# Espaços e Relações na arquitectura dum Modelo de Animação

*José Eduardo Marreiros*
*Manuel João Próspero*
*Fernando Pedro Birra*

Universidade Nova de Lisboa
Departamento de Informática da FCT
Quinta da Torre
2825 Monte de Caparica

## Sumário

Em regra, os sistemas de animação por computador tendem a ser bastante orientados para uma determinada aplicação. Não é comum encontrarem-se sistemas que permitam simultaneamente, por exemplo, animação cinemática e dinâmica. Surgem, por vezes, situações em que se torna necessário definir algumas trajectórias por meio de equações cinemáticas, outras por interpolação, e outras, ainda, a partir da indicação de forças, momentos, etc..

A presente comunicação visa descrever o ponto fundamental dum trabalho que tem vindo a ser realizado no DI da FCT-UNL e que consiste na concepção de um modelo teórico de animação juntamente com a implementação do respectivo protótipo em ambientes NeXT™ e OSF Motif.

# Introdução

Animação denota uma variação temporal de certas características dum determinado universo. Independentemente do tipo de variações que o sistema sofra, em animação todas elas têm um factor em comum: são alterações que se dão ao longo do tempo. Todo o sistema pode ser descrito em função do tempo.

No caso da animação por computador, trata-se do tratamento temporal que se pode dar a um sistema, utilizando modelos computacionais. Um sistema de animação deve permitir, da forma mais geral possível, definir qualquer tipo de animação. Por outro lado, deve ser suficientemente flexível para oferecer facilidades de alto nível na definição duma animação.

No presente estudo, as noções estão mais orientadas para a animação de sistemas realistas com cenários e actores tridimensionais. No entanto, o presente pode servir de base para a realização de sistemas aplicados a modelos bidimensionais ou mesmo modelos abstractos (não realistas).

# Sistemas de Animação

Os sistemas de animação podem ser classificados [Mag88] com base em vários parâmetros:

- pela sua aplicação prática;
- pelos modelos utilizados;
- pela forma como se descreve o movimento;
- por outros menos relevantes.

Centrando a nossa atenção na forma de descrever mutações à cena, podemos classificar os sistemas de animação em três grandes grupos:

- cinemáticos;
- por *key-frame*;
- dinâmicos.

A grande diferença entre os modelos reside no grau de complexidade que as especificações de movimento podem tomar [Pue88].

## Modelo Cinemático

O modelo cinemático permite que todas as alterações ao universo sejam controladas directamente por intermédio de, por exemplo, equações do movimento. Neste caso, todos os intervenientes têm a alteração das suas propriedades bem definidas através de equações matemáticas específicas. As trajectórias podem ainda ser especificadas, a um nível mais abstracto, em função de alguns parâmetros como sejam a velocidade inicial e a aceleração.

Assim, um eventual utilizador do sistema terá grande flexibilidade na especificação duma animação. Em contrapartida, o utilizador tem que detalhar convenientemente todos os movimentos individuais dos diversos actores. Torna-se, assim, num método eventualmente moroso de descrever animações.

## Modelo por *Key-frame*

Surgem, por vezes, situações em que as equações do movimento, ou são desconhecidas, ou difíceis de encontrar. O que se conhece, isso sim, são certos estados do sistema correspondendo a determinados instantes. Nestes casos, a animação é descrita através da interpolação entre os diversos valores amostrados. É o princípio de funcionamento dos modelos por *key-frame*.

Estes modelos permitem um maior conforto na definição de animações, no entanto a custo duma menor flexibilidade na descrição do movimento.

Modelo Dinâmico

Nos modelos dinâmicos, a animação é controlada quase exclusivamente pelo computador, uma vez fornecidas as leis gerais da interacção entre os intervenientes e as suas condições iniciais. Normalmente, a descrição das leis de evolução baseiam-se na lei da atracção universal dos corpos (incluindo descrição de forças, momentos de inércia, velocidades iniciais, etc.) ou em modelos mais abstractos como os autómatos celulares [Vei91].

A evolução dum modelo dinâmico é feita recorrendo a processos numéricos de aproximações sucessivas. A determinação prévia das equações cinemáticas implicaria a resolução de equações diferenciais, por vezes sem resolução simbólica.

Este é, sem dúvida, o modelo mais cómodo, do ponto de vista da descrição duma animação. Com base nas leis gerais da interacção entre os intervenientes, e sabendo as suas condições iniciais, o sistema pode evoluir sem intervenção externa.

Níveis de especificação

À semelhança do que acontece com as linguagens de programação, assim os modelos de animação podem classificar-se [Fol90] do ponto de vista da especificação duma animação como de baixo ou alto nível. Quanto mais elementar e genérica é a forma da especificação, mais difícil e moroso se torna o processo de definição da evolução temporal do sistema. Isto acontece quando o animador descreve explicitamente a posição e os atributos de cada objecto na cena, através de translações, rotações e outros operadores geométricos ou de mudança de atributos. Se, pelo contrário, a forma de definir uma animação for mais específica e intuitiva, o processo torna-se mais fácil, mas menos controlável e por vezes de difícil previsão. É o que sucede quando se usam sistemas *knowledge-based* onde o animador diz: "mover o actor para perto da mesa e virá-lo para a porta" [Tha86]. Esta descrição de alto nível tem, obviamente, que ser traduzida em operadores mais elementares. No entanto, o processo é inegavelmente mais cómodo.

