**UMA CARACTERIZAÇÃO
DE
LINGUAGENS DE ANIMAÇÃO**

Elisa Mayumi Nishimura

Dissertação apresentada
ao
Instituto de Matemática e Estatística
da
Universidade de São Paulo

Para Obtenção do Grau de Mestre
em
Matemática Aplicada

Área de Concentração: Ciência da Computação

Orientador: Prof. Dr. Routo Terada

São Paulo, junho de 1988

Aos meus pais

Agradecimentos:

ao prof. Dr. Routo Terada, pela orientação e
compreensão,

aos meus superiores, na Itaotec e no Inpe, pelo
apoio,

aos meus amigos, pelo estímulo,

e a todos os meus mestres.

SUMÁRIO

Desenhos animados podem ser criados, realizados e testados através do computador.

Embora o objetivo das linguagens apresentadas seja apenas o entretenimento, métodos absorvidos em outras áreas, tais como robótica e simulação tem representado um grande avanço nas técnicas de animação.

Neste trabalho as linguagens de animação mais relevantes são analisadas e discutidas. Procura-se sintetizar as características mais importantes de um sistema para criação de desenhos animados.

Finalmente, propõe-se um modelo de animação baseado nas linguagens existentes.

ÍNDICE

1	Introdução	1
2	Cinema de Animação	
2.1	Cinema	4
2.2	Cinema de animação	4
2.3	Produção de um desenho animado	4
3	Trabalhos Anteriores	
3.1	Histórico	7
3.2	Classificação de sistemas de animação	6
3.3	Algumas linguagens de animação	10
3.4	Abstração de dados e linguagens orientadas a objeto	12
3.5	Linguagens de animação orientadas a objeto	14
3.6	Comparação entre algumas linguagens	18
4	Elementos de um Filme de Animação	
4.1	Requisitos básicos de um sistema para animação	23
4.2	Um modelo de animação	24
4.3	Definições	25
4.4	Proposta de implementação	28
5	A Animação	
5.1	Como é feita uma animação	38
5.2	Diversos caminhos	39
5.3	Algoritmos	40
5.4	A incorporação deste algoritmos numa linguagem	43
6	Implementação	
6.1	Resumo do módulo de animação	45
6.2	O processador	46
6.3	O gerador de quadros	47
6.4	Resumo das funções	48
7	Conclusão	
7.1	Notas sobre o trabalho	53
7.2	O futuro da animação	53
	Bibliografia	55

Capítulo 1

INTRODUÇÃO

Não há quem não goste de desenho animado. Todo mundo já viu e se encantou com os primorosos desenhos animados de Walt Disney.

Alguns trabalhos brasileiros são bastante conhecidos, desde o pioneiro Sinfonia Amazônica aos filmes atuais de Maurício de Souza, Daniel Messias (autor de comerciais de televisão) e Marcos Magalhães (que fez o curta-metragem MEOW, premiado no festival de Cannes).

Um desenho animado, feito manualmente, como veremos, é um produto de um trabalho minucioso e demorado. Estas características é que devem ter levado alguém a imaginar se a animação não poderia ser agilizada pelo computador.

Os primeiros resultados conhecidos nesta área são razoavelmente antigos. O computador já era utilizado, com muito sucesso, na produção dos desenhos da Hanna-Barbera, como, por exemplo, "Os Flintstones".

Mas o que normalmente se chama como animação por computador é um pouco diferente.

A Walt Disney Productions produziu um filme comercial, o "TRON", com várias cenas sintetizadas no computador, feitas por 4 empresas de animação americanas. E o famoso "Star Treck II" (Jornada nas Estrelas) tem belíssimas cenas criadas pela Lucasfilm Ltda.

Outros trabalhos podem ser vistos diariamente, em propagandas na TV e nas premiadíssimas vinhetas da Rede Globo, criadas por Hans Doner.

Enquanto em "Os Flintstones" o computador auxiliou na criação de cada um dos quadros que compõem a animação, no "TRON" e em "Star Treck II", os componentes do desenho foram efetivamente modelados, e a animação definida dentro do computador. E, a partir destes dados, o filme foi gerado automaticamente. Este tipo de animação é denominado **animação modelada por computador**.

Esta área é extremamente promissora e pode representar uma grande evolução na história da animação. A animação modelada direciona o trabalho do animador essencialmente na definição de caracteres e da animação, desobrigando-o da tarefa extenuante de desenhar e pintar cada quadro do desenho animado.

Se por enquanto é permitido apenas aos grandes estúdios fazer um longa-metragem de desenho animado, a animação modelada por computador pode viabilizar o surgimento de muitos pequenos estúdios de animação.

Mas ainda há um caminho a percorrer. Mesmo considerando os excelentes resultados obtidos até agora, não é inverídico afirmar que a animação modelada por computador ainda tem grandes limitações.

Compare o trabalho de Walt Disney e de Hans Doner, por exemplo.

Disney e sua imensa equipe faziam os quadros de seus filmes um a um. Já Doner usa uma linguagem de um nível bem mais alto. Ele apenas dá as características básicas de um objeto (cor, forma), descreve o movimento, e o desenho é sintetizado automaticamente.

Embora o trabalho da equipe de Disney seja bem mais árduo, o processo que ela usava era muito flexível. Certamente Doner teria dificuldades imensas para fazer a Branca de Neve no seu sistema (1).

Note que existem muitas animações computadorizadas utilizando poliedros, esferas e outras formas geométricas. Mas poucas usam elementos não "geométricos", como montanhas, árvores, nuvens e raríssimas são aquelas que tem animais e figuras humanas (que se comportam como tal).

As figuras geométricas mais simples podem ser modeladas facilmente, enquanto figuras humanas são muito mais complexas. Some-se a isto a dificuldade em descrever a locomoção desta figura: o movimento dos braços, das pernas, do tronco, da cabeça.

A dificuldade de criação de caracteres mais complexos, e a dificuldade em definir uma animação são, atualmente, os grandes limitantes que um animador vai encontrar.

No campo do entretenimento, a animação, em si, já é o objetivo final, mas na animação científica, é um meio de visualização, que auxilia na compreensão de uma experiência simulada no computador. Esta área tem despertado um interesse crescente, devido aos grandes avanços em hardware, que tornaram viáveis a simulação de muitos fenômenos físicos, meteorológicos, químicos, etc.

(1) Esta afirmação é verdadeira apenas para sistemas de animação modelada. Os sistemas de auxílio à animação "simulam" o processo manual, com algumas vantagens, com poder corrigir o desenho com mais facilidade, e automatizar a confecção de quadros intermediários.

Esta confluência de interesses é que tem sido responsável pelas maiores contribuições que a animação por computador tem recebido.

Animação por computador ainda é um campo em aberto, à espera de novas técnicas, que ampliem e facilitem o trabalho do artista.

O objetivo deste trabalho é esboçar as características desejáveis em um sistema de animação, especialmente no tocante a linguagens, através da análise dos sistemas já existentes.

A ênfase a linguagens deve-se à constatação de que, no estado atual da animação por computador, é muito importante que os sistemas sejam extensíveis. Característica esta que somente é possível através de uma linguagem. Não é proposta, entretanto, nenhuma nova linguagem, porque as melhores alternativas parecem ser a extensão uma linguagem de programação existente ou então, prover uma biblioteca contendo os recursos de animação. Neste trabalho, a segunda alternativa foi escolhida, devido à facilidade de implementação.

Outras justificativas para a preferência por linguagens são a maior facilidade de fazer uma análise funcional (sem ser confundido pelas facilidades unicamente interativas) e a quase ausência de referências bibliográficas sobre sistemas interativos para animação.

O capítulo 2 dá uma breve introdução às técnicas de cinema de animação tradicional. O capítulo 3 apresenta os sistemas de animação por computador mais conhecidos, com ênfase nas linguagens que foram objeto de estudo mais detalhado. O capítulo 4 caracteriza um modelo de animação baseado nas linguagens apresentadas. No capítulo 5, são apresentados diversos recursos de animação. Uma possível implementação do modelo proposto é discutida no capítulo 6. As conclusões estão no capítulo 7.

Capítulo 2

CINEMA DE ANIMAÇÃO

2.1 CINEMA

"A técnica de projeção cinematográfica, que reproduz imagens em movimento, baseia-se na ilusão de ótica denominada persistência da visão. Esta ilusão decorre da capacidade do olho e do cérebro humano para unir figuras, vistas em sequência muito rápida, em imagem contínua. Os objetos, que ocupam progressivamente posições estáticas diferentes em figuras sucessivas, aparecem como se tivessem movimento uniforme, desde que as imagens seja vistas em velocidades de sucessão maiores que 12 por segundo." [COMO74]

2.2 CINEMA DE ANIMAÇÃO

A animação consiste na técnica de se fotografar cada uma destas figuras (chamadas de fotogramas) separadamente.

A animação mais comum é feita com desenhos, e cada quadro do desenho animado é pintado em acetato com tinta acrílica. Algumas técnicas diferentes são descritas a seguir.

Animação "viva":

Diversos truques de cinema são feito fotografando-se as pessoas ou objetos quadro a quadro. Como cada fotograma é tirado separadamente, é muito fácil fazer uma pessoa sumir de cena ou aparecer num local inesperado. Uma animação muito conhecida, de Norman McLaren, usa esta técnica para contar os desentendimentos entre uma pessoa e uma cadeira.

Animação com recorte:

Os personagens são feitos em papel recortado, Cada parte do corpo é separada uma da outra (cabeça, olhos, perna, mãos, etc.), de modo que se possa movimentar cada membro independentemente.

Animação com bonecos:

Os bonecos são flexíveis, por exemplo, cada parte dele é feita de maneira, com as conexões feitas em arame.

Animação com massa:

Os personagens são modelados em material maleável (plasticina - a massa de modelar usada por crianças).

Nestas técnicas, ao contrário do desenho animado, é praticamente impossível reproduzir a animação pois ela é feita alterando-se a posição ou o gesto dos personagens, para cada quadro do filme.

O canadense Norman McLaren, recentemente falecido, produziu muitos filmes no National Film Board of Canada, principalmente durante as décadas de 40 e 50, e é considerado um dos mais criativos animadores.

Uma técnica que ele criou foi o da animação sem câmera. Para fazer uma animação, McLaren ia desenhando diretamente no filme, com pequenas modificações entre um quadro e outro. McLaren "sonorizava" o filme da mesma forma, fazendo riscos na banda sonora do filme.

2.3 PRODUÇÃO DE UM DESENHO ANIMADO

A criação de um desenho animado tem as seguintes etapas:

a) Criação do storyboard

O storyboard é um roteiro preliminar do desenho animado, contendo as passagens principais do filme. Cada passagem é representada por um desenho e um texto complementar explicando a cena e contendo informações sobre a trilha sonora, etc.

b) Confecção do desenho animado

O processo de confecção de um desenho animado é "top-down", isto é, começa com um resumo - o storyboard - que vai sendo detalhado aos poucos.

b1) Criação dos quadros-chave

O animador desenha todos os movimentos que acontecem no filme.

b2) Criação dos quadros intermediários

Estes quadros podem ser feitos pelo mesmo animador ou por outra pessoa, porque todos os movimentos já estão definidos pelos quadros-chave. O número de quadros intermediários colocado depende da duração que se deseja dar a cada movimento (lembrando sempre que 24 quadros equivalem a um segundo de animação).

b3) Pintura

Os desenhos, feitos originalmente sobre papel, são transferidos para o acetato (manualmente ou xerocopiados). Em seguida, os acetatos são pintados. Como o acetato é transparente, a pintura é feita pelo lado de trás, para não obscurecer o contorno.

