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

TOOKIMA:
UM CONJUNTO DE FERRAMENTAS PARA DESCRIÇÃO DE ANIMAÇÃO MODELADA POR COMPUTADOR

Por :Eng. Marcelo da Silva Hounsell

Orientador :Prof. Léo Pini Magalhães

Este exemplar corresponde à redação final da tese defendida por Marcelo da Silva Hounsell e aprovada pela Comissão Julgadora em 28/02/92.

Orientador

TESE APRESENTADA À FACULDADE DE ENGENHARIA ELÉTRICA, DA UNIVERSIDADE ESTADUAL DE CAMPINAS, COMO PARTE DOS REQUISITOS EXIGIDOS PARA OBTENÇÃO DO TÍTULO DE MESTRE EM ENGENHARIA ELÉTRICA.

FEVEREIRO 1992

DEDICATÓRIA

Dedico esta dissertação a três pessoas que me são caras:

Ao meu pai Eduardo, meu exemplo de sapiência e hombridade.

À minha mãe Regina, meu exemplo de moral e mulher.

À minha esposa Renata, meu amor, minha amiga e companheira.

AGRADECIMENTOS

Ao professor e orientador, Léo Pini Magalhães, pela amizade que construimos, confiança e orientação recebida.

Ao amigo Anastácio pelo incentivo no momento certo.

Aos amigos do ProSIm, Andrea (Andy), Andreia (Déia), Gonzalo, Graciano, Heraldo, Jorge, Luciana, Mário, Perla, Shigueo, Tânia (Taniovsky) e, Tânia (Frac-Tânia).

Aos demais amigos da FEE, Armando, Castanho, Diego, Eloisa (Elô), Fernando, Gorgônio, Gustavo, João (Pará), José Olguin, Josué, Luiz (Pingo), Marcel, Marcia, Miguel, Myriam, Olga, Plínio, Ramiro, Regina Ruschel, Ricardo L., Ricardo Lü., Ricardo G., Rogério, Silvinha, Victor.

Sem a ajuda deles você não estaria lendo esta dissertação.

RESUMO

TOOKIMA (a **TOO**l **KI**t for Scripting Computer **M**odeled **A**nimation) consiste num conjunto de ferramentas para facilitar a descrição e o desenvolvimento de animações cinemáticas. Vários algoritmos foram implementados para a manipulação de quaisquer dos parâmetros dos elementos tais como objetos, câmera, tonalização, fontes de luz, etc.

Ele é, atualmente, uma extensão da linguagem de programação de uso geral C, sobre seus tipos de dados primitivos e sobre sua biblioteca de funções, baseado no *rendering Scan-line* sobre objetos representados por bordos.

Características adicionais estão disponíveis para animação sobre trilhas, geometria temporal, vários comportamentos de aceleração pré-estabelecidos, estruturação hierárquica de objetos e o desenvolvimento incremental da animação.

Nosso objetivo foi, principalmente, desenvolver um ambiente inicial para pesquisas em animação no nosso projeto maior em Computação de Imagens, denominado ProSim, do que propor técnicas ou algoritmos novos.

ABSTRACT

TOOKIMA (a **TOO**l **KI**t for Scripting Computer **M**odeled **A**nimation) consists of a set of tools that intend to help the description and development of kinematics animations. Several algorithms are available for the manipulation of any parameters of elements like objects, camera, shading, light-sources, etc.

These tools, nowadays, are a scanline-based rendering extension to the C general-purpose programming language on its data types and libraries, using boundary representation of objects.

Additional features are available for track animation, temporal geometry, several accelerated behavior, hierarquical structure of objects and tweaking.

Our goal was, mainly, to develop an initial research enviroment for animation in our major project on computer graphics, called ProSIm, rather than propose any new techniques or algorithms.

ÍNDICE

CAPÍTULO 3
TOOKIMA : FERRAMENTAS PARA ANIMAÇÃO MODELADA POR COMPUTADOR

CAPÍTULO 4
TOOKIMA : ELEMENTOS DA CENA

CAPÍTULO 6
TOOKIMA : VISUALIZAÇÃO

CAPÍTULO 7
CONSIDERAÇÕES FINAIS

CAPÍTULO 8
BIBLIOGRAFIA

APÊNDICE
EXEMPLO : MANÉ MOSQUITO E CORUJITO

LISTA DE FIGURAS

CAPÍTULO 1

INTRODUÇÃO

1.1 MOTIVAÇÃO

Animação é um dos temas estudados na área de Computação de Imagens[1]. Por computação entenda-se o ferramental a ser utilizado, tanto em hardware, quanto em software. Imagem indica o sujeito do trabalho computacional. O termo animação incorpora, basicamente, a noção de movimento e sua reprodução através de superposição rápida e progressiva de imagens para a simulação da idéia de tempo, como será tratado mais profundamente no decorrer do trabalho.

A abrangência da área acima é grande, englobando desde a fase da representação e descrição dos dados (objetos, cores, estruturas topológicas, roteiros, hierarquia de objetos, interação, etc.), passando pelos vários algoritmos de manipulação (visibilidade, *rendering*, operações geométricas construtivas, avaliação de características, etc.) até, por exemplo, algoritmos de tratamento final (detecção de padrões, manipulação de características, armazenamento de resultados, compactação, etc.).

Tal abrangência é sentida cabalmente quando se definem sistemas de animação, por exemplo, que pretendam ser flexíveis e extensíveis (como normalmente são os sistemas acadêmicos).

[1] Computação de Imagens é a denominação que adotamos para atividades que tem como objeto imagens, abrangendo assim síntese, processamento de imagens e visão computacional.

Na sequência apresentaremos o projeto ProSIm (Prototipação e Síntese de Imagens Foto-Realistas e Animação) e o relacionamento deste com o **TOOKIMA** (a **TOO**l **KI**t for Scripting Computer Modeled **A**nimation). Em seguida discorreremos brevemente sobre o histórico deste trabalho e finalizaremos esta introdução apresentando a estrutura da dissertação.

1.2 O PROJETO ProSIm

ProSIm (Prototipação e Síntese de Imagens Foto-Realistas e Animação, ver MAGA/91) é um projeto, bastante abrangente, para a implementação de diversos módulos/sistemas relacionados à Computação de Imagens. Os módulos componentes, superficialmente divididos em Modelagem, *Rendering*, Interface e Animação, são *a priori*, independentes entre si mas interdependentes de um módulo de funções básicas, denominado "prosim", de forma que a integração (ou difusão de vantagens) seja facilitada e incentivada (pois é desejável).

O ProSIm pode ser melhor visualizado na figura 1.1, abaixo:

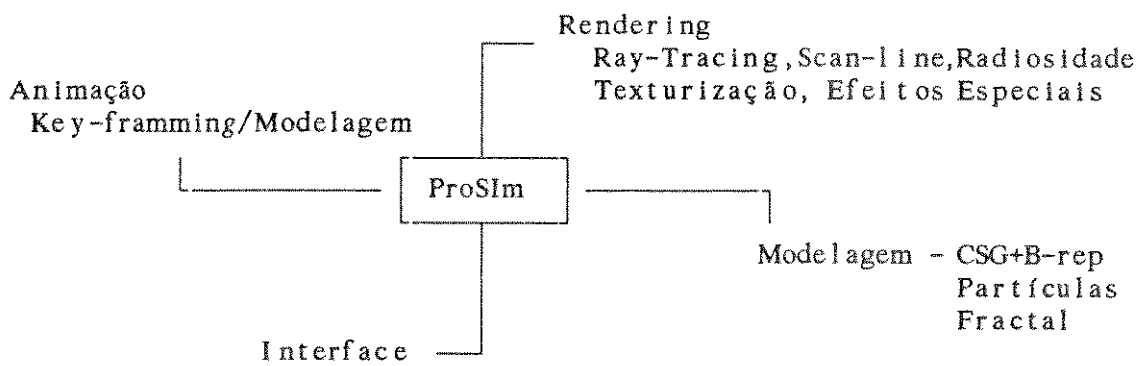


Figura 1.1 : O Projeto ProSIm.

ProSIm, em seu estágio atual, é composto basicamente por quatro partes :

+ um modelador de cenas tri-dimensionais, que oferece a possibilidade de uma pré-visualização (visualização rápida e simplificada) de uma cena, considerando suas

características geométricas (informações métricas) e topológicas (informações de relação entre as partes de um objeto);

+ vários sistemas de *rendering*, oferecendo facilidades de modelagem de efeitos de iluminação, distorções, textura. Os algoritmos implementados são *scan-line*, radiosidade e *ray-tracing*;

+ uma interface procedimental a partir da linguagem C;

+ uma abordagem para animação cinemática modelada por computador via descrição formal de roteiros, a qual é o alvo principal desta dissertação.

O sistema modelador, denominado GeoMod (MADE/92), é composto por três partes principais :

+ um editor de primitivas, tendo por função a definição de objetos com superfícies poligonalizadas (esses objetos também possuem uma descrição CSG - Constructive Solid Geometry). Também está prevista a implementação de objetos de forma livre, baseados em *B-splines*;

+ uma câmera sintética, tendo por função a visualização 3D da imagem gerada pelos outros módulos.

+ um compositor de imagens tendo por funções : posicionar, escalar e orientar objetos no WC (*World Coordinate*, coordenadas do mundo real); realizar as operações CSG entre objetos; gerar novas estruturas e faces resultantes das operações CSG; realizar todas as transformações em objetos (aplicação de operações matriciais); exercer o controle sobre a operação dos outros módulos do GeoMod.

Uma vez definida a geometria da cena e acrescentados os parâmetros para sua visualização final, fontes de luz, características das superfícies, grau de transparência dos materiais, etc., inicia-se o processamento final da imagem, através de um dos algoritmos de *rendering* existentes.

O método *Ray-Tracing* está disponível, produzindo imagens de boa qualidade (com efeitos como trasparência e reflexão) porém a sua idéia básica produz uma demanda computacional muito alta. De forma a melhorar o desempenho, está implementada a

otimização de interseção, por envoltória de volume. A implementação em execução pode ser melhor avaliada em BANN/89 e SONE/92.

O método de Radiosidade, sugerido por Goral (GORA/84), utiliza idéias da área de transferência/equilíbrio de energia, para obter a iluminação em um ambiente predominado pela componente difusa de iluminação.

A atual implementação (detalhada em QUEI/91), partiu do trabalho original de Goral, adicionando-lhe a proposta de utilização do Hemi-Cubo, sugerido por Cohen em 1985 (COHE/85), para acelerar o cálculo de *Form-Factors*.

O trabalho em *Scan-line* (HOUN/90 e PRET/91) iniciou-se em conjunto com nossas atividades em animação. A implentação atual está baseada na proposta de Whitted e Cook (WHIT/84), de um algoritmo *scan-line* com *z-buffer*, para o tratamento de objetos definidos por fronteiras (*B-rep*).

As atividades em Texturização de Imagens concentram-se em duas frentes de trabalho. Uma delas preocupa-se com o estudo geral de texturas, procurando estabelecer, do nosso ponto de vista, uma taxonomia na linha do trabalho de Kaufman (KAUF/85), para classificar o estado atual da arte.

A segunda atividade é dedicada ao estudo de Fractais, objetivando sua utilização tanto para a definição de texturas, como para modelagem de objetos de cenas.

Na parte de Visualização e Pós-Processamento, prevê-se o tratamento de *anti-aliasing* espacial por pós-filtragem de forma a ser aplicado indistintamente do *rendering* utilizado. Também prevê-se a implementação de tratamentos para *anti-aliasing* temporal (*motion blur*) e um sistema para editar e reproduzir em tempo real as animações geradas (denominado de *playback*).

Por fim, a parte dedicada à animação prevê suporte aos tratamentos para a animação cinemática e dinâmica do movimento. Esta última é muito útil em simulações e animações que, de outra forma seriam difíceis (ou impossíveis) de descrever. A primeira se utiliza de uma linguagem formal para descrição (o *script*) do comportamento temporal das entidades físicas cinemáticas sendo, justamente, o alvo principal desta dissertação.

1.3 HISTÓRICO DO TRABALHO

Nosso trabalho se iniciou com a necessidade de implementar um sistema de *rendering* que tivesse a característica de produzir imagens de diversas qualidades em relação aos fenômenos óticos (a critério do usuário), que pudesse partir de uma implementação básica simples e fosse adquirindo recursos modularmente (através da implementação de novos módulos para tratar os diversos fenômenos) e que, principalmente, fosse sensivelmente mais rápido que os sistemas até então implementados no ProSim.

Estas características nos levou a escolher a técnica *Scan-line Z_buffer* que, vem recebendo novas peculiaridades (ver PRET/91) desde a sua versão 1.0 de julho de 1990.

Durante a implementação da versão 1.0, detectou-se a necessidade da definição e implementação de um modelo de "câmera sintética" que possibilitasse ao *rendering* a visualização da cena de qualquer ângulo de visão. Este modelo foi então estudado e implementado quando então os testes detectaram a necessidade de definir cenas mais complexas (tamanho, variação e número de objetos).

Daí então foi implementado um modelador de objetos provisório que não visava garantir características de integridade geométrica nem topológicas, nem realizar pré-visualização ou operações geométricas e construtivas como o GeoMod (ver MADE/92) mas que, seguisse o padrão de descrição de objetos definido para este. Na verdade, o modelador provisório era apenas um gerador de casos de teste (objetos) para o *rendering* em questão.

Só quando estavam prontos e testados o modelador provisório, o módulo da câmera e o *rendering* é que se passou a implementar o **TOOKIMA** (a TOOl KIt for Scripting Computer Modeled Animation) que foi especificado concomitantemente ao trabalho de implementação dos itens acima.

Devido em grande parte a esta evolução histórica do nosso trabalho, o TOOKIMA se apresenta hoje relacionado com os modelos poliédricos de objetos (tipos de objetos definidos pelo GeoMod) e integrado ao *rendering Scan-line*.

Ressalte-se, entretanto, que a ordem em que estão dispostos os módulos no texto a seguir não reflete a ordem cronológica que foram implementados (nem especificados).

1.4 ESTRUTURA DA DISSERTAÇÃO

O texto foi organizado de forma a tornar mais didática (e não cronológica) a apresentação dos trabalhos e do tema, procurando separar e ressaltar as características das partes.

No capítulo 2 apresentamos animação como um todo, sob diversos aspectos procurando esclarecer o conceito de animação, suas fases de produção, suas versões convencionais e computadorizadas e efetuar classificações elucidativas a este trabalho.

No capítulo 3 é apresentado o TOOKIMA como uma das abordagens das classificações vistas no capítulo anterior, através da caracterização do seu contexto e convenções utilizadas e, em seguida, detalhando-se a forma adotada para a descrição da animação (uma interface textual, sintática e semanticamente explorada, que denominamos *script*).

No capítulo 4 separamos um sub-item para cada um dos elementos modelados que definem uma cena a ser sintetizada e animada pelo TOOKIMA a saber: os objetos (atores e decorações), a câmera sintética e as fontes de luz.

No capítulo 5 a geração do movimento é abordada através de duas alternativas cinemáticas: as transformações geométricas e a definição de trajetórias e acelerações.

No capítulo 6 o sistema de visualização *Scan-line*, que está integrado ao TOOKIMA, é discutido considerando as suas fases de visibilidade (determinação dos polígonos, e partes destes, que estejam visíveis) e *rendering* (cálculo da tonalização para superfícies curvas, iluminação dos objetos, etc.).

O capítulo 7 é dedicado a conclusão e está dividido em relacionar possíveis direções para trabalhos futuros e tecer os comentários finais sobre o trabalho.

Por fim, o apêndice traz um exemplo de utilização do TOOKIMA na geração de uma animação, mostrando os objetos usados e os detalhes de algumas cenas.

Resumindo, o texto está dividido de forma a responder, capítulo a capítulo, as seguintes questões em relação ao **TOOKIMA** :

1- O que é animação ?

2- O que foi feito e onde se encaixa este trabalho ?

3- Como descrever uma animação com esta ferramenta ?

4- Quais os sujeitos das animações ?

5- Como gerar o movimento ?

6- Como visualizar as cenas (quadros) geradas ?

7- Quais as extensões possíveis ?

8- Como seria um exemplo real ?

CAPÍTULO 2

ANIMAÇÃO : UMA VISÃO GERAL

2.1 INTRODUÇÃO

"Animação é o controle da variação de parâmetros através do tempo e o armazenamento dos resultados desta variação como uma sequência de imagens" (BADL/87, pg. 107).

A animação sempre envolve o tempo, a quarta dimensão, pois o tempo é fundamental para prover indícios adicionais para a percepção e o entendimento.

Este capítulo pretende apresentar a área de animação de diversas formas, utilizando a ferramenta do computador ou não. Discutiremos sobre as fases envolvidas no processo dito "convencional" da produção de uma animação.

Em seguida passaremos a classificar os sistemas computadorizados de animação segundo vários critérios. E, uma vez compreendido o processo convencional e tendo uma breve visão sobre os sistemas computadorizados, explicaremos onde se pode usar o computador em sistemas de animação e, também, apresentaremos quais os principais e/ou mais conhecidos sistemas onde, efetivamente, tem-se usado o computador nas fases da produção de uma animação.

Em um sub-item separado terminamos o capítulo apresentando os principais aspectos relacionados com características sutís e subjetivas denominadas "Princípios da Linguagem da Animação", aspectos estes usados para garantir um bom produto visual, muito mais dependente do profissional animador do que das ferramentas, computadorizadas ou, não.

2.2 ANIMAÇÃO CONVENCIONAL

Apesar da automação, a animação por computador não pode se eximir do planejamento e produção que ocorrem no Desenho Animado. É muito importante este planejamento, tanto que, certos estúdios afirmam que "alguns projetos vêm cuidadosamente planejados pelo cliente, se não, o Departamento de Artes cria este material em consulta ao cliente" (REYN/82, pg. 295).

Assim, abre-se este trabalho com uma apresentação das diversas etapas de produção de uma animação convencional, para clarear e enfatizar o vocabulário usado pelos artistas da área de animação.

Consideraremos **animação convencional** neste trabalho (por falta de um termo que melhor expresse a idéia), o processo de produção de sequências de desenho animado que não se vale da ferramenta computador para o auxílio em nenhuma fase deste processo. Pode-se considerar pois, animação convencional toda a metodologia que deu origem aos desenhos animados, também chamados de *cartoons*.

A perfeita compreensão das etapas de produção de um desenho animado facilitará a percepção de como e onde poderemos ter o computador auxiliando o artista, pois foi historicamente a partir da análise dos trabalhos repetitivos, tediosos e manuais destas diversas etapas que surgiram as primeiras idéias de Animação Auxiliada por Computador (AAC), dando início à grande diversificação que caracteriza a área de Computação de Imagens e Animação.

As etapas de produção de um Desenho Animado podem ser resumidas como mostrado na figura 2.1.

2.2.1 *STORYBOARD*

Storyboard é uma caricatura da animação desejada, isto é, consiste de um número limitado de ilustrações (esboços) e algumas anotações de interesse de tal forma que as cenas principais possam ser retratadas em poucos quadros e apenas nos momentos importantes (THAL/85b, pg. 4).

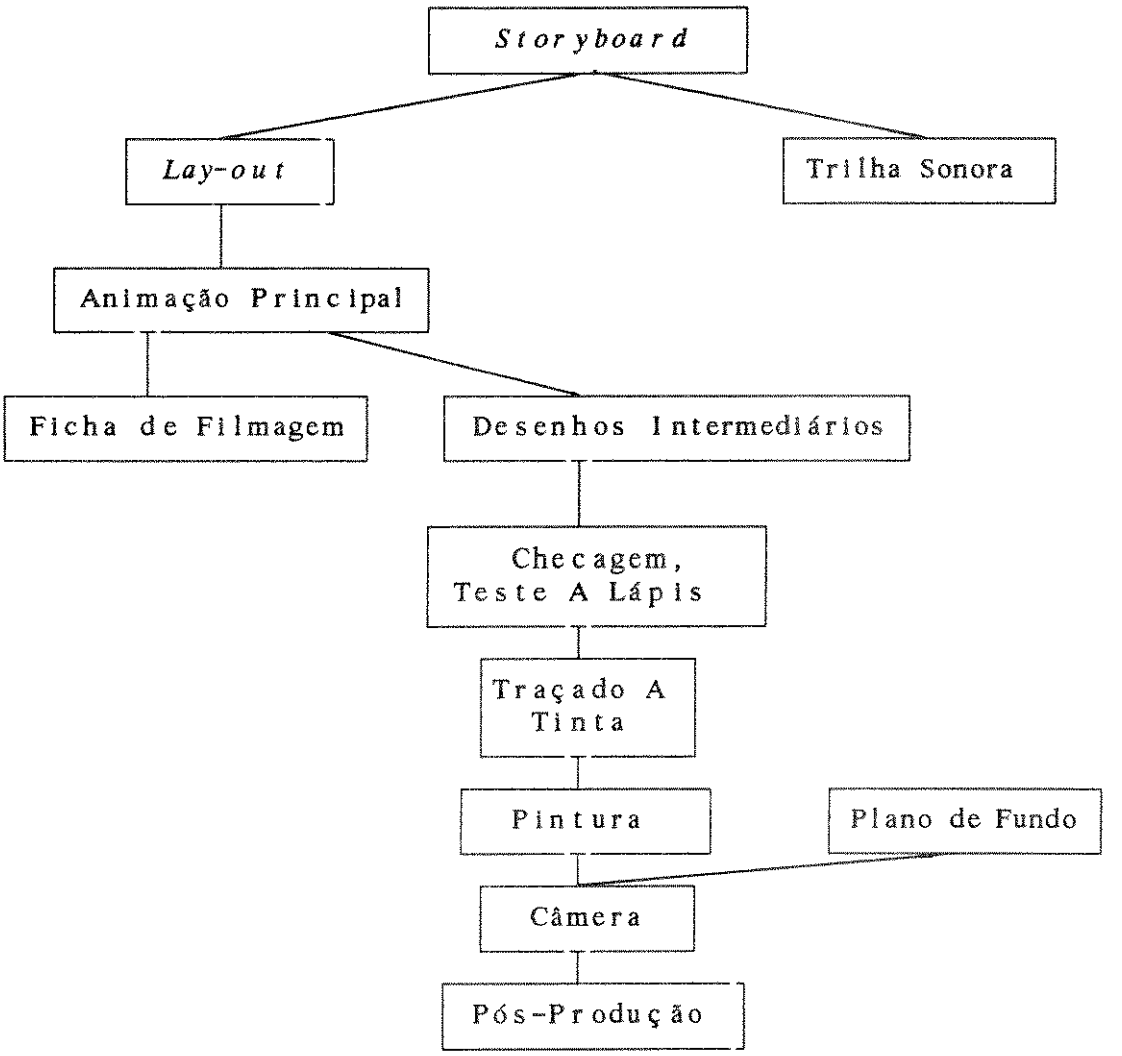


Figura 2.1 : Esquema da Produção de um Desenho Animado.

O *storyboard* deve definir o estilo da animação, o clima, a ação dos personagens chaves e a idéia geral para que ações de objetos secundários possam ser definidas. Neste estágio é aceitável que as ilustrações sejam rabiscos ou esboços com pouco detalhamento.

O *storyboard* tem três finalidades principais :

A primeira, para o produtor, apresentando o primeiro teste visual da idéia.

A segunda, para a equipe de animação, definindo um adiantamento (planejamento) do trabalho que virá pela frente.

E finalmente a terceira, para oferecer ao contratante da animação uma idéia do que ele receberá mais tarde.

O *storyboard* é uma abstração, uma macro visão de como será a animação, por isso ele desencadeia uma série de novas sequências mais detalhadas do filme e estas por sua vez vão originar outras, e assim, toda a animação e suas idéias ficam descritas tanto em desenhos quanto em textos.

Note que este trabalho de criar novas sequências a partir de um ou pouquíssimos esboços requer um exercício de criatividade bastante elaborado. Daí nos grandes estúdios o *storyboard* ser feito por pessoas ou equipes específicas.

2.2.2 *LAY-OUT*

Esta etapa consiste principalmente em projetar o personagem a ser animado. Baseado no *storyboard* definem-se relações entre formas, plano de fundo, cores e tons, tratamento dos planos, definição de efeitos visuais de edição (*fades*, *dissolvers*, *closes*) movimento da câmera (panorâmica, *zoom*, carrinho), etc. É nesta etapa que a personalidade do personagem é definida; ele tem que resumir em si as peculiaridades do tipo ao qual pretende representar.

Nesta etapa são criadas as FOLHAS MODELO (*model sheets*, THAL/85b, pg. 9), que são desenhos dos personagens em vários ângulos (de frente, de cima, de costas, etc.), usados pelos animadores-assistentes (também chamados de *inbetweeners*) para produzirem os quadros (desenhos) intermediários (*inbetweens*).

Uma vez definidas estas características é aconselhável não mudá-las, desde que a facilidade de reprodução é um ítem importante.

No mundo da animação existem padrões pré-estabelecidos de tipos humanos como por exemplo, heróis robustos e fortes, vilões magros e angulosos, tipos alegres que são gordos e de sorriso amplo, heroínas loiras e de grandes olhos, etc.

2.2.3 TRILHA SONORA (*SOUND TRACKING*)

A trilha sonora geralmente é feita antes da animação e serve de inspiração para os desenhos do *lay-out*.

É elaborada por um compositor com o auxílio do pessoal técnico, como diretores, animadores, etc. O compositor deve tomar como base o *storyboard* para fazer seu trabalho.

Nesta fase, também, são escolhidas as vozes dos personagens e os efeitos sonoros.

Logo depois da composição, é feita uma diagramação musical do filme, quadro por quadro, que deve indicar o comprimento e a marcação de cada compasso, também chamada de "minutagem de som" daí então é gerado um documento chamado *bar-sheet* (ver THAL/85b, pg. 9 e BADL/87, pg. 122).

Em certas animações modeladas por computador (ver ítem 2.3.4), é comum a preocupação com a sonorização após a execução do filme. Isto se deve a que, na maioria dos casos, o propósito é validar o ambiente de síntese e animação de imagens, muitas vezes tridimensionais, seguindo certo modelo de controle de movimento, visualização ou sincronização, vindo posteriormente a preocupação com a sonorização.

Isto tem resultado diversas animações geradas por computador, principalmente no ambiente acadêmico, cujas sonorizações têm sido feitas através da busca de uma música, já gravada e difundida, que se encaixe na animação já feita, ou seja, inverte-se a ordem tradicional.

2.2.4 FICHA DE FILMAGEM (*EXPOSURE SHEET*)

Aqui são organizadas progressivamente as folhas de acetato (células) juntamente com o material utilizado para os demais efeitos e componentes, pois a produção de uma animação requer um sistema muito preciso.

Por isso, o diretor, de posse do *lay-out* e da diagramação sonora (*bar-sheet*) preenche as FICHAS DE FILMAGEM (*exposure sheet*), onde estão todos os dados necessários

para auxiliar o cinegrafista na filmagem: incidência da música e diálogo quadro-a-quadro, ordem e disposição das células, número de exposições para uma mesma pose, movimento da câmera, plano de fundo, tratamento dos planos, etc.

2.2.5 ANIMAÇÃO PRINCIPAL E INTERMEDIÁRIA

Nesta etapa os animadores-chefe começam a fazer os quadros chaves (*key-frames*), ou posições principais da animação. São desenhos rápidos que geralmente representam o ponto alto de uma cena. O animador-chefe também faz uma FICHA DE TEMPO, que especifica a posição e movimento dos desenhos intermediários.

Os quadros intermediários são feitos pelos animadores-assistente ou intermediaristas. Com a ajuda da ficha de tempo e das folhas modelo, eles preenchem o espaço entre os quadros principas completando assim o movimento. Todo esse trabalho é feito em folha de papel transparente ou acetato.

2.2.6 TESTE A LÁPIS (*PENCIL TEST*)

Esta é uma etapa fundamental na linha de produção. Os esboços, feitos até então em papel transparente, usando lápis ao invés de tinta, são filmados (geralmente em películas baratas e deixados no negativo) para se ter uma prévia de como ficará a animação final, daí o nome teste a lápis, no sentido de rascunho.

Teste a Lápis, é uma medida de economia pois, permite correções antes que os desenhos sejam transferidos para as folhas de acetato, um processo longo, caro e trabalhoso. Com o computador, o artista pode obter *playback* da cena a medida que a ela vai sendo desenhada ou sintetizada.

O teste a lápis também permite a verificação da qualidade do movimento da animação (se a fluência do desenho está correta ou se serão necessárias mais poses).

Uma facilidade interessante (normalmente não disponível devido ao seu alto preço) é a possibilidade do artista ouvir a trilha sonora correspondente àquele trecho que está sendo visualizado, a fim de que seja alcançado um perfeito sincronismo.

2.2.7 APERFEIÇOAMENTO DE ESBOÇOS E PLANOS DE FUNDO

Os esboços rápidos utilizados no teste a lápis são então limpos e recebem um acabamento final com contornos nítidos e precisos, sendo transferidos usando tinta das folhas de papel transparente para folhas de acetato, processo este denominado de Traçado a Tinta (*Inking*).

O traçado é um estágio relativamente mecânico sendo realizado por um departamento ou grupo especial de artistas do estúdio. Alguns estúdios modernos usam o processo de fotocópia para esse trabalho, o que poupa tempo e dinheiro.

Os planos de fundo (*backgrounds*) também são finalizados, com a escolha de cores e tons que harmonizem com personagens no plano da frente (*foreground*).

2.2.8 PINTURA DOS ACETATOS (*PAINTING* OU *OPACKING*)

As folhas de acetato são então pintadas (daí o termo *PAINTING*) com tinta opaca, no lado oposto àquele em que foram traçados os contornos a tinta, a fim de aceitar que estes se apaguem para possíveis correções.

Este trabalho é feito por pessoas especializadas denominadas de *opackers*, e por isso, é também denominado de *OPACKING*.

2.2.9 FILMAGEM (*SHOOT* ou *FILM TRACKING*)

Antes da filmagem, é feita uma reverificação das folhas de acetato e dos fundos para ver se tudo está em ordem. Começa então o processo da fotografia que é feito pelo cinegrafista com base nos dados da ficha de filmagem (*exposure sheet*) para a geração da trilha de imagem (*film tracking*).

Durante esta etapa são acrescentados os efeitos de câmera como panorâmica,

carrinho ou dupla exposição, dentre outras.

Existem também dois ítens importantes que auxiliam na organização da filmagem: registro e guia de campo.

Para que as folhas de acetato fiquem perfeitamente alinhadas elas possuem um sistema padrão de furos os quais se encaixarão a pequenos pinos localizados na mesa de filmagem (*coumpound table*) e que são chamados de registro. Os desenhos do plano de fundo também possuem o mesmo sistema para que sua movimentação possa ser regulada gradativamente.

O guia de campo nada mais é do que uma folha de papel, subdividida com medidas padronizadas de uma polegada, que é colocada sob a câmera e serve para determinar e localizar qual vai ser o campo de visão da câmera. O campo a ser filmado deve sempre guardar as proporções de 4x3 (comprimento e altura) que corresponde à razão de aspecto dos filmes de super-8, 16 mm e 35 mm.

2.2.10 MONTAGEM DE IMAGEM E SOM

No filme animado, a montagem é a junção final das trilhas de imagem e de som em sequência e sincronismo desejados.

Ao contrário de filmes de ação ao vivo, o desenho animado está quase que completo quando termina a fotografia, podendo restar então, apenas alguns efeitos óticos somente realizáveis em laboratório, minimizando-se o trabalho de montagem após a fotografia.

O processo da montagem pode ser feito, sequência por sequência, à medida em que os quadros vão sendo fotografados.

2.3 ANIMAÇÃO POR COMPUTADOR

Muitas são as abordagens possíveis quando se classificam os sistemas de animação por computador e, um mesmo sistema aglomera diversas características relacionadas a diversos tópicos como qualidade final, controlabilidade do movimento, ambiente de implementação, momento histórico em que surgiu, interação entre ambiente e o objeto animado, modelagem, dimensão do personagem, dentre outras.

Este item apresentará algumas classificações dos sistemas de animação computadorizados, extraidas basicamente de PUEY/88, realçando características peculiares para caracterizar o nosso trabalho (ver caracterização em 3.2.1). Em seguida discutiremos a estruturação dos sistemas computadorizados, identificando áreas ou tarefas dos respectivos sistemas, que são passíveis de computadorização.

Por fim, apresentam-se diversos sistemas, muito usuais e difundidos, que usam o computador e que realizam as tarefas mencionadas antes.

2.3.1 CLASSIFICAÇÕES

Devido à multiplicidade de critérios mencionada acima, nota-se uma certa redundância e repetição de certas características, de uma classificação para outra, o que pode levar a uma complicação e mistificação do tema.

A fim de tornar claro algumas das principais características que melhor representam os sistemas de animação por computador, nos deteremos a apenas três classificações: histórica, quanto ao modo de controle e quanto ao modelo de animação.

Estas classificações são suficientes para situar o TOOKIMA perante a área de animação pois, as consideramos elucidativas e importantes (caso o leitor queira se deter a uma maior variedade ou profundidade nas classificações, sugerimos ver PUEY/88 e THAL/91).

2.3.1.1 CLASSIFICAÇÃO HISTÓRICA

Classificação Histórica
Animação Auxiliada por Computador (AAC) Animação Modelada por Computador (AMC)

Uma análise histórica mostra que, inicialmente, a animação por computador tinha como meta desenvolver processos simples para automatizar algumas tarefas manuais longas e tediosas da produção comercial de filmes de desenho animado: edição, criação dos quadros principais (*key-framing*), intermediação, colorização, etc.

Este tipo de animação foi então denominada **Animação Auxiliada por Computador** (AAC) (também chamada *Image-Based Animation* por autores como BADL/87 e THAL/88, pg. 61). Estes processos são de uso comum na produção comercial de filmes hoje em dia, estando principalmente relacionados com a produção de **desenhos** animados.

De maneira diferente, a **Animação Modelada por Computador** (AMC) propõe modelos de representação das entidades gráficas que compõem a cena (modelo geométrico do objeto, modelo para a câmera sintética, modelo para a iluminação, por exemplo) que o computador deve armazenar e então computar as diferentes ações que podem ser realizadas nestas entidades (isto é, nos modelos).

É importante ressaltar que existem propostas de criar-se modelos para a representação dos objetos em AAC mas, estas são eminentemente representações de desenhos, imagens em duas dimensões, enquanto que, na AMC, a modelagem é feita em três dimensões e depois efetua-se a projeção do modelo para obter-se a imagem em duas dimensões.

Devido a forte relação entre a dimensão do objeto e o tipo de sistema de animação, doravante consideraremos sinônimos os seguintes termos: AAC como Sistemas 2D e AMC como Sistemas 3D.

2.3.1.2 CLASSIFICAÇÃO QUANTO AO MODO DE CONTROLE

Classificação pelo Modo de Controle
Modo Guiado Rastreado (Rotoscopiado) *Key-Framing* Modo Algoritmico Extensões de Linguagens Definição de Novas Linguagens Modo Auto-Descrito

Esta classificação, inspirada nos níveis (*guiding level*, *animator level* e *task level*) de Zeltzer (ZELT/85), enfatizam o modo em que se dá a independência entre o sistema e o operador. O modo de controle pode indicar também, tipos de implementação pois, enfatiza estas características e o modo de entrada/aquisição de dados.

Alguns autores (p. ex. PUEY/88, pg. 283) chamam de **ANIMAÇÃO DE MODO GUIADO** quando o animador deve saber, *a priori*, as diferentes posições que um objeto terá em uma cena a cada momento. A AAC é tipicamente uma tarefa de "guiamento" que, apesar do animador ter um completo controle da cena, animações complexas são difíceis de descrever.

Apesar de proporcionar um controle apurado da cena, para animações de personagens articulados neste modo surge o problema dos "graus de liberdade" (**DOF**, *degrees of freedom*) que impõe ao animador o controle de centenas ou milhares de parâmetros ao mesmo tempo, tornando a descrição muito complexa e enfadonha (ZELT/85, pg. 250). Pode-se realizar a seguinte subclassificação baseada na entrada de dados:

Modo Guiado Rastreado (*Motion Tracking*), também chamado de Rotoscopia, é o modo onde os movimentos reais (que geralmente não são fáceis de descrever analíticamente) são rastreados por sensores apropriados e armazenados em tempo real, para posteriormente, servirem como entrada para a reconstrução do movimento original através da animação por computador.

Uma variante bastante interessante é a seleção e o ajuste fino de valores e instantes feita interativamente através do desenho de curvas na dimensão do tempo (eixo **T**), para cada parâmetro.

Modo Guiado Key-Framing (*Key-Frame Systems*) onde em alguns quadros chaves (importantes) o estado de uma cena é descrito por completo e os quadros intermediários (*Inbetweens*) criados automaticamente através da interpolação entre os quadros chaves (por funções lineares ou interpoladoras *splines*, por exemplo). O resultado da animação não se baseia, necessariamente, em movimentos do mundo real.

Subdividimos o modo guiado *key-framing* em dois: O Baseado em Imagens (*Image-Based Key-framing Animation*) onde os quadros intermediários são obtidos pela interpolação do próprio desenho (seus traços, ver item 2.3.3.1). E, o Paramétrico (*Parametric Key-Framing Animation*, BADL/87, pg. 121), onde os quadros intermediários são obtidos pela interpolação de parâmetros que modelam o desenho (2D) e/ou o objeto (3D).

ANIMAÇÃO DE MODO ALGORÍTMICO, é aquela animação que especifica, através de um conjunto de procedimentos as regras para as transformações e movimentos em uma cena.

Este modo de controle age diretamente sobre a especificação do movimento, explicitando, passo a passo, a evolução ou a posição do mesmo (e não o modo de interação entre atores ou entre ator(es) e o ambiente).

É um método poderoso que permite o desenvolvimento de animações complexas, normalmente, sem restrições quanto ao movimento e, por isso mesmo, requer um intenso desenvolvimento de software pelo animador, o que nos leva a seguinte subclassificação de acordo com a interface de programação, vista pelo animador.

(a) Extensões de Linguagens de Programação Convencionais de uso geral, através do desenvolvimento de bibliotecas de funções gráficas e tipos de dados para animação. São sistemas que interpretam os comandos elementares especificados em um *script* (ou programa, ver item 2.3.2.2.2) pelo animador, que decompõe os movimentos complexos em movimentos elementares, escrevendo e ordenando-os como sentenças de um programa.

É interessante observar, que esta abordagem é tão poderosa e flexível a ponto de subsidiar inclusive o desenvolvimento de outras abordagens, servindo de sustentáculo para estas últimas. Esta idéia foi induzida por Sarachan (SARA/86, pg. 9) quando discute a evolução (ou melhor, as várias facetas) do sistema MIRA, desenvolvido na Universidade de Montreal como uma extensão da linguagem Pascal.

(b) Existem os sistemas a nível de programação que representam o desenvolvimento

de uma Nova Linguagem de Animação onde são construídos comandos que automatizam movimentos padrões e possuem nomes mneumônicos, interfaces padronizadas e voltadas para a aplicação. Enfim, a definição de um novo ambiente de programação específico para animação.

Dependendo do suporte computacional para análise e projeto, a implementação de uma Nova Linguagem para Animação pode ser bastante custosa (em tempo) requerendo exaustivo trabalho de programação de interpretadores, *parser's* ou equivalentes sendo, por vezes, observada a necessidade de características inerentes às linguagens convencionais de uso geral, tipo Pascal, Lisp, C, etc. (ver PICC/88 e VELH/89).

A principal desvantagem das duas abordagens do Modo Algoritmico é que requerem animadores com conhecimento de programação tornando seu uso bastante restrito à maioria dos artistas.

Na **ANIMAÇÃO AUTO-DESCRITA** (também denominada de "animação a nível de tarefa" ou *self-scripting animation*), o computador cria uma base de dados com as informações necessárias sobre o mundo no qual os objetos estão inseridos (leis e restrições físicas, mecânicas, biológicas, nucleares, dentre outras que regem o ambiente). Isto permitirá a visualização, análise, estudo e animação de uma série de fenômenos, sem riscos, perda de corpos de prova, sem perda de clareza, usando-se vários ângulos de visualização, enfim, usufruindo dos diversos recursos da síntese de imagens.

Uma vez modeladas as leis de interação entre os objetos, a animação corre independente e automaticamente (daí a sua denominação de auto descritiva, *self scripting*).

O animador apenas inclui informações adicionais para as inicializações (como posição, velocidade e forças atuantes) e o computador gera os movimentos, por conta própria, seguindo as equações de interação que ele "conhece".

Como pode-se perceber, estes sistemas são especialmente interessantes para *simulações* e para o tratamento de personagens articulados, não tendo neste último caso, a desvantagem do animador ter de controlar **todos** os seus graus de liberdade (DOF).

Estes sistemas estão recebendo atenção especial nos últimos anos porque provém um meio para resolver o problema de interface com usuários: são amigáveis e prontos para uso de pessoas não especializadas (não programadores). Pesquisas neste tópico estão relacionadas com outros campos bastante recentes de investigação, a saber, inteligên-

cia artificial, sistemas especialistas e robótica, dentre outros.

A essência da "animação a nível de tarefa" é o controle automático do movimento. Alguns autores apresentam o conceito de **Animação Comportamental** (*Behavioral Animation*) que tem sido proposto como um nível mais alto de controle (WILH/90 e REYN/87). A animação comportamental avalia o estado do ambiente e gera movimento de acordo com certas regras de "comportamento".

Este casamento entre animação (síntese de imagens) e outras áreas abre novas perspectivas na direção do realismo dos fenômenos usados em animação; e ao mesmo tempo traz um novo grau de dificuldade na simulação de choques, materiais não rígidos (por exemplo, gelatinas), explosões, propagação de ondas mecânicas (ondas do mar), objetos articulados com vários graus de liberdade (robótica), expressões faciais, visualização científica (fluxo hidráulico, reações nucleares e químicas, carregamentos de estruturas estáticas, representação de funções no $\mathbb{R}^n$, etc.)

2.3.1.3 CLASSIFICAÇÃO QUANTO AO MODELO DE ANIMAÇÃO.

O termo "modelo de animação" advém da preocupação com o modelo de interação entre animador e atores (e seu grau de intervenção) e com os modelos de equações e entidades físicas envolvidas na especificação do movimento.

Classificação pelo Modelo de Animação
Modelo Cinemático Modelo Dinâmico

Lembremo-nos da física que, cinemática se detém em estudar a descrição do movimento em si sem se preocupar com a origem deste, tratando as entidades velocidade, espaço percorrido, aceleração e tempo. Já a dinâmica estuda o movimento e suas causas através do estudo de entidades como forças atuantes (atrito, gravidade, torque, p. ex.), energias (cinética, potencial, etc.) e massa.

O **Modelo Cinemático** produz movimento via posicionamento, velocidades e acelerações (*eases*). O sistema mais comum é denominado *Key-Framing*, sendo o método clássico usado principalmente na animação convencional e na animação auxiliada por computador (AAC), onde se especificam posições chaves, velocidades e acelerações entre as posições chaves.

Outro modo de se especificar ações cinematicamente é definir explicitamente as ações elementares a serem tomadas pelas entidades de uma cena como, rotação de alguns graus em torno de um eixo, translação de certa quantidade, etc. Esta forma de especificação supõe um sistema mais sofisticado que o *Key-Framing*, pois deve oferecer uma linguagem que suporte esta tarefa e deve manter disponível todas as transformações possíveis.

O **Modelo Dinâmico** requer que o usuário intervenha apenas inicializando o sistema (com forças e energias, p. ex.) pois, o sistema segue leis físicas e restrições implícitas e previamente definidas que determinarão as interrelações entre os objetos e entre objetos e o ambiente, desta forma determinando tanto a trajetória a ser descrita, quanto as acelerações durante o movimento (estando intimamente relacionado com o modo de controle auto-descrito).

Para a aplicação do modelo dinâmico, o objeto a ser animado deve ser definido (modelado) usando parâmetros mecânicos (tipo de material, elasticidade, massa, juntas, hastes, molas, etc.) (WILH/87).

Comparando os modelos, poderíamos dizer que o modelo dinâmico é útil para descrever movimentos rápidos e complexos que seriam difíceis, e às vezes mesmo impossíveis, de descrever por técnicas cinemáticas. Já o modelo cinemático é menos custoso computacionalmente que o dinâmico e, mais intuitivo de usar em função de que forças de atrito e torques (parâmetros do modelo dinâmico) não são entidades do nosso dia-a-dia como velocidades, acelerações e deslocamentos (entidades cinemáticas).

As classificações sempre intencionam realçar características diferentes entre tratamentos. Procuramos realçar o aspecto cronológico e a grande importância sobre a modelagem do ambiente, na classificação histórica; alguns detalhes de implementação e controle, na classificação quanto ao modo de controle e, os tipos de entidades e modelos físicos que são tratados, na classificação quanto ao modelo de animação.

