arquivo com o formato
                                   de definição de atores: <arq_def_atores> - ver
                                   OBS. no final deste módulo */
<actor_param> ::= <brp_def> | <shade_def> | <color_def> |
                  ⇒ <p0_def>
<brp_def> ::= BRP = "<path_1>" | GEO_MORPH = "<path_2>",
                                ⇒ <inteiro>, <tempo_2>
                                /* onde <path_1> indica um arquivo do formato
                                   B-Rep, com a extensão .brp e <path_2>
                                   indica uma sequência numerada de arquivos
                                   B-Rep, sem a extensão .brp */
<shade_def> ::= SHADE_TYPE = <shade_type>
<shade_type> ::= BASIC | PHONGS | STRAUSS |
```

```
                ⇒ MARBLE | GRANITE | WOOD
<shadow_def> ::= SHADOW = <t_f>
<color_def> :: = COLOR = "<path_1>" |
                ⇒ CLR_MORPH = "<path_2>", <inteiro>,
                            ⇒ <tempo_2>
                            /* onde <path_1> indica um arquivo de cor,
                            com a extensão .cor e <path_2> indica uma
                            sequência numerada de arquivos de cor, sem
                            a extensão .cor */
<p0_def> ::= INITIAL_POSITION = <real>, <real>, <real>
<actor_mov>::= <t_mov_1> <a_movs> | <t_mov_2> <a_movs> <e_t>
<a_movs> ::= <parts> | <translates> | <rotates> | <scales> |
            ⇒ <repaint>
<parts> ::= part | unpart
<translates> ::= translate_actor(<a_r>, {<v_real>,}^2 <real>)
                ⇒ | translate_actor_<xyz>(<a_r>, <v_real>)
<xyz> ::= x | y | z
<rotates> ::= rotate_actor(<a_r>, {<v_real>,}^3 <v_real>) |
             ⇒ rotate_actor_<xyz>(<a_r>, <v_real>) |
             ⇒ free_rotate_actor({<v_real>,}^6 <v_real>)
<scales> ::= scale_actor(<a_r>, <v_real>,<v_real>,<v_real>)
            ⇒ | scale_actor_<xyz>(<a_r>, <v_real>) |
            ⇒ growth(<a_r>,<real>) | shrink(<a_r>,<v_real>)
            ⇒ | free_scale_actor({<v_real>,}^5 <v_real>)
<repaint> ::= repaint( <shade_type>, "<path>" )
                            /* onde <path> indica um arquivo de cor */
```

```
/* OBS.: Formato do arquivo de definição de atores: */
<arq_def_atores> ::= <brp_def> <shade_def> <color_def>
                  ⇒ [<p0_def>]
```

/* Módulo `GROUPS` */

```
<mod_groups> ::= GROUPS {<group_name>: <group_def>
                ⇒ <gc_def> {[<group_mov>]}^t}^n END
                            /* n grupos (n e t quaisquer) */
<group_name> ::= G<inteiro>
                            /* os grupos de atores devem ser numerados em
                            ordem crescente, a partir do 0 */
<group_def> ::= COMPONENTS([<actor_name>,]^n <actor_name>)
                            /* os n+1 nomes de atores do grupo devem ser
                            diferentes */
<gc_def> ::= GC = <v_real>, <v_real>, <v_real> |
            ⇒ GC = <actor_name>
<group_mov>::= <t_mov_1> <g_movs> | <t_mov_2> <g_movs> <e_t>
<g_movs> ::= <translates> | <rotates> | <scales>
                            /* <translates>, <rotates> e <scales> já
                            definidos no módulo ACTORS */
```

/* Módulo `CAMERA` */

```
<mod_camera> ::= CAMERA [<cam_def>] {[<cam_param>]}^s
                 ⇒ {[<cam_mov>]}^t END
                                /* s e t quaisquer */
<cam_def> ::= DEFINITION("<path>")
                                /* <path> deve indicar um arquivo com o formato
                                de definição de câmera: <arq_def_camera> - ver
                                OBS. no final deste módulo */
<cam_param> ::= <obs_def> | <coi_def> | <vup_def> | <d_def>
                ⇒ | <aperture_def> | <window_def>
<obs_def> ::= OBS = <real>, <real>, <real>
<coi_def> ::= COI = <real>, <real>, <real>
                                /* aqui também a linguagem não é livre de contexto;
                                é preciso que o COI seja diferente do OBS */
<vup_def> ::= VUP = <real>, <real>, <real>
<d_def> ::= D = <real_pos*>
                                /* por <real_pos*> entenda-se um real positivo
                                diferente de 0 */
<aperture_def> ::= APERTURE = <real_pos>, <real_pos>
<window_def> ::= WINDOW = <real_pos>, <real_pos>
<cam_mov> ::= <t_mov_1> <c_movs> | <t_mov_2> <c_movs> <e_t>
              ⇒ |<t_mov_1> aim_actor(<actor_name>) |
              ⇒ <t_mov_1> see_track(<tr_name>)
<c_movs> ::= <gos> | <aims> | <sets> | <standards>
<gos> ::= go_forward(<v_real>) | go_backward(<v_real>) |
          ⇒ go_up(<v_real>) | go_down(<v_real>) |
          ⇒ go_right(<v_real>) | go_left(<v_real>) |
          ⇒ go_north(<v_real>) | go_south(<v_real>) |
          ⇒ go_east(<v_real>) | go_west(<v_real>) |
          ⇒ go_flying(<v_real>, <v_real>, <v_real>) |
          ⇒ go_crow(<v_real>, <v_real>, <v_real>)
<aims> ::= aim_up(<v_real>) | aim_down(<v_real>) |
           ⇒ aim_in(<v_real>) | aim_out(<v_real>) |
           ⇒ aim_right(<v_real>) | aim_left(<v_real>) |
           ⇒ aim_north(<v_real>) | aim_south(<v_real>) |
           ⇒ aim_east(<v_real>) | aim_west(<v_real>) |
           ⇒ aim_scene
<sets> ::= set_obs(<v_real>, <v_real>, <v_real>) |
           ⇒ set_coi(<v_real>, <v_real>, <v_real>) |
           ⇒ set_vup(<v_real>, <v_real>, <v_real>) |
           ⇒ set_d(<v_real_pos*>) |
           ⇒ set_viewport({<v_inteiro>,}^3 <v_inteiro>) |
           ⇒ set_aperture(<v_real_pos>, <v_real_pos>) |
           ⇒ set_window(<v_real_neg>, <v_real_pos>,
           ⇒ <v_real_neg>, <v_real_pos>)
                                /* outra dependência do contexto: é preciso que
                                cada par <v_real_neg> e <v_real_pos> tenha o
                                mesmo valor absoluto */
<standards> ::= zoom_in(<v_real>) | zoom_out(<v_real>) |
                ⇒ spin_clock(<v_real>) |
                ⇒ spin_Cclock(<v_real>) |
```

```
                ⇒ tilt_up(<v_real>) | tilt_down(<v_real>) |
                ⇒ pan_right(<v_real>) | pan_left(<v_real>) |
