



Computação Gráfica I

Técnicas de Animação Gráfica I

Jorge Mota, Eng^o

Versão 1.8

Apontamentos para Apoio às aulas

Instituto Superior de Tecnologias Avançadas do Porto

Março/Abril/Maio/Junho de 2007

Resumo

A computação gráfica é um fenómeno ubíquo (omnipresente) no estado actual de uso das ciências da computação e sistemas de informação. Os **interfaces gráficos** massivamente usados em todas as classes de dispositivos, do computador ao leitor mp3, da consola de última geração ao painel de bordo de um automóvel, da **industria mundial de entretenimento** (que tem como maior segmento o dos videojogos) da **medicina** (com as maravilhas tecnológicas que permitem exames ao interior do corpo de forma não invasiva), à comunicação social com o uso extensivo de **efeitos visuais, à internet** e a sua cada vez mais exigente composição gráfica, e centenas de outras aplicações, que tornam este domínio das ciências da computação uma área de saber fundamental para qualquer profissional da engenharia multimédia e informática.

Nesse contexto e como suporte às aulas da disciplina de Técnicas de Animação I, elaborei, compilei e anotei estes elementos que não pretendem ser substituto da bibliografia recomendada no plano de estudos da disciplina, mas meros elementos de apoio de leitura rápida.

Jorge Mota, Março de 2007

Abstract

The graphical computation is a ubiquitous phenomena thing in the current state of use of the computer sciences and Information Science. The World-wide use of graphical interfaces in all kinds of devices, from computers to mp3 devices, from last generation game machines to hi-tech control panels in cars, the massive use of videogames, the marvels of technical medical services, the extensive use of special Effects in films and television, the new services in internet and web, and hundreds of other applications, become this as a domain of the computer sciences essential to a portfolio of skills and knowledge for any computer science and multimedia engineering professional and student. In this context, and as a support to the lessons of Técnicas de Animação I, I elaborated compiled and wrote down these elements to help students to learn the important topics of this field but do not intend to be substitute of the bibliography recommended in the plan of studies of discipline, but mere elements of support of fast reading.

Jorge Mota, Março de 2007

Conteúdo

Resumo.....	2
Abstract	3
1 Introdução à computação gráfica	5
2 Matemática aplicada a computação gráfica	6
3 Ambiente de desenvolvimento algorítmico Raptor	35
4 Estruturas de dados para computação gráfica	38
5 Ambiente de desenvolvimento gráfico Raptor/RaptorGraph	39
6 O ambiente de desenvolvimento Flash 8.0.....	40
7 O Introdução ao ActionScript 2.0	41
8 Representação matricial e vectorial de imagens	61
9 Projecto 1: Tratamento de imagem em ActionScript (Editor de imagem).	72
10 Conceitos sobre física do movimento	75
11 Usabilidade e interacção (ActionScript 2.0).....	83
12 Luzes	84
13 Oclusão: Ordem de visualização (Backface culling e Depth Sorting)	85
14 Projecto 2: Jogo clássico para PSP em Flash Lite	86
15 88	
16 ActionScript – Implementação do Jogo PONG	88
17 Bibliografia	100

1 Introdução à computação gráfica

Todos nós desde muito cedo aprendemos a importância de um gráfico para nos ajudar a compreender a relação entre grandezas. Aprendemos na nossa formação de base a fazer e ler gráficos, e a nossa vida diária está completamente inundada de situações em que temos que interagir com gráficos. Um gráfico de uma função $y=f(x)$, ou o gráfico de barras de uma tabela de valores $(x_i, y_i), i=1..n$ revela aspectos de relação dificilmente discerníveis via análise de formulas ou através de valores numéricos [7].

Um gráfico é uma imagem visual criada usando-se grafismos (pontos e linhas, etc)[7]. As aplicações para os gráficos sejam eles vectoriais ou bitmap, a 1, 2 ou mais dimensões são muito vastas. Neste curso (disciplina) vamos procurar dar exemplos em domínios diversos, mas com um enfoque especial na animação e aplicações para a web e videojogos.

2 Matemática aplicada à computação gráfica

Esta introdução teórica destina-se àqueles que não tem preparação em conceitos básicos de matemática e geometria, ou então aqueles que já esqueceram estes temas e precisam agora de reavivar a memória.

Podemos definir uma imagem em computação gráfica de duas maneiras diferentes:

- a) Imagem: um conjunto de objectos posicionados em suas respectivas coordenadas e projectados num plano.
- b) Imagem: um conjunto de pontos ordenados, com suas cores.

A interpretação de ambas as definições exige prévio conhecimento de alguns conceitos matemáticos e geométricos, os quais serão sucintamente revistos neste capítulo[7].

Gráficos 2D versus gráficos 3D

A principal diferença entre uma imagem bidimensional para uma tridimensional é a sensação de profundidade, que existe nesta última. A realidade é que em ambos os casos a imagem física, tangível é sempre uma representação a duas dimensões. Uma imagem a 3D não é mais do que uma projecção de objectos a 3D numa imagem bidimensional [7].

Considerámos as seguintes três representações para um cubo:

<<inserir imagem de três cubos representados em modelo wireframe, superfícies e sólido>>

As três imagens estão desenhadas em papel a 2 dimensões (a folha de papel) . Mas a forma de representação dá-nos a sensação de tridimensionalidade. Nos casos apresentados respectivamente designados de representação wireframe, superfícies e sólida a sensação de volume só é significativamente sentida da segunda e terceira e a sensação de distância a

um plano só e visível na terceira. Todas estas sensações de tridimensionalidade são interpretações do nosso cérebro cujas complexas capacidades de análise de imagem nos leva a sentir a terceira dimensão. Sugestão para actividade complementar: Criem uma foto com relevo usando a seguinte regra muito simples.

