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

UM AMBIENTE PARA A INTEGRAÇÃO DE SISTEMAS DE ANIMAÇÃO

por

Rodrigo de Losina Silva

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

prof. Flávio R. Wagner
Orientador

prof. Carla Freitas
Co-orientadora

Porto Alegre, outubro de 1995.

SUMÁRIO

LISTA DE FIGURAS

RESUMO

Este trabalho apresenta um modelo de integração de ferramentas de animação que reduz consideravelmente o esforço envolvido na construção de novos sistemas. O modelo proposto distribui as tarefas a serem implementadas pelo sistema entre diversas ferramentas, permitindo que cada uma seja menor e mais fácil de manter que um sistema completo.

Uma implementação do modelo proposto também é apresentada aqui, assim como o é um protótipo de sistema, construído de acordo com o modelo. O protótipo é comparado com as ferramentas de animação atualmente disponíveis na UFRGS, a universidade na qual este estudo foi desenvolvido.

O trabalho conclui com uma análise dos resultados principais obtidos deste estudo. Ao final também são apresentadas algumas sugestões de trabalhos futuros.

Palavras-chave: animação, ambiente, integração, sistemas distribuídos

ABSTRACT

This work presents a model for animation tool's integration, which greatly reduces the programmer's work in developing a new system. The proposed model distributes all tasks among several tools, making each tool smaller and easier to maintain.

An implementation of such model is also described here, and so is a prototype of an animation system made according to the model proposed. The prototype is then compared to the current animation tools available at the UFRGS, the University in which this work was developed.

The work concludes with an analysis of the main results and some suggestions of possible future works.

Keywords: animation, framework, integration, distributed systems

1 Introdução

Neste capítulo são apresentados os conceitos fundamentais de animação modelada por computador. Após, é descrito um estudo-de-caso da criação de animações pelo *grupo de computação gráfica* (CG) da UFRGS. Os objetivos deste trabalho são enumerados ao final, embasados nas conclusões do estudo feito.

O segundo capítulo apresenta o ambiente de integração de ferramentas proposto neste trabalho, descrevendo estruturas de dados, fluxo de mensagens e mecanismos de comunicação.

O capítulo três apresenta as principais alternativas de integração de ferramentas presentes na literatura, e o tipo de problemas que elas se propõem a resolver. Após, são discutidas as particularidades da integração de ferramentas de animação, justificando as decisões de projeto no desenvolvimento do ambiente.

No capítulo quatro são discutidas questões associadas especificamente a ferramentas que podem ser integradas ao modelo e as características que as mesmas precisam ter.

O capítulo cinco descreve uma implementação do modelo, enquanto o sexto capítulo faz uma validação de tal implementação através da comparação entre um protótipo de sistema construído com o ambiente proposto e o conjunto de ferramentas utilizado atualmente pelo grupo de computação gráfica da UFRGS.

O capítulo sete apresenta as principais conclusões do presente trabalho e propõe uma série de possíveis ampliações do modelo como trabalho futuro.

1.2 Animação Modelada por Computador

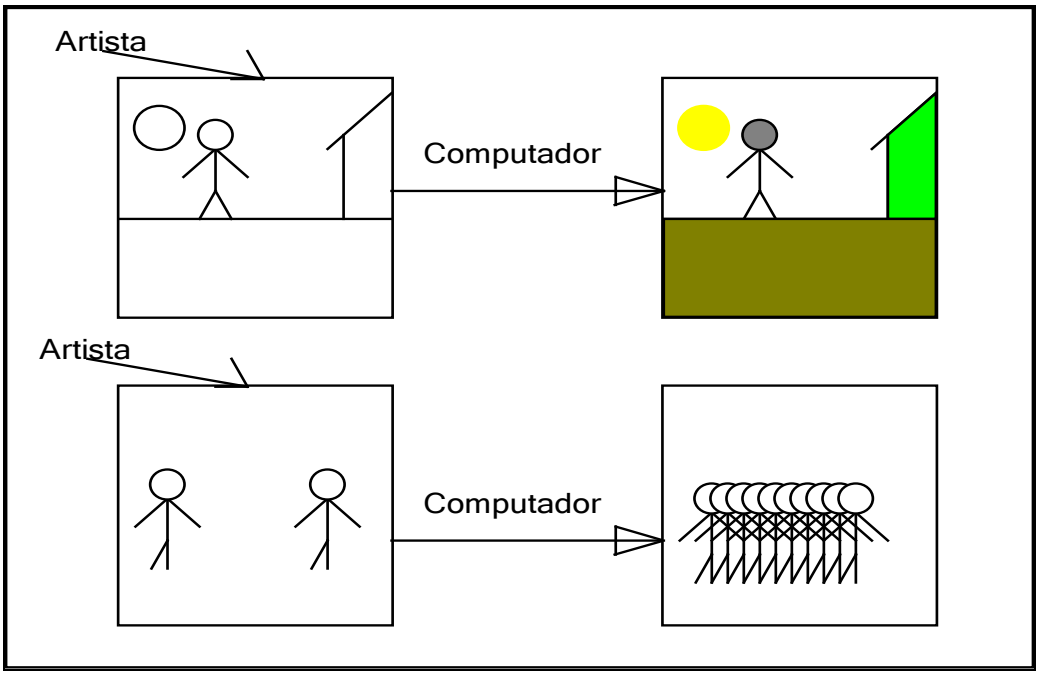

fig 1.1 - Animação assistida por computador

Existem duas técnicas principais de animação com a participação do computador: *animação assistida por computador* e *animação modelada por computador*.

Na *animação assistida por computador*, este é utilizado como auxílio em uma ou mais etapas do processo de construção das animações, por exemplo na colorização das imagens ou na geração de cenas intermediárias, conforme exemplificado na figura 1.1. A *animação assistida por computador* não será tratada neste capítulo por não ser relevante para o presente trabalho.

Na *animação modelada*, o computador mantém armazenadas a forma e a posição dos objetos e, através de cálculos que reproduzem movimentos, gera a animação [MAG 85a]. Nesta alternativa o artista não é responsável por desenhar cada imagem, mas sim por especificar a forma dos elementos da cena e o movimento que eles executam.

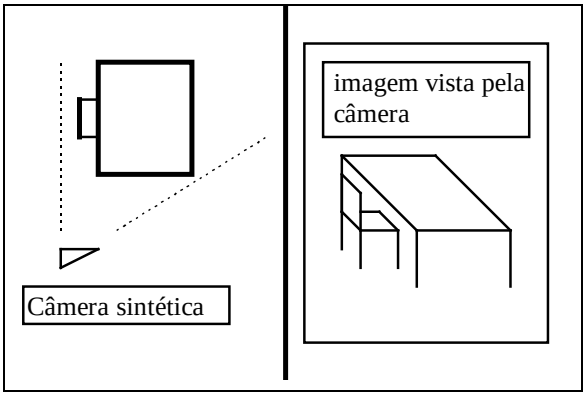

fig 1.2 - Mapeamento de câmera sintética

Em geral, na *animação modelada*, a especificação e geração de movimentos é definida num espaço de três dimensões, sendo então mapeada para o espaço bidimensional, resultando numa imagem de um instante da animação. Tal efeito é obtido definindo-se uma câmera sintética, a qual simula o processo efetuado pelas câmeras tradicionais. A figura 1.2 contém uma imagem que ilustra este mapeamento.

Para produzir a animação, são geradas seqüências de imagens, chamadas de quadros, durante o transcorrer da cena. Estes quadros são exibidos à razão de 24 ou 30 por segundo, resultando na ilusão de um movimento contínuo.

1.2.1 Elementos de uma Cena de Animação Modelada

O elemento principal de uma animação é o ator. Um ator é um elemento visível da cena, geralmente representando um objeto que possui algumas características que variam ao longo do tempo, como forma e posição. Tais características serão denominadas genericamente, neste trabalho, de parâmetros.

Além dos atores, uma cena é constituída também por uma ou mais câmeras sintéticas, que possuem parâmetros como posição, tipo de lente e orientação. Outras entidades, por exemplo, luzes, poderão fazer parte da cena dependendo do grau de sofisticação do sistema que modela a animação.

Atores, câmeras e demais entidades que pertençam a uma cena serão denominados, ao longo deste trabalho, elementos da cena.

1.2.2 Principais Passos da Animação Modelada

Na literatura especializada em animação modelada, por exemplo [MAG 85b] e [BRU 90], o processo de animação por computador é dividido em três etapas principais: a modelagem dos objetos; a geração dos movimentos; a síntese das imagens.

1.2.2.1 Modelagem de Objetos

A modelagem dos objetos é a etapa de construção de um modelo matemático da forma dos objetos presentes na animação. Este modelo pode ser construído através de dados do mundo real (fotos de objetos, dados tomográficos, etc.), ou pode ser gerado diretamente pelo artista, com o auxílio do computador. Em [FOL 90] podem ser encontrados inúmeros métodos de geração de modelos da forma dos objetos.

1.2.2.2 Geração de Movimentos

A etapa de geração de movimentos é aquela na qual os elementos presentes na cena sofrem alguma alteração que resulta na geração de imagens diferentes ao longo do tempo. Geralmente, a animação resulta do movimento translacional, ou seja da mudança de posição, de um ou mais objetos. Entretanto, muitas outras mudanças na cena podem afetar a animação, como alterações de luminosidade e movimentos da câmera sintética.

Duas questões estão associadas a esta etapa: os parâmetros que o animador pode manipular para controlar o movimento e a interface que ele dispõe para especificar estes parâmetros.

Os parâmetros que o animador controla estão associados à forma como os movimentos e transformações são gerados. Sistemas baseados em cinemática oferecem ao animador o controle da velocidade e aceleração dos elementos presentes na cena. Sistemas de análise dinâmica permitem o controle de forças e torques. Outros algoritmos, por exemplo técnicas de animação facial, movimento de partículas e animação comportamental, fornecem ainda outros parâmetros de controle.

A interface que o animador dispõe para controlar tais parâmetros é igualmente importante. Na literatura, [BRU 90] e [STU 90], são apresentadas duas alternativas principais: o controle por roteiro e o controle interativo. A alternativa mais simples, controle por roteiro, possibilita que os comandos de controle dos parâmetros sejam escritos em um arquivo, que é interpretado ou compilado para gerar a animação.

Tal arquivo pode tanto estar numa linguagem de programação de propósitos gerais, como C ou Pascal, sendo transformado em código executável por um compilador, como pode estar numa linguagem específica de descrição de roteiros, que é lida e interpretada pelo próprio sistema. O ambiente que será apresentado nos capítulos que seguem a este trata de ambas as alternativas.

O controle interativo faz uso de um sistema orientado a artistas. Tais sistemas permitem que o usuário selecione diretamente na tela do computador os parâmetros que ele deseja que sejam alterados, recebendo interativamente uma resposta para cada ação que faz. O capítulo 7, na seção de trabalhos futuros, apresenta uma discussão sobre formas de implementar recursos de edição direta dentro da filosofia de funcionamento do ambiente proposto.

1.2.2.3 Síntese das Imagens

Conforme mencionado, a animação tridimensional envolve uma etapa de transformação dos elementos visíveis da cena, do espaço 3D para o espaço 2D. A mera transformação de dimensão, entretanto, não é suficiente para produzir um resultado realístico. Faz-se necessário tratar uma série de aspectos que influenciam a qualidade da imagem final, como o modelo de iluminação usado, a cor e textura dos objetos, entre outros.

Diferentes algoritmos levam em consideração diferentes propriedades dos atores e do modelo de iluminação usado, e, eventualmente, permitem a definição de outras características que podem influenciar a imagem resultante, como simulação do uso de filtros na câmera sintética e simulação de efeitos atmosféricos.

Cabe salientar que, uma vez que os parâmetros mencionados influenciam a imagem resultante, alterações neles também podem produzir uma animação, e portanto eles não podem ser ignorados na fase de geração de movimentos.

1.3 Construção de Animações - Estudo de Caso

O objetivo de fazer um estudo da geração de animações numa situação real é, primordialmente, definir claramente quais os problemas enfrentados pelo uso de ferramentas de animação independentes, e de que forma a construção de um ambiente para a integração poderia solucionar ou minimizar estes problemas.

Este estudo se baseia no desenvolvimento, pelo grupo de computação gráfica da UFRGS, de algumas vinhetas utilizando técnicas de animação modelada por computador, e enfoca o fluxo de informação entre as ferramentas nas fases de geração de movimentos e síntese de imagens. As animações consideradas foram M.A.T.E.[OLA 92], Vinheta da SCT e CG com Filtro.

O trabalho desenvolvido pelo grupo de Computação Gráfica da UFRGS foi escolhido principalmente por ter envolvido o uso de diversas ferramentas diferentes na construção das animações. Na verdade, as dificuldades enfrentadas pela ausência de um ambiente integrado estão entre as motivações do presente trabalho.

1.3.1 Classes de Ferramentas

As ferramentas utilizadas pelo Grupo de Computação Gráfica da UFRGS podem ser divididas em alguns grupos principais, descritos a seguir:

- **Ferramentas de Modelagem da Forma de Objetos**

Foram utilizadas algumas ferramentas desenvolvidas na UFRGS para modelar a forma dos atores presentes nas animações, principalmente na animação M.A.T.E.[OLA 92], sendo exemplos representativos desta classe de ferramentas o SIHMOS e o Bezier4D. Uma vez que o presente trabalho não trata do aspecto de modelagem dos objetos, não foi feito um estudo aprofundado deste grupo de ferramentas.

- **Ferramentas de Geração de Movimentos**

Estas ferramentas, em sua maioria protótipos desenvolvidos como dissertações de mestrado, foram utilizadas para modelar os movimentos mais complexos dos atores. Todas elas seguem um mesmo procedimento de geração de movimentos, facilitando a integração.

Cada ferramenta lê um arquivo de entrada, que determina como um ator irá se movimentar, e calcula seu movimento, gerando ao final um arquivo com sua posição em cada quadro da animação.

As principais ferramentas que foram utilizadas na geração das animações analisadas implementam diferentes técnicas de geração de movimentos, e são listadas a seguir:

1. Sistema de dinâmica direta: a ferramenta CLIC [LEM 92] permite o controle do movimento de atores através da especificação, no seu arquivo de entrada, das forças e torques atuando sobre o ator.

2. Sistema de corpos flexíveis: a ferramenta FLEX-3D, [NED 92] e [NED 93], gera o movimento de corpos bidimensionais flexíveis, sofrendo a ação de forças externas. O sistema gera movimentos semelhantes aos de panos.
3. Sistema de corpos articulados: esta ferramenta [MUS 92] controla atores constituídos de segmentos sólidos unidos por articulações. O movimento dos atores é calculado em função de dinâmica direta.

- **Ferramentas de Integração de Atores**

As ferramentas de integração recebem como entrada um roteiro que descreve toda a cena e identifica os arquivos associados a cada ator. Estas ferramentas são responsáveis por integrar todos os arquivos previamente gerados, criando uma animação composta por todos os atores manipulados pelas demais ferramentas.

A animação gerada é exibida em aramado (*wire frame*) e armazenada numa seqüência de arquivos, que são repassados a sistemas de síntese de imagens. A exibição em aramado permite visualizar a animação, corrigindo eventuais erros no movimento dos atores.

No desenvolvimento das animações estudadas foi utilizada inicialmente a ferramenta PREVIEW [SCH 92a], que executa em estações PROCEDA. Com a disponibilização de estações de trabalho SUN, foi desenvolvida a ferramenta ANIMAKER [SIL 92], visando executar a mesma tarefa em tal plataforma.

- **Ferramentas de Síntese de Imagens**

As ferramentas de síntese de imagens utilizadas geram uma imagem foto-realística a partir de um arquivo que descreve uma determinada cena. As ferramentas PREVIEW e ANIMAKER geram um arquivo de cena para cada quadro gerado da animação, bastando, portanto, fazer o sistema de síntese de imagens ler cada arquivo para gerar todos os quadros da animação.

Cada uma das animações estudadas fez uso de um sistema diferente de síntese de imagens. Tais sistemas são descritos a seguir:

1. O sistema MacShadow [NAS 92] é um sistema de exibição de imagens O formato de descrição de cenas é compatível com o formato gerado pelo sistema PREVIEW.

2. O segundo sistema de síntese de imagens, desenvolvido como trabalho de graduação [SCH 92b], utiliza a técnica de *ray-tracing*. Este sistema foi modificado para aceitar o formato de saída gerado pelo sistema ANIMAKER.
3. O terceiro sistema de síntese de imagens, *POV-RAY*, é um sistema de domínio público. Para sua utilização, foi criada uma ferramenta de conversão do formato gerado pelo ANIMAKER para o formato de arquivo do sistema.

1.3.2 Organização das Ferramentas

As figuras a seguir apresentam a forma como as principais ferramentas utilizadas eram integradas nas animações estudas. A figura 1.3 apresenta um diagrama associado a geração da vinheta M.A.T.E.. Um diagrama mais completo pode ser visto em [OLA 92].

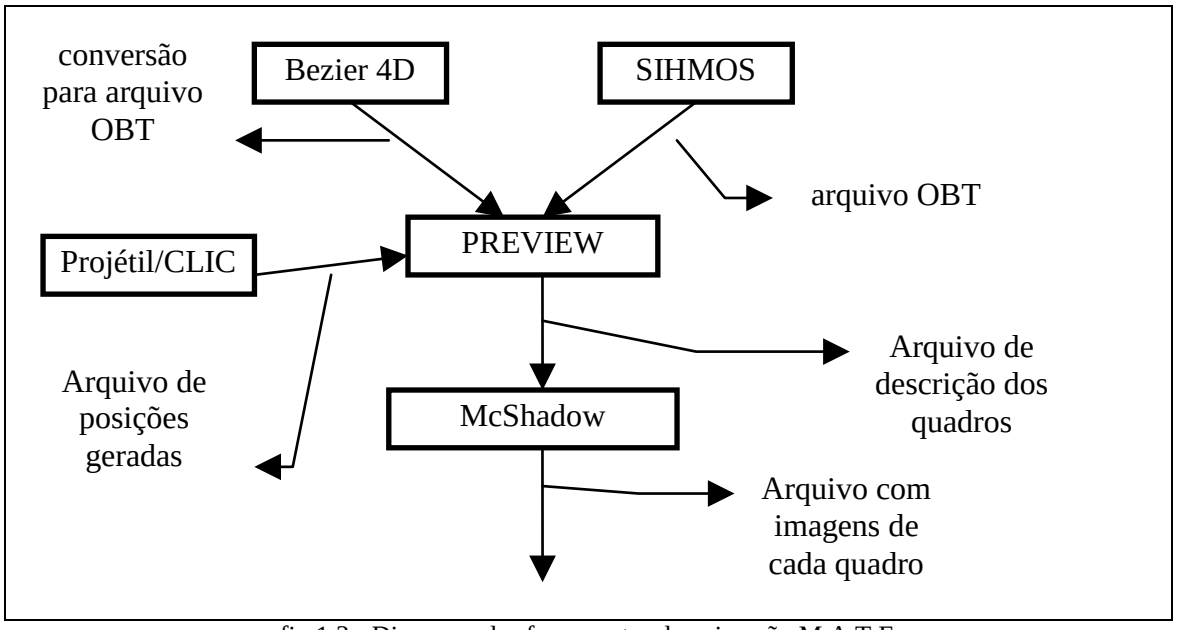

fig 1.3 - Diagrama das ferramentas da animação M.A.T.E.

A figura 1.4, a seguir, apresenta as principais ferramentas utilizadas na animação *CG com Filtro*. A *Vinheta da SCT* fez uso, essencialmente, do sistema ANIMAKER e de uma ferramentas de movimento de partículas construída especialmente para a animação.

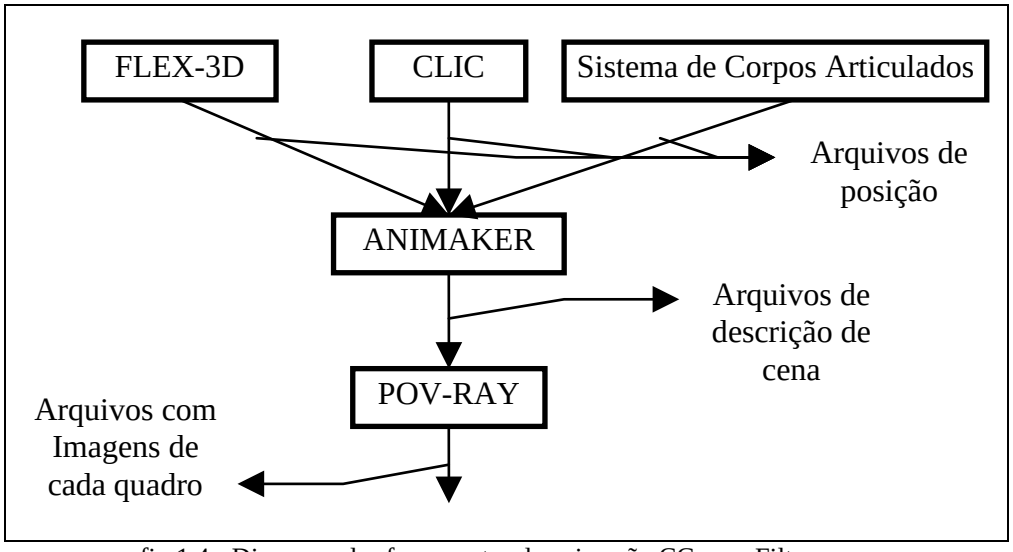

fig 1.4 - Diagrama das ferramentas da animação CG com Filtro

1.3.2 Análise da Geração de Movimentos

Nas três animações estudadas o movimento de cada ator foi gerado de forma isolada dos demais, utilizando as ferramentas de geração de movimentos citadas anteriormente. A estratégia adotada para integração das ferramentas tem algumas desvantagens, sendo as principais descritas a seguir:

Cada ferramenta implementa sua leitura de arquivos de forma completamente independente das demais ferramentas, utilizando um formato próprio de roteiro. Conseqüentemente, o uso de cada uma das ferramenta exigia um novo período de aprendizado.

Uma segunda desvantagem da independência entre ferramentas é que não há possibilidade de especificar uma interação entre os atores nos roteiros. Para que dois atores façam algum movimento em conjunto, os cálculos de posição precisam ser feitos manualmente, já que os movimentos são gerados de forma independente.

1.3.3 Análise da Integração dos Atores

Após a geração dos movimentos de cada ator, é necessário integrá-los num mesmo cenário, fazendo uso das ferramentas de integração mencionadas

Esta forma de integrar o resultado dos sistemas de geração de movimentos possui uma deficiência significativa. Faz-se necessária a intervenção constante de um usuário para viabilizar a comunicação entre as ferramentas, copiando arquivos e alterando os roteiros de entrada de PREVIEW e ANIMAKER. Este custo é considerável, principalmente nas fases de depuração, onde constantemente é necessário alterar o movimento de determinado ator e, conseqüentemente, transportar arquivos entre ferramentas.

Recentemente foram desenvolvidos alguns recursos visando automatizar este processo, chamando automaticamente as ferramentas necessárias para executar todos os procedimentos necessários a geração de uma determinada animação. A intervenção humana, no entanto, continua sendo indispensável quando se deseja gerar novamente uma cena, ou o movimente de um determinado ator.

1.3.4 Análise da Síntese de Imagens

Terminado um ciclo de geração e integração de movimentos, o resultado obtido é um conjunto de arquivos descrevendo cada quadro gerado da animação. Estes arquivos são então usados por sistemas de síntese de imagens para gerar as imagens definitivas que, exibidas em seqüência, produzirão a animação desejada.

A utilização de arquivos para transmitir as informações sobre a cena permitiu o uso de diferentes ferramentas de síntese de imagens, incluindo uma ferramenta de domínio público. Esta abordagem, porém, também tem algumas desvantagens, derivadas da falta de integração de tais ferramentas com os demais sistemas utilizados.