Normalmente, os personagens são pintados separados do cenário, para evitar redesenhar o cenário para cada quadro.

c) Gravação da trilha sonora

d) Filmagem

Os quadros são filmados, um a um, sobrepondo o desenho em acetato ao cenário, adicionando efeitos especiais, se for o caso.

e) Edição e sincronismo com a trilha sonora

Esta descrição sucinta não dá muita idéia da arte e engenhosidade envolvidas num processo de animação. Além do processo mecânico descrito aqui, o animador deve ter um profundo senso de ritmo e movimento.

Capítulo 3

TRABALHOS ANTERIORES

Neste capítulo, será apresentado um histórico sobre o assunto com ênfase em linguagens. Fala-se também em linguagens orientadas a objetos, já que as recentes linguagens de animação incorporam estes conceitos. São dados alguns exemplos de linguagens e sistemas de animação.

3.1 HISTÓRICO

O primeiro filme de animação por computador conhecido foi produzido em 1961, na Bell Telephone Laboratories, por E. E. Zajac. Chamava-se "Two-Gyro Gravity - Gradient Attitude Control System" e tinha 4 minutos de duração (2).

Ainda na Bell, entre 1963 e 1967, vários outros filmes foram produzidos e alguns deles utilizaram a primeira linguagem para animação, BEFLIX, criada por Ken Knowton. Um exemplo de um comando BEFLIX é:

PAINT A,B,WRITE,2

Que significa, desenhar um retângulo, com diagonal dada por A e B, preenchendo o polígono com o padrão 2.

No final da década de 60 e durante toda a década de 70, surgiram muitos trabalhos de destaque resultantes de pesquisas e teses de doutoramento. Alguns deles estão descritos abaixo:

O GENESYS é o resultado da tese de doutoramento de Ronald Baecker no MIT em 1969, e consistia de uma linguagem para definição de figuras estáticas, uma linguagem para a especificação de mudanças e programas para produção de quadros e gravação. Neste sistema uma figura é composta de várias células. Baecker introduziu neste sistema o conceito de P-curva, através do qual ele representava a trajetória e velocidade do objeto.

O sistema interativo ANIMATOR, desenvolvido na Universidade da Pensilvânia, já permitia descrever movimentos paralelos ou sequenciais e a animação era feita através de operadores especificados através de translação, rotação e escala.

(2) Em [HALAS68] existe o extrato de um artigo do mesmo E. E. Zajac, publicado no New Scientist, de 10 de fevereiro de 1966, dando conta da existência de filmes animados em 1951, no histórico computador Whirlwind do MIT. Mas não faz referência nem às animações nem aos autores.

Em 1972, na Universidade de Utah, Edwin Catmull fez o MOP, um sistema tridimensional com algoritmos de remoção de linhas escondidas e sombreamento de figuras.

Na Universidade de Ohio, Charles Csuri liderou um grupo que desenvolveu sistemas muito conhecidos como o GRASS, de Tom De Fanti, o Anima de Csuri e o SAS de David Zeltzer.

Muitos sistemas comerciais são originários destes trabalhos em universidades, por exemplo o ZGRASS de Tom De Fanti, o ASAS de Craig Reynolds [REYNOL82].

Bruce Wallace da Universidade de Cornell, desenvolveu um sistema [WALLAC81] que permite fazer, pelo computador, todo o processo de animação tradicional, com ênfase à composição de desenhos. Este sistema é usado pelos Estúdios da Hanna-Barbera, em Hollywood.

No mesmo estilo, outro sistema bastante conhecido é o ANTICS [ANTICS83], que possui recursos para geração de quadros intermediários, zooms e biblioteca de símbolos.

A Lucasfilm - atual Pixar - que reuniu muitos destes pioneiros, como Edwin Catmull e Alvy Ray Smith, realizou muitos dos trabalhos mais célebres da (recente) história da animação por computador, como efeitos especiais para filmes de ficção científica.

3.2 CLASSIFICAÇÃO DE SISTEMAS DE ANIMAÇÃO

O termo Animação por Computador é um tanto quanto impreciso, porque o computador pode ser usado de várias formas e em fases de uma animação. Podemos classificar a animação por computador pela forma de interação e pela complexidade dos dados.

A forma de interação pode ser por meio de:

1. Linguagem

As linguagens de animação são ainda a ferramenta mais flexível para animação, principalmente se forem extensíveis.

2. Sistema interativo

É mais acessível do que uma linguagem, por que o animador não precisa saber programar, mas é menos maleável e tem menos recursos que uma linguagem.

Em complexidade de dados, classifica-se o tipo de dados que é manipulado pelo sistema. Este parâmetro é muito importante porque o tipo de dado determina as características do sistema.

Abaixo estão descritos os tipos de dados em que um sistema pode se basear. Frequentemente os sistemas são mistos e se enquadram em mais de uma categoria, mas esta classificação auxilia na compreensão das características de um sistema.

1. Por imagem

Imagem é "fotografia" de um quadro, sem qualquer informação geométrica - é descrita por uma matriz $n \times m$, que contém em cada ponto um índice de cor. Tipicamente, os desenhos gerados por sistemas pintores são imagens.

2. Por elemento geométrico plano

Os quadros são descritos pela geometria dos elementos que a compõem. Muitos sistemas de animação key-frame trabalham com elementos geométricos planos. Como estes elementos são projeções bidimensionais de objetos no espaço, existem vários problemas na geração de quadros intermediários [TORI87].

3. Por elemento geométrico tridimensional

Como no caso anterior, os quadros são descritos pela geometria dos seus elementos, só que, neste caso os elementos são descritos no espaço tridimensional, enquanto no caso anterior era descrita apenas a projeção dos elementos.

4. Por elemento físico

A mais recente tendência é a de incorporar informações físicas aos elementos. Na área de visualização, informações como coeficiente de refração e reflexão é que possibilitam que os desenhos sintetizados por computador tenham uma aparência extremamente realista. Alan Barr propôs, no último congresso do Siggraph, que as outras características físicas fossem incorporadas aos sistemas, desobrigando o animador de seguir as "leis de animação tradicional" (veja o último capítulo).

N. Magnenat-Thalmann e D. Thalmann [THALMA85] identificam as 5 fases em que o computador pode entrar:

1. Na criação de desenhos

Desenhos podem ser criados por digitalização, através de um sistema interativo ou por meio de uma linguagem.

2. Na criação do movimento

O movimento pode ser calculado por interpolação (uma vez dado os quadros-chave) ou então gerados por computador.

3. Na geração da imagem

A imagem pode ser criada usando um sistema interativo ou então geradas diretamente por computador.

4. Na filmagem
O computador pode controlar uma câmera. (Thalmann cita também a "câmera virtual" que é programada numa linguagem de animação).
5. No pós-processamento
Na edição e sincronização do desenho animado.

A área que mais tem progredido nos últimos anos é justamente a da geração de imagens.

No início, as imagens geradas por computador, eram muito simples; apenas mostravam o contorno das figuras. Algoritmos de linhas escondidas ou de superfícies escondidas [SUTHER74] tratavam de mostrar apenas a parte visível da figura.

Incentivados pelo surgimento de hardware mais poderoso - capazes de mostrar uma grande variedade de cores no vídeo, e com grande resolução - estudou-se, com muito afinco, algoritmos para visualização de imagens.

Para o desenho animado, os algoritmos de sombreamento mais usados são o de Goraud [GORAUD71] e Phong [PHONG75]. O algoritmo de ray-tracing [WHITED83], apesar dos excelentes resultados que produz, é pouco usado em desenho animado. Isto porque gerar um desenho usando-se o ray-tracing é muito demorado, considerando o número de quadros que compõem uma animação.

3.3 ALGUMAS LINGUAGENS DE ANIMAÇÃO

3.3.1 GRAMPS

O GRAMPS [DONOLS81] é um interpretador para definição e animação de figuras, desenvolvido na Universidade da Califórnia por T. J. O'Donnel e Arthur J. Olson. Embora a maior parte de suas aplicações seja em Química, ele é de propósito geral, e foi usado com sucesso para animar figuras articuladas.

A sintaxe do GRAMPS é simples: um comando é constituído de um verbo, seguido de palavras-chave e argumentos. O verbo especifica o tipo de operação, por exemplo, SAVE, ROTATE ou DRAW.

Exemplos de comandos são:

```
GET CUBE
ROTATE CUBE X 16384
```

Aonde a descrição de CUBE é recuperada do disco, e depois rodado em relação ao eixo X de um dado ângulo de rotação.

Existem três tipos de comandos: os de criação de figuras, como GET, DRAW e TEXT, transformação como ROTATE e SCALE, e os utilitários como SAVE e ADVANCE.

As figuras podem ser agrupadas, e manipuladas como se fossem um elemento único.

Uma característica interessante do GRAMPS é a sua parte interativa; as transformações que as figuras sofrem tanto podem estar associadas a valores numéricos como a dispositivos de entrada de dados como mesas digitalizadoras ou joysticks.

Associando uma determinada figura e uma transformação de rotação a um joystick, por exemplo, esta figura pode ser vista de todos os lados, apenas movendo o joystick.

3.3.2 A linguagem funcional de ARYA

Kavi Arya desenvolveu uma linguagem funcional para animação bidimensional [ARYA86]. Os objetos da linguagem são os quadros-chave (key-frames). Os elementos-chaves devem ser digitalizados por meio de um mouse, e a animação é criada usando-se funções de composição, interpolação e edição de desenhos.

A composição é feita simplesmente enumerando os diversos desenhos-chave:

```
>man = osc [man1, man2, man3, man4]
```

aonde osc é uma função que define uma sequência infinita, a partir de uma lista de polilinhas. Nesse caso, man representará a sequência infinita man1, man2, man3, man4, man1, man2, man3, ..., etc.

A interpolação é feita dando-se os nomes das sequências que devem ser interpoladas:

```
>man_to_bird = inbetween 4 (take 8 walking_man)
flying_bird
```

significa fazer a interpolação, em 4 passos, entre a oitava figura da sequência `walking_man` até a primeira da sequência `flying_bird`.

As "edições" permitem alterar o desenho original. Por exemplo `smaller` diminui o tamanho do desenho, `left` move a figura uma unidade para a esquerda, `beside` copia uma figura acima de outra e assim por diante. Estes comandos de edição pode ser compostos, criando novos tipos de edição.

Uma câmera virtual (dotada também de funções que permitem movê-la, fazer aproximações, etc.) "registra" as cenas.

3.4 ABSTRAÇÃO DE DADOS E LINGUAGENS ORIENTADAS A OBJETO

De uma maneira simplificada, pode-se dizer que uma animação é a evolução de um ou mais elementos, durante o tempo, de forma independente ou sincronizada.

Por esta definição pode-se concluir que os recursos de uma linguagem de programação normal, como procedimentos, loops, variáveis, etc., não são suficientes para uma animação. A complexidade de descrever um desenho animado justifica o fato da animação por computador ter incorporado, cada vez mais, concepções e filosofias de programação mais amplas.

Se, por um lado, a grande crítica sobre as linguagens de animação - a sua dificuldade no uso - ainda continua válida, é também verdade que, nos últimos anos, houveram grandes progressos nesta área. E o primeiro deles ocorreu quando foram incorporados os conceitos de abstração, linguagens orientadas a objeto e paralelismo.

Pode-se dizer então que, genealogicamente, algumas das atuais linguagens de animação são evoluções das linguagens gráficas e de animação anteriores e também descendentes de linguagens orientadas a objetos.

Por isso é primordial que, neste capítulo, além da descrição dos sistemas de animação mais relevantes, seja dada uma breve descrição destes conceitos.

O estímulo principal para adotar o conceito de **orientado a objeto** pode ser tanto o de facilitar a concepção e solução de um sistema [TAKAHA88], quanto o de produzir um código que seja facilmente reutilizável [COX86] [COXHUN86].