                ⇒ traveling(<v_real>, <v_real>, <v_real>)
```

/* OBS.: Formato do arquivo de definição de câmera: */

```
<arq_def_camera> ::= [<obs_def>] [<coi_def>] [<vup_def>]
                    ⇒ [<d_def>] [<viewport_def>]
                    ⇒ [<apert_win_def>]
<apert_win_def> ::= [<aperture_def>] | [<window_def>]
```

/* Módulo `LIGHTS` */

```
<mod_lights> ::= LIGHTS [<lights_def>] {[<lights_param>]}^s
                ⇒ {[<lights_mov>]}^t END
                                /* s e t quaisquer */
<lights_def> ::= DEFINITION("<path>")
                                /* <path> deve indicar um arquivo com o formato
                                de definição de iluminação: <arq_def_lights> - ver
                                OBS. no final deste módulo */
<lights_param> ::= <env_def> | <back_def> | <source_def>
<env_def> ::= ENVIRONMENT = <real_pos>,<real_pos>,<real_pos>
<back_def> ::= BACK = <real_pos>, <real_pos>, <real_pos>
                                /* sempre que se fala em cor, deve-se lembrar que a
                                faixa de valores para cada componente (rgb) varia
                                de 0 a 65535 */
<source_def> ::= <source_name>: <point> |
                ⇒ <source_name>: <sun> |
                ⇒ <source_name>: <spot>
<source_name> ::= L<inteiro>
                                /* as fontes de luz devem ser numeradas em
                                ordem crescente, a partir do 0 */
<point> ::= TYPE = POINT
           ⇒ POSITION = <real>, <real>, <real>
           ⇒ COLOR = <real_pos>, <real_pos>, <real_pos>
<sun> ::= TYPE = SUN
          ⇒ DIRECTION = <real>, <real>, <real>
          ⇒ COLOR = <real_pos>, <real_pos>, <real_pos>
<spot> ::= TYPE = SPOT
          ⇒ POSITION = <real>, <real>, <real>
          ⇒ DIRECTION = <real>, <real>, <real>
          ⇒ COLOR = <real_pos>, <real_pos>, <real_pos>
          ⇒ SOLID_ANGLE = <real_pos>
          ⇒ SPOT_TYPE = <sharp_soft>
<sharp_soft> ::= SHARP | SOFT
<lights_mov> ::= <t_mov_1> <turns> | <t_mov_1> <redefs> |
                ⇒ Between(<tempo>, <tempo_2>) <redefs> |
                ⇒ Between(<tempo>, <tempo_2>) <fades>
                                /* <turns> só pode vir com At; <fades> deve
                                vir com Between */
<turns> ::= TURN_ON(<source_name>) | TURN_OFF(<source_name>)
```

```
<redefs> ::= POSITION(<source_name>)= {<v_real>,}^2 <v_real> |
          ⇒ DIRECTION(<source_name>)= {<v_real>,}^2 <v_real>
          ⇒ | COLOR(<source_name>)={<v_real_pos>,}^2
          ⇒ <v_real_pos> |
          ⇒ BACK = <real_pos>, <real_pos>, <real_pos>
<fades> ::= FADE_IN(<source_name>) | FADE_OUT(<source_name>)
          ⇒ | FADE_IN_ENV | FADE_IN_BACK | FADE_IN_ALL |
          ⇒ FADE_OUT_ENV | FADE_OUT_BACK | FADE_OUT_ALL
```

/* OBS.: Formato do arquivo de definição de iluminação: */

```
<arq_def_lights>::=[<env_def>] [<back_def>] {[<source_def>]}^n
```

/* Módulo RENDER */

```
<mod_render> ::= RENDER [FROM <tempo>] [TO <tempo_2>]
               ⇒ [ONE_FR_IN <int>] END
```

/* Módulo OUTPUT */

```
<mod_output> ::= OUTPUT [<out_path_def>] [<quality_def>]
               ⇒ [<s_shade_def>] [<s_shadow_def>]
               ⇒ [<s_sample_def>] [<viewport_def>] END
<out_path_def> ::= OUTPUT_PATH = "<path>"
<quality_def> ::= QUALITY = <quality_type>
<quality_type> ::= DEBUG | SCANRGB | SCANRAS | SCANMPEG |
                 ⇒ WIRE | SIPPTGA | SIPPPPM | SIPPMPEG
<s_shade_def> ::= SIPP_SHADE = <s_shade_type>
<s_shade_type> ::= PHONG_SIPP | GOUR_SIPP | FLAT_SIPP | LINE
<s_shadow_def> ::= SIPP_SHADOW = <t_f>
<s_sample_def> ::= SIPP_SAMPLE = <digito>
                              /* não é aconselhável usar dígito maior que 3 */
<viewport_def> ::= VIEWPORT = <inteiro>, <inteiro>
```

/* Módulo TRACKS */

```
<mod_tracks> ::= TRACKS { [<tr_name>: <tr_def>] }^n END
<tr_name> ::= T<inteiro>
                              /* as trilhas devem ser numeradas em
                              ordem crescente, a partir do 0 */
<tr_def> ::= <tr_1> | <tr_2>
                              /* <tr_1> é trilha lida em arquivo e <tr_2> é
                              trilha "conhecida" */
<tr_1> ::= TYPE = "<path>" <value_def_1> |
          ⇒ TYPE = "<path>" <value_def_2>
                              /* onde <path> indica o arquivo de pontos de
                              controle da trilha */
<value_def_1> ::= VALUE = <value_type_1>
<value_def_2> ::= VALUE = <value_type_2>
<value_type_1> ::= POINT | VECTOR | COLOR
<value_type_2> ::= DOUBLE
<tr_2> ::= TYPE = <tr_type> <resto_1> |
          ⇒ TYPE = <tr_type> <resto_2>
<tr_type> ::= LINEAR | QUADR | CUBIC | EXP
<resto_1> ::= <value_def_1>
             ⇒ START = <real>, <real>, <real>
             ⇒STOP = <real>, <real>, <real>
<resto_2> ::= <value_def_2>
             ⇒ START = <real>
             ⇒ STOP = <real>
```

/* Um roteiro de animação é composto dos módulos anteriormente definidos */

```
<ROTEIRO_DE_ANIMACAO> ::= <mod_general> <mod_actors>
                          ⇒ [<mod_groups>] [<mod_camera>]
                          ⇒ [<mod_lights>] [<mod_render>]
                          ⇒ [<mod_output>] [<mod_tracks>]
                            /* o roteiro será válido se conter apenas o
                            módulo GENERAL e o módulo ACTORS (com
                            pelo menos um ator), pois todos os outros
                            módulos têm valores default */
```

Para finalizar, alguns comentários devem ser feitos a respeito da formalização da linguagem de roteiros de animação do ProSIm:

1. A notação BNF tem limitações no que diz respeito à dependência do contexto. Isto é, as classes de símbolos são expandidas sem referência ao contexto no qual elas estão inseridas. Entretanto, a BNF foi usada em virtude de sua simplicidade e do pequeno número de situações em que há dependência do contexto na linguagem de roteiros do ProSIm (nessas situações, os comentários na descrição formal tentam suprir a limitação da BNF).