A principal dificuldade observada foi ainda na fase de construção das animações. Para observar a imagem de um único quadro da cena, fazia-se necessário gerar um arquivo descrevendo a cena, transmití-lo para o sistema de síntese de imagens utilizado, executá-lo

e, após a geração da cena, exibí-la. Num ambiente integrado, existem inúmeras formas de otimizar este processo, possibilitando, inclusive, que o animador escolha qual o sistema de síntese de imagens que ele deseja utilizar para ver determinado quadro. Pode ser interessante, por exemplo, utilizar um sistema mais simples e rápido para os ajustes principais, gerando cenas com maior realismo apenas quando realmente necessário.

1.3.5 Conclusões Gerais do Estudo

O grupo de computação gráfica da UFRGS conseguiu gerar diversas animações utilizando de forma integrada uma série de ferramentas diferentes. Deve ser lembrado, ainda, que muitas destas ferramentas foram construídas de forma isolada e independente, limitando a integração apenas a arquivos e exigindo, em algumas etapas, o uso de conversores de formato.

Naturalmente, tais restrições impossibilitaram a construção de um ambiente fortemente integrado, no qual o usuário pudesse mais facilmente utilizar as diferentes ferramentas disponíveis e ignorar a forma como as informações circulam entre elas.

As restrições descritas neste estudo servem de base para definirmos os objetivos que um ambiente de integração deva atingir para permitir que diferentes ferramentas de animação possam ser utilizadas em conjunto.

1.4 Objetivos do Trabalho

O presente trabalho propõe um ambiente que permite que diferentes ferramentas cooperem entre si, trabalhando em conjunto para a construção de animações. Nas fases iniciais da definição deste trabalho, foram traçados vários objetivos que se pretendia atingir, os quais são apresentados nas seções seguintes.

1.4.1 Otimização de Recursos

Independente das técnicas utilizadas, ou do enfoque dado, existe uma série de tarefas comuns a praticamente qualquer sistema de animação tridimensional, como, por

exemplo, a interação com o usuário e a síntese de imagens. Como resultado, existem inúmeros sistemas diferentes implementando algoritmos semelhantes.

Seria muito mais eficiente se cada novo sistema de animação não precisasse tratar tarefas que outros sistemas já tratam de forma eficiente. É claramente um desperdício que um programa de animação tenha que implementar recursos de exibição de imagens, mesmo quando existe uma disponibilidade de ferramentas que executam tal tarefa de forma aceitável.

Uma das alternativas mais comuns é a utilização de ferramentas de síntese de imagens que recebem arquivos como entrada. Nesta alternativa, outros sistemas precisam apenas gerar arquivos segundo a sintaxe da ferramenta. Conforme citado na seção 1.3.3, esta não é uma forma eficiente quando o animador deseja observar apenas uma cena que ele terminou de gerar. Além disso, esta alternativa não se aplica a outras ferramentas que tenham uma maior interação com os demais sistemas, como, por exemplo, ferramentas de interpretação de roteiro.

O presente trabalho busca uma solução alternativa para resolver este problema, através da definição de um ambiente que permite que as ferramentas cooperem entre si, reduzindo o custo da implementação de cada uma delas, já que cada ferramenta poderá tratar apenas de algumas tarefas determinadas, repassando para as demais qualquer tarefa que estas sejam capazes de atender.

Assim, um dos objetivos deste trabalho é definir um protocolo através do qual ferramentas de animação possam trabalhar de uma forma cooperativa, possibilitando que uma animação seja gerada por um conjunto de ferramentas, sem que nenhuma precise ser responsável por todo o processo.

1.4.2 Interação entre Ferramentas

A simples integração de diferentes ferramentas num ambiente torna possível reduzir a complexidade de cada ferramenta, que passa a se dedicar a uma tarefa específica. Entretanto, teremos como resultado que diferentes ferramentas, utilizando diferentes

algoritmos de geração de movimentos, poderão controlar atores que compartilham de um mesmo cenário.

Um objetivo deste trabalho é permitir que tais ferramentas troquem informações sobre a cena, possibilitando que os atores que cada uma controla possam interagir entre si. Com isto, procura-se permitir que atores possam agir em função de toda a cena. Isto torna possível, por exemplo, que uma ferramenta de dinâmica detecte a colisão de um ator que ela controla com outros objetos que eventualmente não são controlados pela ferramenta, mas sim por outros sistemas do ambiente.

1.4.3 Interface Única para o Usuário

É também um objetivo deste trabalho oferecer aos usuários uma interface única de acesso aos recursos do ambiente. Em outras palavras, objetiva-se que o usuário não tenha que ter conhecimento de cada ferramenta que faça parte do ambiente.

O ambiente deve, portanto, permitir que uma ferramenta possa ser responsável por toda a comunicação com o usuário, fornecendo a este acesso a todos os recursos presentes no ambiente.

A razão por trás deste objetivo é relativamente óbvia para quem já lidou com inúmeras ferramentas. O custo de aprender a utilizar cada ferramenta do ambiente, não sendo adotada esta política, crescerá proporcionalmente ao número de ferramentas presentes no ambiente, até atingir o ponto em que seria virtualmente inviável utilizar o ambiente ao máximo de suas potencialidades.

2. O Ambiente de Integração

Neste capítulo apresentamos o ambiente de integração de ferramentas proposto neste trabalho. O ambiente foi construído visando atender os objetivos propostos no capítulo 1.

2.1 Funcionamento Geral do Ambiente

A rigor, o ambiente em si não produz nenhum resultado para um usuário final, ou seja, ele não é um sistema de animação, mas sim um ambiente que facilita a construção de tais sistemas. O ambiente fornece os recursos para que ferramentas funcionem em conjunto, sendo, portanto, dirigido a desenvolvedores de sistemas, e não diretamente aos artistas que criam as animações.

Entretanto, não se trata de um sistema de integração de propósitos gerais, mas sim de um sistema voltado para a área de animação. O ambiente organiza o fluxo de informações entre processos e o tipo de informações transmitidas, definindo assim a forma como as ferramentas irão cooperar para gerar uma animação.

O ambiente opera de forma sincronizada, trocando mensagens com todas as ferramentas em cada instante de tempo que é gerado. Além disso, ele opera seqüencialmente, gerando o tempo *t* imediatamente após o tempo *t-1*. Esta restrição é necessária porque não há como garantir que uma ferramenta não irá gerar alguma mensagem no tempo *t-1* que afete a geração da cena no tempo *t*.

2.1.1 Fluxo de Execução do Sistema

Um sistema construído em cima do ambiente irá iniciar uma execução chamando uma ferramenta que especifique a animação a ser gerada. Esta ferramenta pode interagir diretamente com o usuário, pode ler um roteiro previamente construído ou pode ser ela própria o código da animação a ser gerada.

Em qualquer das hipóteses, a cada instante de tempo esta ferramenta se comunica com o ambiente, que envia mensagens para todas as outras ferramentas. As demais ferramentas serão responsáveis pelas outras tarefas da construção de uma animação, tais como a geração de movimentos e a síntese de imagens.

A geração de cada instante de tempo está dividida em três etapas, descritas a seguir.

2.1.1.1 Início do Instante de Tempo t

As ferramentas que geram novas informações sobre a cena se comunicam com o ambiente de animação nesta fase, transmitindo as informações descritas a seguir.

1. atores e elementos que entram e saem da cena naquele instante,
2. valor dos parâmetros de cada elemento,
3. ferramentas que controlam cada novo elemento,
4. outras ferramentas que passaram a estar presentes na cena (por exemplo, sistemas de síntese de imagens).

2.1.1.2 Geração do Instante de Tempo t

O ambiente executará os seguintes passos, após todas as ferramentas confirmarem que já enviaram suas mensagens:

1. ativação de ferramentas para controlar elementos que entram em cena,
2. desativação de ferramentas cujos elementos saem da cena,
3. retransmissão das informações sobre os elementos da cena para as ferramentas correspondentes.

As ferramentas do ambiente que são responsáveis pelos elementos executam seus algoritmos particulares, neste momento, gerando o instante *t*.

2.1.1.3 Troca de Informações sobre o Instante de Tempo t

Nesta fase, o instante de tempo *t* já foi gerado, porém a informação sobre a cena está distribuída entre diversas ferramentas. As ferramentas que precisam desta informação (por exemplo, sistemas de exibição de imagens) requisitam tais dados do ambiente.

O ambiente responde às requisições das ferramentas, repassando a requisição para a ferramenta adequada e retornando a resposta. Ao final, o ambiente inicia a construção do instante de tempo *t+1*, enviando uma mensagem de controle para todas as ferramentas e reiniciando o ciclo.

2.2 Estrutura Geral do Ambiente

A figura 2.1 apresenta uma visão geral da estrutura do ambiente proposto neste trabalho. Pode-se observar que existem essencialmente dois tipos de elementos: um *gerente* e várias ferramentas.

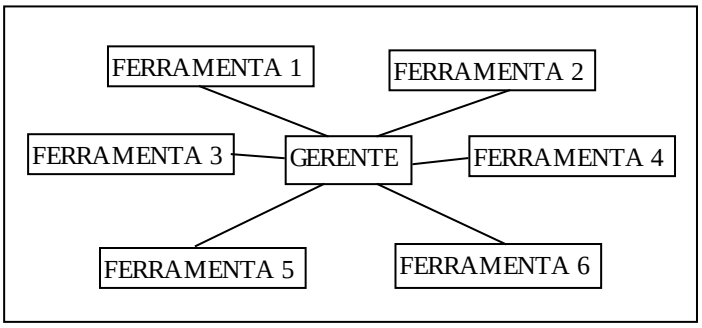

fig 2.1 - Estrutura Geral do Ambiente

Tanto o *gerente* quanto cada ferramenta são processos independentes, que se comunicam entre si conforme mostrado na figura, ou seja, cada processo se comunica apenas com o gerente.

Cada ferramenta é responsável por uma ou mais atividades, como interagir com o usuário, simular movimentos dinâmicos, sintetizar imagens, ou qualquer outra tarefa do processo de animação por computador. Todas estas ferramentas vão transmitindo suas informações para o gerente, que as repassa para outros processos conforme os mesmos vão requisitando.

Assim, um sistema que interage com o usuário enviará, a cada instante de tempo, comandos para o gerente especificando todos os parâmetros relevantes da cena naquele instante de tempo. Um sistema que precise de informações sobre a cena, como um exibidor de imagens, requisitará tais informações do gerente.

Para uma melhor compreensão do ambiente, os elementos que o compõem são melhor descritos nas seções seguintes.

2.2.1 O Gerente

A partir da figura 2.1, pode-se observar alguns aspectos relevantes do gerente, que merecem ser melhor descritos. O gerente, conforme mostrado na figura, é o único processo que se comunica com todos os demais, o que significa que todas as mensagens transmitidas entre processos passam por ele.

O gerente é quem mantém informações sobre o estado do sistema, mais especificamente quais processos estão ativos e quais objetos da cena estão associados aos mesmos. O gerente não armazena informações sobre o estado da cena em si, ou seja, ele não conhece a posição ou o movimento de um ator qualquer, mas ele mantém a informação de qual processo controla cada elemento da cena.

Assim sendo, os demais processos requisitam do gerente qualquer informação que exija um conhecimento sobre a cena, como, por exemplo, a posição de todos os atores ativos ou a força aplicada em determinado elemento. O gerente retransmite tais requisições para os processos responsáveis pelos atores e transmite a resposta de volta para o processo que fez a requisição. Da mesma forma, qualquer comando relacionado à ativação de processos, como a criação de novos atores, é tratado diretamente pelo gerente.

Desta forma, o *gerente* tem como principal atividade fornecer uma interface que permite que as ferramentas abstraiam questões relativas ao ambiente em si, como quais processos controlam que elementos da cena e quais processos precisam dos dados que a ferramenta produz.

É o *gerente*, também, que sincroniza todo o ambiente, garantindo que nenhum processo inicie a geração de um instante de tempo antes que todos os processos tenham terminado a geração do instante anterior.

Cabe salientar, também, que o *gerente* vê todos os demais processos da mesma forma. Ele não classifica processos segundo as tarefas assumidas pelos mesmos. Ou seja, para o gerente, um processo que exibe imagens e outro que controla o movimento de um ator são equivalentes.

2.2.2 As Ferramentas

O ambiente não define as tarefas que são de responsabilidade de cada ferramenta ligada ao gerente. Em princípio uma ferramenta pode executar qualquer tarefa, desde que as mensagens que ela troque com o ambiente sejam consistentes com o protocolo de comunicação do modelo.

A seção 3.3 entra em mais detalhes do protocolo de comunicação em si. Essencialmente, o protocolo define que a ferramenta deve responder a requisições sobre o valor de qualquer parâmetro dos elementos pelos quais é responsável. Não necessariamente todas as ferramentas são responsáveis por elementos da cena, porém o contrário é exigido pelo sistema: qualquer elemento da cena (um ator, uma luz, etc.) necessariamente está associado a uma ferramenta. É para esta ferramenta que o gerente eventualmente requisita informações sobre o elemento.

2.3 Protocolo de Comunicação

Conforme descrito na seção anterior, toda a comunicação no sistema é feita entre o gerente e os demais processos. A presente seção apresenta as informações que os elementos da cena trocam entre si, e o protocolo utilizado nesta comunicação.

2.3.1 Mensagens de Controle

Uma parte das informações trocadas entre o *gerente* e os processos se destinam a controlar o próprio funcionamento do ambiente como um todo. Estas mensagens sincronizam os processos durante a execução da animação.

No início de cada uma das três etapas da geração de um instante de tempo, descritas na seção 2.1.1, o *gerente* envia para todos os processos envolvidos naquela etapa uma mensagem de controle. Os processos ficam parados, após terem terminado o processamento da etapa anterior, esperando por esta mensagem de sincronismo, e só após recebê-la, iniciam o processamento da respectiva etapa.

De forma análoga, cada processo envia ao gerente uma mensagem de sincronismo após ter terminado determinada etapa. O *gerente* só inicia o processamento da etapa seguinte, enviando mensagens de sincronismo, após ter recebido uma mensagem de confirmação de todos os processos que iniciaram a etapa anterior.

2.3.2 Mensagens de Dados

As mensagens de dados contém informação sobre a cena, e podem ser divididas em três grupos: mensagens com informações sobre criação e fim de elementos, mensagens com informações sobre características de elementos da cena e mensagens de requisição de informação.

2.3.2.1 Mensagens de Criação e Destruição de Elementos

Estas mensagens são tipicamente originadas em um processo que é responsável pela definição da animação (um leitor de roteiros, um sistema que interage com um artista, etc.). Uma mensagem destas contém o nome do elemento sendo criado ou destruído. As mensagens de criação ainda informam a que ferramenta tal elemento está associado.

O *gerente* repassa a mensagem para o processo correspondente, ou seja, o processo ao qual o elemento em questão está associado. Se, no caso de uma mensagem de

criação, a ferramenta não está ativa naquele instante de tempo, ela é ativada pelo *gerente* antes da mensagem lhe ser repassada.

2.3.2.2 Mensagens com Informações de Elementos da Cena

Uma importante decisão de projeto está associada à definição da estrutura destas mensagens, que é a definição de que tipo de informações o ambiente permitirá que as ferramentas troquem entre si.

Uma alternativa é oferecer às ferramentas total liberdade para transmitir qualquer tipo de informação usando qualquer sintaxe. Neste caso, todas as mensagens com informações sobre a cena seriam, para o gerente, apenas vetores de *bytes*, que ele repassaria aos processos destinatários. Como conseqüência, cada ferramenta teria que utilizar uma sintaxe compatível com todas as ferramentas com as quais quisesse se comunicar.

Esta alternativa é por demasiado livre para o caso em questão. Eventualmente, se o ambiente proposto fosse de propósitos gerais, a alternativa poderia ser válida, porém, este ambiente destina-se especificamente à integração de ferramentas de animação. Assim, não há necessidade de oferecer às ferramentas uma liberdade maior do que a necessária para transmitir informações sobre a cena.

No outro extremo de possibilidades, há a alternativa de estruturar as mensagens entre ferramentas dentro de um esquema rígido, definindo-se exatamente que informações são pertinentes na descrição de uma animação.

Está alternativa também não foi considerada satisfatória, principalmente pelo fato de que a especificação de uma animação é extremamente dependente da ferramenta utilizada. Por exemplo, uma ferramenta de animação facial pode, na descrição de um movimento, fazer uso de parâmetros como “ângulo de abertura de olhos”, ou “tipo de sorriso”, que não têm nenhum sentido fora do contexto da ferramenta em questão.

Assim sendo, faz-se necessária a definição de um modelo intermediário, que seja rígido o bastante para permitir apenas a troca de informações relevantes para o processo de

animação, porém flexível o suficiente para não restringir o uso de ferramentas não previstas durante a elaboração do ambiente.

Visando atingir um modelo equilibrado, foi definido que as mensagens com informações sobre a cena teriam os seguintes campos: nome do elemento, nome do parâmetro, e valor do mesmo em dado instante de tempo. Uma vez que o sistema é completamente sincronizado, não há necessidade de transmitir também o instante de tempo ao qual a mensagem se refere.

O nome do elemento é simplesmente um identificador único para um elemento durante uma animação, e é definido na mensagem de criação do elemento.

O nome do parâmetro é um identificador para uma característica do elemento que é significativa para a animação. Este parâmetro não é conhecido pelo gerente, mas apenas pelas ferramentas para as quais o parâmetro é relevante. Por exemplo, uma ferramenta de dinâmica é capaz de processar informações sobre parâmetros como “força” e “massa”, que não fazem sentido nem para o gerente nem para ferramentas cinemáticas. Se um exibidor fosse construído com a capacidade de exibir vetores associados à força aplicada a elementos da cena, ele simplesmente enviaria requisições ao gerente, perguntando pelo valor do parâmetro “força” dos elementos da cena. Apenas as ferramentas dinâmicas responderiam a esta requisição, enquanto outras ferramentas simplesmente a ignorariam, por não tratarem tal parâmetro.

O valor do parâmetro é ainda mais livre. Pode ser qualquer combinação de números inteiros, números reais e vetores de caracteres. Por exemplo, um parâmetro de posição normalmente tem como valor um conjunto de três números reais, representando os três eixos.

2.3.2.3 Mensagens de Requisição de Informação

Estas mensagens são bastante semelhante às mensagens contendo informações sobre a cena. Elas são enviadas de ferramentas para o gerente, que as repassa para outras

ferramentas, e podem ser direcionadas a um elemento específico ou a toda uma classe de elementos. As mensagens ainda especificam o parâmetro que está sendo requisitado.

Mensagens para um elemento específico são simplesmente retransmitidas pelo gerente para o processo que controla o elemento. As outras mensagens se destinam a toda uma classe de elementos, como, por exemplo, todos os *"atores"* da cena, ou todas as "*luzes"*. O gerente retransmite esta mensagem para todos os processos que controlam um ou mais elementos do tipo especificado, e retransmite todas as respostas para o processo que originou a requisição.

Para reduzir o fluxo de mensagens, estas requisições tem valor para todos os quadros a partir daquele no qual são enviadas, até serem canceladas por outra mensagem. Assim, um sistema de exibição de imagens, por exemplo, só precisa fazer uma única requisição para cada parâmetro, no quadro inicial da animação.

2.4 Estrutura de Comunicação

Nesta seção é melhor detalhada a estrutura de comunicação entre o gerente e cada um dos processos. Conforme já foi mencionado, o gerente vê todos os processos de forma idêntica, o que significa que toda a estrutura de comunicação entre o gerente e cada ferramenta é idêntica.

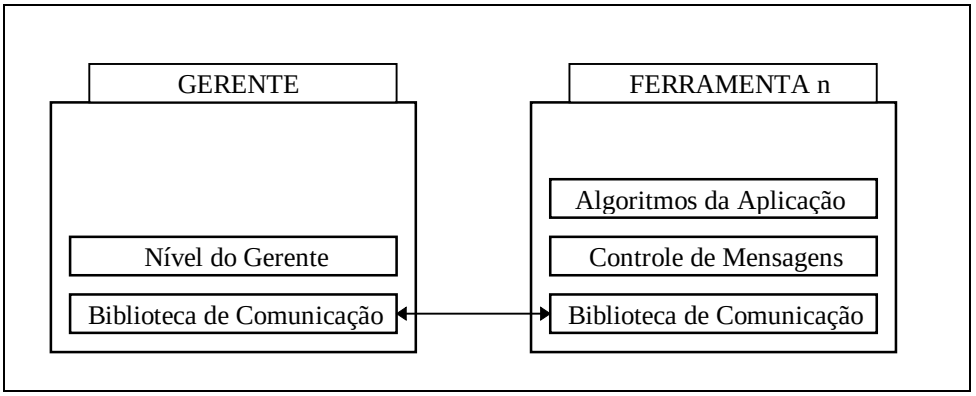

fig 2.2 - Níveis da estrutura de comunicação entre o gerente e as ferramentas

A figura 2.2 apresenta um diagrama mostrando os diferentes níveis nos quais foi dividida a tarefa de estabelecer um protocolo de comunicação. Cada nível é melhor descrito em uma das seções seguintes.

2.4.1 A Biblioteca de Comunicação

A *biblioteca de comunicação* é o nível responsável pela interface entre os processos e o ambiente no qual eles estão sendo executados. Ela oferece às ferramentas e ao gerente um conjunto de funções (anexo A-1) para troca de informações. Estas funções são mapeadas para mensagens que são enviadas através do meio de comunicação entre processos disponível na plataforma onde o sistema está executando.

Essencialmente, a *biblioteca de comunicação* implementa duas tarefas básicas: fornecer aos processos uma interface de comunicação independente de plataforma; garantir a consistência sintática do modelo de comunicação definido no ambiente. Ambas as tarefas são descritas nas seções seguintes.

2.4.1.1 Interface de Comunicação Independente de Plataforma

Ao invés de utilizar diretamente os recursos de comunicação entre processos disponíveis em determinado ambiente, todos os processos fazem uso de funções definidas na *biblioteca de comunicação*. É apenas a biblioteca que é implementada em função dos recursos de determinada plataforma.

Assim, a *biblioteca de comunicação* é dependente da plataforma em que o sistema está executando, fazendo uso de *sockets*, *pipes*, memória compartilhada, *DDE*, ou outros mecanismos de comunicação presentes em determinado ambiente. As ferramentas, por outro lado, só tem contato com estes recursos através da biblioteca.

Com isto o custo de transporte do sistema para outros ambiente é reduzido consideravelmente.

2.4.1.2 Sintaxe do Protocolo de Comunicação

Conforme mencionado na seção 2.3, a comunicação entre as ferramentas do ambiente obedece a um protocolo específico. A *biblioteca de comunicação* esconde das ferramentas a sintaxe adotada pelo protocolo, fornecendo apenas um conjunto de funções para definição das operações e dos parâmetros a serem transmitidos. A *biblioteca de comunicação* também garante, assim, que as mensagens transmitidas entre as ferramentas obedecerão a sintaxe estabelecida.

2.4.2 O Nível do Gerente

Conceitualmente, o gerente é bem mais simples que as ferramentas, já que existe apenas um em todo o ambiente e ele possui uma tarefa bem definida, enquanto as ferramentas podem implementar qualquer tarefa.

Assim sendo, além da *biblioteca de comunicação*, o único outro nível definido para o gerente é a própria implementação do mesmo, cuja responsabilidade é, essencialmente, processar as mensagens que recebe e retransmití-las aos processos corretos. A seção 2.2.1. já apresentou as atividades principais de responsabilidade do *gerente*.

2.4.3 O Nível de Controle de Mensagens

O protocolo de comunicação entre as ferramentas e o gerente define uma sintaxe, ou seja, uma forma como as mensagens são codificadas, e uma semântica, isto é, o efeito que cada mensagem deve causar no ambiente e que respostas deverão ser enviadas para a ferramenta que transmitiu a mensagem.

Na seção 2.4.1, foi descrita a *biblioteca de comunicação*, que é responsável pela sintaxe das mensagens. O *nível de controle de mensagens*, por sua vez, é responsável pela semântica envolvida na transmissão. É este nível que interpreta as mensagens enviadas pelo gerente e responde às mesmas.