O conceito de tipo abstrato de dados, surgido nos anos 70, tentava resolver este problema. Liskov e Zilles definem:

Um tipo abstrato de dados define uma classe de objetos abstratos que é completamente caracterizada pelas operações disponíveis nestes objetos.

Quando o programador usa um tipo abstrato de dados, ele está interessado apenas no comportamento que este objeto exhibe, e não em nenhum detalhe de como este comportamento é implementado.

Os tipos abstratos de dados, tem duas grandes vantagens:

- a) Desvinculam a definição da implementação.
- b) Os dados passam a ser acessíveis apenas através das operações permitidas neste. E com isso, a interface com os dados fica uniformizada e os dados ficam protegidos contra uso inválido. Esta característica é conhecida como encapsulamento.

As linguagens orientadas a objeto implementam estes conceitos, através de:

- Objetos e mensagens
Objetos são entidades que possuem um estado interno e um comportamento. Objetos se comunicam através de mensagens. Um objeto pode mandar mensagens e receber mensagens.
- Classes
Os objetos são colocados em classes, que se dispõem hierarquicamente. O exemplo usual é o da classificação zoológica:

		doberman	REX
		pastor	
	doméstico	...	
		gato	
animal		...	
	selvagem		

Linguagens orientadas a objeto garante a existência de:

- Encapsulamento
Como um objeto só pode mudar o seu estado interno quando receber uma mensagem, o encapsulamento está garantido.
- Paralelismo
Objetos existem em paralelo; um objeto manda mensagens e espera a execução das mensagens.

- Herança

Uma subclasse tem todas as características da classe e algumas a mais. O mecanismo pelo qual uma subclasse herda os atributos da classe é chamada de herança.

É inerente a este tipo de linguagem a "criação" de novos operadores. Assim com o operador + é válido para inteiros, reais e complexos (em Fortran), numa linguagem orientada a objeto eu posso definir novos operadores e fazer com que eles seja válidos para diferentes objetos.

Por isso, a extensão de um sistema fica facilitada; pode ser feita tanto pela introdução de novos objetos, como acrescentando novos operadores aos objetos já existentes.

3.5 LINGUAGENS DE ANIMAÇÃO ORIENTADAS A OBJETO

3.5.1 ASAS

A linguagem ASAS [REYNOL82] é uma extensão do LISP para animação tridimensional. Inicialmente um trabalho acadêmico de Craig Reynolds foi, mais tarde, incorporado a um sistema comercial. Tem sido utilizada com muito sucesso na produção de filmes como o "TRON" (Walt Disney Productions) e aberturas de programas de TV.

ASAS é baseado no conceito de script e atores. O script é um programa que descreve uma sequência animada. Os participantes da animação, chamados de atores, são controlados pelo script.

Sendo uma extensão do LISP, o ASAS possui todos os recursos normais a uma linguagem de programação estruturada, tais como procedimentos, recursão, variáveis locais, loops, etc. Adicionalmente, possui objetos gráficos e recursos para definir eventos paralelos.

Os objetos gráficos do ASAS são **vector**, **color**, **polygon**, **solid**, **group**, **pov**, **subworld** e **light** (respectivamente vetor, cor, sólido, grupo, ponto de vista, submundo e luz).

vector Define uma posição no espaço tridimensional de coordenadas cartesianas (x,y,z).

color pode ser dada tanto no sistema rgb como no sistema ihs. Toda cor é uma composição de tres cores básicas, vermelho, verde e azul. No sistema rgb são dados os valores de cada uma das componentes que compõem a dada cor. No ihs uma cor é descrita pela sua intensidade, tonalidade e saturação.

polygon um polígono é descrito dando-se a sua cor e os seus vértices. Por exemplo:

```
(polygon blue
  (vector 1 0 0)
  (vector 0 1 0)
  (vector 0 0 1))
```

solid é um poliedro é representado pelos seus vértices e faces.

group conjunto de objetos gráficos.

pov ponto de vista do observador; representado por uma base vetorial e um ponto no espaço.

subworld conjunto de objetos associados a um pov.

light fonte de luz, representado por um vetor e uma cor.

Estes objetos geométricos são manipulados por operadores geométricos. Os operadores podem ser globais ou locais, dependendo se as operações se referem ao sistema de coordenadas global ou sobre o sistema de coordenadas particular do objeto.

Os operadores locais são uma extensão 3D das operações da tartaruga da linguagem LOGO [PAPERT70].

Existe um conjunto de operadores básicos que são válidos para todos os objetos geométricos. São estes: **scale**, **move**, **rotate**, **stretch** e **mirror** (respectivamente mudar escala, mover, rodar, esticar e espelhar) e os seus correspondentes para o sistema de coordenadas locais.

Cada um destes operadores permite um parâmetro que especifica o objeto no qual este operador vai atuar. Se o objeto não for dado, será considerado o objeto corrente. O objeto corrente é dado pelo operador **grasp**. Abaixo, um exemplo dado por Reynolds:

```
(grasp airplane)
(forward 0.1)
(cw 0.02)
(up 0.02)
```

```
(grasp camera)
(right pan-speed)
(zoom-out 1.01)
```

aonde:

forward move em direção ao eixo z local,

cw roda no sentido horário, em redor do eixo z local,
right roda para a direita em redor ao eixo y local,
zoom-out reduz o objeto.

O programa principal é chamado de **script** e é nele que se comanda os atores.

Basicamente o **script** tem uma série de blocos animados em que se pode inicializar e terminar a atuação dos atores. Como neste exemplo, dado por Reynolds:

```
(script spinning-cubes
  (local: (runtime 96)
    (midpoint (half runtime)))

  (animate (cue (at 0)
    (start (spin-cube-actor green)))

    (cue (at midpoint)
    (start (spin-cube-actor blue)))

    (cue (at runtime)
    (cut))))
```

onde:

animate define um bloco animado,

cue define o momento em que acontece uma modificação no roteiro,

start inicializa um ator,

cut termina um bloco animado.

Um ator é um bloco de código que é executado uma vez a cada intervalo de tempo. Um ator normalmente é responsável por um objeto visível.

A comunicação entre atores é feita através de mensagens. Mensagens podem ser recebidas ou enviadas por:

send com este operador, um ator envia uma mensagem para outro ator, um grupo de atores ou mesmo para o próprio script.

receive com este operador, um ator recebe uma mensagem.

As mensagens enviadas são colocadas numa "caixa de correio" (mail box) que é implementada como uma fila.

Os conceitos de atores e mensagens, da forma como são encarados no ASAS, foi introduzido por Carl Hewitt [HEWITT77] [AGHA86].

Para auxiliar a animação dos atores, Reynolds introduz o conceito de números animados, chamados de *newtons*. Um newton pode ser usado da mesma forma que um número, só que é automaticamente animado a cada quadro, segundo uma certa curva pré-estabelecida e parametrizada.

3.5.2 CINEMIRA

Nadia Magnenat-Thalmann e Daniel Thalmann, autores do livro "Computer Animation - Theory and Practice" são os líderes de um laboratório de animação por computador na Universidade de Montreal, responsável pelo desenvolvimento de vários sistemas (de nomes parecidos) para modelamento de figuras e animação. Entre eles as linguagens MIRA, MIRA-3D, EXPERTMIRA e CINEMIRA e o sistema MIRANIN [THALMA85d], que é constituído por uma linguagem de comandos para modelamento, chamada BODY-BUILDING, uma linguagem de comandos para animação chamada ANIMEDIT e por uma linguagem básica CINEMIRA-2.

A linguagem CINEMIRA [THALMA85] é inspirado no ASAS - tem script, atores e objetos animados básicos. O CINEMIRA é uma extensão do PASCAL.

Todos os objetos animados do CINEMIRA são descritos por um valor inicial e final, o intervalo de existência e uma lei de evolução. Por exemplo o tipo animado vetor é descrito por:

```
type TVEC = animated VECTOR;
  val <<0,10,4>>..UNLIMITED;
  time 10..13;
  law <<0,10,4>> + <<3,0,0>>*(CLOCK-10)
end;
```

Onde <<0,10,4>> é o valor inicial, o vetor existe entre o tempo 10 e o tempo 13, e o vetor se movimenta de acordo com a lei dada.

Uma variável do tipo TVEC é declarada como:

```
var VEC:TVEC;
```

Os valores inicial e final, o intervalo de existência e a lei podem ser parametrizadas, ou então serem disparadas por um evento.

O tipo de dado ATOR é declarado da seguinte forma:

```

STONE = actor(P:VECTOR,TINIT,TSTART:REAL):
  time TINIT..UNILIMITED;
  type TVEC=animated VECTOR;
  val P..<<P.X,0,P.Z>>;
  time TSTART..<<PX,0,P.Z>>;
  law P - 0.5*9.81*SQR(CLOCK-TSTART)
  end;
var VEC:TVEC;
    ICOSA:ICOSAHEDRON;
begin
  init VEC;
  create ICOSA(VEC,CFA,DIR);
  include ICOSA
end;

```

Uma câmera virtual tem a mesma sintaxe, substituindo apenas a palavra-chave **actor** por **camera**. Dentro de um tipo **camera** pode-se dar comandos que definam a janela do desenho, o tipo de perspectiva, etc.

O script é o subprograma do CINEMIRA que descreve a animação. O script é dividido em cenas. Em cada cena pode-se inicializar variáveis e atores. O comando **shoot** realiza a filmagem:

```

scene ONE:
.
.
shoot until 20
end ONE;

```

3.6 COMPARAÇÃO ENTRE ALGUMAS LINGUAGENS

Neste item, as linguagens descritas anteriormente são analisadas, em termos das classificações descritas no item 3.2, e de outras características consideradas relevantes.

Primeiramente, cabe justificar a escolha das linguagens GRAMPS, de ARYA, ASAS e CINEMIRA como objetos destas análises.

O GRAMPS e a ling. de ARYA, foram escolhidas pelas suas diversidades; enquanto o GRAMPS foi desenvolvido para aplicação científica e permite uma animação interativa e em tempo real, a ling. de ARYA é bidimensional e baseada principalmente em interpolações.

Já o ASAS e o CINEMIRA, que tem muitas semelhanças, foram escolhidas porque incorporam as características consideradas importantes num sistema de animação, sendo, sob este ponto de vista, as linguagens mais evoluídas existentes.

Cronologicamente, o ASAS é a linguagem mais antiga, os conceitos básicos da linguagem já estavam definidos em 1975 [REYNOL75] e em 1982, o ASAS já estava na versão 2.0. O CINEMIRA, inspirado no ASAS, foi implementado em 1984. Apesar de suas peculiaridades, a linguagem de ARYA aproveitou os conceitos introduzidos pelo ASAS e CINEMIRA. Esta linguagem ainda estava incompleta em 1986. O ASAS foi uma linguagem pioneira, e todas as linguagens que seguem o seu modelo são chamadas scripted-languages ou linguagens com roteiro. O artigo que descreve o GRAMPS foi publicado em 1981.

3.6.1 Tipo de sistema

tipo de sistema

	ASAS	CINEMIRA	GRAMPS	ARYA
linguagem	sim	sim	sim	sim
interativo	não	não(1)	sim(2)	sim(3)

- (1) Existe uma sub-linguagem CINEMIRA-2, integrada a um sistema voltado para usuário chamado ANIMEDIT. Mas este sistema não é interativo, é apenas uma linguagem de comandos, à moda do GRAMPS.
- (2) Existem recursos na linguagem que permitem que um certo parâmetro seja dependente de um dispositivo de entrada qualquer, como joysticks ou mesas digitalizadoras.
- (3) Existem duas funções - não implementadas na época da publicação do artigo - que permitiriam a entrada interativa de quadros-chave e de valores numéricos.