2. A linguagem de roteiros apresentada tem grande flexibilidade no que diz respeito à ordem em que os módulos (ou os elementos internos a eles) podem ser

apresentados. Para efeito de simplificação, entretanto, essa flexibilidade não é considerada na formalização utilizada, a qual "obriga" a apresentação dos módulos (e dos elementos internos a eles) em uma sequência específica.

3. As linhas com comentários no roteiro (começadas com " ; ") não foram incluídas na BNF.

Apêndice B

Editor de Texturas

Como no ProSIm não havia recursos disponíveis para a edição de arquivos de texturas, foi necessário o desenvolvimento de um programa externo à interface do TOOKIMA para esse fim. O objetivo é a criação interativa de texturas (arquivos .cor), que podem ser associadas aos atores de uma animação, como visto no capítulo VI.

A interface deste programa é bastante simples e está intimamente relacionada ao *renderer* (SIPP [Yngvesson, 94]), pois na criação de texturas são usados os parâmetros exigidos por ele. A Figura B.1 mostra a janela principal do editor de texturas.

Na primeira linha da janela, é escolhido o tipo de textura que se deseja criar, dentre os seis tipos permitidos pelo SIPP (são os tipos definidos pelo parâmetro `SHADE_TYPE`, no roteiro). Os outros parâmetros são definidos em função desta escolha.

Logo abaixo do tipo de textura, pode-se escolher a primeira cor da textura, manipulando suas componentes rgb. Se a textura escolhida for do tipo mármore, granito ou madeira, ainda é possível definir uma segunda cor, que será a cor das estrias.

Após a definição da(s) cor(es), vem a definição da opacidade associada a cada uma das componentes rgb da luz. Opacidade 1000 (na verdade 1, pois o valor dado está multiplicado por mil) significa completamente opaco, enquanto opacidade 0 significa uma textura completamente transparente à componente.

Logo abaixo, vem a definição do parâmetro “ambiente”, que indica o quanto da cor da superfície será visível quando ela não está diretamente iluminada.

Os parâmetros que vêm em seguida podem variar, de acordo com o tipo de textura a ser criado. Na Figura B.1 (textura do tipo mármore), aparecem os parâmetros “especular”, “C3” e “escala”. O parâmetro “especular” indica a porcentagem da luz incidente refletida especularmente. “C3” é um parâmetro relacionado ao brilho da superfície; 0 significa superfície muito brilhante e 1000 significa superfície sem brilho. A “escala” indica quão compacto é o padrão do mármore (ou do granito, ou da madeira). Valores mais altos deste parâmetro indicam padrões mais compactos.

Se a textura escolhida for do tipo plástico, ainda aparecerão os parâmetros “difuse” (indicando a porcentagem de reflexão difusa) e “Phong” (associado ao brilho). Texturas metálicas têm também o parâmetro “smoothness” (cujo valor 1000 significa superfície lisa e brilhante) e o parâmetro “metalness” (cujo valor 0 significa superfície não metálica e o valor 1000 significa completamente metálica).

Para ajudar a compreensão do significado de cada um desses parâmetros, na parte inferior da janela do editor de texturas há um campo de mensagens. Cada vez que qualquer parâmetro desta janela é alterado, aparece uma mensagem indicando seu significado. Na Figura B.1, por exemplo, aparece a mensagem que é mostrada por ocasião da escolha da textura do tipo mármore.