O que aqui importa não é se a linguagem usada se considera de alto ou baixo nível, mas, sim, quais os mecanismos básicos que nos permitam estabelecer variações temporais. É a partir destes mecanismos que se poderão criar linguagens de especificação mais ou menos elaboradas.

# Modelo proposto

Como tentativa de unificação, foi concebido um modelo de animação que permitisse, do ponto de vista teórico, implementar qualquer tipo de sistema de animação (Figura 1).

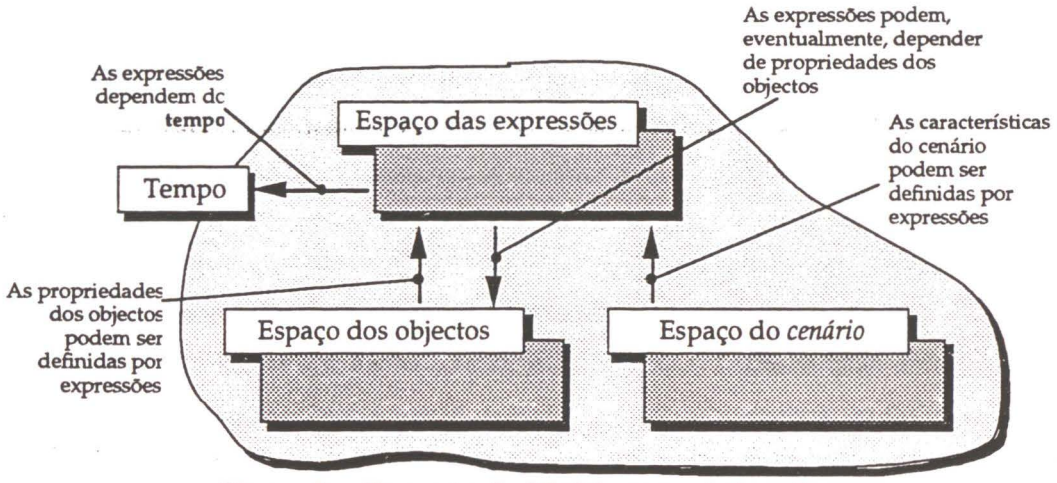



**Figura 1 - Proposta de Modelo para Animação**

Dado um sistema que se pretende animar, há a necessidade de o modelar estaticamente e, posteriormente, impor-lhe regras de movimento que permitam alterar o seu estado em função do tempo. Por modelo estático entende-se a definição da constituição do sistema: entidades intervenientes (actores) e suas características, parâmetros ambientais ou de contexto, etc..

Neste modelo unificador estão definidos três espaços: o das expressões, o dos objectos e o do cenário. Esta divisão permite o desenvolvimento em separado e consoante a aplicação que se lhes queira dar. Consequentemente é possível criar sistemas hierárquicos, programáveis e flexíveis, onde os problemas possam ser sub-divididos, para uma melhor resolução [Kal92].

Como se vê pela Figura 1, podem haver dependências (relações) entre os espaços. A existência, ou não, dalgumas destas dependências permite definir sistemas de animação com comportamentos bem distintos. O tempo, por seu lado, é a variável indispensável, pois dele dependerá, em última análise, todo e qualquer sistema de animação.

## Espaços

As grandes diferenças entre os diversos sistemas de animação prendem-se essencialmente com os modelos matemáticos utilizados [Ary86]. A noção de Espaço das Expressões surge como forma de encapsular vários modelos matemáticos. Por exemplo, há muitas formas de se especificar uma trajectória. Podem ser utilizadas expressões explícitas de leis cinemáticas do movimento. Ou podem, ainda, ser utilizadas leis implícitas, através da especificação da interacção entre objectos [Heg89].

Dado reconhecer-se ao Espaço das Expressões a grande importância do papel desempenhado na especificação da animação, empregaram-se todos os esforços no sentido de se desenvolver um espaço que permitisse definir expressões adequadas a qualquer tipo de sistema. São exemplo dessa diversidade as expressões algébricas, a interpolação de valores e as funções definidas por ramos.

No Espaço dos Objectos, por seu lado, existe toda uma colecção de possíveis intervenientes numa animação. Os objectos têm características próprias, conhecidas como Atributos, e que podem considerar-se como parâmetros afectados por variações temporais. A composição duma cena é feita colocando-se objectos (também denominados Actores) num certo universo.

No espaço do Cenário, por último, existem elementos secundários à animação, no sentido de servirem principalmente para complementar e dar realismo às imagens. São, pois, as fontes de luz, os efeitos especiais (e.g. nevoeiros, neblina, etc.) e os parâmetros das câmaras que definem a forma como a cena irá ser visualizada (e.g. abertura, focagem, etc.).