3.6.2 Complexidade de Dados

Os elementos básicos da ling. de ARYA são polilinhas (segmentos de reta conectados). O GRAMPS trabalha com elementos geométricos tridimensionais. Os objetos do ASAS e o CINEMIRA certamente incorporam informações físicas para a sintetização realista da imagem, tais como cor, índice de reflexão e refração.

Aparentemente, nenhuma das linguagens permite qualquer informação física que possa influenciar na fase de animação,

como o tipo de resistência do material, forças que agem sobre os objetos, etc.

3.6.3 Fases da Animação

fases da animação

	ASAS	CINEMIRA	GRAMPS	ARYA
Criação/desenho	sim	sim	sim(1)	sim
Criação/movimento	sim	sim	sim	sim
Geração/imagem	sim(2)	sim(2)	sim(3)	sim
Filmagem (4)	não	não	não	não
Pós-processamento	não	não	não	não

- (1) O comando DRAW permite a criação interativa de um desenho, mas a forma mais usual de entrada é através de um arquivo de descrição do desenho.
- (2) Geração de imagens realistas.
- (3) Geração de figuras em wire-frame.
- (4) [THALMA85] considera a implementação de câmera virtual como um recurso de filmagem, mas este ponto de vista não foi assumido aqui. Câmeras virtuais foram considerados recursos de animação e não de filmagem.

3.6.4 Características

Foram levantadas algumas características consideradas relevantes para um sistema de animação.

Cada um dos itens está explicado a seguir:

Tipo:

Tipo de objeto gráfico tratado, tridimensional ou bidimensional.

Ling. completa:

Indica se possui todos os recursos de uma linguagem de programação, tais como loops, procedimentos, etc.

Extensão:

Indica, se for uma linguagem completa, se é extensão de alguma linguagem de programação.

Interpretado:

Indica se a linguagem é interpretada.

Script:

Indica se possui o conceito de script.

Ator:

Indica se possui o conceito de ator.

Câmera virtual:

Indica se possui o conceito de câmera virtual.

Extensível:

Embora toda linguagem seja ser estendida, neste contexto extensibilidade que significa existem facilidades para a implementação novos elementos animados.

Tempo Real:

Indica se a animação é em tempo real.

Paralelismo:

Indica se existe paralelismo entre atores.

Comunicação:

Indica se permite comunicação entre atores.

Interpolação:

Indica se implementa interpolação.

Ciclo:

Indica que implementa ciclos. Ciclo é uma sequência de quadros, que é repetida uma atrás da outra. É um recurso tradicionalmente utilizado em animação.

Transformação:

Indica se a animação é feita definindo-se transformações sobre os elementos.

Visualização:

Indica de que forma o resultado final, o quadro, é visualizado. Realista significa que o quadro é mostrado com volume e sombreamento. Arame indica que o volume é indicado por linhas (chamado de wire-frame, em inglês).

características

	ASAS	CINEMIRA	GRAMPS	ARYA
Tipo	3D	3D	3D	2D
Ling. completa	Sim	Sim	Não	Sim(?)
Extensão	Lisp	Pascal	-	ling. própria
Interpretado	Sim	Não	Sim	Sim(?)
Script	Sim	Sim	Não	Sim
Ator	Sim	Sim	Não	Não
Câmera virtual	Sim	Sim	Não(?)	Sim
Extensível	Sim	Sim	Não	Não
Tempo Real	Não	Não	Sim	Não(?)
Paralelismo	Sim	Sim	Sim	Sim
Comunicação	Sim	Sim	Não	Não
Interpolação	Sim(1)	Sim(1)	Sim	Sim
Ciclo	Não	Não	Não	Sim
Transformação	Sim	Sim	Sim	Sim
Visualização	Realista	Realista	Arame	Contorno

(?) Informações deduzidas a partir dos artigos, mas sem nenhuma indicação clara.

(1) A interpolação não é automática; é feita através de uma curva descrita por um valor animado (newton no caso do ASAS, e vector no CINEMIRA).

Capítulo 4

ELEMENTOS DE UM FILME DE ANIMAÇÃO

4.1 REQUISITOS BÁSICOS DE UM SISTEMA PARA ANIMAÇÃO

Um sistema de animação deve ter meios para a **definição e animação** de objetos e para a **visualização** desta animação. As linguagens apresentadas anteriormente implementam, de várias formas, estas características. A preocupação com a definição e a visualização é comum a todos os sistemas gráficos. A forma de definir a evolução do objeto é específica dos sistemas animados.

A questão da animação pode ser encarada, até um certo ponto, desvinculada da parte gráfica. Observe que o ASAS introduziu o conceito de variável básica animada, com o propósito de auxiliar a animação. Esta variável, chamada newton, evolui no tempo da mesma forma que os outros objetos animados, embora não tenha nenhuma característica gráfica.

Pode-se pensar em animações inteiramente não gráficas. Por exemplo, o número de pessoas numa sala, entre as 10:00 horas e 10:30 horas, em intervalos de 1 minuto, é a animação de um valor numérico.

Assim, a animação pode ser vista como um problema de simulação, aonde se deseja descrever vários eventos, que podem ocorrer em paralelo ou sequencialmente, e cada evento pode alterar ou ser alterado por outros eventos.

Sob este ponto de vista, os requisitos básicos de um sistema de animação são:

a) Capacidade de definir eventos paralelos

Um sistema animado deve permitir a definição de eventos que ocorrem simultaneamente. Tanto no ASAS como no CINEMIRA, existem entidade animadas, os atores, que são definidos individualmente. No ASAS, dada ator é ativado pelo comando **start** e no CINEMIRA, na definição de cada ator, é dado o início e o final da animação, que garantem a possibilidade de existência paralela de atores.

b) Capacidade de sincronização

Nas linguagens vistas, a sincronização é feita de duas formas:

b1) Capacidade de um ator interferir no comportamento de outro

Eventos são dependentes e um ator deve poder condicionar a sua atitude dependendo do estado de outro ator. Deve poder também, ativar, desativar ou alterar o comportamento de um outro ator.

- b2) Divisão do script em cenas (ou tomadas)
O roteiro deve ter pontos de sincronização. No ASAS, estes pontos são dados pelo comando **cue** e no CINEMIRA pelo **shoot until**.

São características desejáveis num sistema de animação:

- a) Extensibilidade
É muito importante que o sistema permita a inclusão de novos tipos animados com facilidade, sem necessidade de alterar a codificação já existente.
- b) Animação em tempo real
A edição de uma animação descrita por programa é semelhante ao processo de depuração de um programa comum. E a animação em tempo real permite que os erros sejam corrigidos com mais facilidade.

Como o tempo necessário para a geração de uma imagem realista é muito grande, e impede a animação em tempo real, deve haver uma forma de visualização simplificada, para fins de teste.

Baseado nestas premissas, tenta-se definir um modelo de animação que possua estas características.

Não existe a preocupação em formalizar rigorosamente cada um dos conceitos, da forma que foi feito em [MALLGR82], mas somente esclarecer quais as características principais do sistema proposto. O modelo apresentado é baseado nas linguagens vistas anteriormente, em especial o ASAS e o CINEMIRA.

4.2 UM MODELO DE ANIMAÇÃO

Um filme é formado por uma sequência de tomadas e cada tomada se realiza num ambiente com um ou mais atores.

Os atores agem segundo um roteiro. O tempo na animação não é contínuo; a animação transcorre numa série de instantes de tempo com espaçamento constante. Cada um destes instantes é registrado por uma câmera, que transforma um evento tridimensional em uma projeção no espaço R^2 .

Uma tomada ocorre num ambiente fechado. Supõe-se que, a partir dos dados iniciais, a evolução da animação está determinada, sem nenhuma interferência externa. Cada um dos agentes da animação é chamado de ator.

Atores são entidades que se transformam no decorrer do tempo, capazes de tomar decisões, e capazes de modificar o ambiente (ou seja, os outros atores). Atores podem ser "sensíveis ao contexto", ou seja, a sua transformação pode depender do estado de outros atores.

Atores podem nascer, "evoluir" e morrer durante o transcorrer de uma animação.

Todos os acontecimentos de um filme estão descritos, sequencialmente, através de um programa. Este programa é chamado de script. O script define para cada tomada, os atores que fazem parte do elenco, e quais as suas atribuições. Define também qual o critério de término de uma tomada e início de outra.

Dentro do script, entre uma tomada e outra, pode-se criar, alterar ou retirar um ator do elenco.

4.3 DEFINIÇÕES

O Objeto Animado

Um objeto animado tem as seguintes características:

- **aparência**, compreendendo as características visíveis como forma, cor, posição e características não necessariamente visíveis como o material de que é feito, massa, velocidade, etc.
- **capacidade**, que é o conjunto das ações possíveis que o ator é capaz de executar. Se o objeto animado for um número inteiro, por exemplo, as ações possíveis são somar, subtrair, dividir, multiplicar, etc.
- **atribuição**, que é um ponteiro para uma função que é executada a cada instante de tempo. A capacidade de receber uma atribuição é que caracteriza o objeto animado.

O Ator

Todos as instâncias de tipos animados gráficos (com exceção da câmera), são chamados de atores.

O conjunto de todos os atores, num certo instante, é chamado de **elenco**.

Mensagens

A comunicação de um ator com o exterior se faz através de mensagens. No sistema proposto por Hewitt, as mensagens não são executadas automaticamente, são armazenadas numa caixa de correio (mailbox). Aqui o conceito de mensagem é semelhante ao do Smalltalk [SMALLT86], mensagens são como funções. Assim que uma mensagem é solicitada, ela é automaticamente executada.

O Script

O script define o encadeamento entre as tomadas, o término da tomada, e pode fazer alterações no elenco entre uma tomada e outra.

Da mesma forma que o ASAS, o script do filme é um programa. Um script tem os seguintes passos:

```

inicialização da tomada 1
animação da tomada 1
condição de término da tomada 1

inicialização da tomada 2
animação da tomada 2
condição de término da tomada 2
.
.
.
inicialização da tomada n
animação da tomada n
condição de término da tomada n
  
```

aonde:

A inicialização de uma tomada estabelece as condições iniciais de uma tomada.

A animação define as atribuições de cada um dos atores.

A condição de término é dada por um sentença lógica que, quando verdadeira, determina o final da tomada.

A Tomada

Uma tomada é a animação que se realiza num intervalo de tempo, sem interferência externa, isto é, do script.

A passagem de um instante t para o instante seguinte é chamada de **transição**.

A quantidade de atores do elenco pode variar, após uma transição, dependendo do tipo de atribuição executada.

A Câmera

A câmera é um tipo animado especial, que não faz parte do elenco. A atribuição da câmera é sempre computada após a transição do elenco de um instante para outro.

4.4 PROPOSTA DE UMA IMPLEMENTAÇÃO

Como foi visto, as linguagens ASAS e CINEMIRA tem muitas semelhanças, mesmo porque o CINEMIRA foi inspirado no primeiro. Ambas são extensões de linguagens de programação conhecidas.

É fácil acreditar que é possível fazer extensões semelhantes em outras linguagens, por exemplo em C.

Tanto o CINEMIRA como o ASAS são ricos em tipos de objetos e funções, e certamente, implementadas com muito trabalho. Mas será possível fazer uma implementação simples, baseadas nestas linguagens?

Dada a riqueza das linguagens, a resposta imediata é não. No entanto, veremos que é possível fazer um conjunto de procedimentos em C, capazes de lidar com uma versão simplificada de script e atores.

Observe-se que esta biblioteca não é, de modo algum, equivalente funcionalmente às linguagens mostradas. Ela contém apenas as funções necessárias para definir ações e sincronizá-las. No entanto, com esta biblioteca já será possível produzir um desenho animado.