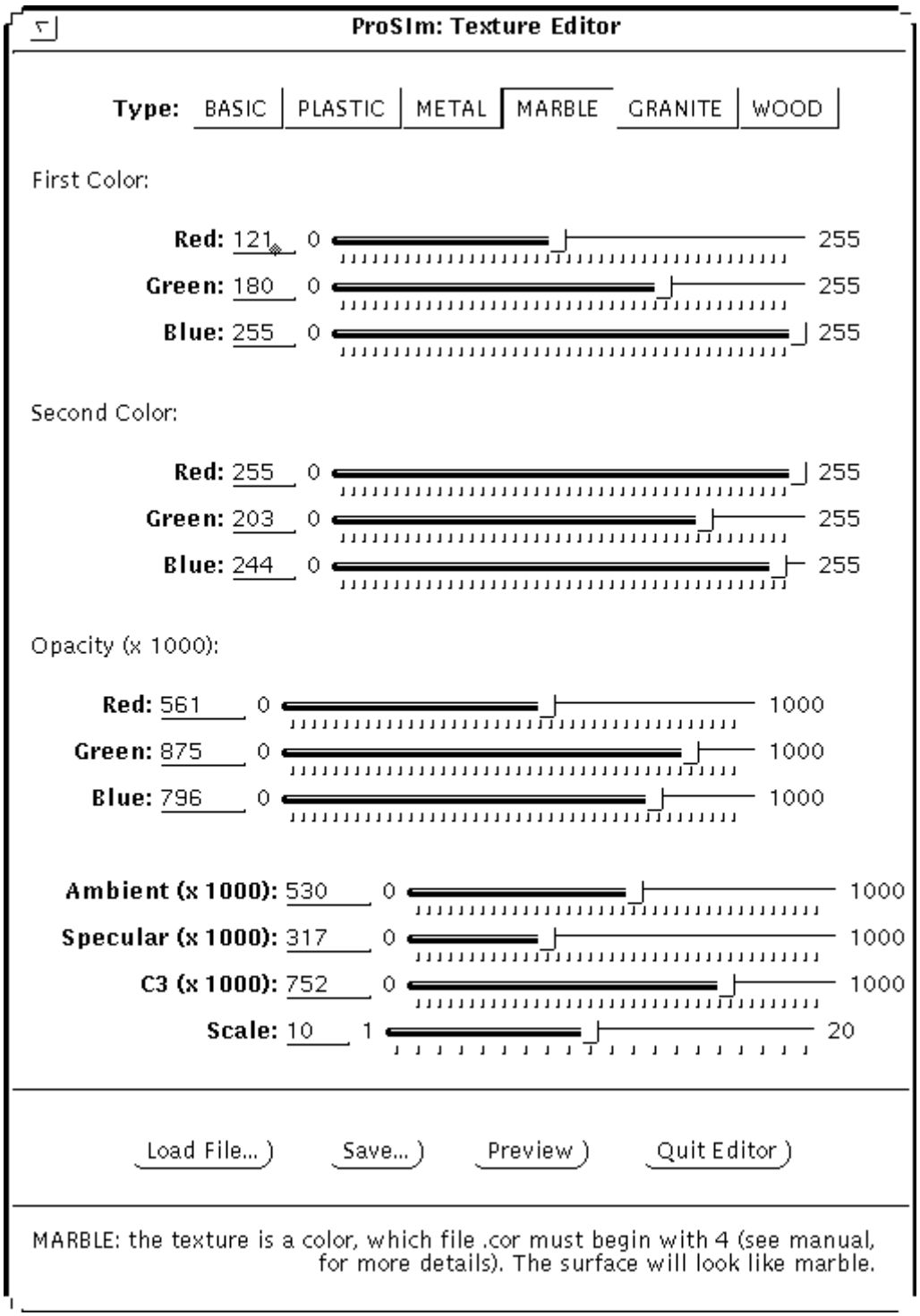

Figura B.1: Janela principal do Editor de Texturas.

O editor de texturas também apresenta quatro botões, localizados quase no final da janela, logo acima da área de mensagens. O primeiro deles (*Load File*) serve para carregar um arquivo de textura previamente criado. O botão *Save* salva os parâmetros da janela num arquivo .cor, cujo nome será pedido. O terceiro botão (*Preview*) faz uma rápida visualização da textura criada, mostrando a imagem de uma esfera gerada com esta textura. O botão *Quit Editor* causa a saída do programa.

Apêndice C

Organização do "Pacote"

O pacote "TOOKIMA 2.0", que engloba a interface, a linguagem de roteiros, além dos módulos adicionais (editor de texturas, de iluminação e de câmera) se encontra organizado segundo a seguinte árvore de diretórios.

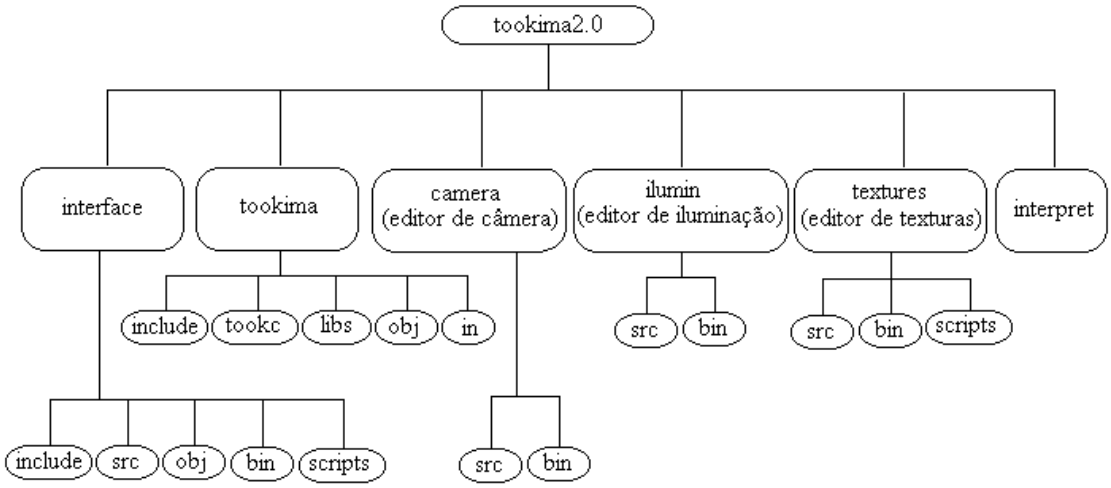

Figura C.1: Árvore de diretórios do TOOKIMA 2.0.

O diretório interface contém os arquivos-fonte (subdiretórios include e src) para a implementação da interface, além de um repositório de arquivos-objeto (subdiretório obj) e dos executáveis (subdiretório bin). No subdiretório scripts se encontram alguns exemplos de roteiros de animação e arquivos de definição de ator, câmera e iluminação.

O diretório tookima contém os arquivos-fonte do TOOKIMA (subdiretório include e tookc), suas bibliotecas, que são utilizadas pela interface (subdiretório libs), um repositório de arquivos-objeto (subdiretório obj) e exemplos de animações escritas na linguagem do TOOKIMA (subdiretório in).

Os diretórios camera e ilumin armazenam, respectivamente, os arquivos referentes ao editor de câmera e de iluminação. Ambos possuem um subdiretório src, com os arquivos-fonte e um subdiretório bin, com os executáveis.

O diretório textures armazena os arquivos referentes ao editor de texturas. Possui um subdiretório src, com os arquivos-fonte, um subdiretório bin, com os executáveis e um subdiretório scripts, com arquivos necessários à pré-visualização das texturas.

O diretório interpret possui arquivos utilizados em versões anteriores do programa, atualmente fora de uso.