2.3.2 ONDE ENTRA O COMPUTADOR ?

Analisando os passos de produção de uma animação convencional vê-se que existe muito trabalho repetitivo, manual e entediante, que é uma característica particular deste tipo de ambiente.

Para melhor entendermos onde podemos usar o computador para automatizar o trabalho, adotamos uma divisão dos sistemas de animação em dois grandes blocos, baseados na classificação histórica vista no ítem anterior.

Vamos apresentar, resumidamente, algumas configurações que caracterizam os dois sistemas definidos. Estas configurações capturam o essencial dentro de um processo de produção de uma animação.

2.3.2.1 ANIMAÇÃO AUXILIADA POR COMPUTADOR (AAC)

O papel do computador na animação vem crescendo a cada dia causando certa imprecisão no termo "animação por computador", por causa da variedade de diferentes aplicações (THAL/88).

Nos casos de AAC, também chamados de *key-framing systems*, a aplicação do computador já se estende a quase todas as etapas, desde os desenhos, o traçado à tinta, a pintura, a geração dos quadros intermediários até a tonalização, a texturização e a geração do plano de fundo (MACN/90, pg. 42).

Apesar da técnica dos quadros chaves existir também em sistemas AMC, é muito frequente o uso do termo *key-framing systems* associado a sistemas AAC (isto é, animações iminentemente bidimensionais), por isso, neste texto também consideraremos, daqui para frente, estes dois termos como sinônimos, a menos que seja ressaltado o contrário.

Por uma questão de didática e simplicidade adotaremos os sistemas AAC com a seguinte configuração esquemática:

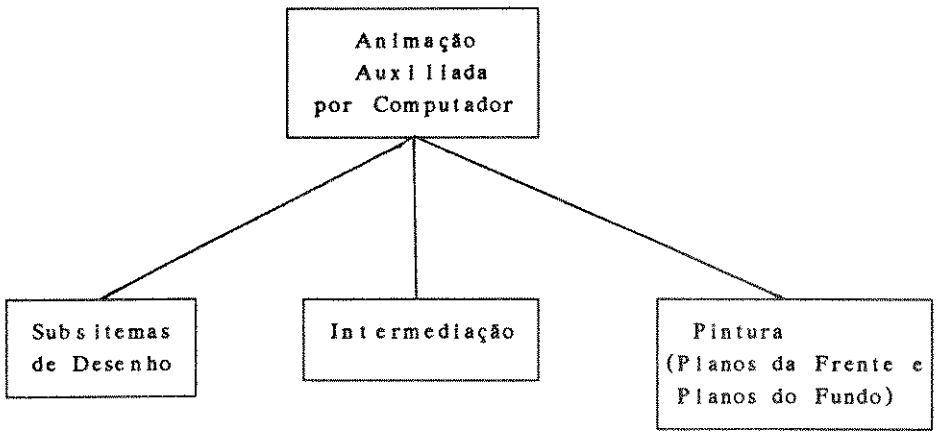


Figura 2.2 : Esquema da Animação Auxiliada por Computador (AAC).

Cada um dos blocos acima é uma etapa na qual o computador pode facilitar muito o trabalho manual. Cabe aqui uma descrição de cada um dos subsistemas acima.

2.3.2.1.1 SUBSISTEMAS DE DESENHO (*DRAWING SYSTEMS*)

Estes sistemas tentam substituir ou simular uma prancheta de desenho através de dispositivos gráficos de entrada como *mouses*, mesas digitalizadoras, *scanners* (THAL/88), etc. e onde a visualização é feita em um terminal de vídeo.

Apesar de um certo receio que ainda impera sobre os artistas, estes subsistemas vem sendo bastante aperfeiçoados com o uso mais intenso de recursos computacionais, passando a dispor de facilidades como criação, salvamento, recuperação, recodificação, apagamento, alteração, etc. de desenhos.

2.3.2.1.2 INTERMEDIAÇÃO (*INBETWEENING*)

Os subsistemas denominados de intermediação são responsáveis pela especificação do movimento (quando da definição dos quadros chaves) e da geração das diversas imagens intermediárias (quando do processo de interpolação).

Esta tarefa é responsável pelo maior trabalho manual e, potencialmente

automatizável, dentro do estudio de animação. Este subsistema tenta substituir os profissionais denominados *inbetweeners* (discutidos nos itens 2.2.2 e 2.2.5).

Um detalhe interessante neste processo e na automação deste, é que existe uma lacuna de informações preenchida pelo trabalho artístico, ou seja, para compor uma imagem intermediária o intermediador humano dispõe de várias perspectivas e posições, além das imagens extremas, para que ele infira sobre elas e componha a imagem necessária, além de que, a percepção da "idéia", ou "enredo", da animação contribui para que ele determine como e quais traços deve introduzir no desenho. Sem contar com o detalhe de que, "muitos dos desenhos são na realidade projeções 2D de imagens 3D na cabeça do animador" (CATM/78).

Note a complexidade de executar a intermediação via computador, sem dispor da capacidade de percepção e inferência sobre a idéia e o enredo disponível no ser humano.

As soluções, métodos e problemas deste subsistema serão discutidos no item 2.3.3.

2.3.2.1.3 SUBSISTEMAS DE PINTURA (*PAINTING SYSTEMS*)

É o subsistema que auxilia a colorização do "acetato eletrônico" (chamemos assim a tela do computador, para gerar uma relação com o material usado na animação convencional) onde se encontram os personagens desenhados pelo animador.

O operador indica que cor cada área deve receber. É óbvio que se deve tomar cuidados para garantir que não se vejam saliências ou cortes na imagem (CATM/78, pg. 349). "O subsistema de pintura é o sistema que oferece uma distinta vantagem em relação às operações manuais" (THAL/85b, pg. 84). Subsistemas de pintura são muito mais que sistemas interativos com programas de preenchimento de área (ver ROGE/85), estes subsistemas visam simular as ferramentas de um pintor".

Tipicamente, um subsistema de pintura é um programa dirigido por *menus*, para pintura de imagens bidimensionais à mão, onde o pintor usa um pincel (caneta, *stylus*, buril). Primeiro, ele escolhe a cor em uma palete, isto é, em um *menu* de cores. Depois o artista seleciona o pincel que pode ter qualquer forma desejada. Daí a pincelada (pintura) é o desenho sucessivo de cópias do pincel na cor e posição indicadas.

2.3.2.2 ANIMAÇÃO MODELADA POR COMPUTADOR (AMC)

Em sistemas tridimensionais a apresentação esquemática de um ambiente pode ser assim representada (THAL/85a, pg. 61):

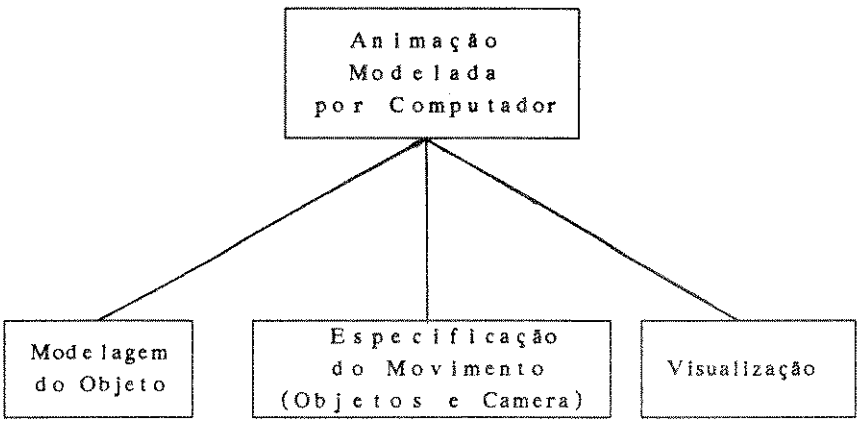


Figura 2.3 : Esquema da Animação Modelada por Computador (AMC).

2.3.2.2.1 SUBSISTEMA DE MODELAGEM

O subsistema de modelagem é composto pelas ferramentas para definição, composição, instanciação e manipulações de primitivas tridimensionais.

Como primitivas entenda-se uma base de dados com características geométricas, geralmente normalizadas, de alguma entidade gráfica como cubo, esfera, cilindro, cone, superfícies de revolução, corpos de extrusão, dentre outros.

Este sistema tem um paralelo com os sistemas de desenho da figura 2.2, não quanto a implementação nem resultados mas, quanto ao objetivo: a criação dos personagens a serem animados. Assim, o sistema de modelagem faz o "desenho dos personagens da AMC".

Este desenho (modelagem) é feito através da criação, agrupamento e manipulação de primitivas (operações estas que permitirão localizar, adicionar, subtrair e intersecionar outras primitivas, por exemplo).

Conforme seja a flexibilidade deste sistema tem-se maior liberdade para criar objetos com curvaturas suaves, peças mecânicas com dimensionamento exato, etc.

2.3.2.2.2 ESPECIFICAÇÃO DO MOVIMENTO (*SCRIPTING*)

No caso da AAC, quando se desenhava um objeto em seus respectivos quadros já estávamos embutindo no desenho a informação de como e qual movimento o personagem estava fazendo, pois "o controle do movimento é embutido, em grandeparte, na sequência de imagens e na temporização das células" (BADL/87, pg. 115).

Agora, nos sistemas AMC, os objetos não precisam ser desenhados (definidos) em novas posições, pois o ambiente permite a visualização em qualquer ângulo de visão, através da simulação de uma câmera sintética móvel, a partir da definição da posição da câmera e da definição da nova posição do objeto.

Entretanto a passagem de uma posição para outra precisa ser descrita segundo uma linguagem que o computador entenda, ou seja, em um *script*.

Esta linguagem espelha os movimentos básicos a que o objeto está sujeito, isto é, são sequências de instruções do tipo escale, rotacione, translade, por exemplo.

2.3.2.2.3 VISUALIZAÇÃO (*RENDERING*)

Em um sistema AAC a pintura é um trabalho artístico que visa passar ao observador os efeitos de iluminação, sombra, rugosidade, reflexão, etc., além de ajudar na percepção de profundidade e fazer parte do apelo que o personagem tenta passar (ficar com o rosto vermelho: enrubecer).

Em sistemas AMC, este trabalho artístico é transferido para o subsistema de visualização (*rendering*) no computador. Este deverá ser capaz de, a partir de informações geométricas e topológicas do objeto, de características físicas visuais (curva de reflectância de Fresnel, por exemplo, GOME/90, pg. 202), da posição do observador e da posição de um modelo de fonte de luz, calcular como, onde, e que cor

receberão as diversas partes do objeto, naquele ângulo de visão.

Este cálculo deve ser possível para quaisquer valores (qualquer superfície, posição, fonte de luz, etc.) e efeitos como sombra, transparência e reflexão também podem estar disponíveis, a menos de um compromisso com o tempo.

2.3.2.3 SISTEMAS DE TRATAMENTO FINAL

O subsistema de tratamento final, que não foi especificado nas figuras 2.2 e 2.3, compreende etapas como teste a lápis, composição, edição e pós-produção. Estas etapas são independentes do modo de geração da imagem, ou seja, dadas várias imagens este subsistema tem tarefas específicas que incidem na imagem em si e não sobre os objetos da cena.

A seguir disporemos mais detalhadamente sobre algumas partes importantes deste subsistema.

2.3.2.3.1 TESTE A LÁPIS

Resume-se em gerar uma pré-visualização com baixa qualidade ou com esboços.

O *playback* é o processo que reproduz uma sequência de imagens prontas com velocidade tal que possa dar a impressão de movimento, no acetato eletrônico (tela do computador). Se as imagens prontas forem imagens dos esboços então estaremos fazendo um teste a lápis mas, se a imagem estiver pronta e com a qualidade final desejada estaremos fazendo uma reprodução simplesmente.

O que pode diferenciar entre os sistemas AAC e AMC é que o segundo pode gerar este teste a partir do modelo geométrico da primitiva (objeto) reproduzindo apenas uma representação aramada (*wire-frame*), não precisando desta maneira gerar efetivamente a imagem, enquanto que nos sistemas AAC, a primitiva é a própria imagem.

2.3.2.3.2 COMPOSIÇÃO

Depois de pintado, antes de entregar o desenho para a filmagem ou armazenamento em vídeo, é interessante dispor-se de facilidade para gerar *zoom*'s, panorâmicas para os componentes da figura além de juntar-se o plano da frente (*foreground*) ao plano de fundo (*background*) para compor a imagem (CATM/78, pg. 349).

No computador isto é feito no *frame-buffer* (uma memória rápida de acesso aleatório que armazena a imagem) que contém, por exemplo, 8 bits para cada componente da cor. "Isto não é óbvio para a maioria das pessoas, mas é uma capacidade importante para se ter boa qualidade nas imagens".

2.3.2.3.3 PÓS PRODUÇÃO E EDIÇÃO

Envolve uma série de operações laboratoriais onde as imagens expostas no filme são reveladas. Editar consiste principalmente em montar, ordenar e cortar o filme. A sincronização do som também é gravada na pós-produção (THAL/85b, pg. 9).

Na etapa de edição empregam-se alguns efeitos de câmera tais como *fade* (aparecimento gradual da imagem a partir do preto ou, desaparecimento para o preto), *wipe* (a sobreposição de uma cena sobre outra por deslocamento), *dissolver* (quando uma cena desaparece e outra aparece em seguida) (THAL/85b, pg. 6 e BADL/87, pg. 224) que são comuns no entrelaçamento de cenas. Este subsistema pode ainda incluir, ferramentas de processamento de imagens (diminuição de definição -mosaicos-, por exemplo) tanto para auxiliar no retoque das imagens quanto para constituirem efeitos especiais.

Uma vez introduzida a classificação histórica adotada e conhecendo-se esquematicamente a composição dos sistemas para AAC e AMC, nos deteremos agora em aprofundar os detalhes funcionais destes dois tratamentos, considerando o funcionamento básico de cada uma, algumas extensões e melhorias bastante difundidas na literatura, bem como mencionando alguns sistemas, originariamente do meio acadêmico, que se tornaram referências comparativas para implementações (acadêmicas e comerciais) posteriores.

2.3.3 SISTEMAS DE ANIMAÇÃO AUXILIADA POR COMPUTADOR (AACs)

2.3.3.1 IDÉIAS BÁSICAS E ALGUMAS LIMITAÇÕES

Muito já foi debatido sobre qual parte da linha de produção da realização de uma animação pode ser melhor beneficiada com a assistência do computador. A resposta depende do estilo desejado de animação, do contexto da produção, recursos disponíveis e vários outros fatores. As técnicas mais largamente usadas desde os priomórdios da AAC são a automação da geração dos quadros intemediários e da pintura (ver item 2.3.2.1.3).

Uma implementação pioneira dos sistemas de intermediação de quadros foi, no final dos anos 60, o sistema GENESYS, desenvolvido por Ron Baecker. Foi um dos primeiros sistemas voltados para desenho onde tanto a imagem quanto a animação eram definidos graficamente (BOOT/83, pg. 45) e realizando apenas animações isométricas (ver ítem 2.3.3.2.1).

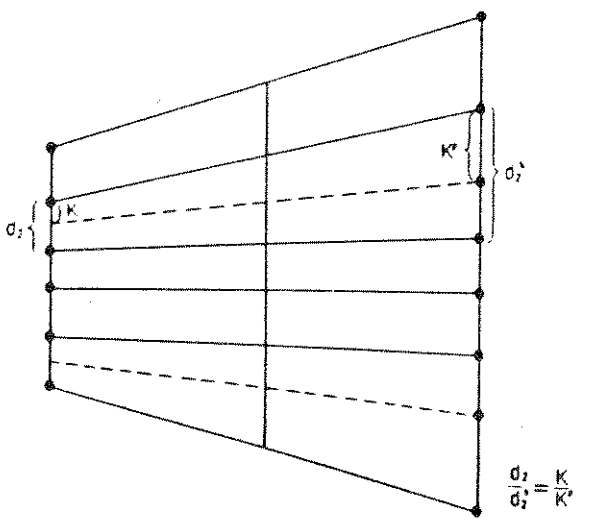
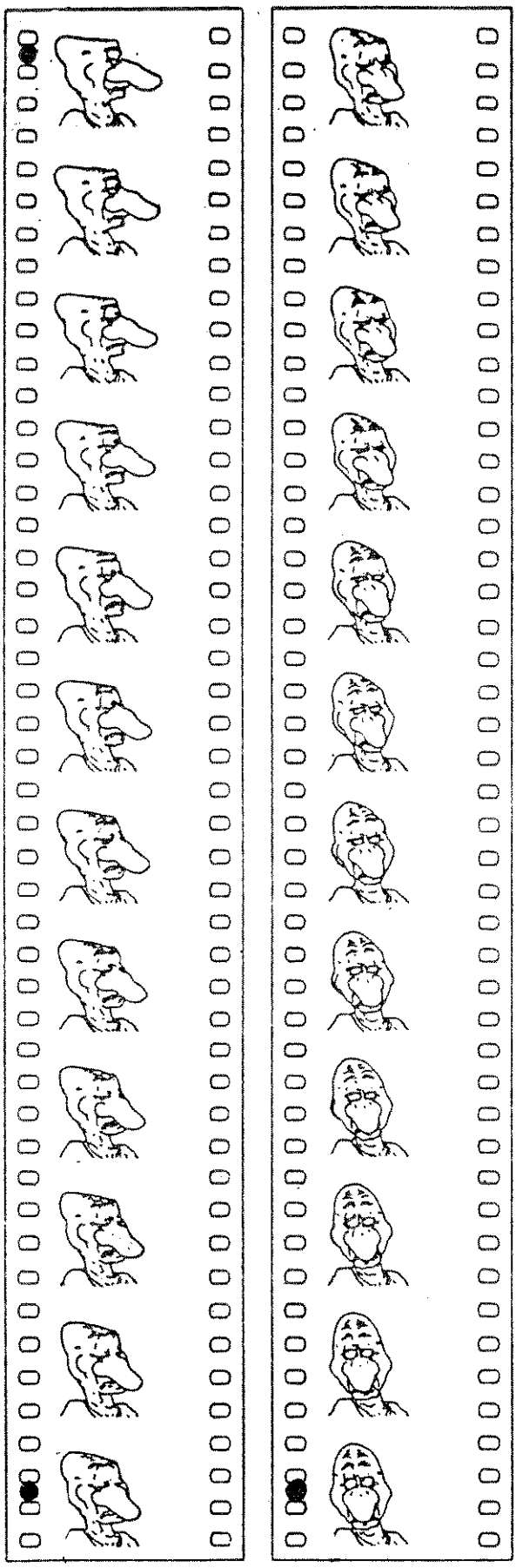


Figura 2.4 : A Técnica do *Inbetweening* (THAL/88).

Basicamente, a técnica de intermediação de desenhos também chamada de *Image Interpolation* ou *cell-oriented animation* (BADL/87, pg. 115) trabalha do seguinte modo: o animador especifica dois desenhos chaves, consecutivos indicando a distância em



quadros entre eles, o computador então, calcula os desenhos adicionais necessários entre eles através do cálculo linear de distâncias entre pontos correspondentes, como na figura 2.4.

Uma proposta de implementação para a intermediação de desenhos foi apresentada por Burtnyk e Wein (BURT/71), a qual passamos a detalhar por ser considerada clássica e eficiente (ver BURT/76, pg. 564).

Os desenhos chaves são decompostos em células ou camadas (Burtnyk explica que célula, ou *cell*, é um termo derivado de *celluloids*, o material transparente no qual os desenhos são originalmente preparados na Animação Convencional, daí o porquê da denominação *cell-oriented animation*).

As partes de um desenho que se movem de maneira diferente, normalmente, são desenhadas em células separadas e, na hora da filmagem, sobrepostas, como um sanduiche, para compor o desenho completo. Estas células são, decompostas em traços (*strokes*, linhas curvas ou arcos do desenho) definidos por uma sequência de segmentos de retas visíveis entre pontos (são armazenados no computador como uma sequência de pontos, BOOT/83, pg. 47). Executa-se então a interpolação entre todos os pontos correspondentes, de todos os traços de todas as células que compõem dois desenhos chaves.

Figura 2.5 : Exemplo do *Inbetweening*. Os pontos pretos na figura indicam os quadros chaves (*key-frame*).

Se o número de traços de uma célula ou, o número de pontos de um determinado traço em um desenho chave, não for igual na célula e traço correspondentes do segundo desenho, então é necessário um pré-processamento para garantir estas duas condições.

Caso estes valores sejam iguais, então executa-se o cálculo da interpolação, diretamente.

Para a determinação da correspondência (qual traço e/ou ponto se transformará em outro no quadro chave seguinte), o animador deverá desenhar os traços da primeira figura em uma certa ordem. Quando o segundo quadro chave é desenhado, a ordem na qual os novos traços são desenhados pode ser usada para a determinação da correspondência entre os dois extremos.

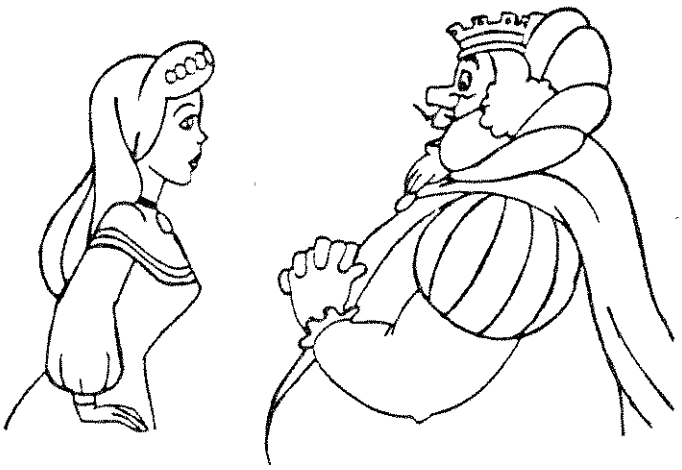
Ou seja, o primeiro traço do primeiro quadro chave será gradualmente transformado no primeiro traço do segundo quadro. Se necessário, traços podem ser agrupados ou divididos. Assim, fica delegado ao animador garantir que o número de traços desenhados em ambos os quadros chaves, para uma determinada célula, sejam idênticos, caso contrário o computador não será capaz de "casar" e interpolar os traços ou terá que proceder um pré-processamento.

Se os dois extremos não forem baseados no mesmo desenho, o sistema de animação necessitará de mais informações e/ou desenhos para gerar um conjunto aceitável de desenhos intermediários (BOOT/83, pg. 46). Veja uma aplicação na figura 2.5.

Uma estratégia típica e vencedora de se introduzir uma ferramenta mais rápida e poderosa em um ambiente tido como conservador, é adotar-se a maneira tradicional de abordar o problema como modelo e melhorá-lo, incluindo a nova ferramenta sem modificar a sequência tradicional de se abordar o trabalho (FOLE/84, pg. 239).

Esta afirmação deixa claro que a solução proposta por Burtnyk e Wein (descrita acima) teve uma ascenção e aceitação bastante grandes dentre os animadores pois, apenas transportava a prancheta de desenho para a mesa digitalizadora, com o acetato eletrônico e um sistema de AAC, daí o animador adquiria ainda, diversas outras ferramentas para manipular os desenhos criados.

Mas, uma das grandes limitações e dificuldades no uso do computador para a execução da intermediação foi mencionado por Catmull (CATM/78, pg. 350) quando este percebeu que executar esta tarefa era mais complicado do que poderia parecer, *a priori*.



(a)

(b)

(c)

Figura 2.6 : Problemas na AAC (CATM/78).

Ele percebeu que para a intermediação, em alguns casos reais, a necessidade da inteligência era indispensável frente à grande quantidade de informações (inclusive subjetivas) necessárias à execução de uma interpolação.

Ele ilustra com um exemplo (figura 2.6): dados dois desenhos a e c, tirados de uma animação real, o animador assistente produziu a figura b. Pode-se perceber que "a principal dificuldade é que o desenho do animador é, na verdade, uma projeção bidimensional de como é visto o personagem tridimensional. Portanto, perde-se informações".

Uma pessoa pode recriar o objeto original pelo desenho porque sabe o que o objeto original é, podendo então facilmente criar, por exemplo, uma linha nova que indique o aparecimento de um braço obscurecido por parte do corpo. Além disso, o animador assistente recebe o *storyboard*, as folhas modelo, o *lay-out*, dentre outras informações, criando a imagem do personagem em sua mente.

Este problema na prática faz com que o computador não encontre correspondência entre traços de duas células, impossibilitando o cálculo da interpolação. Isto pode ser eliminado se o animador desenhar mais quadros chaves em posições onde apareçam novos detalhes de um personagem. Mas, esta solução além de ser subjetiva (pois é preciso determinar com

certa precisão o instante do novo desenho), é custosa visto que o animador é quem terá que desenhar mais um quadro chave que deveria ser calculado pelo computador, automaticamente.

Uma situação típica do problema acima (falta de correspondência entre células) ocorre juntamente com o *temporal aliasing* (ver figuras 7.1). Neste caso é quase que obrigatória a interferência do animador assistente, criando as linhas de velocidade (*speed-lines*), os desenhos intermediários e a repetição de parte dos movimentos.

Em BADL/87, pg. 115, comentam-se algumas limitações da técnica *Image Interpolation*. Nesta técnica a imagem é a primitiva, sendo a interpolação limitada a transformações 2D. Assim, é difícil julgar relações 3D corretamente. O controle do movimento é embutido, em grande parte, na sequência de imagens e na temporização das células.

A aplicação direta da interpolação sobre uma reta entre dois pontos do desenho, em quadros chaves sucessivos (ou seja, interpolação linear no espaço), pode produzir resultados indesejáveis. Isto pode ser melhor observado pelo exemplo clássico da figura 2.7 onde ocorre o indesejado encurtamento do braço do personagem.

"Também, interpolações lineares podem gerar descontinuidades perceptíveis no movimento, tanto na trajetória quanto na aceleração, denominadas *clicks*" (REEV/81, pg. 264). Isto impõe a necessidade de funções interpoladoras, ou aproximadoras que, com a informação de poucas posições (poucos quadros consecutivos, por exemplo) possam interpolar dois pontos através de curvas com algum grau de continuidade matemática maior que 1. Uma solução bastante adequada é o uso de funções *spline* ou *B-spline* por serem curvas de qualquer grau (p. ex. cúbicas) contínuas por partes (THAL/88 pg. 75).

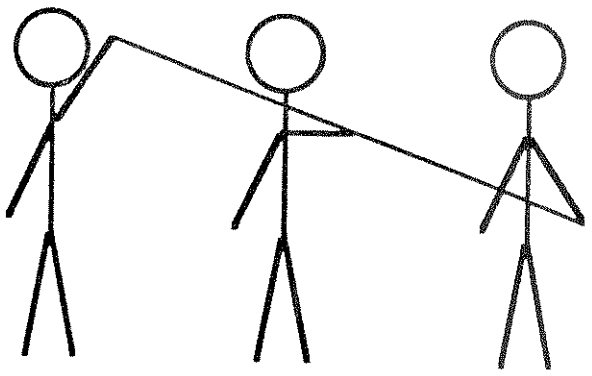


Figura 2.7 : Interpolação Linear e a Flexão de um Braço (THAL/88).

O movimento de um personagem em uma animação parecerá menos com o de um robô à medida que se especifica um movimento não linear no espaço entre quadros chaves, isto é, a linha imaginária que liga os dois pontos correspondentes a interpolar não deve ser uma linha reta e sim uma curva (BOOT/83, pg. 46).

A suavidade e o realismo de um movimento são determinados tanto pela trajetória (modelo de interpolação no espaço) quanto pela aceleração (modelo de interpolação no tempo) (veja discussão sobre trajetória e aceleração no item 5.3).

O que se pode notar é que a proposta básica de Burtnyk e Wein para a intermediação de desenhos não é completa, deixando margem para diversas melhorias em diversas vertentes dentre suas limitações (descrição do movimento, descrição das trajetórias e acelerações, etc.) mas, foi uma proposta que influenciou positivamente os sistemas posteriores, dos quais passaremos a citar alguns.

2.3.3.2 TÉCNICAS E SISTEMAS AVANÇADOS

Vamos discutir neste item algumas proposições de melhoria que incidem sobre a idéia básica de Burtnyk e Wein (BURT/71) para intermediação de quadros chaves da AAC (explorada no item acima) de forma a solucionar algumas das deficiências iniciais da idéia.

2.3.3.2.1 TÉCNICAS QUANTO AOS TIPOS DE TRANSFORMAÇÃO

Todos os movimentos em uma animação podem ser classificados e/ou divididos em componentes isométricas e componentes metamórficas (BOOT/83, pg. 45).

A componente **isométrica** da animação produz mudanças de posicionamento do objeto (ou personagem), que podem ser decompostas em translações e rotações do objeto original.

Genesys é um exemplo de sistema que é primeiramente interessado em facilitar animações isométricas (BOOT/83, pg. 46).

O animador usa uma mesa digitalizadora para desenhar o objeto e o caminho do movimento (*motion path* ou *p_curve*) o qual o objeto deverá seguir. Durante o desenho da *p_curve* a mesa digitalizadora é amostrada em intervalos de tempo constantes de forma a calcular o deslocamento proporcional, de um quadro para outro, que o sistema deverá produzir (BOOT/83 pg. 46). Portanto, *p_curve* é uma técnica bastante natural para especificar tanto a trajetória quanto a aceleração do movimento.

A componente **metamórfica** da animação produz transformações no próprio objeto, tais transformações podem ser conseguidas com escalamentos, distensões e mudanças na aparência do mesmo.

Metamorfoses podem ser obtidas, produzindo *p-curve*'s para cada parte da imagem entretanto, esta solução não produz bons resultados.

A idéia básica de intermediação de Burtnyk e Wein (BURT/71), parte da suposição de que sempre se necessita de uma transformação metamórfica (com efeito, as transformações isométricas podem ser obtidas pelas transformações metamórficas, com bons resultados) entretanto, sempre abordar o problema metamorficamente é mais custoso do que separar as duas transformações.

A animação com transformações misturadas permite o uso indistinto e transparente das duas anteriores. O problema neste caso é a correta separação quando se trata de interpolar imagens, ou a correta descrição das sequências e alternâncias entre transformações isométricas e metamórficas porém, um sistema assim otimizaria o esforço do computador.

2.3.3.2.2 TÉCNICAS QUANTO AOS MODOS DE INTERPOLAÇÃO

Outra classificação importante é quanto ao modo de se interpolar, propriamente dito.

Classificação quanto aos Modos de Interpolação	
Interpolação por Casamento de Imagens	
Interpolação por Caminhos ou Guiada	*p-curve* Esqueleto da Imagem *Moving Points*
Interpolação por Parametrização	Rotoscopia Avaliação de Funções Valores dos Usuários

A essência da intermediação do tipo **Casamento de Imagens** (ou *Matching Interpolation*) é a determinação da correspondência entre quadros chaves (REEV/81, pg. 263). Este tipo de abordagem tem na idéia básica de Burtnyk e Wein (BURT/71), o seu exemplo clássico devido a que existe a busca de casamento entre células para efetivar a correspondência de imagens.

A grande limitação deste tipo de intermediação, resulta do controle incompleto da dinâmica do movimento, tanto em complexidade quanto em suavidade ou continuidade (BURT/76, pg. 565) pois, o movimento de cada ponto da imagem é uma linha reta em direção à posição do seu correspondente, e numa mesma célula a mesma aceleração é aplicada a todos os pontos. Estas limitações podem ser atribuídas ao fato de que existe um único caminho para ser aplicado a toda uma região ou conjunto de pontos ou traços (*strokes*) (THAL/85b, pg. 50).

Nestes sistemas há um dilema em função da própria aplicação da técnica; por um lado consegue-se suavidade maior com poucas imagens chaves (e portanto, bastante espaçadas no tempo) e por outro consegue-se melhor controle com uma maior quantidade de imagens chaves (e, menos espaçadas no tempo). Todavia, uma grande quantidade de ima-

gens próximas no tempo nega muito da vantagem de economia no uso de computadores que é a automatização (REEV/81, pg. 565) porque força uma maior intervenção do animador.

Outra técnica largamente usada, e que vem de encontro com as limitações da técnica anterior, é a **Interpolação por Caminhos** (*Path interpolation* ou, *Movealong interpolation*) (REEV/81 pg. 263) a qual envolve o uso de curvas suavemente desenhadas para controlar o processo de interpolação das imagens, é o caso do sistema Genesys que usa a técnica de *p-curves* (discutida no item 2.3.3.2.1). Entretanto, um único caminho (*path*) para controlar a interpolação é uma solução limitada e satisfatória somente para animações isométricas ou quando as distorções de imagem são mínimas.

Para uma imagem que sofre distorções (metamorfoses), diferentes partes da imagem devem seguir caminhos totalmente distintos, isto é, vários caminhos devem ser definidos para as diversas partes da imagem.

Burtnyk e Wein apresentaram (BURT/76) uma extensão da sua idéia inicial onde incorporam o uso de esqueletos da imagem, dividindo o processo de produção de uma sequência em duas fases: fase interativa de criação das imagens chaves e a fase de determinação de imagens intermediárias pelo esqueleto (quantas forem necessárias).

Esqueleto é uma rede de polígonos que forma uma malha que se sobrepõe ao desenho, são imagens simples compostas por poucos pontos. A imagem é envolvida por esta malha e, uma vez guiadas as transformações sobre a malha isto será estendido para a imagem contida na mesma (em cada um dos polígonos), veja figura 2.8.

"O efeito do controle do esqueleto é o de pegar qualquer área específica do plano de visualização e torcê-la para uma outra área deste plano como se elas tivessem sido feitas sobre uma folha de borracha" (BURT/76, pg. 567).

A idéia por trás da técnica do esqueleto é que, o esqueleto da figura (e não a própria figura) é usado como base da intermediação.

Várias vantagens podem ser atribuídas à esta técnica e sua implementação. Esta técnica é completamente compatível com a idéia básica de Burtnyk e Wein, portanto uma vez determinado um quadro chave pela transformação do esqueleto de uma figura inicial, pode-se aplicar a idéia básica (método de casamento) para interpolar da figura inicial até a figura obtida pelo seu esqueleto.

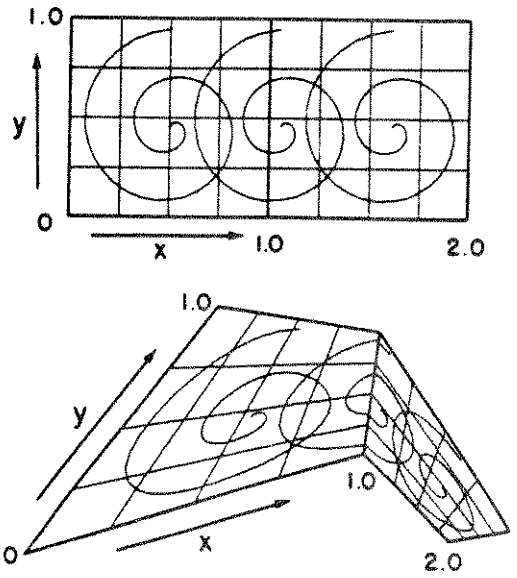


Figura 2.8 : A Técnica do Esqueleto (BURT/76).

O total aproveitamento desta técnica é alcançado quando o animador pode aplicá-la seletivamente, isto é, apenas nos instantes onde ache interessante acrescentar um desenho, então ele cria este desenho pelo esqueleto de uma imagem anterior.

De fato, os autores ponderam que a técnica é mais atrativa se ela pode ser usada para melhorar a aceleração do movimento de uma sequência que já foi previamente criada (esta foi a forma a qual ela foi implementada).

Outra vantagem é a aplicação de funções suavizantes para interpolar os desenhos. Com os quadros chaves inicial e final, desenhados pelo animador, e mais os quadros obtidos pela transformação do esqueleto dos dois primeiros, tem-se informações suficientes para introduzir funções interpoladoras no processo convencional que suavize o resultado final.

Um exemplo de aplicação desta técnica está na figura 2.9 onde, vê-se uma linha central indicando o esqueleto.

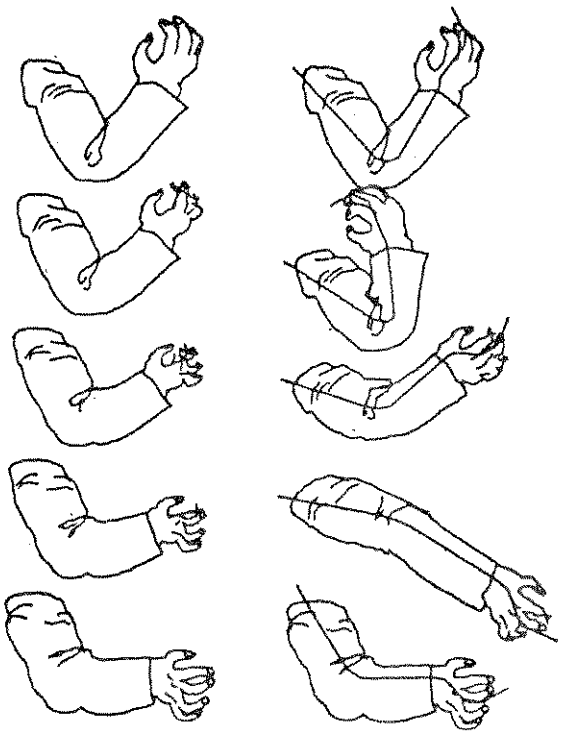


Figura 2.9 : Flexão do Braço Via Técnica do Esqueleto (BURT/76).

Outra abordagem nos sistemas do tipo interpolação por caminhos foi denominada *Moving Points Constraints* com a qual se objetiva melhorar as técnicas até então existentes, através de um mecanismo simples e intuitivo para especificar correspondências, permitindo a especificação de vários caminhos e velocidade de interpolação entre quadros chaves e, permitindo o controle da discontinuidade que, por vezes, ocorre numa sequência (REEV/81 pg. 263).

O princípio desta técnica é associar uma curva variando no tempo e no espaço com algum ponto do objeto animado. Esta curva é chamada *Moving Point* (MP) e controla tanto a trajetória quanto a aceleração do ponto, semelhante à *p_curve* (THAL/85b, pg. 54). Com esta técnica somente os MP precisam ser definidos; os traços e as células nos quadros chaves não precisam ser numeradas nem contadas.

Com um Editor de MP, é estabelecida a ponte que permite a fácil especificação da correspondência entre pontos de sucessivos quadros, a definição da trajetória e da aceleração do movimento.

Curvas MP são normalmente estabelecidas (desenhadas) pelo animador via uma mesa digitalizadora, podendo ter qualquer trajetória e dinâmica. A trajetória é definida

pelo formato que a curva MP adquire. A dinâmica é alcançada através de uma amostragem feita no dispositivo de entrada a intervalos de tempo constantes (daí a semelhança com a *p_curve*).

Estas amostragens podem ser visualizadas como marcas nas curvas MP (ver figura 2.10) e que representam o posicionamento do ponto num instante de tempo.

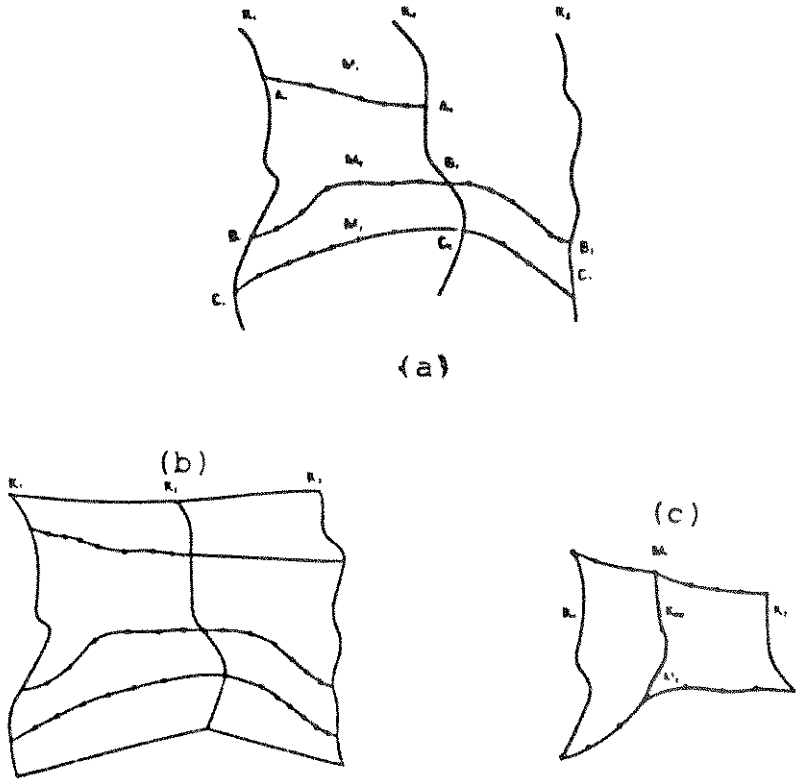


Figura 2.10 : A Técnica do *Moving Points* (REEV/81).

A figura 2.10, mostra três quadros chaves (K1, K2, K3) e três *Moving Points* (M1, M2, M3). A forma da curva especifica a trajetória de interpolação (por exemplo, A1 será transformado em A2 usando o movimento M1). As marcas existentes nas curvas M1, M2 e M3 indicam o tempo, cada duas marca consecutivas representa um espaço de tempo igual (ver THAL/88, pg. 54)

A correspondência entre pontos é automaticamente estabelecida pelo sistema, pela intersecção de duas curvas (dois traços) em diferentes quadros chaves (KF) com um mesmo MP.

Diversos algoritmos podem ser usados para o cálculo das intermediações: algoritmos lineares, cúbicos, etc. Tais algoritmos são aplicados a uma malha completa de *patches* (a figura 2.10.b é a malha completa da figura 2.10.a, obtida por um pré-processamento).

Segundo os autores, o algoritmo de Coons (*B-spline*) foi o que melhor alcançou os critérios de eficiência, generalidade de aplicação, suavidade e economia de especificação (REEV/81, pg. 268).

Na figura 2.11 tem-se uma aplicação desta técnica para a sequência de um braço sendo flexionado onde, foram usados dois quadros chaves e nove MP (como vimos na figura 2.7 na aplicação direta das idéias básicas de Burtnyk e Wein, esta sequência produzia um indesejável encurtamento do braço).

O terceiro tipo de técnica segundo o modo de interpolação, a **Interpolação Parametrizada**, é discutida em detalhe por Badler (BADL/87, pg. 119).

Esta técnica trabalha na estrutura do objeto, não na imagem final necessitando pois, de uma representação para o objeto da imagem. Permite a animação de qualquer elemento parametrizável, como o ângulo do braço de um boneco. Pode-se ter, p. ex., um parâmetro para cada grau de liberdade controlável do objeto.

Uma das grandes vantagens desta técnica é a fonte de valores destes parâmetros, podendo ser supridos por dados externos, obtidos por sensores, dados empíricos ou rotoscopia (BADL/87, MACN/90 e THAL/85b, pg. 45), avaliação computacional de alguma função, ou ainda pelo próprio usuário, interativamente.

Caso não sejam suficientes para todas as posições em todos os quadros, pode-se aplicar um algoritmo de interpolação de pontos, para determinar os valores necessários.

Neste modo parametrizado de interpolação, entretanto, frequentemente necessitam-se de vários parâmetros (várias centenas) tornando a coordenação destes vários parâmetros uma tarefa muito difícil.

Acima dispomos os principais detalhes de várias abordagens, relacionados com o funcionamento dos sistemas de AAC e algumas limitações de suas aplicações, de maneira cronológica e evolutiva.

As técnicas acima citadas são facilmente encontradas nos sistemas AAC atuais, com diversas melhorias e integradas com outros subsistemas (mencionados antes), necessários a este tipo de animação.

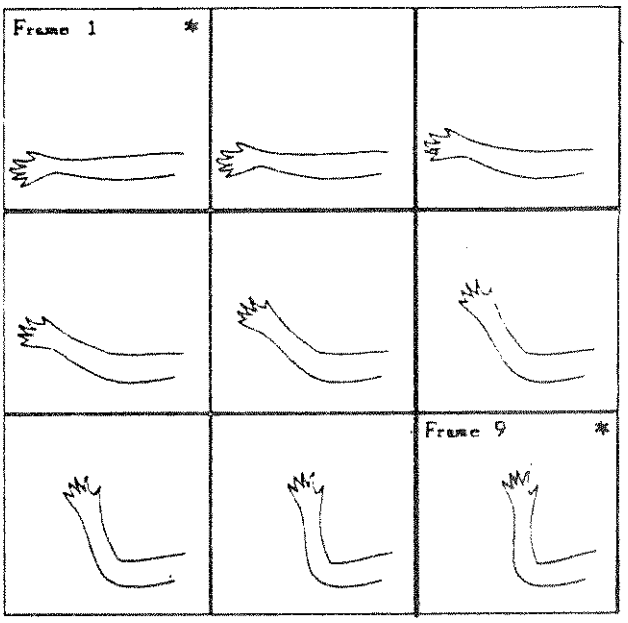


Figura 2.11 : Flexão do Braço Via Técnica do *Moving Points* (REEV/81).