O *nível de controle de mensagens* faz uma interface entre a *biblioteca de comunicação* e os algoritmos que implementam as tarefas de determinada ferramenta. O objetivo deste nível, portanto, é isolar os algoritmos particulares de cada ferramenta do ambiente de integração, permitindo que tais algoritmos sejam desenvolvidos sem uma constante preocupação com as mensagens que precisam ser trocadas com o restante do ambiente.

Ao contrário da *biblioteca de comunicação*, o *nível de controle* possui uma implementação específica para cada ferramenta, variando de acordo com as informações que ele precisa trocar com o ambiente, as quais dependem das tarefas que cada ferramenta implementa.

As seções seguintes descrevem em mais detalhes as principais tarefas do *nível de controle de mensagens*.

2.4.3.1 Requisição de Informações

Assim que a ferramenta é ativada pelo gerente, o *nível de controle* envia uma mensagem requisitando todas as informações que aquela ferramenta irá necessitar. Por exemplo, no caso de um exibidor, tipicamente ele enviará uma mensagem requisitando que a cada instante de tempo lhe seja informado um conjunto de parâmetros de cada elemento da cena. Os parâmetros em si dependerão das características específicas do exibidor: um exibidor em aramado (*wire frame*) poderia requisitar apenas a posição e orientação dos atores, enquanto um exibidor mais sofisticado poderia necessitar de informações de transparência, características de superfície, etc.

Embora o procedimento padrão seja requisitar tais informações no início da cena, o *nível de controle de mensagens* pode eventualmente fazer tais requisições em qualquer instante de tempo.

2.4.3.2 Troca de Informações

Cada ferramenta pode possuir um conjunto de informações que são acessíveis ao ambiente. Tipicamente, tais informações se referem aos elementos da cena, mais especificamente o valor dos parâmetros dos elementos que aquela ferramenta manipula.

É função do *nível de controle de mensagens* garantir ao ambiente acesso a estas informações. Isto é implementado através do processamento de dois tipos de mensagens, mensagens de atribuição e mensagens de leitura.

Quando recebe uma mensagem de atribuição, o *nível de controle de mensagens* atribui ao parâmetro especificado na mensagem um novo valor também especificado. De forma análoga, quando recebe uma mensagem de leitura, lê o valor de determinado parâmetro e envia uma mensagem de resposta, informando seu valor naquele instante de tempo.

Assim, o *nível de controle* está permitindo um acesso controlado do ambiente a uma determinada parte da área de dados da ferramenta. Está área de dados comum é atualizada tanto pela ferramenta, executando seu processamento normal, quanto pelo ambiente, só que indiretamente, passando pelo *nível de controle*, em função de mensagens originadas em outras ferramentas.

Na prática, o *nível de controle* está implementando, em conjunto com o *gerente*, um armazenamento distribuído das informações sobre a cena.

2.4.3.3 Controle de Múltiplas Instâncias

O *nível de controle* é responsável também pela criação de múltiplas instâncias dos algoritmos que implementam as tarefas particulares de uma ferramenta. Isto visa permitir que uma mesma ferramenta possa controlar vários elementos de uma cena de forma transparente para tais algoritmos.

Supondo, por exemplo, uma ferramenta que controle o movimento de um corpo articulado: quando surge um elemento deste tipo numa determinada animação, a ferramenta é acionada e passa a controlar tal ator. Porém, podem existir vários corpos articulados presentes no mesmo instante de tempo, todos tendo que ser controlados pela mesma ferramenta.

Uma solução seria criar vários processos, um para controlar cada ator. O problema desta solução é principalmente o custo, já que em determinadas circunstâncias pode ser demasiado caro computacionalmente criar um processo para controlar cada ator de um determinado tipo.

Ainda mais grave é o caso de ferramentas que precisam ter conhecimento de toda a cena, como sistemas de exibição de imagens e detectores de colisão. Para estes sistemas seria inviável definir um processo como responsável por cada elemento da cena.

O *nível de controle* fornece uma alternativa: cada vez que um novo elemento é criado, o *nível de controle* cria uma nova instância de todo o conjunto de algoritmos que controlam ou precisam de informações sobre este elemento, e ainda controla e redistribui as mensagens de/para o ambiente conforme necessário.

2.4.4 Nível dos Algoritmos da Ferramenta

Cada ferramenta implementa uma ou mais tarefas específicas. O ambiente não faz nenhum tipo de consistência destas tarefas, nem se comunica com este nível por qualquer outra forma que não o *nível de controle*, conforme descrito anteriormente.

Todo o processamento que este nível faz que afeta algum elemento da cena precisa necessariamente ser armazenado na área de dados que é compartilhada com o ambiente através do *nível de controle*. Da mesma forma, sempre que alguma informação da cena for necessária ao nível, ela é buscada nesta área compartilhada de dados.

Assim, este nível está quase completamente isolado do ambiente. Toda a comunicação é feita através de uma parte da sua área de dados, de forma que para ele é

irrelevante se o programa está associado a todo um ambiente de integração, ou se simplesmente esta área de dados é atualizada por outras rotinas internas à própria ferramenta.

É importante salientar, também que, exceto pelo uso desta área compartilhada de dados, nada mais é assumido pelo modelo sobre o funcionamento dos algoritmos neste nível. Assim, o sistema é bem flexível, permitindo a implementação de virtualmente qualquer tipo de tarefa.

3. Integração de Ferramentas

Existem, na literatura, vários enfoques diferentes para a integração de ferramentas. Esta diversidade é resultado direto da complexidade do problema, não havendo uma solução genérica que se aplique bem a todas as formas de integração possíveis. Neste capítulo são descritas as principais alternativas presentes na literatura e comparadas com as necessidades do caso particular tratado neste trabalho, a integração de ferramentas de animação.

3.1 Frameworks

Segundo Wagner em [WAG 94], "*frameworks* são plataformas para a construção de ambientes integrados e abertos de projeto de sistemas". Para compreender corretamente esta frase é necessário ter claro o conceito de sistemas abertos: um sistema aberto é um sistema composto por diferentes *softwares* ao qual novas ferramentas podem ser facilmente incluídas.

Assim sendo, os *frameworks* fornecem a infra-estrutura para a construção de sistemas abertos, oferecendo recursos que são independentes das características particulares de cada sistema. Para os construtores dos sistemas, eles oferecem facilidades para integrar novas ferramentas ao ambiente, enquanto que para os usuários, disponibilizam recursos de gerenciamento da execução das ferramentas.

A principal justificativa para seu uso está associada ao custo envolvido: o desenvolvimento de um sistema aberto seria uma tarefa muito complexa e, possivelmente, inviável economicamente, não fossem utilizados os recursos que já estão prontos e disponíveis nos *frameworks*, como o armazenamento das informações em bancos de dados e os recursos de comunicação entre as ferramentas.

Embora diferentes *frameworks* ofereçam diferentes recursos tanto aos construtores de sistemas quanto aos usuários finais, existem alguns recursos básicos que devem ser fornecidos, dos quais os principais são citados a seguir:

Armazenamento, manipulação e recuperação de dados: todos os dados utilizados pelas ferramentas devem ser armazenados numa base de dados única, evitando, assim, problemas de redundância e inconsistência nas informações manipuladas por diferentes ferramentas. Portanto, devem estar disponíveis serviços semelhantes aos de um sistema de gerência de banco de dados.

Gerência do processo: o usuário deve ter a possibilidade de controlar a execução de um conjunto de tarefas. Dada a especificação de uma tarefa a ser executada, podem existir diferentes caminhos que levam para soluções satisfatórias, envolvendo a ativação de diferentes ferramentas em seqüências diferentes. O *framework* deve permitir que os usuários possam interferir na escolha das ferramentas a serem chamadas.

Encapsulamento de ferramentas: o *framework* deve possibilitar a integração de ferramentas que não tenham sido construídas em função do mesmo. Ou seja, deve ser possível integrar ferramentas de outras empresas ou centros de pesquisa, que tenham sido desenvolvidas sem a preocupação com compatibilidade com os recursos do *framework*.

Interação com o Usuário: o *framework* deve também fornecer recursos para a construção de interfaces uniformes para as ferramentas, de tal forma que os comandos utilizados em cada ferramenta não sejam incompatíveis entre si.

3.2 IPSE - Integrated Project Support Environment

A expressão IPSE pode ser traduzida como ambiente de auxílio ao projeto integrado de sistemas. Segundo Morgan em [MOR 87], um ambiente IPSE consiste de um conjunto coordenado de sofisticadas ferramentas de *software* capazes de ajudar no desenvolvimento, projeto e gerenciamento de sistemas de software complexos.

A metodologia IPSE propõe uma integração entre todas as ferramentas desde o nível do sistema operacional, fornecendo um banco de dados onde as informações de cada ferramenta são armazenadas e disponibilizadas para as demais e uma interface comum aos

usuários. Neste enfoque, o ambiente UNIX é apresentado em [MOR 87] como o primeiro passo para o desenvolvimento de um ambiente IPSE. O passo seguinte incluiria a utilização de um banco de dados no lugar do sistema de arquivos e recursos mais avançados de comunicação entre as ferramentas.

A principal particularidade da metodologia IPSE, quando comparada com a alternativa de *frameworks* descrita anteriormente, é que aqui a proposta é muito mais abrangente, visando construir todo um sistema operacional voltado para a integração de ferramentas. Assim, todas as ferramentas que estivessem disponíveis em determinado sistema operacional estariam automaticamente integradas entre si, já que este é que se responsabilizaria por esta integração.

O principal problema associado a esta alternativa para a integração das ferramentas é o alto custo envolvido na sua implementação. Existe o custo considerável de construir o ambiente IPSE e ainda o custo de construir um conjunto significativo de ferramentas voltadas para o uso de tal metodologia.

3.3 Componentes de Software

Uma alternativa de integração de ferramentas que pode se tornar um padrão de fato na indústria é o uso de *componentes de software*. Neste enfoque, o usuário não utiliza um sistema, mas sim ferramentas que se comunicam entre si, das quais ele pode escolher as que lhe forem mais convenientes. Assim, ao invés de utilizar um único programa, desenvolvido por um único fabricante, ele monta seu próprio sistema escolhendo algumas entre as ferramentas disponíveis no mercado.

Um dos exemplos mais bem sucedidos desta alternativa é o padrão OLE 2.0 - Object Linking and Embeding - desenvolvido pela Microsoft, que “é composto de uma série de funções que oferecem poderosos recursos para a criação de documentos constituídos de múltiplas fontes de informação originárias de diferentes aplicações [MIC 93]”.

O padrão OLE é voltado para a manipulação de documentos, onde um documento pode ser qualquer tipo de informação, como um texto, uma imagem, uma descrição de um circuito integrado, uma animação, ou uma composição de outros documentos. Pela filosofia de funcionamento da OLE, o usuário define um documento que ele deseja manipular, e o ambiente lhe oferece a escolha entre as ferramentas que podem ser utilizadas para editar tal documento.

Diferente dos *frameworks*, a OLE não prevê a utilização de um banco de dados para armazenar as informações das ferramentas. As informações são todas armazenadas no documento que está sendo manipulado, organizadas numa estrutura de diretórios interna a um arquivo, a qual é manipulada através de funções disponibilizadas pela OLE.

Sob certo aspecto, esta proposta é um meio termo entre *frameworks* e ambientes IPSE. À semelhança de ambiente IPSE, a proposta do padrão OLE prevê a integração de ferramentas genéricas, que não são direcionadas para um fim específico. Por outro lado, a proposta é menos radical que os ambientes IPSE, por não propor a construção de um sistema operacional voltado à integração de ferramentas. Como os *frameworks*, a OLE executa sobre um sistema operacional tradicional.

Futuras versões do padrão OLE possivelmente executarão de forma mais integrada com os sistemas operacionais Windows e principalmente Windows NT, resultando num ambiente que atende a muitos dos objetivos das propostas de ambientes IPSE.

3.4 Integração de Ferramentas de Animação

Embora na literatura especializada sejam apresentadas diversas técnicas de integração de ferramentas, nenhuma delas é ideal para o objetivo que este trabalho busca atingir, a integração de ferramentas de animação. Ainda que existam alguns pontos em comum entre a proposta deste trabalho e as alternativas descritas anteriormente, existem também alguns aspectos do ambiente que o distinguem dos casos típicos apresentados na literatura. Tais diferenças são melhor descritas nas seções seguintes.

3.4.1 Cooperação entre Ferramentas

As alternativas apresentadas, em especial os *frameworks*, enfocam principalmente a integração das ferramentas que executam em série sobre um conjunto de dados. Pode-se comparar esta integração com uma linha de montagem, onde cada ferramenta atua sobre uma ou mais peças, produzindo um resultado que é repassado à ferramenta seguinte. A figura 3.1 exemplifica esta estrutura.

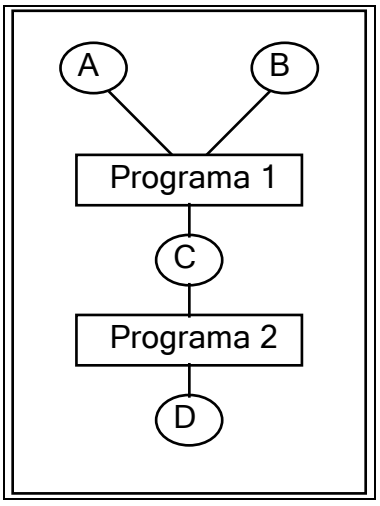

fig 3.1 - Integração de ferramentas estilo *Framework*.

Conseqüentemente, uma parte significativa do estudo de ambientes de integração é voltada para otimizar este processo, por exemplo com a definição de formas de armazenar as informações em bancos de dados e o tratamento de questões como controle de versões e de acesso.

O padrão OLE, ainda que operando sobre outra filosofia, também se concentra em ferramentas que processam dados: cada ferramenta atua sobre uma área de dados própria - um documento - que é parte de um documento maior.

A proposta de integração de ferramentas apresentada neste trabalho é significativamente diferente porque os programas que farão parte do ambiente não executarão em série, mas sim de uma forma cooperativa, aplicando seus algoritmos em conjunto sobre uma mesma base de dados. No caso do padrão OLE, por exemplo, isto seria equivalente a diversas ferramentas atuarem simultaneamente sobre uma mesma parte de um documento.

Na presente proposta, diferentes programas controlarão atores diferentes que compartilham um mesmo cenário. Não há uma seqüência de passos a serem tomados um após o outro, mas sim um conjunto de ações que devem ser efetuadas por todas as ferramentas em cada instante de tempo da animação.

Esta diferença fundamental entre a presente proposta e os sistemas tradicionais de integração tem como conseqüência que muitas questões tratadas na literatura não são relevantes para o presente problema, enquanto algumas dificuldades na definição deste ambiente não são tratadas pelos estudos de integração voltados para outras áreas, principalmente a manipulação de determinadas informações simultaneamente por várias das ferramentas do sistema.

3.4.2 Sincronização de Ferramentas

Como conseqüência direta do funcionamento cooperativo das ferramentas, temos um problema de sincronização. Se as ferramentas não atuarem de uma forma sincronizada teremos como resultado que as cenas serão geradas de forma inconsistente, com cada ator posicionado em função de um instante diferente de tempo, conforme a velocidade maior ou menor da execução desta ou daquela ferramenta.

A solução mais simples para esta questão é sincronizar todas as ferramentas, fazendo com que um novo instante de tempo só comece a ser gerado após todas as ferramentas terem completado o instante de tempo anterior. Esta alternativa é perfeitamente aceitável para o funcionamento do ambiente em uma única máquina, já que não há uma perda muito significativa de desempenho, pois o processador estará sempre executando uma das ferramentas.

Quando, entretanto, supomos um ambiente com vários processadores, a alternativa de sincronização de todo o ambiente começa a ficar bem mais cara, pois alguns processadores eventualmente ficarão ociosos enquanto esperam que outras ferramentas terminem de executar em outros processadores.

Uma vez que o presente trabalho propõe um sistema orientado a um único processador, a alternativa de sincronização de todo o ambiente foi considerada como perfeitamente razoável. O capítulo 7 discute alternativas que poderiam ser implementadas numa possível revisão do sistema para a execução de forma paralela.

3.4.3 Armazenamento dos Dados

Num sistema típico de *framework*, bem como num ambiente IPSE, a forma como os dados são armazenados é uma questão fundamental, como o é no ambiente proposto, porém por razões diferentes.

As aplicações que compõem um ambiente integrado típico são, muitas vezes, intensivas em processamento de dados, e armazenar e recuperar estes dados de uma forma eficiente e segura é uma das principais vantagens oferecidas pelos ambientes de integração.

As aplicações que o presente ambiente busca integrar, por outro lado, são intensivas em processamento, e não em dados. Mais precisamente, a quantidade de informações que precisa ser trocada entre as ferramentas é relativamente reduzida, porém o processamento destes dados pode ser bem caro computacionalmente. Na verdade, o conjunto total de dados manipulado pelas ferramentas é bem maior, porém apenas um pequeno conjunto de informações precisa ser compartilhado pelas ferramentas do ambiente. Por exemplo, os sistemas de síntese de imagens geram mapas de bits que ocupam muito espaço, porém tais dados não precisam ser compartilhados porque não influenciam na animação.

Assim, no presente ambiente a quantidade de informações que precisa ser trocada entre as ferramentas não é muito significativa. Por outro lado, estas informações precisam ser trocadas a cada instante de tempo do processo de animação. Obviamente, torna-se necessário otimizar ao máximo o processo de troca de informações.

Visando reduzir ao máximo o fluxo de mensagens, a alternativa mais eficiente é manter os dados de uma forma distribuída, na qual cada informação permanece no processo que a produziu, sendo repassada para outras ferramentas apenas no momento em que for feita uma requisição neste sentido. Assim, apenas aquelas informações que são necessárias em outro processo são transmitidas pela ferramenta.

%%%fechar com inicio da seção.

3.4.4 Reorganização Dinâmica do Fluxo de Mensagens

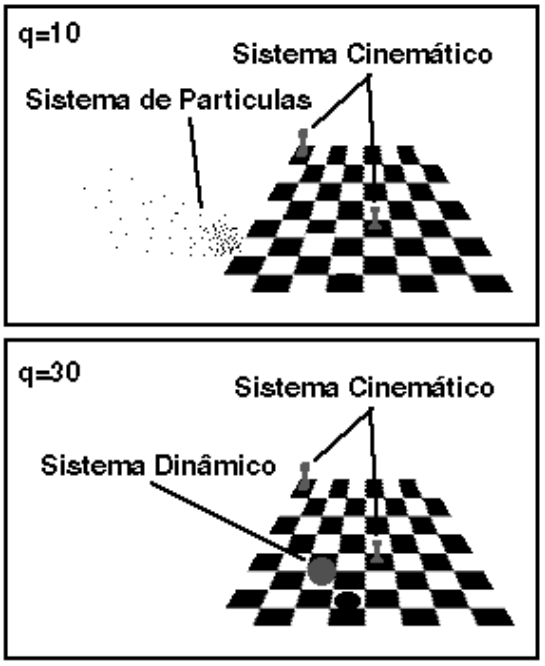

fig 3.2 - Diferentes ferramentas em diferentes instantes de tempo.

O ambiente de integração proposto precisa ainda lidar com outra questão: conforme uma animação vai sendo construída e atores vão entrando e saindo de cena, processos que controlam estes atores vão sendo ativados e desativados. A figura 3.2, que mostra os processos presentes num ambiente em dois instantes de tempos diferentes ilustra esta questão.

O problema está fortemente relacionado com o fato das informações sobre os atores estarem distribuídas entre as ferramentas. Uma vez que a estrutura do ambiente muda dinamicamente, é necessário definir como cada ferramenta pode descobrir que processos estão ativos e associados a cada ator. Esta informação é indispensável para que a ferramenta possa obter informações sobre o ator em questão e, eventualmente, trocar comandos com o processo que o controla.

Duas alternativas foram consideradas para resolver este problema, as quais são apresentadas nas seções seguintes.

3.4.4.1 Atualização Constante das Ferramentas

Uma alternativa para permitir que o ambiente reorganize dinamicamente a comunicação entre seus processos é informar todas as ferramentas sempre que alguma mudança na cena resulte na ativação ou desativação de alguma ferramenta.

Por exemplo, pode-se supor que em determinado momento um novo elemento entra em cena, elemento este controlado por uma ferramenta de animação até então inativa. Todas as ferramentas do ambiente que eventualmente poderiam trocar informações sobre o objeto precisam ser informadas de que devem estabelecer uma comunicação com o processo recém ativo.

A principal razão que levou à decisão de não utilizar esta alternativa é que ela coloca nas ferramentas a responsabilidade de conhecer a cena em cada instante de tempo. Está sendo colocada uma responsabilidade nas ferramentas que nada tem a ver com a tarefa específica de cada uma delas, além de se acrescentar ao ambiente a tarefa de manter atualizada e coerente uma informação duplicada em inúmeros processos.

3.4.4.2 Construção de um Sistema de Distribuição de Mensagens

A segunda alternativa considerada foi concentrar em uma única ferramenta a informação de como está organizada a cena em cada instante de tempo. Apenas esta ferramenta fica responsável pela comunicação com todas as demais, retransmitindo as mensagens que recebe de acordo com sua informação de como o ambiente está estruturado em determinado momento.

A principal vantagem desta alternativa é que, uma vez que as demais ferramentas do ambiente não mais se comunicam diretamente entre si, elas já não estão executando de uma forma estática e, portanto, não armazenam nenhum conhecimento sobre o estado do ambiente.

Por exemplo, utilizando-se a mesma situação apresentada na seção anterior, onde a entrada de um novo elemento em cena causa a ativação de um novo processo. Nenhuma

outra ferramenta é notificada sobre a ativação do novo processo, exceto a ferramenta central, pois todas as mensagens trocadas entre as ferramentas não são endereçadas diretamente para o processo. Uma ferramenta de exibição pode requisitar informações sobre todos os elementos presentes na cena, sem saber quais processos controlam tais elementos, já que o processo central tem conhecimento suficiente para distribuir sua mensagem para as ferramentas corretas.

4. Integração de Ferramentas ao Ambiente

O presente capítulo descreve as características de ferramentas que podem fazer parte do modelo e a forma como são integradas no ambiente.

O modelo não determina as atividades que cada ferramenta do ambiente executa, e é perfeitamente possível integrar ao ambiente novas ferramentas com funcionalidades ainda não previstas. Isto é possível porque o ambiente não assume nenhum conhecimento sobre o funcionamento interno de cada ferramenta.

Entretanto, na construção de um sistema de animação algumas ferramentas básicas precisam existir. Determinadas tarefas, como a especificação da animação e a geração das imagens, fazem parte mesmo dos sistemas mais simples de animação modelada. Assim, embora seja impossível prever todas as ferramentas que podem fazer parte de um sistema, é possível analisar os casos básicos, comuns a qualquer sistema que venha a ser construído com base neste ambiente.

As seções seguintes descrevem as características mínimas de ferramentas que sejam integradas ao sistema visando executar as atividades básicas de especificação da animação, geração dos movimentos e síntese das imagens.

4.1 Especificação da Animação

Uma das tarefas essenciais de um sistema de animação por computador é fornecer ao usuário algum recurso para especificar como determinada animação irá transcorrer. Na literatura, são apresentadas inúmeras formas do usuário interagir com um sistema para construir uma animação. As seções seguintes descrevem a integração de ferramentas que implementam as técnicas mais conhecidas no ambiente proposto.

4.1.1 Especificação de Animações Utilizando Linguagens de Propósitos Gerais

Esta é a alternativa mais poderosa de construção de animações, porém a de maior complexidade para o usuário. A animação é definida numa linguagem de programação, aproveitando-se de todos os recursos que tal linguagem fornece.

Normalmente, o uso de linguagens de propósitos gerais é feito em conjunto com bibliotecas ou pré-processadores, que acrescentam recursos específicos de animação a linguagem em questão, facilitando seu uso na construção de uma animação.