## Relações

Do ponto de vista do modelo apresentado, uma animação é definida a partir dum modelo estático da cena que depois é animado. A construção do espaço dos objectos e do espaço do cenário define, imediatamente, uma situação estática. É no espaço das expressões que se define um conjunto de leis ou regras permitindo impor animação ao sistema.

O modelo prevê a atribuição de uma regra de movimento a qualquer parâmetro que defina atributos de objectos ou elementos de cenário. Desta forma, qualquer caracterização (numérica) da cena pode ser animada por meio duma expressão. São disso exemplo as usuais transformações geométricas aplicadas a objectos, as texturas, o ponto de vista da câmara, ou até a tonalidade e/ou intensidade da luz ambiente.

Estando estes três espaços (expressões, objectos e cenário) definidos de forma independente, é através das relações que se estabelecem as dependências entre eles. A implementação das relações permite, num dado instante, calcular todas as expressões relevantes e, de seguida, atribuir os seus valores aos actores em causa.

O funcionamento das relações pode ser descrito como uma recolha de valores e posterior distribuição dos mesmos. Em primeiro lugar é calculado o valor da expressão associada. Seguidamente, esse valor é colocado nas variáveis referenciadas pela relação, alterando assim os seus valores. Esta implementação permite ainda que uma expressão seja calculada uma única vez, independentemente do número de atributos que dela dependam.

Uma outra abordagem possível consiste em calcular as expressões à medida que os seus valores vão sendo necessários. Tem-se, assim, a garantia que apenas se calcula o que é estritamente

necessário. Da mesma forma que no método anterior, as expressões apenas são calculadas uma única vez.

# Protótipo dum sistema de animação

O objectivo final dum sistema de animação é, usualmente, a produção de uma sequência de imagens que, depois de mostradas em sequência, constituem o filme final. Assim, cooperando com o motor do sistema de animação, é necessário um sintetizador de imagens capaz de gerar os sucessivos quadros (*frames*) correspondentes a cada um dos instantes.

O protótipo que está correntemente implementado usa um *ray-tracer* [Dua91a] para a geração dos quadros. Este *ray-tracer* possui uma linguagem de especificação de cena que utiliza alguns objectos geométricamente simples (cubos, esferas, cones, arcos, pirâmides, cilindros, planos, anéis, torus, etc.) para criar objectos compostos e definir a cena [Dua91b] . Veja-se então, na Figura 2, a arquitectura global do sistema.

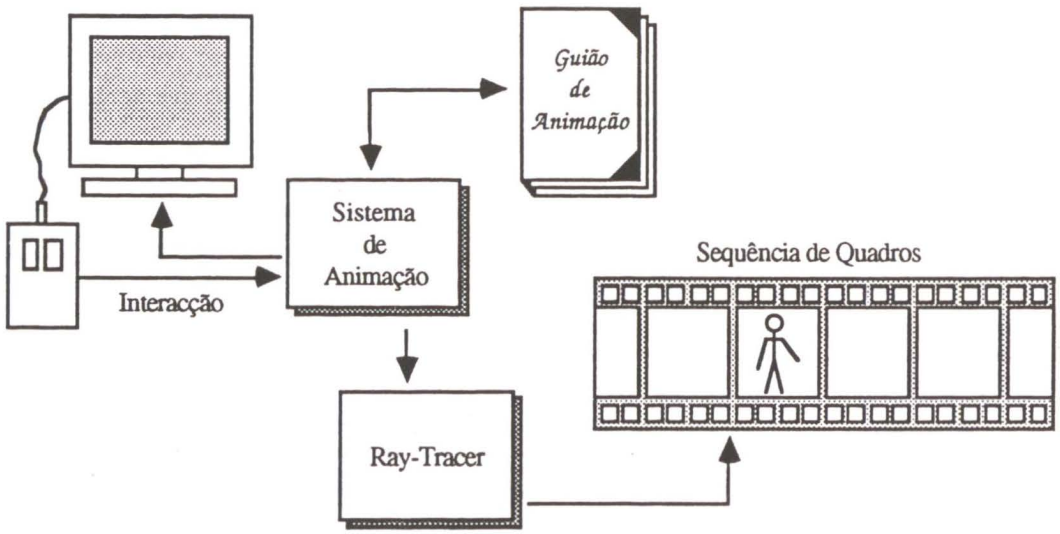



**Figura 2 - Arquitectura do protótipo**

Embora os espaços do modelo proposto (Figura 1) sejam independentes, podendo existir desenvolvimentos em separado, é lógico ser necessário manter um protocolo de comunicação entre eles. Isto é levado a cabo, como já foi referido, pelas relações. É este o espaço que necessita comunicar com os outros, i.e. avaliar as expressões e associar os seus valores aos atributos dos objectos e do cenário. A linguagem utilizada para a implementação do protótipo foi o C++. Esta linguagem, uma vez que possui mecanismos de herança , e visto ser orientada para os objectos,

permite construir um sistema evolutivo, ou seja, um sistema que possa ser expandido facilmente, sem obrigar a reformular o que já existe.