Note por fim que, a descrição e o controle na AAC são fundamentalmente baseados nas intervenções do animador sobre a primitiva "desenho", auxiliado (ou não) por dispositivos periféricos de entrada, tornando pouco formal tanto a descrição quanto o controle da animação.

2.3.4 SISTEMAS DE ANIMAÇÃO MODELADA POR COMPUTADOR (AMCs)

2.3.4.1 IDÉIA BÁSICA

O homem sempre encontrou dificuldades em representar objetos tridimensionais (efetivamente poucas pessoas podem fazer um desenho em perspectiva real em pouco tempo). Daí na animação modelada por computador (AMC), o computador é mais do que um suporte (como na animação auxiliada por computador - AAC) e sim, o responsável pelo principal papel na criação de um mundo virtual (THAL/85a, pg. 69). Algumas espetaculares sequências de animação por computador usam a AMC (BOOT/83 pg. 48).

Apesar de podermos riscar um desenho a mão livre para transmitirmos algumas idéias, às vezes, a necessidade de realismo e/ou precisão obrigam a extensão das pranchetas (que representam um ambiente bidimensional) para o computador e a busca do tridimensional.

Partindo-se pois, para uma representação mais formal (geométrica ou perceptual) do nosso ambiente pode-se visualizar fenômenos físicos, até então apenas analiticamente descritos ou imagináveis na cabeça de poucos, e produzir sequências de animação cujo caráter realista-científico vem aumentando significativamente ao longo dos últimos anos.

Diferentemente da AAC, a AMC propõe introduzir uma completa modelagem do ambiente gráfico tridimensional, impondo restrições construtivas (topológica e geométricas) mas que, ao mesmo tempo, abre possibilidades irrestritas de representar, com realismo, uma série de fenômenos e efeitos.

A descrição e o controle da AMC são baseadas, principalmente, em modelos de descrição das relações temporais e nas técnicas de controle das mudanças de parâmetros, quadro a quadro, que controlam as transformações geométricas (ou não) que criam a dinâmica gráfica.

Quanto à descrição da animação, os sistemas de AMC, são mais flexíveis que os AAC pois, tanto podem seguir uma trajetória bem definida, quanto um comportamento

desconhecido (quando simula leis físicas e a interação destas leis com os objetos e entre os próprios objetos da cena, por exemplo). Tudo isto, ao preço de uma definição formal do objeto (modelagem geométrica) e do movimento destes (*script*).

Quanto à modelagem geométrica, com a flexibilidade que se tem alcançado nos atuais sistemas, uma variedade muito grande de objetos já pode ser representada. Isto inclui objetos de engenharia (peças mecânicas), objetos de forma livre (produzidos por *B-splines*, por exemplo), *soft objects* (cuja descrição são funções potenciais), sistemas de partículas, "modelos procedimentais" (que representam com facilidade a formação de nuvens, florestas, terra e elementos complexos de descrever usando as técnicas anteriores), etc.

Apesar desta grande flexibilidade dos sistemas de modelagem geométrica atuais, algumas peculiaridades chamam a atenção quando se trata de definir uma cena tridimensionalmente.

Na ausência de peso no mundo virtual ou outras limitações físicas, *a priori*, pode parecer mais fácil de construir e manipular os objetos. Esta é uma grande vantagem da AMC (os modelos criados no computador podem fazer manobras complexas sem a necessidade de suportes ou hastes, por exemplo), pois não há esforço físico no posicionamento dos modelos.

A ausência de restrições reais do nosso cotidiano (p. ex. gravidade) e restrições sólidas (integridade geométrica e topológica, inpenetrabilidade, plasticidade, etc.) entretanto, também pode criar alguns problemas (a menos que sejam tomadas precauções, dois objetos podem se interpenetrar indevidamente) a ponto de exigir um explícito, correto e cansativo posicionamento de todos os apêndices de um corpo.

Em função destas possibilidades fascinantes que podem se tornar problemáticas ou trabalhosas, os primeiros sistemas modeladores geométricos encontravam nos animadores experimentados da AAC ou da animação convencional, uma certa resistência pois não estavam habituados a manipular os modelos tridimensionais gerados pelo modelador.

Estas dificuldades encontradas pelos animadores devia-se a que eles tinham que aprender sobre novos conceitos, novas ferramentas, sintaxe, terminologias, procedimento, etc. (KROY/87, pg. 2)

Mais que isto, passar de um ambiente 2D para 3D carrega consigo a tarefa de

pensar e especificar detalhes em uma terceira dimensão (a profundidade) e não apenas criar a ilusão desta dimensão pelos personagens em 2D. O computador "pensa" (atua) não somente em X e Y mas, também em Z.

Kroyer (KROY/87, pg. 3) comenta que os animadores convencionais se acham em frente de um novo problema, a criação de uma imagem desejada em 2D, através de um método de criação, mais complexo, em 3D, isto é, eles pensam na imagem 2D final e não na cena real. Ele apresenta um exemplo tácito de que "esta dimensão adicional da animação por computador obviamente multiplica o esforço do animador na criação do movimento. Em 2D, a ação deve apenas parecer correta para estar correta. Em 3D, a ação tem que ser correta para parecer correta".

A figura 2.12 mostra que, apesar de alguns desenhos serem idênticos em 2D (mostrado em 2.12a), a partir de um ângulo de visão particular a realidade pode ser criticamente diferente em 3D (veja a fig. 2.12b), assim, qualquer outro ângulo da câmera pode revelar a diferença (um erro na maioria das vezes).

Quanto à modelagem da "câmera sintética" na qual os objetos podem ser apresentados como se fossem vistos de qualquer ponto do espaço, pode-se produzir vistas projetivas em perspectiva, paralela ortogonal, etc., enfim simulando-se o processo de visualização (FOLE/90, pg. 237), passando a ser incorporada como uma vantagem e flexibilidade da AMC.

Esta câmera sintética se adapta perfeitamente a aplicações em engenharia e arquitetura onde, partindo-se de um mesmo objeto (uma casa) deseja-se obter diversos ângulos de visão com perspectiva, e com as medidas proporcionalmente exatas.

Movimentos suaves e muito complexos para a câmera podem ser simulados pela mudança do ponto de visualização e do ângulo de visão.

Nestas câmeras não se tem perda de foco, mesmo que o objeto seja grande ou extenso filma-se o monitor 2D e não um objeto sintético 3D. Entretanto, esta ausência de foco pode ser uma desvantagem, pondera Booth (BOOT/83, pg. 50) devido à perda da correspondência com o processo fotográfico convencional (daí, perda de realismo).

Mas, nem tudo está bem estabelecido e estudado em AMC, muitos dos trabalhos desenvolvidos em Computação de Imagens têm-se voltado prioritariamente aos métodos de modelagem da visualização permitindo que diversas soluções já sejam hoje bastante

difundidas (trabalhos com técnicas como *Scan-line* e *Ray-tracer* tiveram seus primórdios no final dos anos 60), e que resolvem uma grande parte dos problemas de visualização enquanto que, algumas técnicas específicas da animação, como a *Motion Blur*, ainda encontram questões sérias a serem solucionadas.

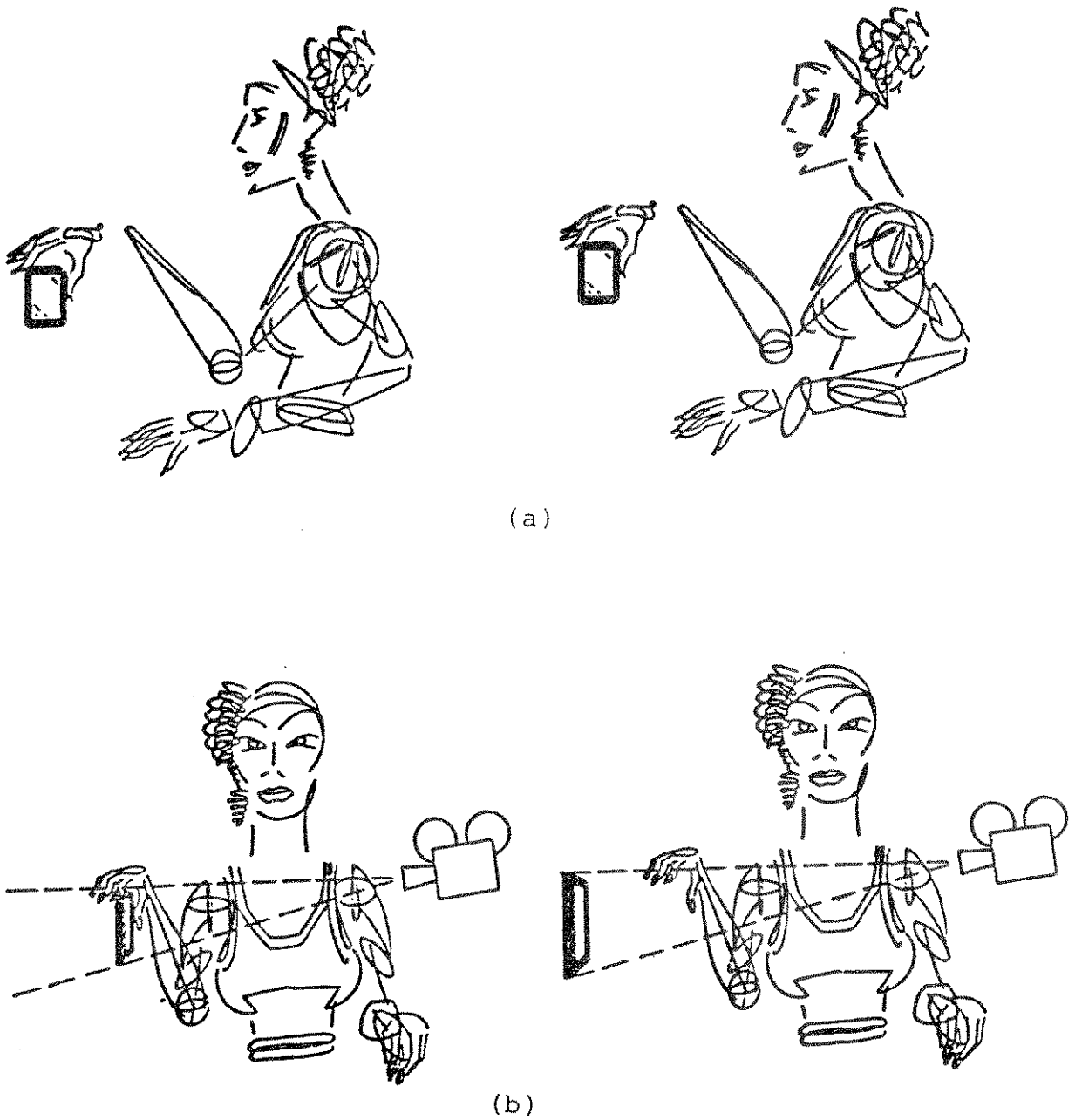


Figura 2.12 : 2D é Diferente de 3D ! (KROY/87).

Um dos primeiros sistemas de AMC, o ASAS (*Actor Scripting and Animation System* - REYN/82), responsável pela primeira sequência de entretenimento veiculada no cinema - partes do filme TRON - e feita com AMC, teve sua versão inicial apresentada em 1975, bastante depois de trabalhos na área de visualização.

O principal fator que tem dificultado o uso da maioria destas técnicas de animação por computador vistas acima (câmera sintética, modelagem geométrica, formalismo na descrição, etc.) é a quantidade de computação necessária para se conseguir resultados de alta qualidade (alto grau de realismo).

Apesar de um perfeito conhecimento sobre a modelagem e sobre a visualização serem de grande valia na produção de uma AMC (principalmente a algorítmica - ver item 2.3.1.2) vamos concentrar nossos comentários nos subsistema de descrição e controle da animação, os *scripting systems*, que por vezes, suas classificações são confundidas com as classificações dos próprios sistemas de animação como um todo, os quais passaremos a discutir algumas técnicas.

2.3.4.2 TÉCNICAS E SISTEMAS AVANÇADOS

O método prático mais usual (segundo BADL/87, pg. 171) para automatizar o processo de animação nos ambientes modelados, 3D, é o uso de *scripts*, que são interpretados de maneira não ambígua pelo sistema de animação, assim como os elementos *bar-sheet* e ficha de filmagem, são interpretados sem ambiguidade no processo convencional.

Uma vez que o *script* seja criado (combinando as especificações de tempo, acelerações e consequências gráficas) o resto do processo de produção da animação é completamente automatizado. Uma consequência disto é que uma vez escrito, testado e aprovado o *script*, todo o trabalho humano termina e então, o resto fica por conta do computador, isto é, a fase de computação intensiva para a visualização, colorização e sequencialização de imagens pode ficar executando durante horas seguidas enquanto a equipe de criação pode se dedicar aos aspectos de um novo trabalho.

Além de liberar a equipe durante a árdua fase da visualização, uma outra vantagem do *script* é a estruturação hierárquica que impõe no tratamento da cena, como na figura 2.1, tal característica nos dá confiança de termos um trabalho em evolução. *Scripting Animation*, como qualquer processo complexo, é normalmente feito em pequenos passos

onde, cada mudança ajuda a convergir para um resultado desejado.

A cada passo, o animador acrescenta um novo aspecto ou modifica algum já existente, sabendo que o resto do *script* (e da animação) continua o mesmo. Se depois do teste, as modificações provocam erro, então estas podem ser reajustadas ou simplesmente removidas para se obter a versão anterior, e correta, do *script*. Caso a mudança seja satisfatória, esta pode passar a ser permanente para aquele *script*.

A vantagem desta abordagem incremental, chamada *tweaking* (REYN/82, pg. 289; BADL/87, pg. 171) é a seletividade que ela impõe durante a busca de possíveis erros. Se uma parte já foi testada e aprovada, é muito improvável que, depois de acrescentada outra parte, ela venha a causar erros. Esta afirmativa só não é válida se as partes interagem mutuamente.

A parte do sistema de AMC que nós iremos discutir daqui em diante é a parte responsável pela descrição e sincronização (ver THAL/85a, pg. 69), este sistema é denominado Sistema de Descrição ou *Scripting Systems*.

Abaixo dicutimos parte da classificação dos sistemas de descrição, segundo o modo de controle da animação, apresentada em BADL/87, pg. 165, discutindo alguns sistemas modelados e aprofundando a classificação apresentada no ítem 2.3.1.2, ao mesmo tempo exemplificamos com alguns sistemas.

Estes sistemas modelados, são do tipo animação cinemática segundo o modelo de animação (ver ítem 2.3.1.3) portanto, todas as ações devem ser planejadas e estabelecidas *a priori*. Os sistemas não incorporam conhecimento suficiente sobre o mundo, de forma a tornar as ações automáticas.

Sistemas de Descrição (*Script*)
Animação Algoritmica Novas Linguagens Extensões de Linguagens Existentes Sistemas Orientados a Objetos

A AMC algoritmica, ou *Programming Aproach*, supõe que o *script* seja o desenvolvimento de uma sequência de comandos que deve ser ordenada e passada ao

computador para que ele opere sobre o objeto modelado.

O desenvolvimento de **Novas Linguagens**, voltadas para o ambiente modelado, gráfico e de animação é uma vertente oriunda dos profissionais de processamento de dados, polarizados pela ânsia de escrever compiladores. Esta solução visa criar uma sintaxe bastante próxima daquela usada pelos profissionais de animação. Seu produto são comandos de alto nível semelhantes aqueles usados no *storyboard* por exemplo: mover, andar, *zoom*, etc. (BADL/87, pg. 123).

Esta técnica fica mais próxima do profissional de animação quanto mais alto for o nível dos comandos, abstraindo-se mais e mais os detalhes de implementação. Entretanto, para manter a flexibilidade é necessário o desenvolvimento de uma grande quantidade de comandos a fim de suprir todas as necessidades do animador.

Isto por sua vez obriga a aprendizagem de um amplo e novo vocabulário e, a medida que este aumenta, mais difícil é a aprendizagem. Se este vocabulário for mantido limitado, limitada também será a capacidade de criação. Obtem-se deste modo um dilema interessante relacionando a flexibilidade e o tamanho do vocabulário.

Por vezes o animador necessita de recursos como declaração de variáveis, laços e outras construções de nível mais baixo, isto representa uma flexibilidade muito grande necessária na criação de novos efeitos e/ou efeitos especiais. Estes recursos tendem a aproximar a nova linguagem das linguagens de programação de uso geral.

Note também que, para oferecer comandos de alto nível em uma nova linguagem, é necessário que se desenvolva uma série de comandos, gráficos no caso, em baixo nível em uma linguagem de programação já existente. Isto é, por trás de um comando novo e abstrato existe um comando (ou um conjunto de comandos) em baixo nível que o implementa de fato. Assim, mesmo antes de se desenvolver um compilador para animação é míster o desenvolvimento convencional de uma série de funções e procedimentos em uma linguagem de desenvolvimento como C, Pascal, Fortran, Lisp, etc.

Se para tal solução (o desenvolvimento de uma nova linguagem para animação), é preciso desenvolver funções de baixo nível, porque não concentrar esforços apenas no desenvolvimento em baixo nível? A resposta está centrada na dificuldade de interação dos ambientes de programação com os animadores tradicionais que, na sua grande maioria, possuem pouco ou nenhum conhecimento em programação (e às vezes, se recusam a aprender - com razão).

Apesar da grande desvantagem acima citada, **Extensões de Linguagens** de propósitos gerais já existentes constituem-se, de fato, mais uma das alternativas factível para os Sistemas de Descrição.

Neste nível de desenvolvimento é mais enfática a necessidade do conhecimento profundo quanto ao resto do sistema (o modelamento e a visualização) pois, "tudo que pode ser modelado, deve poder ser animado".

Nota-se uma tendência crescente nesta vertente pela flexibilidade facilmente alcançável, comparada às Novas Linguagens. Com efeito, é esta abordagem que tem suprido as telas de cinema com espetaculares sequências de AMC.

Daí passa-se a ter preocupações especiais na melhor definição dos componentes do sistema gráfico: estruturas e tipos de dados; interfaces com funções e entre funções; modularização e hierarquização; padronização; metodologias de desenvolvimento e programação; etc..

Segundo SARA/86, pg. 8, a extensão de linguagens de propósito geral é bastante conveniente para construir uma animação. Além das construções padrões de linguagem, aprimora-se esta com tipos de dados extras para representar estruturas geométricas e construções especiais para animação, que alteram parâmetros como uma função do tempo.

Pesquisas em psicologia cognitiva indicam que os humanos podem manipular 7 "coisas" distintas ao mesmo tempo, não mais (BADL/87, pg. 200). Por isso, a melhor maneira para abordar a necessidade de centenas ou milhares de sequências numa animação é o uso da idéia de sub-rotinas (módulos ou procedimentos) ou elas próprias, para dividir a massa de sequências em vários grupos menores mutuamente relacionados e, se este grupo ainda for grande, repete-se o processo sucessivamente.

Este procedimento deve ser aplicável ao *script*, tornando-o multinível onde, os detalhes são postergados a níveis mais baixos e o *script* principal passa a ter um formato mais abstrato sendo pois, uma boa documentação .

Um dos caminhos de desenvolvimento do Sistema de Descrição de boa eficiência é:

1) Desenvolver ambiente de síntese de imagens estática.
2) Introduzir relações temporais para manipular com o ambiente, em baixo nível.

3) Elevar o nível da descrição gerando uma linguagem dedicada às animações.

4) Desenvolver interfaces Homem-Máquina que permitam o aumento da eficiência do usuário na geração de sequências.

5) Embutir regras de (interrel)ações para os atores.

Vários sistemas começaram e se estabeleceram pela alternativa de estender um ambiente computacional já existente: MIRA, com a linguagem Pascal; ASAS, com a linguagem LISP; SCRIPTS, com a linguagem C; etc., respectivamente comentados a seguir.

O caminho trilhado pelo grupo da "Universität del Montréal" no desenvolvimento do sistema **MIRA** passou pelas seguintes fases MIRA-2D, MIRA-3D, MIRA SHADING, CINEMIRA, MIRANIM.

Uma das primeiras preocupações do ambiente MIRA foi a definição dos dados abstratos para atores e câmeras (ver THAL/83).

A base do sistema MIRA é MIRA-3D, uma linguagem gráfica tridimensional que permite a definição de figuras a partir do desenho de linhas ou especificação de superfícies, e é uma extensão da anterior MIRA-2D (SARA/86, pg. 9).

MIRA SHADING é a aplicação de técnicas de *rendering* e *shading* sobre o MIRA_3D. CINEMIRA foi acrescentado posteriormente. A principal contribuição de CINEMIRA foi permitir a definição de "tipos básicos animados". Estes são valores escalares ou vetorais que variam com o tempo (animar usando MIRA-3D sem CINEMIRA requer a construção explícita de laços).

MIRANIM é uma apresentação do MIRA, mais orientado ao animador. MIRANIM é voltado aos usuários que não são capazes de escrever programas de computador.

Usar MIRANIM ainda é um processo de programar, entretanto o uso de objetos gráficos pré-definidos e tipos básicos animados permite que os programas sejam uma simples e direta listagem de comandos.

O **ASAS**, por sua vez, "é uma notação procedimental, uma linguagem de programação para animação e gráficos" (REYN/82, pg. 289). "ASAS pode ser considerado tanto como uma implementação em Lisp quanto uma extensão desta linguagem". Pois, "unindo o soft-ware ASAS ao interpretador Lisp teremos o "Interpretador ASAS" (REYN/82, pg. 290).

ASAS introduziu a entidade ATOR, um número ou participante simulado da animação que pode controlar um ou mais aspectos de uma sequência.

A idéia por trás da estrutura ATOR está em modularizar e localizar o código relacionado a um determinado aspecto, isolando-o do código não relacionado a ele.

ATOR no ASAS é, formalmente, um processo computacional independente em um sistema não hierárquico com ativação sincronizada, capaz de comunicar-se com outro ATOR através da troca de mensagens e processar sua própria animação (THAL/85a, pg. 65). Isto identifica um certo grau de concorrência entre a execução do código referente a cada ATOR (REYN/82, pg. 293).

Outro exemplo de sistema de controle do modo algorítmico é o **SCRIPTS** (VELH/89) desenvolvido no MIT (*Massachusets Intitute of Technology*) que se vale das construções e sintaxe da linguagem C para subsidiar sua implementação. Ele incorpora os conceitos de ATOR do ASAS mas, tem o intuito de ser mais expressivo para animações dinâmicas além de incorporar o tratamento de Eventos.

Eventos no SCRIPTS são instâncias do *script* em tempo de execução que modelam suas propriedades temporais e algoritmicas, sendo composto por um corpo de instruções e uma memória privada (que incorpora inclusive um relógio local) e cada evento pode disparar outros eventos.

Ativar um evento, no SCRIPTS, consiste em atualizar seu tempo local, avaliar todas as suas expressões e suas ativações recursivas.

Também, SCRIPTS incorpora uma estrutura chamada *Track* que armazena os elementos necessários para descrever parâmetros variáveis no tempo. O valor de um determinado parâmetro em um determinado instante é acessado através de uma função, normalmente de interpolação.

A abstração cada vez maior, a introdução de conceitos como "ator", a necessidade de modularização e localização de código relacionado a um ator, a herança de características de uma certa classe de atores, a extensibilidade e a comunicação entre atores, dentre outras, têm na programação orientada a objetos uma boa representação. Daí, algumas inplementações tentarem se valer destas características e promover um casamento com a computação de imagens, notadamente a AMC, gerando o que já se denomina de

Sistemas Orientados a Objetos para animação.

O sistema **CLOCKWORKS** (BREE/87) é um destes sistemas. Implementado em linguagem C convencional, nele foi primeiramente desenvolvida uma metodologia de programação orientada a objetos no ambiente convencional para satisfazer os requisitos desta técnica.

O CLOCKWORKS é definido como uma coleção de objetos que se comunicam com o usuário e entre si, através de mensagens.

As facilidades da programação orientada a objetos estendem-se desde a engenharia do software (manutenabilidade, extensibilidade e depurações facilitadas) até a animação (a troca de mensagens é encarada com naturalidade no *script*), segundo os autores.

Como característica marcante do CLOCKWORKS ressalta-se que, devido a linguagem de modelamento ser a mesma do *script*, "qualquer coisa que possa ser modelada, pode ser animada onde o *script* nada mais é do que, uma série de mensagens entregues ordenadamente aos objetos" (BREE/87, pg. 278).

Muito se discutiu sobre ambas as técnicas (AAC e AMC) que desfrutam da ferramenta do computador mas, as duas vertentes produzem resultados visuais diferentes. Como determinar qual das duas é a melhor ? Cabe aqui então uma sucinta e despretenciosa comparação entre AAC e AMC.

MacNicol (MACN/90, pg. 42), por exemplo, defende a AAC enumerando algumas de suas qualidades frente à AMC: a velocidade de interpolação; a facilidade de uso, dado que novos sistemas gráficos vêm sendo propostos a artistas sem nenhum conhecimento na área de computação; alta liberdade de criação, não fica restrito pelos padrões ou limitações de integridade comumente encontrados nos sistemas de modelamento 3D; serve para a prototipação rápida de idéias visto que, a animação é executável mesmo com o esboço dos personagens (sem detalhes excessivos) que o sistema os considera como desenhos fins.

Uma das grandes vantagens da AAC enunciada por Macnicol, à pg. 44, é a facilidade e realismo que se alcança na representação de movimentos vivos e/ou humanos, sendo que

em AMC, este tema tem levado muito esforço, requerendo detalhamentos cinemáticos e/ou dinâmicos que envolvem conceituação e conhecimento de outras áreas como a engenharia e a física.

Na dúvida entre escolher um sistema AAC ou AMC. Macnicol adverte (MACN/90, pg. 49) que "as pessoas que desenvolvem sistemas AAC para fins gerais estão centrando seu foco prioritariamente na animação, não na computação, na representação do movimento, não no realismo da imagem. Muitas destas pessoas vieram de ambientes tradicionais de filmagem e animação onde computadores são meramente ferramentas e não brinquedos".

Como características marcantes da AMC, advertidas também por Macnicol, e que podem determinar a aquisição destes sistemas, destacamos as aplicações cuja necessidade seja o foto-realismo e/ou onde o interesse seja um objeto a ser visto de vários ângulos (aquí é indispensável o recurso da câmera sintética) que pode vir concomitantemente a outras necessidades como, simulações, visualizações científicas e animações dinâmicas.

O resultado visual final dos sistemas de AAC e AMC são significativamente diferentes, a primeira é iminentemente bidimensional e oriunda da automatização do desenho animado (*cartoon-like*) e a última vem adquirindo um apelo realista cada vez maior com o passar dos anos. Assim, cada qual tem sua linguagem visual própria e, dentro de cada vertente, pode-se até vislumbrar diferentes sublinguagens visuais que não entraremos no mérito pois foge do escopo deste texto.

O que o mercado tem mostrado é que não existe predominância entre um tipo ou outro de sistema, existem implantados quantidades iguais destes sistemas (dados do autor até 1990).

Mas, o que se conclui é que a escolha entre um sistema ou outro está centrada na aplicação a que se destina, ou seja, parâmetros como flexibilidade, velocidade, realismo, tipos de personagens, precisão, necessidade de várias vistas, enfim, são determinantes da aplicação que incidirão sobre a escolha do sistema.

2.4 ALÉM DO AMBIENTE COMPUTACIONAL

Desde o princípio da animação convencional, existe uma preocupação em estabelecer regras e estilos condizentes com a estética e com a mídia da animação. Estes elementos, muito particulares a cada artista, vem sendo difundidos de forma a criar uma proposta de comunicação relativa às técnicas de animação, portanto particular a este meio.

Este conjunto de estilos, denominado de **linguagem da animação**, relaciona alguns princípios e recursos subjetivos, que estão além dos recursos tecnológicos da mídia (estão num nível diferente das preocupações quanto aos equipamentos, quanto ao material usado, quanto a qualidade da revelação, etc.), estando relacionados com o sentimento, o apelo visual, com a necessidade de cativar a atenção e agradar o espectador.

Estas técnicas ou princípios nasceram da observação criteriosa e constante dos filmes de ação ao vivo, dando assim, um novo brilho de sofisticação e realismo à animação e podendo ser aplicadas tanto a AAC quanto à AMC.

John Lasseter (LASS/87) adaptou para animação 3D os princípios tradicionais de animação apresentados por Frank Thomas e Ollie Johnston na excelente obra "Disney Animation: The Ilusion of Life" (THOM/81, pg. 47-69).

Estes princípios são discutidos abaixo, no intuito de mostrar que existe muita técnica e filosofia, além do computador, que devem ser considerados para alcançar uma boa animação (principalmente no sentido do entretenimento). Isto deixa claro que o computador não fará um bom filme sozinho, somente com a intervenção de um bom profissional de animação (*film-maker*) poderá o computador alcançar um bom produto visual.

2.4.1 DISTENSÃO E ESMAGAMENTO (*STRETCH* E *SQUASH*)

Distorção da forma do objeto ao longo do seu trajeto. Estes princípios são aplicados na animação convencional para dar maior expressividade e verossimilhança à

ação. Excetuando-se objetos rígidos (rocha, vidro, madeira, etc.) qualquer outro elemento feito de material maleável ou articulável (carne, borracha, etc.) irá sofrer alterações em sua forma original a medida em que ele executa uma ação.

Esmagamento (*squash*) consiste basicamente no achatamento do objeto seja por pressões externas ou pelo próprio peso e densidade. Por exemplo, uma bola é deformada ao chocar-se com o chão. Isso dá credibilidade ao impacto, a maleabilidade da bola contra a dureza do solo.

Distensão (*stretch*) é o alongamento do objeto, às vezes em seu máximo comprimento, na direção do seu movimento. Por exemplo, uma bola se alonga na sua máxima extensão.

O grau de exagero da distensão e do esmagamento aplicado no objeto coincide e traduz a constituição do mesmo: quanto mais maleabilidade mais acentuado será o esmagamento e a distensão. Veja o exemplo da aplicação destes recursos sobre uma bola pulando sobre uma superfície na figura 2.13.

Podem ser obtidos em AMC através do escalamento da figura. Quando o objeto sofrer um aumento de escala no eixo Z por consequência ele sofrerá redução no eixo X e Y. Esta compensação de escalas é fundamental para manter o "volume" do objeto, pois do contrário (se não houvesse nenhuma compensação), teríamos a impressão de que o objeto inchou ou cresceu de tamanho.

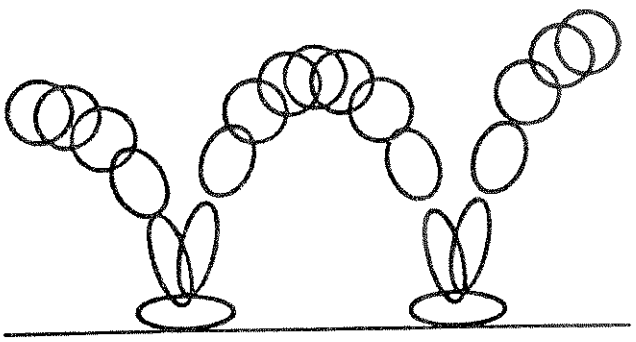


Figura 2.13 : Distensão e Esmagamento (LASS/87).

Estas técnicas podem também, ser usados para evitar um efeito indesejado chamado *strobing* quando não se dispõe de recursos de *motion blur* (ver item 7.3.1). Na medida em que um objeto aumenta sua velocidade obrigatoriamente aumenta também o espaço entre

cada posição do desenho.

Estes espaços vazios começam então a ser percebidos pelo olho humano e a ação adquire um aspecto seccionado. Para que a ação flua suave e uniformemente os desenhos são então distendidos até que suas extremidades se interseccionem com o desenho do quadro anterior, assim sucessivamente, preenchendo o espaço vazio (ver figura 2.14).

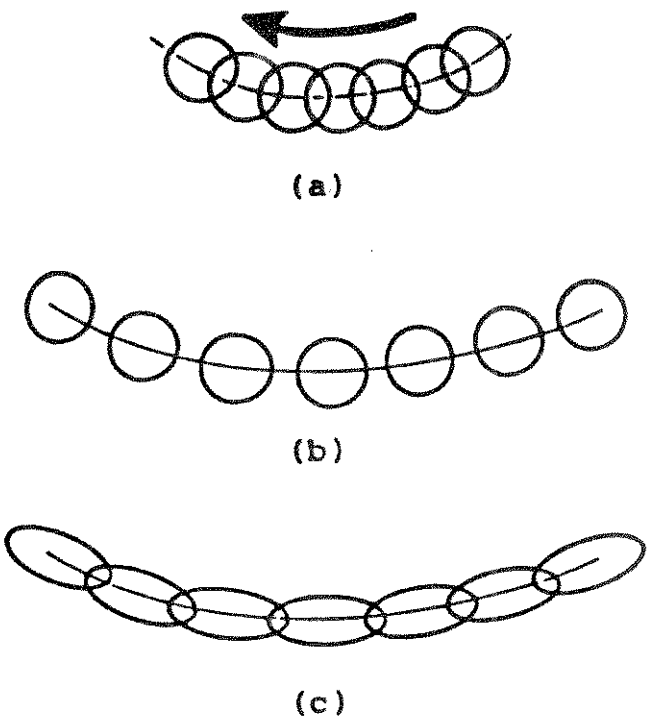


Figura 2.14 : Distensão, Esmagamento e *Strobing* (LASS/87). a) Movimentação lenta; b) *Strobing* em um movimento rápido; c) Uso da distensão para evitar o *Strobing*.

2.4.2 TEMPORIZAÇÃO (*TIMING*)

Temporização é a marcação do tempo e da velocidade de uma ação. Para muitos a temporização é tão importante quanto a própria representação estética do objeto. Um gigante não parecerá um gigante se não agir como tal. Ele tem mais peso, massa e inércia que outro homem portanto, move-se mais lentamente e, uma vez em movimento é difícil mudar este estado.

A temporização na animação poderá definir o tamanho, o peso (objetos perfeitamente iguais podem ter pesos diferentes com diferentes temporizações) e até mesmo estado emocional. Uma mesma sequência de desenhos poderá ter diferentes conotações

psicológicas pela aplicação de diferentes temporizações (diferentes quantidades de quadros intermediários).

2.4.3 ANTECIPAÇÃO (*ANTECIPATION*)

Antecipação é a execução de uma "ação ligeira", oposta à ação principal. Basicamente usada para satisfazer três aspectos da animação :

1- a preparação (da platéia) para a ação que ocorrerá em seguida;

2- a antecipação natural que o próprio corpo dá ao iniciar um movimento (um certo *swing*, molejo);

3- para direcionar a atenção da platéia para um determinado lugar da cena.

Com certeza, todos nós já vimos em desenhos animados algumas cenas fantásticas de quando o personagem, após ultrapassar a beira de um precipício, congela em pleno ar, dá um "tchauzinho" para depois então despencar vertiginosamente no abismo, que parece ter 5 mil metros de altitude. Essa política de transgressão das leis físicas, seguidas também pelo exagero são conceitos plenamente aceitos na linguagem da animação.

Quanto mais rápida, inesperada e violenta a ação mais exagerada será sua antecipação, deixando bem claro o que irá acontecer.

2.4.4 *STAGING*

É a arte de apresentar uma ação no lugar certo e na hora certa e com a clareza necessária. O olho tem que se dirigir para o lugar exato onde ocorrerá a ação. Se "empanturrarmos" a tela com várias ações todas acontecendo ao mesmo tempo, a atenção do espectador irá vagar e a ação principal será ofuscada.

Portanto, é desejável que a ação principal contraste do resto da cena e que sua performance seja convincente para o público. Em resumo, uma ação tem que ser entendida, uma personalidade marcante, uma expressão (corporal ou facial) vista, e um "clima" tem que comover.

Um auxilio importante neste caminho é o uso da "silueta" como recurso para garantir o *staging* onde, procura-se clarear a ação planejando-a como se só fosse possivel apresentar a sua silueta. Esta técnica oriunda dos tempos do desenho somente em preto e branco, afirma que é sempre melhor planejar uma ação em silueta para mostrá-la inclusive em mídias mais sofisticadas.

2.4.5 *FOLLOW THROUGH*

Follow through é o processo de encerramento de uma ação. Um corpo que está em movimento nunca é acometido de uma súbita e completa parada, ela é feita através de vários estágios. Este princípio é o equivalente à antecipação mas, aplicado ao final de um movimento.

Assim como numa locomotiva haverá sempre uma primeira engrenagem que terminará (ou iniciará) um movimento (chamemos de "líder") e se manterá nessa dianteira para qualquer mudança de percurso. Esse líder é seguido (*follow through*) pelo resto do corpo.

Quando o líder desacelera e pára, alguns de seus membros ou apêndices por inércia irão continuar o movimento e só pararão num segundo (ou terceiro, quarto, etc.) tempo após o corpo principal.

Esse processo também pode explicitar a densidade e peso dos apêndices, pois componentes leves terão menor tendência em continuar o movimento e se arrastarão por menos tempo atrás do "líder", o contrário acontecendo para corpos pesados.

2.4.6 INTERSEÇÃO ENTRE AÇÕES (*OVERLAPPING ACTION*)

Uma ação não precisa caminhar para uma parada completa, para que uma nova ação se inicie. Elas devem fundir-se umas nas outras e criar uma sensação de continuidade para os olhos.

Quando um personagem se levanta para atender a porta ele não precisa parar entre cada estágio do trajeto para pensar no que fazer em seguida. Ele simplesmente se

levanta, a perna já vai na dianteira para iniciar a caminhada, quando está próximo à porta (sem interromper a caminhada) a mão se estende para a maçaneta e, antes que a porta esteja totalmente aberta, a cabeça já se adianta para ver quem se acha atrás da porta. Foram várias ações num só pensamento: "Vou abrir a porta".

2.4.7 DESENHO DIRETO (*STRAIGHT AHEAD ACTION*)

É o processo pelo qual o animador executa o primeiro desenho e a partir deste, segue direto (*straight ahead*) para o fim da cena sem interrupção de modo algum.

Essa técnica requer experiência e determinação pois, a cena evolui e se modifica durante o próprio trajeto sem prévios estudos. Não há quadros chaves nem intermediação e, o resultado final, é de desenhos soltos e espontâneos.

Poderíamos compará-lo com a filmagem ao vivo quando a câmera é disparada e vai registrando tudo indiscriminadamente, tirando é claro o recurso da montagem depois.

2.4.8 DESENHO QUADRO A QUADRO (*POSE-TO-POSE ACTION*)

Aqui há pleno planejamento prévio da cena. Utilização de quadros chaves, técnicas de intermediação, quantidade de desenhos necessários para determinada ação, intersecções, temporizações,

O processo de fazer um desenho completo e animá-lo, um por um, preenchendo assim o espaço entre os quadros chaves seria por demais trabalhoso e desgastante se aplicado à animação por computador. É a técnica do *layer-by-layer* (ou *tweaking*), levando em conta um sistema de modelagem hierárquico.

Assim, ao invés de se animar uma pose completa de cada vez foi criada uma completa transformação por níveis: começando por animar o tronco de uma figura, faz-se o seu ciclo completo; vem em seguida a animação dos membros principais, novamente o ciclo completo; depois membros secundários e assim sucessivamente, até chegar às partes mais detalhadas.

O que permanece em comum entre a animação convencional à mão-livre (*hand-drawn*) e animação por computador é o caráter de planejamento prévio de tempos e poses.

2.4.9 COMEÇO E SAÍDA LENTAS (*SLOW IN* E *SLOW OUT*)

É o efeito obtido através do controle do espaçamento entre os quadros intermediários. Também conhecido por *easy-in-out*. Está intimamente relacionado com a aceleração do movimento.

Saída lenta (*slow out*) redução da velocidade para uma parada.
Começo lento (*slow in*) aumento da velocidade.

Assim, se você quizer obter um movimento lento, fluído, que será plenamente degustado pela platéia, é só agrupar as poses uma bem perto das outras.

Em oposição para um efeito de velocidade o espaço entre poses terá que ser cada vez maior (lembre-se do efeito *strobing* causado pelo mau emprego da saída lenta).

Slow in-out são obtidos com o auxílio de cartas de tempo (*timing chart* ou, minutagem), que mostra ao animador onde serão colocadas as poses.

Em AMC os quadros intermediários são feitos automaticamente usando, por exemplo, interpolações por *spline*. Assim, saídas e começos lentos são controlados pelo ajuste da tensão, direção e continuidade dos *splines*.

Um problema que poderá ocorrer com esse método é a "transgressão" dos limites dos quadros chaves pelo *spline*. Isso acontece quando a mudança de valor entre poses é muito grande. Dependendo da variável que o *spline* está controlando (translação, rotação ou escalamento). O movimento seguirá na direção errada logo antes ou após a grande mudança da ação. Quando isso ocorre antes poderá ser tomado como antecipação, mas na maioria das vezes o efeito é indesejável.

Para evitar esse defeito a solução é colocar uma outra imagem (exatamente igual) logo ao lado da pose onde ocorreu a "transgressão". Assim o movimento é travado e permanece estável. É como se aumentássemos o peso de um ponto de controle de uma *spline*.

No desenho animado convencional é feita uma prática semelhante. Quando queremos que uma imagem permaneça mais tempo na tela basta registrar no filme mais de uma vez o mesmo desenho assim na projeção final sua presença será enfatizada.

2.4.10 ARCOS (*ARCS*)

O caminho percorrido por um objeto de um extremo a outro é geralmente descrito por um arco. Os arcos proporcionam maior naturalidade ao movimento.

Na maioria das AMC o percurso de uma ação é controlada pela mesma *spline* que controla a temporização simplificando a computação mas, acarretando efeitos indesejados: Quando a ação é lenta e os quadros intermediários próximos uns aos outros (*slow-in*) o arco é representado, mas quando a ação é rápida e os quadros intermediários bem espaçados então isso achatará o arco.

Uma solução para o problema seria fazer a *spline* que controla o arco separadamente da *spline* que controla a temporização. Assim, poderíamos manipular uma sem que isso necessariamente afetasse a outra.

2.4.11 EXAGERO (*EXAGGERATION*)

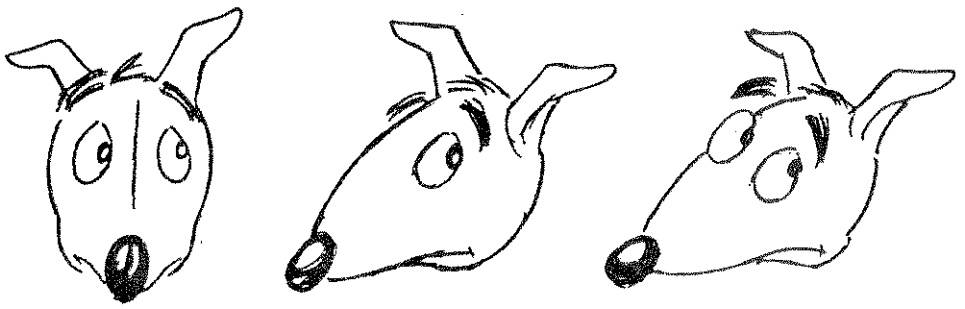
O princípio básico desse tópico é o seguinte: em animação convencional se você quer passar uma expressão ou movimento, você terá sempre que exagerar na dose, ir além dos limites dos padrões da realidade. Afinal esse é o grande trunfo criativo do desenho animado, não ter que dar satisfações à realidade. Assim, se temos que fazer uma figura triste, fazemo-la muito muito triste, se ao contrário ele está alegre podemos fazê-la brilhar e levitar de tanto contentamento.

Exagero contribui para dar mais poesia e apelo à animação e, pode ser alcançada pela forma, ação, distensão, esmagamento, a cor e sonorização.

Tudo isso é claro dentro de uma dosagem equilibrada entre o exagero e o natural para que o resultado final não caia no grotesco e na deformação da linguagem e da

arte.

Um recurso de exagero que dá maior expressividade ao personagem é o *Cheat* (do inglês: engano, embuste) que, apesar de fácil em 2D, requer um esforço e ferramentas extras em 3D pois, não é um processamento óbvio, real (ver figura 2.15).



(a) (b) (c)

Figura 2.15 : Engano, *cheat* (KROY/87).

2.4.12 AÇÕES SECUNDÁRIAS

É uma ação diretamente resultante de uma outra, esta a principal. Ela pode ocorrer logo antes (antecipação, p. ex) ou depois. O caso do *follow-through*, exemplifica bem a separação de ação principal e ação secundária. Assim o movimento do "lider" (como foi apelidado) é a ação principal, e o movimento de seus membros e apêndices, ações secundárias e subordinadas a ela.