É válido relembrar que, além destes arquivos, a versão 2.0 do TOOKIMA também necessita das bilbiotecas X, do XV, do codificador e do visualizador de MPEGs.

BIBLIOGRAFIA

[Alegre, 94]
Alegre, I. A. V.
"Texturas em Síntese de Imagens"
Dissertação de Mestrado, DCA - FEE - UNICAMP - 1994.

[Alegre, 95]
Alegre, P. V.
"Interações 3D através de uma Classe de Dispositivos 2D"
Dissertação de Mestrado, DCA - FEE - UNICAMP - 1995.

[Autodesk, 89]
"Autodesk Animator"
Autodesk, Inc. - Sausalito, CA - 1989.

[Autodesk, 90]
"Autodesk 3D Studio"
Autodesk, Inc. - Sausalito, CA - 1990.

[Badler, 95]
Badler, N. I.
"Computer Animation Techniques"
In: "Introduction to Computer Graphics" - Bailey, M. et alli.
Course Notes for SIGGRAPH 95.

[Bernardes, 95]
Bernardes, M. C. e Wu Shin-Ting
"Fuzzy sets on drawing fair plane curves"
SIBGRAPI 95 (VIII Simpósio Brasileiro de Computação Gráfica e Processamento de Imagens) - pp.87-94 - 1995

[Besuievsky, 93]
Besuievsky, S. G. e Magalhães, L. P.
"Cálculo da iluminação através de ray-tracing estocástico"
SIBGRAPI 93 (VI Simpósio Brasileiro de Computação Gráfica e Processamento de Imagens) - pp.19-26 - 1993.

[Blinn, 78]
Blinn, J. F.
"Simulation of Wrinkled Surfaces"
Computer Graphics - v.12, n.3, pp.286-292 - 1978.

[Booth, 83]
Booth, K. S. and Kochanek, D. H.
"Computers animate films and video"
IEEE Spectrum, pp.44-51 - February, 1983.

[Bouknight, 70]
Bouknight, J.
"A Procedure for Generation of Three-dimensional Half-toned Computer Graphics Presentations"
Communications of the ACM - v.13, pp.527-536 - 1970.

[Bradley, 93]
Bradley, J.
"XV - Interactive Image Display for the X Window System - Version 3.00"
(Manual do Usuário) - 1993.

[Breen, 87]
Breen, D. E. et alli.
"The Clockworks: An Object-Oriented Computer Animation System"
EUROGRAPHICS'87 - pp.275-282 - 1987.

[Camargo, 93]
Camargo, J. T. F. e Magalhães, L. P.
"Técnicas de controle em animação"
Tutorial apresentado no SIBGRAPI 93 (VI Simpósio Brasileiro de Computação Gráfica e Processamento de Imagens) - 1993.

[Camargo, 94]
Camargo, J. T. F., Magalhães, L. P. and Raposo, A. B.
"Modeling motion simulation with DEDS"
Proc. of 13th. IFIP Congress - International Federation of Information Processing pp.162-167 - 1994.

[Camargo, 95a]
Camargo, J. T. F.
"Animação modelada por computador: técnicas de controle de movimento"
Dissertação de Doutorado, DCA - FEE - UNICAMP - 1995.

[Camargo, 95b]
Camargo, J. T. F., Magalhães, L. P. e Raposo, A. B.
"Fundamentos da animação modelada por computador"
Tutorial apresentado no SIBGRAPI 95 (VIII Simpósio Brasileiro de Computação Gráfica e Processamento de Imagens) - 1995.

[Camargo, 95c]
Camargo, J. T. F., Magalhães, L. P. e Raposo, A. B.
"Local and global control in computer animation"
SIBGRAPI 95 (VIII Simpósio Brasileiro de Computação Gráfica e Processamento de Imagens) - pp.151-157 - 1995.

[Card, 83]
Card, S. K., Moran, T. P. and Newell, A.
"The Psychology of Human-Computer Interaction"
Erlbaum - 1983.

[Catmull, 74]
Catmull, E.
"A Subdivision Algorithm for Computer Display of Curved Surfaces"
Doctoral thesis - University of Utah - 1974.

[Celes, 95]
Celes Filho, W., Figueiredo, L. H. e Gattass M.
"EDG: uma ferramenta para criação de interfaces gráficas interativas"
SIBGRAPI 95 (VIII Simpósio Brasileiro de Computação Gráfica e Processamento de Imagens) - pp.241-248 - 1995.

[Cook, 82]
Cook, R. L. and Torrance, K. E.
"A Reflectance Model for Computer Graphics"
ACM Transactions on Graphics - v.1, pp.7-24 - 1982.

[Cook, 87]
Cook, R. L., Carpenter, L. and Catmull E.
"The Reyes Image Rendering Architecture"
Computer Graphics - v.21, n.4, pp.95-102 - 1987.

[Cordy, 92]
Cordy, J. R.
"Why the User Interface Is *Not* the Programming Language - and How It *Can* Be"
Capítulo 6 de [Myers, 92a] - 1992.

[Costa, 95]
Costa, M., Feijó, B. e Schwabe, D.
"Reactive Agents in Behavioral Animation"
SIBGRAPI 95 (VIII Simpósio Brasileiro de Computação Gráfica e Processamento de Imagens) - pp.159-165 - 1995.

[Costa, 96]
Costa, M., Feijó, B.
"Agents with Emotions in Behavioral Animation"
Computer & Graphics - v.20, n.3 - 1996.

[Daldegan, 95]
Daldegan, A. e Guadagnin, R. V.
"An Integrated Approach to Improve the Facial Emotional Behavior of Virtual Actors"
SIBGRAPI 95 (VIII Simpósio Brasileiro de Computação Gráfica e Processamento de Imagens) - pp.167-173 - 1995.

[Deering, 95]
Deering, M. F.
"HoloSketch: A Virtual Reality Sketching / Animation Tool"
ACM Transactions on Computer-Human Interaction - v.2, n.3, pp.220-238 - 1995.

[Dertouzos, 92]
Dertouzos, M. L.
"The User Interface is *The* Language"
Capítulo 2 de [Myers, 92a] - 1992.

[Diz, 95]
Diz, J. A.
"Síntese de Efeitos Fotográficos em Computação de Imagens"
Dissertação de Mestrado, DCA - FEE - UNICAMP - 1995.