No protótipo está implementado um núcleo de expressões que permite desenvolver uma vasta gama de animações. A implementação das expressões está feita pensando em futuros desenvolvimentos, daí o C++. Vejamos então qual a ajuda que a herança e os objectos nos podem dar.

Existe uma classe de objectos, designada por **expressão**, que contém tudo o que é necessário do ponto de vista do espaço das relações (função para avaliar uma expressão) de modo a que este possa fazer a ligação com os outros espaços (objectos e cenário). Esta classe define uma função virtual (que pode ser redefinida pelas classes suas derivadas) denominada **avaliar**, cuja finalidade é, como o seu nome indica, retornar o valor da expressão. A partir daqui podemos definir um vasto conjunto de expressões como classes derivadas da classe **expressão**. As relações apenas conhecem a classe base, e isto é tudo o que necessitam saber, pois a função de avaliação adequada a cada tipo de expressão é chamada, automáticamente, através dos mecanismos que a linguagem possui para redefinição de funções virtuais.

Uma vez que o sistema de animação foi construído depois do *ray-tracer*, o espaço dos objectos já estava, à partida, determinado, assim como o espaço do cenário. O *ray-tracer* funciona tendo como entrada uma descrição da cena, onde vêm mencionados todos os valores dos atributos dos seus constituintes. Por exemplo, se quisermos um cone na posição (0, 0, 10), de raio 2 e altura 1, podemos dizer:

```
• cone { escala 2, 1, 2 transl 0, 0, 10 }.
```

Observe-se que as tranformações são efectuadas da esquerda para a direita. Optámos então por desenvolver uma pequena linguagem adequada à definição de expressões e incorporá-la, juntamente com a linguagem do *ray-tracer*, numa outra que permitisse efectuar simultaneamente dois processos:

- Descrever as expressões que regem os movimentos;
- Afectar essas expressões aos actores e cenário.

De seguida daremos alguns exemplos de animações, bem como um breve resumo do espaço de expressões que está implementado.

## Espaço das Expressões (um exemplo)

Este espaço pode ser implementado da forma que se achar mais conveniente, tendo em vista o objectivo final do sistema. A linguagem que for usada poderá ser mais ou menos elaborada, tudo depende do grau de complexidade que se lhe quiser atribuir.

No espaço das expressões existem várias entidades, que podem ser agrupadas em duas classes. A primeira consiste em objectos cujo valor tem que ser conhecido *a priori*. São as chamadas constantes. A segunda classe contém objectos cujo valor pode ser calculado *a posteriori*. Temos nesta classe as expressões, as condições e as funções.

## Constantes

As constantes são úteis para podermos rescrever o mesmo valor de uma forma mais abreviada. Nesta classe encontram-se os domínios, os pontos e os vectores.

### Domínios

Um domínio é definido a partir da noção básica de intervalo. Tal como na matemática, permite-se a coexistência de intervalos com extremos abertos e/ou fechados. Podem, ainda, definir-se intervalos com extremos infinitos. Sobre os intervalos podem realizar-se as usuais operações matemáticas de conjuntos: intersecção, reunião e diferença. Vejam-se então alguns exemplos de domínios:

```
• dom1 := [ -3, 4 ]
• dom2 := ] 7, _ [                          /* _ representa +∞  */
• dom3 := ] _, 8 ]                          /* _ representa -∞  */
• dom4 := # - ( dom1 + dom2 ) * ] _, 0 ]    /* # representa R   */
```

### Pontos

A noção de ponto deriva directamente da matemática. São pontos de um espaço real n-dimensional. Vejamos exemplos de pontos:

```
• ponto1 := < 0.0, 1.1, 2.1 >
• ponto2 := < 1.0, 3.3, 4.7 >
• ponto3 := < 3.2, -4.2 >
```

### Vectores

A noção de vector aplica-se a um conjunto de pontos, todos do mesmo espaço (i.e. com o mesmo número de coordenadas). São extremamente úteis para definir interpolações:

```
• vector1 := { ponto1, < 2.7, -3.3, 1.2 >, ponto2 }
```

## Expressões

Consideram-se as expressões, dentro deste espaço, das entidades mais importantes. São todas de tipo real e representadas numa estrutura em árvore que reflecte, desde logo, a sua composição e permite a sua avaliação sem ambiguidades. As expressões estão divididas em 3 grandes grupos:

- Operações aritméticas

- Chamada de funções
- Interpolações e aproximações

Operações aritméticas

São completamente indispensáveis na construção de expressões matemáticas os operadores aritméticos a que todos estamos habituados: soma, subtracção, multiplicação, divisão e o menos unário. A sintaxe utilizada é a mais tradicional possível:

```
• pi = 3.1514, raio = tempo, area = pi * raio * raio
• angulo = - pi / 2
```