Se entretanto, essa ação secundária e subordinada causa um conflito, torna-se mais interessante, ou domina a ação principal então, algum erro de *staging* está acontecendo (lembrando que *staging* é a coordenação das ações para que elas ocorram no lugar e hora certas). Portanto, um movimento secundário tem que ser bem posicionado e dentro do seu próprio tempo, para que não coincida e se choque com uma ação maior.

As Produções Disney caracterizaram-se por um cenário (inclusive o plano de fundo)

rico de movimentos realistas o que pode se contrapor a idéia inicial de manter a atenção do espectador em um único "ator" em movimento. Entretanto, as ações do cenário são ações simples, sem importância no contexto da história ou da cena em questão. Logo, os movimentos do cenário não contribuem para desviar o espectador do foco principal de atenção mas sim, para embelezar e tornar mais real (menos estática) e dinâmica a cena.

2.4.13 APELO (*APPEAL*)

Não é meramente colocar um gatinho ou uma mulher sensual em cena. Falando objetivamente o apelo pode estar na clareza de uma ação, numa boa comunicação, ou no magnetismo de uma personalidade. É a capacidade de cativar e manter a atenção do público para um personagem ou enredo da história.

Desenhos fracos e inseguros, ações confusas e um argumento bobo não combinam com o apelo.

Uma dica que facilitará a obtenção natural do apelo para um personagem é evitar sempre que possível, o fenômeno chamado *twins* onde, membros (braços ou pernas) estão fazendo a mesma coisa ao mesmo tempo. É pertinente dar uma certa assimetria entre ambos os lados para maior naturalidade ao movimento. Do mesmo modo que um lado da face humana nunca é o espelho da outra.

2.4.14 PERSONALIDADE

Personalidade de um personagem não é um princípio propriamente dito mas sim, a resultante do trabalho conjunto de todos os citados anteriormente.

Se o personagem e sua história vingarão na função de entretenimento, passando a técnica de sua execução a um segundo plano, então o objetivo foi alcançado.

Uma boa animação é resultado do uso inteligente de suas técnicas. Antes de animarmos um personagem temos que estabelecer previamente como vai ser sua aparência, seu modo de agir e pensar. A partir destes elementos estudados e pré-fixados é que

vamos construir o universo de ações e reações do personagem que contribuirão para delinear sua personalidade.

Um personagem não deve realizar uma ação em particular da mesma maneira em duas situações emocionais diferentes. Mas, suas ações devem ser previsíveis conforme a caracterização que se queira fazer. Além disso, dois personagens diferentes não devem realizar uma ação da mesma maneira.

2.5 CONCLUSÃO

Este capítulo procurou apresentar o que é animação, como ela é feita (convencionalmente, ou não), como ela se apresenta e o uso do computador nesta tarefa (apresentando classificações e detalhes sistêmicos e de implementação) além de considerar e enumerar algumas características da Linguagem da Animação.

Nos capítulos que se seguem entraremos nos detalhes do tratamento adotado por nós, procurando dispor detalhes filosóficos e pragmáticos sentidos durante o desenvolvimento do trabalho em cada um dos subsistemas componentes do TOOKIMA.

Muitos dos tópicos ou subtópicos que se seguem, constituem em si, uma vertente de investigação bastante vasta e, por isso mesmo, impossível de serem tratados em toda sua completude neste espaço. Assim uma certa simplicidade foi aqui adotada de forma a podermos nos aprofundar nos tópicos de nosso maior interesse.

CAPÍTULO 3

TOOKIMA : FERRAMENTAS PARA ANIMAÇÃO MODELADA POR COMPUTADOR

3.1 INTRODUÇÃO

Este capítulo apresenta o sistema TOOKIMA (a **TOO**l **KI**t for Scripting Computer Modeled Animation) desenvolvido no DCA-FEE-UNICAMP dentro do projeto ProSIm (Prototipação e Síntese de Imagens Foto-Realistas e Animação). Este sistema é uma concepção de Animação Modelada por Computador segundo as características expostas nos itens 2.3.1.1 e 2.3.2.2

Apresentamos a seguir, as diversas premissas de projeto e implementação que nortearam a implementação deste sistema (na verdade um conjunto de ferramentas). Posteriormente apresentamos os principais tipos de dados definidos no sistema ProSIm e importados para o *Scan-line* e TOOKIMA. Então detalhamos o ambiente de descrição da animação e o módulo descritor (denominado *script*).

3.2 AMBIENTE E CONVENÇÕES

3.2.1 DEFINIÇÕES E PREMISSAS

TOOKIMA (a **TOO**l **KI**t for Scripting Computer Modeled Animation) é um conjunto de ferramentas desenvolvidas para dar suporte a animação 3D, modelada por computador (não animação auxiliada, ver 2.3.1.1) via descrição algoritmica (através de procedimentos

que descrevem o comportamento de parâmetros) ou guiada (através caminhos/trajetória definidos pelo animador). Usa-se como abordagem de implementação a extensão de uma linguagem de uso geral (ver 2.3.1.2), a linguagem C, permitindo o modelo de controle cinemático das ações (ver 2.3.1.3).

Assim como ASAS é uma implementação/extensão da linguagem Lisp (REYN/82, pg. 290), TOOKIMA é tanto uma implementação em C, quanto uma extensão desta linguagem. TOOKIMA oferece uma biblioteca voltada à definição de atores, descrição de ações através do tempo e visualização 3D. Sua versão atual (1.0) atua com objetos modelados por *B-rep*, usando o algoritmo de visualização *Scan-line Z-buffer*, também apresentado neste trabalho. Sua interface com o usuário é textual (programa, *script*), descrevendo as relações temporais sobre as entidades gráficas, segundo uma organização sintática e estruturada.

A linguagem C foi escolhida por sua portabilidade, versatilidade, extensibilidade e performance (segundo as considerações em MCLA/85, pg. 43).

TOOKIMA, em relação à figura 2.3, é o nó raiz, ou seja, o módulo gerenciador da animação que recorre a outros módulos do projeto ProSIm (a saber, o modelador geométrico e um sistema de *rendering*), e possuindo módulos implementados especificamente para ele (estes incorporam meios de descrição -*script*-, sincronização e movimentação dos diversos elementos da cena).

Os módulos do ProSim (no caso o GeoMod e o *Scan-line*) são independentes entre si e do TOOKIMA. Todos implementados em estações gráficas SPARC-SUN com sistema operacional UNIX. Exceto o sistema de visualização *Scan-Line* que também tem versões para PC-DOS.

TOOKIMA é uma camada de software que se utiliza da camada de funções do ProSIm (que por sua vez se utiliza da linguagem C e do sistema operacional) de tal forma a facilitar e dar início às diversas investigações na área de animação e, para esse propósito, também pretende ser extensível e flexível, dando acesso de baixo nível (geométrico e topológico) às entidades gráficas definidas (ator, iluminação, etc.) no sistema.

Abaixo enumeram-se os princípios que inspiraram o desenvolvimento do TOOKIMA:

1) Possibilitar a descrição da variação no tempo, de características do objeto e

do ambiente, de maneira não ambígua, de forma que tudo que possa ser modelado possa ser animado;

2) Estas variações devem poder ser descritas através de várias possibilidades:

- Analiticamente (funções previamente definidas pelo usuário);
- Incremental simples (o uso de um tempo absoluto como referência, é um exemplo);
- Algoritmicamente (pela enumeração de transformações elementares, por exemplo);
- Graficamente (através do uso de funções interpoladoras ou aproximadoras de curvas conhecidas ou livres - ver SARA/86, pg. 9);

3) Prover algum mecanismo de sincronização entre fragmentos do *script* e a geometria temporal;

4) Permitir a abstraçao multinível e estruturada, a geração de código reutilizável e a descrição incremental do script (*tweaking*);

5) Deve permitir a descrição da trajetória e acelerações dissociadas;

6) Ser flexível e extensível de forma a poder ser aplicado a objetos CSG e outros sistema de visualizaçção como *Ray-Trace*, etc. e;

7) Executar nas plataformas UNIX e DOS (para tal, deve ser sequêncial e não concorrente).

As convenções apresentadas a seguir, bem como os detalhes de funcionamento de cada um dos módulos e a codificação das diversas funções podem ser melhor analisadas em HOUN/92. Aqui apenas trataremos as principais características da implementação.

3.2.2 CONVENÇÕES

Três convenções devem ser ressaltadas :

1) a sintática/semântica;
2) as que norteiam o ambiente 3D, e;
3) as convenções de estruturação.

3.2.2.1 CONVENÇÕES SINTÁTICAS E SEMÂNTICAS

Adotamos a sugestão de estilo de programação dada por Kernigham e Ritchie (KERN/86) de que macro-funções, constantes e estruturas de dados definidas pelo sistema comecem com letra maiúscula:

Exemplos :

EPSILON E-20	(Define Constante)
SET_POINT	(Define Macro-função)
Polyhedron	(Define estrutura de dados de ProSIm)
Actor	(Define estrutura de dados do TOOKIMA)

E, consequentemente, palavras que começam com letra minúsculas indicam variáveis, sub-rotinas (funções), palavras reservadas ou funções da biblioteca do ProSIm e TOOKIMA.

Cada módulo possui uma convenção sintática para suas funções que deverá ser ressaltada no capítulo que descreve o respectivo módulo.

Exemplos :

canon	(variável da estrutura Camera)
actors	(variável da estrutura Actor)

3.2.2.2 CONVENÇÕES DO AMBIENTE

Abaixo enumera-se algumas das convenções que nortearam a atual implementação.

1) O Sistema de coordenadas é da mão direita (*Hight Hand*).

2) Os objetos são modelados por bordas (*Boundaring Representation - B-rep*).

3) As faces dos objetos são orientadas no sentido horário.

4) O algoritmo de visualização (*rendering*) é do tipo *Scan-line Z-buffer*.

5) A iluminação é calculada pelo modelo de iluminação de Phong.

6) Os modelos de tonalização (*shading*) são *Flat*, *Gourand* ou *Phong*.

7) As fontes de iluminação são pontuais.

8) As principais variáveis do sistema são globais.

9) É permitido o acesso direto a elementos e as variáveis do sistema (inclusive seus campos) para quaisquer procedimentos desenvolvidos pelo usuário-animador.

10) As restrições que definem um sólido são aceitas mas, não são verificadas pelo sistema, permitindo que os objetos possam ser definidos mais flexivelmente, isto é, os objetos podem ser poliedros, polígonos ou superfícies no espaço mas não incorporam características físicas inerentes como inpenetrabilidade, cor, massa, etc. Estas podem ser associadas ao objeto, *a posteriori*.

3.2.2.3 CONVENÇÕES DE ESTRUTURAÇÃO

O TOOKIMA integra varios módulos que foram implementados objetivando o mínimo de acoplamento e o máximo de modularidade. Os módulos foram então concebidos tentando ressaltar diversas funcionalidades que podem, por si só, serem expandidas ou melhoradas, são eles:

- MÓDULO *CAMERA*
- MÓDULO *LIGHT*
- MÓDULO *SCANLINE*
- MÓDULO *ACTOR*
- MÓDULO *MOTION*
- MÓDULO *SCRIPT*

Cada módulo possui no mínimo, um arquivo cabeçalho (*header*, arquivo com extensão ".h") com o nome do módulo, contendo as estruturas e protótipos associados à funções do módulo.

Os diversos arquivos de programa (com extensão .c) identificam no seu nome a funcionalidade das rotinas neles contidas e o módulo a que pertencem (este último indicado pelas primeiras letras do arquivo).

Exemplos :

Módulo Camera	- cabeçalho	camera.h
	- arquivos	c_define.c
		c_transf.c
		etc.
Módulo Scanline	- cabeçalhos	scanline.h
		sl_wine.h
	- arquivos	sl_main.c
		sl_sort.c
		sl_line.c
		etc.
Módulo Actor	- cabeçalho	actor.h
	- arquivos	a_define.c
		a_rotate.c
		etc.

3.3 PRINCIPAIS ESTRUTURAS DE DADOS DO SISTEMA ProSIm

Abaixo listamos as principais estruturas de dados definidas no ambiente ProSIm e que foram intensivamente utilizadas no TOOKIMA e no *SCANLINE*.

Um tipo de dado não disponível na linguagem C, mas, facilmemte definível, é o tipo booleano, como segue abaixo:

```
typedef  enum  { FALSE, TRUE }  Boolean;
```

Para representar um ponto ou um vetor tridimensional do espaço euclidiano virtual no sistema de coordenadas denominado do Mundo (WC), o ProSIm define a seguinte tríade para compor a estrutura Point e Vector:

```
typedef  struct { double x,y,z;}        Point;
typedef  struct { double x,y,z;}        Vector;
```

As transformações geométricas são modeladas internamente como uma multiplicação matricial (ver FOLE/90, cap. 5) onde se aplica uma matriz homogênea, que contém informações sobre a transformação, sobre um vetor (cujas componentes foram abstraidas das estruturas Point ou Vector). Esta representação tem muitas vantagens para manipulação e concatenação de transformações:

```
typedef  double [4] [4]     Matrix;
```

Para representar uma cor na tela do monitor (uma vez que este se utiliza de uma tríade de componentes básicas para reproduzir a cor), adotamos a representação RGB (*red, green and blue*, que são justamente as componentes básicas de cor), disponível através da seguinte estrutura:

```
typedef  struct { double r,g,b;}        Color;
```

Para a técnica de visualização em questão o modelo de objeto mais largamente utilizado é o modelo por bordos (*B-rep*), apesar de algumas implementações utilizando árvores CSG - Construtive Solid Geometry - (ver PUEY/87) onde as primitivas são funções analíticas parametrizadas.

O modelo *B-rep* pode ser muito bem definido por uma estrutura que represente um **poliedro**, por isso, de agora em diante utilizar-se-ão indistintamente os termos face e polígono como equivalentes :

```
Objeto   <=====>  Poliedro
Face     <=====>  Polígono
```

O polígono é usado para definir um poliedro pois, considera-se que um poliedro é um arranjo de vários polígonos dispostos harmonicamente no espaço, de forma a representar uma determinada forma. Assim, o polígono, ou face, será descrito pela seguinte estrutura:

```
struct {    nvert;
            *v_list;
        } Face;
```

E o poliedro então usa na sua estrutura uma lista de faces que referenciam os pontos do poliedro, em outra lista :

```
struct {    npoints;
            Point *p_list;
            nfaces;
            Face  *f_list;
        } Polyhedron;
```

Exemplo :

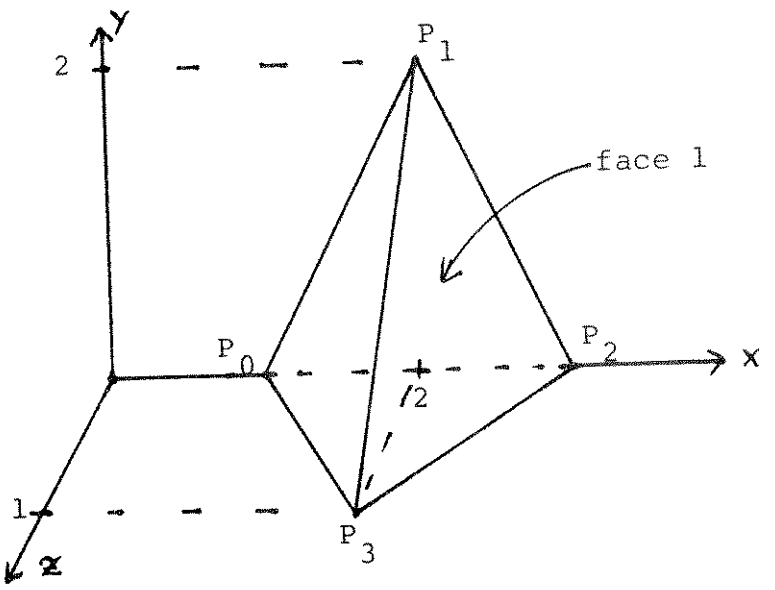


Figura 3.1 : Objeto Poliédrico em *B-rep* (HOUN/90).

Avaliando a figura 3.1 podemos obter, a título de exemplo, os seguintes dados, conforme as estruturas definidas :

```
npoints =   4
p_list  =   (1,0,0), (2,2,0), (3,0,0) e (2,0,1) /*ponto 0 à n-1*/
nfaces  =   4
f_list  =   (3, (0, 1, 3)),          /* face 0 */
            (3, (1, 2, 3)),          /* face 1 */
            (3, (0, 3, 2)),          /* face 2 */
            (3, (0, 2, 1))          /* face 3 */
```

Por fim, queremos ressaltar algumas macro-funções definidas no ProSIm que também são muito úteis:

SET_POINT, instancia os campos x, y e z de um ponto (Point).
SET_VECTOR, instancia os campos x, y e z de uma variável vetor (Vector).
SET_RGB, instancia os campos r, g e b de uma variável cor (Color).

3.4 DESCRIÇÃO DA ANIMAÇÃO

Este item é destinado a apresentar a maneira usada para descrever uma animação no TOOKIMA, apresentando a sua entidade de controle, denominada SCRIPT, sua divisão (estruturação), seus diversos elementos e seu funcionamento.

"Dividir para Vencer" é um conceito muito útil em qualquer área do conhecimento e tem uma relação muito grande com o princípio da abstração e projeto *top-down*. Influenciado por este princípio, a própria animação convencional é feita em etapas (*storyboard*, quadros chaves, etc.) o que também acontece na animação modelada e, consequentemente, no TOOKIMA.

3.4.1 DEFINIÇÕES E HIERARQUIAS

Uma estória normalmente é dividida em pequenas idéias sequencialmente dispostas que compõem o enredo. Badler (BADL/87, pg. 112) mostra um tipo comum de divisão :

- Animação, a estória, o enredo, o todo;
- Episódio, partes do todo com introdução, desenvolvimento e conclusão;
- Cena, fragmento do episódio caracterizado por algum tipo de continuidade (lugar, atores ou ações);
- Sequência, divisão subjetiva das idéias de uma cena que tenham alguma característica em comum no tempo e/ou espaço;
- Corte, intervalo com continuidade no movimento da câmera;
- Quadro, unidade de movimento da imagem.

Mantendo consistência com a representação de atores complexos, aqui também usamos uma hierarquia (árvore) para representar o processo de descrição da animação.

A entidade de mais alto nível e mais abstrata (a raiz) é o SCRIPT que conterá, de forma sucinta, uma descrição geral da animação.

Esta entidade se desdobra em CENAS, com maior nível de detalhes e assim sucessivamente, criando-se SUB-CENAS, aumentando mais e mais os detalhes de variação e movimento dos atores, até chegar ao nível mais explícito de detalhes onde já não são mencionadas as mudanças mas, efetivamente mostrados quadro a quadro (as folhas). A figura 3.2 ilustra uma árvore de estruturação de uma animação.

O *Script*, que nada mais é do que um programa principal, gerencia as cenas sendo responsável pelo macro controle de animação.

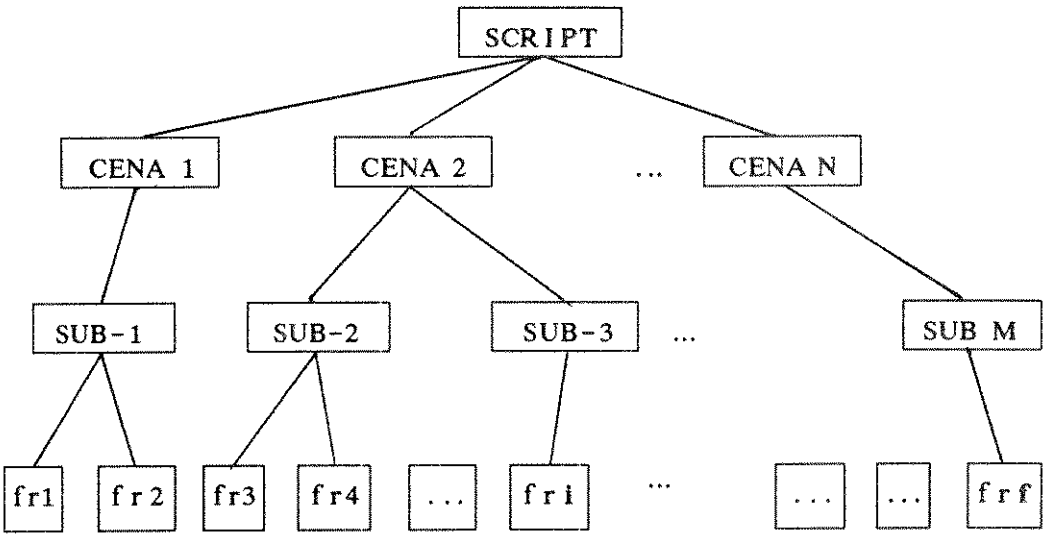


Figura 3.2 : Estrutura de uma Animação.

As cenas gerenciam as suas sub-cenas e perfazem um micro controle sobre estas, e assim sucessivamente em quantos níveis forem necessários.

3.4.2 ESTABELECENDO CENAS

Estabelecer cenas através de uma divisão inteligente de sequências é fundamental para garantir clareza, facilidade de descrição e corretude do *script*. Portanto, é importante considerar fatores para que uma boa divisão de cenas possa ser alcançada.

Uma cena pode ser estabelecida pelo animador de várias formas. Usualmente

escolhe-se enclausurar numa cena um determinado ator, ou determinadas ações ou ainda, um espaço definido de tempo.

Quando se tenta enclausurar um ator numa cena aglomeram-se nela todas as ações (transformações geométricas) que o ator deverá seguir.

Quando o enclausuramento é de ações, aglomera-se na cena toda uma sequência de transformações sobre o(s) ator(es) que tenha algum significado teatral (uma ação, um close, um "gancho" para uma outra cena), um fragmento contínuo de uma história ou uma ação complexa do enredo.

Quando o enclausuramento é puramente temporal, existirão parâmetros subjetivos que definirão fragmentos do enredo delimitados no tempo e portanto, cada um destes lapsos de tempo (que podem conter toda uma idéia) pode ser considerado como uma cena.

Entretanto, como estas três opções não são absolutas pode-se delimitar uma cena através de uma mistura das formas acima. A intenção na definição da cena no nosso ambiente é que ela possa facilitar e modularizar a descrição da animação.

Independentemente de qual método seja utilizado para delimitar uma cena, as transformações de seus atores sofrem influência da sequência em que foram definidas. Essa influência quanto a sequência acontece porque as transformações são descritas matematicamente através de matrizes homogêneas e as matrizes, de um modo geral, não tem a propriedade comutativa da multiplicação ($A * B \neq B * A$).

Portanto, a ordem das transformações é importante, bem como a ordem em que as cenas foram dispostas.

A ações e atores internos a uma cena só influenciarão os quadros se estiverem compreendidos num período de tempo pré-estabelecido (como veremos adiante) para esta cena. Assim, a cena está sob controle do *script* mas, só será usada dentro do período de tempo a ela determinado.

Dentro de cada cena é opcional que o diretor crie sub-cenas, enclausurando detalhes da cena e/ou de um ator. No nível mais baixo da descrição da animação, ou seja no nível mais detalhado, estão os diversos quadros (imagens estáticas) que compõem aquela sequência.

O número de quadros obtidos em uma animação independente da duração ou posição de cenas, depende exclusivamente do tempo total (*TotalTime*), em segundos, definido para a animação e a taxa de quadros por segundo (*Rate*).

Note-se pois que existem, no mínimo, três níveis de descrição da animação, ou seja, o script, a cena e o quadro são níveis obrigatórios que descrevem uma animação no TOOKIMA.

O sistema foi construído de tal forma que as ações são independentes dos atores (ou seja, uma ação está disponível para qualquer tipo de ator). Isto permitirá a descrição de ações, trajetórias e/ou acelerações complexas para um ator simples, a nível de teste e depois, aplicar ou reutilizar aquela descrição para outros atores ou o ator complexo definitivo.

3.4.3 RELÓGIO ABSOLUTO

Embora a animação crie a ilusão do movimento contínuo no tempo, isto é fundamentalmente baseado em modelos discretos do tempo.

Valores discretos do tempo, como o número do quadro, é talvez, um dos mais elementares meios de se descrever o tempo na animação (GOME/84, pg. 292). A amostragem é feita no final do processo da animação, justamente antes que a descrição específica de um quadro seja passada para o subsistema de visualização (assim como definido em BADL/87, pg. 109).

No TOOKIMA optou-se por representar o tempo para o animador em segundos, denominando-o de **Tglobal**, em conjunto com a taxa de quadros desejados por segundo, denominando-a de **Rate**, que deve ser compatível com a mídia na qual a animação será veiculada (normalmente 24 q./seg. para cinema e, 30 q./seg. para vídeo).

Este é, pois, o tempo absoluto que rege a animação cujo valor máximo é fornecido pelo animador (em segundos) através de uma variável denominada **TotalTime**.

Para que uma cena possa ser encaixada em qualquer instante da animação concebeu-se uma estrutura descritora da localização temporal da cena (seu começo e término) denominada **Cue**, (comentários em MCLA/85, pg. 25), que nos permitirá realizar

a geometria temporal e que é implementada como uma variável global.

A estrutura Cue contém dois valores que identificam em que instante absoluto (*Tglobal*) começa (cue_start) e termina (cue_stop) uma determinada cena, em segundos.

Uma cena, dentro de seus limites de começo e término, é dita ATIVA, podendo ocorrer que várias cenas estejam ativas num mesmo instante (*Tglobal*), semelhantemente a GOME/84, pg. 293, um instante ou quadro pode ser considerado como a união de todas as cenas ativas.

Dentre os elementos que são acessíveis a qualquer instante tem-se o *Tglobal*, TotalTime e Cue, além de outros indicadores (*flags*) e estruturas do sistema (como os atores, as decorações, a iluminação e a câmera) que serão oportunamente detalhadas.

3.4.4 TIPOS DE CENAS E *CUES*

A cada Cue estão associadas uma, ou mais, cenas e, cada cena deve estar associada a, pelo menos, uma Cue. Devido a esta relação, estes dois termos são considerados *quase* sinônimos neste texto, isto é, quando nos referenciarmos a um dos dois termos, a afirmação se aplica ao outro também, a menos que se especifique o contrário.

Caso duas cenas devam ocorrer sempre num mesmo intervalo de tempo então pode-se usar para elas a mesma Cue.

Os limites de uma cena podem ser manipulados por qualquer cena (inclusive ela mesma), bastando manipular os limites da Cue a ela associada, o que permite definir vários tipos de Cue's e/ou cenas: condicionais, cenas de duração variável, cenas cíclicas (ver MCLA/85, pg. 31), perfazendo complexas manipulações na atividade das cenas.

CENAS CONDICIONADAS são aquelas que estão relacionadas a Cue's condicionais e esta tem sua atividade determinada por outra cena. Existem dois tipos de ativação da Cue, ativação simples e ativação completa.

Na ativação simples (Active_Cue(*number*)), a Cue só pode ser ativada antes de um instante determinado. Se tentarmos ativar uma Cue depois deste instante ela não será

ativada. Neste tipo de ativação a Cue já tem seu instante de desativação estabelecido e, recebendo uma instrução de ativação, ela estará ativa no próximo quadro.

Na ativação completa (Active_Cue_Until(*number*,t)) a Cue sempre estará ativa no próximo instante e será estabelecida pelo agente ativador a sua duração.

Cenas com duração variáveis são aquelas associadas a cenas de duração variável cuja desativação é feita por alguma cena, inclusive uma cena daquela Cue. Também existem dois tipos de desativação: Desativação Simples e Desativação Retardada.

Na desativação simples (Kill_Cue(*number*)), de imediato a Cue deixará de ser avaliada, já no próximo instante logo, as cenas a ela associadas não influenciarão o próximo quadro.

Na desativação retardada (Kill_Cue_At(*number*,*time*)), a desativação se dará num instante (*time*) estabelecido pelo agente desativador.

DUPLICAÇÃO DE CENAS é um mecanismo que permite ao animador reutilizar por completo uma cena para compor um outro instante da animação. Isto é facilmente implementado definindo várias Cue's para a cena e colocando as várias chamadas com as respectivas Cue's na lista de cenas (LIST_SCENES - ver 3.4.6).

CENAS CÍCLICAS são aquelas que, uma vez iniciadas, se repetem indefinidamente com o período estabelecido na sua Cue. Devem ser declaradas associadas a uma Cue cíclica e conter comandos diferenciados para começo e término de sua descrição, BEGIN_CICLIC_SCENE e END_CICLIC_SCENE, que são responsáveis por atualizar os valores de *cue_start* e *cue_stop* quando a cena estiver prestes a terminar.

Os efeitos de CONGELAMENTO DE CENA (repetir uma imagem estática várias vezes para dar a impressão, na animação, de que se congelou a imagem) e ESPELHAMENTO DE CENA (permitir a apresentação da ação de maneira normal *-forward-* ou reversa *-backward-*) são recursos que não estão disponíveis pois, foram deixados para um subsistema posterior encarregado do PLAYBACK. Sendo assim, o sistema TOOKIMA se responsabilizará em gerar todas as diferentes imagens de uma animação e a edição será feita pelo PLAYBACK, evitando que o TOOKIMA perca tempo calculando uma mesma imagem repetidas vezes.

Atores que interagem entre si devem estar numa mesma cena. Caso estes atores

estejam em cenas distintas, sua intercomunicação é possível, de maneira limitada, através de variáveis globais. Deve-se notar que esta solução oferece um nível muito baixo e difícil para comunicação, adequada somente para interações simples (ver também VELH/89, pg. 511).

Não é aconselhável que duas cenas, que manipulem um (ou mais) ator(es) em comum, estejam ativas num mesmo instante. Isto, normalmente, é fruto de uma definição ou modularização errônea da animação, poucas vezes é o que o animador realmente deseja (apesar desta situação ser perfeitamente factível e viável).

Também, é aconselhavel que as cenas sejam independentes entre si, evitando que ações feitas em uma cena ocasione alterações ou ações sobre atores em outra(s) cena(as). Isto pode ser bastante complicado e levar a laços eternos se a dependência for recíproca.

3.4.5 RELÓGIO LOCAL

A fim de garantir a independência da dimensão temporal de uma cena da dimensão temporal absoluta da animação (*Tglobal* dado em segundos), define-se, para cada cena, o seu relógio local com sua duração local máxima (invocado obrigatóriamente com o comando CLOCK no começo da descrição de uma cena ou NO_CLOCK, caso não se queira nenhum relógio local), seu tempo relativo (denominado *Tlocal* - ver também ROSE/88, pg. 60-65), e diversos outros parâmetros que permitirão sincronizar e compatibilizar o tempo interno da cena com o instante absoluto de criar uma imagem estática (o quadro).

Os mecanismos acima nos permitirão descrever uma cena usando um relógio conhecido e posteriormente definirmos o encaixe desta cena na nossa animação. Esta manipulação é denominada de Geometria Temporal. Assim, se uma determinada cena é conhecida numa dimensão temporal de anos, pode-se adotar esta escala de tempo na cena e, o sistema se encarregará de efetuar os devidos ajustes para o sincronismo e geração das imagens.

O termo geometria induz a existência de transformações como translação, escalamento e rotação do tempo.

São estes os termos usados na geometria temporal ? A estas e outras peguntas nos dedicaremos abaixo pois a manipulação destas características juntamente com os reló-

gios, local e absoluto, permitirá estabelecer o sincronismo entre as cenas no TOOKIMA.

3.4.5.1 GEOMETRIA TEMPORAL

O tempo, em muitas aplicações em animação, é considerado como um eixo, uma extensão do sistema euclidiano e a ele podem ser atribuidas transformações geométricas. Mas, o que pode significar uma translação no eixo T ? (ver ROSE/88, pg. 31).

Significa que em uma estória, onde nós podemos especificar os fragmentos desta e o sincronismo entre as partes, que um determinado fragmento ocorrerá mais cedo ou mais tarde em relação ao tempo absoluto conforme se translade negativa ou positivamente o fragmento no eixo T, respectivamente.

Logo, a translação no eixo T tem a ver com o **quando** de uma determinada ação.

O escalamento no eixo T se relaciona com a duração de uma cena ou, fragmento de estória. Isto pode nos proporcionar os efeitos câmera lenta (*slow motion*) e câmera rápida (*fast motion*), comuns na terminologia dos vídeo-cassetes domésticos e que permitem a projeção lenta ou rápida, aumentando ou diminuindo a escala do tempo, respectivamente.

Escalar no eixo T tem a ver com o **quanto** de uma determinada ação.

As idéias acima podem ser absorvidas graficamente através da visualização de um eixo com um segmento sobre ele sofrendo as operações acima. Estas operações são fáceis de assimilar porque são independentes do eixo (daí podermos fazer uma relação com os efeitos sobre os eixos euclidianos) e o tempo não interfere com os outros eixos (o espaço), o que não ocorre com a operação de rotação, por isso não mencionaremos esta transformação sobre o eixo T (para maiores detalhes, ver ROSE/88, pg. 31-65).

Com estes elementos da geometria temporal (translação e escalamento) pode-se estabelecer o instante inicial e a duração de qualquer cena, desde que se tenha uma referência temporal para o eixo T. A sincronização é conseguida pois no TOOKIMA, pela concepção de *Tlocal* e *Tglobal* e Geometria Temporal.

3.4.6 ESTRUTURA DO TEXTO DO *SCRIPT*

A entrada principal do TOOKIMA é um texto (um *script*) definindo dados de entrada (objetos, cores, lista de pontos de controle para trajetórias, etc.) e relações (transformações) sobre estes dados de entrada, com o objetivo de gerar um certo número de imagens.

Deve-se notar que a saída tanto pode ser uma sequência de arquivos descritores de cenas estáticas a serem interpretados por qualquer um dos sistemas de *rendering* do ProSim, quanto as próprias imagens estáticas já prontas. Como opção para visualização imediata da imagem, na atual implementação do TOOKIMA decidiu-se pela segunda opção, ou seja, a integração com o *Scan-line* para a imediata geração de imagens, pela facilidade dos testes.

Para usufruir de todos os mecanismos acima apresentados, estabeleceu-se uma estrutura para o SCRIPT representando diversos blocos de interesse.

Esta estruturação reflete os diversos elementos que compõem uma animação, instanciando parâmetros necessários à definição do ambiente completo e estabelecendo as condições para a execução e geração da animação, devendo-se iniciar pela estruturação destes módulos para depois se deter às minúcias das cenas.

```
BEGIN_SCRIPT
        set_studio (cfg_name, render_quality, picture_name)
        Rate, TotalTime
        cast, furniture
        paint
        wings
        cue_0, cue_1, cue_2...
        LIST_SCENES
            Cena_0
            Cena_1
            Cena_2
            ...
        END_ LIST
END_SCRIPT
```

- *Set_studio* é a função responsável pela inicialização da iluminação e da câmera, através da leitura de valores num arquivo de configuração (*cfg_name*), definição da qualidade da visualização (*render_quality*) e do nome das imagens (*picture_name*).

Em seguida é estabelecida a taxa de quadros por segundo (*Rate*) e o tempo total (*TotalTime*) da animação a ser gerada, em segundos.

- *Cast* (elenco) é a listagem de todos os atores, seus tipos e suas instâncias (poliedro, face, referência a ponto, etc.) e os respectivos nomes dos personagens que irão representar.

- *Furniture* (decoração) é a enumeração de todas as mobílias (poliedros) que serão usadas nos cenários de todas as cenas, nomeando-as de acordo com o cenário que comporão.

- *Paint* (figurino) é o módulo que estabelece todas as roupas (no caso cores e tipo de tonalização, ver item 4.4) que os atores e a decoração terão.

- *Wings* (bastidores) são as inicializações necessárias nos elementos da animação antes do "espetáculo" começar. esta função é necessária para compor e posicionar os atores e estabelecer suas relações de hierarquia.

Grande parte (senão todos) dos instanciamentos e dos posicionamentos desta fase podem ser feitos no sistema modelador GeoMod (MADE/92).

Em seguida, estabelecem-se todas as "deixas" (Cue's) definindo-se os intervalos de cada cena, seus sincronismos e seu tipo (cíclica, condicional, etc.).

Enfim, listam-se as cenas e estabelece-se a relação cena-cue bem como, a ordem em que as cenas devem ser avaliadas num determinado instante (muito importante se duas cenas estiverem ativas num mesmo instante, com atores em comum). Note que a ordem de ativação é diferente da ordem temporal, esta última é estabelecida pelas Cue's.

Vamos supor que já definimos várias cenas coerentemente para então explicar o funcionamento do sistema.

O TOOKIMA interpreta o comando BEGIN_SCRIPT e já estabelece alguns parâmetros para funcionamento. Em seguida os módulos de inicialização do estúdio, do elenco, da

decoração e das deixas são interpretados.

Quando se encontra a instrução LIST_SCENES é criada uma variável global que será a referência absoluta de tempo do sistema (*Tglobal*) a qual norteará as deixas. Daí então, o *script* "incansavelmente" passará o controle do ambiente para cada uma das cenas, ordenadamente listadas e estas, caso estejam ativas, serão avaliadas executando-se as ações e/ou subcenas nelas contidas.

Caso a cena não esteja ativa naquele instante, então o TOOKIMA passará o controle para a próxima cena e assim, sucessivamente, até avaliar todas as cenas quando, transparentemente ao usuário, passará o controle para o subsistema de visualização (*shoot*) para gerar a imagem estática relativa às transformações e/ou cenas naquele instante *Tglobal*, com o número do quadro associado.

Ao final da visualização é gerado um incremento (*TICK*) que atualizará o relógio global de forma a garantir *Rate* imagens por segundo na animação final e então, recomeça-se o processo para aquele novo instante até alcançar o tempo total desejado.

A cena é a entidade onde se encapsula as ações representadas por transformações sobre os atores. Na cena, implementada como uma função da linguagem C, usa-se um subconjunto do total de atores e decorações definidos no *Script* portanto, é necessário inicialmente estabelecer quais atores e decoração participam desta cena (com os comandos *part* e *decor*, respectivamente) para que eles realmente apareçam na imagem.

As cenas podem tambem ter parâmetros que modificam seu funcionamento interno, o que pode permitir a comunicação de duas cenas ou a descrição de cenas para propósitos semelhantes em situações diferentes.

A cena é a única entidade (o único nível), além do *script*, que tem definido uma dimensão temporal própria, permitindo que as relações entre estes dois níveis gere a geometria temporal. As sub-cenas e outros níveis não geram dimensão temporal própria. Partiu-se do pré-suposto que dois níveis serão suficiente para a maioria dos casos.

Por fim, termina-se o *script* com o comando END_SCRIPT a fim de que o sistema libere as possíveis áreas de memória e variáveis dinâmicas utilizadas durante a geração das imagens.

3.5 ESTRUTURA DO MÓDULO *SCRIPT*

O subsistema de *script* mais as características temporais de definição de relógios e geometria temporal estão melhor descritos no arquivo cabeçalho *script.h*.

Este arquivo contém as principais estruturas de dados relativas à temporização e, principalmente, a definição das diretivas usadas no texto do *script*.

Dentre as estruturas de dados deste arquivo temos a Cue e a descritora do relógio local Clock :

```
typedef struct {double      start,
                            stop;
               } Cue;

typedef struct {double      start,
                            stop,
                            inv_dura,
                            t_local,
                            d_time;
               } Clock;
```

Quanto às diretivas (implementadas como macro funções da linguagem C) vamos, subdividí-las em estruturais e temporais.

As diretivas estruturais são aquelas usadas na estruturação ou descrição do texto do *script* :

BEGIN_SCRIPT e END_SCRIPT;
BEGIN_CICLIC_SCENE e END_CICLIC_SCENE;
BEGIN_SCENE e END_SCENE;
LIST_SCENES e END_LIST;
SET_CUE, SET_CUE_COND e SET_CUE_CICLIC;
FRAME, converte segundos para o número do quadro mais próximo e;
CLOCK e NO_CLOCK.

As diretivas temporais são usadas no texto descritivo das cenas, sob controle

desta :

At (t), After (t), Before (t) e Between (t);
Active_Cue (n) e Active_Cue_Until (n, t);
Kill_Cue (n) e Kill_Cue_At (n, t);

Além das estruturas e diretivas, o módulo *script* descreve as funções :

- *set_studio*, estabelece parâmetros, padronizados ou definidos pelo animador previamente, para todas as variáveis do sistema.

- *read_cfg*, lê um arquivo em particular que contém as definições das variáveis do sistema que identificam a preferência de algum animador ou para uma determinada cena.

- *write_cfg*, gera o arquivo de parâmetros especificado acima.

- *shoot*, executa o *pipeline* de visualização e chama o *rendering*.

- *apply_transformation*, aplica as transformações acumuladas pelas cenas aos atores devidos.

- *prep_to_render*, executa uma preparação dos dados para o *rendering*.

- *list_act_actors*, lista os atores ativos naquele instante (quadro).

- *list_act_decors*, lista os decorações ativas.

- *list_act_lights*, lista as luzes ativas.

- *reset_all_activities*, desativa todos os elementos da cena inicializando o próximo instante.

3.6 CONCLUSÃO

Descrever a animação é uma fase fundamental na produção. A adoção de um tratamento formal e não ambíguo (o *script*) pode parecer trabalhoso ou ineficiente mas, tem vantagens relevantes (documentação precisa, reusabilidade de partes de animações, etc.). Além disso, é mais importante criarmos um potencial de uso sobre a abstração feita para as entidades do sistema (como o ator) do que nos concentrarmos em interfaces amigáveis cujas entidades manipuladas sejam pouco expressivas (ver também ZELT/85, pg. 238).

Se algumas das fases da animação convencional forem utilizadas (como o *storyboard*, trilhas de exposição, etc. enfim, o planejamento progressivo) a confecção do *script* do TOOKIMA pode ser sensivelmente facilitada.

No tocante às Cues pode-se manipular arbitrariamente os seus limites visto que estes valores são acessíveis globalmente mas, estas manipulações alteram também o relógio local da cena, alterando diversos parâmetros pré-calculados para aceleração e trajetória desta. Estas alterações desviam o comportamento das ações planejadas, por isso aconselha-se o estudo cuidadoso e o uso criterioso destes recursos ou, usar somente as ferramentas de ativação e desativação mencionada antes neste capítulo.

Por fim, é interessante ressaltar que, o processo iterativo de avaliação consecutiva das cenas (o laço que garantirá a geração das várias imagens) é poupado ao usuário, estando embutido nas diretivas estruturais LIST_SCENES e END_LIST.

Nos próximos capítulos veremos quais elementos serão usados no texto do *script*, passíveis de serem animados além do mecanismo de geração do movimento e do sistema de visualização.

CAPÍTULO 4

TOOKIMA : ELEMENTOS DA CENA

4.1 INTRODUÇÃO

Este capítulo se concentra em esclarecer quais os elementos do ambiente virtual que estão disponíveis para serem animados.

Apresentam-se as concepções dos seguintes elementos gráficos disponíveis no TOOKIMA : Ator, Decoração, Câmera e Iluminação, enfatizando suas características, potencialidades e diferenças.

A entidade ator é um dos principais alvos da animação e, através dela, estenderemos os movimentos e ações (ver capítulo 5) a diversos outros elementos do ambiente virtual de síntese de imagens.

Também, faremos alusão a algumas características particulares da atual implementação de ator, em alto nível (isto é, não nos importarão os detalhes construtivos geométricos nem topológicos mesmo porque, intenciona-se estendê-los a vários modelos) e, das entidades que modelam a câmera sintética e as fontes de luz, bem como, as principais estruturas de dados, convenções e nomenclaturas referentes a estas entidades.

4.2 ATORES E DECORAÇÕES

4.2.1 DEFINIÇÕES

Ator é, no teatro e no cinema, o elemento que interpreta (representa) uma personalidade desejada, caracterizando alguma situação, alguma expressão. Esta caracterização depende muito da capacidade expressiva do ator (seus movimentos, p. ex.) e menos de sua aparência (seu sexo, altura podem ajudar mas, não são decisivos na caricaturização de um personagem).

Ator no TOOKIMA é um tipo de estrutura de dados que representa um elemento do nosso ambiente sintético que receberá "expressões" e "movimentos" no sentido de caracterizar (ou caricaturar) um personagem.

Na abordagem do TOOKIMA, o ator não possui movimentos e/ou leis próprias que lhe governem, diferentemente dos conceitos de ator encontrados no MIRA-3D (THAL/83) e no ASAS (REYN/82). Nossa implementação tem um paralelo mais próximo do conceito de "figuras" do MIRA-3D.

Nossa proposta de extensão da linguagem C se detém em estender a disponibilidade de tipos de dados e funções pré-definidas, não é uma extensão para a abstração de dados visto que a linguagem C não dispõe de mecanismos para esta finalidade (SILV/88, pg. 176). Esta última facilidade poderia ser alcançada com Orientação à Objetos ou simulada através ferramentas como YACC e LEX, criando uma extensão da sintaxe da linguagem C, facilidades estas propostas como um próximo trabalho.