[Eisenberg, 95]
Eisenberg, M.
"Programmable Applications: Interpreter Meets Interface"
SIGCHI Bulletin - v.27, n.2, pp.68-93 - 1995.

[Fekete, 95]
Fekete, J. D. et alli.
"TicTacToon: A Paperless System for Professional 2D Animation"
Proc. of SIGGRAPHI'95 - pp.79-89 - 1995.

[Foley, 92]
Foley, J. D., van Dam, A., Feiner, S. K. and Hughes, J. F.
"Computer Graphics: Principles and Practice"
2nd. ed. - Addison-Wesley - 1992.

[Freiwald, 92]
Freiwald, L. and Marrs, L.
"AutoDesk Animator - Guia Completo de Animação no PC"
Berkeley Brasil Editora - 1992.

[Gear, 74]
Gear, C. W.
"Computer Organization and Programming"
2nd. ed. - McGraw-Hill Book Co. - 1974.

[Gong, 94]
Gong, K. L.
"Berkeley MPEG-1 Video Encoder - User's Guide"
Computer Science Division - University of California - Berkeley, CA - 1994.
Encontrado via ftp em: tr-ftp.cs.berkeley.edu (/pub/multimedia/mpeg)

[Goral, 84]
Goral, C. M., Torrance, K. E., Greenberg, D. P. and Battaile, B.
"Modeling the Interaction of Light Between Diffuse Surfaces"
Computer Graphics - v.18, n.3, pp.213-221 - 1984.

[Gouraud, 71]
Gouraud, H.
"Computer Display of Curved Surfaces"
IEEE Transactions C-20, pp.623-628 - 1971.

[Green, 85]
Green, M.
"The University of Alberta User Interface Management System"
Computer Graphics - v.19, n.3, pp.205-213 - 1985

[Hartson, 90]
Hartson, R. H., Siochi, A. C. and Hix, D.
"The UAN: A User-Oriented Representation for Direct Manipulation Interface Designs"
ACM Transactions on Information Systems - v.8, n.3, pp.181-203 - 1990.

[Heller, 90]
Heller, D.
"Xview Programming Manual"
X Window System Series - Volume Seven
O'Reilly & Associates, Inc. - 1990.

[Hix, 93]
Hix, D. and Hartson, H. R.
"Developing User Interfaces: Ensuring Usability Through Product & Process"
John Wiley & Sons, Inc. - 1993.

[Horta, 95]
Horta, A. A. e Wu Shin-Ting
"Deformação de superfícies não rígidas baseada em princípios físicos"
SIBGRAPI 95 (VIII Simpósio Brasileiro de Computação Gráfica e Processamento de Imagens) - pp.175-182 - 1995.

[Hounsell, 92a]
Hounsell, M. S.
"TOOKIMA: uma ferramenta para animação modelada por computador."
Dissertação de Mestrado, DCA - FEE - UNICAMP - 1992.

[Hounsell, 92b]
Hounsell, M. S. e Magalhães, L. P.
"TOOKIMA - animação cinemática"
SIBGRAPI 92 (V Simpósio Brasileiro de Computação Gráfica e Processamento de Imagens) - pp.269-273 - 1992.

[Ierusalimschy, 95]
Ierusalimschy, R., Figueiredo, L. H. e Celes Filho, W.
"Lua - an extensible extension language"
MCC 12/95 - Departamento de Informática - PUC-Rio
Encontrado em: http://www.inf.puc-rio.br/projetos/roberto/lua.html

[Jacob, 86]
Jacob, R. J. K.
"A Specification Language for Direct Manipulation User Interfaces"
ACM Transactions on Graphics - v.5, n.4, pp.283-317 - 1986.

[John, 89]
John, N. W. and Willis, P. J.
"The Controller Animation System"
Computer Graphics Forum - v.8, n.2, pp.133-138 - 1989.

[Kalra, 92]
Kalra, D. and Barr, A. H.
"Modeling with Time and Events in Computer Animation"
Computer Graphics Forum - v.11, n.3, pp-C-45-C-58 - EUROGRAPHICS'92.

[Kay, 79]
Kay, D. S.
"Transparency, Refraction and Ray Tracing for Computer Synthetized Images"
Masters thesis - Program of Computer Graphics - Cornell University - 1979.

[Koga, 94]
Koga, Y., Kondo, K., Kuffner, J. and Latombe J. C.
"Planning Motions with Intentions"
Proc. of SIGGRAPHI'94 - pp.395-408 - 1994.

[Lasseter, 87]
Lasseter, J.
"Principles of Traditional Animation Applied to 3D Computer Animation"
Computer Graphics - v.21, n.4, pp.35-44 - 1987.

[Le Gall, 91]
Le Gall, D.
"MPEG: A Video Compression Standard for Multimedia Applications"
Communications of the ACM - v.34, n.4, pp.46-58 - 1991.

[Lengyel, 90]
Lengyel, J., Reichert M., Donald, B. R. and Greenberg, D. P.
"Real-Time Robot Motion Planning Using Rasterizing Computer Graphics Hardware"
Computer Graphics - v.24, n.4. pp.327-335 - 1990.

[Levy, 93]
Levy, C. H.
"IUP/LED: uma ferramenta portátil de interface com usuário"
Dissertação de Mestrado, Depto. de Informática, PUC-Rio, 1993.

[Linton, 89]
Linton, M. A., Vlissides J. M. and Calder P. R.
"Composing User Interfaces with Interviews"
IEEE Computer - v.22, n.2, pp.8-22 - 1989.

[Magalhães, 95]
Magalhães, L. P.
" Animação - Integrando Conceitos"
Palestra apresentada no SIBGRAPI 95 (VIII Simpósio Brasileiro de Computação Gráfica e Processamento de Imagens) - 1995.

[Malheiros, 94]
Malheiros, M. G.
"Manual da Interface Gráfica do Sistema ProSIm"
Relatório Interno - DCA - FEE - UNICAMP - 1994.

[Malheiros, 95]
Malheiros, P.
"3D Studio 4.0 - Guia completo"
Berkeley Brasil Editora - 1995.

[Mandelbrot, 82]
Mandelbrot, B. B.
"The fractal geometry of nature"
Freeman - 1982.

[McCormack, 88]
McCormack, J. and Asente P.