Chamada de funções

Uma expressão aritmética ficaria muito pobre se não fosse permitido o uso de funções. As funções são todas de tipo real, i.e. retornam todas um valor real. As expressões do tipo "chamada de função" são avaliadas da seguinte forma: avaliam-se primeiro os seus argumentos, que também são expressões, e de seguida enviam-se para a função que trata de calcular o resultado, retornando esta o seu valor. As chamadas recursivas de funções são permitidas pelo sistema, dada a sua grande importância: nem sempre é possível reformular uma função recursiva, numa outra que não o seja. Existe um pequeno conjunto de funções primitivas, também extensível, entre o qual se destacam as funções trignométricas usuais e as funções exponenciais/logarítmicas:

```
• px = sin( tempo )          /* Descrição da posição de um objecto   */
• pz = cos( tempo )          /* cuja trajectória é uma espiral       */
• py = exp( tempo/100)       /* com andamento exponencial.           */
```

Interpolações e aproximações

Para os modelos de animação baseados em interpolações considerou-se um conjunto de expressões apropriadas, com o próprio nome de interpolações. Existem, actualmente, 3 tipos de interpolação e aproximação, mas o conjunto pode ser estendido sem implicações no resto do sistema. Para as interpolações existe a interpolação linear e os polinómios de Lagrange. Para as aproximações implementou-se a aproximação por B-Splines.

Uma interpolação é, de certa forma, semelhante a uma função. A partir dum dado valor (o valor independente) é calculado um resultado, baseado num vector de amostragens. O sistema permite, com base na noção de vector atrás descrita, que se calculem vários valores em função dum mesmo conjunto de valores da variável independente.

Assim, a partir dum conjunto de tuplos (vector)

$(x_1, y_{11}, ..., y_{1m})\ (x_2, y_{21}, ..., y_{2m})\ ...\ (x_n, y_{n1}, ..., y_{nm})$ com $x_1 < x_2 < ... < x_n$

podem ser calculadas m interpolações com base nos mesmos valores de **x**. A construção duma expressão para interpolação passa pela indicação dum vector de pontos e de qual a coordenada desses pontos que irá ser considerada. As interpolações são sempre efectuadas como função da primeira coordenada dos pontos indicados. Adicionalmente, é necessário referir o parâmetro da interpolação. Este parâmetro é, por sua vez, uma expressão e corresponde ao valor da variável independente para o qual se deseja conhecer o valor de uma das variáveis dependentes.

Vejamos alguns exemplos de aplicação das interpolações, assim como a sua sintaxe:

```
• p0 = < 0, 0, 0, 0 >          /* No instante 0 o objecto está em (0, 0, 0)     */
• p1 = < 25, 0, 0, 10 >        /* No instante 25 o objecto está em (0, 0, 10)   */
• p2 = < 50, 10, 0, 10 >       /* No instante 50 o objecto está em (10, 0, 10)  */
• p3 = < 75, 10, 0, 0 >        /* No instante 75 o objecto está em (10, 0, 0)   */
• p4 = < 100, 0, 0, 0 >        /* No instante 100 o objecto está em (0, 0, 0)   */
• v1 = { p0, p1, p2, p3, p4 }  /* Definição do vector de interpolação           */
```

/* Interpolação das coordenadas **x** e **z** ( 2 e 4 de cada um dos pontos ) */

```
• ex = interpolar( linear, v1, tempo, 2 )   /* a var. ind. é dada pelo tempo */
• ez = interpolar( linear, v1, tempo, 4 )   /* a var. ind. é dada pelo tempo */
```

## Condições

Para uma construção genérica de funções por ramo, criou-se a entidade condição. Uma condição não é mais do que uma expressão booleana. As condições são construídas a partir de relações entre expressões ou através de operações sobre outras condições. Temos assim definidos dois tipos de operadores para a construção destas entidades: os operadores lógicos e os operadores relacionais. Os primeiros operam sobre condições, enquanto que os segundos operam sobre expressões.

### Operadores lógicos

Os operadores lógicos existentes correspondem à implementação das usuais operações lógicas: a disjunção, a conjunção e a negação.

Considere que **a**, **b** e **c** são condições. Também o são o resultado das seguintes operações:

```
• a and b or c
• a and not b
```

### Operadores relacionais

Os operadores relacionais permitem a construção de condições a partir de expressões reais. Surgem, portanto, os operadores $\leq$, $<$, $=$, $\neq$, $>$ e $\geq$ que comparam valores de duas expressões. Existe ainda um operador para testar a inclusão de um elemento num dado domínio.

Sejam **a**, **b** e **x** expressões e **dom** um domínio. Exemplos de condições são:

- `@c1 = ( a + b ) ≥ x`
- `@c2 = not @c1 and ( x in dom ) or @c3`
- `@c2 = false`

## Funções

Como já foi referido, este sistema permite que se criem funções para tornar mais flexível a construção de expressões matemáticas. Estas funções são definidas tendo como domínio $R^n$ ($n \geq 1$); ou seja, uma função pode ter um número qualquer de parâmetros* . Como acontece com as expressões, o contra-domínio das funções é R.