- Tirem duas fotografias uma com a câmara posicionada num dos olhos e outra do outro olho – a câmara deve mover-se paralela ao objecto a fotografar.
- Usem o programa callipygias (<http://callipygian.com/3D/>) para compor as duas fotos para que possam ver com uns óculos com um filtro vermelho e um azul.
- Observem sem óculos a fotografia
- Agora observem com os óculos com filtro vermelho e azul.

Fantástico não é, o cérebro prega cada partida' !

Representação vectorial e matricial de imagens

REPRESENTAÇÃO VETORIAL E MATRICIAL DE IMAGENS

Um vector é basicamente um segmento de recta orientado. Podemos pensar num vector 2D, V , como uma seta que vai da origem do sistema de coordenadas, para o ponto (x,y) , tendo assim uma direcção, um sentido e um comprimento especificado (ver tópico seguinte pontos e vectores). [1]

Uma matriz é um arranjo (array) de elementos em duas direcções. Quando trabalhamos, primeiro definimos quantos elementos existem em cada direcção, uma matriz 4x4, por exemplo, é do tipo:

1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1

A representação vectorial das imagens é principalmente empregue, em computação gráfica, para a definição e modelação dos objectos sintéticos que serão representados pela imagem.[1]

Na representação vectorial das imagens, são usados como elementos básicos os pontos, as linhas, as curvas, as superfícies tridimensionais ou mesmo os sólidos que descrevem os elementos, que formam as imagens sintéticas no computador.

Esses elementos são denominados primitivas vectoriais da imagem. As primitivas vectoriais são associadas a um conjunto de atributos que define sua aparência e a um conjunto de dados que define sua geometria (pontos de controle).

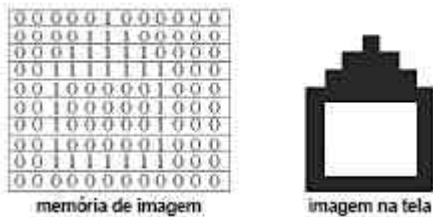
Para esclarecer melhor, vamos considerar alguns exemplos. Dois elementos facilmente caracterizados como vectoriais, pela noção de vectores já discutida são os pontos e linhas rectas.

A cada elemento de um conjunto de pontos associa-se uma posição, que pode ser representada por suas coordenadas (geometria), e uma cor, que será como esses pontos aparecerão na tela (atributos). No caso de um conjunto de linhas retas, cada uma pode ser definida pelas coordenadas de seus pontos extremos (geometria) e sua cor, espessura, ou ainda se aparecerá pontilhada ou tracejada (atributos).

A descrição matricial é típica das imagens digitalizadas capturadas por scanners ou utilizadas nos vídeos. É a forma de descrição principal na análise e no processamento de imagens. Em computação gráfica sintética, surgem nos processos de finalização (ray tracing, z-buffers).

Na representação matricial, a imagem é descrita por um conjunto de células num arranjo espacial bidimensional, uma matriz. Cada célula representa os

pixels (ou pontos) da imagem matricial. Os objetos são formados usando adequadamente esses pixels. A Figura explica melhor as formas de descrição de imagens matricial. Essa é a representação usualmente empregada para formar a imagem nas memórias e telas dos computadores e na maioria dos dispositivos de saída gráficos (impressoras e vídeos).[1]



ARQUITECTURA DE SISTEMAS gráficos

Os sistemas para computação gráfica precisam de alguns dispositivos gráficos de entrada e saída (In/Out ou I/O) ligados a um computador. Assim, os dispositivos gráficos são elementos críticos de um sistema de computação gráfica. Através dele interagimos com o sistema na busca de uma extensão dos limites de nosso corpo e uma melhor comunicação com a máquina.

Dispositivos Gráficos de Entrada

Os dispositivos de entrada são componentes electrónicos que permitem a movimentação e interacção com os sistemas. A cada dia surge um novo dispositivo com novas propostas ergonómicas, recursos adicionais que agilizam a tarefa de interacção ou simplesmente reduzem a quantidade de fios em sua mesa. Dentre os dispositivos mais usados na computação gráfica podemos citar:

Teclado: Basicamente podemos definir um teclado como um conjunto de teclas associadas a um código que corresponde ao caractere ou função. Diversos dispositivos de teclado foram inventados ao longo de décadas, porém o mais usado é o teclado QWERTY. É irônico pensar que esse teclado foi inventado para a redução da velocidade do digitador e, como consequência, causar menores danos à sua saúde.

Mouse: Os mouses atuais utilizados por profissionais da computação gráfica são compostos por sensores ópticos e processadores digitais para scanear a superfície sob o mouse sem a bola de rolagem. Enviam 1.500 sinais por segundo para rastrear com segurança o menor movimento possível.

Joysticks: São alavancas de comando que determinam a direcção e velocidade do cursor na tela. São usados geralmente nos jogos de videojogos, estações de realidade virtual e estações industriais de controle de robôs.

Tablet: Os tablets são extensões dos monitores sensíveis ao toque. Fruto de anos de pesquisas sobre como os profissionais realmente trabalham, os novos tablets são calibrados com perfeição para ler com absoluta precisão os movimentos da caneta, que opera com 1.024 níveis de sensibilidade à pressão. Estes são traduzidos em curvas suaves, transições graduais e controles precisos do traço. Um software incluso nos pacotes dos hardwares de tablet possibilita o reconhecimento da escrita.