"An Overview of the X Toolkit"
Proc. of UIST 88 (ACM SIGGRAPH Symposium on User Interface Software and Technology) - pp.46-55 - 1988.

[McLachlan, 85]
McLachlan, D. R.
"CORY: An Animation Scripting System"
Masters thesis - Rensselaer Polytechnic Institute - Troy, New York - May 1985.

[Microsoft, 92]
"Visual Basic Programmer's Guide"
Microsoft, Inc. - 1992.

[Moltedo, 93]
Moltedo, L. and Morigi, S.
"ANIMA: An Interactive Tool for Scientific Data Animation"
Computer Graphics Forum - v.12, n.5, pp.277-288 - 1993.

[Moran, 81]
Moran, T. P.
"The Command Language Grammar: a representation for the user interface interactive computer systems"
International Journal of Man-Machine Studies - n.15, pp.3-50 - 1981.

[Morrison, 94]
Morrison, M.
"Becoming a Computer Animator" - Sams - 1994.
Resumo da história da computação gráfica encontrada em http://www.toystory.com

[Myers, 88]
Myers, B. A.
"Creating User Interfaces by Demonstration"
Academic Press - 1988.

[Myers, 92a]
Myers, B. A.
"Languages for Developing User Interfaces"
Jones and Bartlett Pub. - 1992.

[Myers, 92b]
Myers, B. A. and Rosson, M. B.
"Survey on User Interface Programming"
Proc. of ACM - CHI 92 (Conference on Human Factors in Computing Systems)
pp.195-202 - 1992
Encontrado em: http://www.cs.cmu.edu/afs/cs/project/garnet/www/papers.html

[Myers, 93]
Myers, B. A., Wolf, R., Potosnak, K. and Graham, C.
Panel: "Heuristics in Real User Interfaces"
Proc. of INTERCHI 93 (Human Factors in Computing Systems) - pp.304-307
1993
Encontrado em: http://www.cs.cmu.edu/~bam

[Myers, 96]
Myers, B. A.
"User Interface Software Tools"
Submetido para publicação.
Encontrado em: http://www.cs.cmu.edu/afs/cs/project/garnet/www/papers.html

[Neibauer, 92]
Neibauer, A. R.
"The ABC's of Windows 3.1"
Sybex Inc. - 1992.

[Newman, 79]
Newman, W. M. and Sproull, R. F.
"Principles of Interactive Computer Graphics"
2nd. ed. - McGraw-Hill Book Co. - 1979.

[Nielsen, 90]
Nielsen, J.
"Traditional Dialog Design Applied to Modern User Interfaces"
Communications of the ACM - v.33, n.10, pp.109-118 - 1990.

[Nielsen, 92]
Nielsen, J.
"Finding Usability Problems Through Heuristic Evaluation"
Proc. of CHI (Conference on Human Factors in Computing Systems) - pp.373-380
1992.

[Nielsen, 93]
Nielsen, J.
"Noncommand User Interfaces"
Communications of the ACM - v.36, n.4, pp.83-99 - 1993.

[Norman, 86]
Norman, D. A. and Draper, S. W. (eds)
"User Centered System Design: New Perspectives on Human-Computer Interaction"
Hillsdale, NJ - Lawrence Erlbaum and Associates, Publishers - 1986.

[Open, 89]
Open Software Foundation - "OSF/Motif Style Guide"
Prentice Hall - 1989.

[Palamidese, 94]
Palamidese, P. and Crise, A.
"Animations to Represent Multivariate Data"
Computer Graphics Forum - v.13, n.4, pp.243-248 - 1994.

[Payne, 86]
Payne, S. J. and Green, T. R. G.
"Task-Action Grammars: A Model of the Mental Representation of Task Languages"
Human-Computer Interaction - v.2, pp.93-133 - 1986

[Perez, 79]
Lozano-Perez, T. and Wesley, M. A.
"An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles"
Communications of the ACM - v.22, n.10, pp.560-570 - 1979.

[Phong, 74]
Phong, Bui-Toung
"Illumination for Computer Generated Images"
Doctoral thesis - University of Utah - 1973.

[Pressman, 92]
Pressman, R. S.
"Software Engineering - A Practitioner's Approach"
3rd. Edition - McGraw-Hill, Inc. - 1992.

[Preston, 94]
Preston, M and Hewitt, W. T.
"Animation using NURBS"
Computer Graphics Forum - v.13, n.4, pp.229-241 - 1994.

[Preto, 92]
Preto, T. M.
"Scanline: um sistema para visualização de imagens foto-realistas"
Dissertação de Mestrado, DCA - FEE - UNICAMP - 1992.

[Pueyo, 88]
Pueyo, X. and Tost, D.
"A Survey of Computer Animation"
Computer Graphics Forum - v.7, pp.281-300 - 1988

[Queiroz, 91]
Queiroz, M. S. e Magalhães, L. P.
"O método radiosidade em um ambiente de síntese de imagens foto-realistas"
SIBGRAPI 91 (IV Simpósio Brasileiro de Computação Gráfica e Processamento de Imagens) - pp.155-166 - 1991.

[Raposo, 95a]
Raposo, A. B.
"TOOKIMA: uma Ferramenta Cinemática para Animação"
Relatório Interno - 001/95 - DCA - FEE - UNICAMP - 1995.

[Raposo, 95b]
Raposo, A. B.
"Uma Linguagem para Desenvolvimento de Roteiros de Animação"
Relatório Interno - 002/95 - DCA - FEE - UNICAMP - 1995.

[Reeves, 81]
Reeves, W. T.
"Inbetweening for Computer Animation Utilizing Moving Point Constraints"
Computer Graphics - v.15, n.3, pp.263-269 - 1981.

[Reeves, 83]
Reeves, W. T.
"Particle systems - a technique for modeling a class of fuzzy objects"
Computer Graphics - v.17, n.3, pp.359-376 - 1983.

[Reynolds, 82]
Reynolds, C. W.
"Computer Animation with Scripts and Actors"
Computer Graphics - v.16, n.3, pp.289-296 - 1982.

[Roberts, 63]
Roberts, L. G.
"Machine Perception of Three Dimensional Solids"
MIT Lincoln Lab, TR 315 - 1963

[Robertson, 93]
Robertson, G. G., Card, S. K. and MacKinlay, J. D.