A implementação prevê dois tipos de funções: as simples e as compostas. A diferença consiste, apenas, na forma como é calculado o resultado da função.

### Funções Simples

Na sua essência, as funções simples são expressões. A única diferença reside no facto de poderem ser feitas referências aos seus parâmetros. A avaliação duma função simples, consiste apenas na avaliação da expressão que a compõe (tendo em conta o valor dos seus parâmetros).

- `x1( t ) = interpolar( linear, vect1, t, 2 )`
- `pos( u, v ) = sin( u ) * cos( v ) + 10`

### Funções Compostas

As funções compostas são constituídas por um conjunto de ramos. Por ramo entende-se um par ( condição, expressão ). Um ramo é considerado aplicável se, face aos valores dos parâmetros, a condição que o define tiver o valor Verdade. O seu valor é o valor da expressão associada.

Após a determinação de qual (ou quais) os ramos aplicáveis, o valor duma função composta é dado pelo valor desse mesmo ramo. Havendo mais do que um ramo aplicável, numa dada situação, o resultado é não determinista. Ou seja, o seu valor depende da implementação, pelo que estas situações são de evitar.

As funções compostas permitem definições híbridas de trajectórias ou outras propriedades do sistema. Troços de uma trajectória podem ser definidos por expressões algébricas, enquanto outros podem ser o resultado da interpolação de certas posições pré-definidas. Segue-se um exemplo de função, neste caso a função factorial.

- `fact2(x) = x * fact(x-1)`
- `fact(x) = {`
  `x ≤ 0 ? 1 ;`

* Funções sem parâmetros reduzem-se a simples expressões.

```
    x > 0 ? fact2(x)
}
```

## Exemplos de Aplicação

Vejamos, de seguida, dois exemplos de animação, criada pelo protótipo, e que demonstram outros tantos tipos de animação: por interpolação e cinemática.

O objectivo da primeira animação é fazer percorrer um objecto ao longo de uma trajectória, definida pela interpolação de 4 pontos. O objecto deverá rodar em torno de si mesmo, sendo o eixo de rotação paralelo ao eixo dos yy. O sentido da rotação é o sentido inverso. Conhecem-se as posições do objecto nos instantes t=0, t=33, t=66 e t=99. Com base nisto, vamos definir o vector de interpolação e os respectivos pontos:

```
• p0 = < 0, 0, 0, 10 >          /* t=0  ⇒  (0, 0, 10)   */
• p1 = < 33, 10, 0, 5 >         /* t=33 ⇒  (10, 0, 5)   */
• p2 = < 66, 13, 0, 13 >        /* t=66 ⇒  (13, 0, 13)  */
• p3 = < 99, 20, 0, 5 >         /* t=99 ⇒  (20, 0, 5)   */
• v1 = { p0, p1, p2, p3 }       /* vector de interpolação */
```

A posição do objecto pode ser obtida interpolando sobre o vector v1 da seguinte forma:

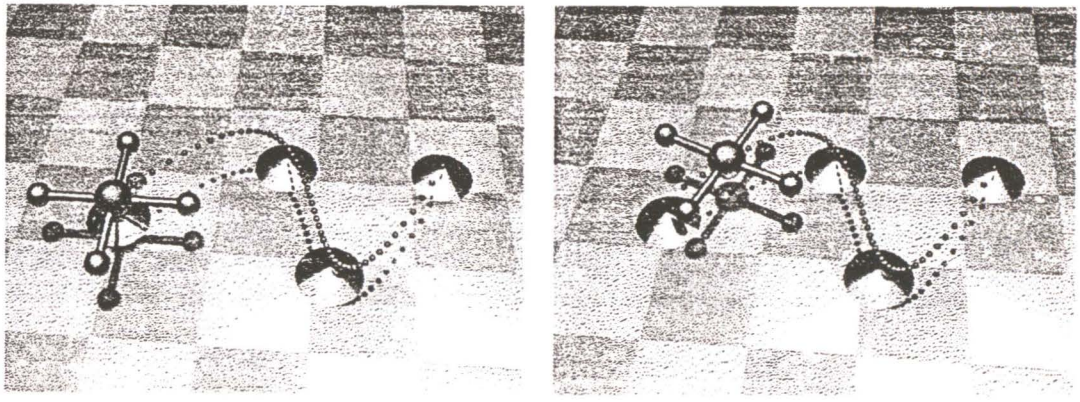
```
• e1 = interpolar(lagrange,v1,tempo,2)   /* a 2ª coordenada é o valor de x*/
• e3 = interpolar(lagrange,v1,tempo,4)   /* a 4ª coordenada é o valor de z*/
```

Não é necessário interpolar a coordenada y visto ela manter-se constante ao longo do tempo.