Mesa Digitalizadora: Dispositivo vectorial que consiste de uma mesa e de um apontador. A cada vez que o usuário toca a mesa com o apontador é informado ao computador a coordenada deste ponto da mesa. Existem diversos trabalhos em andamento para a substituição deste periférico por sistemas mais baratos com câmaras digitais softwares de reconhecimento de padrões.

Dispositivos de Entrada 3D

Com a popularização dos sistemas 3D (na maioria dos jogos), e o barateamento dos componentes electrónicos, começaram a surgir uma diversidade de dispositivos de entrada 3D. Em qualquer casa de jogos electrónicos, e não raramente em lojas de produtos de informática,

podemos ver simuladores de esqui, skate entre outros. Basicamente, esses dispositivos permitem a movimentação e interacção dentro de um espaço 3D qualquer.

Digitalizador Tridimensional: Trata-se de um dispositivo vetorial e consiste em uma espécie de braço mecânico com um sensor de toque na ponta. A cada vez que o sensor atinge um ponto na superfície de um objeto, a coordenada deste ponto em relação a um ponto referencial (origem) é transmitida ao computador.

Scanners Tridimensionais: Existem diversas tecnologias de scanners disponíveis no mercado. As mais baratas utilizam câmaras digitais acopladas a uma mesa especial que fornece as coordenadas para os sistemas. Esta tecnologia quase sempre requer a intervenção de modeladores para o acabamento das peças. A tecnologia de scanners a laser, de alto custo, é sem dúvida a tecnologia de dispositivos de entrada que vem atraindo mais atenção no mundo. Suas aplicações são grandes e muitas delas ainda estão por se descobrir.

Luvas: Luvas para interacção 3D são dispositivos que, através de sensores, detectam e medem as flexões e pressões dos dedos. Os sensores podem ser mecânicos, ópticos ou híbridos.

Capacetes: Existem diversos tipos de capacetes para visualização de Realidade Virtual disponíveis no mercado. A principal característica desses equipamentos é que podem ser: Estereoscópios ou monoscópios (isto é, usados com uma ou duas cenas); Binoculares ou monoculares (um ou os dois olhos são estimulados); Opacos ou translúcidos (substituem ou complementam a realidade).

3D Controllers: São dispositivos para interactividade com ambientes 3D capazes de tornar o ambiente participativo, seguindo os movimentos executados pelo usuário. Trabalham tanto em cima de uma mesa como no ar, pois possuem um giroscópio que tem comunicação por rádio com o

computador. São capazes de medir a velocidade e a força que estão sendo aplicadas pelo usuário.

Roupa de RV: A roupa para Realidade Virtual (ou data suits) é uma indumentária que permite a interação com o mundo virtual. A comunicação pode ser realizada de várias maneiras, sendo que o acompanhamento óptico de marcadores vem sendo o mais utilizado. Essas roupas são usadas para gerar informações do movimento humano, a partir daí surge uma infinidade de aplicações para animações, esporte, desenvolvimento de produtos, medicina etc.

Dispositivos Gráficos de Saída

É possível classificar os dispositivos de saída em duas principais categorias, segundo a forma pela qual as imagens são geradas (veja secção anterior de descrição vectorial e matricial de imagens): vectoriais e matriciais. Os dispositivos vectoriais conseguem traçar segmentos de recta perfeitos entre dois pontos. Os dispositivos matriciais apenas conseguem traçar pontos, ou seja, segmentos de recta são traçados como sequências de pontos próximos, são entretanto, bastante adequados para desenhar áreas cheias e sombras, onde os vectoriais mostram deficiência.

Impressoras de Jato de Tinta: São equipamentos matriciais com cabeçote que ejectam tinta sobre o papel. Podem utilizar tintas de várias cores e chegar a níveis altos de realismo na imagem impressa.

Impressoras Laser: São as que têm melhor qualidade. Um feixe de raio laser varre uma chapa em processo óptico parecido com o do cabeçote de uma impressora, o bombardeio do feixe deixa a chapa carregada com uma carga electrostática. Por efeito da atracção eléctrica, uma tintura (toner)

adere à chapa e por pressão e aquecimento é fixada no papel formando a imagem.

Impressoras Térmicas: São equipamentos silenciosos, com boa resolução, podem trabalhar com ampla gama de cores. As impressoras térmicas precisam utilizar um papel termo-sensível especial.

Plotters: São dispositivos vectoriais e eletromecânicos que, de uma forma geral, produzem o desenho pelo movimento de uma caneta na superfície do papel. Existem dois tipos, em um, o papel permanece fixo e a caneta produz desenhos sobre o mesmo pela combinação de movimentos horizontais e verticais. No outro tipo, o desenho é produzido pela combinação dos movimentos do papel e da caneta.