"Information Visualization Using 3D Interactive Animation"
Communications of the ACM - v.36, n.4, pp.56-71 - 1993.

[Robbins, 95]
Robbins, D. C. et alli.
"Practical 3D User Interface Design"
Course Notes for SIGGRAPH 95 - 1995.

[Rodrigues, 93a]
Rodrigues, M. A. F.
"Animado: um protótipo de um sistema de animação modelada por dinâmica"
Dissertação de Mestrado, DCA - FEE - UNICAMP - 1993.

[Rodrigues, 93b]
Rodrigues, M. A. F. e Magalhães, L. P.
"ANIMADO - Um protótipo de um Sistema de Animação modelada por dinâmica"
SIBGRAPI 93 (VI Simpósio Brasileiro de Computação Gráfica e Processamento de Imagens) - pp.107-115 - 1993.

[Rogers, 85]
Rogers, D. F.
"Procedural Elements for Computer Graphics"
McGraw-Hill Book Co. - 1985.

[Rohlf, 94]
Rohlf, J. and Helman, J.
"IRIS Performer: A High Performance Multiprocessing Toolkit for Real-Time 3D Graphics"
Proc. of SIGGRAPHI'94 - pp.381-394 - 1994.

[Scheifler, 86]
Scheifler, R. W. and Gettys, J.
"The X Window System"
ACM Transactions on Graphics - v.5, n.2, pp.79-109 - 1986.

[Shneiderman, 92]
Shneiderman, B.
"Designing the User Interface: Strategies for Effective Human-Computer Interaction"
2nd. Ed. - Addison-Wesley - 1992.

[Silveira, 94]
Silveira, L. M.
"A reprodução cromática em síntese de imagens: um estudo comparativo à pintura"
Dissertação de Mestrado, IA - UNICAMP - 1994.

[Snibbe, 95]
Snibbe, S. S.
"A Direct Manipulation Interface for 3D Computer Animation"
Computer Graphics Forum - v.14, n.3, pp.C-271-C-283 - EUROGRAPHICS'95.

[Souza, 93a]
Souza, C. S.
"Regularidade e Generalização em Interfaces Gráficas"
SIBGRAPI 93 (VI Simpósio Brasileiro de Computação Gráfica e Processamento de Imagens) - pp.183-191 - 1993.

[Souza, 93b]
Souza, C. S.
"The Semiotic Engineering of user interface languages"
International Journal of Man-Machine Studies - n.39, pp.753-773 - 1993.

[Souza, 96]
Souza, C. S.
"The Semiotic Engineering of Concreteness and Abstractness: From User Interface Languages to End User Programming Languages"
Dagstuhl Seminar on Informatics and Semiotics - 1996.
Encontrado em http://www.inf.puc-rio.br/~sergweb

[Stahlke, 93]
Stahlke, T. M.
"Geometria Fractal: um estudo da teoria com proposição de taxonomia baseada no proceso de geração"
Dissertação de Mestrado, DCA - FEE - UNICAMP - 1993.

[Sun, 89]
Sun Microsistems - "OpenLook Graphical User Interface Application Style Guidelines"
Addison-Wesley - 1989.

[Sutherland, 63]
Sutherland, I. E.
"Sketchpad: A Man-Machine Graphical Communication System"
Proc. of AFIPS Spring Joint Computer Conference - v.23, pp.329-346 - 1963.

[Thalmann, 85a]
Magnenat-Thalmann, N. and Thalmann, D.
"Computer Animation: Theory and Practice"

Springer-Verlag - 1985.

[Thalmann, 85b]
Magnenat-Thalmann, N., Thalmann, D. and Fortin, M.
"Miranim: An Extensible Director-Oriented System for the Animation of Realistic Images"
IEEE Computer Graphics and Applications - v.5, pp.61-73 - March 1985.

[Thalmann, 89]
Magnenat-Thalmann, N. and Thalmann, D.
"The Problematics of Human Prototyping and Animation"
Computer Graphics Forum - v.8, n.2, pp.115-123 - 1989.

[Thalmann, 91]
Magnenat-Thalmann, N. and Thalmann, D.
"Complex Models for Animating Synthetic Actors"
IEEE Computer Graphics and Applications - v.11, n.5, pp.32-44 - 1991.

[Upstill, 90]
Upstill, S.
"The RenderMan Companion: A Programmer's Guide to Realistic Computer Graphics"
Addison-Wesley - 1990.

[Warnock, 69]
Warnock, J. E.
"A Hidden-Surface Algorithm for Computer Generated Halftone Pictures"
University of Utah Computer Science Dept. - Rep. TR 4-15, NTIS 761 995 - 1969.

[Watt, 93]
Watt, A. and Watt, M.
"Scripting and Artistic Control in Computer Animation"
SIBGRAPI 93 (VI Simpósio Brasileiro de Computação Gráfica e Processamento de Imagens) - pp.123-132 - 1993.

[Whitted, 79]
Whitted, J. T.
"An Improved Illumination Model for Shaded Display"
Communications of the ACM - v.23, pp.343-349 (Proc. SIGGRAPH 79) - 1979.

[Wilhelms, 90]
Wilhelms, J. and Skinner, R.
"A 'Notion' for Interactive Behavioral Animation Control"
IEEE Computer Graphics and Applications - v.10, n.3, pp.14-22 - 1990.

[Wyvill, 90]
Wyvill, B.
"A Computer Animation Tutorial"
Cap. 4 de "Computer Graphics Techniques: Theory and Practice"
Rogers, D. F. and Earnshaw, R. A. (Editors)
Springer-Verlag - 1990.

[Yngvesson, 94]
Yngvesson, J. and Wallin, I.
"User's Guide to SIPP - a 3D Rendering Library - Version 3.1"
1994.

[Young, 94]
Young, C. and Wells, D.
"Persistence of Vision Ray Tracer (POV-Ray) - Version 2.0 - User's Documentation"
1994.

[Zeleznik, 91]

Zeleznik, R. C. et alli.
"An Object-Oriented Framework for the Integration of Interactive Animation Techniques"
Computer Graphics - v.25, n.4, pp.105-111 - 1991.

[Zeltzer, 85]
Zeltzer, D.
"Towards an integrated view of 3-D computer animation"
The Visual Computer - v.1, n.4, pp.249-259 - 1985.