Embora tais recursos possam ser utilizados em conjunto com o ambiente, não é tão necessário no presente caso, pois o ambiente, incluídas aí as demais ferramentas, já trata todas as questões mais específicas do processo de animação, como geração de movimentos cinemáticos e dinâmicos, síntese de imagens, etc.

Portanto, nesta alternativa o ambiente já está fazendo o papel de uma biblioteca, fornecendo vários recursos de animação. O programa que conterá todas as ações da animação não precisa tratar tais questões, ele precisa apenas transmitir para o ambiente as mensagens com informações sobre o transcorrer da cena.

4.1.2 Especificação Utilizando Roteiros

Nesta alternativa, uma ferramenta lê e interpreta um roteiro contendo informações sobre a cena numa linguagem específica de descrição de animações. Esta alternativa é relativamente fácil de implementar e é bem mais fácil de ser utilizada por um usuário leigo que uma linguagem de propósitos gerais.

Integrada ao ambiente proposto, uma ferramenta de leitura de roteiro precisa apenas retransmitir ao ambiente as informações que lê de arquivo nos instantes de tempo corretos.

Existe, porém, uma questão relevante que não pode deixar de ser tratada por um leitor de roteiro que trabalhe respeitando a metodologia proposta pelo modelo. O leitor não deve acrescentar restrições que não existem no modelo.

Mais especificamente, o leitor deve ser capaz de aceitar qualquer parâmetro como associado a qualquer ator da cena, como o faz o gerente que controla o ambiente. De outra forma, o leitor estará assumindo algum conhecimento sobre os parâmetros dos elementos em cena, e, portanto, sobre as demais ferramentas presentes no ambiente. Com isso, o ambiente não estará trabalhando na forma flexível proposta.

4.1.3 Especificação Utilizando Sistemas Orientados a Artistas

Sistemas orientados a artistas permitem que seus usuários editem visualmente as mudanças que desejam efetuar na animação, utilizando uma interface onde selecionam os elementos e definem suas propriedades em qualquer instante de tempo.

Integrar ao ambiente uma ferramenta de edição direta é relativamente simples, desde que a ferramenta trabalhe com um determinado conjunto pré-definido de características dos elementos presentes na cena. Porém, conforme citado na seção anterior, tal alternativa restringe consideravelmente a flexibilidade do sistema.

A construção de uma ferramenta de edição interativa capaz de trabalhar de forma elegante com todas as ferramentas do ambiente, porém, não é uma tarefa trivial, pois, assim como cada ferramenta de geração de movimentos trata com parâmetros diferentes, tais parâmetros exigem um tratamento especifico numa ferramenta de edição direta.

Em outras palavras, suponha que determinada ferramenta manipula parâmetros como forças e torques. A ferramentas de edição da animação precisará saber como oferecer ao usuário recursos para editar diretamente estas forças e torques. Esta tarefa não é trivial, pois a ferramenta de edição não tem como prever recursos de interação com todos os parâmetros possíveis de serem utilizados num sistema de animação qualquer.

Prever o uso de um sistema orientado a artistas de forma completamente integrada ao ambiente é uma tarefa de grande complexidade, sendo, desde o início do desenvolvimento deste trabalho, considerada como uma possível extensão do ambiente. Assim, uma descrição mais detalhada desta questão pode ser encontrada no capítulo 7, na seção de trabalhos futuros.

4.2 Geração de Movimentos

Sem dúvida, uma das tarefas mais difíceis na construção do modelo proposto foi possibilitar a integração de ferramentas que implementem diferentes algoritmos de geração de movimentos.

O problema está associado ao fato de que algoritmos diferentes manipulam estruturas de dados diferentes, e, até certo ponto, incompatíveis. A seção 4.2.1 apresenta alguns dos principais algoritmos de geração de movimentos encontrados na literatura, e descreve os elementos que, para eles, são relevantes numa cena. A seção ...

4.2.1 Principais Algoritmos de Controle de Movimentos

Constantemente surgem, na literatura, novos métodos para gerar movimentos, que procuram atingir dois grandes objetivos: facilitar a tarefa de descrever o movimento desejado e gerar movimentos com um grau maior de realismo. Dentre os algoritmos que já foram criados, alguns foram mais largamente estudados e usados, e são apresentados aqui.

- **Interpolação de Quadros-Chaves:**

Sturman, em [STU 90], apresenta uma descrição bastante clara da técnica de animação através de quadros-chaves: “um quadro em uma animação é basicamente a descrição de um particular estado do mundo num particular instante de tempo. Num sistema de quadros-chaves, o animador não precisa descrever cada quadro. Ao invés disto, ele descreve um conjunto de quadros-chaves a partir dos quais o sistema de animação pode interpolar os quadros intermediários. Interpolação dos valores intermediários pode ser linear, *splines* cúbicas, cosenos, etc.”

Embora a interpolação de quadros-chaves seja principalmente usada na definição de toda a cena, ou pelo menos de todas as características de determinado elemento, muitos sistemas são flexíveis o suficiente para definir quadros-chaves que se aplicam apenas a um elemento da cena [GOM 84], ou mesmo a uma característica de um elemento. Assim sendo, a rigor a técnica não define um determinado modelo para a cena, podendo ser aplicada diretamente a qualquer parâmetro de qualquer elemento da cena, como forças, torques, velocidades, etc.

- **Geração de Movimentos Cinemáticos:**

Algoritmos cinemáticos são usados na geração de movimentos, utilizando equações cinemáticas. O movimento dos objetos é definido com base na variação de suas posições em função de valores de velocidade e aceleração. Embora seja relativamente simples definir um movimento com esta técnica, fazer com que o movimento seja realístico é mais difícil, pois o próprio artista tem que aplicar velocidades e acelerações semelhantes as de uma situação real, o que pode ser bem difícil em situações que envolvem forças de atrito e colisões.

Naturalmente, para um algoritmo cinemático uma animação é definida em função da posição, velocidade e aceleração de cada elemento presente na cena.

- **Geração de Movimentos com Dinâmica Direta:**

Em algoritmos que usam dinâmica direta, o movimento dos objetos é resultado das forças aplicadas sobre eles. Conforme Green em [GRE 91], “esta técnica tem a vantagem de produzir movimentos realísticos e exigir uma quantidade modesta de comandos de controle”. Porém, existem algumas dificuldades inerentes a técnica que ainda não foram completamente resolvidas, especialmente o fato das forças e torques que comandam os movimentos não serem, para a maioria dos animadores, uma forma natural de especificar movimentos.

Para estes algoritmos, o cenário de uma animação é constituído por objetos aos quais estão associadas, além da posição e orientação no espaço, grandezas físicas como

massa e centro de gravidade, e, eventualmente, coeficientes de atrito e elasticidade. Além disso, fazem parte da animação forças e torques que atuam sobre os objetos.

- **Geração de Movimentos com Dinâmica Inversa:**

Na dinâmica inversa, o movimento dos objetos ainda é feito baseado em parâmetros como forças e torques, porém estas forças, ao invés de serem especificadas diretamente pelo artista, são calculadas automaticamente em função de objetivos a serem atingidos, como posições que o objeto deve ocupar em determinados instantes de tempo.

O objetivo de técnicas como esta é facilitar ao animador o controle dos movimentos, já que, como já foi dito, o controle direto de forças e torques não é uma forma natural dos animadores controlarem a animação.

O cenário de uma animação, para estes elementos, é constituído pelos mesmos parâmetros de um sistema de dinâmica direta, adicionados dos parâmetros que o artista utiliza para controlar o movimento, que podem ser conjuntos de restrições e valores dos parâmetros para determinados quadros da cena.

- **Movimentos de Corpos Articulados:**

Os algoritmos de movimentos de corpos articulados tratam da movimentação de objetos que são compostos por sub-objetos ligados por articulações. Na verdade, a movimentação de corpos articulados geralmente está associada a uma das outras técnicas de geração de movimentos. Em [WIL 87], corpos articulados são controlados utilizando dinâmica direta, enquanto em [BOU 92], o controle é feito utilizando dinâmica direta e dinâmica inversa. Cinemática direta e inversa também são usadas em conjunto para definir movimentos de corpos articulados, como pode ser visto em [BOU 92] e [PHI 90]. Mesmo o técnica de quadro-chave é utilizada [STU 90].

O cenário para um sistema de corpos articulados depende naturalmente das técnicas de movimentação empregadas, bem como de informações sobre a forma como os objetos estão ligados entre si para formar o corpo articulado, além, naturalmente, de outras informações como, por exemplo, restrições.

- **Movimentos de Partículas:**

Na movimentação de partículas, ao invés do artista especificar diretamente, seja por qual algoritmo for, a movimentação de cada elemento presente na cena, ele especifica apenas a movimentação de um elemento de determinado grupo, ou do grupo como um todo. Os algoritmos de movimentação de partículas calculam a movimentação particular de cada elemento de um grupo baseados nas instruções do artista.

O cenário dependerá consideravelmente das técnicas adotadas para a movimentação definida pelo artista, que podem envolver interpolação de quadros-chaves, definição de uma trajetória ou outra forma qualquer de definição de movimento. Como parte do cenário estarão presentes, também, características específicas de como as partículas interagem entre si.

- **Animação Baseada em Tarefas:**

Nesta alternativa, são construídos movimentos complexos como composições de movimentos mais simples. Os movimentos básicos podem ser definidos por qualquer outra técnica de animação...

4.2.2 Integração de Algoritmos Diferentes

A seção 4.2.1 apresentou uma pequena amostra da diversidade de formas de gerar movimentos na animação por computador. Nesta seção trataremos de como o ambiente busca colocar ferramentas que se baseiem nestes ou noutros algoritmos num ambiente integrado.

A questão fundamental não é como cada algoritmo gera determinado movimento, ou como ele funciona internamente. A questão fundamental é que para cada algoritmo uma animação é constituída por informações diferentes, e nosso problema é permitir que todas estas informações coexistam simultaneamente numa mesma cena.

Quem tem o conhecimento de que informações são necessárias para determinado algoritmo é apenas a ferramenta que o implementa, e é neste conceito que se baseia o

modelo de integração proposto. Nenhuma ferramenta além daquelas que trabalham com informações cinemáticas precisa tratar parâmetros como velocidade e aceleração, por exemplo.

Assim sendo, o mecanismo de comunicação entre ferramentas foi construído de forma a ser capaz de transmitir informações sobre qualquer parâmetro de qualquer elemento, sem fazer nenhuma consistência sobre o mesmo. Isto é feito passando-se um identificador único para cada parâmetro, conforme descrito na seção 3.3.2. A consistência sobre a validade dos parâmetros fica como responsabilidade única das ferramentas que os tratam.

Com isso temos solucionado metade do problema. Cada ferramenta pode receber os parâmetros que precisa, que são gerados conforme descrito na seção 4.1, e é capaz, portanto, de gerar movimentos segundo seu algoritmo particular.

A segunda metade do problema é permitir a cooperação entre as ferramentas. O modelo tem como objetivo não apenas permitir que várias ferramentas participem da geração de uma cena, mas principalmente possibilitar que tais ferramentas cooperem entre si e que os elementos que elas controlam interajam na cena.

Essa cooperação é possível através do uso de um mesmo identificador para os parâmetros comuns utilizados por diferentes ferramentas. Por exemplo, todas as ferramentas integradas ao ambiente podem usar um mesmo identificador para seus parâmetros associados a posição dos objetos, pois, ainda que este parâmetro seja calculado por algoritmos diferentes em cada ferramenta, ele se refere a uma mesma propriedade dos objetos. Da mesma forma, todos os sistemas que utilizam alguma forma de cálculo de dinâmica podem utilizar os mesmos identificadores para parâmetros como força, torque e massa.

Isso torna possível que as ferramentas interajam umas com as outras. Por exemplo, uma ferramenta pode gerar movimentos baseado na posição dos demais

elementos na cena, mesmo que tais elementos sejam controlados por outras ferramentas que usam qualquer outro algoritmo.

Ainda assim, existe um problema quando algumas ferramentas não possuem os parâmetros que outras necessitam para interagir. Por exemplo, suponhamos uma ferramenta que trate colisões. Ela verifica todos os elementos da cena em cada instante de tempo e, quando identifica que dois elementos se chocaram, calcula a força resultante da colisão e envia para as duas ferramentas responsáveis pelos elementos uma mensagem que cria a nova força. Suponhamos, no entanto, que uma das ferramentas é controlada por um sistema de cinemática: tal ferramenta não é capaz de compreender informações de força e massa, essenciais no cálculo da colisão.

Não existe uma solução genérica para o problema, já que é uma questão completamente dependente das ferramentas de controle de movimento e da forma como elas interagem. O ambiente, entretanto, foi modelado de forma a fornecer as ferramentas todos os recursos para que elas próprias tratem do problema de uma forma que for aceitável.

Isto é possível porque cada ferramenta é capaz de lidar da forma que lhe for conveniente com parâmetros que ela não compreende, e o ambiente não define que informações cada ferramenta irá trocar com as outras, deixando espaço para que tratamentos específicos sejam criados.

No exemplo de colisão, existem inúmeras alternativas que podem ser adotadas. A mais simples é simplesmente que o tratador de colisões verifique se os objetos envolvidos são controlados por sistemas que respeitam as leis dinâmicas e só tratar as colisões nestes casos. Outras alternativas também existem, como, por exemplo, implementar um tratamento específico nas ferramentas. O tratador de colisões poderia, por exemplo, enviar uma mensagem informando que houve uma colisão para aquelas ferramentas que não trabalham com forças. Estas ferramentas poderiam, então, implementar seu tratamento próprio específico para o caso, como, por exemplo, iniciar um movimento na direção contrária a colisão, ou simularem uma explosão, ou ainda implementarem uma ação

qualquer definida no roteiro. Esta última alternativa foi implementada como exemplo, por ser a mais sofisticada, e é descrita no capítulo 6.

4.3 Síntese de Imagens

A terceira tarefa essencial num sistema de animação é a exibição da animação, que é feita visando atender dois objetivos básicos: permitir que o artista acompanhe o resultado da geração da animação e gerar as imagens foto-realísticas que comporão a animação.

Para oferecer ao artista o acompanhamento da animação, a prioridade é gerar as imagens de forma rápida, para diminuir o tempo de espera. Já na síntese das imagens finais, a prioridade é no resultado final, sendo o tempo necessário um fator de menor importância. Estes diferentes enfoques implicam em algumas diferenças na forma como cada alternativa deve ser implementada no modelo, e as seções seguintes discutem isso.

4.3.1 Acompanhamento da Animação

Uma ferramenta com este propósito funciona integrada diretamente ao ambiente. Ela requisita do ambiente as informações que é capaz de exibir ao longo da geração da cena. Por exemplo, um sistema que tenha condições de exibir imagens com texturas irá requisitar informações da textura dos objetos, enquanto um sistema mais simples, para o qual tal informação não é relevante, simplesmente não perguntará por parâmetros deste tipo.

Assim sendo, durante a execução de um sistema haverá um fluxo constante de informações sobre a cena, que serão transmitidas pelas ferramentas que geram movimentos, e recebidas pela ferramenta que exibe a cena.

4.3.2 Síntese de Imagens

A única diferença importante nas ferramentas que tem por objetivo a geração final das imagens está relacionada ao tempo necessário para gerar as cenas. Normalmente,

o artista só terá interesse em gerar imagens realísticas depois de um período de testes no qual tenha observado a cena várias vezes. Assim sendo, nossa questão é que as ferramentas de síntese de imagens possivelmente só executarão num segundo momento, e não no funcionamento do sistema junto com o artista.

Existem inúmeras alternativas para executar a síntese de imagens em separado, algumas das quais citamos a seguir.

- **Nova Execução com Todo o Sistema**

É a alternativa mais trivial. Após toda a animação ter sido gerada e testada, uma nova execução é feita, porém desta vez o exibidor para acompanhamento da animação é substituído por um sistema de síntese de imagens.

- **Geração de Arquivos de Cena**

Esta hipótese é interessante para permitir o uso de ferramentas de síntese de imagens que não estejam integradas ao ambiente. No lugar de um exibidor, é utilizado um sistema que simplesmente gera arquivos descrevendo cada cena da animação, na linguagem específica de um sistema de síntese de imagens. Tendo tais arquivos, a cena pode ser gerada de forma independente do ambiente.

- **Uso de um Sistema de Buffers**

Esta é a alternativa mais sofisticada e flexível, porém a de mais difícil implementação. É construída uma ferramenta que apenas armazena em arquivo as mensagens que lhe são enviadas e retransmite tais mensagens em seqüência numa execução em separado.

Esta ferramenta é executada no lugar de um exibidor, durante a geração da cena. Em outro momento, ela pode ser executada em conjunto com um sistema de exibição de imagens que faça parte do ambiente. A vantagem é que a animação não precisará ser gerada novamente, e pode-se usar em separado qualquer sistema de exibição que faça parte do sistema.

5. Implementação do Modelo

Nos capítulos anteriores foi apresentado um modelo para a integração de ferramentas de animação. O presente capítulo descreve uma implementação que foi desenvolvida a partir do modelo proposto.

5.1 Plataforma de Desenvolvimento

O Modelo foi proposto para ser o mais independente de plataforma possível. A mesma filosofia dirigiu os esforços de implementação, o que levou a implementação do sistema simultaneamente em duas plataformas distintas, a Silicon Graphics [SIL 91], e a SUN [SUN 90].

Por razões semelhantes adotou-se a linguagem C++ [MAR 90]. Por ser uma linguagem bem difundida, e existirem inclusive ferramentas que a traduzem automaticamente para C, ela pode ser utilizada em praticamente qualquer plataforma.

A segunda razão para sua escolha deve-se a linguagem estar se tornando o padrão de fato no que se refere a linguagens orientadas a objetos. Cabe salientar que o modelo permite que ferramentas escritas em outras linguagens sejam também integradas, desde que respeitem o protocolo de comunicação do modelo.

O compilador utilizado em ambos os ambiente foi o *gcc*, que era, no momento da implementação deste trabalho, o melhor compilador C++ disponível nos laboratórios da UFRGS, onde foi feita a implementação do modelo.

A comunicação entre os processos é implementada através do mecanismo de *sockets*. Não apenas é um mecanismo fornecido por diferentes plataformas, desde estações UNIX até plataformas Windows, como tem a vantagem de poder ser utilizado para comunicação entre processos executando em diferentes máquinas, o que possibilita uma eventual paralelização do ambiente no futuro.

5.2 A Biblioteca de Comunicação

O Modelo prevê uma biblioteca de comunicações que ofereça uma interface independente de plataforma aos demais elementos do sistema. Nesta implementação, esta biblioteca é implementada por uma classe, a classe *Mensagem*, cujos métodos são chamados pelos diferentes programas que constituem o sistema. O Anexo A-1 apresenta os métodos disponíveis na biblioteca.

Ao ser criada uma instância da classe *Mensagem*, esta recebe um identificador único de socket, que usará para estabelecer um canal de comunicação, possivelmente com outra instância da classe *Mensagem* que estará sendo simultaneamente criada em outro processo.

Quando os métodos de uma classe *Mensagem* são chamados, eles convertem os parâmetros recebidos para estruturas internas a classe. Quando é chamado um método de envio de mensagem, todas as informações recebidas até aquele momento são compactadas num único vetor de bytes, que é enviado através do socket.

A biblioteca, naturalmente, é responsável também pelo recebimento de mensagens. Quando uma mensagem é recebida, ela é convertida para estruturas internas de dados, podendo sua informação ser acessada através de métodos análogos aos utilizados para o envio de mensagens.

<OBS:IMG>

A classe Mensagem possui métodos para adicionar a mensagem informações sobre parâmetros dos tipos básicos *real*, *string* e *vetor*. Para tratar parâmetros mais complexos, eventualmente utilizados por determinadas ferramentas, são derivadas classes da classe *Mensagem*, caso melhor descrito na seção 5.4.

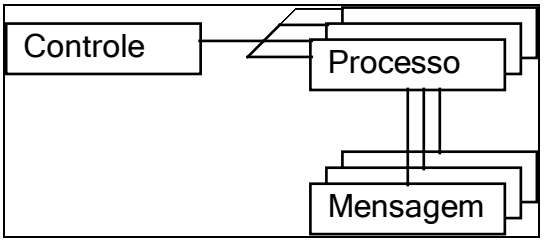

fig 5.1 - Classe principais do gerente

5.3 Implementação do Gerente

Conforme mostrado na figura 5.1, o *gerente* possui apenas três classes principais, a classe *Mensagem*, descrita anteriormente, a classe *Controle* e a classe *Processo*. As duas últimas classes são apresentas a seguir:

- **Controle**

A classe *Controle* pode ser descrita de forma sucinta como um grande distribuidor de mensagens, cuja atividade principal é percorrer várias estruturas de dados com informações sobre a cena e trocar mensagens com todas as ferramentas presentes no ambiente.

A classe permanece toda a execução do sistema num ciclo contínuo, percorrendo a sua lista de processos ativos no ambiente a cada instante de tempo que é gerado, recebendo e tratando as mensagens que os processos enviaram e enviando novas mensagens para os mesmos.

Sempre que um novo elemento é criado na cena, a classe *Controle* verifica se o processo que controlaria tal elemento já está ativo e, se não estiver, cria uma nova instância da classe *Processo*, e acrescenta na sua lista de processos ativos. A classe *Processo* é que se preocupará com questões de criação de processos e troca de mensagens.

- **Processo**

Toda instância da classe *processo* é criada com a informação da chamada de sistema necessária para ativar um determinado processo. Associada a cada instância da classe existe uma instância da classe *Mensagem*, que fará a comunicação com o processo.

Assim, na sua criação, a classe executa o processo que precisava ser criado e, através da classe *Mensagem*, cria um canal de comunicação. A classe *Mensagem* criará um *socket*, o qual estabelecerá este canal com a classe *Mensagem* do processo recém criado.

5.4 Implementação das Ferramentas

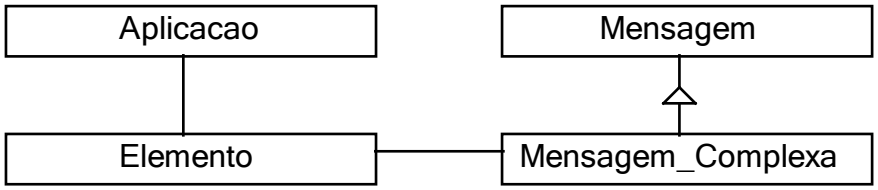

fig 5.2 – Classes Principais

O ambiente prevê a inclusão de qualquer ferramenta, construída segundo qualquer estrutura e em qualquer linguagem, desde que a ferramenta respeite o protocolo de comunicação do ambiente. Entretanto, junto com a implementação do ambiente, foi também implementada uma estrutura básica padrão para as ferramentas. Esta estrutura é gerada automaticamente pelo *Gerador de Ferramentas*, descrito na seção 5.5, sendo sua utilização na construção das ferramentas extremamente aconselhável, ainda que não essencial.

A figura 5.2 apresenta as classes principais que compõem o *framework* básico proposto para qualquer ferramenta integrada ao ambiente. A classe *Mensagem* já foi apresentada anteriormente, enquanto que as demais classes são descritas a seguir.

- **Mensagem_Complexa**

Esta classe derivada de *Mensagem* implementa o tratamento de todos os tipos de dados que são utilizados pela ferramenta e não pertencem aos tipos básicos. Por exemplo, suponha uma ferramenta que manipule cores, e tenha o parâmetro “cor” composto pelos valores “r”, “g” e “b”, todos valores inteiros. Neste caso, a classe Mensagem_Complexa desta ferramenta terá uma rotina para atribuir o valor de um parâmetro e outra para lê-lo, ambas recebendo como parâmetros três valores inteiros. Tais rotinas irão codificar as informações de “r”, “g” e “b” para o vetor de dados que é transmitido ao ambiente.

- **Aplicação**

A classe *Aplicação* implementa uma parte do *Nível de Controle de Mensagens*, descrito na seção 3.4.3. Esta classe possui uma instância da classe *Mensagem*, ou da classe *Mensagem_Complexa* se esta existir na ferramenta, e uma lista de instâncias da classe *Elemento*, ou de classes herdadas da mesma.