O conjunto de funções é pequeno, fácil de implementar e tem características que permitem estendê-lo com facilidade.

A biblioteca tem duas partes; uma é fixa e pode ser utilizado para qualquer tipo de animação.

A outra parte da biblioteca é constituída da implementação de classes de objetos animados. A implementação destas classes deve incluir a classe câmera e pelo menos mais uma classe animada.

Novos objetos animados podem ser facilmente acrescentados à biblioteca. Procurou-se, tanto quanto possível, isolar o objeto animado, garantindo o encapsulamento dos dados.

Apesar da linguagem não possuir recursos para programação orientada a objeto, a biblioteca incorpora os conceitos de objetos e mensagens.

4.4.1 Biblioteca de definição de animação

A parte fixa é constituída pelos procedimentos que controlam os atores através de passagem de mensagens e da definição de ações animadas.

Estes procedimentos são utilizados no script que é um programa C aonde se define tudo o que deve acontecer na animação.

Objetos animados devem ser declarados como de tipo OBJ_TP, independentemente da função que ele vai exercer.

por exemplo:

```
OBJ_TP    objeto;
```

Qualquer ação em cima de um objeto é executado pelo procedimento M:

M(objeto,código-da-mensagem,par1,par2,...)

Aonde o objeto executa a função indicada por código da mensagem, com parâmetros par1, par2, ..., etc.

Por exemplo:

- Todos os objetos são capazes de receber a mensagem CRIAR e RETIRAR.
M(objeto,CRIAR,PONTO) cria um novo objeto de tipo PONTO.
M(objeto,RETIRAR) retira o objeto do elenco.
- Outras mensagens são particulares de cada tipo de objeto:
M(objeto,MOVER,x,y) pode ser válido para objetos do tipo ponto e linha mas não para outros.

Uma mensagem é apenas uma função que deve ser executada por um objeto. Os argumentos objeto e código da mensagem são obrigatórios e par1, par2, etc, são os parâmetros da mensagem, cuja quantidade e tipos variam de mensagem para mensagem.

Existem dois tipos de mensagens: as ações, que alteram o estado de um objeto, e as de consulta, que devolvem um valor como um dos parâmetros.

As mensagens de consulta também pode ser acessadas da seguinte forma:

```
origem = M_ORIGEM(objeto)
é equivalente a:
M(objeto,ORIGEM,&origem)
```

Uma atribuição é assinalada a um objeto por:

ASS(objeto,atribuição,validade)

ASS assinala, a um certo objeto, uma atribuição. Esta atribuição é um apontador para uma função f() que define o

papel do ator. *f* é uma função com argumento do tipo OBJ_TP, que é executada com o argumento *objeto*.

A atribuição é uma versão da idéia original do ASAS, de que cada objeto tem o seu bloco de código, que é executado a cada instante de tempo. A diferença é que neste proposta, é possível redefinir este bloco de código, atribuindo-se outra função para o objeto.

A validade desta atribuição pode ser:

- a) TEMPORARIA, se a atribuição só deve valer durante a tomada,
- b) GLOBAL, quando deve ser mantida mesmo após o término da tomada.

Existe apenas uma atribuição definida para um ator a cada instante. Portanto as atribuições, sejam globais ou temporárias, pode ser desassinaladas, assinalando-se uma outra atribuição. Se o ponteiro para a função atribuída for nula, significa que não existe atribuição e o objeto deve permanecer em repouso.

PARADA(condicao)

Esta função define uma condição lógica para o término de uma tomada. Num script, a ocorrência desta função divide o código entre a descrição de uma tomada e outra.

Exemplo de um programa animado

```

main()
{
/*----- Declaração dos tipos animados -----*/
OBJ_TP    camera, grande, pequeno;

/*----- TOMADA 1 -----*/

/*----- Define a situação inicial dos objetos -----*/
M(camera,CRIA,&camera);
M(camera,INIT,...);

M(grande,CRIA,&grande);
M(grande,INIT,verde,raio_grande);
M(grande,MOVABS,0,0,0);

M(pequeno,CRIA,&pequeno);
M(pequeno,INIT,azul,raio_pequeno);
M(pequeno,MOVABS,120,0,0);

/*----- Define o movimento dos objetos -----*/
ASS(grande,fmov_vell),GLOBAL);
ASS(pequeno,fmov_vel2,TEMPORARIA);

/*----- Define a condição de parada -----*/
PARADA(ORIGEM(grande).x + RAIIO(grande) >=
        ORIGEM(pequeno).x - RAIIO(pequeno));
}

/*----- TOMADA 2 -----*/
/*----- Define a situação inicial dos objetos -----*/
M(pequeno,COR,vermelho);      /* muda a cor */

/*----- Redefine o movimento de pequeno -----*/
ASS(pequeno, fmov_vel3,GLOBAL);/* para cima */

/*----- Define a condição de parada -----*/
PARADA(quadro == 100);      /* para no quadro 100 */
}

/*----- Move para a direita -----*/
fmov_vell(obj)
OBJ_TP    obj;
{
M(obj,MÓVREL,5,0,0);
}

```



```

/*----- Move para a esquerda -----*/
fmov_vel2(obj)
OBJ_TP    obj;
{
M(obj,MOVREL,-10,0,0);
}

/*----- Move para cima -----*/
fmov_vel3(obj)
OBJ_TP    obj;
{
M(obj,MOVREL,10,0,5);
}

```

4.4.2 Biblioteca de classes animadas

Esta biblioteca contém a definição das classes animadas. As funções desta biblioteca não devem ser acessadas diretamente, só apenas através de mensagens (usando a função M - veja bib. de definição da animação). Esta restrição garante a uniformidade de acesso aos objetos animados.

Estas classes estão dispostas hierarquicamente, da seguinte forma:

Objeto animado inclui, entre outros, o objeto animado gráfico. Objetos animados gráficos podem ser do tipo câmera, fonte de luz ou ator (todos os objetos gráficos que não forem nem do tipo câmera nem do tipo fonte são chamados de atores).

Um tipo de ator especial é o grupo, que agrupa um conjunto de atores.

Abaixo, estão descritas as principais ações de cada tipo gráfico. Informações de como estas características podem ser implementadas estão no capítulo 6.

Basicamente, as funções obrigatórias de um objeto animado são as ações de criação, término e de posicionamento.

Objetos animados

São funções obrigatórias para um objeto:

CRIA classe validade

Inicializa um objeto como instância de uma determinada classe. Deve ser a primeira função utilizada.

classe é o código da classe do objeto a ser instanciado. validade é TEMPORARIA se o objeto deve existir apenas durante a tomada, e GLOBAL, caso contrário.

ELIMINA

Elimina o objeto.

ANIMA atribuição validade

Assinala um determinado procedimento de animação ao ator. É equivalente à função ASS do módulo de definição da animação.

Objetos animados gráficos

Objetos animados gráficos estão definidos no espaço R^3 e tem um sistema de coordenadas locais.

Existem ações definidas sobre o sistema de coordenadas globais e sobre o sistema de coordenadas locais.

Ações locais são:

MOVREL vetor

RODREL vetor (contendo os angulos para cada eixo)

ESPREL eixo

ESCREL vetor (contendo uma constante de escala para cada eixo.

Ações globais são:

MOVABS vetor

RODABS vetor (contendo os angulos para cada eixo)

ESPABS eixo

ESCABS vetor (contendo uma constante de escala para cada eixo.

Ação de mudança de cor:

COR indices coeficientes

Esta ação altera os indices (sistema r g b) e os coeficientes de transmissão, refração e reflexão de um objeto.

SIMPLICADO

Esta ação mostra no vídeo uma visualização simplificada do objeto

A visualização realista não pode ser feita por uma ação individual porque a visualização de um objeto pode sofrer a interferência de outros objetos. Nestes casos, deve existir mensagens de consulta, de um nível mais baixo, que permitam a codificação do algoritmo dentro do objeto câmera.

Câmera virtual - Características

A função da câmera virtual é registrar, a cada instante, a cena que está sendo vista. Sob este ponto de vista esta câmera virtual é semelhante a uma câmera comum.

O significado de registrar é bastante elástico, significa que a cena "vista" pela câmera está sendo processada de alguma forma. Tanto pode ser que cada uma das cenas esteja sendo mostrada na tela do vídeo, como pode ser que um arquivo contendo a descrição da cena seja gravado em disco, para um pós-processamento.

Pode-se implementar vários tipos de câmeras, mas apenas um objeto câmera é permitido em cada instante de animação.

A câmera pode ser animada, e para isso, devem ser implementadas funções que permitam a câmera mudar de posição e direção. A câmera trata também do tipo de projeção, da janela de visualização e do plano de projeção.

As funções obrigatórias de uma câmera são:

QUADRO

Registra uma cena.

Para registrar a cena, a câmera deve acessar todos os atores que fazer parte do elenco. Se o tipo de registro que se deseja é simples, basta prover todas as classes de atores com a função MOSTRA.

Modos de registros mais complexos como uma visualização com cálculo de linhas escondidas, dependem de testes entre um ator e outro. Para fazer este tipo de cálculo, a câmera deve acessar um conjunto de funções mais específicas dos atores (e garantir que estas funções estão implementadas em todas as classes de atores).

QUADRO é executado automaticamente a cada instante de tempo, e a não ser em casos especiais, não deve ser utilizado pelo animador.

Animação da câmera:

POSICIONA

Define a posição da câmera e sua direção no espaço R^3 .

JANELA

Define a janela de visualização de uma câmera.

VIEWPORT

Define a janela de projeção da cena no dispositivo de saída.

FILTRA f

Define uma função de tratamento da imagem. Tem mais ou menos a mesma função de um filtro de uma câmera de verdade. O tipo de tratamento é bastante simples, e não trata da imagem como um tipo. Supondo-se que o dispositivo de saída é do tipo raster, f deve ser uma função:

$\text{cor-de-saída} = f(\text{cor-de-entrada}, \text{linha}, \text{coluna})$. Dada uma cor de entrada, em uma determinada posição, f devolve a cor processada para esta mesma posição.

Internamente, esta função é aplicada a todo pixel da imagem.

Com esta função é possível, por exemplo, "clarear" uma imagem ou colocar uma faixa vermelha atravessando todo o quadro.

VISUAL

Define o tipo de visualização: SIMPLIFICADA ou REALISTA.

SAIDA

Define o dispositivo de saída: ARQUIVO ou VIDEO.

Ator - Características

As funções obrigatórias de um ator são:

INSERE

Insere um ator no elenco.

RETIRA

Retira o ator do elenco.

ATIVA

Ativa a visualização do ator.

DESATIVA

Desativa a visualização do ator.

Grupo - Características

O grupo é conjunto de atores que devem ser tratados como uma entidade.

INIC n obj1 obj2 ... objn

Define um agrupamento de n atores.

Fonte de luz - Características

Este tipo animado é exclusivo para animações realistas tridimensionais. Fontes de luz são pontos no espaço que emitem luz, iluminando uma cena, e são essenciais para a sintetização de imagens realistas. Normalmente, estas fontes de luz são implementadas como objetos pontuais, não visíveis.

Uma fonte de luz deve ter a função:

COR-INT cor intensidade

Define a cor e intensidade da luz

Capítulo 5

A ANIMAÇÃO

Apesar da primeira linguagem de animação datar de 1961, e de trabalhos muito importantes terem surgido desde então, só recentemente o problema de descrever uma animação tem despertado um interesse mais generalizado. Isso pode ser constatado verificando o crescimento da quantidade de artigos sobre o assunto, publicados em revistas especializadas.