Considerando que o nome do objecto é **bolas**, a descrição do seu movimento seria dada pela seguinte declaração:

```
• bolas metal {escala 1.5, 1.5, 1.5 roty -tempo*16 transl e1, 1, e3}
```

O objecto tem o seu centro definido na origem. Como a escala é efectuada em primeiro lugar, a única consequência desta transformação será uma ampliação do objecto de 50%. De seguida aplica-se uma rotação, dependente do tempo, que permite orientar o objecto, imprimindo-lhe um movimento rotativo, em torno de si mesmo, durante toda a animação. Por último coloca-se o objecto no plano y=1, sendo a trajectória dele neste plano definida pelas interpolações já mencionadas. O resultado é o que se pode ver na Figura 3. Nela pode ver-se, ainda, a trajectória do objecto ao longo de toda a animação. Uma maior densidade das esferas que marcam a trajectória significa uma menor velocidade no movimento do objecto. Os cones com o vértice orientado para cima definem os pontos por onde o objecto deverá passar nos instantes atrás mencionados.



**Exemplo de animação por interpolação (key-frame)**

**Figura 3**

O segundo exemplo consiste numa animação cinemática. O movimento que se pretende ilustrar é o de um pêndulo. Este pêndulo faz parte de um tradicional relógio de mesa. Considere que o ângulo descrito pelo pêndulo é directamente proporcional ao seno duma função linear do tempo, sendo a sua amplitude máxima de 25 graus. O pêndulo move-se no plano yz e o ponto sobre o qual ele deverá rodar situa-se em (0, 0.175, 0).

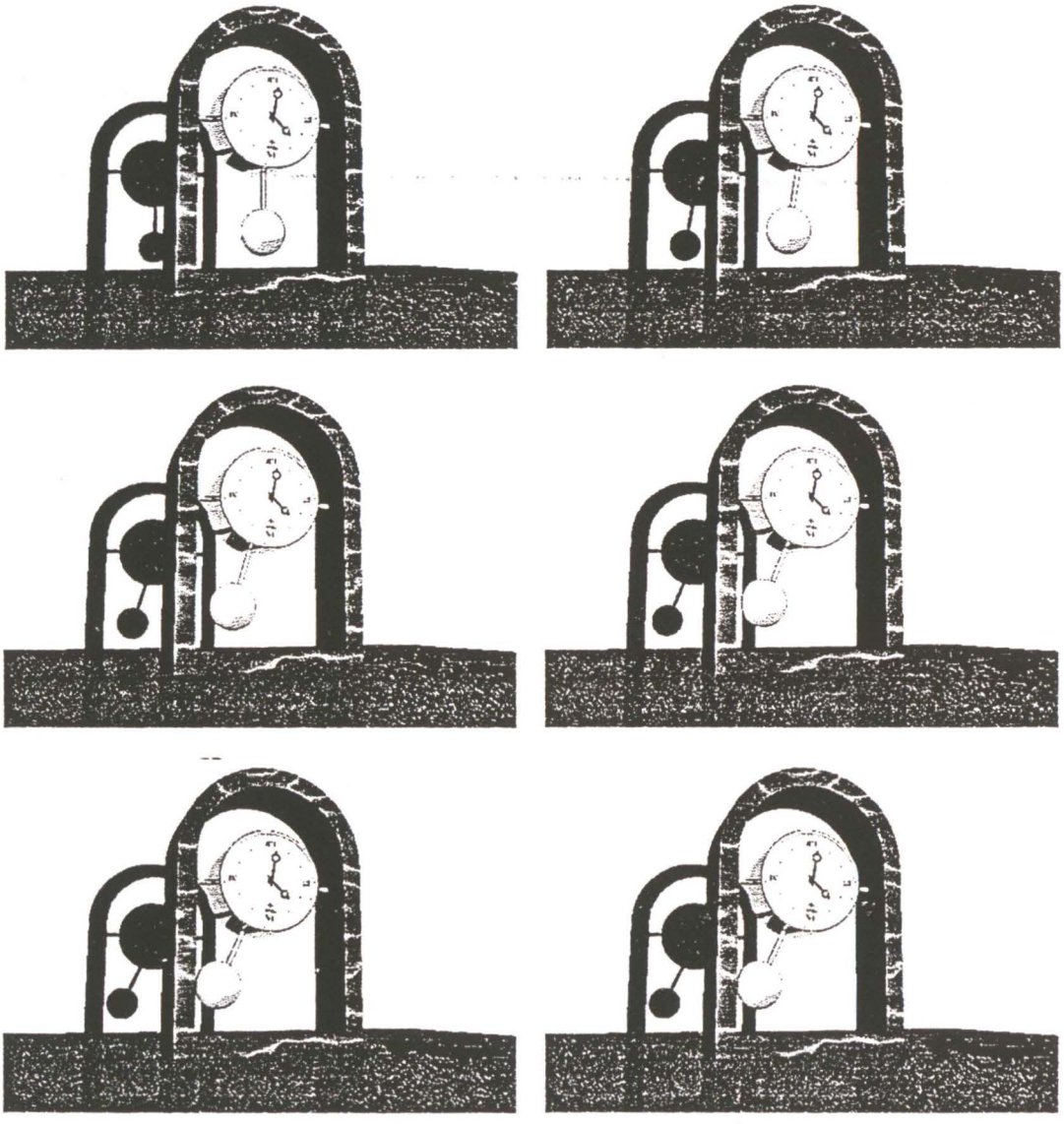
A primeira transformação a efectuar sobre o pêndulo consiste em deslocá-lo para a origem, efectuando uma translação negativa de 0.175 unidades em y. De seguida efectua-se a rotação, assumindo que o o ângulo é dado pela função f(t) = t*20, instanciando o argumento com o valor do tempo. Por último, resta colocar o centro de rotação na sua posição original efectuando uma nova translação, desta vez, simétrica à anterior.