Outro elemento que compõe o centro de atenções é a **decoração** (mobília ou cenário) que por sua vez tem uma expressividade limitada devido a, geralmente, ser um objeto inanimado (THAL/83, pg. 14).

Estes dois elementos (ator e decoração) são dois tipos de dados, disponíveis no TOOKIMA que representam a funcionalidade destes nas outras mídias (cinema, teatro, etc.):

■ Ator é o elemento componente da cena que tem expressividade e movimento, portanto, conforme o tempo e o roteiro, ele terá instâncias diferentes.

■ Decoração é o elemento componente da cena (um poliedro) que não tem movimento.

A expressividade que um ator sintético tem, está relacionada a sua forma, cor (expressões próprias do ator) e seus movimentos (discutidos no capítulo 5) adquiridos segundo um roteiro estabelecido por uma sequência de instruções (vistas no item 3.4).

Alguns outros detalhes concorrem para o aumento da expressividade dos atores mas estes, são sutís abordagens para outros aspectos do enredo (ver item 2.4).

4.2.2 TIPOS DE ATORES

A nossa representação virtual do mundo nos permite trabalhar com detalhes e/ou entidades desconsideradas no mundo real, mas, muito mencionadas nas áreas de modelagem geométrica como arestas, vértices, faces, etc.

Assim, nossa representação poliédrica (*B-rep*) para uma esfera nos permitirá exercer uma metamorfose nesta, apenas modificando o valor de um dos pontos da sua representação.

O que seria interessante pois, é que tanto a esfera quanto o ponto pudessem ser encarados como atores e que tivessem as mesmas funções de manipulação com a mesma funcionalidade.

Este então será o objetivo da abstração "atores": prover uma representação única, com poucas regras, para diversos tipos de dados que nada mais são do que, entidades do nosso ambiente virtual em vários níveis topológicos do modelo (*B-rep*) adotado. Além disso, eles devem ser tratados consistentemente (ver FOLE/90, pg. 404).

Para tal objetivo estão disponíveis no TOOKIMA atores que podem ser de vários tipos, os quais passamos a listar apresentando algumas das inúmeras aplicações possíveis :

□ DOUBLE, um valor real que pode ser usado para características desta natureza, tal como ângulo entre braço e antebraço. Pode representar também, um parâmetro do ambiente como a distância focal da câmera ("**d**").

□ POINT, um ponto com suas coordenadas euclidianas (x, y, z) que pode servir como referência para diversas transformações ou ainda, pode representar um ponto da estrutura de um "ator maior" (FACE ou POLY), que daí então poderia ter suas transformações próprias promovendo verdadeiras metamorfoses ou deformações sobre o "ator maior". Pode ainda representar a posição do observador no nosso modelo de câmera ou a posição de uma fonte de luz do modelo de iluminação. Estes últimos exemplos serão explorados em detalhe nos itens destinados a câmera e iluminação, respectivamente.

□ VECTOR, um vetor na origem, com suas coordenadas direcionais no espaço euclidiano que podem servir tanto como eixo para rotações livres quanto para influenciar os parâmetros do modelo de iluminação.

□ COLOR, uma cor, segundo a concepção R, G e B normalizada entre 0 e 60.000 pela convenção do ProSIm, e que pode representar qualquer cor no espectro visível. Serve para representar características do ambiente como cor de plano de fundo ou cor de uma das fontes de luz. É implementada funcionalmente como um POINT limitado e modificado.

□ FACE, não uma face solta no nosso espaço virtual mas, uma das faces de um ator do tipo POLY, disponível para permitir, por exemplo, transformações metamórficas.

□ POLY, um poliedro, é a representação do nosso modelo de objeto e ator. Sem dúvida, este será o tipo de ator mais usado na maioria das animações, pois o modelo já possui uma grande expressividade embutida que é a sua própria forma.

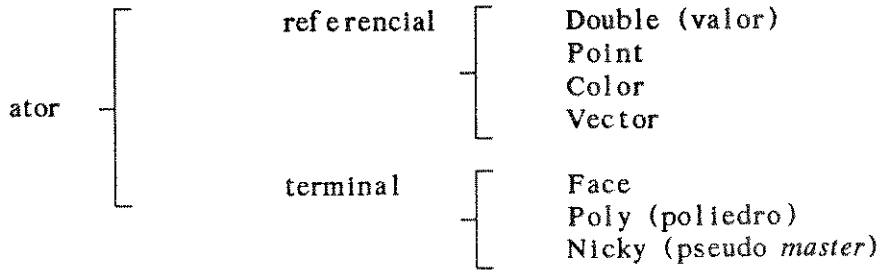
□ NICKY, disponível para "apelidar" ou, aglutinar, sob uma mesma referência pré-estabelecida, um conjunto de atores que tenham algum tipo de relacionamento entre si. O uso inteligente deste tipo de ator facilitará sobremaneira a descrição e animação de personagens complexos, compostos de diversas partes (como um corpo, ver RODR/92).

A disponibilidade de atores com DOUBLE, POINT, VECTOR, p. ex., deve-se em grande parte a uma tentativa de manter consistência e flexibilidade nos diversos níveis da modelagem (ou seja, nos diversos níveis de descrição de um objeto).

integridade geométrica (não planaridade de faces, não satisfação da fórmula de Euler (MORT/85)) em um ator do tipo POLY, de cuja estrutura topológica eles podem ter sido definidos. Isto ocorre porque o acesso e manipulação do animador a quaisquer elementos geométricos de um objeto é feita de maneira direta e irrestrita.

4.2.3 CLASSES DE ATORES

A nível de implementação trata-se distintamente os diversos tipos de atores sendo subclassificados internamente de dois tipos, referenciais e terminais, de acordo com o esquema abaixo:



Esta classificação interna se deve a três motivos principais:

1) princípio reativo;
2) consistência de hierarquia, e;
3) eficiência.

Em certos casos onde se trabalha com diversas visualizações de um mesmo elemento é desejavel que a alteração no valor ou na apresentação de uma de suas representações desencadeie uma atualização imediata na outra, visto que são um só. Este princípio é chamado de "princípio reativo" (TAKA/90).

Assim, usando o exemplo da esfera, é desejável que, por exemplo, se um determinado ator ponto (que faz parte da estrutura topológica da esfera) sofrer uma alteração geométrica, a esfera também sofrerá uma alteração na sua forma.

Este princípio reativo é alcançado de maneira restrita no nosso ambiente pois, os elementos que tem esta propriedade - doravante denominados referenciais - são tratados por referências (e não por valor), ou seja, como apontadores para suas instâncias, daí a reação ser estendida a nível de valores e não a nível de representação.

Quanto a consistência da hierarquia, esta deve garantir que os atores se comportem uniformemente e de acordo com o esperado dele. Daí, quando se usa um ator como fonte de referência para transformações, a qualquer instante, sobre outros atores e para que isto transcorra de maneira previsível e correta, o tratamento sobre os atores referenciais deve ser diferente dos terminais. Os referenciais precisam ter as suas instâncias (os valores de seus campos) atualizadas, caso contrário, a referência será diferente do esperado.

A ordem assumida para atualização das instâncias é função do tipo de ator (exerce influência direta no algoritmo de preparação para a visualização -*shoot*-) expressa a seguir :

DOUBLE, (COLOR e PONTO), VETOR, FACE, POLY e, NICKY.

Exemplificando a ordenação nas instâncias :

Suponha três atores, DOUBLE, POINT e FACE. O ator tipo DOUBLE é a coordenada x do ator tipo POINT que, por sua vez é um ponto da estrutura do ator do tipo FACE.

Suponha que o ator DOUBLE sofra uma translação, o ator POINT uma translação diferente e o ator FACE uma rotação sobre o ator POINT.

Se não estivesse implementado o princípio reativo então necessitaríamos alterar os valores dos atores, no algoritmo de preparação para o rendering, segundo a ordem supra citada.

A atualização das instâncias deve ser garantida mesmo quando o princípio reativo não precisar ser aplicado, ou seja, quando o ator POINT não tiver nada a ver com o ator FACE pois, estamos considerando um ponto que, pode servir como refêrencia para outro ator e transformação e que, deve estar com seu valor atualizado.

Por outro lado, os atores FACE, POLY e NICKY, doravante denominados terminais, estão com suas instâncias desatualizadas mas, com uma matriz refletindo (acumulando) todas as transformações que foram aplicadas a eles naquele lapso de tempo (quadro).

Esta matriz está contida na estrutura de dados *Actor*, discutida em 4.2.5, e é

chamada *atm*, *accumulated transformation matrix*.

Com esta matriz, atualiza-se as instâncias dos atores terminais antes da visualização (no algoritmo *shoot*) sendo que os atores FACE serão tratados antes dos atores tipo POLY e estes, antes dos atores do tipo NICKY.

Para um mesmo tipo de ator, existe precedência na atualização das suas instâncias segundo a ordem em que eles foram invocados no elenco (*cast*), os primeiros que foram definidos serão os primeiros a serem processados. Esta precedência deve sempre ser considerada em animações complexas, com muitas dependências e referências entre atores.

Como a matriz *atm* não é exclusiva dos atores terminais devido a estrutura de dados que representa o ator ser única, os atores referenciais, que são mantidos sempre atualizados, possuem *atm* igual à identidade (indicando que nenhuma transformação sobre o ator referencial deixou de ser aplicada à sua estrutura).

Este mecanismo de manter, ou não, atualizados os valores das instâncias dos atores conforme sua classe referencial ou terminal, respectivamente, é garantida através da aplicação ou armazenamento de *atm*, das transformações sofridas pelos atores.

Note que, a "eficiêcia" não é comprometida (ao contrário é enfatizada) com este tratamento, visto que, os atores referenciais são estruturas com poucos elementos o que torna rápida qualquer transformação sobre eles.

Detalhe: No caso de um ator do tipo ponto, considerando que as transformações são representadas por matrizes 4x4, é mais rápido ir aplicando a matriz ao ponto e atualizando o seu valor do que ir acumulando as transformações (que internamente é a multiplicação das duas matrizes) e depois aplicar a transformação resultante (matriz acumulada) ao ponto.

Por outro lado, se a mesma idéia fosse aplicada a atores terminais (que são equivalentes a centenas ou milhares de atores referênciais) tornaria as transformações sobre estes, ineficientes.

Assim, para os atores terminais, é melhor termos uma matriz que acumule as transformações sobre este e, antes de os passarmos para a visualização, efetuarmos uma

única transformação.

Para a avaliação de alguma característica do ator terminal (centro de gravidade, p. ex.) que seja bastante custosa, a função deverá calcular uma única vez e, se valer da matriz acumulada (*atm*) para atualizar este valor, se isto for possível, para se alcançar maior eficiência.

4.2.4 HIERARQUIA DE ATORES

A fim de melhor estruturar um personagem e mesmo uma animação, ou para definir uma associação entre atores, dispõe-se de um mecanismo para atribuir hierarquia entre atores. A idéia de hierarquia aqui, resume-se em associar as transformações aplicadas a um ator a apêndices definidos para ele, propagando em profundidade (para os apêndices) as ações que porventura o ator venha a sofrer.

Um ator pode ter associado a ele vários outros atores, de diversos tipos, cujos movimentos destes últimos podem ser de três tipos: adquiridos em função da hierarquia, próprios devido ao enredo, ou uma mistura das duas opções anteriores.

O meio de estabelecer a hierarquia entre atores é através das funções *group* e *grasp*, que são responsáveis pelo "agrupamento" ou "atracamento" dos atores. A diferença básica entre estas funções é que a segunda pode ser aplicada durante a animação e pode ser desfeita, enquanto que a primeira é usada para definir um personagem como uma aglutinação de partes (atores) de maneira permanente, na fase dos bastidores (*wings*, ver item 3.4.6).

A hierarquia estabelece um ator como principal (denominado *master*) e outro(s) a ele atracado(s) e dependente(s) (denominado *slaves*), onde as transformações aplicadas a um *master* são propagadas aos seus respectivos *slaves*. E, também, declarações de que um ator *master* participa de uma cena implica que seus *slaves* também participam.

Esta configuração é expansível na medida que os *slaves* passam a ter seus próprios *slaves*, fazendo fluir para estes as transformações herdadas, e assim sucessivamente.

Este esquema fica melhor representado numa árvore como no exemplo abaixo:

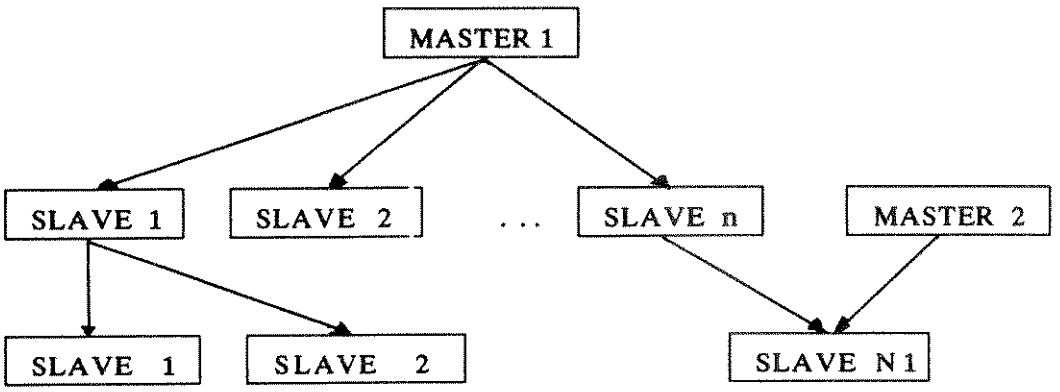


Figura 4.1 : Hierarquia de Atores.

Onde cada nó ou folha corresponde a um ator (com o nome e um elemento associado) e cada arco a uma relação de dependência (a seta indica a direção de propagação das transformações).

A hierarquia pode ficar bastante complexa se um ator for *slave* de vários *masters* (que é permitido, se for esta a necessidade), porém não recomendamos pois em certos casos ficaria muito complicado de prever os movimentos que o *slave* executaria (imagine na figura acima se *master2* fosse *slave* de *master1*), devido a ele herdar transformações de mais de uma fonte.

Esta flexibilidade é bastante intuitiva e útil, mas, deve ficar bem claro que, o que é herdado é a matriz de transformação sobre o *master* e não a transformação em si, ou seja, passa-se a considerar o *slave*, topologicamente incorporado ao *master* na aplicação da transformação mas, não no cálculo de possíveis referências.

Exemplificando: Um relógio com dois ponteiros. O corpo do relógio (master) tem os ponteiros como slaves dependentes dele. A rotação relativa do corpo transformará os ponteiros como se estes fossem parte integrante topologicamente do corpo mas, o referencial da rotação, o *cg-actor* do corpo do relógio não é influenciado pelos ponteiros.

Quando se deseja criar um referencial diferente para um ator, ou varios atores, que não seja o seu centro de gravidade então define-se um ator do tipo NICKY e daí, passa-se a trasformar e referenciar este(s) ator(es) através do NICKY.

Um ator NICKY não incorpora nenhuma estrutura (como poliedros, nem faces, nem

pontos, p. ex.), ele serve apenas para mudança de referência de atores ou para aglutinar um conjunto de atores sob uma única esfínge (o nome do ator do tipo NICKY, que será o *master* destes atores).

Existem certas situações interessantes quanto à hierarquia entre terminais e referenciais que devem ser conhecidas e consideradas pelo animador (relembramos que nosso sistema é caracterizado pelo acesso a qualquer elemento construtivo dos atores permitindo uma flexibilidade extrema mas, exigindo cuidado, para que esta facilidade não leve a inconsistências geométricas ou topológicas no ator).

Por definição, todos os atores são independentes de quaisquer outros, entretanto;

1) Atores referenciais que fazem parte da estrutura de um ator terminal, comportam-se como se estivessem atracados (lembre-se do Princípio Reativo). Um ator FACE é definido normalmente desatracado do seu respectivo POLY.

2) Atores terminais que são atracados a atores referenciais devem ser cuidadosamente projetados pois cada referencial tem uma forma de propagar as transformações sobre ele, para seus *slaves*.

Ex: translação de um ator VECTOR, não acontece nada nem no master nem nos slaves.

Assim, as transformações são propagadas segundo a influência que elas exerceram sobre os *master* e não sobre o que deveria acontecer com os *slaves*.

Os atores do tipo NICKY comportam-se como se fossem do tipo POLY na maioria das situações.

4.2.5 ESTRUTUTURA DO MÓDULO *ACTOR*

Os conceitos acima descritos estão embutidos no módulo *actor*, composto do cabeçalho actor.h e diversos arquivos de programa (com extensão .c), onde estão os códigos das funções deste módulo (todos os arquivos são identificados por começarem com "a"), os quais passamos a discutir.

Para implementar as características de eficiência, princípio reativo, hierarquia

e outras, definiu-se a seguinte estrutura de dados para descrever um ator :

```
typedef struct {String     *name;
                Boolean    activity;
                ValueType  t;
                Element    *master;
                Matrix     atm; /* Accumulated Transformation Matrix */
                int        n_slaves;
                int        slaves[ N_MAX_SLAVES ];
                Point      gc;
                Boolean    gc_changed, gc_calculated;
                } Actor;
```

Onde:

- *name* é uma cadeia (*string*), que representa o nome do ator;

- *activity* é um Boolean, que identifica se o ator está ativo, em pelo menos uma das cenas avaliadas no respectivo instante;

- t é um *ValueType*, que identifica o tipo do ator;

- *master* é uma referência para *Element*, que conterá efetivamente a estrutura geométrica e topológica do ator;

- *atm* é uma Matrix, a matriz que acumula as transformações sobre o ator num intervalo de 1/*Rate* segundos.

- *n_slaves* é um inteiro e indentifica o número de *slaves* (dependentes) do ator;

- *slaves* é uma lista de referências para atores e identifica os atores escravos;

- *gc* é um ponto e identifica a referência básica para acesso ao ator e que determinará as suas transformações relativas. Está associado a *gc_calculated* e *gc_changed*.

As estruturas *ValueType* e *Element*, são definidas da seguinte forma :

```
typedef enum { DOUBLE, POINT, VECTOR, COLOR,
               FACE, POLY, NICKY;
             } ValueType;

typedef union  {double      *d;
                Point       *p;
                Vector      *v;
                Color       *c;
                FacePlus    *f;
                PolyPlus    *o;
             } Element;
```

Sendo que esta última usa os seguintes tipos de dados, também definidos no módulo *actor*:

```
typedef struct  {int  a;    /* Original actor (Polyhedrical one) */
                 int  fn;   /* Face number in the polyhedric struct */
                 int  nvert; /* number of vertices */
             } FacePlus;

typedef struct  {Polyhedron     *p;
                 Surface        *v;
                 ShadeType      s;
                 int            a_list, r_list;
                 int            size_list;
                 Wireframe      wire_list;
                 Boolean        wf_calculated;
             } PolyPlus;
```

Devido ao elemento decoração ser definido neste módulo, abaixo apresentamos sua estrutura de dados, juntamente com o tipo que estabelece a referência para as TGLR sobre um ator:

```
typedef struct  {String     *name;
                 Boolean    activity;
                 PolyPlus   *decor;
             } Decor;
```

typedef enum { ABSOLUTE, RELATIVE } A_R;

Além destas estruturas e tipos de dados, o módulo *actor* incorpora diversas funções para definição de tipos e atores, agrupamento e transformações abaixo listadas com os respectivos arquivos de programa (extensão .c).

Em um arquivo deste módulo, o a_define.c, estão as funções de definição do elenco, decorações, pintura e participação destes nas cenas:

- *cast_nicky*, *cast_poly*, *cast_face*, são as chamadas para definir atores terminais do tipo NICKY, POLY e FACE, tendo como parâmetros para a função o nome do arquivo do poliedro (com extensão .brp - padrão ProSIm), o tipo de tonalização inicialmente definida para o ator e o nome a ele atribuido durante a descrição;

- *cast_point*, *cast_vector*, *cast_color*, *cast_double*, são as chamadas para definir atores referenciais;

- *furniture*, define uma decoração através do nome desta e do arquivo do poliedro;

- *paint_actor*, *paint_decor*, atribuem características físicas visuais para as superfícies de um ator e decoração, respectivamente, para serer usadas no modelo de iluminação, através de uma referência para o elemento e o nome do arquivo de características (com extensão .cor - padrão ProSIm);

- *part*, *decor*, determinam, no interior de uma cena, qual(is) atores e decorações participarão da cena (e portanto, estarão ativos);

- No arquivo a_miscel.c temos as seguintes funções denominadas de miscelâneas para este módulo:

- *free_face*, inicializa um ator do tipo face, liberando o polígono em questão da estrutura do poliedro;

- *id_actor* (*id_decor*) identifica numerica e internamente um ator (ou decoração);

- *grasp* e *group*, estabelecem a hierarquia entre atores durante o decorrer de uma cena e, na inicialização, respectivamente;

• *ungrasp* e *grasped*, desfaz e testa, respectivamente, se um determinado ator está hierarquizado a outro;

• *gc_actor*, avalia o centro de gravidade de um ator (baricentro);

• *transf_actor*, efetua uma transformação sobre um ator, a partir de uma matriz homogênea como parâmetro;

• *broad_actor*, propaga para os *slaves* de um ator, as transformações a ele aplicadas e assim, recursivamente em profundidade;

• *shade_actor*, manipula o tipo de tonalização (*shading*) de um ator.

Nos arquivos a_transl.c a_scale.c e a_rotate.c estão as funções para as TGRL (ver item 5.2) sobre um ator:

• *translate_actor_x*, *translate_actor_y*, *translate_actor_z*, transladam um ator absoluta ou relativamente de uma certa quantidade em um dos eixos euclidianos do WC, respectivamente em x, y e z;

• *translate_actor*, efetua a translação de um ator, também absoluta ou relativamente mas, nos três eixos de uma vez;

• *scale_actor_x*, *scale_actor_y*, *scale_actor_z*, efetua um escalamento sobre um ator, absoluta ou relativamente, de uma certa percentagem, em uma das direções euclidianas;

• *scale_actor*, efetua a escala sobre um ator mas, nos três eixos concomitantemente;

• *growth*, *shrink*, são semelhantes à anterior mas, os fatores de escala são iguais nos três eixos produzindo um efeito de crescimento e diminuição, respectivamente;

• *rotate_actor_x*, *rotate_actor_y*, *rotate_actor_z*, são capazes de efetuar a rotação de um ator, absoluta ou relativamente, de uma certa quantidade de graus, sobre um dos eixos euclidianos;

- *rotate_actor*, efetua a rotação de um ator sobre um eixo qualquer passando pela origem, especificado por uma direção (vetor);

- *free_rotate_actor*, efetua uma rotação sobre um eixo arbitrário, passando por uma referência (ponto) e com uma direção (vetor), especificados.

O TOOKIMA define pois, a sua entidade principal para animação como um ator, esta definição abrange diversos tipos de entidades do mundo virtual em questão, de maneira consistente e flexível.

Dentre os vários tipos de atores acima citados, seguramente os tipos POLY e NICKY serão os mais amplamente utilizados pois são os próprios objetos modelados. Os demais são extensões para necessidades específicas e mais complexas.

A correta estruturação de um personagem, com uma estrutura hierárquica bem montada, pode facilitar a documentação e a própria descrição da animação.

O ator é, decisivamente, o elemento que concentrará grande parte das atenções de uma produção, nas fases de criação e movimentação.

4.3 CÂMERA

Outro elemento que compõe a cena mas que não é visível na imagem como os atores, as decorações e os movimentos é a câmera sintética que, apenas influencia o modo de visualizar.

Este elemento da cena é um recurso muito interessante da síntese de imagens e, por vezes, pode "roubar a cena", isto é, pode produzir animações inteiras mesmo que os objetos estejam sem movimento.

Apresentaremos então, o modelo de câmera sintética adotada no TOOKIMA (lembre que AMC sugere modelos formais para todos os elementos da cena), enfatizando seus parâmetros, as transformações a ele aplicadas, suas variações e diferenças, terminando o item com a estrutura do módulo *camera*.

4.3.1 MODELAGEM DA CÂMERA

Câmera sintética é um conjunto de dados e procedimentos que permitirá a visualização da cena a partir de qualquer posição.

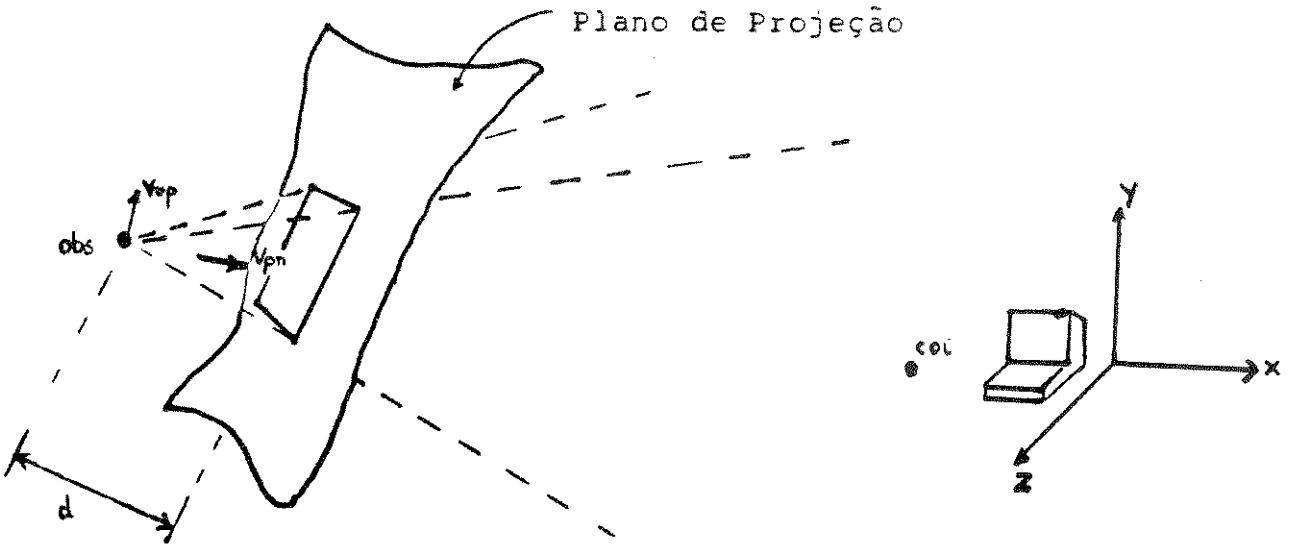


Figura 4.2: O Modelo de Câmera

O modelo de câmera utilizado tem os seguintes parâmetros (ver figura 4.2):

- *obs*, é a posição onde se encontra o olho do observador e, quando na projeção perspectiva, é o centro de projeção;

- *coi - center of interest* -, é o centro de atenção do observador, um ponto da cena para o qual ele está olhando e onde, possivelmente, estará um ator de interesse;

- *vup - view up vector* -, é a direção que orienta a "cabeça" do observador, identificando o que é "para cima";

- *view_plane* - plano de projeção ou visualização, onde são projetados os objetos;

- **d** - distância focal -, é a distância entre o observador e o plano de visualização (*view_plane*).

Além destes parâmetros, todos em coordenadas da mão direita (*hight_hand*), define-se o enquadramento da imagem através dos parâmetros:

- *window* - janela -, porção retangular do *view_plane* que delimita a imagem de interesse e a pirâmide de visualização usada na perspectiva;

- *viewport* - porta de visualização -, é a porção retângular da tela do monitor que será efetivamente usada para expor a imagem contida na *window*. Esta é dimensionada em *pixels* e define as coordenadas bidimensionais do dispositivo (DC - Device Coordinate), chamadas de u e v.

Define-se **vpn** como o vetor que identifica a direção de visualização e dada por (*vpn = coi - obs*).

Os elementos acima, parâmetros do modelo da câmera e parâmetros do enquadramento (*viewport* e *window*), acrescidos do tipo de projeção a ser realizada, paralela ortogonal ou perspectiva, são dispostos numa estrutura de dados, denominada Camera, capaz de modelar as seguintes funcionalidades da "câmera sintética":

1) Permitir a visualização arbitrária;
2) Executar a projeção da imagem;
3) Estabelecer qual parte da imagem será mostrada e em que parte da tela.

Para que esta estrutura de dados realmente alcance a funcionalidade desejada é necessário usá-la numa sequência ordenada de procedimentos denominada "*pipeline* de visualização", como mostra a figura 4.3.

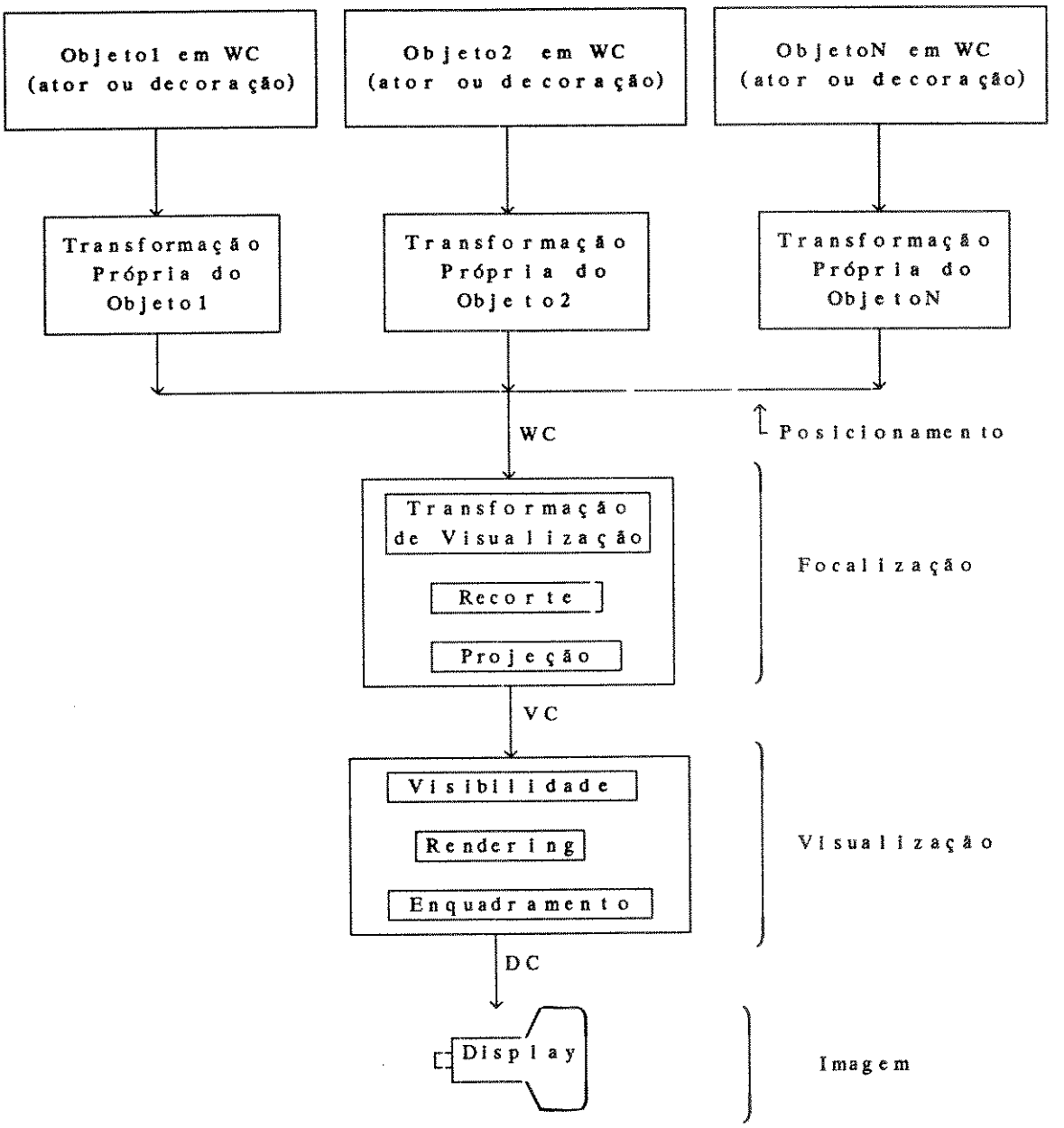


Figura 4.3 : *Pipeline* de Visualização.

Onde:

WC, *World Coordinate*, sistema de coordenadas do mundo, absoluto, do objeto;

VC, *View Coordinate*, sistema de coordenadas de visualização;

DC, *Device Coordinate*, sistema de coordenadas do dispositivo de visualização.

Como pode ser inferido da figura, existem três grandes blocos na sequência (POSICIONAMENTO, FOCALIZAÇÃO e VISUALIZAÇÃO) para gerar a Imagem:

O bloco POSICIONAMENTO define uma transformação geométrica aplicada a cada objeto em particular, oriunda do desejo de manipular sua posição (daí o seu nome) sendo diferente, *a priori*, para cada objeto. Esta transformação não muda o sistema de coordenadas do objeto ou, coordenadas do mundo (WC).

Já o bloco FOCALIZAÇÃO é responsável pelas funcionalidades 1 e 2 (ver pag. 106) da câmera, representadas nos subblocos TRANSFORMAÇÃO DE VISUALIZAÇÃO e PROJEÇÃO respectivamente.

É este bloco que determina, a gosto do animador, o quê e para onde está sendo direcionada a câmera efetivamente (o termo foco aqui não tem significado semelhante aos focos das câmeras fotográficas mas, foi adotado por falta de uma opção melhor).

O subbloco RECORTE (ainda na FOCALIZAÇÃO), é responsável pela eliminação das linhas e/ou polígonos que estejam fora do campo visual do observador ou atrás dele. Esta eliminação se faz necessária principalmente no caso da projeção perspectiva, porque essa transformação é simétrica em relação ao centro de projeção (no caso, a origem do sistema euclidiano) produzindo deformações nos objetos que estejam parcialmente à frente (z positivo) e parcialmente atrás (z negativo) do observador.

O recorte não é efetivamente feito nas direções x e y devido as características de implementação do bloco de visualização que se encarrega desta etapa. Entretanto o recorte na direção z deve efetivamente ser programado pois, não é considerado nem na câmera, nem no algoritmo de *rendering*.

A idéia empregada no subbloco da TRANSFORMAÇÃO DE VIZUALIZAÇÃO, amplamente difundida na literatura (ver NEWN/79 e FOLE/90), é determinar a transformação de coordenadas do mundo (WC) para as coordenadas de vizualização (VC).

Isto é feito concatenando-se as transformações que levam o observador para a origem do WC garantindo que *vup* coincida com o eixo Y e que *vpn* coincida com o eixo Z. Estas transformações são acumuladas em uma matriz homogênea denominada *NormalizedView*, que é uma variável global do sistema (isto é feito pelo procedimento "camera").

Assim, aplica-se *NormalizedView* aos objetos da cena para obtermos o efeito da vizualização arbitrária a partir de um determinado "ponto de vista".

O bloco VISUALIZAÇÃO, dividido nos subblocos VISIBILIDADE (ou *HLHS - hidden line and hidden surface*), RENDERING e ENQUADRAMENTO, tem a função determinar quais objetos e quais partes destes estão visiveis para o observador, com que características visuais (cor, transparência, etc.), em qual área da superfície de visualização (*view_plane*) e onde será exibida no dispositivo de saída, respectivamente.

Os dois primeiros receberão maiores atenções no capítulo 6 pois constituem parte importantíssima no sentido de criar realismo e aparência visual agradavel à imagem.

O subbloco de ENQUADRAMENTO implementa a funcionalidade 3 (ver pg. 106) da câmera sintética antes de enviar o objeto para o vídeo, efetuando uma outra mudança de coordenadas, desta vez bidimensional, das coordenadas de visualização (VC) para as do dispositivo de saída (DC).

4.3.2 TRANSFORMAÇÕES SOBRE A CÂMERA

Para manipulação da câmera dispomos de três conjuntos de transformações:

1) Inicialização;
2) Transformações bem Conhecidas;
3) Transformações Extras.

As INICIALIZAÇÕES encarregam-se de estabelecer os valores dos campos da estrutura de dados da câmera e, devidamente relacioná-los com outros campos se necessário for. Existe pelo menos uma função de inicialização para cada parâmetro, todas estando listadas no item 4.3.4.

Por TRANSFORMAÇÕES BEM CONHECIDAS, entenda-se algumas das transformações usuais aos animadores convencionais e/ou cinema. Dentre estas destacam-se:

- *Zoom_in (out)*, capacidade de aumentar (diminuir) os detalhes de parte de uma cena. Isto é feito aproximando-se (afastando-se) o observador do centro do interesse;

- *Pan right (left)* - panorâmica -, varre horizontalmente para a direita (esquerda) o campo visual;

- *Tilt up (down)*, varre verticalmente para cima (baixo) o campo visual;

- *Spin Clock (clock)*, inclinação da câmera no sentido horário (anti-horário). Modifica-se a direção indicada por *vup*;

- *Travelling* - carrinho -, a direção e inclinação da câmera permanece constante mas, a posição modifica. Isto é conseguido movendo-se o centro de interesse da mesma quantidade que a posição do observador.

TRANSFORMAÇÕES EXTRAS são oriundas de uma extensão às manipulações sobre a câmera. Foram considerados o observador (*obs*) e o centro de interesse (*coi*) como origem de dois tipos de sistemas de coordenadas: quadrado e esférico sendo que um serve como parâmetro referencial para o outro. Assim, estabeleceu-se a seguinte semântica:

- *go* - vá -, identifica manipulação com o observador;

- *aim* - focar -, identifica manipulação com o centro de interesse (*coi*);

- *up, down, right* e, *left* (para cima, para baixo, direita e esquerda), movimentos em relação ao sistema definido sobre um cubo (ver figura 4.5);

- *north, south, east* e, *west* (norte, sul, leste e, oeste), movimentos em relação ao sistema de coordenadas esféricas (ver figura 4.4);

Para completar o repertório de operações sobre estes dois tipos de referenciais de coordenadas (quadrado e esférico), temos ainda as seguintes funções:

- *go_forward (go_backward)* são idênticos a *zoom_in (zoom_out)*, modificando a posição do observador, para mais próximo (longe) da cena (do *coi*);

- *aim_in (aim_out)* são semelhantes a *zoom_in (zoom_out)*, no efeito mas, modificam a posição do *coi* e não do *obs*.

Exemplificando, go_north (vá par o norte) indica o movimento do observador sobre uma esfera com o centro sobre o coi na direção do vup (ver figura 4.4).

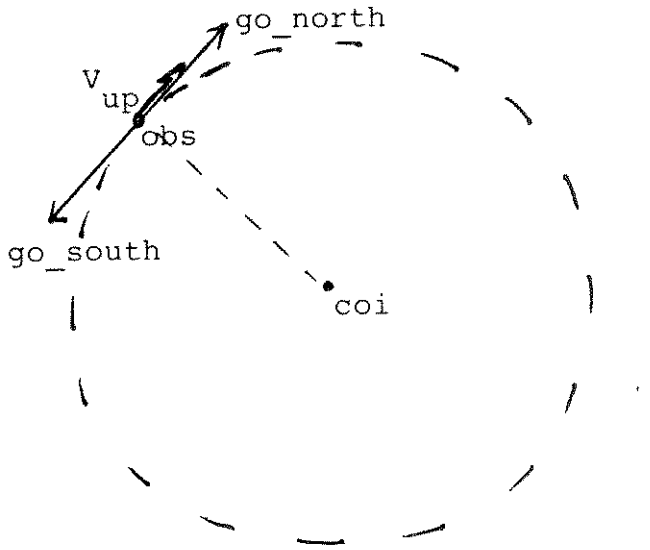


Figura 4.4 : *Go_north* com centro em *coi*.

E, aim_right (focaliza à direita) indica o movimento do centro de interesse, sobre um cubo com centro no obs, para a direita do observador (ver figura 4.5).

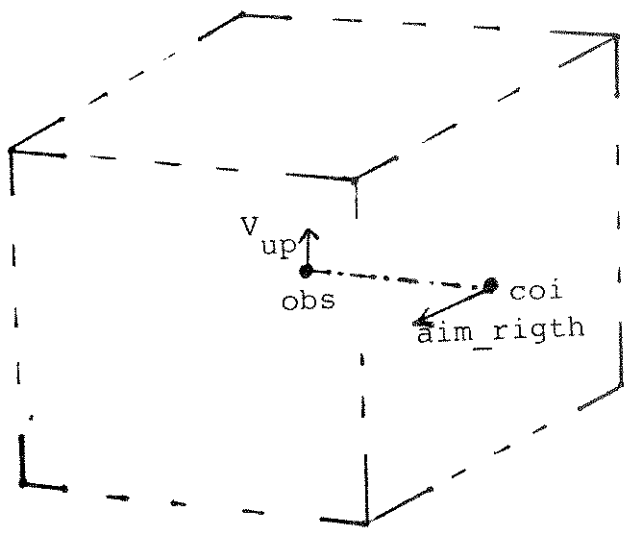


Figura 4.5 : *Aim_right* com centro no observador.

Além das manipulações acima, a nossa experiência detectou outras funções muito úteis e usuais para a câmera e que estão incluidas como transformações extras:

• *go_flying* (vá voando), representa o movimento do observador sobre uma trajetória qualquer de forma que sua direção de visualização (*vpn*) mantenha-se na direção do movimento (ou seja, tangente a este), semelhante ao que acontece com um observador piloto de uma aeronave (daí o termo voar);

• *go_crow*, que, mantendo a síntaxe já estabelecida, produz o efeito do carrinho (*traveling* como querem os animadores convencionais);

• *aim_actor* (*aim_scene*), estabelece o interesse de manter a atenção (o foco, o *coi*) voltado para um ator específico (ou para a cena em geral, procurando enquadrar todos os atores).

Note que muitas das transformações sobre *obs* tem uma semelhante sobre *coi* mas, o que realmente ocorre é diferente; acompanhe o exemplo.

Suponha que as transformações *go_north* e *aim_south* aplicadas a cena e sobrepostas na figura 4.6.

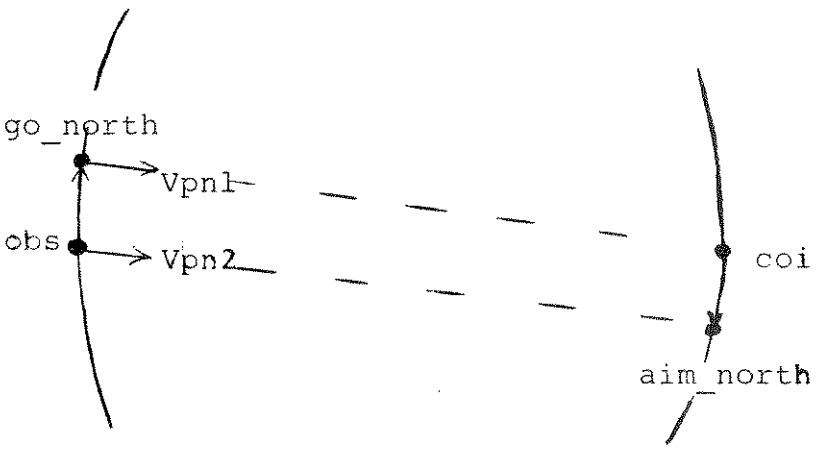


Figura 4.6 : *Go*'s são diferentes de *aim's*.

Verifique que as direções de *Vpn1* (obtidos após o *go_north*) e *Vpn2* (obtido após *aim_south*) são iguais mas a porção da cena que será visualizada será diferente pois os *Vpn*'s estão em posições distintas.

4.3.3 ELEMENTOS DA CÂMERA COMO ATORES

Os parâmetros da câmera não são manipuláveis apenas pelas funções acima citadas, que apesar de serem uma gama razoavelmente grande de opções, podem inibir a sua utilização por terem uma sintaxe própria.

Caso o animador escolha, cada um dos campos da estrutura de dados da câmera pode se constituir em um ator distinto e, desta forma adquirir todas as características deste último (trajetórias, acelerações, hierarquias, etc.), o que torna uniforme o tratamento destes elementos da cena, facilitando a assimilação e o uso por parte do usuário-animador.

Assim, por exemplo, obs e coi podem ser atores do tipo POINT, vup um ator do tipo VECTOR e "d" um ator do tipo DOUBLE, etc.

Desta feita, as TGLR (Transformações Geométricas Lineares Rígidas, ver item 5.2) podem ser aplicadas ao observador, transladando-o, por exemplo, bem como fazer com que um ator siga um caminho segundo uma aceleração especificada (following_track), etc.

4.3.4 ESTRUTURA DO MÓDULO *CAMERA*

A câmera sintética está definida no módulo *camera*, no qual podemos encontrar a definição do tipo de dados, denominado Camera e as diversas funções de manipulação apresentadas acima, sendo que estas funções estão nos arquivos programas (com extensão .c) identificados pelos nomes iniciando com "c_".