Enquanto a questão da visualização da imagem tem sido intensamente estudada por mais de 10 anos, obtendo ótimos resultados, nota-se a escassez de trabalhos realmente inovadores na área de animação. Em [THALMA85b] observa-se, muito acertadamente, que algumas animações devem o seu efeito espetacular muito mais aos algoritmos de visualização do que à animação em si.

A animação por computador demanda, ainda, um esforço muito grande por parte do animador.

Relembrando a maneira como um desenho animado é produzido manualmente, não é de se estranhar que sistemas como o ANTICS e o de Wallace se propõem a reproduzir pelo computador o processo de animação manual, seguindo as mesmas fases e os mesmos processos, sejam também demorados. Naturalmente, o computador agiliza o processo, automatizando algumas tarefas, como a geração de quadros intermediários, provendo meios para a pintura dos desenhos e facilitando enormemente a edição (alteração ou correção) da animação. Assim mesmo, o número de quadros que devem ser produzidos interativamente pelo animador continua sendo muito grande.

Em outros sistemas e linguagens, como o ASAS e o GRAMPS, a animação se faz de maneira totalmente diferente do modo manual. N. Magnenat-Thalmann e D. Thalmann chamam o primeiro enfoque de **animação auxiliada por computador** e o segundo de **animação modelada** [THALMA85].

A animação modelada permite uma automatização maior na especificação da animação, e pode automatizar totalmente a fase de visualização (3).

(3) Embora a animação assistida seja apresentada sob um aspecto um pouco negativo por este ponto de vista, ela tem grandes vantagens. A primeira é que ela é um avanço em relação ao processo manual. Um mesmo desenho animado pode ser feito mais rapidamente por bom sistema de auxílio à animação, do que seria manualmente (embora não tenha conhecimento de nenhum teste neste sentido). A segunda vantagem é justamente que qualquer desenho animado pode ser feito neste sistema. No caso da computação modelada, sempre se estará restrito aos tipos de elementos e transformações que o sistema permite. E até agora não existe nenhum sistema abrangente.

Na animação modelada, existem outros métodos que tratam a animação com enfoques muito diversos. De qualquer maneira, quanto mais interrelacionada for uma animação, maior dificuldade terá o animador para descrever como é que os seus objetos se comportam.

Diversos artigos [ZELTZE85] [WILHEL87] analisam a situação atual da animação, e indicam os caminhos para resolver impasse.

Neste capítulo, são apresentados os métodos usuais para definir uma animação e também alguns algoritmos mais recentes que tratam de casos particulares de objetos.

5.1 COMO É FEITA UMA ANIMAÇÃO

Um desenho animado é composto por cenas estáticas mostradas uma atrás da outra em sequência muito rápida. Na animação manual, como já foi visto, o animador desenha os quadros-chave, deixando para o seu assistente a confecção dos quadros intermediários. Baseado neste método de trabalho surgiram os sistemas baseados em interpolações, em que o usuário desenha os quadros-chave e deixa para o sistema a tarefa de gerar os quadros intermediários (interpolações) automaticamente.

Este método tem inconvenientes, principalmente em sistemas bidimensionais [TORI87]. Como um desenho bidimensional é uma projeção de um espaço tridimensional, algumas informações são perdidas. Um desenhista consegue fazer os quadros intermediários sem dificuldade, porque ele pode imaginar o que está escondido. Mas, pelo computador, esta tarefa fica extremamente difícil.

A questão da geração de quadros intermediários, pode ser formulada, simplificada, como:

Dada uma curva no instante t e uma curva no instante $t+k$, calcular as curvas dos instantes $t+1$, $t+2$, até $t+k-1$.

A solução tem dois passos:

- a) calcular uma associação entre os pontos da curva 1 e a da curva 2.
- b) Para cada associação, deve-se descobrir $k-1$ pontos intermediários, que serão os pontos que descrevem as curvas intermediárias.

O primeiro tipo de interpolação usado foi a interpolação linear, cujos resultados foram considerados insatisfatórios, em muitos casos. Para suavizar a animação, foi necessário que o usuário especificasse também a trajetória e a

velocidade de cada um dos pontos. Cada uma destas curvas é chamada de caminho dos pontos em movimento [REEVES81].

Diversos algoritmos foram desenvolvidos utilizando este enfoque. Entre eles o de Miura, de Coons e o de Reeves. Uma descrição de cada um dos métodos pode ser encontrada em [REEVES81].

Na animação modelada por computador, existe um método equivalente de animação de quadros-chave [SHEGRE82] [STEBAD85].

Com a especificação do caminho e da velocidade em geral se consegue bons resultados, mas este método é mais adequado aos sistemas interativos, aonde o animador pode descrever a trajetória com facilidade.

Outro problema é que, mesmo que sejam apenas os quadros-chave, o número de desenhos que precisam ser feitos pelo animador continua muito grande.

As linguagens de animação acrescentaram um novo meio de animar um figura. Ao invés de desenhar uma determinada figura em vários intervalos de tempo e eventualmente indicar o caminho desta figura, dava-se uma transformação que ocorria a cada um destes intervalos de tempo.

As tranformações mais comuns são as transformações lineares (rotação, escala, etc.) acrescidas da translação.

5.2 DIVERSOS CAMINHOS

Embora os métodos derivados da interpolação sejam ainda a maneira mais usual de animação, a década de 80 tem sido pródiga nos diferentes meios de ataque à questão da animação.

A animação, classicamente, tem sido formulado da seguinte maneira:

Dado um objeto, como descrever a maneira como este objeto evolui no espaço e se transforma, durante um certo intervalo de tempo.

E é este o problema que os métodos de interpolação se propõem a resolver.

Mas esta não é a única forma de animar um objeto, por exemplo, o "problema" da robótica pode ser formulado, de maneira muito simplificada, da seguinte forma:

Dado um robô (um objeto articulado), qual a trajetória (sujeita a um conjunto de restrições) que este deve percorrer para alcançar um determinado objetivo.

De fato, muitas vezes, ao descrever uma animação, é mais fácil dar o objetivo do que a trajetória.

Este e outros métodos são chamados por Jane Wilhelms [WILHEL87] de **animação automática**. Estes métodos liberam o animador da necessidade de especificar cada passo da animação.

Embora o objetivo das linguagens vistas aqui seja simplesmente a animação para entretenimento, ou seja a animação pela animação, métodos utilizados em robótica ou simulação viriam, por um lado, facilitar enormemente a tarefa do animador e por outro, ampliar o espectro de utilização destas linguagens.

5.3 ALGUNS ALGORITMOS

As linguagens atuais implementam inicialmente alguns tipos de objetos diferentes e possibilitam a implementação de novos tipos de objetos. Novos objetos enriquecem a linguagem e a sua diversidade amplia as possibilidades de um animador.

Os objetos mais comuns são aqueles que podem ser definidos parametricamente, assim como esferas, cubos ou superfícies suavizadas, do tipo spline ou Bézier.

No entanto, os objetos do mundo real são muito mais complexos do que isto. É difícil imaginar como uma árvore, por exemplo, pode ser descrita usando-se apenas superfícies.

Em especial, formas da natureza, como árvores, montanhas e nuvens tem sido estudadas e implementadas de forma algorítmica e não algébrica.

Portanto, os algoritmos, no contexto deste trabalho, referem-se a um conjunto particular de algoritmos que animam elementos gráficos.

Partículas

As partículas [REEVES83] implementam uma classe de objetos que não tem uma superfície bem definida, como a fumaça, o fogo ou a nuvem.

Uma partícula é fisicamente um ponto no espaço, dotado de aparência (cor, transparência, etc), velocidade e tempo de vida. A cada intervalo de tempo, a partícula se move, de acordo com a sua velocidade.

Um sistema de partículas, é constituído de um conjunto variável de partículas, em que a cada momento, novas partículas "nascem" e outras "morrem", expirado o período de existência.

O número de partículas existentes a cada momento é dado por:

$$N\text{Parts}_f = \text{Média Part}_f + \text{Rand}() \times \text{Var Parts}_f$$

onde,
 Média Part_f é o número médio de partículas,
 Var Parts_f é a variância e
 Rand() é uma função pseudo-aleatória.

Este modelo básico pode ser incrementado de várias formas, por exemplo, alterando-se o valor desta "média" para cada intervalo de tempo.

Implementações particulares de partículas foram usadas para implementar fogos de artifício e fogo. Partículas também serviram para sintetizar grama. Neste caso, ao invés de animar as partículas, toda a animação foi colocada num quadro só.

Embora sejam muito simples, as partículas são um dos recursos mais utilizados, isoladamente ou em complemento a outros modelos (veja adiante o modelo de ondas marítimas).

Fractais

A caracterização dos fractais, por Benoit Mandelbrot [MANDEL82], tem servido para modelar muitos fenômenos da natureza. Um fractal é definido como um conjunto cuja dimensão Hausdorff Besicovitch excede a dimensão topológica. Porém, informalmente se diz que fractal é um elemento que, quanto mais se olha de perto, mais parece igual. Um exemplo sempre usado é o das linhas de costa. Quando se olha no mapa, as linhas de costas foram um desenho cheio de reentrâncias e saliências, e quando se vê mesmo o contorno da costa, as pedras, rochedos e praias, formam um desenho semelhante. Se se olha mais de perto, os grãos de areia, formam o mesmo desenho.

Existem implementações de fractais para superfícies de terreno e árvores. E uma montanha gerada por um fractal é extremamente convincente, ela tem as irregularidades que seria de se esperar numa montanha real.

Linguagens Formais e Graftais

Árvores são implementadas de uma maneira interessante através de graftais - uma extensão de linguagens formais para grafos.

Um tipo de gramática especial - os sistemas de Lindenmayer, que foram criados para simular o crescimento biológico - são usados para a geração de árvores. Os sistemas de Lindenmayer são semelhantes às gramáticas comuns, exceto que não tem terminais e qualquer produção gerada pela gramática é uma palavra válida do sistema. Outra particularidade é que todas as regras de produção devem ser aplicadas ao mesmo tempo.

Um exemplo de um sistema L é dado por Alvin R. Smith:

O alfabeto é { 0, 1, [,] },
 O axioma é 0,
 As regras de produção são:
 0 -> 0, 1[0]1[0]0
 1 -> 11
 [-> [
] ->]

As primeiras produções desta linguagem são 0, 1[0]1[0]0 e 11[1[0]1[0]0]11[1[0]1[0]0]1[0]1[0]0. Estas produções são interpretadas graficamente da seguinte forma:

0 e 1 são galhos da árvore,
 e tudo que estiver entre colchetes representa um galho da árvore.

A animação do crescimento da árvore está "embutida" na própria gramática. Cada vez que se aplica as regras de produção à palavra obtida, a árvore está crescendo.

A simples visualização da gramática não dá uma representação convincente de uma árvore, porque os galhos não tem espessura e nem folhas. A "apresentação" visual da árvore é feita numa fase de pós-processamento.

Modelos para Ondas Marítimas

Modelos para animação de ondas do mar são muito mais complexos que os modelos apresentados acima. Implementações de dois modelos apresentado na Conferência do SIGGRAPH de 1986 apresentam resultados bastante convincentes. Para implementar estes modelos foram utilizados conceitos de hidrodinâmica, que devem levar em conta o contorno do fundo do mar.

O modelo para a superfície do mar apresentado por [PEACHE86] é baseado numa função $y = f(x, y, t)$, aonde (x, y, z) é o espaço cartesiano R^3 . Este modelo não permite dois valores na mesma altura. Este modelo simula inclusive o comportamento das ondas quando batem em algum obstáculo, como uma rocha.

O modelo de Fournier e Reeves [FOURNIE86] é baseado numa função do tipo $U = f(x, y, z, t)$. A força dos campos de vento é parametrizável.