A classe faz a comunicação entre a *biblioteca de comunicação*, implementada na classe *Mensagem*, e os *algoritmos da ferramenta*, implementados na classe *Elemento* e nas suas classes derivadas.

Outra tarefa do *Nível de Controle de Mensagens* que a classe implementa é o controle de múltiplas instâncias, apresentado na seção 3.4.3.3. Essencialmente, esta tarefa se resume a criar uma nova instância dos algoritmos da ferramenta para cada novo elemento da cena que a ferramenta passe a controlar, e remover tal instância quando o elemento não for mais controlado pela ferramenta. A implementação desta tarefa é feita manipulando uma lista de instâncias da classe *Elemento*. Quando um novo elemento da cena é criado, a classe adiciona em sua lista uma nova instância da classe *Elemento*, e quando tal elemento é retirado da cena a instância é removida.

As demais tarefas do *Nível de Controle de Mensagens*, a requisição e a troca de informações com o ambiente, são implementadas tanto na classe *Aplicação* quanto na classe *Elemento*.

A classe *Aplicação* analisa todas as mensagens recebidas do ambiente, as quais estão normalmente associadas a determinado elemento da cena, e as retransmite para a instância da classe *Elemento* que controla tal elemento. De forma análoga, a classe também retransmite todas as informações das instâncias da classe *Elemento* para o ambiente.

- **Elemento**

A classe *Elemento* é responsável por implementar uma parte do *Nível de Controle de Mensagens* e por implementar os algoritmos particulares da ferramenta.

Cada instância da classe *Elemento* é responsável por controlar um determinado elemento, sendo criada no momento em que o elemento entra e destruída quando ele sai de

cena. Cada instância troca mensagens com o ambiente sobre o elemento que controla. Quando recebe uma requisição do valor de determinado parâmetro do elemento, a instância da classe lê o valor da sua estrutura de dados, e retorna uma mensagem, que é transmitida de volta ao ambiente pela classe *Aplicação*. Da mesma forma, quando recebe uma mensagem definindo o valor de determinado parâmetro, a instância atribui o valor correto na sua estrutura de dados.

Na classe Elemento é implementado, também, o algoritmo particular de cada ferramenta. Assim, em cada instância da classe Elemento existirá uma cópia de todas as variáveis necessárias ao algoritmo, com valores específicos do elemento de cena que aquela instância controla.

- **Classes Herdadas de Elemento**

Ainda que a maioria das ferramentas não tenha necessidade de implementar classes derivadas de *Elemento*, isto é necessário para ferramentas que tratam diferentes tipos de elementos.

Um exemplo desta situação são ferramentas de exibição de imagens. Mesmo os sistemas de exibição mais simples precisam lidar no mínimo com elementos sólidos (*atores*) e com pelo menos um elemento que observa a cena (*câmera*). Neste caso, é necessário implementar duas classes distintas, preferencialmente herdadas de uma mesma classe Elemento. Uma destas classes implementa o tratamento dos elementos do tipo *ator* enquanto a outra implementa o tratamento do elemento *câmera*.

Para a classe Aplicação é indiferente se existem ou não classes herdadas de *Elemento*, já que ela mantém uma lista de ponteiros para instâncias e repassa para estas as mensagens, não importando se estas instâncias são da classe *Elemento* ou são de uma classe herdada da mesma.

5.5 Gerador Automático de Ferramentas

O grande objetivo deste trabalho é reduzir o esforço envolvido na construção de uma nova ferramenta de animação. Para atingir este objetivo busca-se reaproveitar o

esforço feito anteriormente, permitindo que cada ferramenta tenha acesso aos recursos que outras ferramentas implementem. Existe uma questão importante associada a esta alternativa, que é o fato que ao mesmo tempo que reduzimos a complexidade da ferramenta oferecendo a mesma acesso aos recursos do ambiente, aumentamos sua complexidade com a necessidade de implementarmos protocolos de comunicação.

A solução ideal para resolvermos esta questão é retirar do usuário a tarefa de implementar manualmente os protocolos utilizados pelo ambiente. Com isto, atinge-se três objetivos importante: o usuário não precisa se dedicar a compreender e implementar os mecanismos de comunicação, que nada tem a ver diretamente com os algoritmos que ele busca implementar; o ambiente passa a ser mais confiável numa das suas áreas mais críticas, a comunicação entre os processos, reduzindo o risco de processos enviarem mensagens que não obedecem ao protocolo; o ambiente pode ser mais facilmente alterado ou transportado para outros ambiente, já que o código de comunicação pode ser gerado novamente.

Assim sendo, foi construída uma ferramenta que automaticamente gera grande parte do código das ferramentas, seguindo a estrutura de classes apresentada na seção anterior.

5.5.1 Funcionamento do Gerador de Ferramentas

O Gerador de Ferramentas interage com o usuário para obter determinadas informações sobre a ferramenta e, com base em tais informações, gera e compila um código fonte para um protótipo da ferramenta. Este protótipo é incluído na lista de ferramentas integradas ao ambiente e já pode ser utilizado numa animação, embora ainda não faça nenhuma tarefa significativa.

Todas as classes apresentadas anteriormente são geradas automaticamente. As classes Elemento e, se existirem, suas classes derivadas, não incluem o código do algoritmo particular de determinada ferramenta, mas todos os códigos que dizem respeito a comunicação com o ambiente são gerados.

Através de uma interface, o usuário define que elementos a ferramenta irá manipular e quais são os parâmetros dos mesmos. A interface permite ainda que seja definido o tipo dos valores que cada parâmetro armazena. Por exemplo, para gerar uma ferramenta de movimentos cinemáticos o usuário definiria que a ferramenta trabalha com elementos do tipo “ator”, os quais possuem parâmetros como “posição”, “velocidade” e “forma”, sendo a “velocidade” um vetor, composto por quatro valores reais (direção e velocidade), a “posição” três valores reais representando uma posição no espaço, etc.

Eventualmente uma ferramenta pode lidar com parâmetros cujos valores sejam relativamente complexos, como o exemplo das cores citado anteriormente. A interface permite a criação de novos tipos de parâmetros, cujos valores podem ser qualquer combinação de inteiros, reais e vetores de caracteres (*strings*). Havendo pelo menos um parâmetro que não é de nenhum tipo básico, será gerada automaticamente a classe Mensagem_Complexa, que tratará tais casos.

O usuário pode ainda definir que sua ferramenta tratará um ou vários tipos de elementos. Por exemplo, um sistema de cinemática trataria apenas de *atores*, mas já um sistema de exibição possivelmente trataria também de elementos como *luzes* e *câmeras*. Se a ferramenta trata mais de um tipo de elemento, automaticamente são geradas várias classes como herdadas de *Elemento*, e o usuário precisa apenas incluir o algoritmo de cada tipo de elemento numa classe diferente.

5.5.2 Interface para Utilização do Gerador

A figura 5.3 apresenta a tela inicia da interface. Nesta tela o usuário tem a possibilidade de associar elementos a sua ferramenta (botão “Cria Elemento”), salvar ou ler uma configuração pré-definida e gerar o código fonte da ferramenta.

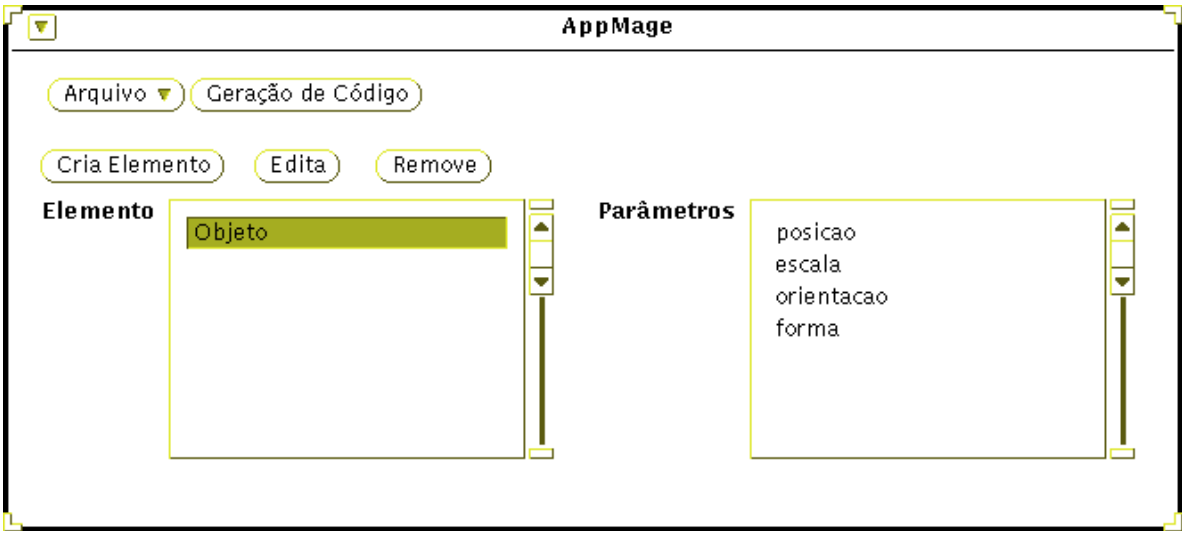

fig 5.3 - Janela Principal da Interface

Ao ser selecionado o botão de criação de elementos, é exibida a janela mostrada na figura 5.4. Nesta janela são exibidas duas listas, sendo que a lista esquerda apresenta os parâmetros que farão parte do elemento que está sendo definido, enquanto a lista da direita contém todos os parâmetros já definidos no ambiente. O usuário pode passar para a lista de elementos qualquer parâmetro já criado.

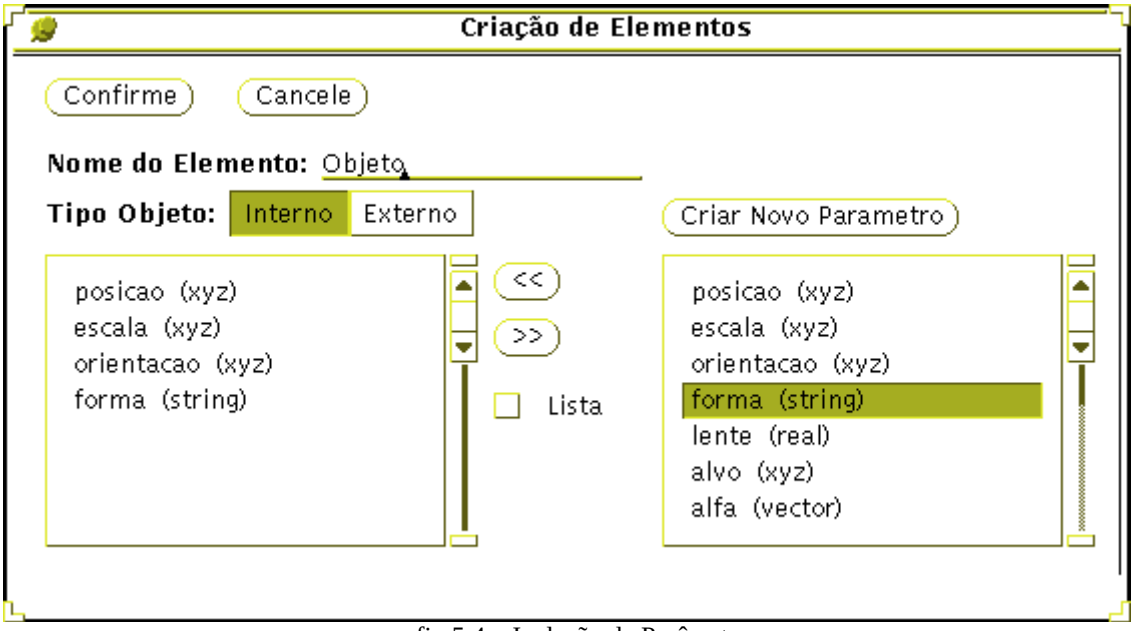

fig 5.4 – Inclusão de Parâmetros

Se os parâmetros do elemento não tiverem sido criados previamente, o botão “Criar Novo Parâmetro” exibe uma janela (figura 5.5) que oferece a opção de criar novos parâmetros.

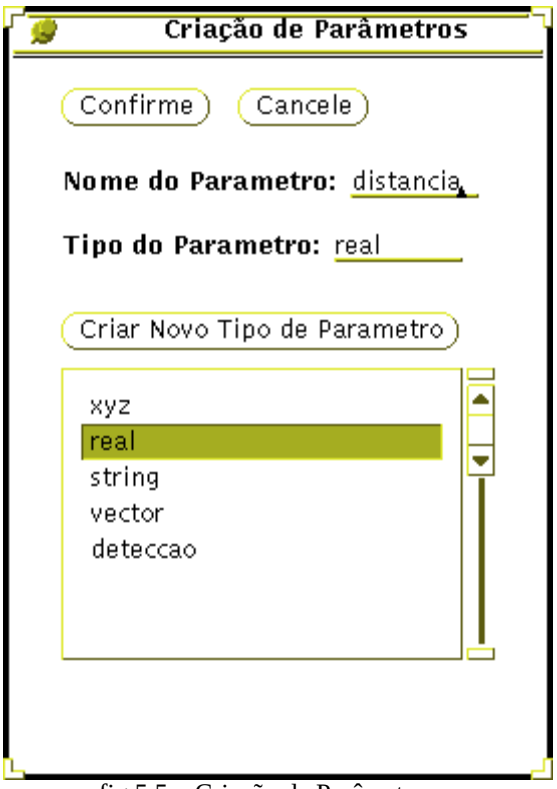

fig 5.5 – Criação de Parâmetros

Na janela de criação de parâmetros o usuário especifica o nome do novo parâmetro e tipo de valor associado ao mesmo. O tipo de valor pode ser um dos tipos básicos (“xyz”, “real” e “string”), como pode ser um novo tipo criado pelo próprio usuário.

A figura 5.6 exibe a janela para construção de novos tipos de valores. Um novo tipo de valor pode ser uma combinação de qualquer conjunto de números inteiros e vetores de caracteres.

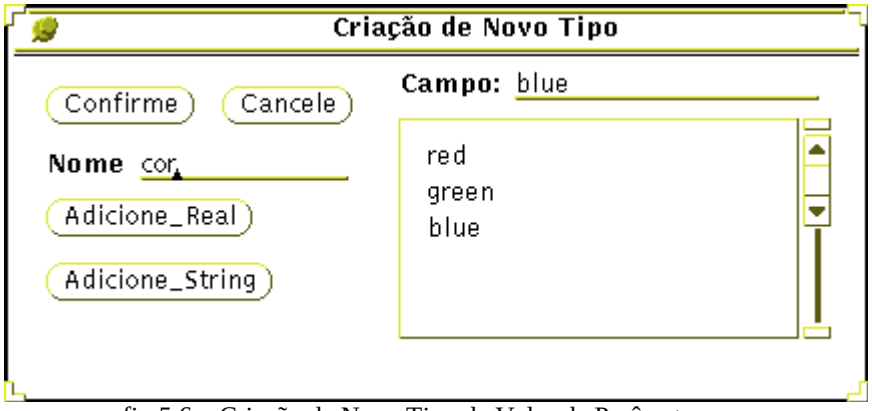

fig 5.6 – Criação de Novo Tipo de Valor de Parâmetro

5.5.3 Implementação do Gerador de Ferramentas

O *Gerador de Ferramentas* na verdade é composto por dois programas separados e relativamente independentes: a *Interface de Geração de Ferramentas* e o *Gerador de Código*.

A interface é um programa em linguagem “C”, construído com o uso do gerador de interfaces do XView[HEL 90] - *GUIDE* - que simplesmente gera um arquivo texto com todas as informações que o usuário colocou na interface. A linguagem “C” foi utilizada por ser a linguagem que o *GUIDE* utiliza. Após gerar o arquivo com as informações, a *Interface de Geração de Ferramentas* faz uma chamada de sistema para executar o *Gerador de Código*.

O *Gerador de Código* é na verdade um pré-processador, que lê uma série de arquivos num formato próprio semelhante ao “C++”, e gera os arquivos fontes da ferramenta a partir destes arquivos e das informações que a *Interface de Geração de Ferramentas* lhe transmite.

Uma vez que o *Gerador de Código* lê um arquivo para gerar cada código fonte, é possível fazer consideráveis alterações no código gerado, possivelmente incluindo gerar em outras linguagens que não “C++”, sem ser necessário alterar a programa em si.

A divisão da ferramenta em dois executáveis separados também tem uma razão de ser. O código da interface é completamente voltado para as estações de trabalho SUN, pois o *GUIDE* gera código compatível apenas com o padrão Xview. Já o *Gerador de Código* é um programa em ANSI C++, podendo ser compilado em qualquer máquina que tenha um compilador para a linguagem.

Assim sendo, transportar a ferramenta para outro ambiente, como a Silicon Graphics, implicaria apenas em criar uma nova interface para o usuário entrar com seus dados.

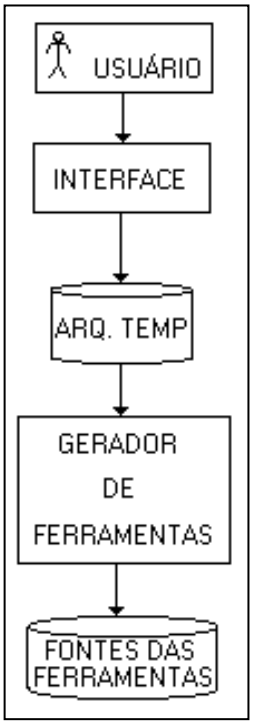

fig 5.7 – Gerador de Ferramentas

6 Validação do Ambiente

Nos capítulos anteriores deste trabalho foi apresentada uma proposta de integração de ferramentas de animação e descrita sua implementação. Neste capítulo nós faremos uma validação do modelo em função dos objetivos deste trabalho, visando provar que o modelo proposto de fato atende aos objetivos definidos inicialmente.

6.1 Um Protótipo de Sistema de Animação

Uma vez concluída a implementação do ambiente proposto neste trabalho, havia ainda a questão de provar que o ambiente efetivamente atendia os objetivos propostos. Foi desenvolvido um protótipo de um sistema mínimo de animação, utilizando os recursos do ambiente, com o objetivo de compará-lo com o conjunto de ferramentas de animação disponíveis, no momento, no Instituto de Informática da UFRGS.

Optou-se por construir um protótipo simples, e não um sistema de animação mais elaborado, por duas razões: primeiro, desenvolver um sistema completo de animação, incluindo movimentos dinâmicos, síntese de imagens, etc., envolve um custo que dificilmente permitiria seu desenvolvimento dentro dos prazos de um trabalho de mestrado; segundo, e mais importante, o sistema de animação não é a parte central do trabalho, mas apenas um elemento adicional para provar o funcionamento e eficiência do ambiente proposto, não sendo justificável concentrar uma parte mais significativa do projeto em seu desenvolvimento.

Seguindo a filosofia proposta no modelo, o sistema desenvolvido não é um único programa de animação, mas sim um conjunto de ferramentas, cada uma responsável por uma determinada atividade do processo de desenvolvimento de animações.

As seções seguintes descrevem cada ferramenta.

6.1.1 A Ferramenta Animação_Teste

Esta ferramenta na verdade é apenas um exemplo da utilização de linguagens de propósitos gerais na definição de animações. A ferramenta simplesmente envia mensagens de criação e movimentação de alguns objetos, que ficam girando em torno de um ponto no espaço.

Embora seja uma animação extremamente simples, ela já demonstra uma vantagens desta alternativa de especificação de movimentos, fazendo uso dos recursos de cálculos matemáticos da linguagem *C++* para gerar movimentos circulares.

6.1.2 O Leitor de Roteiros

O *Leitor de Roteiros* implementa uma outra alternativa de especificação por roteiro, o uso de uma linguagem específica de animação. A linguagem permite a definição de qualquer tipo de elemento como parte da cena, permite associar qualquer parâmetro ao elemento e qualquer tipo de valor ao parâmetro. Assim, mantém-se a flexibilidade de integrar diferentes ferramentas ao sistema.

O *Leitor* faz uso da técnica de quadros-chaves e interpolação para geração dos quadros intermediários. Uma vez que pode-se interpolar qualquer parâmetro que possua valores inteiros ou reais, tem-se uma flexibilidade muito grande de gerar animações com base em mudanças contínuas de qualquer parâmetro de qualquer elemento da cena.

O *Leitor* fornece ainda o recurso de ativar uma seqüência específica de comandos quando recebe uma determinada mensagem, recurso este utilizado para permitir o uso da ferramenta de *detecção de proximidade*, descrita na seção 6.1.5. O anexo A-4 apresenta um exemplo do uso deste recurso do *Leitor*.

Normalmente, numa execução do sistema em que seja lido um roteiro, a ferramenta *Animação_Teste* não seria chamada, porém não há nada que impeça isto. Assim, o protótipo de sistema possibilita inclusive a combinação de roteiros com o uso de linguagens de propósitos gerais.

6.1.3 A Ferramenta de Cinemática Direta

Esta ferramenta bastante simples permite o movimento de atores da cena com base na especificação de suas velocidades e acelerações.

Trata-se de um exemplo de ferramenta de geração de movimentos. Embora outras metodologias mais sofisticadas, como análise dinâmica e movimentação de corpos articulados, envolvam a execução de métodos numéricos muito mais caros computacionalmente, em termos do ambiente existe pouca diferença entre tais alternativas e a ferramenta de cinemática construída.

6.1.4 A Ferramenta de Exibição de Imagens

Na verdade foram implementadas duas ferramentas de exibição de imagens, uma em estações SUN e outra em Silicon Graphics, já que não existe uma compatibilidade entre as bibliotecas gráficas das duas plataformas.

As duas ferramentas implementam recursos bem simples de exibição: a ferramenta em SUN exibe imagens em aramado (*wire-frame*), como mostrado na figura a seguir. Já a ferramenta em Silicon Graphics faz uso da biblioteca OpenGL para exibir as imagens com um maior realismo.

6.1.5 O Detector de Proximidade

Esta ferramenta verifica, a cada instante de tempo, a distância entre o centro geométrico de todos os atores presentes na cena. Quando dois atores encontram-se a uma distância menor que determinado valor limite, especificado no roteiro, a ferramenta envia uma mensagem para o *Leitor de Roteiros*, que então executa um tratamento específico.

Na prática, esta ferramenta tem uma utilidade bem restrita na geração de uma animação, porém ela serve o importante propósito de testar o uso de ferramentas que atuam sobre a cena como um todo, como, por exemplo, sistemas de detecção de colisões.

Para o ambiente, um sistema de detecção de colisões nada mais é que uma ferramenta equivalente ao *Detetor de Proximidades,* que exige alguns parâmetros a mais e executa cálculos internos mais sofisticados.

6.2 Etapas da Geração de Animações

Os recursos oferecidos pelo protótipo desenvolvido neste trabalho de forma alguma se comparam ao total de recursos oferecidos por todas as ferramentas desenvolvidas e utilizadas pelo grupo de computação gráfica da UFRGS. Entretanto, uma análise realística do modelo pode ser feita enfocando-se as diferenças gerais de funcionamento e utilização dos dois ambientes.

Como base para uma análise dos recursos oferecidos pelo sistema, é apresentada, a seguir, uma comparação entre o protótipo e as ferramentas disponíveis na UFRGS, em termos de suas utilizações na construção de uma animação.

- **Geração de animações utilizando o protótipo desenvolvido**

No momento que se inicia o desenvolvimento de uma animação, fazendo uso do protótipo, a primeira decisão de projeto a ser tomada é optar pelo uso do Leitor de Roteiros ou pelo desenvolvimento de um programa C++ que gere a cena. Esta decisão dependerá da complexidade da manipulação dos atores, já que o Leitor permite apenas interpolações lineares para controlar os parâmetros, enquanto que um programa não oferece virtualmente nenhuma restrição.