A estrutura de dados Camera é um aglomerado direto dos parâmetros vistos no item 4.3.1:

```
typedef struct { /*------ Observer and Main Parameters -------*/
                 Point  obs,
                        coi,
                        vup;
                 double d;
```

```
        /*----- Window in the ViewPlane ----------*/
        double u_min, u_max,
               v_min, v_max;

        /*----- Viewport on the screen ----------*/
        int x_corner, y_corner,        x_length, y_length;

        /*----- Projection parameter ----------*/
        ProjectionType prj;
    } Camera ;
```

Onde o tipo de projeção está especificado na estrutura abaixo.

typedef enum { ORTHO, PERSP } *ProjectionType*;

Para manipular os parâmetros da câmera tem-se várias funções que estão dispostas abaixo, conforme foram discutidas no item 4.3.2. O arquivo c_define.c, p. ex., contém as funções de inicialização:

- *set_obs*, *set_vup*, *set_coi*, *set_d*, *set_viewport*, *set_window*, *set_ortho*, *set_persp*, são as funções de inicialização básicas;

- *set_aperture*, estabelece a *window* por um ângulo de abertura da pirâmide de visualização que ocorre quando da projeção de perspectiva;

- *set_cam_defaults*, inicializa toda a estrutura da câmera sintética com valores padrões;

- *camera_list*, lista, na fase de preparação para o *rendering*, todos os parâmetros da Camera.

No arquivo c_observ.c estão dispostas todas as funções para manipulação do observador : *go_forward*, *go_backward*, *go_up*, *go_down*, *go_right*, *go_left*, *go_north*, *go_south*, *go_east*, *go_west*, *go_flying* e *go_crow*.

Semelhantemente, o arquivo c_aimpnt.c, contém as manipulações sobre o *coi* (também denominado de *aim point*) daí o nome do módulo: *aim_up*, *aim_down*, *aim_in*, *aim_out*,

aim_right, *aim_left*, *aim_north*, *aim_south*, *aim_east*, *aim_west*.

Que se completam com o arquivo c_mix.c que contém as funções *aim_actor* e, *aim_scene*, que se optou por deixá-lo separado pois, somente estas funções levam em consideração o tipo de objeto para influenciar os parâmetros da câmera sintética, daí diz-se que este é responsável por misturar (daí o nome do arquivo) o módulo *camera* com a estrutura *Actor*.

As operações bem conhecidas pelos animadores são acessadas pelas funções do arquivo c_standr.c que, por vezes são implementadas pelas funções dos arquivos c_observ.c e c_aimpnt.c : *zoom_in*, *zoom_out*, *spin_clock*, *spin_Cclock*, *tilt_up*, *tilt_down*, *pan_right*, *pan_left*, *do_ortho*, *do_persp*, *traveling*.

Outros arquivos completam este módulo mas, é suficiente ressaltar o arquivo c_arbview.c que contém a função que calcula a transformação de visualização (gerando a matriz *NormalizedView*) do bloco FOCALIZAÇÃO (ver figura 4.3), permitindo a visualização arbitrária (daí o nome do arquivo), dentre outras coisas.

Concluindo pudemos perceber, através deste item, que a modelagem da câmera sintética é muito importante e produz efeitos interessantes para a cena. A ênfase aqui demonstrada se deve a grande utilidade deste recurso tanto para imagens estáticas quanto para animações, além de se constituir, hoje em dia, um indispensável recurso nos pacotes de Computação de Imagens.

As funções apresentadas incidem diretamente sobre os parâmetros do modelo e cada qual produz um efeito visual diferente. A integração da câmera com o elemento ator dá uma nova dimensão e "vida" à estas entidades tornando-se uma facilidade de significativo interesse.

4.4 ILUMINAÇÃO

Este item discute como é simulada e tratada a iluminacão no ambiente TOOKIMA, sem no entanto aprofundar-se na teoria de iluminação. A escolha do modelo baseou-se na disseminação dele pela literatura (conhecido como um bom modelo quanto a sua relação entre qualidade perceptual e tempo dispendido).

Em seguida dispõe-se sobre os tipos de fontes luminosas integradas a este modelo, o porquê das fontes serem consideradas pontuais, sua integração ao ambiente, terminando-se com a apresentação das principais estruturas de dados definidas neste módulo.

4.4.1 MODELOS DE ILUMINAÇÃO

Um modelo de iluminação é uma formulação matemática que nos permitirá calcular a influência das fontes de luminosidade sobre as superfícies dos objetos (atores e decorações) da cena e nos *pixels* produzidos por estes objetos na tela, aumentando consideravelmente o realismo da imagem.

Quando a energia radiante atinge a superfície de um objeto, ocorre uma interação de modo que parte da energia luminosa retorna ao meio de propagação, fenômeno chamado de reflexão e, conforme o material, parte da energia se propaga através do material, fenômeno chamado de refração (discutido com detalhes no item 6.3.2).

A influência da energia refletida pode ser dividida em três componentes : ambiente, especular e difusa (GOME/90, pg. 197).

A COMPONENTE AMBIENTE, também chamada de luz de fundo, incide e é refletida em todas as direções pelas superfícies. Representa a influência de uma fonte de luz completamente distribuida pelo ambiente, ou seja, não é originária de uma fonte de luz única e sim, da complexa interação e troca de energia luminosa entre todos os elementos da cena (fontes de luz e superfícies).

Esta componente é a mais "cara" de ser calculada mas, produz resultados fantásticos. Isto pode ser sentido pelos tempos demandados nos algoritmos chamados de *radiosidade*, que se dedicam a calcular a componente ambiente e a componente difusa com exatidão (ver GORA/84 e COHE/85). Entretanto, no modelo aqui implementado, a componente ambiente é considerada uma constante.

A COMPONENTE DIFUSA é uma componenete uniforme e em todas as direções, proveniente da incidência de uma fonte de luz sobre uma superfície que, microscopicamente, nunca é totalmente polida. Esta componente depende do polimento, do material da superfície e do comprimento de onda da luz incidente.

COMPONENTE ESPECULAR, esta componete representa a quantidade de energia refletida por uma superfície na direção de reflexão. É dada pela lei de Snell (ângulo de incidência, ⊖i, é igual ao angulo de reflexão, ⊖r), conforme ilustra a figura 4.7. Esta componente é relacionada com o brilho (*highlight*) que aparece sobre uma superfície, normalmente da cor da luz incidente.

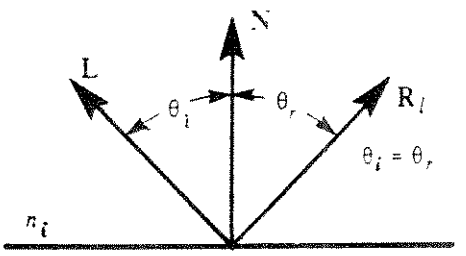


Figura 4.7 : Lei de Snell da reflexão (GOME/90).

Quanto mais espelhada a superfície, maior será a contribuição especular, produzindo um brilho maior que pode ser mais ou menos concentrado, dependendo do material.

Cada componente acima tem associada, para cada objeto, um parâmetro que ponderará a influência da energia incidente (proveniente de cada fonte de luz), para produzir a cor sobre a superfície.

A cor de um objeto será representada por estes parâmetros de influência : ambiente, difusa e especular, para cada uma das componentes primárias de cor,

reproduzíveis em um monitor (estas são conhecidas pela tríade R, G e B, letras indicativas das cores vermelho, verde e azul em inglês - *red, green, blue* -).

Portanto, o modelo de iluminação será uma função matemática que leva em consideração os parâmetros de cor do objetos e o posicionamento da fonte de luz em relação ao ponto iluminado para cada uma das componentes mencionadas.

O modelo considerado no TOOKIMA e' o modelo de iluminação de Phong (ver PHON/75, ROGE/85, pg.340 e HALL/86), reproduzido na equação 4.1.

$$I = Ia\ ka + f(d) \sum_{j=1}^{ls} I(j)\left(kd\ \cos(\ominus j) + ks\ \cos^{n}(\alpha j)\right) \qquad (4.1)$$

Onde **I**, é a intensidade luminosa resultante, naquele ponto;

Ia, é a intensidade luminosa do ambiente;

ls, é o numero de fontes de luz;

I(j), é a intensidade luminosa produzida pela fonte j;

f(d), é a influência da distancia sobre a fonte de luz;

ka, é o coeficiente de cor de luz ambiente, do objeto;

kd, é o coeficiente de cor de luz difusa, do objeto;

ks, é o coeficiente de cor de luz especular, do objeto;

n,é o parâmetro que controla a concentração do brilho;

$\ominus j$, é o ângulo de incidência do raio luminoso;

αj, é a diferença entre a normal da superfície e o vetor bissetor **H** (definido a meio caminho entre o vetor de visão, **V**,e o de iluminação, **L**, do ponto em questão) (ROGE/85, pg. 176).

A formulação acima pode ser melhor interpretada se levarmos em consideração a análise da figura 4.8 que ilustra a geometria da iluminação em um ponto de uma superfície.

Como a cor é discretizada em três componentes primárias, a equação 4.1 deve ser aplicada ao ponto, levando-se em consideração estas três componentes (ou seja, temos que avaliar a equação três vezes).

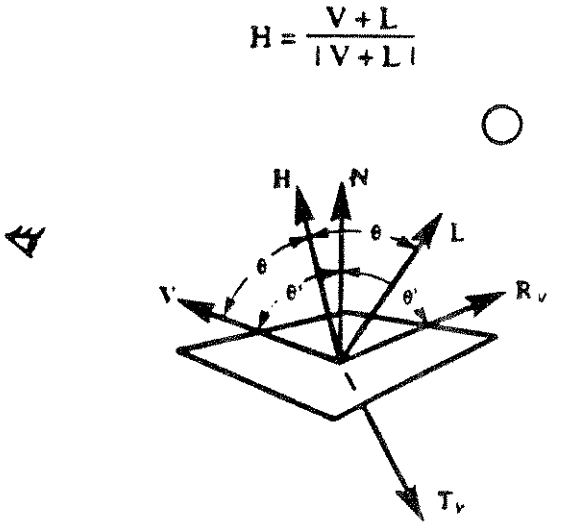


Figura 4.8 : Geometria da Iluminação (HALL/86).

Assim, a cor de uma superfície é dada pela composição de uma matriz de parâmetros com o modelo de iluminação nos diversos pontos visíveis desta, onde a matriz é da forma :

$$\begin{vmatrix} kar & kag & kab \\ kdr & kdg & kdb \\ ksr & ksg & ksb \\ n & & \end{vmatrix}$$

Além destes, aparecerão na matriz acima, os parâmetros ktr, ktg e ktb, estes parâmetros identificarão a transparência de um corpo, como será discutido no item 6.3.2.

Note que, quanto mais difusa for uma superfície, menos especular ela será, o que nos leva a estabelecer que ka + kd = 1 seja uma boa relação entre estas influências em cada uma das componentes de cor (ROGE/85, pg. 333) mas, esta relação não é uma restrição forte pois, o ambiente não a testa e pode-se usar valores diferentes para a produção de certos efeitos visuais.

O modelo de iluminação de Phong, foi criado como uma tentativa empírica de capturar o comportamento da luz sobre o observador e tem pouca base teórica (HALL/86, pg. 272).

4.4.2 TIPOS DE FONTES DE LUZ

As fontes de luz, quanto a sua área, podem ser pontuais ou extensas.

As fontes de luz PONTUAIS, no ambiente virtual da computação de imagens, são aquelas fontes representadas por um ponto no espaço emitindo energia radiante ou, aquelas fontes cujas dimensões são desprezíveis comparadas às dimensões dos objetos iluminados.

Para fontes pontuais, a aplicação do modelo de iluminação que detalhamos acima, é direta visto que, os ângulos e distâncias usadas no modelo levam em consideração somente a posição da fonte de luz e não o seu tamanho ou extensão.

Fontes de luz EXTENSAS são, na realidade, todas as fontes reais mas, considera-se apenas as fontes cuja sua área ou extensão não seja desprezível frente às dimensões dos objetos iluminados.

Para as fontes extensas, a aplicação do modelo de iluminação de Phong teria que ser acompanhada de algum método de integração sobre a área da fonte, o que elevaria sobremaneira o custo computacional desta avaliação.

Desta feita, algoritmos de visualização que trabalham com a simulação de fontes pontuais são notadamente mais rápidos que os algorítmos que trabalham com fontes extensas (como o método de *radiosidade*).

Dentre as fontes pontuais que o sistema de visualização *Scan-Line* (e, consequentemente o TOOKIMA também), permite definir estão *lâmpada, spot* e *sol.*

As fontes do tipo *lâmpada* são fontes onidirecionais, ou seja, irradiam energia luminosa em todas as direções, sendo modeladas pela sua cor (a característica de sua energia radiante) e sua posição no espaço.

As fontes do tipo *sol* são fontes unidirecionais de energia, ou seja, produzem raios de energia luminosa paralelos entre si, tal qual o sol o faz. São um caso particular da fonte do tipo *lâmpada*, considerando a sua posição muito distante da cena, provocando os raios paralelos. Esta fonte é representada pela sua cor e pela direção dos raios luminosos paralelos.

As fontes do tipo *spot* são fontes pontuais direcionais, ou seja, são fontes que tem uma direção principal na qual ocorre a máxima concentração de energia luminosa fornecida por ela, fora desta direção, ocorre uma atenuação desta energia.

Estas últimas são melhor representadas pelo "diagrama goniométrico de irradiação" que caracteriza a intensidade de energia fornecida pela fonte, em função do ângulo entre a direção de máxima irradiação e a direção que se deseja aferir (ver fig. 4.9).

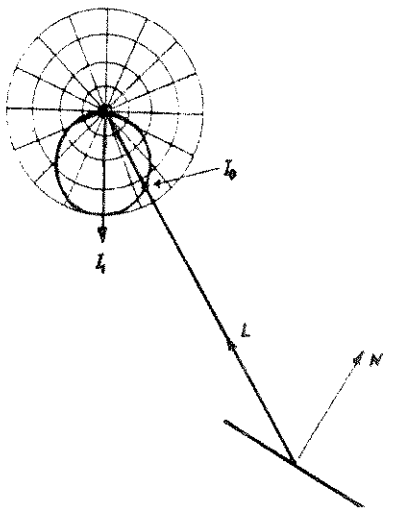


Figura 4.9 : Diagrama Goniométrico (GOME/90).

Onde Il é a energia na direção de máxima irradiação;

Iθ é a energia irradiada, a menos de um desvio de θ graus da direção máxima;

L é a direção de iluminação da superfície;

N é a normal a superfície, no ponto de iluminação.

Note que o diagrama goniométrico enfatiza um lóbulo que representa a direcionalidade da fonte do tipo *spot*.

Este lóbulo é matematicamente descrito por um cosseno do ângulo entre a direção de iluminação e a direção de irradiação máxima (θ da figura 4.9).

Para estabelecermos um controle sobre a concentração espacial da fonte de luz *spot*, é definido um parâmetro chamado "ângulo sólido" que estabelece, a partir da direção máxima, uma relação com um ângulo tri-dimensional onde existe a influência da fonte de luz deste tipo.

Este parâmetro, que na figura 4.9 é 1 identificando um ângulo sólido de 180 graus, influencia o lóbulo do diagrama goniométrico, estreitando-o para valores maiores que 1, aumentando a concentração da intensidade luminosa na direção principal, comparativamente às outras direções.

Este ângulo sólido nada mais é que um expoente aplicado sobre o cosseno que dá a direcionalidade da fonte *spot*, levando-nos a conceber a equação 4.2 para I(j) da equação 4.1, no caso de uma fonte *spot*.

$$I(j) = Il \cdot \cos^{sa} \beta \qquad (4.2)$$

Onde,

I(j), é a intensidade de luz da fonte j do tipo spot (ver equação 4.1);
Il, é a intensidade máxima da fonte;
sa, é o angulo sólido ($c \geq 1$);
β, é o ângulo entre a direção máxima e a direção de iluminação.

Portanto, modelamos uma fonte do tipo *spot* pela sua cor, a sua posição no espaço, a direção de máxima irradiação e o seu ângulo sólido de irradiação.

4.4.3 ELEMENTOS DA ILUMINAÇÃO COMO ATORES

Assim como os campos da estrutura câmera, os campos das estruturas que definem as fontes de luz podem ser tratados como atores.

Exemplificando, a posição de uma fonte pode ser considerada como um ator do tipo POINT; a direção de iluminação de um spot como um ator do tipo VECTOR; a cor de iluminação como um ator do tipo COLOR; o ângulo sólido de um spot como um ator DOUBLE, etc.

E, uma vez estabelecido o ator, este também pode ser referenciado e tratado com todos os recursos definidos para os atores, permitindo que complexas manipulações sobre a iluminação da cena possam até, por si só, compor um efeito visual agradável e uma animação completa.

4.4.4 ESTRUTURA DO MÓDULO *LIGHT*

A estrutura de dados que modela a iluminação da cena está descrita no arquivo cabeçalho *light.h* que, juntamente com os arquivos iniciados com a letra l (*l_define.c* e *l_transf.c*) caracterizam o módulo *light* desta implementação do TOOKIMA.

A estrutura abaixo engloba as características da iluminação do ambiente, do fundo e das diversas fontes de luz, identificando a cor atribuida para o fundo, para a luz ambiente, além de estabelecer a constante de atenuação da intensidade luminosa pelo meio, o ar.

```
typedef struct {Color      background,
                           ambient;
                double     air_index;
                int        n_lights;
                Light      *lights ( N_MAX_LIGHTS );
        } Ilumination;
```

O parâmetro *n_lights* identifica quantas fontes de luz estão definidas para a cena e *lights* é um vetor de estruturas que descrevem as fontes de luz:

```
typedef enum { LAMP, SPOT, SUN } LightType;

typedef union  {PointLight     lamp;
                SunLight       dist;
                SpotLight      spot;
        } LightValue;

typedef struct {Boolean        activity;
                LightType      t;
                LightValue     *v;
                Color          c;
        } Light;
```

Nesta estrutura,

activity identifica se estamos ou não considerando a fonte de luz já definida,

t identifica um dos tipos definidos em *LightType*.

v discrimina os parâmetros dos tipos de fontes.

c é a cor irradiada por um tipo de fonte qualquer.

As estruturas *sunlight*, *spotlight* e *pointlight* contém a descrição dos parâmetros intrínsecos do seu tipo de fonte de luz.

```
typedef struct {Point p;
                double sa;
                Point d;
        } SpotLight;

typedef struct {Point p;
        } Pointlight;

typedef struct {Vector d;
        } Sunlight;
```

Além da caracterização da iluminação, o módulo *light* também define a estrutura *Surface*, responsável pela caracterização das características de cor de um ator ou decoração:

```
typedef struct {Color ka, kd, ks, kt, ps;
                double n, i_refr;
                } Surface;
```

No arquivo l_define.c, estão as definições ou inicializações das estruturas de dados que modelam a iluminação da cena.

- *set_back*, inicializa a cor do plano de fundo;
- *set_env_light*, inicializa a cor da luz do ambiente;
- *set_lamp* , define uma fonte de luz do tipo lâmpada, definindo sua potência;
- *set_point_light*, define uma fonte do tipo lâmpada através de todos os

parâmetros da estrutura *PointlIght*;

- *set_sum*, define uma fonte de luz do tipo *sol*;
- *set_spot*, define uma fonte de luz do tipo *spot*;
- *reset_light_p*, altera a posição de uma fonte de luz;
- *reset_light_d*, altera a direção de iluminação de uma fonte;
- *reset_light_c*, altera a cor de uma fonte de luz;
- *read_color*, lê, de um arquivo com a extensão .cor, a estrutura que identifica a cor de uma superfície.

Já o outro arquivo deste módulo, o l_transf.c, contém funções específicas para transformações diretas sobre a iluminação, assim como a câmera sintética tinha as suas funções específicas para manipulação.

- *turn_on* (*turn_off*), ativa/acende (ou apaga) uma fonte de luz definida;
- *fade_in* (*fade_in_all*), gera o efeito de *fade_in*, variando a cor de uma (todas) fonte de luz do preto até uma cor determinada;
- *fade_out* (*fade_out_all*), gera o efeito de *fade_out*, variando a cor de uma (todas) fonte de luz de um valor até o preto.

Portanto, sobre o módulo *light* como um todo, pode-se dizer que este é o responsável por modelar um grau de realismo razoável à imagem, tornando-a mais agradável de ser vista, sem que o animador tenha que se preocupar em colorir o objeto, parte por parte, ele apenas escolhe a cor do mesmo, através da coerente escolha dos parâmetros para o modelo de iluminação que, inclusive, podem estar catalogados em uma biblioteca de cores.

Além disso, os diversos tipos de fontes luminosas e, mais ainda, a manipulação destes módulos através da entidade ator, concorrem para criar um novo grau de liberdade e flexibilidade para a animação, induzindo o animador a usar e desfrutar destes recursos.

4.5 CONCLUSÃO

Vimos neste item os diversos elementos que compõem uma cena. Como nas outras mídias, vários destes estão obscurecidos pela óbvia presença do ator. Assim, a iluminação e a câmera são muitas vezes menosprezadas pelo espectador mas, o animador ou diretor tem a obrigação de manipular, e bem, estes elementos da cena.

Com respeito as funções de transformações exclusivas de certos elementos, estas são muito úteis e conhecidas, por isso estão disponíveis.

Quanto a possibilidade de tratar todos os elementos como atores, esta facilidade concorre para uniformizar a interface textual com o usuário, tornando mais fácil a descrição dos movimentos e, em última análise, incentivando o uso de todos estes recursos em uma animação.

No próximo capítulo abordaremos as facilidades disponíveis para "animar" (atribuir movimento) os elementos da cena vistos neste capítulo.

CAPÍTULO 5

TOOKIMA: MOVIMENTO

5.1 INTRODUÇÃO

Neste capítulo nos deteremos a expor o mecanismo da geração da dinâmica gráfica, da geração da variação de parâmetros através do tempo, enfim, do movimento.

Veremos quais ferramentas temos disponíveis para animar os atores e como estabelecer referências para estas variações, mecanismo que nos permitirá criar animações algoritmicas.

Em seguida analizaremos um tratamento para interpolação que nos permitirá definir trajetórias que serão seguidas por um determinado ator com uma aceleração específica, discutindo antes os conceitos de trajetória e aceleração no âmbito cinemático do nosso sistema.

Por fim detalharemos as estruturas de dados e as funções disponíveis para estas tarefas, inclusas no módulo *motion*.

5.2 TRANSFORMAÇÕES GEOMÉTRICAS

"Transformações geométricas são essenciais para muitas aplicações gráficas" (FOLE/90, pg. 201) e, animações inteiras podem ser feitas utilizando apenas este

recurso.

As transformações geométricas podem ser usadas tanto internamente pelo sistema e fora do controle do usuário (no *pipeline* das transformações de vizualização que implementam a câmera sintética, p. ex.) quanto, externamente sob escolha e quantificação do usuário.

"As transformações geométricas são o meio pelo qual se obtém sobre um ator uma unidade de movimento" (BADL/87, pg. 123), esses movimentos tanto mudam sua posição quanto, sem mudar sua posição mudam seu tamanho ou orientação, no espaço. Estas transformações geométricas serão usadas para implementar o movimento sobre caminhos arbitrários (alvo de discussão no item 5.3).

As transformações geométricas podem ser (BADL/87, pg. 123):

- LINEARES RÍGIDAS, implementadas por matrizes cujos elementos são constantes (ex: escala, translação, rotação, espelhamento, *skew*, etc.);

- LINEARES NÃO RÍGIDAS, implementadas por manipulação nos objetos usando funções lineares (ex: distensão -*stretch*-, esmagamento -*squash*-, etc.);

- NÃO LINEARES, onde a manipulação sobre o objeto é obtida via funções não lineares (ex: curvar -*bend*-, torcer -*twist*-, ondular -*curl*-, diminuição gradual da expessura -*taper*-, etc.);

- SOBRE ELEMENTO LOCAL, que são operações muitas vezes construtivas e usadas em modeladores de objetos (ex: extrusão -*extrude*-, inflar -*inflate*-, murchar -*deflate*-, etc.).

As mais comuns e largamente disponíveis nos sistemas são as Transformações Geométricas Lineares Rígidas (**TGLR**) pois, são facilmente implementadas por matrizes homogêneas simples e permitem agrupar transformações de maneira eficiente (multiplicação de matrizes).

As TGLRs simples, que estão disponíveis no TOOKIMA são: Translação, Rotação e Escalamento.

A translação geométrica altera os valores do objeto produzindo o efeito de

deslocamento, normalmente sob um determinado eixo.

A rotação, é responsável por modificar a orientação do objeto no espaço euclidiano, especificada pelo ângulo de rotação sobre um dos eixos.

O escalamento por sua vez, altera as dimensões do objeto, a partir de uma percentagem e do eixo de variação.

Por definição, um ator VECTOR, no TOOKIMA, está sempre na origem por isso, a translação aplicada neste ator não produz efeito sobre ele nem sobre seus *slaves*.

Os atores DOUBLE, POINT e COLOR (estes últimos, funcionalmente iguais) também não sofrem influência da transformação de escalamento (nem seus *slaves*).

Conforme seja implementada e conforme seja a referência adotada para a transformação pode-se obter, indesejavelmente, uma outra transformação.

Para estabelecer com clareza o referencial da transformação, convencionou-se nomear ABSOLUTO ou RELATIVO (doravante usaremos resumidamente o termo A_R para esta opção que representa o tipo de dado definido para esta finalidade, ver 4.2.5), os referênciais na origem ou no centro de massa do ator ao qual estiver sendo aplicada a transformação, respectivamente.

Assim, as TGLR mencionadas acima estão disponíveis com o parâmetro ABSOLUTO ou RELATIVO, para serem aplicadas a um ator, sobre um eixo separadamente ou sobre os três eixos concomitantemente (a exceção da rotação).

A rotação não dispõe de uma função única em que se possa especificar concomitantemente ângulos de rotação para os três eixos porque a ordem em que são implementadas as rotações, quando se usa matrizes para esta tarefa, influencia o resultado final e, portanto, o usuário teria que ficar atento para a sequência de transformações que foi escolhida para esta implementação em particular.

Uma alternativa a esta limitação é o uso de **quaternium**'s (SHOE/85), uma velha maneira de modelar rotações (que produz o mesmo resultado final independente da ordem em que as rotações sobre os eixos são executadas) mas que, entretanto, é pouco popular.

Ao invés desta rotação sobre os três eixos, está definida outra, de alguns graus, sobre um eixo arbitrário, com referencial na origem, especificado na chamada do procedimento e que pode ser ABSOLUTO ou RELATIVO.

Assim, 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.

Rotação absoluta rotaciona o ator em relação ao eixo especificado em torno da origem modificando também o posicionamento do ator no espaço.

Escalamento relativo 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.

Escalamento absoluto aumenta ou diminui o tamanho do objeto em uma ou três direções em relação à origem, causando a movimentação do ator, além do escalamento.

Translação relativa desloca o ator de um valor na(s) direção(ões) especificada(s) em relação à sua posição anterior (seu centro de gravidade).

A translação absoluta teria o mesmo efeito visual da translação relativa visto que é uma transformação acumulativa, então optou-se por dar o sentido de "translação para" um determinado ponto especificado, ou seja, forçar um determinado posicionamento.

Além destas transformações básicas dispomos no TOOKIMA de algumas variantes (já discutidas no item 4.2): *growth*, *shrink*, *free_scale_actor* e, *free_rotate_actor*, sendo que as duas primeiras tem a opção A_R.

A convenção sintática assumida para as funções de TGLR são as seguintes:

1) O condicional, A_R, se existir, será sempre o primeiro argumento;

2) O nome do ator, alvo da transformação, será sempre o último argumento;

3) Os parâmetros dependentes da transformação virão entre os dois acima.

5.3 ESPECIFICAÇÃO CINEMÁTICA

Como já foi estabelecido no capítulo 3, o TOOKIMA é um conjunto de ferramentas para AMC (Animação Modelada por Computador) visando o modelo cinemático de animação.

Dentro dos sistemas cinemáticos existem os termos trajetória e aceleração específicos para este contexto, os quais passamos a elucidar.

Trajetória é o caminho no espaço bidimendional ou tridimensional a ser percorrido por um determinado elemento da cena.

Aceleração refere-se ao modo como o elemento seguirá uma trajetória, basicamente a cinemática a ele empregada; este modo pode ser acelerado, desacelerado, linear, dentre outros.

O que se pretende no contexto da animação cinemática é a possibilidade de definição arbitrária e flexível de trajetórias suaves e acelerações e, a livre e independente combinação entre elas (de tal forma que um caminho possa ser percorrido de diversas maneiras (com diferentes acelerações) pelo(s) ator(es) da cena).

5.3.1 TRAJETÓRIA

Nosso trabalho visa pois, fornecer subsídios para uma animação cinemática sem a necessidade do animador estabelecer todos os valores que o elemento deverá assumir. Assim, através da especificação de alguns pontos, o sistema se responsibilizará por gerar os valores intermediários necessários para que o guiamento seja efetuado (não precisando ser completo nem exaustivo).

E, para que o movimento pareça suave, não robótico (*not jerkyness*), é desejavel que exista alguma continuidade matemática com relação ao movimento. Esta continuidade deve ser sentida na posição, na velocidade e na aceleração do ator se deslocando sobre a trajetória. Isto confere à curva a necessidade de uma continuidade matemática de segunda ordem obtida p. ex., por curvas interpoladoras ou aproximadoras cúbicas como *splines, Bézier*, etc.

Algumas curvas possuem parâmetros que controlam inclusive o modo de interpolação gerando uma relação com a aceleração que o ator deverá seguir naquele caminho de maneira integrada mas, isto pode se tornar complexo para o animador, pela difícil associação dos parâmetros com a aceleração desejada.

Lasseter (LASS/87) comenta que diversos problemas neste tratamento podem ser resolvidos se admitirmos que a curva que modela a trajetória (*path*) seja diferente da curva que modela a aceleração (*Timing*), tornando a especificação mais fácil e enfatizando o estreito relacionamento entre a curva da aceleração e o seu comportamento.

No TOOKIMA, ambas as curvas *Spline* e *B_Spline* Cúbicas estão disponíveis a critério do usuário.

A curva resultante (CR) obtida por um dos métodos acima é uma sequência de *grain* pontos ordenados em um total de (npc-3) segmentos consecutivos, onde **npc** é o número de pontos de controle definidos pelo animador e *grain* é o número de pontos desejados para serem calculados. Este cálculo, para esta implementação, é feito uma única vez e em um instante determinado quando então a formulação *Spline* e *B-spline* são "esquecidas" e passa-se a trabalhar apenas com CR.

Um segmento é uma certa quantidade de pontos calculados, definidos e influenciados pela combinação de quatro pontos de controle consecutivos. Cada um dos pontos de controle, por sua vez, influencia quatro segmentos (daí a característica de controle local: a alteração de um ponto de controle influencia o formato da CR em uma região limitada, um número limitado de segmentos) (MORT/85, pg. 132).

A *Spline* Cúbica segue a formulação matricial de CATM/78 para c = 0.5, reproduzido na equação 5.1.

$$P_i[u] = [u^3 u^2 u^1 1] * \begin{vmatrix} -c & 2-c & c-2 & c \\ 2c & c-3 & 3-2c & -c \\ -c & 0 & c & 0 \\ 0 & 1 & 0 & 0 \end{vmatrix} * \begin{vmatrix} Pc[i] \\ Pc[i+1] \\ Pc[i+2] \\ Pc[i+3] \end{vmatrix} \qquad (5.1)$$

Para $0 \leq u \leq 1$ e, $0 \leq i \leq npc-3$.

Onde,

i é o número do segmento de curva que é influenciado por 4 pontos de controle.

Pi [u] é o ponto do segmento **i** calculado na posição paramétrica **u**.

u é a variável paramétrica.

Pc [i] é o ponto de controle de índice **i** (o mesmo do segmento).

Ressalve-se nesta formulação que se deve duplicar o primeiro ponto na lista de pontos de controle a fim de que a interpolação resultante comece graficamente neste ponto (de outra forma esta começaria a partir do segundo ponto de controle) e, para garantir os *grain* pontos deve-se, para cada segmento, executar um laço cujo incremento é, du = (npc -3) / *grain*.

A *B_Spline* Cúbica, também, segue uma formulação matricial semelhante, detalhada em MORT/85, pg. 136, reproduzida na equação 5.2:

$$Pi[u] = [u^3 u^2 u^1 1] * \begin{vmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{vmatrix} * \begin{vmatrix} Pc[\ i-1\ ] \\ Pc[\ i\ ] \\ Pc[\ i+1\ ] \\ Pc[\ i+2\ ] \end{vmatrix} \quad (5.2)$$

Para $0 \leq u \leq 1$ e, $0 \leq i \leq npc-2$.

Ressalte-se nesta formulação a característica de que a CR não passa por nenhum ponto de controle. Achou-se interessante que, pelo menos o primeiro e último pontos deveriam ser interpolados e daí usou-se o artifício de recalcular estes pontos extremos de forma a garantir que o primeiro e último pontos calculados (P1[0] e Pn-2[1]) resultassem no primeiro e último ponto de controle especificado pelo usuário, através das equações 5.3 e 5.4.

$$P1[0]_{novo} = 6\ Pc[0] - 4\ Pc[1] - Pc[2]. \quad (5.3)$$

$$Pnpc-1[1]_{novo} = 6\ Pc[npc-1] - 4\ Pc[npc-2] - Pc[npc-3]. \quad (5.4)$$

E, de maneira semelhante, calcula-se du = (npc-3) / *grain*.

O valor *grain* é super dimensionado para que a CR calculada seja bem definida e possamos obter qualquer localização sobre ela. Esta localização sobre a curva será definida pelo mecanismo Trajetória versus Aceleração que será explicado no item 5.3.3 e que, garantirá a independência na definição destas duas entidades.

Mas, não só as curvas livres serão tratadas por este mecanismo, as curvas bem

conhecidas, como quadráticas, logarítmicas, etc., também são convertidas para esta abordagem.

Esta conversão nos parece indicada para uniformizar o tratamento das curvas de trajetória e da geração das trilhas, que veremos logo a seguir.

Assim, qualquer curva conhecida ou combinação destas será amostrada para a obtenção de alguns pontos de controle e então recalculada via *B-Spline* ou *Spline* de maneira totalmente transparente ao usuário.

As curvas usadas para especificar a trajetória foram denominadas de trilhas ou *track*'s, inspirado na nomenclatura e idéias expostas em GOME/85 e VELH/89.

Uma trilha é o encapsulamento do resultado do cálculo das curvas aproximadoras e interpoladoras usadas para interpolar os pontos de controle (ou pontos chaves, para relacionar com quadros chaves - *Key-frame* -) responsável por especificar o caminho a ser percorrido por um ator.

Uma trilha pode ser seguida por vários atores numa mesma cena e, um ator pode seguir várias trilhas distintas.

Duas são as funções de inicialização/criação de trilhas: trilha_conhecida (*known_track*) e trilha_livre (*free_track*). O uso de quaisquer das funções para trilha retornará ao programa/cena que o chamou um valor, de um dos tipos possíveis (DOUBLE, VECTOR, POINT, COLOR), para ser usado em uma TGLR qualquer.

known_track (como o nome sugere), especifica as curvas conhecidas, tendo como parâmetros um tipo de trajetória (LINEAR, QUADRÁTICA, CÚBICA, EXPONENCIAL, LOGARITMICA) que pode ser facilmente estendido para qualquer formulação matemática de funções ou combinações destas, o tipo de valor que está sendo tratado.

Para *free_track*, precisamos ter gerado um arquivo com os pontos de controle da *B_spline* pois este será um dos argumentos desta função. Este arquivo faz parte de uma biblioteca de curvas (sugestão de MACL/85, pg. 37) ou pode ser criado especificamente para aquela aplicação. O relógio local também é parâmetro de entrada como no caso anterior, pois ambos precisam ter, uma indicação da quantidade mínima de pontos que devem ser calculados para satisfazer qualquer tipo de aceleração.

Depois de gerada a trilha os procedimentos de caminhamento e de acoplamento com a

aceleração são independentes do modo de geração. Veja, a seguir, o esquema destes dois procedimentos.

Para "caminhar-se" por uma trilha tem-se um procedimento denominado seguindo_trilha (*following_track*) que, assim como as transformações geométricas, o primeiro parâmetro é o índice de referêncial A_R e o último parâmetro é o relógio local.

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 ultima posição) significa perguntar ao procedimento quanto o ator andou neste último *TICK* do relógio, ou seja, quanto variou o parâmetro do ator entre a última "posição" e a atual.

Os efeitos produzidos pelas opções de referencial são distintos: Caso escolha-se a ABSOLUTE, para a trilha e a transformação, o ator segue rigorosamente a curva definida exatamente na mesma posição em que foi definida a curva. Caso escolha-se a referência RELATIVA para a trilha com uma transformação geométrica RELATIVA o ator segue a curva, mas a partir da posição em que ele está (como se a curva tivesse sido definida a partir da posição do ator).

A mesclagem de referencial no caminhamento e na transformação geométrica, assim como o uso de acelerações de referêncial relativo, podem levar a efeitos interessantes mas raramente previsíveis.

As funções acima existem no nível da cena (não no *script*). Para que o ambiente saiba que na primeira ativação da cena deve calcular todos os *grain* pontos da trajetória (CR), nas funções de inicialização/criação da trilha, *track*, a estrutura de dados possui uma variável booleana, denominada done, que identifica se a trilha já foi calculada ou não. Se a trilha e o relógio local forem modificados por algum motivo (ex: Cue de duração variável), então esta variável deve ser reinicializada para que a trajetória seja recalculada.

Este mecanismo funciona pois a trilha é uma estrutura estática ao módulo que só pode ser vista localmente à cena e subcenas, sendo inicializada pelo ambiente da linguagem C com done = 0 = FALSE (conforme definição da linguagem C em KERN/86, pg. 81).

Um detalhe interessante neste mecanismo é que a trilha só será calculada se a cena for ativada pelo menos uma vez e, na sua primeira ativação. Isto pode parecer natural na tentativa de evitar o cálculo de trajetórias de cenas condicionais que não foram ativadas, entretanto causa uma certa descontinuidade no fluxo dos cálculos da animação pois o sistema demandará um certo tempo assim que uma cena com trilha seja ativada causando um atraso na pré-visualização em arame (*wire-frame*, facilidade prevista para o sistema) naquele instante.

Este detalhe não é preocupante visto que nosso ambiente está voltado a geração das imagens não na sua visualização em tempo real, deixando esta tarefa para um subsistema subsequente denominado *playback* que dentre outras coisas, deve ser responsável por compactações, edições e compilação diferencial de imagens (DENB/86).

5.3.2 ACELERAÇÃO

Discutindo sobre a aceleração, optou-se por desenvolver funções que retornassem valores normalizados (entre 0.0 e 1.0) para a curva da aceleração obedecendo a seguinte sintaxe:

kinematic_function (A_R, [parâmetros,] *clock*);

Onde,

- se A_R for ABSOLUTO, a avaliação do valor do tempo na curva da aceleração é absoluta, indicando o valor exato sobre esta, ou seja, calcula "onde deveria estar o ator" sobre a curva da aceleração naquele instante;
- se A_R for RELATIVE, deseja-se avaliar o quanto variou a aceleração do ator neste último segundo transcorrido. É usada em transformações acumulativas, que dependem da última posição do objeto;
- parâmetros, é um conjunto de dados opcionais, dependendo da função;
- *clock*, é a variável que faz o controle do relógio local.

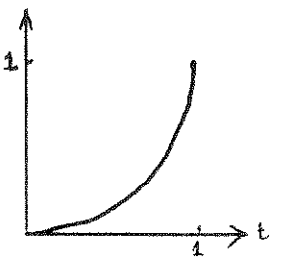
Como convenção deste módulo, temos que o condicional A_R será o primeiro argumento e o *clock* será o último argumento.

As funções de aceleração são curvas que expressam um determinado tipo de

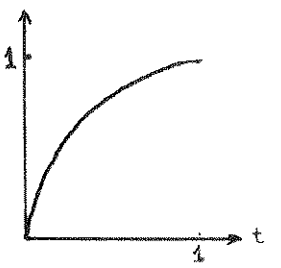
movimento. Exemplo: *slow_in* (movimento acelerado), *slow_out* (movimento retardado), *slow_io* (movimento acelerado depois desacelerado). Estas curvas são também conhecidas por *eases* (*ease_in*, *ease_out* e *ease_io*, respectivamente, ver GOME/85, p. ex.).

As curvas desses movimentos podem ser obtidas através de diversas funções matemáticas. Assim como em THAL/85b, pg. 49, escolhemos as funções trigonométricas para esta finalidade:

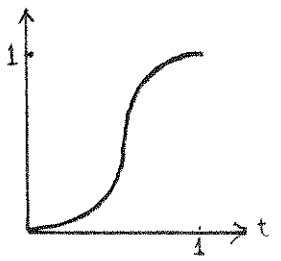
slow_in acelerado $1 - \cos(90\ t)$



slow_out desacelerado $\sin(90\ t)$

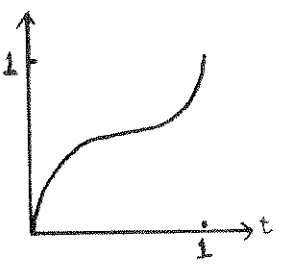


slow_io acelerado_desacelerado $\frac{1 - \cos(180\ t)}{2}$

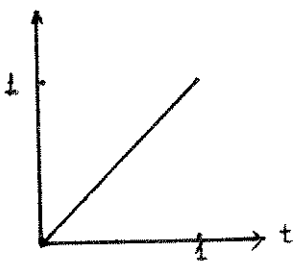


As funções acima, tem extensões imediatas como abaixo:

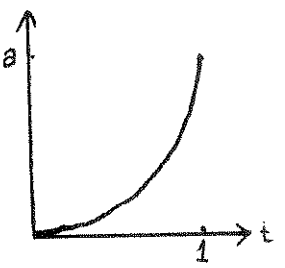
dec_ac desacelerado_acelerado



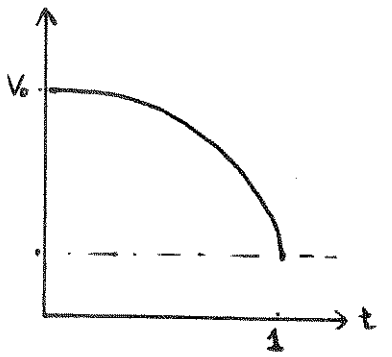
linear



acc aceleração **a** $a\ t^2$



decc desaceleração **a**, Vo t - 0.5 **a** t^2
a partir de Vo



Note que nas funções acc e decc apareceram os parâmetros **a** e Vo, que são particulares destas funções.

A avaliação da função relativa a A_R é realizada da seguinte forma:

```
Se ABSOLUTO
  | aceleração(t) = f(t).
Se RELATIVO
  | aceleração(t) = f(t - delta_time)
  |
  | t=duração
  | ∑ aceleração(t) = 1, se a curva da aceleração é slow_in, slow_out e slow_io
  | t=0
```

Onde,

- **t** é o tempo local à cena variando entre 0 e 1, ou seja, o tempo local normalizado (tn = *Tlocal*/duração);

- **delta_time** é o equivalente a 1/**Rate** em função do relógio local da cena;

- **aceleração** (t) é o valor a ser retornado pela função e;

- f(...), é a função de uma aceleração em particular.

Caso o animador queira criar a sua própria curva de aceleração, está disponível uma função (*free_dynamic*) cujos parâmetros incluem uma série de pontos de controle que serão usados internamente para gerar uma curva interpolada para daí então dispormos desta função de aceleração.

Com o valor obtido pela função da aceleração, através da avaliação de uma curva (f(...)), e juntamente com a curva calculada da trajetória usa-se o mecanismo de mesclagem destas duas curvas para alcançar a independência entre elas, mecanismo este que passaremos a detalhar.

5.3.3 MESCLANDO TRAJETÓRIA E ACELERAÇÃO (TxA)

Partindo do pré-suposto de que todas as curvas de trajetória são trilhas (*track*) compostas de *grain* pontos calculados, precisamos detalhar o mecanismo que garante a independência entre a descrição da trajetória e uma aceleração qualquer.