Nos dois modelos, foram acrescentadas partículas para simular os pingos de água.

Controle de Comportamento e Resposta a Estímulos

A inteligência artificial é uma outra área que está bastante relacionada com a animação. As animações descritas por meio de scripts são às vezes criticadas por serem de baixo nível, no sentido de que as instruções dadas aos objetos são detalhadas passo a passo. Se os objetos fossem mais "inteligentes" e capazes de "tomar" decisões, as instruções dadas pelo script poderiam ser escritos num nível mais elevado. Entre as características desejáveis num ator estão a capacidade de condicionar o comportamento de um ator a um evento externo, e a capacidade dum ator "aprender".

Craig Reynolds, o criador do ASAS, acrescentou "inteligência" às partículas [REYNOL87]. Ele fez uma simulação de uma revoada de pássaros, aonde cada pássaro é uma partícula, dotada de poder de decisão. Cada pássaro segue a rota direcionada pelo pássaro "condutor", evitando colisões com os outros pássaros. Este sistema alia os modelos de partículas aos modelos comportamentais.

5.4 INCORPORAÇÃO DESTES ALGORITMOS NUMA LINGUAGEM

No início da história da animação modelada, todos os objetos que eram modelados por sistemas gráficos, eram compostos de planos e superfícies geometricamente determinados. A introdução de superfícies curvas, do tipo de Bézier ou Spline, contribuiu para tornar o mundo da animação mais rico.

Mas o enfoque continuava o mesmo. Devia-se criar um objeto gráfico e de alguma forma, permitir que este objeto sofresse transformações, de modo a criar uma animação.

Sob este ponto de vista, partículas, fractais e graftais representam uma evolução, porque:

- a) A definição da forma é mais algorítmica do que por estrutura de dados. A dificuldade de definir estes objetos não é proporcional à complexidade da imagem.
- b) São intrinsecamente animados.

A implementação de quaisquer dos elementos mencionados não é fácil, por exemplo, no filme Star Treck II: The Wrath of Khan, o fogo no planeta (partículas), demorou 5 homens/mês para ser codificado e testado.

Mas é importante notar que partículas e outros elementos "algorítmicos" são compatíveis com o modelo de animação proposto. De fato, a noção, introduzida pelo ASAS, de que um ator tem um bloco de código repetido a cada instante de animação, é ideal para a implementação destes elementos.

E qualquer objeto implementado é uma nova classe animada, e uma extensão do sistema de animação.

Capítulo 6

IMPLEMENTAÇÃO

Este capítulo descreve os aspectos implementacionais do modelo proposto. Apesar do modelo não ter sido implementado, mostra-se a sua viabilidade. Esta proposição baseia-se na descrição da linguagem C dada no manual do compilador Microsoft C5.0 para microcomputadores compatíveis com o IBM-PC [MICROS87], mas não faz uso de nenhuma característica particular deste compilador.

6.1 RESUMO DO MÓDULO DE ANIMAÇÃO

Segue-se um resumo do modelo descrito no capítulo 4 e em seguida, são discutidos alguns aspectos de uma possível implementação.

Para fazer um filme, o animador-programador deve codificar um script. O script é um programa C que faz uso das funções do módulo de animação, que é composto por duas bibliotecas:

Biblioteca A de definição da animação

Nesta biblioteca, estão as funções de animação, que permitem:

1. Enviar mensagens
2. Atribuir ações animadas a atores
3. Definir a condição de término da animação

Biblioteca B de classes animadas

Esta biblioteca contém a definição de várias classes animadas. O programador tem acesso indireto a estas classes através das mensagens. Para fazer uma animação ele deve estar ciente apenas das classes animadas disponíveis e das mensagens válidas para cada classe. Esta biblioteca deve conter pelo menos a definição de uma classe do tipo ator e de uma classe do tipo câmera.

A extensibilidade do módulo está nesta biblioteca. Qualquer classe acrescentada à biblioteca é automaticamente disponível ao animador. A codificação de qualquer acréscimo a esta biblioteca deve ser feita de uma forma padronizada.

A implementação do módulo de animação compreende portanto duas partes; a implementação da biblioteca de definição da animação e a implementação da biblioteca de classes animadas.

A biblioteca A, de definição da animação, tem as funções para envio de mensagens (M), animação de funções (ASS) e definição do término de uma tomada (PARADA).

A descrição sucinta destas funções está no item 6.4. Nesta implementação, a função que executa as atribuições de cada ator numa instância de tempo é o PROCESSADOR (detalhado em 6.2).

No caso da biblioteca B, o item 6.4 mostra o exemplo da implementação de um objeto tipo camera e um objeto do tipo ator.

6.2 O PROCESSADOR

Chama-se processador ao procedimento de transição de um instante da animação para outro. O processador é a "máquina" que, a partir das definições do script, executa o bloco de código correspondente a cada ator ativo no momento.

Como já foi dito, para se passar do instante t para o instante seguinte $t+1$, existe uma função de transição T tal que, $T(E_t) = E_{t+1}$. Esta função T é uma espécie de "túnel do tempo", que conduz o elenco de um instante para outro.

Para realizar a transição, basta executar, para cada ator do elenco, a atribuição correspondente. Em princípio, estas execuções podem ser vistas como se fossem executadas paralelamente. Mas alguns cuidados devem ser tomados quando existe alguma comunicação entre os atores.

No caso do ASAS a comunicação é feita através de uma mailbox, com mensagens do tipo `send` e `receive` e no CINEMIRA, a comunicação é feita através de sinalizadores que habilitam a ativação ou o término de um ator. Mas na implementação apresentada, a atribuição de um ator não tem restrições, e pode consultar ou alterar diretamente o estado de todo ator. Assim, alguns cuidados tem que ser tomados.

Por exemplo, tome-se o exemplo de dois inteiros animados chamados A e B.

Suponha que a atribuição de A é ser incrementado de 1 a cada instante e a atribuição de B é passar a valer 0, quando A for maior que 2, e que o valor inicial de A é 0, e o de B é 5.

Dependendo da ordem em que as atribuições são computadas, as transições são diferentes.

Se a atribuição A for executada antes da B:

$t=0$: $A=0$, $B=5$

(A) A=1, B=5
 (B) A=1, B=5
 t=1: A=1, B=5

(A) A=2, B=5
 (B) A=2, B=0
 t=2: A=2, B=0

Se a atribuição B for executada antes da A:

t=0: A=0, B=5
 (B) A=0, B=5
 (A) A=1, B=5
 t=1: A=1, B=5
 (B) A=1, B=5
 (A) A=2, B=5
 t=2: A=2, B=5
 (B) A=2, B=0
 (A) A=3, B=0
 t=3: A=3, B=0

Para evitar esta contradição, convencionou-se que sempre que uma atribuição B depender de um ator A, A é executado antes de B. Então, neste exemplo, as transições serão iguais à primeira alternativa.

Pode-se convencionar que, ao contrário, as atribuições dependentes são executadas antes das independentes. Mas a convenção escolhida é muito mais fácil para se implementar.

Neste esquema, não é possível a interdependência entre atores.

No entanto, existem casos em que a interdependência é necessária. Por exemplo, suponha que necessariamente, um dentre os atores A e B deve ter valor 0. A atribuição de A é algo do tipo:

Se B não vale 0, então A=0
 e a atribuição de B é:
 Se A não vale 0, então B=0

Nestes casos a atribuição deve, explicitamente, esclarecer que estes testes devem ser realizados levando-se em conta o estado atual do elenco. E a evolução da animação tanto pode ser uma forma como de outra; depende da implementação.

6.3 O GERADOR DE QUADROS

A geração de quadros é uma função do tipo animado câmera. Esta função é ativada pelo próprio processador, após a "computação" de todos os atores.

6.4 RESUMO DAS FUNÇÕES

O sistema deve manter, acessível ao usuário, uma tabela índices de tipos animados e outra de mensagens válidas. Estas tabelas assinalam (com o comando de pré-compilação `#define`) cada palavra chave a um número inteiro distinto.

Por exemplo:

```
#define CAMERA 1
#define CUBO 2
#define ESFERA 3

#define COPIAR 1
#define MOVREL 2
#define MOVABS 3
```

Existe também, interno ao sistema, uma tabela que atribui a cada índice de tipo animado, um apontador para a função que trata deste tipo. Por exemplo:

```
tabela(CAMERA) = fcamera;
tabela(CUBO) = fcubo;
```

OBJ_TP está definido como:

```
typedef char *OBJ_TP;
```

6.4.1 Biblioteca de definição da animação

```

/*-----*/
/* PARADA define a condição de parada de uma tomada. */
/* A parada não é uma função, é uma macro definida pelo */
/* pré-compilador */
/*-----*/
#define PARADA(condição) /* condição é uma expr. lógica */
    While (!condição)
    {
        PROCESSADOR(elenco);
        M(câmera,QUADRO);
    }

```

```

/*-----*/
/* M envia a mensagem MENSAGEM para o objeto animado ATOR */
/*-----*/
M(ator,mensagem,par1,par2,....)

```

```

ANIM_TP    ator;
int        mensagem;

{
    if (mensagem == CRIAR)
    {
        f = "apontador para a função correspondente à
            par1";
        (*f)(ator,CRIAR);
    }
    else
    {
        f = "apontador para a função correspondente à
            classe(ator)";
        (*f)(ator,mensagem,par1,par2,...)
    }
}

```

A função M tem um número variável de argumentos; os dois primeiros argumentos, ator e mensagem são obrigatórios, os outros dependem da mensagem. Existem mensagens que não tem parâmetros, outros necessitam de um parâmetro do tipo inteiro, outros de dois parâmetros reais, e assim por diante.

O comp. Microsoft possui tres macros que lidam com listas de argumentos de tamanho variável. Estas macros seguem a proposta de normalização C do ANSI, de modo que esta solução não é particular para este compilador.


```

/*-----*/
/* ASS atribui uma função animada a um ator */
/*-----*/
ASS(ator, função, validade)

    ANIM_TP ator;           /* Identificador do ator */
    FUNC_TP função;        /* apontador para funcao */
    int     validade       /* validade da atribuicao */
    {
        FUNC_TP f;

        f = "apontador para a função correspondente à
            classe(ator)";
        (*f)(ator, ASSINALA, função, validade)
    }

```

6.4.2 Biblioteca de classes animadas

Esta biblioteca contém a definição das classes animadas.

Cada objeto animado é acessível apenas por meio de uma mensagem. Implementações adicionais de objetos devem prover a capacidade do objeto de responder às mensagens de maneira padronizada.

Em termos de implementação, internamente, cada classe possui uma definição de uma structure do C e um conjunto de procedimentos que manipulam esta structure.

Cada vez que um objeto é instanciado, cria-se uma structure particular ao objeto.

Para criar uma instância de um objeto, deve-se declarar uma variável do tipo animado e usar a função CRIAR. Esta função aloca espaço em memória para conter estrutura contendo as características particulares desta instância.

```

/*-----*/
/* Exemplo da implementação de um tipo animado ATOR */
/*-----*/

```

```

typedef
{
    int tipo;
    vetor origem;
    .
    .
} ESFERA_TP;

```

```

fesfera(ator, mensagem, par1, par2, ...)