Monitores: A maioria dos modelos atuais se baseia na tecnologia de tubos de raios catódicos (CRT – Catode Ray Tube), já madura e capaz de oferecer uma boa relação custo/benefício, para produzir imagens de qualidade em computadores pessoais. Mas dentro de poucos anos, encontrar um monitor CRT em uma loja poderá ser quase impossível. Em várias partes do mundo, já é difícil encontrar um modelo CRT nas lojas. A Apple Computer, por exemplo, aboliu os monitores CRT de seus sistemas. Os fabricantes NEC Mitsubishi e Hitachi também já deixaram de produzir modelos CRT. A queda drástica nos preços dos monitores LCD (Liquid Cristal Displays), seu pouco peso e espessura são as principais causas dessa derrocada. Porém, os CRTs têm ainda uma vantagem substancial em relação aos LCDs no que diz respeito ao brilho e à versatilidade em resolução. Monitores CRT: Até há pouco tempo era praticamente o único tipo de vídeo utilizado. A resolução máxima com a qual um monitor CRT pode trabalhar depende de sua habilidade física em focar o feixe de elétrons sobre os pontos de fósforo monitores CRT são compostos por um canhão que gera um feixe de elétrons. Um aquecedor é utilizado para

liberar elétrons de um cátodo, razão pela qual os monitores demoram um pouco para apresentar a primeira imagem depois de ligado. Esses elétrons são atraídos por ânodos (cargas positivas) próximos à parte da frente do monitor (Figura 1.3). O feixe de elétrons percorre um caminho da esquerda para a direita e de cima para baixo, orientado por diversos componentes chamados bobinas deflectoras. Ao atingir a extremidade direita da tela, o feixe é desligado para retornar à extrema esquerda da linha inferior e, quando atinge a extremidade de baixo, também é desativado para retornar novamente à primeira linha. Esse processo é chamado de varredura. Aumentando ou diminuindo a intensidade do feixe, consegue-se controlar o brilho dos pontos de fósforo da tela para gerar a imagem. A velocidade com que o feixe percorre toda a tela é chamada de taxa de renovação (refresh rate) ou frequência de varredura vertical. O padrão antigo para monitores determinava que a taxa de renovação ideal era de 60 Hz, mas um novo modelo desenvolvido pela VESA (Video Electronics Standards Association) recomenda a frequência de 75 Hz para monitores trabalhando com resolução de 640 por 480 pixels ou maior. Quanto maior a taxa de renovação, menos sensível é o fenómeno de cintilação (flicker). Para oferecer maior resolução, sem que o custo do monitor se elevasse muito, foi criada a técnica de entrelaçamento. Nos monitores entrelaçados, o canhão de elétrons renova apenas metade das linhas em uma passada (por exemplo, apenas as linhas ímpares e num passo e, no seguinte, as linhas pares). Como apenas parte das linhas é refeita por vez, é possível apresentar o dobro de linhas por ciclo de renovação, aumentando consequentemente a resolução vertical oferecida pelo monitor. Em outras palavras, os modelos entrelaçados podem oferecer a mesma resolução que um não-entrelaçado, mas a um custo menor. A desvantagem dessa técnica fica por conta do tempo de resposta menor, o que pode ser crítico em aplicações de animação e vídeo, e do possível efeito de cintilação. Monitores CRT Coloridos: A cor da luz emitida vai depender do fósforo usado. Os monitores monocromáticos, mais simples, produzem imagens na cor verde, branco ou âmbar e, durante muito tempo, foram os únicos a oferecer custo acessível para o usuário de computadores de mesa.

Os monitores coloridos tornaram-se populares no início da década de 1980. Esses modelos usam o padrão RGB (Red, Green e Blue), um sistema de representação de todas as cores com base no vermelho, verde e azul (Capítulo 5). Para gerar qualquer cor do espectro, os monitores coloridos precisam de três sinais separados, que vão sensibilizar, respectivamente, os pontos de fósforo das três cores primárias, suficientemente pequenos para parecer ao olho humano como um único ponto de luz.

Além da resolução máxima, outro factor que influencia na qualidade final da imagem gerada pelo monitor é o chamado dot pitch, que especifica a distância entre dois pontos de fósforo da mesma cor em trios RGB adjacentes. Esse valor é uma medida que também deve ser levada em conta para determinar a qualidade de um monitor. De nada adianta uma tela grande, com alta resolução, ou seja, muitos pontos RGB, se esses pontos estiverem muito distantes entre si, gerando uma imagem reticulada. Assim, quanto menor o dot pitch, mais pontos por polegada terá o monitor e maior sua capacidade de resolução máxima. O dot pitch é medido em milímetros, e os valores encontrados nos modelos de monitores mais comuns são de 0,28mm ou menores. Existe uma diferença substancial no modo como o dot pitch é medido entre os diversos fabricantes de monitores. Por essa razão, tais valores não devem ser comparados directamente. Algumas vezes, o dot pitch é medido como a distância entre dois pontos de fósforo da mesma cor em tríades adjacentes, na diagonal. Diversos fabricantes fazem a medida na horizontal, para fornecer um valor mais palpável ao consumidor. Há ainda quem meça a distância entre os orifícios na máscara e não entre os pontos de fósforo. Em monitores que se baseiam no aperture grill, a medição é feita entre duas tiras de fósforo da mesma cor.

Resolução: Um conceito estreitamente ligado ao tamanho da tela dos monitores é a resolução, que descreve a quantidade de informação que o equipamento pode apresentar em um determinado instante. A resolução é medida em pixels (palavra criada a partir da expressão em inglês picture element), uma unidade básica da imagem que pode ser controlada

individualmente e que contém informações sobre cores e intensidade (brilho). O pixel deve ser encarado como uma unidade lógica e não física.