Vamos então definir a função e a expressão que traduz o ângulo de rotação:

```
• f(t) = t * 20
• angulo = 25 * sin(f(tempo))          /* o ângulo de rotação      */
```

Resta-nos colocar o pêndulo na cena. Como ele faz parte de um objecto composto, o objecto a colocar é o relógio. O resultado está ilustrado na sequência de imagens da Figura 4.

As imagens foram obtidas para os instantes t=0, t=1, t=2, t=3, t=4 e t=5 Não se pretendia obter um movimento suave, mas, se esse fosse o caso, a função linear do tempo poderia ter um factor menor, de modo a que o pêndulo mudasse de posição mais lentamente em relação ao tempo.



Exemplo de animação cinemática

Figura 4

Eis a declaração resumida do relógio:

```
objecto pendulo {
   cilindro { escala 0.002, 0.09, 0.003 transl 0, 0.13, 0 }
   esfera { escala 0.003, 0.025, 0.025 transl 0, 0.006, 0 }
}

objecto relógio {
   base cinza, mármore { }
   suporte { }
   pendulo { transl 0, -0.175, 0 rotx angulo transl 0, 0.175, 0 }
   maquina { }
}
```

E agora a sua inclusão no nosso mundo (cenário):

```
relogio dourado { }
```

Os nomes **dourado** e **mármore** representam texturas. O identificador **cinza** representa uma superfície. Os outros objectos que constituem o **relógio** são: o **suporte**, e a **máquina**.

# Conclusões

Este trabalho tem como principal objectivo a criação dum modelo teórico de animação por computador. O modelo base possui a flexibilidade necessária para, a partir dele, se poderem criar sistemas bastante elaborados, bem como mais específicos. A um nível mais elementar, pode-se considerar este modelo como completo. No entanto, para aplicações mais específicas, o sistema pode ser rodeado por um conjunto de operações a um nível mais elevado.

A principal preocupação é a criação dum modelo que permitisse a evolução separada dos diversos espaços, mantendo a sua integridade. Como o Espaço das Expressões constitui o fulcro de todo o sistema, seria nele que se iriam descrever as entidades de forma mais abstracta e mais orientadas para a aplicação concreta.

A construção dum protótipo provou a viabilidade prática da definição de animações de diversos tipos, usando inclusivamente diversas plataformas de hardware/software.

Neste trabalho colaboraram também Diogo Ferreira da Costa, tanto na fase de concepção, como na da codificação inicial em ambiente NeXT™, e João Próspero Luís, na adaptação do protótipo ao ambiente OSF Motif.

# Referências


[Ary86] Arya, K. (1986). *A Functional Approach to Animation.* North-Holland Computer Graphics Forum, 5, 297-312.

[Dua91a] Duarte, V., & Duarte, S. (1991). *"Ray" o ray-tracer* . Relatório Interno, Faculdade de Ciências e Tecnologia da UNL.

[Dua91b] Duarte, V. (1991). *Descrição de objectos num "ray-tracer" por uma hierarquia de primitivas geométricas fixas.* Actas do IV EPCG, 203-214

[Fol90] Foley, J. D., van Dam, Andries, Feiner, Steven K., & Hughes, John F. (1990). *Computer Graphics - Principles and Practice* (2. ed.). Addison-Wesley Publishing Company.

[Heg89] Hegron, G., Palamidese, P., & Thalmann, D. (1989). *Motion Control in Animation, Simulation and Visualization.* North-Holland Computer Graphics Forum, 8, 347-352.

[Kal92] Kalra, D., & Barr, A.H. (1992). *Modeling with Time and Events in Computer Animation.* Proc. of the EG'92, 45-58, Blackwell Publishers.

[Mag88] Magnenat-Thalmann, N., & Thalmann, D. (1988). *A Classification of Computer Animation Methods.* Tutorial notes on Image Synthesis and Computer animation

[Pue88] Pueyo, X., & Tost, D. (1988). *A Survey of Computer Animation.* North-Holland Computer Graphics Forum, 7, 281-300.

[Tha86] Thalmann, D., & Magnenat-Thalmann, N. (1986). *Artificial Intelligence in Three-Dimensional Computer Animation.* North-Holland Computer Graphics Forum, 5, 341-348.

[Vei91] Veiga, P. (1991). *Autómatos Celulares - simulação multidimensional multi-informativa.* Actas do IV EPCG, 321-330