Independente da decisão tomada, o segundo passo é identificar que ferramentas estão disponíveis no ambiente e quais os parâmetros que as mesmas manipulam. Para isso pode ser utilizado o mesmo *gerador de ferramentas* descrito no capítulo 5, que apresenta uma lista de todas as ferramentas do sistema e os parâmetros associados as mesmas. Infelizmente esta alternativa não é suficiente por si própria, pois embora apresente ao usuário a lista de parâmetros de cada ferramenta, não fornece nenhuma outra indicação do significado dos parâmetros e dos algoritmos aplicados aos mesmos pelas ferramentas.

Futuras versões do *gerador de ferramentas* poderiam conter textos explicativos incluídos na fase de criação das ferramentas.

O próximo passo, uma vez conhecendo-se os recursos disponíveis no sistema, é construir a animação. Uma alternativa é construir um código fonte, com o auxílio do *gerador de ferramentas*, que se encarregará de gerar todo o código relativo a comunicação com o ambiente. A outra alternativa, mais simples, é simplesmente descrever a animação utilizando-se a linguagem aceita pelo Leitor de Roteiros.

Especificada a animação, executa-se o programa *Gerente*, indicando por parâmetro como está especificada a animação (anexo A-2), que chamará as ferramentas necessárias. Nesta versão o resultado é exibido na forma de aramado (em SUN) e *flat-shading* (em Silicon Graphics).

Cabe salientar que não é gerado nenhum arquivo descrevendo a cena, não permitindo a geração de imagens foto-realísticas da cena. Lembrem-se, é apenas um protótipo.

- **Geração de Animações utilizando as Ferramentas Desenvolvidas pelo CG**

Inicialmente faz-se necessário identificar que ferramentas de geração de movimentos estão disponíveis, e quais destas serão efetivamente utilizadas para criar determinada animação.

O segundo passo é gerar a animação dos atores associadas a cada ferramenta. Esta é possivelmente a fase mais problemática, já que se faz necessário conhecer a linguagem de entrada de cada uma das ferramentas a serem utilizadas. Executando-as teremos disponíveis arquivos contendo a posição e, eventualmente, orientação de todos os atores da cena.

O passo seguinte é integrar todos os atores num mesmo cenário. Isto é feito atualmente utilizando-se a ferramenta ANIMAKER. Um roteiro é escrito, na linguagem própria da ferramenta, identificando os arquivos gerados e como estão associados a cada

ator da cena. Os recursos de interpolação da ferramenta permitem especificar diretamente atores que tenham movimentos mais simples.

Executando-se a ferramenta ANIMAKER, pode-se observar a animação na forma de aramado (ou *flat-shading* na versão IRIS-ANIMAKER[COH 93], desenvolvida em Silicon Graphics). Além disso, a ferramenta também gera arquivos descrevendo a cena, que, passando por uma ferramenta de tradução, servem de entrada para o sistema de síntese de imagens POV-RAY.

6.3 Objetivos Atingidos pelo Protótipo

Nas seções seguintes o protótipo é analisado em função dos objetivos deste trabalho.

6.3.1 Otimização de Recursos

O primeiro objetivo de todo este trabalho é a otimização dos recursos na construção de ferramentas de animação, conforme descrito na seção 1.4.1. As seções seguintes analisam em que aspectos o protótipo desenvolvido utiliza de forma mais otimizada os recursos disponíveis.

6.3.1.1 Interação com o Usuário

Nas ferramentas desenvolvidas pelo Grupo de Computação Gráfica (CG) da UFRGS, a interação com o usuário é de responsabilidade de cada ferramenta. Assim, o sistema de dinâmica CLIC[LEM 92] possui seu código particular para ler de arquivo as informações sobre seus objetos dinâmicos, o sistema ANIMAKER[SIL 92] define uma linguagem própria para especificar a cena, e todos os demais seguem a mesma filosofia.

No protótipo de sistema desenvolvido, a interação com o usuário é feita através de um único programa, o *Leitor de Roteiros*, que é flexível o bastante para tratar as informações necessárias a qualquer ferramenta que faça parte do sistema. As principais

vantagens desta abordagem, principalmente em se tratando de otimizar recursos, são citadas a seguir:

- **Uma ferramenta não precisa implementar seu próprio mecanismo de interação com os usuários.**

Uma vez que sistemas como PREVIEW[SCH 92a] e ANIMAKER são voltados para a interação com os usuários visando construir uma animação, é perfeitamente compreensível que uma parte do trabalho de desenvolvimento fosse direcionado ao desenvolvimento de mecanismos de interação. Porém, em trabalhos direcionados para a geração de movimentos (trabalhos em análise dinâmica, corpos articulados, etc.) a ênfase é no estudo dos algoritmos relacionados a estes movimentos. Para estes trabalhos, os mecanismos de interação com os usuários não são questões relevantes, e só foram implementados mecanismos de leitura de arquivos para os protótipos desenvolvidos por ser a única forma viável de testar os citados protótipos.

Se estes trabalhos fossem desenvolvidos dentro do ambiente proposto neste trabalho, a implementação de tais recursos não se faria necessária, já que no próprio protótipo desenvolvido já está presente um *Leitor de Roteiros* capaz de atender as necessidades de teste e uso dos protótipos que fossem desenvolvidos.

- **Todos os recursos de interação estão disponíveis a todas as ferramentas:**

Para termos clara esta questão, vamos considerar um caso real: o sistema de dinâmica CLIC permite que o usuário especifique as forças e torques que atuam sobre os objetos apenas no primeiro instante de tempo da cena. Os algoritmos em si poderiam tratar forças aplicadas em outros instantes de tempo, porém o roteiro não trata tais situações. O sistema ANIMAKER, mais recente, possui um roteiro bem mais sofisticado, permitindo interpolar valores de parâmetros, possibilitando inclusive uma variação contínua ao longo do tempo.

Assim o sistema ANIMAKER possui um código capaz, em teoria, de lidar com parâmetros de *força* que, por exemplo, aumentam de intensidade ao longo do tempo. Ainda assim, é impossível aplicar, na ferramenta CLIC, uma força que varie sua intensidade ao

longo do tempo, pelo simples motivo que o CLIC não tem como ter acesso aos recursos implementados internamente a ferramenta ANIMAKER.

Já no protótipo desenvolvido neste trabalho, a situação é completamente diferente. Existe um único *Leitor de Roteiros*, que fornece informações a todas as demais ferramentas. Supondo que o sistema CLIC fosse transportado para dentro deste ambiente, automaticamente seria possível criar variações na força aplicada aos atores dinâmicos, pois a força seria apenas mais um parâmetro do roteiro. Assim, sendo implementados novos recursos no roteiro, por exemplo formas mais sofisticadas de interpolações, todas as ferramentas do ambiente passariam a ter acesso a tais recursos.

6.3.1.2 Exibição das Imagens da Animação

No conjunto de ferramentas desenvolvidas pelo grupo de computação gráfica da UFRGS, duas ferramentas, PREVIEW e, posteriormente, ANIMAKER, tratam da questão da exibição de imagens simples da animação. Ambas recebem vários arquivos, gerados pelas demais ferramentas, e exibem uma seqüência de quadros em aramado.

No protótipo de sistema desenvolvido a ferramenta de exibição da cena está ligada a todo o ambiente, exibindo as imagens a medida que as mesmas são geradas.

Assim, a principal diferença entre estas alternativas é que o protótipo exibe as imagens durante a execução das ferramentas, enquanto no primeiro caso o mesmo processo é feito em duas etapas: primeiro a cena é gerada, e depois é exibida.

Esta diferença se torna mais visível quando o animador está gerando um determinado quadro da cena. Usando ferramentas isoladas faz-se necessário gerar um novo arquivo para cada teste e executar as ferramentas envolvidas na geração daquele determinado quadro. No protótipo toda a comunicação entre as ferramentas é transparente para o animador, que simplesmente gera novamente aquele quadro.

6.3.2 Interação entre Ferramentas

O segundo objetivo deste trabalho é o de possibilitar que diferentes ferramentas interajam entre si, permitindo que os atores que as mesmas controlem possam atuar de uma forma conjunta numa cena.

Para ilustrar a interação entre ferramentas, um dos elementos do protótipo é o *Detector de Proximidade*, uma ferramenta que analisa toda a cena a cada instante de tempo e gera mensagens em função da distância entre os objetos presentes na cena, mensagens estas que podem afetar a movimentação dos atores nos instantes subseqüentes da animação. O anexo A-4 apresenta um exemplo de animação com o uso do *Detector de Proximidade*.

Naturalmente esta ferramenta não foi construída com o propósito exclusivo de adicionar novos recursos ao protótipo, mas sim primordialmente para demonstrar as possibilidades de interação que o ambiente oferece aos sistemas que sejam construídos sobre o mesmo. Um sistema completo construído sobre o ambiente aproveitaria esta possibilidade para o desenvolvimento de recursos mais úteis, como o desenvolvimento de um sistema único de detecção de colisões, por exemplo.

Para compararmos este recurso do protótipo com as ferramentas do grupo de CG, temos que analisar o custo de implementar um mecanismo semelhante em ferramentas que executam de forma isolada.

A detecção da distância entre os atores, ou mesmo a detecção de uma colisão, por si só, não é uma tarefa inviável, seguindo a filosofia de integração adotada nas ferramentas do grupo de CG. Bastaria tal ferramenta ler a mesma entrada do sistema ANIMAKER, e, comparando as informações passadas por cada ferramenta de geração de movimentos, fazer as comparações necessárias. O problema é a realimentação das ferramentas.

Suponha que num dos primeiros quadros ocorreu alguma situação que cause uma mudança na cena. Esta informação tem que ser repassada as ferramentas (o que possivelmente implicaria em modificações nos arquivos de entrada de todas as ferramentas cujos atores tiveram seus movimentos afetados) e a animação tem que ser gerada

novamente a partir daquele quadro. Após, novamente o ferramenta de detecção precisará analisar as informações geradas, numa seqüência de iterações até todas as situações especiais, como colisões, sejam detectadas. É fácil perceber que está alternativa é virtualmente inviável na geração de uma animação mais sofisticada.

6.3.3 Interface Única para o Usuário

Uma das principais dificuldades na utilização de ferramentas isoladas para gerar uma animação é o custo de compreender o funcionamento e a interface de cada uma das ferramentas.

Na alternativa implementada no protótipo apenas uma ferramenta se comunica com o animador, que não precisa interagir diretamente com as demais ferramentas do ambiente.

O animador, assim, ignora todo o funcionamento interno do sistema. Ele não precisa saber que ferramentas são necessárias para gerar determinado movimento, qual o procedimento para chamá-las, que arquivos são necessários as mesmas e todos os demais detalhes semelhantes.

A única informação necessária a um usuário do protótipo do sistema é saber que parâmetros cada tipo diferente de ator necessita para executar seus movimentos. Ele precisa saber, por exemplo, que um ator dinâmico necessita de informações de forças e torques, que um ator cinemático exige valores de velocidade e aceleração, etc.

Um sistema completo desenvolvido sobre o ambiente poderia inclusive fornecer ao usuário facilidades maiores de identificar os parâmetros associados a cada ator, por exemplo oferecendo uma interface para a construção dos roteiros na qual sejam enumerados os tipos possíveis de atores e parâmetros manipulados pelo sistema.

7 Conclusões

Neste trabalho foi mostrada uma alternativa de integração de ferramentas de animação, fazendo uso de um ambiente que distribui mensagens entre processos. O ambinte foi utilizado na construção de um protótipo de sistema de animação, o qual foi comparado com ferramentas desenvolvidas isoladamente, disponíveis na UFRGS.

Este capítulo encerra o presente trabalho apresentando as conclusões obtidas e propondo alternativas de trabalhos futuros.

7.1 Questões Respondidas

Quando este trabalho foi iniciado, haviam várias questões que precisavam ser respondidas, para as quais a mera pesquisa na literatura da área não nos fornecia uma resposta adequada. Apenas a construção do ambiente e sua validação, através da implementação de um protótipo de sistema, forneceram as informações necessárias para responder tais questões.

7.1.1 Construção de um Ambiente Aberto.

Sem dúvida, a principal questão associada a este trabalho é a resolução do problema de construir um sistema capaz de funcionar corretamente mesmo com ferramentas de animação que venham a ser desenvolvidos. Em outras palavras, o problema é tratar novas ferramentas que tenham novos recursos, que antes não faziam parte do sistema. Por exemplo, se um novo sistema de exibição de imagens é capaz de exibir objetos com transparência, como um ambiente que nunca havia tratado tal informação pode oferecer ao usuário a possibilidade de definir a transparência dos atores de uma animação?

A solução normalmente adotada na construção de um ambiente de animação é definir um conjunto restrito de elementos e parâmetros que o sistema tratará. Por exemplo, determinado sistema pode tratar apenas objetos dinâmicos. Integrar um sistema de

movimento de partículas, neste caso, só será possível quando o sistema for ampliado para tratar tal recurso.

Este trabalho buscou uma alternativa que não envolvesse a modificação do ambiente em si a cada integração de uma nova ferramenta. Alterar as estruturas de mensagens ou o sistema que controla as ferramentas, o *Gerente*, vai contra os objetivos deste trabalho, de tornar o mais fácil possível a tarefa de integração de ferramentas.

A solução para este problema só começou a surgir quando colocamos a questão da seguinte forma: para poder se comunicar de forma perfeita com uma nova ferramenta, o sistema precisa conhecer os recursos da mesma. A princípio, isto parece significar que o ambiente de integração, ou seja, o núcleo do sistema, precisa conhecer tal ferramenta, o que implicaria inevitavelmente em constantes alterações no mesmo. A principal conclusão que nos permitiu solucionar o problema foi perceber que, embora o sistema como um todo precise conhecer a nova ferramenta, este conhecimento pode não estar no núcleo, mas sim nas outras ferramentas do sistema.

A solução encontrada foi, portanto, de delegar a responsabilidade de tratar cada ferramenta para as demais ferramentas presentes no sistema. Por exemplo, havendo um sistema de síntese de imagens capaz de exibir objetos transparentes, não é necessário que o núcleo do sistema conheça o conceito de transparência. Basta que alguma das ferramentas conheça e transmita para o sistema de síntese de imagens esta informação.

A estruturação do ambiente para viabilizar esta alternativa foi, possivelmente, uma das etapas mais difíceis e complexas do presente trabalho. Toda a estrutura de comunicação foi especialmente definida visando permitir a existência de um único leitor genérico, independente de qualquer ferramenta porém capaz de se comunicar com todas elas e lhes transmitir qualquer informação necessária. Da mesma forma, a estrutura de comunicação permite que qualquer ferramenta troque mensagens com qualquer outra, sem que o núcleo do ambiente precise tomar conhecimento do conteúdo das informações transmitidas.

7.1.2 Compartilhamento de uma Interface Única

No início da definição do presente trabalho, haviam poucas dúvidas sobre a viabilidade de integrar ferramentas de geração de movimentos entre si e com sistemas de síntese de imagens. Embora limitassem a integração apenas a troca de arquivos, as ferramentas PREVIEW e ANIMAKER já forneciam alguma forma de integração semelhante a pretendida.

O mesmo, entretanto, não ocorre em relação a integração destas ferramentas com um sistema de interação com os usuários. Nem nos trabalhos já desenvolvidos na UFRGS nem na literatura especializada são mencionados estudos visando utilizar uma única ferramenta para a leitura e interpretação de roteiros de animação.

Este trabalho, particularmente com a implementação de um protótipo de sistema, provou que é possível isolar a tarefa de interação com o usuário numa única ferramenta. Para tanto é necessário um protocolo sofisticado de comunicação e o uso de algum mecanismo eficiente de troca de mensagens.

7.1.3 Uso em Conjunto de Diferentes Técnicas de Animação

Existem, na literatura, estudos sobre o uso em conjunto de duas ou três técnicas no controle de um mesmo ator. Em [BOU 92], por exemplo, algoritmos de dinâmica e cinemática são utilizados em conjunto para controlar o movimento de um corpo articulado. Entretanto, neste trabalho buscou-se uma forma diferente de integração, permitindo que diferentes atores que compartilham um mesmo cenário pudessem ser controlados por técnicas diferentes de animação. Este enfoque de integração é muito pouco tratado na literatura.

Com o desenvolvimento do ambiente e do protótipo de sistema de animação, pode-se comprovar que é efetivamente possível compartilhar uma cena entre diferentes atores usando diferentes técnicas. Para tanto, é necessário oferecer as ferramentas a flexibilidade suficiente para tratarem cada situação de interação com outros atores da forma

que lhes for mais conveniente, como apresentado na seção 4.2.2, que discutiu esta situação em bem mais detalhes.

Por exemplo, supondo que um ator que obedece leis dinâmicas e outro que segue leis cinemáticas colidam numa animação. Oferecendo a cada ferramenta a possibilidade de tratar a colisão de uma forma particular, pode-se viabilizar uma maior interação entre os atores. Um ator dinâmico pode receber uma força na direção oposta ao ponto de colisão, enquanto um ator cinemático pode ter sua velocidade alterada.

7.2 Flexibilidade do Ambiente

Durante as fases finais do desenvolvimento deste trabalho, surgiu a questão do grau de flexibilidade que o ambiente havia atingido. O modelo foi construído visando permitir a integração de qualquer ferramenta de animação, independente das tarefas executadas pelas mesmas. Assim sendo, não seria possível utilizar o ambiente para construir sistemas integrados voltados para outras atividades que não a geração de animações?

A rigor, a resposta é sim. Uma vez que o ambiente não restringe de nenhuma forma as tarefas que as ferramentas podem executar, não há nada que impeça que qualquer ferramenta seja integrada ao ambiente, desde que obedeça ao protocolo de comunicação do modelo.

Na prática, porém, utilizar o ambiente na construção de sistemas que não sejam de animação não é uma alternativa interessante, embora seja tecnicamente possível. A questão essencial é que todos os recursos de integração e todos os protocolos de comunicação foram construídos visando integrar ferramentas de animação genéricas, executando de forma sincronizada.

Assim, as vantagens de utilizar o ambiente só são significativas quando as ferramentas integradas fazem uso dos recursos oferecidos, e tais recursos foram desenvolvidos especificamente para a tarefa de geração de animações. Utilizar o sistema com outro fim, ainda que possível, dificilmente seria vantajoso. Naturalmente isto não

descarta hipóteses como a ampliação do ambiente para permitir a integração de ferramentas de simulação, por exemplo.

7.3 Trabalhos Futuros

Uma vez que o presente trabalho é essencialmente um ambiente para a construção de sistemas de animação, naturalmente existem inúmeros projetos que poderiam dar continuidade a este, relacionados com o desenvolvimento de ferramentas de animação integradas ao ambiente. Na verdade, dada a flexibilidade do ambiente, virtualmente qualquer ferramenta de animação poderia ser integrada ao mesmo.

Nesta seção o enfoque será em alterações que afetem a estrutura do ambiente em si. Os exemplos apresentados são alternativas que não foram incluídas no presente trabalho devido ao elevado custo de desenvolvimento das mesmas.

7.3.1 Interface Orientada a Artistas

Um dos aspectos mais fascinantes e complexos da proposta aqui apresentada de integração de ferramentas é a possibilidade de desenvolver uma interface orientada a artistas capaz de fornecer ao usuário acesso a todas as ferramentas do sistema.

Durante as fases inicias do presente trabalho, a hipótese de desenvolver tal ferramenta foi analisada, porém considerada inviável, principalmente devido aos custos extremamente elevados de desenvolvimento. Nas seções a seguir é apresentada uma proposta de ferramenta orientada a artista e os principais passos necessários a construção da mesma, com base nas considerações feitas ao longo do desenvolvimento do presente trabalho.

7.3.1.1 Funcionamento Geral da Ferramenta

Uma ferramenta orientada a artistas permite que o usuário edite diretamente o movimento dos atores, indicando de forma intuitiva as ações em qualquer instante de tempo. Exemplos de sistemas orientados a artistas podem ser vistos em [BRU 90], [PHI 90]

e [STU 90]. Sistemas comerciais de animação, mesmo os mais simples, são geralmente orientados a artistas, já que o uso de especificação por roteiro exige um conhecimento maior de programação por parte dos usuários do sistema.

Construir uma ferramenta orientada a artistas não é uma tarefa trivial. O custo é elevado, principalmente devido a complexidade da edição direta dos elementos da cena. Mesmo para um sistema simples, utilizando interpolação de quadros-chaves ou cinemática, uma interface para especificar vetores de velocidades, ou posição de atores em diferentes quadros, envolve um custo de implementação bem considerável.

Construir uma interface destas seguindo a filosofia do ambiente é ainda mais caro e complexo, pois diferentes técnicas de geração de movimentos exigem diferentes formas de interação com o usuário.

7.3.1.2 Integração com Múltiplas Ferramentas

O principal problema de construir uma interface orientada a artistas é tratar com diversas ferramentas. Se, na construção da interface, todas as ferramentas presentes no ambiente forem conhecidas, basta definir recursos de interação compatíveis com todas elas. O problema é que esta alternativa foge da filosofia de funcionamento proposta, na qual novas ferramentas podem ser incluídas no ambiente a qualquer momento.

O problema, portanto, é fornecer recursos de interação que sejam abrangentes o suficiente para funcionarem corretamente com todas as ferramentas que estejam presentes no ambiente. Observando-se na literatura [STU 90] e [PHI 90] a variedade de mecanismos de interação com usuários associados às diferentes técnicas de animação, percebe-se que o problema é relativamente complexo. Duas alternativas são apresentadas a seguir que poderiam representar soluções para o problema.

- **Recursos de interação baseados nos tipos dos parâmetros tratados:**

Esta alternativa se baseia na forma como as ferramentas mantém informações sobre os elementos da cena. Conforme definido no modelo, cada elemento da cena

controlado por uma ferramenta é composto por parâmetros, os quais tem um tipo de valor associado.

A forma mais simples de oferecer ao usuário uma interface para edição de animações é construí-la voltada para os tipos de parâmetros dos atores. Neste caso seriam construídos recursos de interação para controlar vetores, outros para controlar posições no espaço tridimensional, outros ainda para especificar valores escalares, etc. Alguma forma, mesmo que simples, precisaria existir para especificar valores de tipos quaisquer, construídos especificamente para uma ferramenta.

Esta solução está longe de ser a ideal. Nesta solução, a especificação do vetor de velocidade de um ator cinemático e a especificação da força atuando sobre um ator dinâmico seriam feitas de forma exatamente idêntica. Isto pode ser interessante para certos casos, mas surgirão inúmeras situações onde o tratamento genérico não se mostra o mais adequado.

Por outro lado, esta técnica tem a grande vantegem de não ter um custo de implementação demasiado alto, além de ser uma alternativa muito melhor para o usuário do que a mera especificação através de roteiros.

- **Recursos de interação definidos pelas próprias ferramentas**

O protocolo OLE, apresentado rapidamente no capítulo 2, propõe uma técnica bem interessante de integração de ferramentas, permitindo que uma ferramenta atue diretamente sobre a interface de outra, por exemplo abrindo uma janela sua dentro das janelas da outra ferramenta.

Uma estratégia semelhante poderia ser utilizada na construção de uma interface orientada a artistas associada ao ambiente. Nesta alternativa cada ferramenta implementaria o código de tratamento de seus parâmetros, e a interface gerenciaria a execução destes códigos.

Idealmente, esta alternativa estaria associada a alguma organização das ferramentas que permitisse que novas ferramentas se aproveitassem dos recursos de

interação já implementados. Assim, uma nova ferramenta precisaria apenas implementar recursos de interação com o usuário quando não houvessem disponíveis recursos semelhantes.

Possivelmente, a implementação de um protocolo robusto o suficiente para possibilitar estes recursos é um trabalho bem maior que o aqui apresentado, e envolveria alguns anos de estudos e implementação. Por outro lado, tal recurso permitiria a construção de sistemas de animação particularmente poderosos e robustos, graças ao reaproveitamento dos recursos de interação incluídos nas ferramentas.

7.3.1.3 Sistemas de Buffers

O funcionamento da interface orientada a artistas de forma integrada com diferentes ferramentas de animação é o principal problema associado ao seu desenvolvimento, mas não é o único. Outro problema está associado à geração de forma seqüencial dos quadros da animação.