O tamanho físico de um pixel vai depender de como a resolução da tela foi configurada. Assim, se estiver sendo usada a resolução máxima, um pixel será equivalente a uma unidade física do monitor. Porém, para resoluções menores do que a máxima, um pixel será composto por mais de um ponto físico da tela. A resolução pode ser descrita pelo número de pixels (pontos individuais de uma imagem) apresentados na tela de um monitor, expressos nos eixos horizontal e vertical. A forma de uma imagem na tela depende da resolução e do tamanho do monitor. A mesma resolução produzirá uma imagem de melhor qualidade, menos reticulada, em um monitor menor, e perderá gradualmente a forma, à medida que forem usados modelos maiores. Isso acontece porque o mesmo número de pixels terá de ser espalhado por uma área maior da tela. O padrão actual de resolução adotado por grande parte das páginas na Web é de 800 por 600 pontos, para citar um exemplo conhecido pela maioria dos usuários de computador. Já programas baseados em janelas (windows) costumam adoptar resoluções de 1.024 por 768 pixels. A principal vantagem de usar resoluções maiores é a redução da necessidade de ampliar a imagem (usando recursos de zoom), pois mais informação da imagem é apresentada de uma só vez.

Existe uma relação estreita entre a resolução usada e o tamanho do monitor. Resoluções muito altas em uma tela pequena podem resultar em problemas, pois alguns pacotes de software usam texto limitado a um número fixo de pontos. Desse modo, quando apresentado em telas pequenas configuradas com alta resolução, o texto aparecerá muito reduzido para propiciar uma leitura confortável. Por outro lado, aumentar simplesmente o tamanho do monitor nem sempre é a solução indicada. Se a resolução não puder acompanhar o aumento da tela, o resultado será caracteres e imagens mais reticulados.

A relação entre o tamanho da tela e a resolução padrão (default) é mostrada nesta

tabela:

Medida nominal Resolução recomendada

14" 800 × 600

15" 800 × 600

17" 1.024 × 768

19" 1.280 × 1.024

21" 1.600 × 1.200

Monitores LCD: Os monitores de cristal líquido (LCD – Liquid Crystal Display) não possuem um canhão de elétrons, o que lhes confere uma série de vantagens e algumas desvantagens. Em substituição ao tubo de raios catódicos, existe um sistema de células contendo cristal líquido e filtros coloridos polarizados. As moléculas do cristal líquido (substância descoberta em 1888) expostas a campos eléctricos são alinhadas (fenómeno da polarização). Estas moléculas de cristal líquido ficam entre duas camadas de filtros formados por um conjunto de finíssimos fios dispostos em paralelo. Cada filtro tem os fios em posição perpendicular ao outro, levando as moléculas a formarem uma coluna “torcida” para passagem dos raios luminosos. A fonte de luz fluorescente que deverá passar pelo cristal líquido é denominada backlight. O objectivo é fazer com que esses raios luminosos passem pelas células contendo cristal líquido. Quando é aplicado um campo eléctrico aos cristais, vemos as moléculas arranjadas no sentido vertical, permitindo que os raios de luz passem por elas até encontrarem o segundo filtro. Como estão em posições defasadas um ao outro, os raios luminosos não passarão e não há geração de imagem. Ao contrário, sem tensão aplicada sobre o cristal líquido, os raios luminosos passarão pelo primeiro filtro, sofrendo realinhamento na coluna torcida de

crystal líquido contido nas pequenas células, alinhadas na tela, até atingir o segundo filtro, gerando a imagem. Em monitores LCD monocromáticos o pixel é formado por uma célula, enquanto os monitores LCD policromáticos (coloridos), cada pixel é formado por três células de cristal. Cada uma dessas células tem um filtro vermelho, verde ou azul, barrando a entrada da luz. Passando pelas células com as cores filtradas, a luz gerada pelo backlight resultará na imagem com as cores vistas na tela de um monitor LCD. As células fazem o papel dos pixels nos monitores do tipo CRT, e um pixel pode ser formado por uma ou mais células. No entanto, essas células não podem variar em suas dimensões. Nos monitores CRT, podemos aumentar ou diminuir, sempre proporcionalmente, o número de linhas e colunas, aumentando ou diminuindo o número de pixels. Nos monitores tipo LCD, isso não é tão simples, pois cada célula tem dimensões predefinidas. Ao alterarmos a resolução da imagem, provocamos a distorção da mesma. Isso ocorre porque não é possível fazer um aumento proporcional (altura e largura) da tela.

Monitores LCD de Matriz Passiva – Bastante utilizados inicialmente em notebooks monocromáticos ou não, hoje esta tecnologia é empregada na fabricação de telas para celulares, handhelds, notepads etc. Nessa tecnologia, a tela é formada por uma grade de fios condutores. Cada intersecção desses fios forma um pixel, e a corrente necessária para a polarização será transmitida por esses fios. O controle do brilho dos pixels é obtido aplicando-se uma corrente eléctrica forçando os cristais a se realinharem, ou seja, haverá alteração na direcção dos raios luminosos. Esse processo é repetido linha a linha de pixels, da parte superior à inferior da tela do monitor LCD até a corrente específica chegar à célula específica. Essa técnica tem o inconveniente de produzir cintilação, o que levou à utilização de um tipo de cristal com baixo tempo de resposta às diferentes correntes aplicadas sobre os cristais. Quer dizer que há uma demora maior que a desejada para o re-alinhamento dos cristais. Essa demora causa menor nitidez na geração de imagens, e se percebe mais na execução de arquivos de vídeo ou games.

Outra desvantagem é o efeito crosstalk, causado pela interferência do campo magnético de uma célula sobre células de cristal líquido vizinhas, alterando a imagem que deveria ser gerada. Daí surgiu a tecnologia Dual Scan Twisted Nematic – DSTN, para amenizar esse efeito negativo dos monitores de matriz passiva. Nela a tela é dividida em uma parte superior e outra inferior, com varreduras independentes. Para compensar essas dificuldades, alguns monitores de matriz passiva dispõem do recurso de endereçamento simultâneo para duas linhas diferentes.