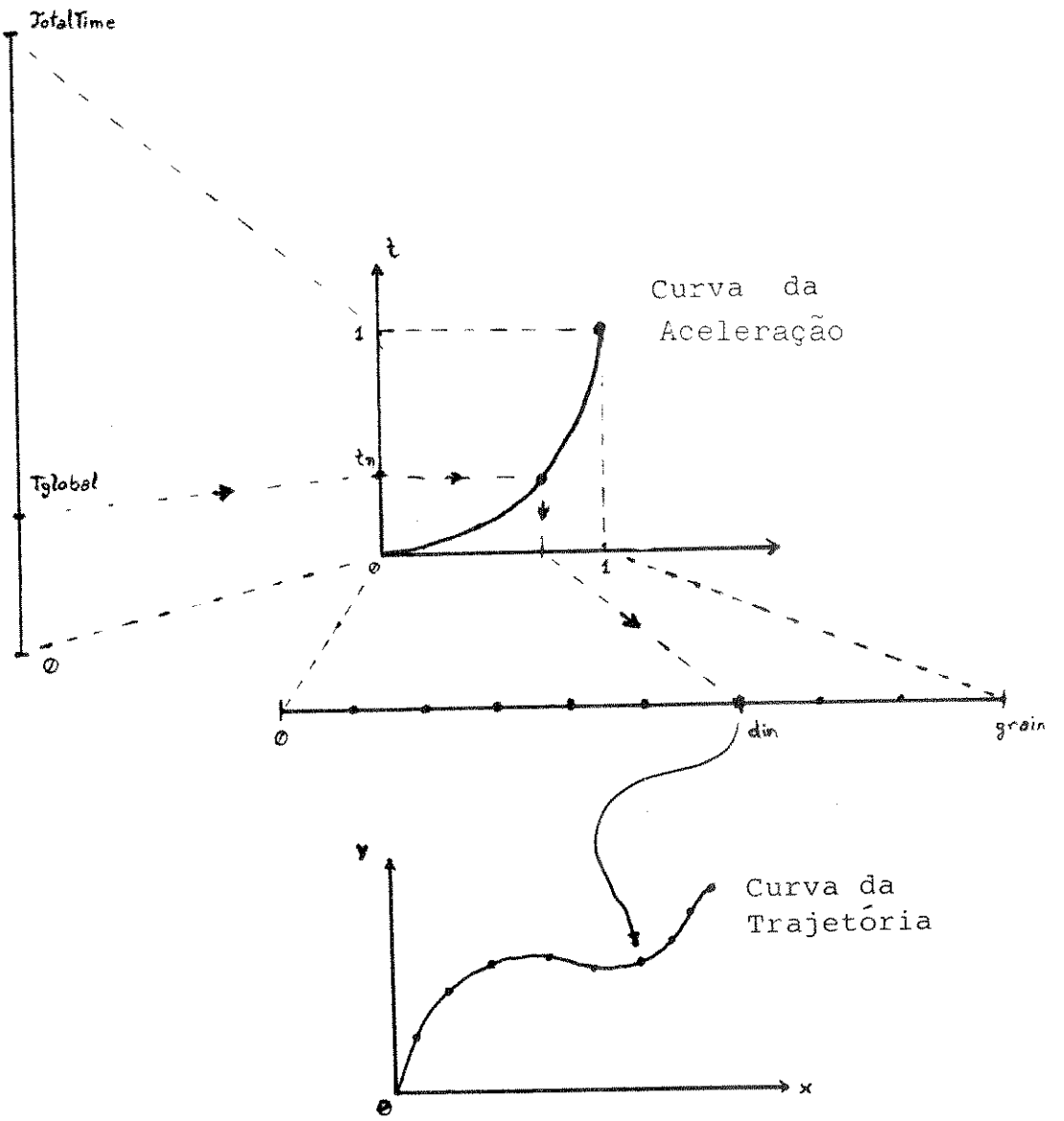


Figura 5.1 : Mecanismo TxA.

A figura 5.1, mostra o esquema de funcionamento do mecanismo Trajetória versus Aceleração (TxA) que, sucintamente, é a avaliação de um valor na curva da aceleração para definir a escolha de um ponto na curva da trajetória.

Um quadro vai ser gerado para um determinado Tglobal (começo do processo). Este valor de Tglobal é convertido para o relógio local da cena, no começo desta. Esta hora local é normalizada pelo valor da duração da cena obtendo-se então o valor tn (ver 5.3.2).

Quando se invoca qualquer curva da aceleração, este valor de tempo normalizado é passado para a função através da variável *clock*, a partir daí, avaliando o valor de tn através da curva especificada para a aceleração, encontra-se o valor dim que é o valor de retorno das funções de aceleração (no caso da figura 5.1 é uma aceleração ABSOLUTA). Se estivéssemos trabalhando com aceleração RELATIVA este valor seria subtraído do valor dim no último instante (chamado de *last_d_value*) e então retornar-se-ía esta diferença.

Este valor da função da aceleração, quando usado de maneira relativa e multiplicado por um valor *amount*, é suficiente para garantir que uma transformação produza, para aquela cena, uma tranformação de *amount* unidades, independente do número de quadros gerados pela cena.

Exemplo:

```
dim = linear (REL, clock);
Rotate_actor_x (REL, 360 * dim , "coruja");
```

As sentenças acima produzirão uma rotação de 360 graus sobre o eixo x no ator coruja, de forma linear e uniforme. Independente do número de quadros em que a cena esteja ativa garante-se que o total de graus da rotação é 360.

Para mesclar a curva da aceleração com a trilha, o processo não para por aqui.

De posse do valor dim da aceleração, por uma regra de três simples, determina-se quanto valeria este valor numa escala de 0 a *grain* unidades discretas que, corresponde exatamente ao ponto que deverá ser usado para determinar a posição absoluta sobre a trilha.

Se a trilha está sendo seguida de maneira relativa então o ambiente precisaria do valor da aceleração no instante imediatamente anterior.

Isto é feito armazenando o valor do último instante na própria estrutura *Track* que representa a trilha, para uso posterior, se necessário. Daí, de posse do último valor da aceleração, a condição relativa é alcançada pela diferença da posição obtida sobre o *track* na posição atual e a posição do *track* no último instante.

Com este processo, podemos associar qualquer aceleração ou até várias à mesma trajetória. Também pode acontecer de uma mesma trilha ser caminho de vários atores.

Exemplificando, pode-se chegar a casos de termos uma aceleração cúbica sobre uma trajetória cúbica, uma spline sobre outra spline, etc.

Averiguando algum *script* que use transformações nos atores (*translate, scale, rotate*) com valores obtidos por uma determinada aceleração observa-se que se usam duas escolhas de referêncial. Um A_R para a transformação e outro para a aceleração que tem significados diferentes devido as suas essências.

A condição A_R para a aceleração identifica 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.

A condição A_R para as TGLR identifica se o referêncial da transformação é a origem ou o centro de gravidade do ator, respectivamente. Exceto para a translação, cuja aplicação das condições acima incorreriam na mesma operação, optou-se por considerar a translação absoluta uma transformação de posicionamento forçado (*translate_to*) enquanto que a relativa tem significado de deslocamento acumulativo, como já vimos.

A escolha de dois valores A_R (que, *a priori*, não tem nenhum relacionamento) pode levar a confusões ou dúvidas na hora da codificação do *script*, por isso vale ressaltar que as transformações tem, como argumento, valores acumulativos (exceto *translate_actor*(ABSOLUTE, ..)) e é assim que estes são interpretados internamente. Daí ser normal a condição da aceleração ser RELATIVA.

Se, entretanto, a condição A_R da aceleração for definida ABSOLUTA, então o resultado será bastante diferente do que, normalmente, o usuário estaria esperando, a

menos que este último saiba exatamente o que esteja fazendo.

Acompanhe no exemplo:

```
dim = linear ( ABSOLUTE, clock );
Rotate_actor_y ( RELATIVE , 50 *dim , "cubo" );
```

A transformação acima provocará a rotação do "cubo" de uma quantidade cada vez maior começando do zero até chegar a 50 graus portanto, a rotação será "linearmente acelerada".

Para que o "cubo" sofresse uma rotação constante que garantisse que naquela cena ele rotacionaria no máximo 50 graus em relação à posição original a condição da função da aceleração teria que ser RELATIVE.

Mas, porque então as funções de aceleração tem uma condição complicadora ?

A razão está basicamente, em que as transformações são efetuadas em relação ao seu último estado/valor/posição, daí serem chamadas de acumulativas, e não a partir da posição inicial (nem da cena nem do *script*). Um critério de implementação muito comum.

Esta solução não complica as possíveis aplicações em física, matemática, química e/ou dinâmica (pode-se armazenar temporariamente o valor do último estado e subtraí-lo mais tarde a fim de detectar-se a variação).

Note que, apesar de tanto a trilha quanto a aceleração poderem ser usadas separadas para outros propósitos, a integração destes dois recursos, via mecanismo TxA, é que garante a flexibilidade e o uso de todo o potencial destes tratamentos.

5.4 ESTRUTURA DO MÓDULO *MOTION*

A seguir enumeraremos os principais tipos de dados definidos no arquivo *motion.h* para tratar as trilhas e as acelerações. As transformações geométricas aplicadas a um ator (que poderiam ser apresentadas aqui) foram apresentadas junto com o módulo *actor*.

Inicialmente lembramos que as trilhas podem ser oriundas de curvas conhecidas ou geradas por *Spline* ou *B_Spline*. Para o primeiro caso, está definida uma estrutura que

enumera os tipos de curvas conhecidas que podem ser solicitadas (isto é, já estão implementadas), sem menosprezar as possíveis expansões nesta enumeração:

```
typedef enum { LINEAR, QUADR, CUBIC, EXP, N_TYPES } TrajType;
```

Uma variável do tipo acima é usada no algoritmo que avalia a curva analítica e gera a estrutura que descreve a trilha, denominado de *Track* e descrita abaixo:

```
#define TRACK( name )   static  Track *( name )

typedef struct {Boolean     done;
                int         n_ele;
                double      last_d_value;
                ValueType   type;
                Element     *eles;
        } Track ;
```

Cujos campos são auto-explicativos ou já foram mencionados no texto acima.

Além dos tipos de dados definidos acima, o módulo *motion* define funções relacionadas com o movimento sobre as trilhas e para aceleração, já devidamente mencionadas:

No arquivo m_track.c encontram-se as funções para criação e uso de trilhas (*track*):

- *known_track* constrói uma trilha com o formato de uma curva conhecida, especificada por um parâmetro do tipo *TrajType*. Esta trilha contém quaisquer valores especificados em ValueType, também tem como parâmetro o relógio local e retorna uma variável do tipo *Track*;

- *free_track*, constrói uma trilha a partir de um arquivo de pontos de controle para uma *spline* ou *B_spline*, tendo como parâmetro o nome do arquivo, o nome do *Track* e o relógio local;

- *spline* e *bspl* são duas funções responsáveis por gerar as curvas *spline* e *B_spline*, respectivamente, para as duas funções acima;

- *read_pc*, é usada para interpretar o arquivo de pontos de controle especificado

nas duas primeiras funções;

- *following_track*, responsável por retornar um *Element*, que resulta no caminhamento da trilha, tendo A_R como parâmetro além do nome da trilha a ser seguida e do instante de tempo em questão.

Existe neste módulo um arquivo denominado m_math.c, que contém funções matemáticas específicas para as funções ***spline*** e ***bspl*** mostradas acima, funções estas que não reproduziremos aqui (caso o leitor se interesse por estas funções ou detalhes mais aprofundados de toda a implementação, sugerimos se reportar à HOUN/92).

Para a aceleração, o módulo do movimento reserva um arquivo, denominado m_dynamic.c, para implementar estas funções.

- *linear*, *slow_in*, *slow_out*, são as mais conhecidas acelerações (comportamentos) que se dispõem numa animação. Implementam diretamente as funcionalidades expressas em seus nomes e já discutidas neste capítulo;

- *acc_dec*, *dec_acc*, são combinações das anteriores para comporem movimentos mais elaborados porém comuns de aceleração seguida da desaceleração (semelhante à *slow_io*) e vice-versa (semelhante a *slow_oi*), respectivamente;

- *acc_motion*, *dec_motion*, são algumas extensões elementares, usando-se equações cinemáticas para implementar o movimentos linearmente acelerados ou desacelerados tendo-se entre os parâmetros a variável aceleração (a) e a velocidade inicial (Vo);

- *free_dynamic*, implementa uma aceleração especificada por uma curva de pontos de controle.

Mas, o movimento também se faz através de TGLR (como explanado no texto) que não fazem parte deste módulo (pelo menos não estão definidas nos arquivos deste módulo) por estarem intimamente relacionados com o ator (e por isso estão no módulo *actor*, ver 4.2.5).

5.5 CONCLUSÃO

Procuramos apresentar nos itens acima a forma como é produzido o movimento na nossa abordagem cinemática da animação, apresentando as TGLR como base deste processo e uma variante mais amigável (trilhas) mesclada com recursos de aceleração (estas trilhas se utilizam internamente das TGLR para sua implementação) de forma a alcançar princípios como arcos, começos e saídas lentas e temporização (ressaltados em 2.4.2, 2.4.9 e 2.4.10).

Ambas as abordagens permitem a escolha do referencial (ABSOLUTO ou RELATIVO) que deve seguir a transformação, gerando desta forma uma gama muito flexível e útil de alternativas.

Estabelecemos uma forte relação entre as TGLR e ator quando resolvemos excluir deste item, as funções que as aplicam sobre os atores, deixando transparecer uma constatação iminente: animações são basicamente a definição e movimentação de atores.

CAPÍTULO 6

TOOKIMA : VISUALIZAÇÃO

6.1 INTRODUÇÃO

Uma ferramenta que vise produzir animações com algum apelo realista não pode prescindir de um sistema de geração de imagens estáticas. Este sistema, denominado por nós de sistema de visualização, é composto por duas fases apresentadas na figura 4.3., a **visibilidade** e o **rendering** (na realidade integradas em um único algoritmo).

A fase de **visibilidade**, preocupa-se com a determinação das linhas e superfícies visíveis.

A fase de **rendering** preocupa-se com a integração da cena com os modelos de iluminação e o realismo da imagem.

Neste capítulo apresentaremos, por questões didáticas, a evolução do algoritmo de visibilidade adotado no TOOKIMA e sua implementação (bastante detalhado em HOUN/90) bem como sua integração com os modelos de tonalização e iluminação na direção do *rendering* (detalhado em PRET/91).

6.2 VISIBILIDADE

Existem, basicamente, duas grandes classes de algoritmos de visibilidade (também conhecidos por HLHS, *Hidden Line Hidden Surface*): No **Espaço Objeto** e no **Espaço Imagem** (ROGE/85, pg. 190).

A primeira é assim denominada, pois os algoritmos são implementados no sistema de coordenadas denominado físico, onde os objetos são definidos. Dentre estes algoritmos tem-se o *Ray-Tracing* (ver BANN/89), por exemplo.

A segunda caracteriza-se por algoritmos implementados no espaço da tela, onde os objetos são projetados. Dentre os algoritmos destaca-se *Z-buffer*, Algoritmo de Subdivisão de Áreas e *Scan-line* (ver SUTH/74).

Teoricamente o esforço computacional para os algoritmos do espaço objeto, que compara cada objeto da cena com todos os outros, cresce com o quadrado do número de objetos (n^2). Similarmente, o trabalho para os algoritmos do espaço imagem, que compara cada objeto da cena com cada pixels da tela, teoricamente cresce com o número de objetos e o tamanho da tela (n * N). Onde N é tipicamente, 512 X 512.

Assim, podemos supor que os algoritmos do espaço objeto necessitam menos tempo do que os algoritmos do espaço imagem, para n < N. Entretanto, na prática não é isto que ocorre, os algoritmos do espaço imagem são mais eficientes pois neles é possível tirar vantagem da coerência espacial nas implementações com exploração por rastreio.

Coerência é a tendência que a imagem tem de ser localmente constante quanto às suas características visuais. Assim, se um objeto é visível em um determinado *pixel* então existe uma probabilidade muito grande de que no *pixel* vizinho (acima, ao lado ou abaixo) o objeto também esteja visível.

Esta coerência ocorre de diversas maneiras (FOLE/90, pg. 657) ressaltamos duas: por linhas, denominada Coerência de Arestas e, numa mesma linha, denominada Coerência de *Scan-Lines*. Estas duas serão melhor ilustradas nos itens que se seguem pois o *rendering* se utiliza delas para garantir sua eficiência.

O *Scan-line*, portanto é uma técnica de visualização no espaço imagem que processa os objetos da imagem segundo a ordem de exploração (y) e que são projetados num plano denominado de *view_plane*, e daí para a tela (*viewport*). *Scan-line* é um algoritmo que apresenta uma boa relação qualidade X tempo porque se beneficia bastante da implementação das idéias de coerência e por isso, é uma das técnicas mais comuns nos ambientes de animação, inclusive comerciais, onde o tempo de uma imagem é muito importante no cômputo final do tempo dispendido para a geração de uma animação.

6.2.1 O ALGORITMO *Z_BUFFER*

Z_buffer (ROGE/85, cap. 4) é um dos mais simples algoritmos de visibilidade. É uma extensão da idéia do *frame buffer* (este armazena os atributos de cor de cada *pixel*). O *Z_buffer* é uma memória usada para armazenar a profundidade, ou coordenada z, de cada *pixel* visível no espaço da imagem. Durante o processamento cada novo *pixel* tem sua profundidade (z) comparada com um valor antes calculado e armazenado no *Z_buffer*. Se for constatado que o novo *pixel* está na frente do *pixel* então armazenado, o valor da cor é armazenado no *frame buffer* e a profundidade no *Z_buffer*.

A grande vantagem deste algoritmo é a sua simplicidade sendo inclusive, independente da complexidade da cena (trata com trivialidade os casos complexos de interseção de polígonos). Assim, considerando o tamanho da imagem constante, o esforço computacional aumenta **linearmente** com a complexidade da cena. Além disso, os elementos da cena podem ser processados em ordem arbitrária quanto a profundidade, daí o esforço computacional gasto para ordenar os objetos em ordem de prioridade segundo sua profundidade (necessário em alguns algoritmos como o "Algoritmo do Pintor" - ver ROGE/85, cap. 4) poder ser eliminado.

O "gargalo" no uso desta técnica é a grande quantidade de memória necessária ao processamento de uma imagem, sendo da mesma dimensão (ou maior), que o *frame buffer*. Subdivisões podem ser feitas na imagem (a razões de 4, 16 ou mais subquadros) para sobrepor a esta dificuldade, enfatizando a potencialidade paralela deste algoritmo.

Considerando o sistema de coordenadas da mão direita, o Algoritmo *Z_buffer* é (ROGE/85, pg. 267) :

```
BEGIN
Inicializa o frame buffer com a cor do background
Inicializa o Z_buffer com o valor mínimo de z
Converter cada polígono em pares de interseções de  linhas
FOR cada pixel de cada polígono DO
      BEGIN
      Calcular a profundidade z(x, y)
      Comparar z(x,y) com o valor no Z_buffer na posição (x,y)
      IF z(x, y) > Z_buffer(x, y)
            THEN
                  BEGIN
                  Escreve a cor do polígono no frame buffer
                  Atualize Z_buffer com z(x, y)
                  END
            ELSE Não faz nada
END
```

6.2.2. *SCAN-LINE*

6.2.2.1 TIPOS DE ALGORITMOS *SCAN-LINE*

Scan-line é um algoritmo do espaço imagem que faz uso intensivo da coerência espacial de uma imagem. Esta coerência se verifica, na maioria das implementações, nas duas dimensões da imagem - x e y -. Quando ela ocorre na direção de exploração (y, vertical) é chamada de *Coerência de Arestas* e quando esta coerência ocorre ao longo de uma linha (x, horizontal) é chamada de *Coerência de Scan-Lines* capaz de produzir regiões *(span)* visíveis em uma linha.

Scan-Line, diferentememte de outros algoritmos do espaço imagem (como o *Z_buffer* que não ordenam os polígonos segundo o *display* -y-), processa os elementos da cena segundo a ordem de rastreamento do monitor, aproveitando a coerência de arestas.

Esta abordagem para o *Scan-Line* é uma evolução dos algoritmos de *Z_buffer* onde a memória de imagem é do tamanho de uma linha de exploração, somente. Portanto, é implementado tirando o máximo de proveito da coerência entre *scan-line's* daí este ser comumente chamado de *Z_buffer Scan-Line.*

Se entretanto, além das características acima, incorpora-se o tratamento dos objetos visando usufruir também da coerência de *scan-line*'s (ou de regiões, *span*) em uma mesma linha de exploração então, diz-se tratar de um *Spanning Scan-Line.*

6.2.2.2 A IDÉIA DO *SCAN-LINE*

A idéia básica do *scan-line* é **reduzir o problema tridimensional para bidimensional**. Os passos que caracterizam esta abordagem são :

a) Passar um plano (denominado *scan plane*) nas diversas linhas de exploração da imagem (*scan-line*);

b) Achar a interseção deste plano com os polígonos que compõem a cena;

c) A cada *pixel* do vetor que caracteriza a linha (chamado de *line buffer*), avaliar o atributo de cor que ele receberá através da escolha do polígono que está mais próximo do observador.

d) Visualizar os atributos armazenados no *line buffer* daquela linha e passar à próxima linha.

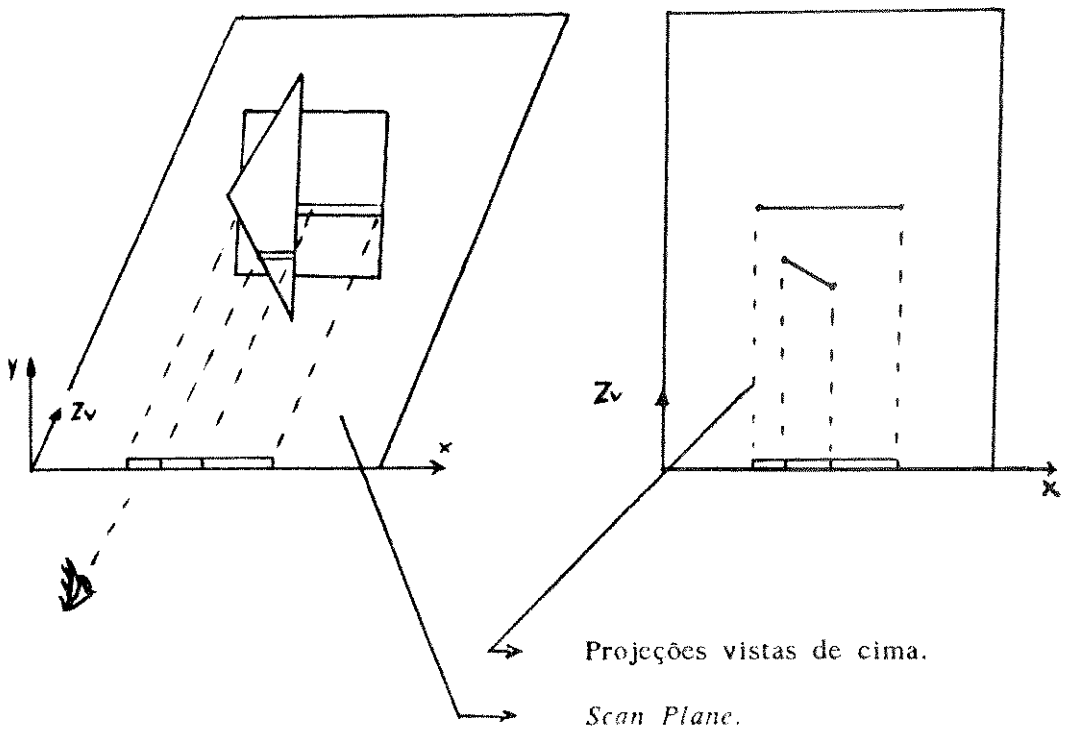


Figura 6.1: A Idéia do *Scan-Line* (HOUN/90).

Quanto a interseção do polígono com o *scan plane* nota-se que este cálculo pode ser reduzido ao cálculo da interseção de arestas com o *scan plane* daí teremos pares de interseção e estaremos caminhando no sentido do algoritmo *scan polygon converting* (ver ROGE/85).

Entretanto, graças à coerência de arestas, este cálculo é substituido por uma estrutura e ordenação em y (característica deste algoritmo) que o substituirá por incrementos ao invés dos exaustivos cálculos. E, a estrutura de dados utilizada, incopora nos seus campos esta idéia de incremento. A estrutura é preenchida a partir de dois vértices **não horizontais** e, obtém-se os seguintes parâmetros (ver figura 6.2):

```
struct {    xtop, ztop,
            dx,   dz,
            ybot,
            next, prev
        }  edge;
```

Onde,

xtop, é a coordenada x do vértice mais acima;
dx, é o incremento de x, entre duas linhas, segundo a inclinação da aresta;
ztop, é a coordenada z do vértice mais acima;
dz, é o incremento de z, entre duas linhas, segundo a inclinação da aresta;
ybot, é o valor da *scan-line* mais baixa que a aresta ainda intercepta.

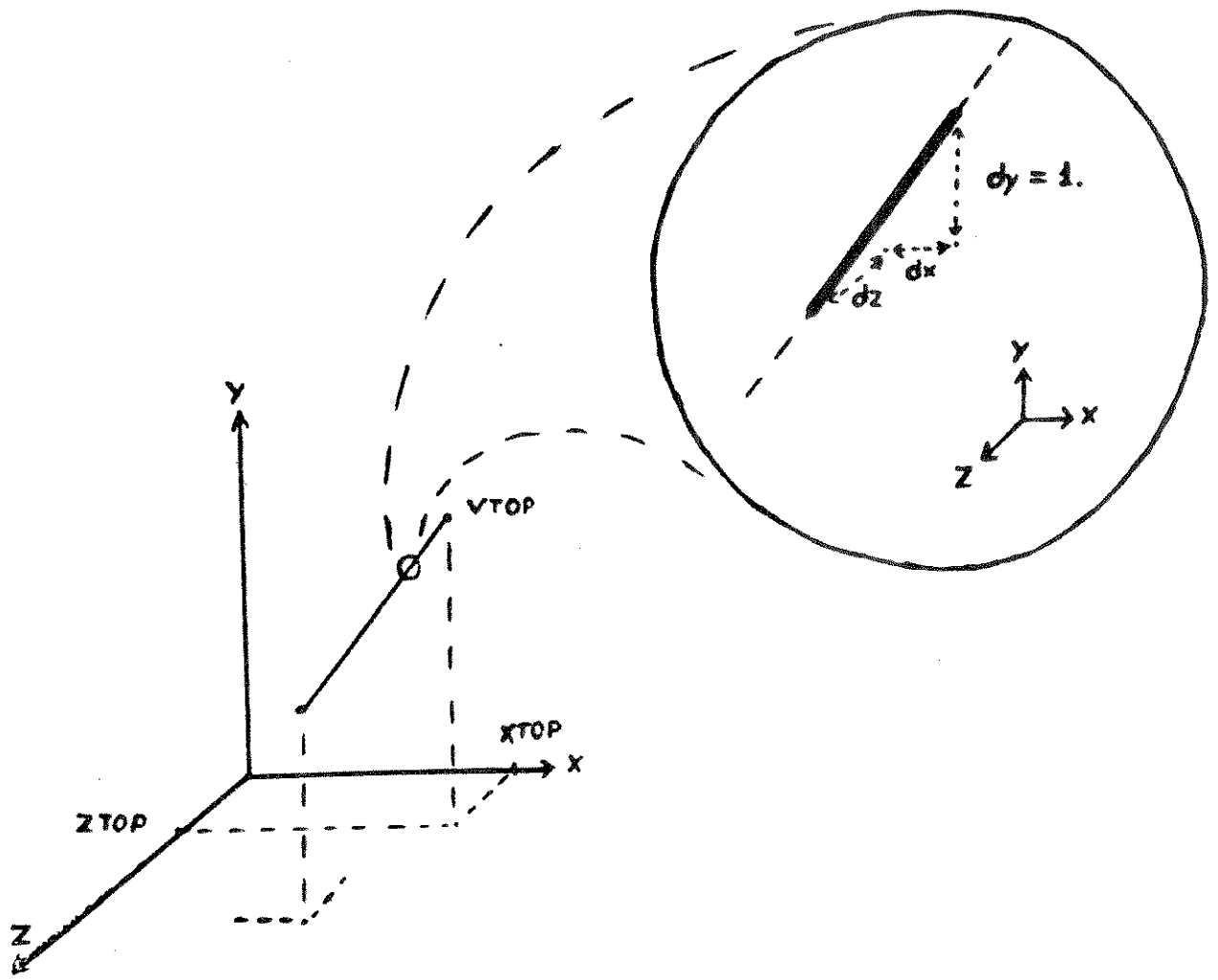


Figura 6.2 : Modelagem da Aresta (HOUN/90).

A informação *ytop* (o valor em y onde a aresta começa) é suprimida, porém ela aparece implícita na ordenação em y. Esta ordenação gera um vetor, de tantas posições

quantas forem as linhas da imagem chamado de **y_bucket**, e que nada mais é do que um escaninho onde se guardam os polígonos que passam a ficar ativos naquela *scan-line*. Veja a figura 6.3 para melhor entendimento.

Como se trabalha com polígonos é preciso ter uma tabela que armazene as informações da cor dele. Esta tabela é uma estrutura com as informações sobre as arestas do polígono, sua equação do plano, informações de cor e *delta_y* (o número de *scan-line*'s cortadas pelo polígono) :

```
struct {    *e_list;
            a, b, c, d;    /* identifica a normal */
            informações de cor;
            delta_y;
        } poly_table;
```

Os polígonos ativos numa determinada *scan-line* são mantidos numa Lista de Polígonos Ativos (LPA), a qual é usada para determinar a visibilidade no *Z_buffer*.

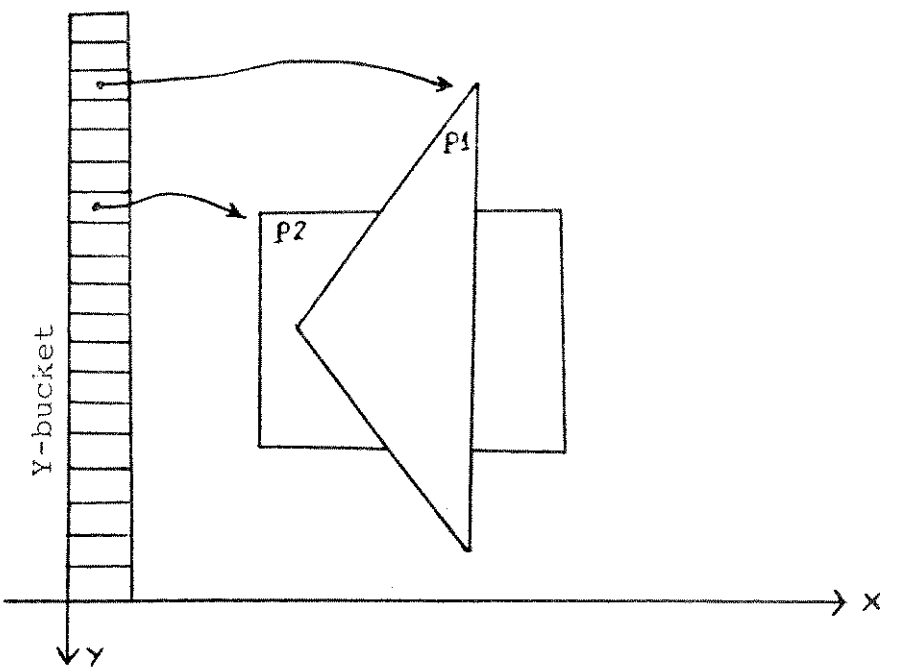


Figura 6.3 : O *Y-bucket* do *Scan-Line* (HOUN/90).

6.2.2.3 LIMITAÇÕES

O *scan-line* como algoritmo de visibilidade é muito eficiente mas, *a priori*, não há preocupação com a qualidade dos efeitos visuais pois, "tecnicamente, transparência, textura e reflexão não são parte do problema de HLHS e sim, do *rendering*" (ROGE/85, pg. 190). Isto se traduz numa dificuldade e custo computacional elevados quando se tenta implementar efeitos visuais para aumentar o realismo de uma imagem.

Esta dificuldade se deve a que os *pixels* são processados em ordem arbitrária no z, ao contrário do exigido pelos algoritmos de *anti-aliasing* por pré-filtragem, transparência, translucência, etc.

6.3 RENDERING (ver PRET/91)

6.3.1 TONALIZAÇÃO (*SHADING*)

Uma vez estabelecida a técnica de visibilidade, no caso *Scan-Line*, associaremos a ela um modelo de iluminação (o modelo de Phong) e diversos modelos de fontes de luz (estes dois últimos discutidos no item 4.4) a fim de efetuarmos o *rendering* dos objetos da cena.

Entretanto, a integração direta e simples destes modelos na visibilidade, quando aplicado ao modelo aproximador que é o modelo poliédrico, produzindo uma cor para cada face, resulta em uma aproximação visual muito limitada, pois tende a realçar a aparência poliédrica do objeto, principalmente para objetos curvos.

Este tipo de implementação da iluminação com o modelo *B-rep* é conhecido na literatura como Tonalização Constante ou Facetada (*Flat Shading*).

Neste ponto percebe-se a necessidade de incluir técnicas de tonalização de superfícies curvas (*shading*), que consigam inferir sobre as características geomé-

tricas do objeto identificando-o como uma superfície curva e daí produzir um resultado visual compatível.

Para esta finalidade, dispõe-se de algoritmos de interpolação de características visuais através de uma face e entre faces vizinhas que tenham continuidade de curvatura (ou seja, faces vizinhas que não são um canto do objeto), sem precisar alterar a estrutura do objeto (aumentar o número de faces que o definem poderia melhorar o seu *rendering*, mesmo com tonalização facetada mas, aumentaria sobremaneira o esforço computacional).

As duas abordagens mais difundidas são a Tonalização de Gouraud (*Gouraud Shading*) e a Tonalização de Phong (*Phong Shading*, não confundir com o modelo de iluminação de Phong) (ver HALL/86).

Estas técnicas são também denominadas de "suavizações incrementais" pois, se utilizam da coerência entre linhas para reduzir os cálculos das informações de iluminação à incrementos, do mesmo modo que a técnica de visibilidade *Scan-Line* o faz e por isso, inclusive, são normalmente implementados como uma extensão deste algoritmo (HALL/86, pg. 271).

A tonalização de Gouraud (GOUR/71) é mais comum, simples e rápida das duas. Ela calcula a cor (via modelo de iluminação) nos vértices das faces e interpola linearmente esta cor através da face, produzindo a variação/suavização da cor pela face.

O cálculo da cor nos vértices leva em consideração as normais das faces vizinhas que compartilham o vértice se as faces tem continuidade de curvatura.

O teste de continuidade de curvatura considera que duas faces vizinhas tem continuidade se suas normais tem uma diferença (se o cosseno entre elas) menor que um valor arbitrado (no caso 60 graus). Caso a diferença seja maior, então considera-se que as faces compõem um canto do objeto.

Mas, a solução de Gouraud não é completa, ela só é razoavel quando a superfície é pouco polida, produzindo um brilho (*highlight*) pouco acentuado (isto é, quando o coeficiente de reflexão difusa - kd - é alto e o coeficiente de reflexão especular - ks - é baixo), isto se deve a que esta técnica de tonalização despreza a variação da

normal no interior do polígono, desprezando a ênfase que a componente especular provoca sobre cor final do polígono (ver Modelo de Iluminação de Phong no item 4.4.1).

Outra limitação da técnica de Gouraud está associada às descontinuidades das faces para as aproximações de superfícies curvas. Esta descontinuidade provoca o efeito denominado *Mach Band*.

"*Onde quer que a curva de intensidade luminosa de uma superfície iluminada tenha uma flexão côncava ou convexa em relação à abcissa, este lugar em particular aparecerá mais claro ou escuro, respectivamente, que os seus arredores. (E. Mach, 1865)*" (ver em PHON/75).

As flexões mencionadas por Mach aparecem nas bordas das faces e, mesmo com uma variação contínua e linear no valor das componentes de cor, a mudança na inclinação da curva de interpolação de cores (que ocorre entre duas faces) causa o efeito *Mach Band* realçando as cores nestas bordas, destruindo, em parte, a suavização proporcionada pelo método de Gouraud.

Estes efeitos só são efetivamente eliminados quando se considera um modelo de superfície não aproximado, realmente curvo.

Outra proposta de técnica para a tonalização, também bastante difundida, é a Tonalização de Phong (PHON/75).

A proposta de Phog é interpolar as normais no interior da face e, para cada ponto do seu interior, calcular o modelo de iluminação, determinando então a cor. Portanto, precisa-se determinar, como na tonalização de Gouraud, as normais nos vértices das faces e daí efetuar o cálculo da interpolação linear de normais (e não de cores como no Gouraud) através da face. Isto significa um aumento significativo no esforço computacional.

Esta última proposta melhora sensivelmente as imagens de objetos polidos, reproduzindo com maior realismo perceptual os brilhos na superfície.

Phong também noticia uma sensível melhora (diminuição) no efeito *Mach Band* mas não a sua eliminação, devido ao fato de que é usada uma interpolação linear (que não

garante uma derivada de primeira ordem contínua para a função de tonalização, quando cruza as arestas do modelo polinomial aproximado) deixando indicado que uma interpolação de maior ordem eliminaria o efeito *Mach Band* mas seria inviável pelo tempo requerido (PHON/75, pg. 317).

Note que no processo de tonalização, diversos parâmetros entraram no nosso vocabulário, a saber, normal e cor de um vértice, interpolação de normal e cor através de uma face, tipo de interpolação (linear ou não).

Estes parâmetros não são elementos manipuláveis pelo TOOKIMA, não podendo converter-se em atores. Isto se deve a que estes são elementos internos do processo de visualização, calculados por este e "invisíveis" externamente.

Mas, as características visuais da superfície, como ka, kd, ks, podem ser atores (como visto no item 4.4.3) e, além deste, o tipo de tonalização pode ser manipulado pelo animador, não como ator mas, como uma característica deste, acessível a qualquer módulo.

6.3.2 TRANSPARÊNCIA

Além dos algoritmos de tonalização para suavizar as cores em superfícies curvas, outro efeito visual incorporado ao *rendering* é uma "pseudo-transparência".

Transparência é um fenômeno ótico que ocorre quando o material permite que a luz se transmita através dele. Esta propagação, por analogia com a luz refletida, pode ser especular ou difusa, produzindo materiais transparentes ou translúcidos.

Quando a luz atravessa o material ela sofre o fenômeno da **refração**, que é governado pela lei de Snell, produzindo distorções/deslocamentos no objeto encoberto pelo transparente, em função do ângulo de incidência do raio luminoso, do tipo de material e da espessura do mesmo.

Apesar da transparência ser facilmente implementada em algoritmos do espaço objeto (como o *Ray-Tracing*) ela pode ser corretamente simulada nos algoritmos do

espaço imagem, com algum esforço, principalmente nos algoritmos *Z_buffer Scan-Line* (ROGE/85, pg. 344).

Para eliminar muito do esforço na simulação da transparência nos algoritmos do espaço imagem, concebe-se a **pseudo-transparência**, ignorando-se os efeitos da refração, espessura do material e a curvatura de superfícies curvas (ROGE/85, cap. 5).

Esta implementação efetua diretamente uma combinação linear entre as contribuições luminosas das superfícies transparente na frente e, uma outra que esteja atrás, segundo a equação 6.1 da pseudo-transparência.

$$\mathbf{I} = \mathbf{Kt} * \mathbf{Ifrente} + (1 - \mathbf{Kt}) * \mathbf{Itrás} \qquad (6.1)$$

Onde,

I é a intensidade luminosa resultante;

Kt é o coeficiente de transparência do objeto na frente;

Ifrente é a intendidade luminosa correspondente ao polígono do objeto que está na frente;

Itrás é a intensidade luminosa provocada pelo polígono mais próximo do observador.

E, como as superfícies (parcialmente) obscurecidas também podem ser transparentes, estes cálculos são feitos retroativamente, do polígono mais distante àquele mais próximo do observador.

Apesar da limitada aproximação para a transparência, sua implementação é razoavelmente simples não encarecendo muito o tempo de processamento mas, elevando consideravelmente a flexibilidade e o resultado visual obtido, motivo pelo qual foi adotada.

6.4 ESTRUTURA DO MÓDULO *SCAN-LINE*

O módulo *Scan-line* tem como arquivo cabeçalho o scanline.h e diversos arquivos programas, todos identificáveis por começarem com as iniciais "sl_", os quais não entraremos em detalhes pois as funções dispostas nestes são, em sua grande maioria, exclusivas do sistema de visualização.

Dentro dos arquivos de programa, a função que tem interesse para qualquer programa do ProSIm que queira se integrar ao *Scan-Line* é a função scanline contida no arquivo sl_main.c que tem como argumentos, uma variável do tipo *ScanIn* (a ser apresentada abaixo), o nome da imagem a ser gerada e, a qualidade a ser empregada (determina a gama de efeitos a serem empregados no processamento) e é a responável por invocar e controlar a execução da visualização (visibilidade e *rendering*).

No arquivo cabeçalho encontramos as estruturas de dados mencionadas antes neste item e que abaixo repetimos porém, desta vez com mais detalhes relativos a atual implementação.

A estrutura da aresta é uma das principais:

```
typedef struct { int next,
                      prev,
                      right_mom,
                      right_dad,
                      ybot;
                 double xtop, ztop,
                      dx, dz;
                 Parent poly[2];
        } EIist;
```

Parent contém informações sobre a normal e/ou cor de cada aresta em relação às faces vizinhas a uma aresta, e também o índice das referidas faces na lista de polígonos. São dois "parentes" da aresta pois ela pode ter associada uma face para cada lado.

Cada elemento da lista de polígonos é descrito pela seguinte estrutura:

```
typedef struct {int       cor,
                double    cos_alfa;
                Boolean   calculated;
                Color     intensidade;
                Point     normal, baricenter, ls_dir;
                Side      in_out;
        } Poligono;
```

Onde,

```
typedef enum { OUT, IN } Side;
```

Outra estrutura importante no processamento é aquela que armazena o resultado visual de um *scan plane*:

```
typedef struct  {double   z;
                 int      n,
                          pnumber;
                 Color    intensity;
        } LineBuffer;
```

A única estrutura de dados que qualquer programa integrado ao scanline deve conhecer é a estrutura *ScanIn*, que define os parâmetros de entrada para a função scanline.

```
typedef struct { int          n_o;
                 Polyhedron   *geo[ N_MAX_OBJ];
                 Surface      *viz[ N_MAX_OBJ];
                 ShadeType    shade[ N_MAX_OBJ];
        } ScanIn;
```

6.5 CONCLUSÃO

Este módulo tem uma versão independente que representa o sistema de visualização *Scan-line* do ProSIm (assim como o *Ray-Trace* e o Radiosidade) cuja entrada é um arquivo texto (com extensão .in) enumerando os atores, posições e atributos para que compõem uma imagem estática. Este arquivo é então interpretado pelo sistema de visualização e daí é gerada a imagem. Outra versão deste módulo se encontra integrada ao TOOKIMA.

O módulo de visualização mostrado acima poderia ser suprimido do TOOKIMA, desde que este só se preocupasse com o controle e a descrição dos movimentos de uma animação. Daí, teríamos como saída uma sequência de arquivos descritores de imagens estáticas que poderia ser passada a qualquer *rendering* do projeto ProSIm que pudesse interpretá-la. Tal qual a interface entre o GeoMod e o TOOKIMA (ou seja, o GeoMod atualmente se comunica com o TOOKIMA através de seus produtos, representados por arquivos descritores de objetos - com extensão .brp -).

Entretanto, a imediata integração entre *Scan-Line* e TOOKIMA nos facilitou muito os testes pois avaliávamos uma sequência de imagens e não uma sequência de números em arquivos, o que, indubitavelmente, é mais fácil de analisar. Além disso, pudemos trabalhar de forma a obtermos um resultado visual final pronto que poderia ser interpretado como resultado de todo o processo de AMC (nos referimos a uma animação em vídeo, descrita em detalhes em RODR/92, e que está resumida no apêndice 1).

CAPÍTULO 7

CONSIDERAÇÕES FINAIS

7.1 INTRODUÇÃO

Esta conclusão apresenta os resultados obtidos pelo nosso trabalho, seguido de um breve comentário quanto a sua avaliação, posteriormente discorre sobre facilidades não abordadas na atual implementação. Por fim apresenta algumas vertentes para o desenvolvimento dos conceitos embrionários disponíveis no TOOKIMA em forma de sugestão para próximos trabalhos.

7.2 RESULTADOS OBTIDOS

Este trabalho tratou da implementação de uma versão inicial de um sistema de visualização, denominado *Scan-Line*, e de um conjunto de ferramentas para animação modelada por computador, denominado de TOOKIMA, integrado ao primeiro. Ambos fazem parte de um projeto na área de computação de imagens, denominado ProSIm, em desenvolvimento no Departamento de Computação e Automação Industrial (DCA) da FEE-UNICAMP.