Como já foi dito, o sistema gera uma animação qualquer de forma seqüencial, gerando um instante de tempo *t* apenas se o instante *t-1* já foi gerado. Esta restrição é bem mais significativa quando se faz uso de uma ferramenta de edição direta de animações. Neste caso, é muitas vezes necessário observar e editar um quadro qualquer da cena, o que é extremamente ineficiente, já que constantemente será gerada toda a cena para gerar e exibir este único quadro.

Para solucionar este problema, faz-se necessária a existência de um sistema de *buffers*, que mantenha armazenadas todas as informações geradas pela animação. Assim, se um determinado quadro precisar ser exibido, por exemplo, o ambiente buscará diretamente no sistema de *buffers* a informação sobre o quadro, ao invés de gerar toda a animação novamente.

Naturalmente, o problema não é tão simples. A cada edição de um parâmetro de um elemento qualquer da cena, todos os quadros a partir do quadro editado não estão mais garantidamente corretos, e é bem difícil prever que outros elementos da cena podem ter

sido afetados. Com isto, a menos que seja usado um algoritmo extremamente sofisticado, cada edição de algum quadro resultará na invalidação do buffer para todos os quadros subseqüentes.

7.3.2 Execução Paralela das Ferramentas

Uma ampliação óbvia do sistema é a execução do mesmo em várias máquinas de forma simultânea. A própria estrutura do modelo, que executa cada ferramenta num processo em separado, facilita esta tarefa. Nada impede, por exemplo, que as ferramentas sejam executadas em máquinas diferentes de uma rede, bastando para isso modificar algumas chamadas na biblioteca de comunicação para estabelecer a comunicação dos *sockets*.

Fazer uma versão paralela do sistema que seja eficiente, porém, é um problema completamente diferente, envolvendo várias questões relacionadas ao fluxo de mensagens e à distribuição de cargas entre máquinas. As seções seguintes apresentam algumas destas questões, enfatizando as mudanças necessárias para adaptar o ambiente para a execução paralela.

7.3.2.1 Execução Sincronizada

A versão atual do ambiente trabalha de forma sincronizada: cada ferramenta só inicia a geração de determinado quadro após todas as ferramentas terem terminado de gerar o quadro anterior. Numa máquina com um único processador isto não é problemático, já que o processador estará todo o tempo ocupado executando alguma das ferramentas. Numa execução paralela, porém, esta questão se mostra bem mais relevante, já que algumas máquinas eventualmente ficarão esperando caso uma ferramenta qualquer demore mais que as outras para gerar uma cena.

Existem algumas alternativas para resolver ou minimizar este problema, das quais duas são citadas a seguir:

- **Distribuição Dinâmica da Carga:**

Ao longo da execução de uma animação qualquer, o ambiente analisa o tempo gasto por cada ferramenta para gerar cada quadro, e reorganiza os processos de uma forma cada vez mais eficiente.

Por exemplo, se as ferramentas que estiverem gerando determinado quadro numa máquina demorarem bem mais tempo que as outras ferramentas, no próximo quadro, o sistema poderá diminuir o número de processos naquela máquina, executando algumas das ferramentas nas máquinas menos sobrecarregadas.

- **Geração Não Sincronizada**

Nesta alternativa, ao invés de todas as ferramentas executarem de forma sincronizada, elas executam de forma independente e contínua, sem se preocuparem se estão avançando na animação mais rapidamente que as demais ferramentas.

Quando um evento ocorre em algum ponto da animação, por exemplo, uma colisão, qualquer dos elementos afetados que já tenha gerado quadros após o evento invalida-os, e recomeça a geração a partir do ponto onde recebeu alguma mensagem relativa a tal evento.

Num ambiente mono-processado, esta técnica só causa uma maior perda de tempo, fazendo com que o processador desperdice tempo gerando quadros que serão invalidados. Num ambiente multi-processado, por outro lado, esta alternativa pode ser vantajosa, evitando que todas as ferramentas tenham que executar na velocidade da mais lenta.

7.3.2.2 Ambiente Centralizado

Na versão atual do ambiente, toda a comunicação entre as ferramentas é feita através do *Gerente*, que retransmite as informações conforme o estado atual da cena. Num ambiente multi-processado, o custo desta estratégia é muito maior que num ambiente mono-processado, pois o *Gerente* estará concentrando toda a comunicação entre as ferramentas, e o custo da comunicação será bem mais significativo, pois as mensagens estarão sendo transmitidas pela rede.

Assim sendo, as vantagens de isolar informações da cena no *Gerente* podem não compensar o custo envolvido, em se tratando de um sistema multi-processado. Naturalmente, faz-se necessário um estudo bem mais aprofundado para determinar com exatidão as vantagens e desvantagens desta alternativa.

Supondo-se que o custo de manter a informação da cena no *Gerente* seja efetivamente demasiado alto, existem algumas alternativas. A principal delas é repartir a tarefa efetuada pelo gerente entre as *bibliotecas de comunicação* de todas as ferramentas do ambiente, conforme melhor descrito na seção 3.4.4.1.

BIBLIOGRAFIA

[BOU 92] BOULIC, R; THALMANN, D. Combined Direct and Inverse Kinematic Control for Articulated Figure Motion Editing. COMPUTER GRAPHICS forum, v.11, n. 4, p.189-202, 1992.

[BRU 90] BRUDERLIN, A. Towards Realistic Human Figure Animation. SIGGRAPH 90 Course Notes - Human Figure Animation: Approaches and Applications, p.11-20, 1990.

[COH 93] COHEN, M. IRIS Animaker: a Adaptação do Sistema Animaker para Ambiente Silicon Graphics. Porto Alegre: UFRGS, 1993.

[FOL 90] FOLEY, J. D. *et al*. Computer Graphics: Principles and Practice. 2nd. ed. Reading:Addison-Wesley. 1990. 1174p.

[GOM 84] GÓMEZ, J. E. Twixt: A 3D Animation System. EUROGRAPHICS 84, p.121-133, 1984.

[GRE 91] GREEN, M. Using Dynamics in Computer Animation: Control and Solution Issues. Making Them Move, p.281-313,1991.

[LEM 92] LEMOS, R. R. Animação da Dinâmica do Movimento e Colisões de Corpos Rígidos. In: CLEI'92, Las Palmas de Gran Canaria, Set. 1992.

[MAG 85a] MAGNENAT-THALMANN, N; THALMANN,D. Miranim: An Extensible Director-Oriented System for the Animation of Realistic Images. IEEE Computer Graphics and Applications, Los Almitos, v. X, n. X, p. 61-73, Mar, 1985.

[MAG 85b] MAGNENAT-THALMANN, N; THALMANN,D. Computer Animation - Theory and Practice. Springer-Verlag, Tokio, 1985.

[MIC 93] MICROSOFT DEVELOPER NETWORK. Object Linking and Embedding 2.0 Backgrounder, 1993.

[MOR 87] MORGAN, D. The Iminent IPSE, Datamation, 33(7):60-68, 1987

[NAS 92] NASCIMENTO, M. E. Estudo de algoritmos para geração de sombras projetadas na síntese de imagens realísticas. Porto Alegre: CPGCC da UFRGS, 1992. Dissertação de mestrado.

[NED 92] NEDEL, L. P. Modelagem e Animação de Superfícies Deformáveis. In: CLEI'92, Las Palmas de Gran Canaria, Set. 1992.

[NED 93] NEDEL, L. P. Simulação de Objetos Deformáveis Baseada na Análise Dinâmica. Porto Alegre: CPGCC da UFRGS, 1993. Dissertação de mestrado.

[OLA 92] OLABARRIAGA, S. D. *et al*. Integrating Graphic Software - The making of "MATE". In: CLEI'92, Las Palmas de Gran Canaria, Set. 1992.

[PHI 90] PHILLIPS, C. B. *et al*. Interactive Real-time Articulated Figure Manipulation Using Multiple Kinematic Constraints. SIGGRAPH 90 Course Notes - Human Figure Animation: Approaches and Applications, p.235-240, 1990.

[SCH 92a] SCHMIDT, A. E., MUSSE, S. R. PREVIEW: Sistema de Animação. Porto Alegre: CPGCC da UFRGS, 1992.

[SCH 92b] SCHWINGEL, D. APART2: Uma arquitetura paralela assíncrona para ray-tracing em transputers. Porto Alegre: UFRGS, 1992.

[SIL 92] SILVA, R. L. ANIMAKER: Sistema de Animação. Porto Alegre: UFRGS, 1992

[STU 90] STURMAN, D. Interactive Keyframe Animation of 3-D Articulated Models. SIGGRAPH 90 Course Notes - Human Figure Animation: Approaches and Applications, p.32-41, 1990.

[WAG 94] WAGNER, F. R. Ambiente de Projeto de Sistemas Eletrônicos. IX Escola de Computação, Jul. 1994.

[WIL 87] WILHELMS, J. Using Dynamic Analysis for Realistic Animation of Articulated Bodies. IEEE Computer Graphics and Animation, p. 12-27. 1987.

ANEXO A-1 Funções da Biblioteca de Comunicação

A biblioteca de comunicação oferece às ferramentas uma interface padrão para se comunicarem obedecendo o protocolo de comunicação proposto no modelo. A biblioteca é acessada pelas ferramentas através da chamada de vários métodos da classe *Mensagem*, que implementa a biblioteca na versão atual do ambiente. Estes métodos são descritos a seguir:

A-1.1 Funções de Envio de Mensagens

```
void Mensagem::SetParameterName( char *string1 )
```

Esta função define o nome do parâmetro associado a determinada mensagem que será enviada.

```
void Mensagem::SetParameterValue(char *string1 )
void Mensagem::SetParameterValue (int integer1 )
void Mensagem::SetParameterValue (double real1, double real2, double real3 )
```

Estas três funções definem que o valor do parâmetro que fará parte de determinada mensagem será o definido nos parâmetros.

```
void Mensagem::AddMessage( int Message_Code )
```

Adiciona uma mensagem na fila de mensagens a serem enviadas. A operação definida na mensagem é especificada em *Message_Code*. O parâmetro e respectivo valor associados a mensagens são definidos nas funções *SetParameterName* e *SetParameterValue*.

```
void Mensagem::SendMessage( void )
```

Envia todas as mensagens colocadas na fila através do comando *AddMessage*.

A-1.2 Funções de Recepção de Mensagens

void Mensagem::GetParameterName(char *string1)

Esta função recebe em *string1* o nome do parâmetro associado a última mensagem decodificada da lista de mensagens recebidas.

void Mensagem::GetParameterValue(char *string1)
void Mensagem::GetParameterValue (int &integer1)
void Mensagem::GetParameterValue (double &real1, double &real2, double &real3)

Retornam nos parâmetros o valor enviado na última mensagem decodificada.

void Mensagem::Decode_Next_Msg(int &Message_Code)

Decodifica uma nova mensagem da lista de mensagens recebidas, e retorna em Message_Code o código da mesma.

void Mensagem::ReadMessage(void)

Le um pacote de mensagens. Se não há nenhum pacote sendo transmitido, espera até chegar alguma transmissão.

A-1.3 Exemplo de Código de Envio de Mensagens

```
Msg *com = new Msg;
com->Set_Obj_Name("bola");
com->Set_Par_Name("posicao");
com->Set_Par_Value(10.0, 30.0 , 30.0);
com->Add_Message(OP_LET);
com->Send_Message();
```

A-1.4 Exemplo de Código de Recepção de Mensagens

```
int id;
Msg *com = new Msg;

com->Read_Message();
while(com->Decode_Next_Msg(id) )
{
  com->Get_Obj_Name( obj_name );
  com->Get_Par_Name(  par_name );
  com->Get_Par_Value(  par_value );
}
```

ANEXO A-2 Códigos de Mensagens

A tabela a seguir apresenta todos os códigos de mensagens pré-definidos no *gerente*. É importante lembrar que nada impede duas ou mais ferramentas de utilizarem códigos próprios nas mensagens trocadas entre as mesmas.

OP_CREATE	Informa para um processo que um novo elemento foi criado na cena.
OP_LET	Informa o valor de determinado parâmetro
OP_ADVANCE	Avança para um determinado instante de tempo
OP_ASK	Pergunta pelo valor de determinado parâmetro
OP_ASKOBJ	A ferramenta requisita ser informada quando elemento do tipo especificado for criado na cena.
OP_ASKCOMP	Requita o valor de todos os parametros de todos os elementos de determinado tipo
OP_NEW_OBJ	Informa que determinado elemento foi criado na cena para as ferramentas que requisitaram tal informação (“OP_ASKOBJ”).
OP_ANSWER	Informa valor de determinado parâmetro de

	determinado elemento da cena.
OP_START	Inicia uma animação.
OP_END	Informa fim de uma animação.
OP_PHASE_LET	Início da primeira das três fases da geração de um instante de tempo.
OP_END_LET	Fim da primeira das fases da geração de um instante de tempo.
OP_PHASE_ASK	Início da fase de perguntas sobre os elementos da cena.
OP_END_ASK	Fim da fase de perguntas sobre os elementos da cena.
OP_PHASE_ANSWER	Início da fase de respostas sobre elementos da cena.
OP_END_ANSWER	Fim da fase de respostas.
MSG_TRACE_ON	Informa o processo que ele deve imprimir todas as mensagens que recebe.
MSG_TRACE_OFF	Informa o processo que ele não deve mais imprimir as mensagens que recebe.
OP_USER	Primeiro código livre para o usuário. Todos os valores a partir do código OP_USER

	não são usados internamente pelo gerente.

ANEXO A-3 Linguagem de Entrada do Leitor

A seguir é apresentada a linguagem de entrada aceita pelo leitor de roteiros implementado no protótipo. As seções seguintes apresentam vários exemplos de roteiros que facilitarão a compreensão da sintaxe apresentada.

<ROTEIRO> : <TEMPO> <OBJETOS>

<TEMPO>: Time = **<INTEIRO>** - **<INTEIRO>**

<OBJETOS>: [<OBJETO> <OBJETOS>]*

<OBJETO>: <TIPO_DO_OBJETO> <NOME_DO_OBJETO>
"Classe" <NOME_DA_CLASSE>
[<COMANDO>]*

<COMANDO>: { **<COMANDO>]*** }
<COMANDO>: t = **<INTEIRO>** -> **<NOME_PARAMETRO>** = **<VALOR>**
<COMANDO>: t = **<INTEIRO>** - **<INTEIRO>** ->
<NOME_PARAMETRO> = **<VALOR>** - **<VALOR>**
<COMANDO>: OnMessage (**<IDENT>** = **<VALOR>**) **<COMANDO>**

<VALOR>: [<INTEIRO> ,<REAL>,<STRING>]*
<IDENT>: <STRING>

Os tipos *STRING, REAL e INTEIRO* são tipos básicos representando, respectivamente, vetores de caracteres, números reais e números inteiros.

ANEXO A-4 Exemplos de Animações

Este anexo apresenta alguns exemplos de animações desenvolvidas utilizando o protótipo apresentado neste trabalho.

A-4.1 Movimento de Dois Objetos

O roteiro a seguir define o movimento de dois objetos que se movem em direção a um mesmo ponto no espaço. Os objetos são observados por uma câmera.

```
Time = 0 - 20

Objeto z
Classe cinematico
{
  t = 0 - 20 posicao = 0.0 100.0 40.0 - 0.0 0.0 0.0
}

Objeto y
Classe cinematico
{
  t = 0 - 20 posicao = 0.0 100.0 0.0 - 0.0 0.0 0.0
}

Camera view
Classe viewer
{
  t = 0 posicao = -400.0 0.0 0.0
  t = 0 alvo    = 0.0 0.0 0.0
  t = 0 lente   = 90.0 0.0 0.0
  t = 0 rastro  = sim
}
```

Este exemplo utiliza dois atores cinemáticos (“Classe cinematico”), mas não utiliza recursos de animação cinemática. O movimento dos atores é gerado pela interpolação do parâmetro “posicao”, utilizando os recursos de interpolação do próprio leitor de roteiros.

A figura A-4.1 apresenta uma imagem da cena. Os quadros estão sobrepostos devido à especificação do parâmetro “rastro” no roteiro.

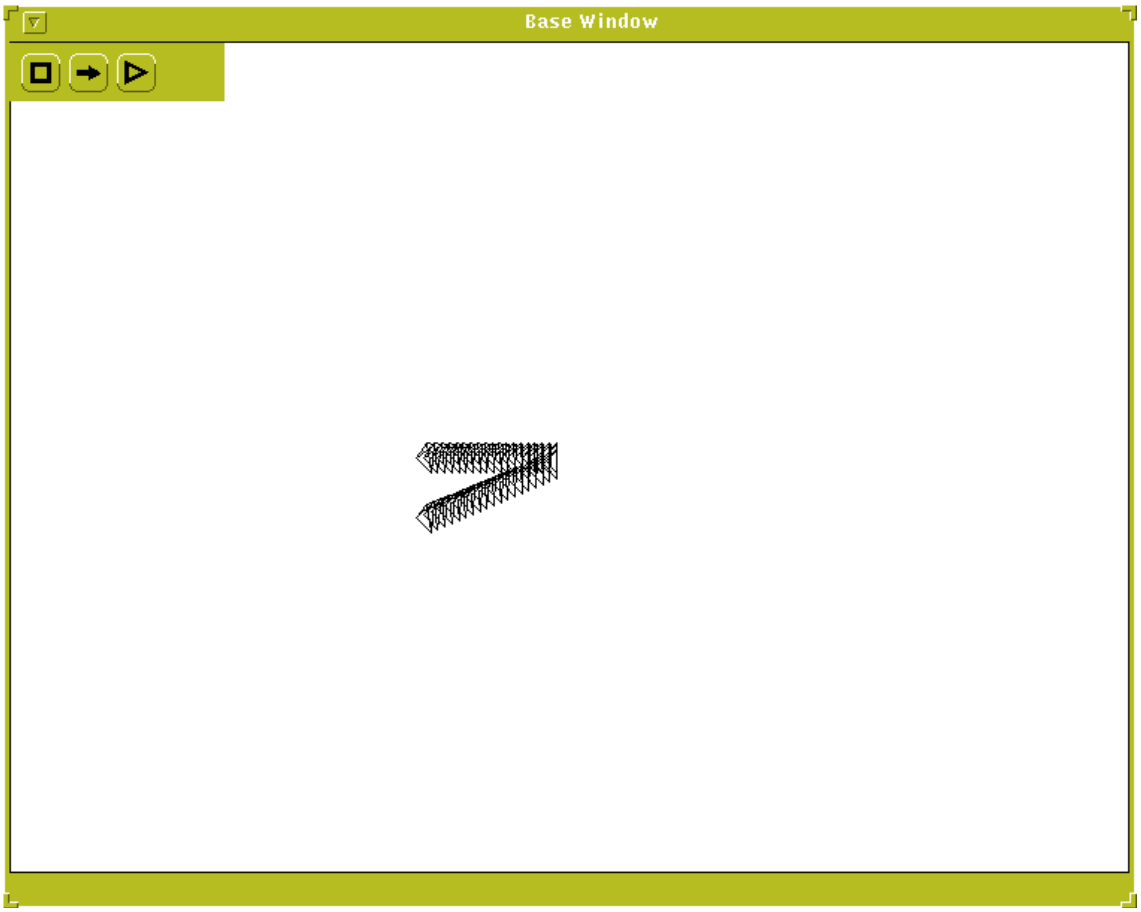

fig A4.1 – Imagem da Cena com Rastro

A-4.2 Movimento Visto por Duas Câmeras

Uma vez que o roteiro permite especificar qualquer ferramenta presenta na animação, é possível definir, por exemplo, que uma animação será exibida por duas ferramentas diferentes. O roteiro a seguir ilustra esta alternativa:

```
Time = 0 - 25

Objeto z
Classe cinematico
{
  t = 0 - 20 posicao = 0.0 100.0 40.0 - 0.0 0.0 0.0
}

Objeto y
Classe cinematico
{
  t = 0 - 20 posicao = 0.0 100.0 0.0 - 0.0 0.0 0.0
}

Camera view
Classe viewer
{
  t = 0 posicao = -400.0 0.0 0.0
  t = 0 alvo    = 0.0 0.0 0.0
  t = 0 lente   = 90.0 0.0 0.0
  t = 0 rastro  = sim
}

Objeto view2
Classe viewer2
{
  t = 0 posicao = 0.0 200.0 200.0
  t = 0 alvo    = 0.0 0.0 0.0
  t = 0 lente   = 90.0 0.0 0.0
}
```

A figura A-4.2 apresenta uma seção da tela de uma estação de trabalho SUN, onde aparecem as duas janelas das duas ferramentas. Observe que são dois exibidores diferentes, embora construídos a partir de um mesmo modelo básico, sendo que apenas um deles exibe as imagens com rastro.

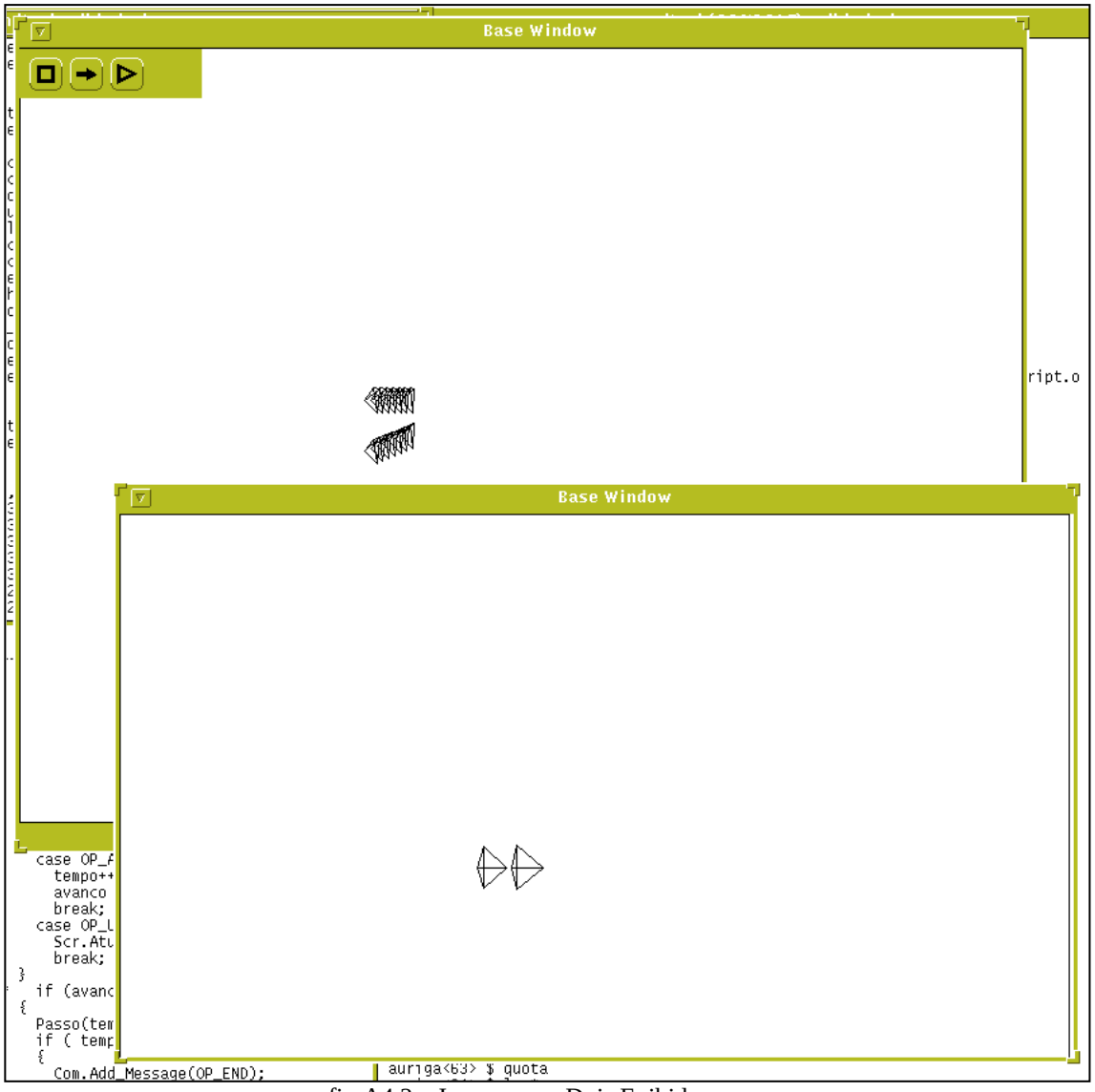

fig A4.2 – Imagem em Dois Exibidores

A-4.3 Geração de Movimentos Cinemáticos

O terceiro exemplo apresenta o uso dos recursos extremamente simples de movimentação cinemático presentes no protótipo. Ao invés de gerar o movimento diretamente através da interpolação do parâmetro “posicao”, o roteiro especifica o parâmetro “velocidade”, que faz com que o sistema de cinemática gere automaticamente a posição do ator em cada quadro da cena. A seguir está o roteiro desta animação:

```
 Time = 0 - 20

Objeto obj1
Classe cinematico
{
  t = 0 - 20 posicao = 0.0 100.0 0.0 - 0.0 -100.0 0.0
}


Objeto obj2
Classe cinematico
{
  t = 0 posicao = 0.0 -100.0 0.0
  t = 0 velocidade = 0.0 10.0 0.0
}

Camera view
Classe viewer
{
  t = 0 posicao = -400.0 0.0 0.0
  t = 0 alvo    = 0.0 0.0 0.0
  t = 0 lente   = 90.0 0.0 0.0
  t = 0 rastro  = sim
}
```

A figura A-4.3 a seguir apresenta a tela nos instantes iniciais da animação. A figura seguinte (figura A-4.4) apresenta a tela após vinte instantes de tempo.