Monitores LCD de Matriz Activa: Actualmente os novos monitores LCD utilizam a tecnologia de matriz activa, na qual um transistor especial – TFT Thin-Film-Transistor – alimenta cada célula individualmente. Neste caso, cada célula recebe uma corrente eléctrica, inferior à utilizada nos monitores de matriz passiva, independente das outras. Essa é a solução completa para os problemas existentes ao gerarmos imagens em sequência (filmes, games etc.) além do crosstalking. Como resultado, têm-se imagens mais nítidas com cores mais intensas e ausência da cintilação. Outra vantagem é a possibilidade de fabricação de telas com mais de 20". Em compensação, a fabricação desses monitores implica em algumas dificuldades. As telas de LCD são montadas sobre um substrato de vidro com um único chip (conjunto de transistores). Levando em conta que uma tela preparada para atingir resolução de 800 x 600 pixels pode conter mais de 6 milhões de transistores, as chances de haver transistores defeituosos não é pequena. A verdade é que toda tela LCD de matriz activa tem defeitos, e para descobri-los basta preenchê-la com fundo branco ou preto. Os pontos que mais se destacarem são os defeituosos.

Monitores See-through: Os monitores ditos see-through são aqueles em que é possível enxergar através do monitor. Esse tipo de aparato é muito usado em situações onde se precisa sobrepor uma imagem real a uma imagem gerada por computador ou vice-versa. É constituído de monitores do tipo LCD pois essa tecnologia permite que se veja através do monitor. O maior problema desse tipo de monitor é a pouca intensidade da imagem

exibida, o que muitas vezes prejudica a aplicação. Recentemente, esses monitores vêm sendo usados para visão noturna e aproveitamento de espaço em automóveis e veículos de combate. Uma forma alternativa à implementação desse monitor é utilizar um monitor convencional do tipo CRT, filmar a imagem do ambiente real (com uma pequena câmara presa à cabeça do usuário) e sobrepor esta imagem à imagem gerada pelo computador. Esse mecanismo permite uma melhoria significativa na qualidade da imagem mostrada, mas nem sempre será a solução dependendo do tipo de aplicação.

Displays de Retina: O Human Interface Technology Lab, da Universidade de Washington, desenvolveu um tipo de monitor onde o equipamento é um laser que exhibe as imagens directamente na retina do usuário. Por enquanto, o equipamento ainda é muito grande e com uma resolução máxima de 1000x1000 pontos monocromáticos. Esse equipamento tem despertado o interesse da comunidade científica por diferentes propósitos. Primeiro, porque ele permite que portadores de alguns tipos de doença da retina possam enxergar, já que o equipamento força a entrada da luz para dentro do olho. Segundo, porque a redução do seu custo poderia determinar o fim dos monitores.

Head Mounted Displays: Também conhecidos como “óculos de realidade virtual” ou capacetes de realidade virtual”, os HMDs (Head Mounted Displays) operam exibindo em duas pequenas telas (uma para cada olho) imagens de uma cena virtual. Os HMDs são construídos, normalmente, usando dois tipos de monitores: CRT ou LCD. Os monitores de CRT, em função da avançada tecnologia disponível nesta área, podem exhibir imagens de alta resolução com uma qualidade de cor excelente, mesmo em pequenas dimensões. Entretanto, são relativamente pesados, volumosos e colocam altas voltagens muito próximas à cabeça do usuário. Os monitores LCD, por sua vez, são leves e podem ser usados com pequenas voltagens. Entretanto, devido às limitações tecnológicas, a resolução disponível em monitores pequenos ainda é baixa. Acoplados aos HMDs, em geral existem

sistemas de rastreamento da posição da cabeça, a fim de permitir que se actualize as imagens do mundo virtual de acordo com a direcção para onde o usuário está olhando. Stereo Glasses ou Shutter Glasses: Os Stereo Glasses são uma extensão dos monitores "see-through". Útil em aplicações onde várias pessoas precisam observar a mesma imagem estereó, como visualização científica, cirurgias e passeios virtuais em parques de diversão, estes dispositivos, bem mais baratos do que os HMDs, buscam gerar imagens a partir de uma tela convencional de computador. A ideia é colocar nos usuários pares de óculos com lentes de cristal líquido capazes de bloquear a visão de um dos olhos quando necessário. A ideia básica desses dispositivos consiste em exibir na tela a imagem correspondente à do olho esquerdo e bloquear a visão do olho direito escurecendo a tela de cristal líquido, a seguir faz-se o contrário, ou seja, exibe-se a imagem do olho direito e bloqueia-se a visão do esquerdo. Cave: Os primeiros registros de simulações realizados pelo homem são da época das cavernas. Na pré-história, os guerreiros desenhavam nas cavernas cenas de batalhas e caça de animais. Usando uma tocha, iluminavam a sequência e passos para demonstrar às crianças e aos jovens como seria na prática. Daí vem o nome Cave (caverna) ou, mais recentemente, Surround-Screen Project-Based. Esses dispositivos

usam a ideia de colocar o usuário em uma sala com paredes que são na verdade telas para projecções de imagens.