Nossa principal preocupação era estabelecer e implementar conceitos e ferramentas básicas que pudessem ser extensíveis, flexíveis e de alto potencial, no sentido de prover os subsequentes pesquisadores do ProSIm de um substrato mínimo para que estes pudessem usufruir do sistema e se concentrar em aspectos mais específicos de seu interesse (relacionados, basicamente, com animação e visualização).

Com relação ao *Scan-line*, foi implementado o algoritmo básico com tonalização facetada (*flat shading*, ver HOUN/90), versão 1.0, que está sendo otimizado e estendido através de uma atividade paralela (ver PRET/91) na direção da versão 2.0.

Com relação à animação muitas características se mostraram realmente úteis na sua descrição (como trilhas *-tracks-*, manipulações dos elementos da cena, etc.) e, outras demonstraram um grande potencial depois de implementadas, uniformizando e facilitando a descrição e inclusive permitindo manipulações de modelos dinâmicos (o uso de atores para tratar quaisquer elementos da cena e manipulação da atividade das cenas, respectivamente).

O trabalho resultou em dois *softwares*: o *Scan-Line* 2.0 com cerca de 3400 linhas de código (já incluindo o trabalho em PRET/91 e com versões também para PC-DOS) e, o TOOKIMA 1.0, com aproximadamente 8400 linhas de código (incluindo o *Scan-line*) em diversos arquivos de programas subdivididos em módulos, como explicado no item 3.2.2.3 (dados obtidos pelo avaliador de linhas de código "wc" do UNIX), ambos escritos em linguagem C e executando atualmente em estações gráficas Sparc SUN-UNIX. As principais características dos *softwares* implementados são:

1) Quanto ao *Scan-line* 1.0:

- é uma versão básica para visibilidade e *rendering*;
- permite a simulação de fontes de luz através do modelo de iluminação de Phong;
- trata objetos poliédricos disponíveis em arquivos descritores (padrão ProSIm);
- processa a cena usando tonalização facetada;
- tem como entrada um arquivo texto descritor de cena;
- produz como saída três arquivos descrevendo a imagem com suas componentes primárias de cor.

2) Quanto ao TOOKIMA 1.0:

- tem como entrada um arquivo texto descrevendo a animação;
- está integrado ao *Scan-line* 2.0;
- trabalha com objetos poliédricos;
- permite a definição de quaisquer parâmetros do ambiente (inclusive da câmera e do modelo de iluminação) como ator;
- tem funções específicas para animação de elementos da cena (câmera e fontes de luz e atores);
- permite a estruturação e hierarquização de atores;

- permite a descrição de trajetórias e acelerações no contexto cinemático da animação;
- permite a descrição hierárquica da animação através de cenas e sub-cenas;
- permite o sincronismo entre cenas, a geometria temporal e a manipulação da atividade e do tipo das cenas;
- tem como saída várias imagens numeradas, descritas por arquivos de suas componentes primárias de cor.

7.3 PONTOS DE DISCUSSÃO

O uso intensivo do TOOKIMA evidenciará muitas características positivas e negativas intimamente relacionadas com o ambiene da animação. Dentre as características negativas, algumas são bastante perceptíveis para o espectador, o *temporal aliasing* é um exemplo, e outras são sentidas pelo usuário-animador em sua interação com o sistema ou quanto aos recursos envolvidos para a instalação do mesmo (como a falta de uma interface amigável e ferramentas de otimização de memória).

Abaixo nos deteremos a algumas destas características pois, na atual implementação do TOOKIMA não temos nenhuma técnica para seus tratamentos e, sem menosprezar outros trabalhos, consideramos que estas técnicas devem receber atenções o mais breve a fim de serem solucionadas.

7.3.1 *TEMPORAL ALIASING*

Um ponto de discussão que surge quando se trabalha com imagens e movimento é o *temporal aliasing*. Para melhor entendermos o problema vamos analisá-lo um pouco mais a fundo, começando pela fotografia.

Quando se faz uma foto, a intenção é captar a imagem de um objeto em um determinado instante; para tal, o obturador (componente da câmera que se abre por um curto período de tempo deixando que os raios de luz penetrem e incidam sobre a película foto-sensível do filme, ver FOTO/81) deve estar ajustado à velocidade do movimento.

"Sempre que um objeto se move em frente a câmera fotográfica, sua imagem, projetada sobre o filme, também se move. Se o movimento do objeto é rápido, ou se o obturador fica aberto por um tempo relativamente longo, esta imagem em movimento será registrada como um borrão, um tremor, ou uma imagem confusa. Um tempo de exposição curto pode "paralisar" o movimento de um objeto, mostrando sua posição num dado momento. Um tempo de exposição longo, por outro lado, pode ser usado deliberadamente para acentuar o borrão ou tremor, sugerindo a sensação de movimentos na fotografia" (FOTO/81, pg. 54).

Na filmagem o borrão é um efeito desejável pois, ele é um componente adicional de informação, impressa no filme, que tenta sobrepor-se à discretização do movimento. Na verdade ele contribui no processo de integração pela persistência retiniana para a reconstituição da cena e do movimento.

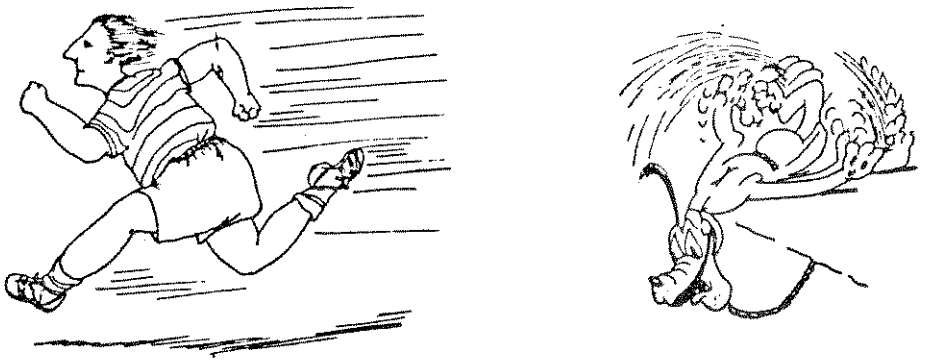
Numa AMC, obtém-se a mesma quantidade de imagens estáticas por segundo que a máquina de filmar mas, estas imagens são instantâneas, nítidas e, *a priori*, independentes da velocidade dos caracteres, isto é, como se tivéssemos usado uma máquina cujo tempo de abertura do obturador fosse infinitamente pequeno.

Esta discretização do tempo na filmagem provoca, na reprodução de movimentos rápidos, a sensação de que o objeto está se deslocando abruptamente pela cena, de um quadro para o outro, sem realçar a trajetória por ele descrita, devido a inexistência do borrão. É como se estivéssemos usando uma luz estroboscópica na filmagem, daí um destes efeitos ser também conhecido como *strobing* (BADL/87, pg. 90).

A causa destes efeitos é uma ineficiente amostragem no tempo ou, mais conhecido por *temporal aliasing*.

Algumas soluções para este tipo de *aliasing* são denominadas de *Motion Blur*, isto sugere a inclusão de borrões nas imagens nítidas e estáticas do computador.

No caso de sistemas AAC, Korein e Badler (KORE/83) demonstram como o *Motion Blur* pode ser aplicado contra o *temporal aliasing*, duas alternativas são ilustradas: a primeira é o desenho das "linhas de velocidade" *(speed-lines)* ilustradas na figura 7.1.a e, a outra é a superposição de desenhos imediata e infinitamente anteriores (*drawings overlapped*) nas partes com maior movimento, veja figura 7.1.b. Estas soluções são aproximações que na prática alcançam o seu objetivo no Desenho Animado, onde são largamente utilizadas.



(a) (b)

Figura 7.1 : *Speed-line*'s e *Overllap*'s para *Motion Blur* (KORE/83).

Para sistemas AMC a problemática é maior, "é necessário determinar não somente qual objeto será projetado sobre um determinado *pixel* mas, quando isto ocorrerá, se o objeto tem movimento" (KORE/83, pg. 378).

Uma abordagem, sugerida por Korein e Badler, determina para cada objeto uma função de intensidade temporal derivada com base nos atributos de visibilidade dos objetos. Esta função sofrerá uma convolução com um filtro apropriado, podendo ser estimada pela super-amostragem da animação, um tratamento simples e direto (segundo

MACN/87, pg. 49) porém, como os os próprios autores destacam (KORE/83, pg. 385), com alto custo computacional.

Outra abordagem interessante, citada por Korein e Badler, é propor um modelo de câmera sintética, até então muito simplório comparada às câmeras fotográficas reais, que incorpore além de efeitos de lentes e abertura, profundidade de campo e tempo finito de abertura do obturador.

Uma última abordagem, muito importante para se mencionar, é a integração de soluções para os *aliasing* temporal e espacial (GRAN/85).

Vê-se pois que, *Motion Blur* é uma ramificação muito custosa da computação de imagens e ainda em estudo, apesar das várias alternativas propostas.

7.3.2 *PLAYBACK* (ver MCLA/85, pg. 18).

O *playback* é mais um dos recursos originalmente utilizados na animação convencional, muito importante para a monitoração da qualidade do movimento (identificando necessidade de retoques para *Motion Blur*) e sincronização com a música (quando esta última pode ser reproduzida concomitantemente ao filme, ver item 2.2.6). Esta facilidade representa a posssibilidade da reprodução de rascunhos ou da imagem em diversas qualidades diretamente no "acetato eletrônico" (monitor) de maneira a produzir o efeito de movimento.

A dificuldade envolvida neste tema é um compromisso entre a capacidade de processamento e a memória otimizada, como veremos a seguir. Para reproduzir a animação consideramos três alternativas:

A primeira alternativa, REPRODUÇÃO *OFF-LINE*, implica em armazenar os quadros já gerados, em fitas de vídeo ou filme para depois reproduzi-las. Isto requer a disposição de *hardware* e *software* especiais para fazer a conversão da imagem digital para o outro meio, o que encarece prematuramente o produto final. Esta abordagem sugere uma farta quantidade de memória disponível para armazenar todas as imagens.

A segunda opção, REPRODUÇÃO EM TEMPO REAL DIRETO DA AVALIAÇÃO, significa que a medida que o computador vai calculando os diversos quadros, ele vai mostrando-os

imediatamente para o usuário sem armazená-los (economizando memória) e, toda vez que for solicitada uma reprodução o sistema tem que recalcular a animação e seus quadros. Esta solução supõe um computador de altíssima capacidade de processamento, e geralmente só é possível, quando os objetos, os movimentos dos objetos e a qualidade da imagem são simples (por exemplo, representação aramada -*wireframe*-).

A terceira opção, REPRODUÇÃO BASEADA EM RASTREIO EM TEMPO REAL, é uma tentativa de sobrepor-se às duas limitações anteriores, mantendo na memória do computador, as imagens já geradas, mas compactadas (portanto exigindo uma quantidade de memória bem menor que a primeira abordagem) e, executando-se na reprodução apenas os cálculos necessários à descompactação (esforço menor do que recalcular toda a animação novamente).

Entretanto, esta última solução de animação apresenta uma variedade de novos problemas relacionados com os requisitos para armazenamento de dados da imagem e interação do usuário (ver DENB/86, pg. 21). Isto é normalmente feito através da compactação, imagem por imagem, (usando técnicas como *run_lengh* e *cel_encod*, ROGE/85, pg. 56).

Denber e Turner (DENB/86) propõem um sistema baseado na compressão de dados no domínio temporal chamado de Compilador Diferencial de Imagens (CDI) que, se utiliza da redundância estatística entre quadros sucessivos de uma animação e não da redundância em um mesmo quadro.

A taxa de compressão depende prioritariamente do grau de redundância entre quadros sucessivos da cena a ser animada e não da complexidade da descrição analítica do movimento propriamente dito.

Em seguida discorreremos sobre aspectos da animação, que uma vez tendo uma abordagem inicial no TOOKIMA, podem ser melhorados ou extendidos.

7.4 PRÓXIMOS TRABALHOS

Dentre os aspectos de uma animação abordados pelo TOOKIMA podemos sugerir muitas intervenções sobre eles de maneira a melhorar, estender ou otimizar seu funcionamento (além dos tratamentos expostos em 7.3). Estes foram separados em duas grandes classes:

Relativos à **MODELAGEM DO AMBIENTE**, nos referimos a possíveis extensões nos modelos já existentes (principalmente do objeto), elevando o grau de abstração, incorporando características do nosso dia-a-dia (impenetrabilidade, choques, massa, etc.) de forma a caminharmos para um sistema de animação de modelo dinâmico (ver classificação no item 2.3.1.3).

Abaixo sugerimos algumas direções de trabalhos que nos parecem claras e fáceis de introduzir no sistema a fim de alcançarmos o pretendido acima.

A primeira direção seria o estudo e o aproveitamento mais intenso das manipulações sobre as atividades e tipos de cenas (condicionais, com parâmetros, variantes, etc., ver item 3.4.2). Uma extensão ou melhoramento nesta característica já pode facilitar bastante algumas animações dinâmicas.

A segunda direção é quanto a extensão dos tipos de atores incorporando tipos com restrição particulares a uma aplicação ou genéricas. Atores do tipo linha ou polilinha (LINE e POLYLINE) poderiam enriquecer o visual da cena mas requerem que a visualização também os reconheça.

Atores do tipo junta ou haste (LINK ou LIMB) poderiam ter restrições associadas ao movimento, tais como "este ator LINK só executa movimento de rotação num intervalo de -90 a +90 graus com relação ao eixo principal de ator *master* associado a ele".

Atores procedimentais (PROC) para representar atores sem formas definidas e cuja construção fosse puramente procedimental (como *Fractais*) e o próprio alvo da animação.

Outra direção de trabalhos seria o estudo sobre a comunicação entre atores e/ou cenas de uma maneira consistente que permita a iteração destes sem causar *deadlock* ou colisões. Mecanismos como *blackboard*, *mailbox*, ou *semáforos* talvez fossem interessantes facilidades para o sistema (e que abriria novas aplicações devido a a esta integração com outras áreas do conhecimento).

Outro grupo de aspectos, no sentido dos **COMPONENTES DO SISTEMA DE AMC** (modelador, *rendering* e *script*, ver figura 2.3) referem-se a outros módulos do ProSim.

A integração e manipulação de primitivas CSG na modelagem, é um recurso que pode ser usado pelo TOOKIMA para promover animações metamórficas (garantindo integridade

topológica e geométrica) e a definição de personagens complexos a serem animados.

Quanto ao *rendering*, há necessidade iminente do TOOKIMA ser independente do mesmo. Isto é, o sistema a ser utilizado para a visualização poderia ser uma escolha *a posteriori*, em função do tempo e do grau de complexidade visual requeridos. Uma solução óbvia é a geração de uma descrição padrão para cada quadro gerado pelo TOOKIMA, em forma de um arquivo texto p. ex., que pudesse ser interpretado por quaisquer um dos sistemas disponíveis no ProSIm.

Para o *script*, facilitando a descrição dos movimentos, evolutivamente sugerimos:

1) o uso de ferramentas YACC e LEX para gerar uma Nova Linguagem para animação orientada ao animador e, portanto, de mais fácil manipulação para estes, e que permitisse a abstração de comportamentos para os atores;

2) o uso de Orientação à Objetos, garantindo a extensibilidade, facilidade de manutenção, flexibilidade, troca de mensagens entre atores, etc;

3) a integração com recursos como cinemática inversa, dinâmica embutida e dinâmica inversa;

4) a confecção de uma interface gráfica que permitisse o uso do TOOKIMA, ou de sua versão evoluida, por leigos em programação e;

5) a integração com recursos de Inteligência Artificial, como p. ex., a inclusão de uma base de conhecimento que pudesse prever o comportamento de certos personagens sem a necessidade de sua exaustiva descrição.

Para finalizar enfatizamos que, o resultado do nosso esforço se consolidou e, inclusive, já produziu seus primeiros resultados na forma de uma animação de dois personagens complexos e articulados, de 67 segundos, intitulada "Mané Mosquito e Corujito" descrita parcial e sucintamente no apêndice (e na íntegra em RODR/92).

Uma melhora substancial quanto à facilidade de descrição foi alcançada quando comparamos a descrição de "Mané Mosquito e Corujito" com as descrições das animações anteriores, "O PROSIM" e "QUATRO ESTAÇÕES", produzidas no contexto do ProSIm (ver FITA/01 e SILV/91, respectivamente).

CAPÍTULO 8

BIBLIOGRAFIA

8.1 REFERÊNCIA

Nos artigos usados como referência, listados abaixo, usam-se as seguintes abreviaturas :

CG	Computer Graphics (ACM SIGGRAPH)
C&G	Computer & Graphics (Pergamon Press)
CGW	Computer Graphics World
CG&A	IEEE Computer Graphics and Applications
CACM	Communication of The ACM

AMAN/87

AMANATIDES, John.

"Realism in Computer Graphics: A Survey".

CG&A, pp 44-56.

Jan 1987.

BADL/87

BADLER, Norman I.

"3D Computer Modeling and Animation with Emphasis on Human Figures".

Course Notes of ZGDV (Zentrum für Graphische Datenverarbeitung) Seminar (based on SIGGRAPH'87 Avanced Computer Animation Course).

Oct 1987.

BANN/89

BANNWART, Cláudio V. et alli.

"Projeto PROSIM - Ray Tracing".

Relatório Interno DCA-RT-014/89. FEE-UNICAMP.

1989.

BOOT/83

BOOTH, Kellogg S. and KOCHANEK, Doris H.

"Computer Animated Films and Video".

IEEE Spectrum, pp 44-51.

Feb 1983.

BREE/87

BREEN, D. E. et al.

"The ClockWorkse: An Object-Oriented Computer Animation System".

Eurographics'87, pp 275-282.

1987.

BREE/88

BREEN, D. E. et al.

"An Object-Oriented Programming Methodology for Conventional Programming Envirommment".

Proc. Second IEE/BCS Software Engineering Conf. pp 65-72

July 1988.

BURT/71

BURTNYK, N. and WEIN, M.

"Computer Generated Key Frame Animation".

J. Soc. Motion Picture and Television Engineers 80, 3, pp 149-153.

1971.

BURT/76

BURTNYK, N. and WEIN, M.

"Interactive Skeleton Techniques for Enhancing Motion Dynamic in Key Frame Animation".

National Research Concil of Canada.

CACM, Vol 9, # 10, pp 564-569.

Oct 1976.

CATM/78

CATMULL, Edwin.

"The Problems of Computer-Assisted Animation".

CG, vol 12, # 1 e 2, pp 348-352.

Jun 1978.

CLAR/76

CLARK, J.

"Hierarchical Geometric Models for Visible Surface Algorithms".

CACM, vol 9, # 10.

Oct 1976.

COHE/85

COHEN, M. F. e GREENBERG, D. P.

"The Hemi-Cube : a Radiosity Solution for Complex Enviroments".

CG, vol 19, # 3, pp 31-40.

Jul 1985.

DENB/86

DENBER, Michel J. & TURNER, Paul M.

"A Differential Compiler for Computer Animation".

SIGGRAPH' 86, CG, vol 20, # 4, pp 21-27.

Aug 1986.

FITA/01

Fita de vídeo em VHS,

Animação "O PROSIM".

Duração 90 seg. DCA-FEE-UNICAMP.

Jul 1990.

FOLE/84

FOLEY, James D. and VAN DAM, Andries

"Fundamentals of Iteractive Computer Graphics".

Addison Wesley Publ. Co.

1984.

FOLE/90

FOLEY, James D. et al.

"Computer Graphics : Principles and Practice"

2nd ed. Addison Wesley Publ. Co.

1990.

FOTO/81

"Fotografia: Manual Completo de Arte e Técnica".

Editora Abril Cultural

1981

GOME/85

GOMÉZ, Julian E.

"TWIXT: A 3D Animation System".

C&G, vol 9, # 3, pp 291-298.

1985.

GOME/90

GOMES, Jonas Miranda e VELHO, Luiz C.

"Conceitos Básicos de Computação Gráfica".

7a. Escola de Computação - São Paulo.

Jul 1990.

GORA/84

GORAL, C. M. et al.

"Modeling The interaction of Light Between Diffuse Surfaces".

CG, vol 18, # 3, pp 213-221.

Jul 1984.

GOUR/71

GOURAUD, H.

"Computer Display of Curved Surfaces".

IEEE Transaction C-20, pp 623-628.

1971

GRAN/85

GRANT, Charles W.

"Integrated Analytic Spatial and Temporal Anti-Aliasing for Polyhedra in

4-Space".
ACM SIGGRAPH'85, vol 19, # 3, pp 79-84.
Jul 1985.

HALL/86
HALL, Roy.
"A Characterization of Illumination Models and Shading Techniques".
The Visual Computer # 2, pp 268-277.
1986.

HOUN/90
HOUNSELL, Marcelo da Silva e MAGALHÃES, Léo P.
"Relatório PROSIM - Scan-Line".
Relatório Interno RT-DCA-005/90, FEE-UNICAMP.
Jul 1990.

HOUN/91
HOUNSELL, Marcelo da Silva.
"TOOKIMA : Uma Ferramenta Para Animação Modelada Por Computador".
I Seminário Interno de Computação de Imagens do DCA.
DCA-FEE-UNICAMP.
Jul 1991.

HOUN/92
HOUNSELL, Marcelo da Silva.
"Relatório PROSIM - TOOKIMA : Implementação e Uso"
Relatório Interno RT-DCA-004/92. FEE-UNICAMP.
Em confecção.

KAUF/85
KAUFMAN, A. and AZARIA, S.
"Texture Synthesis Techiniques For Computer Graphics."
C&G, vol 9, # 2, pp 139-145.
1985.

KERN/86
KERNIGHAM, Brian W. e RITCHIE, Dennis M.
"C: A Linguagem de Programação".

EDISA, Editora Campus.
1986.

KORE/83
KOREIN, Jonathan and BADLER, Norman I.
"Temporal Anti-Aliasing in Computer Generated Animation".
CG, vol 17, # 3, pp 377-389.
Jul 1983.

KROY/87
KROYER, Bill and BERGERON, Philippe.
"3D Character Animation by Computer".
Course Notes of SIGGRAPH'87, course # 5.
1987.

LASS/87
LASSETER, John.
"Principles of Traditional Animation Applied to 3D Computer Animation".
CG, vol 21, # 4, pp 35-44.
Jul 1987.

MCLA/85
MCLACHLAN, Danil R.
"Cory: An Animation Scripting System".
Tese de Mestrado, Rensselaer Polytechnic Institute.
May 1985.

MACN/90
MACNICOL, Gregory.
"2d Animation: Alive and Well".
CGW, vol 13, # 3, pp 41-50.
Mar 1990.

MADE/92
MADEIRA, Heraldo França.
"Relatório PROSIM - GeoMod : Especificação e Implementação".
Relatório Interno RT-DCA-001/92. FEE-UNICAMP.
Em confecção.

MAGA/91

MAGALHÃES, Léo Pini e HOUNSELL, Marcelo da Silva.

"ProSIm - Um Sistema para Prototipação e Síntese de Imagens Foto-Realistas e Animação."

Relatório Interno RT-DCA-030/91. FEE-UNICAMP.

Dez 1991.

MAX/85

MAX, Nelson L. and LERNER, Douglas M.

"A two-and-a-Half-D Motion-Blur Algorithm".

CG, vol 19, # 3, pp 85-93.

Jul 1985.

MORT/85

MORTENSON, Michael E.

"Geometric Modeling".

John Wiley & Sons. New York.

1985.

NEWM/79

NEWMAN, W. M. e SPROULL, R. F.

"Principles of Interactive Computer Graphics".

McGraw-Hill Book Co.

1979.

PHON/75

PHONG, Bui-Tuong.

"Illumination for Computer Generated Images".

CACM vol 18, # 6, pp 311-317.

1975.

PICC/88

PÍCCOLO, Homero Luiz.

"Editor de Figuras Tridimensionais Dotado de Operações Poliédricas Iterativas".

Tese da Universidade de Brasília - UnB.

Jun 1988.

PRET/91

PRETO, Tania M. e MAGALHÃES, Léo P.

"Relatório PROSIM - Shading".

Relatório Interno RT-DCA-028/91. FEE-UNICAMP.

Dez 1991.

PUEY/87

PUEYO, Xavier and MENDONZA, Joan Carles.

"A New Scan Line Algorithm For The Rendering of CSG Trees".

Eurographics'87, pp 347-361.

1987.

PUEY/88

PUEYO, Xavier and TOST, Daniela.

"A Survey of Computer Animation".

C&G, Forum 7, pp 281-300.

1988.

QUEI/91

QUEIROZ, Mário S. e MAGALHÃES, Léo 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, São Paulo, pp 155-166.

Jul 1991.

REEV/81

REEVES, Willian T.

"Inbetweening for Computer Animation Utilizing Moving Points Constraints".

CG, vol 15, # 3, pp 263-269.

Aug 1981.

REYN/82

REYNOLDS, Craig W.

"Computer Animation with Scripts and Actors".

CG, vol 16, # 3, pp 289-296.

Jul 1982.

REYN/87

REYNOLDS, Craig W.

"Flocks, Herds and Schools: A Distributed Behavioral Model".

CG, vol 21, # 3, pp 25-34.

Jul 1987.

RODR/92

RODRIGUES, Andreia F.

"Relatótio PROSIM - A Animação "Mané Mosquito e Corujito"".

Relatório Interno RT-DCA-002/92. FEE-UNICAMP.

1992.

ROGE/85

ROGERS, David F.

"Procedural Elements for Computer Graphics".

McGraw-Hill Book Co. Singapore.

1985.

ROSE/88

ROSENBUSH, Judson. et al.

"Introduction to Computer Animation".

SIGGRAPH'88, COURSE # 3.

1988.

SARA/86

SARACHAN, Brion D.

"A User Interface For Iteractive Computer Animation Scripting".

Tese de Mestrado, Rensselaer Polytechnic Institute.

Dec 1986.

SHOE/85

SHOEMAKE, Ken.

"Animating Rotation With Quaternion Curves".

CG, vol 19, # 3, pp 245-254.

Jul 1985.

SILV/88

SILVA, José C. G. and ASSIS, Fidelis S. G. da.

"Linguagens de Programação, Conceitos e Avaliações".
McGraw Hill / EMBRATEL.
1988.

SILV/91
SILVEIRA, Luciana Martha e TORIGOI, Shigueo.
"Relatório PROSIM - Quatro Estações".
Relatório Interno RT-DCA-029/91. FEE-UNICAMP.
Dez 1991.

SONE/92
SONEGO, Graciano, Jr e MADEIRA, Heraldo F.
"Relatório PROSIM - Ray-Trace : Especificação e Implementação".
Relatório Interno RT-DCA-005/92. FEE-UNICAMP.
1992.

STEK/85
STEKETTE, Scott N. and BADLER, Norman I.
"Parametric Keyframe Interpolation Incorporating Kinetic Adjustment and Phrasing Control".
ACM SIGGRAPH'85, vol 19, # 3, pp 255-262.
Jul 1985.

SUTH/74
SUTHERLAND, I. E.; SPROULL, R. F. e SCHUMACKER, R. A.
"A Characterization of Ten Hidden-Surface Algorithms".
Computing Surveys, vol 6, # 1, pp 1-55.
Mar 1974.

TAKA/90
TAKAHASHI, Tadao e LIESENBERG, Hans K. E.
"Programação Orientada a Objetos".
7a. Escola de Computação - São Paulo.
Jul 1990.

THAL/83
THALMANN, Nadia Magnenat and THALMANN, Daniel.
"The Use of Hight-Level 3D Graphical Types in the Mira Animation System".

CGA, Vol 3, # 9, pp 9-16.
Dec 1983.

THAL/85a
THALMANN, Nadia Magnenat, THALMANN, Daniel and FORTIN, Mario.
"Miranim: An Extensible Director-Oriented System for the Animation of Realistic Images".
CG&A, pp 61-73.
Mar 1985.

THAL/85b
THALMANN, Nadia Magnenat and THALMANN, Daniel.
"Computer Animation, Theory and Practice".
Springer Verlag, New York. ISBN4-387-70005-6.
1985

THAL/87
THALMANN, Nadia Magnenat and THALMANN, Daniel.
"The Direction of Synthetic Actors in the Film Rendevouz à Montreal".
CG&A, pp 9-86.
Dec 1987.

THAL/88
THALMANN, Nadia Magnenat and THALMANN, Daniel.
"Advanced Course on Image Synthesis and Computer Animation".
VI Escola de Computação - Campinas - SP - Brasil.
Jul 1988.

THAL/91
THALMANN, Nadia Magnenat and THALMANN, Daniel.
"Complex Models for Animating Synthetic Actors".
CG&A, pp 32-44.
Sep 1991.

THOM/81
THOMAS, F and JOHNSTON, O.
"Disney Animation - The Ilusion of Life".
Abbeville Press, New York.
1981.

VELH/89

VELHO, Luiz.

"SCRIPTS: A Computer Animation System".

Anais do II SIBGRAPH'89, pp 509-518.

abr 1989.

WILH/87

WILHELMS, J.

"Using Dynamic Analysis for Realistic Animation of Articulated Bodies".

CG&A, vol 7, # 3, pp 12-27.

June 1987.

WILH/90

WILHELMS, J. and SKINNER, R.

"A "Notion" for Interactive Behavioral Animation Control".

CG&A, pp 14-22.

May 1990.

WHIT/84

WHITTED, Turner and COOK, Rob.

"Shading for Computer Image Synthesis".

1984.

ZELT/85

ZELTZER, David.

"Towards an Integrated View of 3D Computer Animation".

The Visual Computer, pp 249-259.

Dec 1985.

APÊNDICE

EXEMPLO : MANÉ MOSQUITO E CORUJITO

A.1 INTRODUÇÃO

Neste apêndice apresentaremos de maneira sucinta alguns detalhes relacionados a uma animação, intitulada "Mané Mosquito e Corujito", produzida com o intuito de testar toda a funcionalidade do TOOKIMA. Caso o leitor queira maiores detalhes sobre este exemplo, reporte-se a RODR/92. Caso queira maiores detalhes quanto a implementação ou uso do TOOKIMA, recomendamos reportar-se a HOUN/92.

Primeiramente apresentamos as fases que consideramos indispensáveis na produção de uma animação com o TOOKIMA. A seguir, apresentaremos algumas destas fases no contexto do exemplo, passando pela sinopse, a estruturação dos atores, até a listagem de algumas cenas.

A.2 COMO PROCEDER

Como sugestão para quem for produzir uma animação usando a ferramenta TOOKIMA, sugerimos a abordagem do trabalho de maneira sistemática e deixando a confecção do *script* e o seu respectivo teste e aperfeiçoamento, para as etapas finais.

Uma boa abordagem seria:

1) Descrever a animação em linguagem coloquial e breve;

2) Detalhar a descrição acima em linguagem de vídeo (sinopse);

3) Definir os personagens da animação;

4) Compor e, se necessário, criar os atores;

5) Definir as sequências e os cortes (ver 3.4.1);

6) Definir as cenas (no sentido do item 3.4.2) e as *Cues*;

7) Especificar os movimentos e os relógios locais;

8) Descrever as cenas que estejam ativas em um mesmo instante, do começo ao fim do tempo absoluto;

9) Testar e aperfeiçoar parâmetros e o sincronismo entre cues.

Das fases listadas acima, a seguir vamos reproduzir algumas relacionadas a animação "Mané Mosquito e Corujito".

A.3 O ROTEIRO

A animação começa com uma coruja (Corujito) que está descansando em um galho e, depois é importunada e picada por um mosquito (Mané). Em função da picada, a coruja fica atordoada permitindo a fuga do mosquito sem dar tempo para reações da primeira.

Em linguagem de vídeo diria-se : "Corujito pousado num galho, pisca duas vezes enquanto a cor de fundo muda do preto para o azul e câmera e lua se movimentam. Em seguida, Corujito segue Mané Mosquito com os olhos, bate asas, movimenta a cabeça em espiral (primeiro para cima, depois para baixo) até ficar zonzo, embaralhando os olhos. É picado por Mané Mosquito na altura da cabeça. Finalmente, volta à posição inicial e Mané Mosquito sai de cena."

A.4 CRIAÇÃO DOS ATORES

A criação dos atores foi devidamente especificadas num papel milimetrado. Desta forma, tornou-se relativamente mais simples, testar todas as funções de movimentos e inicializações das partes que compunham os atores.

Os atores foram compostos por primitivas que já existiam (esfera, cubo, cilindro, cone) e outras, foram criadas (asas, unhas, galho, lua, calota-esférica, dentes) pelo modelador provisório antes mencionado.

Nossa animação é composta de dois personagens principais, Corujito e Mané Mosquito, que foram especificados como atores do tipo NICKY.

Além destes, foi utilizado o ator do tipo POINT (a cabeça do Corujito foi deformada aplicando-se uma transformação em um ponto de seu poliedro, que pode ser visualizada quando Mané Mosquito pica a testa do Corujito).

Também compõem a animação os movimentos dos elementos do ambiente tais como: cor de fundo (a cor de fundo se altera durante a animação, começando no preto e terminando em um tom de azul), observador (mudança da sua localização), foco (centro de interesse) e lâmpada (alteração na cor e localização).

Abaixo estão relacionados alguns atores, dentre eles POINT (o ponto no rosto da coruja e a posição da fonte de luz) , POLY e NICKY, todos retirados da lista do elenco desta animação que se encontra antes das inicializações.

```
cast_nicky("Corujito",          0.,0.,0.);
cast_poly("Corpo_Corujito",     "../PROSIM/brp/cone.brp");
cast_point("ponto",             &(Ppoly(id_actor("cabeça_Corujito"))->p_list[2]));
cast_point("luz",               &lumen.lights[0] -> v -> lamp);
```

Depois de listado o elenco, é estabelecida a hierarquia ou estrutura dos atores, dentre os principais, destacamos o Corujito na figura A.1.

A.5 LISTA DE CENAS

Abaixo listamos todas as cenas que compõem a animação. Elas tem nomes mneumônicos, alusivos às ações que perfazem de forma a, quando se lê esta lista de cenas podermos nos situar na sinopse antes apresentada.

```
LIST_SCENE
    Fade_in(cue[ ]);
    PiscaOlhos_Corujito(cue[ ], abre);
    MovePupilas_Corujito(cue[ ], cima);
    PiscaOlhos_Corujito(cue[ ], fecha);
    MovePupilas_Corujito(cue[ ], baixo);
    Muda_background(cue[ ]);
    Move_lua(cue[ ]);
    MoveCamera_pan_esq(cue[ ]);
    BateAsas_Corujito(cue[ ], cima);
    BateAsas_Corujito(cue[ ], baixo);
    Entra_ManeMosquito_OlhaCorujito(cue[ ]);
    CorteSeco_ManeMosquito(cue[ ]);
    ZoomIn_Lateral_ManeMosquito(cue[ ]);
    CloseCabeca_Corujito(cue[ ]);
    OlhosCorujito_seguem_ManeMosquito(cue[ ]);
    MoveCabeca_Corujito(cue[ ], cima);
    Move_galho(cue[ ], baixo);
    MoveCabeca_Corujito(cue[ ], baixo);
    Move_galho(cue[ ], cima);
    CloseOlhos_Corujito(cue[ ]);
    AtordoaOlhos_Corujito(cue[ ], rapido);
    ZoomIn_Frontal_ManeMosquito(cue[ ]);
    Choque_ManeMosquitoCorujito(cue[ ], violento);
    Choque_ManeMosquitoCorujito(cue[ ], calmo);
    Piscada_ManeMosquito(cue[ ]);
    AtordoaOlhos_Corujito(cue[ ], devagar);
    Sai_ManeMosquito(cue[ ]);
END_LIST
```

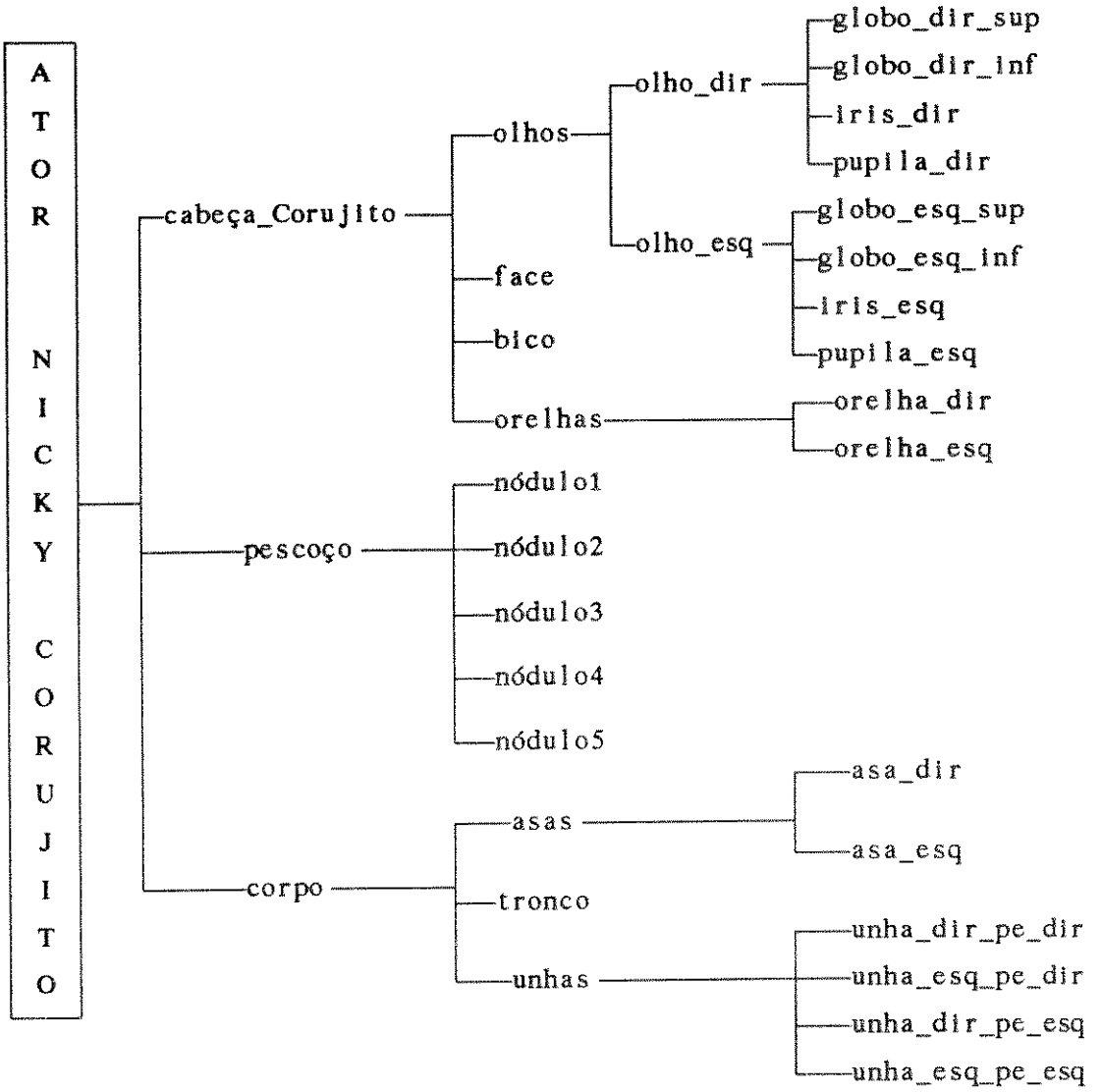


Figura A.1 : Estrutura do Ator Corujito (RODR/92).

A.6 DESCRIÇÃO DE ALGUMAS CENAS

Depois de estruturar o *Script*, passa-se a compor cada *cena* que conterá uma série de instruções, transformações, testes e avaliações de um ou mais ator(es), visando posicioná-lo(s) ou movimentá-lo(s). A seguir, detalhamos algumas *cenas* que consideramos importantes por conterem uma variedade grande de funções, mostrando assim diversos exemplos de funções disponíveis, acompanhadas da respectiva listagem.

Entra_ManeMosquito(cue)

Mané Mosquito entra em cena pela esquerda da tela.

```
void Entra_ManeMosquito(cue)
Cue *cue;
{
Vector vetor;
Point aux, ref1, ref2;
double teta1, teta2;

BEGIN_SCENE
NO_CLOCK;

/* Declaração dos atores participantes da cena */
part("pupila_dir_Corujito");
part("pupila_esq_Corujito");
part("iris_dir_corujito");
part("iris_esq_Corujito");
part("cabeca_ManeMosquito");

set_obs(-7., 6., 25.); /* Reposicionamento do observador */

/* Setagem de um vetor para a rotação no plano xy */
SET_VECTOR(vetor, 0., 0., 1.);

/* Utilização da função grasp que atraca os slaves (pupila_dir_Corujito e
pupila_esquerda_Corujito) aos masters (iris_dir_Corujito e iris_esq_Corujito)
*/
grasp("pupila_dir_Corujit"iri"iris_dir_Corujito");o");
grasp("pupila_esq_Corujito", "iris_esq_Corujito");

/* Incrementos angulares com acelerações variadas no tempo */
teta1 = -170. * acc_dec(RELATIVE, clock);
teta2 = -40. * dec_acc(RELATIVE, clock);

/* Escolha de dois pontos de referência: os centros de
gravidade das duas iris do Corujito */
ref1 = gc_actor("iris_dir_Corujito");
```

```
ref2 = gc_actor("iris_esq_Corujito");

free_rotate_actor(ref1, vetor, teta1, "iris_dir_Corujito");
free_rotate_actor(ref2, vetor, teta2, "iris_esq_Corujito");

/* Setagem de um ponto para a translação */
SET_POINT(aux, 0.5, 0.05, 0.);
translate_actor(RELATIVE, aux, "cabeca_ManeMosquito");

END_SCENE
}
```

AtordoaOlhos_Corujito

Pupilas do Corujito rotacionam embaralhadamente no plano xy, primeiro rapidamente, depois lentamente. Estes movimentos possuem acelerações variadas, quando setadas no modo rápido são *slow_in*, quando setadas no modo lento são *slow_out*.

```
void AtordoaOlhos_Corujito(cue, modo)
Cue *cue;
{
Vector vetor;
Point ref1, ref2;
double teta1, teta2;
BEGIN_SCENE;
NO_CLOCK;

/* Definição dos atores que comporão a cena */
part("iris_dir_Corujito"); part("iris_esq_Corujito");
part("pupila_dir_Corujito"); part("pupila_esq_Corujito");

set_back(10000., 30000., 40000.); /* Setagem da cor de fundo */

reset_light_c(0, 40000., 40000., 40000.);      /* resetagem da intensidade de luz */
reset_light_p(0, 0., 5., 50.);                 /* reposicionamento da luz */

/* Definição do centro de interesse da cena: cabeça do Corujito */
aim_actor("cabeca_Corujito");
```

```
SET_VECTOR(vetor, 0., 0., 1.); / Definição do plano de rotação, xy */

/* Utilização da função grasp para atracar os atores slaves (pupila_dir_Corujito,
pupila_esq_Corujito) nos atores masters (iris_dir_Corujito, iris_esq_Corujito). */
grasp("pupila_dir_Corujito", "iris_dir_Corujito");
grasp("pupila_esq_Corujito", "iris_esq_Corujito");

/* Incremento angular com acelerações variadas no tempo */
teta1 =  360. * acc_dec(RELATIVE, clock);
teta2 = -360. * dec_acc(RELATIVE, clock);

/*Setagem de pontos de referência para a rotação das iris do Corujito*/
ref1 = gc_actor("iris_dir_Corujito"); ref2 = gc_actor("iris_esq_Corujito");

if (strcmp(modo, "rapido") == 0)
 {
 set_obs(0., 3.0, 15.); /* reposicionamento do observador */
 free_rotate_actor(ref1,    vetor,    -teta1,    "iris_dir_Corujito");
 free_rotate_actor(ref2, vetor,  teta1, "iris_esq_Corujito");
 }
else {
  set_obs(0., 3.0, 10.); /* Posicionamento do observador */

  /* rotação dos atores iris no plano xy, de teta graus, de acordo com os
respectivos pontos de referências */
  free_rotate_actor(ref1, vetor, teta1, "iris_dir_Corujito");
  free_rotate_actor(ref2, vetor, teta2, "iris_esq_Corujito");
 }
END_SCENE
}
```

Sai_ManeMosquito(cue)

Afastamento da *cena* em relação ao Mané Mosquito, tendo como centro de atenção a própria *cena*, restando no final do movimento, somente a cor de fundo na tela (tom de azul).

```
void Sai_ManeMosquito(cue)
Cue *cue;
{
BEGIN_SCENE
NO_CLOCK;

part("cabeca_ManeMosquito");

zoom_out(9.); /* Movimentação do observador */
translate_actor_y(RELATIVE, 0.1, "cabeca_ManeMosquito");
aim_scene();

END_SCENE
}
```