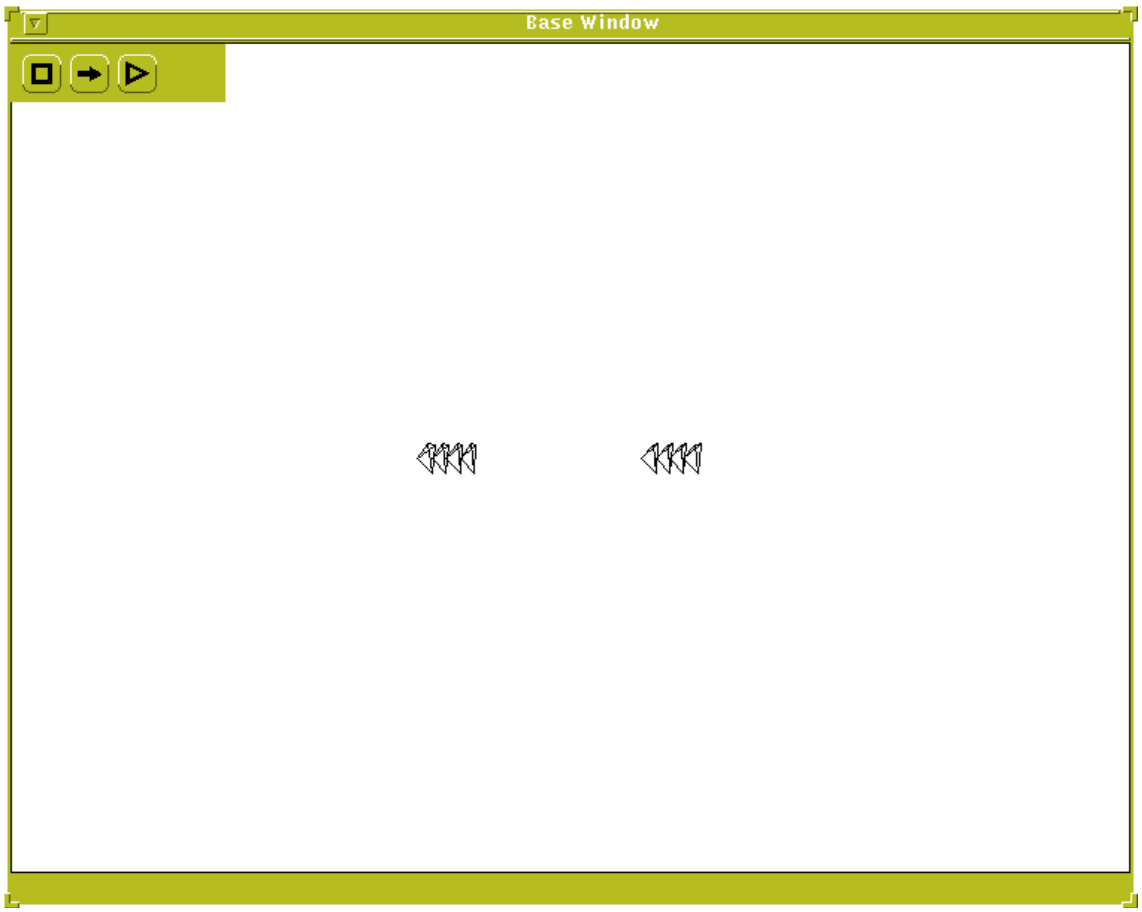

fig A4.3 – Imagem nos Instantes Iniciais

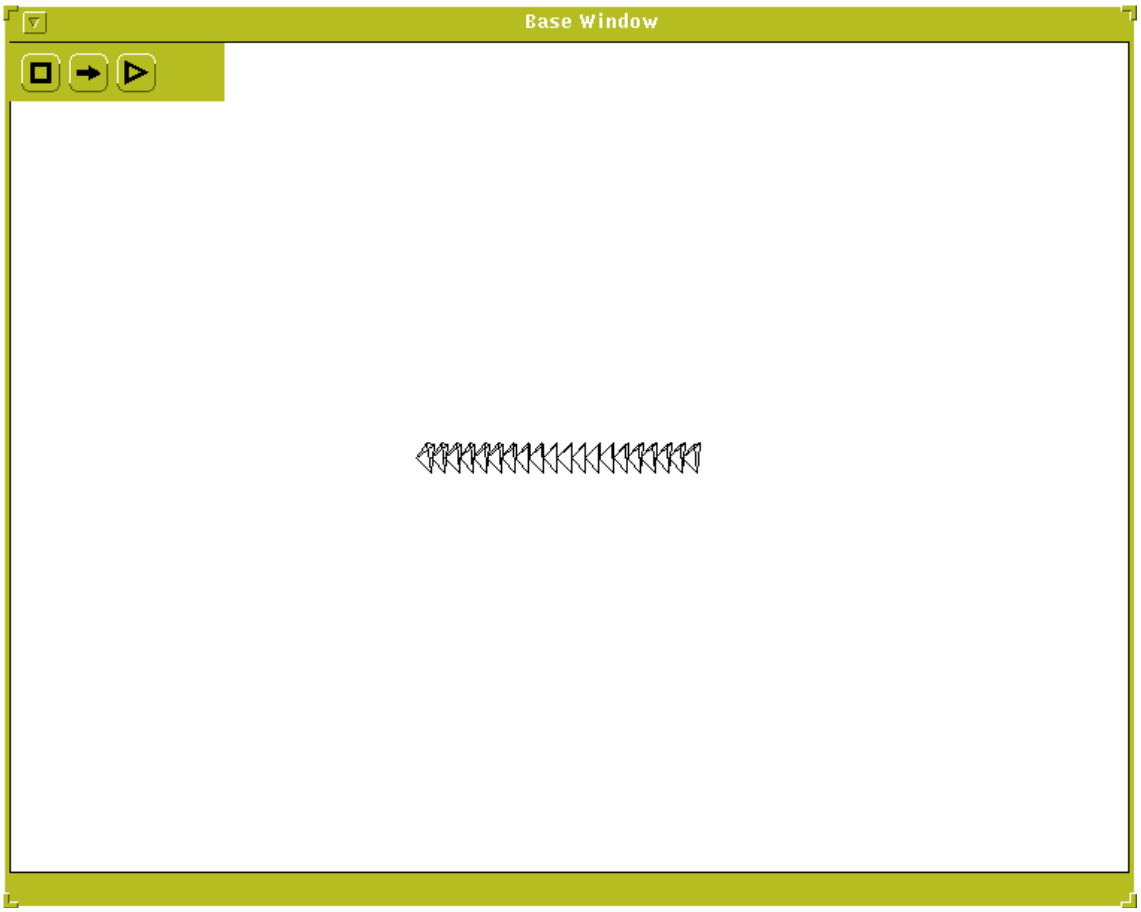

fig A4.4 – Imagem Após 20 Instantes de Tempo

A-4.4 Exemplo com Detector de Proximidade

O detector de proximidades implementado neste protótipo atua de forma integrada com o leitor de roteiros. Quando percebe que dois dos elementos da cena encontram-se a uma distância menor que um limiar definido, o detector envia uma mensagem definida no roteiro para o leitor. O leitor, por sua vez, permite que o roteiro defina um tratamento particular quando da recepção de determinadas mensagens.

A seguir está um exemplo bastante simples do uso desta técnica. O roteiro é idêntico ao apresentado na seção anterior, exceto pelo tratamento da detecção de proximidade. A figura A-4.5 exibe a animação resultante, na qual pode-se observar as diferenças em relação a animação anterior.

```
Time = 0 - 20

Objeto z
Classe teste
{
  t = 0 - 20 posicao = 0.0 100.0 0.0 - 0.0 -100.0 0.0
}

Objeto z2
Classe cinematico
{
  t = 0 posicao = 0.0 -100.0 0.0
  t = 0 velocidade = 0.0 10.0 0.0
  ON_MESSAGE( colisao = Colisao_Iminente )
  {
    t = 0 velocidade = 0.0 10.0 -30.0
  }
}

Camera view
Classe viewer
{
  t = 0 posicao = -400.0 0.0 0.0
  t = 0 alvo    = 0.0 0.0 0.0
  t = 0 lente   = 90.0 0.0 0.0
  t = 0 rastro  = sim
}

Detetor detec
Classe detetor
{
  t = 0 distancia = 20.0 10.0 10.0
  t = 0 mesg = Colisao_Iminente
}
```

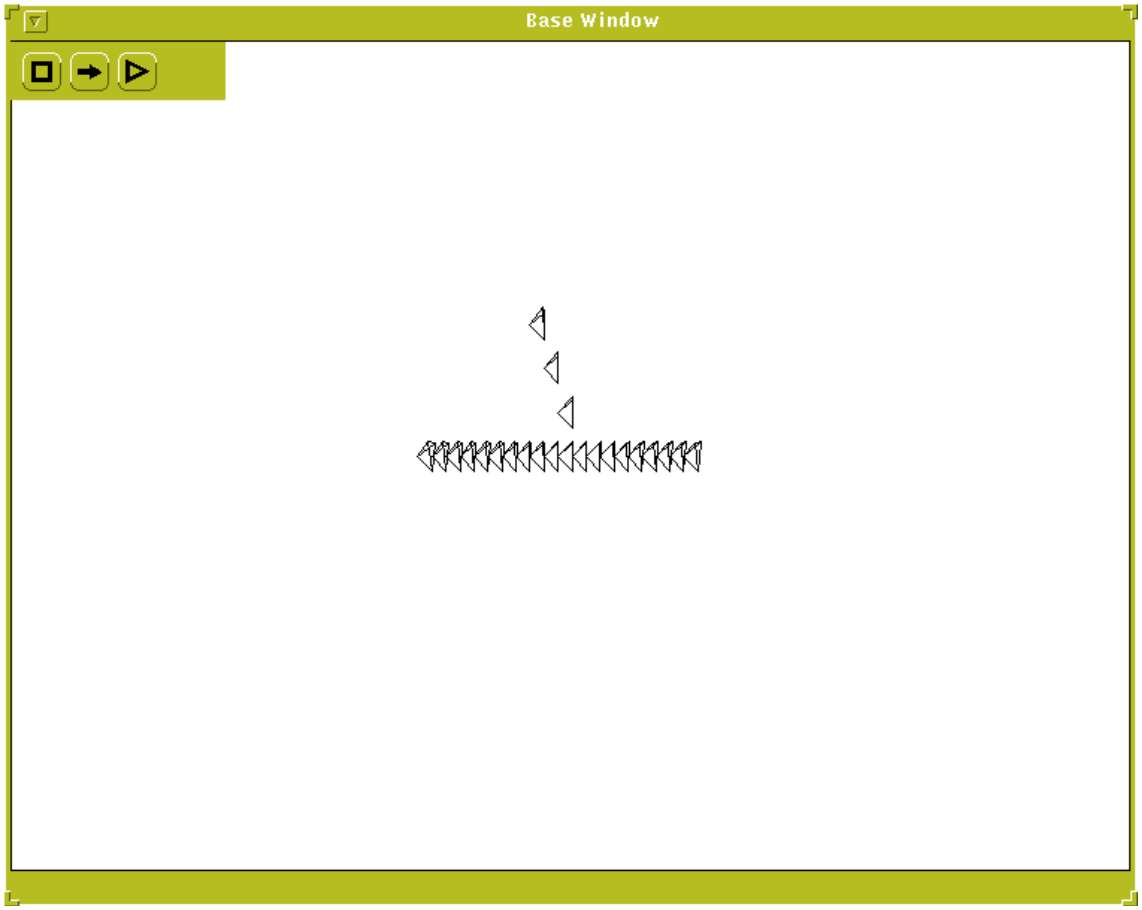

fig A4.3 – Imagem com Detector de Colisão

ANEXO A-5 Animação Implementada em C++

Neste exemplo, dois atores executarão movimentos circulares em torno de um ponto no espaço. Não existe como gerar tal animação apenas utilizando a linguagem de especificação de roteiro, já que nenhuma das ferramentas desenvolvidas possibilita a especificação deste tipo de movimentos. Assim, este exemplo serve principalmente para mostrar as possibilidades de integração de roteiros com funções construídas diretamente em C++.

A forma mais simples de gerar uma animação utilizando os recursos da linguagem C++ é criar uma ferramenta de geração de movimento integrada ao ambiente, com o detalhe de que os algoritmos desta ferramenta em particular serão direcionados a geração de uma única animação, embora, é claro, nada impeça que venham a ser utilizados em futuras animações

A construção desta animação precisa ser feita em duas etapas: a construção do código que irá implementar o movimento circular e a construção de um roteiro para a animação.

A-5.1 Implementação do Movimento Circular

A primeira questão a ser definida é o tipo de informações que precisam ser colocadas a disposição do animador, através do roteiro. Neste exemplo, para a definição de um movimento circular foram considerados três parâmetros: o centro do movimento circular, o raio do movimento e o eixo em torno do qual o ator iria executar o movimento. Naturalmente outros parâmetros poderiam ser incluídos para tornar o movimento mais flexível.

O primeiro passo é informar os parâmetros na interface de geração de ferramentas, que gerará um esqueleto ao qual serão incluídos os algoritmos necessários.

O código fonte a seguir apresenta uma parte do código da ferramenta gerada. Os comandos com fonte em negrito foram incluidos diretamente, enquanto os demais foram gerados automaticamente.

```
void Objeto::Init( void )
{
  eixo_fixo.Value = new char[strlen("NULL")];
  strcpy(eixo_fixo.Value,"NULL");

  angulo_atual = 0; // inicializacao do angulo no aparecimento do objeto.
}

void Objeto::Advance( void )
{
  int i,total;

  fprintf(stderr,"\nanim:posicao = ");
  fprintf(stderr,"(%lf,%lf,%lf)",posicao.x, posicao.y, posicao.z);
  fprintf(stderr,"\nanim:raio = ");
  fprintf(stderr,"(%lf)",raio.Value);
  fprintf(stderr,"\nanim:eixo_fixo = ");
  fprintf(stderr,"(%s)",eixo_fixo.Value);
  fprintf(stderr,"\nanim:centro = ");
  fprintf(stderr,"(%lf,%lf,%lf)",centro.x, centro.y, centro.z);

  if ( !strcmp(eixo_fixo.Value,"z"))
    {
      posicao.z = centro.z;
      posicao.x = centro.x + sin(angulo_atual)* raio.Value;
      posicao.y = centro.y + cos(angulo_atual)* raio.Value;
    }
    else if ( !strcmp(eixo_fixo.Value,"x"))
    {
      posicao.x = centro.z;
      posicao.y = centro.x + sin(angulo_atual)* raio.Value;
      posicao.z = centro.y + cos(angulo_atual)* raio.Value;
    }
    else
   {
      posicao.y = centro.z;
      posicao.z = centro.x + sin(angulo_atual)* raio.Value;
      posicao.x = centro.y + cos(angulo_atual)* raio.Value;
    }
   angulo_atual+=0.3;
}
```

A seguir está apresentado o roteiro de uma animação com a presença de dois atores executando movimentos circulares. Após, são mostradas imagens da cena em dois instantes distintos de tempo.

```
Time = 0 - 20

Objeto z
Classe anim
{
  t = 0 centro = 0.0 0.0 0.0
  t = 0 raio = 300.0 0.0 0.0
  t = 0 eixo_fixo = z
}

Objeto z2
Classe anim
{
  t = 0 centro = 0.0 0.0 0.0
  t = 0 raio = 300.0 0.0 0.0
  t = 0 eixo_fixo = y
}

Camera view
Classe viewer
{
  t = 0 posicao = -550.0 -100.0 -100.0
  t = 0 alvo    = 0.0 0.0 0.0
  t = 0 lente   = 90.0 0.0 0.0
  t = 0 rastro  = sim
}
```

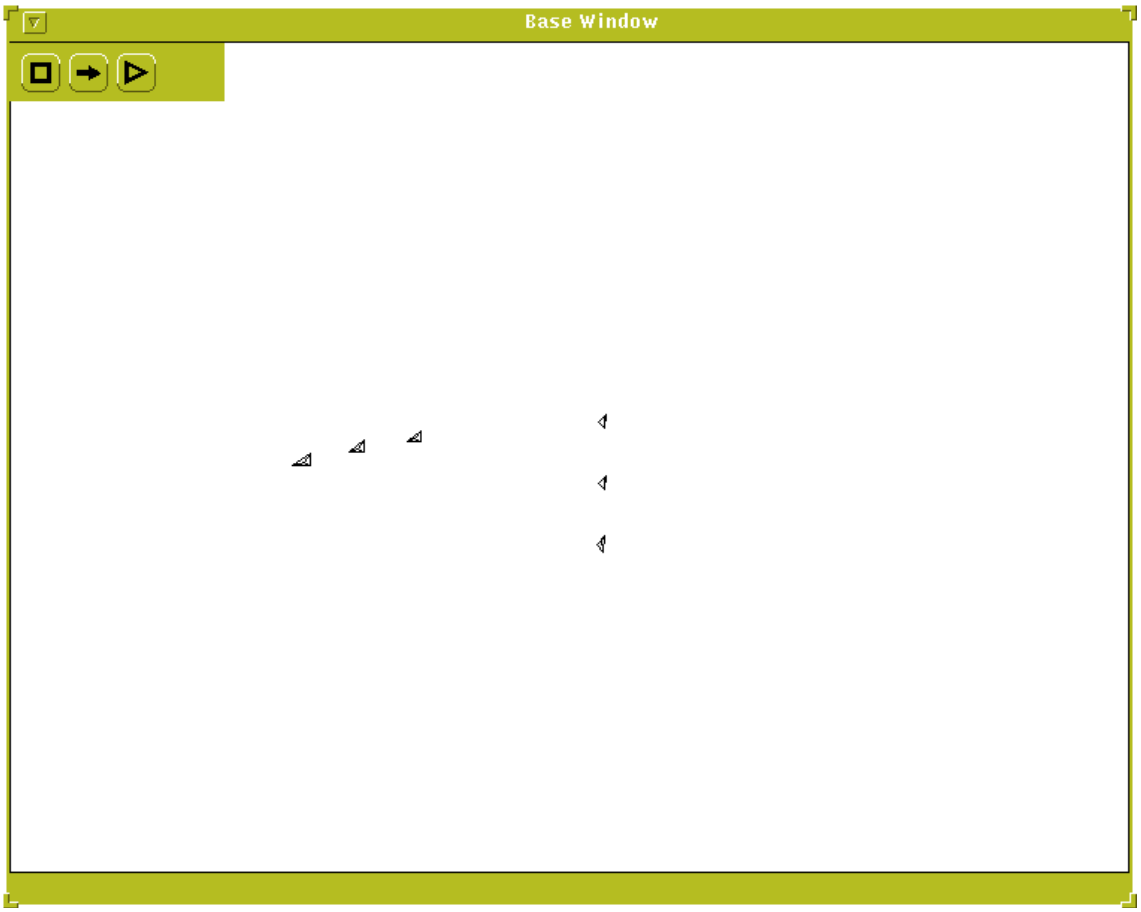

fig A5.1 – Início da Animação com Movimento Circular

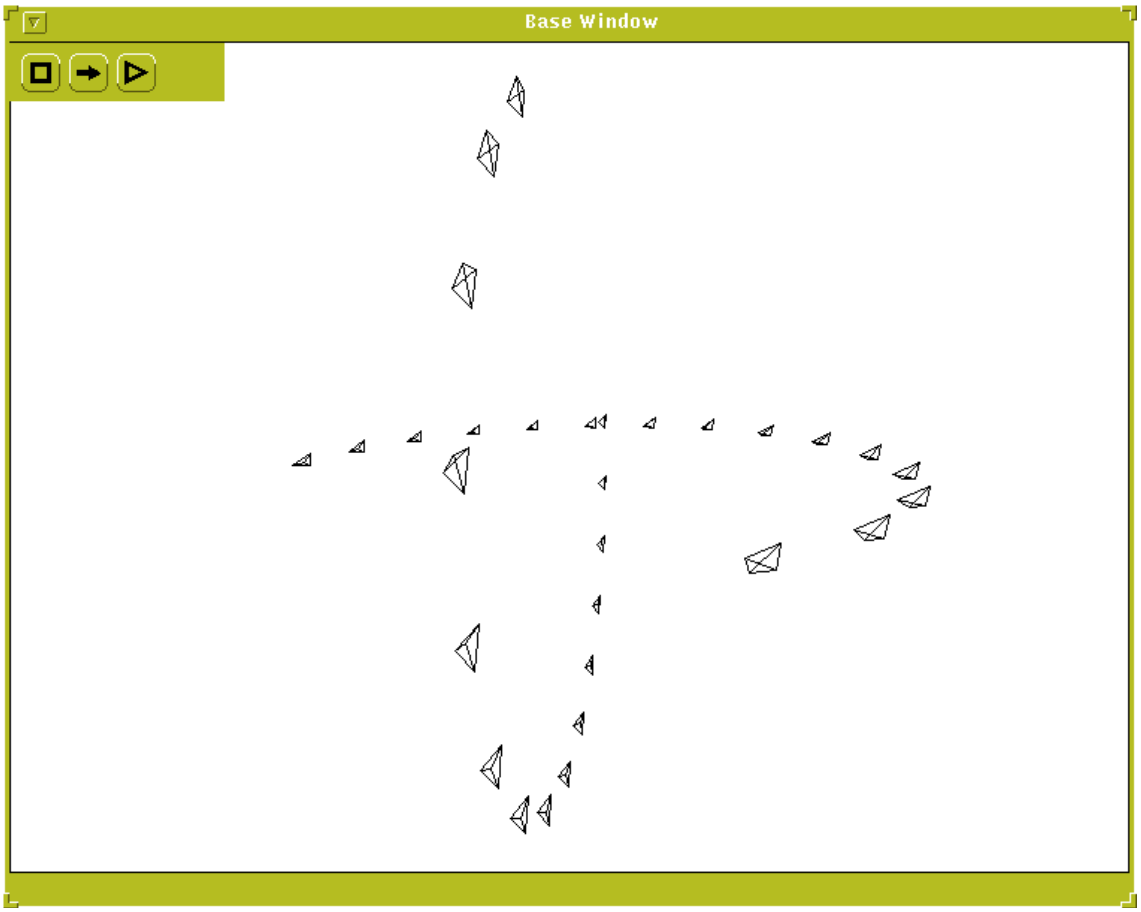

fig A5.2 – Imagem com Movimento Circular