Placas Aceleradoras de Vídeo: Os monitores interpretam sinais analógicos para apresentar imagens na tela. Para isso, o processador existente na placa de vídeo precisa transformar os sinais digitais em analógicos antes de enviá-los ao monitor. Nem todo o processamento de imagens é realizado pelo processador de vídeo. Parte desse trabalho é realizada pelo processador principal, mas quanto mais poderoso o processador de vídeo, menos sobrecarregado fica o processador principal, ficando disponível para efectuar outras tarefas. Processar imagens é, basicamente, fazer cálculos. Quanto mais complexa uma imagem, maior o

número de pontos que devem ser criados, ocorrendo o mesmo se desejarmos melhores resoluções de imagem. As placas aceleradoras 3D de uso profissional são normalmente optimizadas para trabalhar com OpenGL, DirectX e alguns softwares de modelagem. A indicação para uso profissional desse tipo de placa também foi reforçada pela Microsoft que implementou a compatibilidade nativa para OpenGL desde o Windows NT e 2000.

AGP: A porta AGP (Accelerated Graphics Port) segue como padrão mínimo para boas placas gráficas. É usada para conectar placas gráficas directamente para a CPU e para a memória principal. O modo AGP permite que as placas gráficas se comuniquem com a CPU e com a memória principal a taxas de dados de cinco a oito vezes mais rápido do que o barramento PCI.

Sistemas de coordenadas

PONTOS E VECTORES

Neste tópico vamos discutir aspectos relacionados com pontos e vectores.

Os conceitos explanados são válidos para os domínios 2D e 3D embora as explicações sejam apresentadas a 2D.

Os pontos e os vectores são entidades matemáticas diferentes. Os pontos não têm dimensão, e representam posições no espaço. Um vector por outro lado, não tem uma localização definida no espaço e os seus únicos atributos são a direcção e magnitude. Uma razão para que muitas vezes as pessoas confundam vectores e pontos é o facto de muitas vezes representarmos um vector v como estando associado a um ponto P , sendo que o vector aponta da origem para P . Ambos pontos e vectores são representados por pares (ou trios) de pontos, mas os números têm significados diferentes. Um ponto de coordenadas (3,4) é a localização 3 unidades a direita do eixo dos y e 4 unidades para cima do eixo dos x . Um vector com componentes (3,4) contudo aponta na direcção $4/3$ ou seja tem inclinação $4/3$ (move 3 na

direcção do eixo dos x e 4 na direcção do eixo dos y) e a sua magnitude é de $\sqrt{3^2 + 4^2} = 5$ e pode ainda estar localizado em qualquer lugar.

Em matemática as entidades estão sempre relacionadas com operações. Por tal facto vamos agora discutir as operações com pontos e vectores.

A primeira operação é multiplicarmos um ponto P por um numero real a. O produto aP é um ponto localizado na linha que une a origem ao ponto P (linha infinita) e o produto aP pode estar localizada em qualquer ponto desta linha dependendo do valor de a.

A próxima operação é a subtracção de dois pontos. Consideremos que $P_0 = (x_0, y_0)$ e que $P_1 = (x_1, y_1)$ são dois pontos. A diferença entre $P_1 - P_0 = (x_1 - x_0, y_1 - y_0) = (? x, ? y)$ é bem definida e é um vector (definido pela direcção entre os pontos P_0 e P_1 e a distância entre ambos).

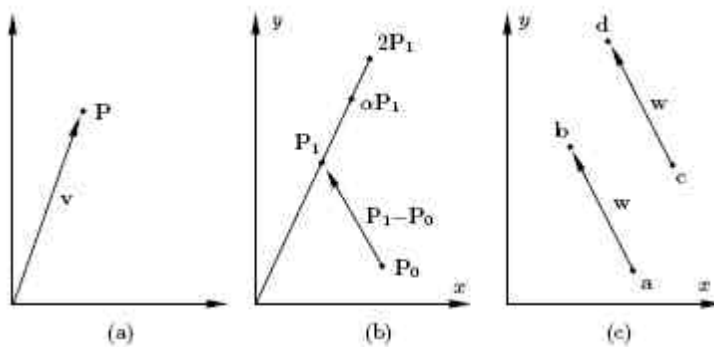


Figura 1.1

A figura 1.1c mostra dois pares de pontos a e b e c e d . Os pontos a e c são diferentes, tal como também o são os pontos b e d . Os vectores $b - a$ e $d - c$, contudo são idênticos.

Exemplo : Os pontos $P_0 = (5, 4)$ e $P_1 = (2, 6)$ são subtraídos para produzir o par $P_1 - P_0 = (-3, 2)$. O novo par é um vector porque representam uma direcção e uma distância. Para irmos de P_0 para P_1 , necessitamos de mover - 3 unidades na direcção x e 2 unidades na direcção y. De forma semelhante, $P_0 - P_1$ é a direcção de P_1 para P_0 . Estas propriedades não dependem do sistema de coordenadas usado. Se por exemplo fizermos a translação da origem estamos a transladar os pontos para novas posições mas a diferença mantém-se. Esta mesma propriedade é válida para as designadas transformações afins (mapeadas) como a rotação, escala, reflexão e corte.

O somatório de um ponto e um vector é um ponto. A figura 1.2a mostra o somatório de duas somas $P_1^* = P_1 + v$ e $P_2^* = P_2 + v$. É fácil visualizar que as posições relativas de P_1^* e P_2^* são as mesmas de P_1 e P_2 . Outra forma é reescrever a relação $a - b = v$ como $a = b + v$, que mostra que do ponto b e do vector v é o ponto a.

Dados dois pontos P_0 e P_2 , a expressão $P_0 + a(P_2 - P_0)$ é a soma de um ponto e um vector, ou seja é um ponto que podemos chamar P_1 . O vector obtido pela subtração dos pontos $P_2 - P_0$ adicionado ao ponto P_0 produz um ponto na linha que liga P_0 e P_2 .