```

```

OBJ_TP *objeto;
int mensagem;
{
    vetor vet;

    if (mensagem == CRIAR)
    {
        *objeto = malloc(sizeof(ESFERA_TP));
        objeto->tipo = ESFERA;
    }
    else {
        switch(mensagem)
        {
            case MOVREL:
                vet = "leitura de par1 como tipo vetor";
                objeto->origem.x += vet.x;
                objeto->origem.y += vet.y;
                objeto->origem.z += vet.z;
                break;

                :
                :
            }
        }
    }
}

```

```

/*-----*/
/* Exemplo da implementação de um tipo animado CAMERA */
/*-----*/
typedef
{
    int tipo;
    .
    .
} CAMERA_TP;

fcamera(camera,mensagem,par1,par2,...)

    OBJ_TP *camera;
    int mensagem;
    {
        if (mensagem == CRIAR)
        {
            camera = malloc(sizeof(CAMERA_TP));
        }
        else {
            switch(mensagem)
            {
                case QUADRO:
                    :
                    :
                    break;

                case FIM:
                    :
                    :
                    break;

                :
                :
            }
        }
    }
}

```


Capítulo 7

CONCLUSÃO

7.1 NOTAS SOBRE O TRABALHO

Procurou-se, neste trabalho, apresentar uma visão geral do tema Animação por Computador, analisando algumas linguagens conhecidas e dando uma breve introdução aos algoritmos de animação. O módulo de animação apresentado procura condensar as características mais importantes numa pequena biblioteca de funções.

Neste trabalho, a implementação foi baseada na linguagem C. Entretanto, a melhor alternativa seria implementá-lo em uma linguagem orientada a objeto. Existem pelo menos duas extensões do C com estas características, o Objective-C e o C++. Nestas linguagens, a implementação do tipo vetor com todas as operações associadas, por exemplo, poderia ser feita com extrema facilidade, sem necessidade de pré-compilação.

7.2 O FUTURO DA ANIMAÇÃO

Apesar da animação por computador ter ser tornado uma grande moda, e ser vista por milhões de pessoas em propagandas de TV e em filmes de ficção científica, pesquisadores e usuários são quase unânimes em afirmar que a animação continua sendo uma tarefa árdua.

E o desafio que está proposto é o de encontrar caminhos que tornem a animação mais natural e mais fácil de utilizar. Como foi visto, muitas alternativas estão sendo propostas: desde a melhoria em algoritmos de interpolação, até métodos de robótica e inteligência artificial.

Este é um assunto polêmico, e isso ficou evidente no recente debate da conferência do Siggraph'87, aonde, no debate intitulado "Simulação Física e Representação Visual de Fenômenos Naturais", dois conceituados profissionais, Alan Barr e William Reeves defenderam posições opostas [SIGGRA87] [FRENKE88].

Alan Barr, professor do Instituto de Tecnologia da Califórnia, critica o sistema atual de animação que, segundo ele, segue a "cinemática do movimento":

1. Um objeto em repouso permanece em repouso ignorando qualquer força que possa atuar sobre este.

2. Um objeto move somente pela influência de uma spline de interpolação.
3. Para qualquer ação, não há nenhuma reação.

Estas "leis", segundo ele, acabam criando um movimento irreal e inconvincente. Barr conclui que os objetos devem seguir as leis de Newton para se movimentarem de maneira realística, levando em consideração dados como o material do objeto, forças aplicadas sobre ele, o meio em que o objeto se encontra, etc.

William Reeves, um especialista em efeitos especiais da Pixar, discorda de Barr, afirmando que a simulação de fenômenos físicos geralmente é tão complexa que nenhum computador é capaz de calcular em tempo hábil. Assim, a simulação física só deve ser usada quando for possível. Reeves termina mostrando que, mesmo sem nenhuma base "física", os efeitos podem parecer realísticos. São ilusões, ele diz, e não há porque ter vergonha disso.

Quem está certo, é difícil dizer. De qualquer forma os belos resultados obtidos por Reeves (Star Treck II) e por Barr [BARR87], provam que nenhum deles está totalmente destituído de razão.

O futuro talvez esteja na combinação de todas as técnicas aqui vistas. E de outras que ainda estão por surgir.

BIBLIOGRAFIA (*)

- [AGHA86] Agha, G., "An Overview of Actor Languages", Sigplan Notices, vol. 21 no. 10, outubro de 1986.
- [ANTICS84] Antics, Do-it Yerself Animation Machine, UNIVAC, folheto publicitário, maio de 1983.
- [ARYA84] Arya, K. "A Functional Approach to Picture Manipulation", Computer Graphics Forum 3, pp. 35-46, 1984.
- [ARYA86] Arya, K. "A Functional Approach to Animation", Computer Graphics Forum 6, pp. 2975-312, 1986.
- [COMO74] Como Funciona - Enciclopédia da Ciência e Técnica, Abril S.A. Cultural e Industrial, 1974
- [COX86] Cox, B., Object-Oriented Programming - An Evolutionary Approach, Addison-Wesley Publishing Company, 1986.
- [COXHUN86] Cox, B. e B. Hunt, "Objects, Icons and Software-ICS", Byte, pp. 161-176, agosto de 1986.
- [DONOLS81] O'Donnel, T. J. e A. J. Olson, "GRAMPS - A Graphics Language for Real Time, Interactive, Three-Dimensional Picture Editing and Animation", Computer Graphics Proc. Siggraph'81, vol.15, no. 3, pp. 133-134, agosto de 1981.
- [FOURNIE86] Fournier, A. e W. T. Reeves, "A Simple Model of Ocean Waves", Computer Graphics Proc. Siggraph'86, vol. 20, no. 4, agosto de 1986.
- [FRENKE88] Frenkel, K. A., "The Art and Science of Visualizing Data", Communications of the ACM, vol. 31, no. 2, pp. 111-121, fevereiro de 1988.
- [GORAUD71]* Goraud, H., "Continuous Shading of Curved Surfaces", IEEE Transactions on Computer, vol. 20, no. 6, pp. 47-57, outubro de 1985.
- [HALAS68] Halas, J. e R. Manvel, The Technique of Film Animation, The Focal Press, 1968.

(*) A bibliografia inclui os livros e artigos notoriamente relevantes, que foram citados pelas fontes pesquisadas, aos quais a autora não teve acesso. Estas fontes estão marcadas com um asterisco.

- [HEWITT77]* Hewitt, C., e Atkinson, R., "Parallelism and Synchronization in Actor System", ACM Symposium on Principles of Programming Languages 4, Los Angeles, janeiro de 1977.
- [KAEPAT86] Kaehler, T. e D. Patterson, "A Small Taste of Smalltalk", Byte, pp. 145-159, agosto de 1986.
- [LASSET87] Lasseter, J., "Principles of Tradicional Animation Aplied to 3D Computer Animation", Computer Graphics Proc. Siggraph'87, vol.21. no. 4, pp. 35-44, julho de 1987.
- [LISZIL74]* Liskov, B. e S. Zilles, "Programming with Abstract Data Types", ACM Sigplan Notices, vol.9 no. 4, pp. 50-59, abril de 1974.
- [MALLGR82] Mallgren, W., Formal Specification of Interactive Graphics Programming Languages, The MIT Press, 1982.
- [MANDEL82] Mandelbrot, B. B., The Fractal Geometry of Nature, W. H. Freeman and Company, 1982.
- [MICROS87] Microsoft C for the MS-DOS Operating System - Language Reference, Microsof Corporation, 1987.
- [PAPERT70]* Papert, S., "Teaching Children Thinking", Proc. IFIP World Conference On Computer Education, pp. I/73-I/78.
- [PEACHE86] Peachey, D. R., "Modelling Waves and Surf", Computer Graphics Proc. Siggraph'86, vol. 20, no. 4, agosto de 1986.
- [PHONG75]* Phong, B. T., "Illumination for Computer Generated Pictures", Communications of the ACM, vol. 28, no. 6, pp. 311-317, junho de 1975.
- [REEVES81] Reeves, W. T., "Inbetweening for Computer Animation Utilizing Moving Points Constraints", Computer Graphics Proc. Siggraph'81, vol. 15, no. 3, agosto de 1981.
- [REEVES83] Reeves, W. T., "Particles - A Technique for Modeling a Class of Fuzzy Objects", Computer Graphics Proc. Siggraph'83, vol. 17, no. 3, julho de 1983.
- [REYNOL75]* Reynolds, C. W., A Multiprocesing Approach to Computer Animation, MIT EECS Department, 1975.
- [REYNOL82] Reynolds, C. W., "Computer Animation With Scripts and Actors", Computer Graphics Proc. Siggraph'82, vol. 16, no. 3, julho de 1982.

- [REYNOL87] Reynolds, C. W., "Flocks, Herds, and Schools, A Distributed Behavioral Model", Computer Graphics Proc. Siggraph'87, vol. 21, no. 4, pp. 25-34, julho de 1987.
- [SETUBA87] Setubal, J. C., FIG - Uma Linguagem para Especificação de Figuras, tese de mestrado, Unicamp, agosto de 1987.
- [SHEGRE82] Shelley, K. L., D. P. Greenberg, "Path Specification and Path Coherence", Computer Graphics Proc. Siggraph'82, vol. 16, no. 3, julho de 1982.
- [SIGGRA87] Computer Graphics Proc. Siggraph'88, vol. 21, no. 4, pp. 335-336, julho de 1987.
- [STEBAD85] Steketee, S. N., N. I. Badler, "Parametric Keyframe Adjustment and Phrasing Control", Computer Graphics Proc. Siggraph'85, vol. 19, no. 3, julho de 1985.
- [SMALLT86] Smalltalk/V - Tutorial and Programming Handbook, Digital Inc., 1986.
- [SMITH84] Smith, A. R., "Plants, Fractals and Formal Languages", Computer Graphics Proc. Siggraph'87, vol.21. no. 4, pp. 1-10, julho de 1987.
- [STERN79] Stern, G., "SoftCel . An Apllication of Raster Scan Graphics to Conventional Cel Animation", Computer Graphics Proc. Siggraph'79, vol. 13, no. 2, agosto de 1979.
- [SUTHER74] Sutherland, I. E, R. F. Sproull e R. A. Schumaker, "A Characterization of Ten Hidden-Surface Algorithms", Computing Surveys, vol. 6, n. 1, pp. 1-55, 1974.
- [TAKAHA88] Takahashi, T., Introdução a Programação Orientada a Objeto, EBAI, 1988.
- [THALMA85] Magnenat-Thalmann, N. e D. Thalmann, Computer Animation, Theory and Practice, Springer-Verlag, 1985.
- [THALMA85b] Magnenat-Thalmann, N. e D. Thalmann, "An Indexed Bibliography on Computer Animation", IEEE Computer Graphics & Applications, pp. 76-86, 1985.

- [THALMA85c] Magnenat-Thalmann, N. e D. Thalmann, "Three-Dimensional Computer Animation: More an Evolution Than a Motion Problem", IEEE Computer Graphics & Applications, pp. 47-57, outubro de 1985.
- [THALMA85d] Magnenat-Thalmann, N., D. Thalmann e M. Fortin, "Miranim: An Extensible Director-Oriented System for the Animation of Realistic Images", IEEE Computer Graphics & Applications, vol 5. no. 3, pp. 47-57, março de 1985.
- [TORI87] Tori, R., R. Arakaki, A. M. A. Massola, L. V. L. Filgueiras, Fundamentos de Computação Gráfica - Compugrafia, Livros Técnicos e Científicos Editora S. A., 1987.
- [WALLAC81] Wallace, B., "Merging and Transformation of Raster Images for Cartoon Animation", Computer Graphics Proc. Siggraph'81, vol. 15, no. 3, agosto de 1981.
- [WILHEL87] Wilhelms, J., "Toward Automatic Motion Control", IEEE Computer Graphics & Applications, pp. 11-22, abril de 1987.
- [WHITED80] Whited, T., "An Improved Illumination Model for Shaded Display", Communications of the ACM, vol. 23, no. 6, pp. 343 -349, junho de 1980.
- [ZELTZE85] Zeltzer, D., "Towards an Integrated View of 3-D Computer Animation", Computer-Generated Images, The State of the Art, Proceedings of Graphics Interface'85, Springer-Verlag, 1985.