Conclusão : os pontos P_0 , P_1 , e P_2 são colineares. Podemos reescrever a expressão $P_1 = P_0 + a(P_2 - P_0)$ da seguinte forma $P_1 = (1 - a)P_0 + aP_2$, mostrando que P_1 é uma combinação linear de P_0 e P_2 . De uma forma geral três pontos colineares podem se definidos como uma combinação de dois.

Exercício 1.1: Dados três pontos $P_0 = (1, 1)$, $P_1 = (2, 2.5)$, e $P_2 = (3, 4)$, verifique se são colineares?

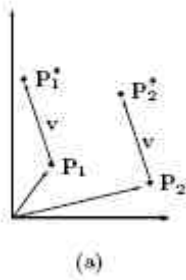


Figura 1.2: (a) Adicionar um ponto a um vector .

Operações sobre Vectores

A notação $|P|$ indica a MAGNITUDE (OU valor absoluto) de um vector P . A adição de vectores é definida como a adição dos elementos individuais dos vectores a serem adicionados : $P + Q = (P_x, P_y, P_z) + (Q_x, Q_y, Q_z) = (P_x + Q_x, P_y + Q_y, P_z + Q_z)$. Esta operação é comutativa $P + Q = Q + P$ e associativo $P + (Q + T) = (P + Q) + T$. A subtracção é semelhante.

Os vectores podem ser multiplicados de três maneiras diferentes :

1. O **produto de um numero real a por um vector P** é representada por aP e produz o vector (ax, ay, az) . Esta altera o valor da magnitude de P do factor a , mas não muda a direcção.

2. O **produto vectorial de dois vectores** é representado por $P \cdot Q$ e definido como o escalar :

$$(P_x, P_y, P_z)(Q_x, Q_y, Q_z)^T = PQ^T = P_xQ_x + P_yQ_y + P_zQ_z.$$

É também igual a $|P| |Q| \cos \theta$, onde θ é o ângulo entre os dois vectores. O produto vectorial de dois vectores ortogonais é zero. O produto vectorial é comutativo, $P \cdot Q = Q \cdot P$.

O produto triplo $(P \cdot Q)R$ é por vezes útil. Pode ser representado por

$$\begin{aligned} (P \cdot Q)R &= (P_x Q_x + P_y Q_y + P_z Q_z)(R_x, R_y, R_z) \\ &= ((P_x Q_x + P_y Q_y + P_z Q_z)R_x, (P_x Q_x + P_y Q_y + P_z Q_z)R_y, \\ &\quad (P_x Q_x + P_y Q_y + P_z Q_z)R_z) \\ &= (Q_x, Q_y, Q_z) \begin{pmatrix} P_x R_x & P_y R_x & P_z R_x \\ P_x R_y & P_y R_y & P_z R_y \\ P_x R_z & P_y R_z & P_z R_z \end{pmatrix} \\ &= Q(PR), \end{aligned}$$

Onde a notação (PR) é uma matriz 3×3 matrix da equação acima.

3. O **produto vectorial de dois vectores** é representado por $P \times Q$ e definido como o vector :

$$(P_2 Q_3 - P_3 Q_2, -P_1 Q_3 + P_3 Q_1, P_1 Q_2 - P_2 Q_1).$$

$$(P_2 Q_3 - P_3 Q_2, -P_1 Q_3 + P_3 Q_1, P_1 Q_2 - P_2 Q_1).$$

É fácil demonstrar que $P \times Q$ é perpendicular a ambos os vectores P e Q .

Projectar um Vector

Uma operação comum com vectores é projectar um vector **a** num vector **b**. A ideia é partir um vector em duas componentes perpendiculares c e d , tal tal forma que c está na direcção de b . (ver figura):

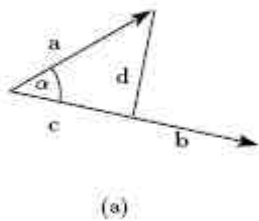


Figura 1.5a mostra que $a = c + d$ e $|c| = |a| \cos \alpha$. Por outro lado, $a \cdot b = |a| |b| \cos \alpha$, obtendo a magnitude de c:

$$|c| = |a| \frac{(a \cdot b)}{|a| |b|} = \frac{(a \cdot b)}{|b|}$$

Exemplo: Dados os vectores $a = (2, 1)$ e $b = (1, 0)$, podemos calcular a projecção de a em b.

$$c = \frac{(a \cdot b)}{|b|^2} b = \frac{2 \times 1 + 1 \times 0}{1^2 + 0^2} (2, 0) = (4, 0), \quad d = a - c = (-2, 1).$$

Sumário : As seguintes operações foram abordadas:

ponto - ponto = vector, escalar \times ponto = ponto, vector \pm vector = vector,
 escalar \times vector = vector, ponto + vector = ponto, vector \cdot vector = escalar,
 vector \times vector = vector.

Definições do Dicionário

Vector: (1) Uma quantidade variável que pode ser decompor-se em componentes. (2) Um segmento linear cujo comprimento é a magnitude e a orientação é a direcção no espaço.

Definições da Wikipédia

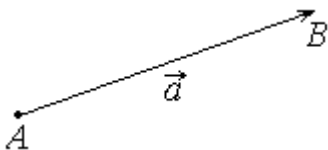
Em física e em cálculo vectorial, vector é um conceito caracterizado por uma magnitude, que é um número positivo (também chamado módulo do vector), e uma orientação (direcção e sentido), que pode ser definida por 2 ângulos em um espaço de 3 dimensões.

Apesar de ser comum descrever um vector em um espaço de 3 dimensões por suas 3 coordenadas, ou componentes, um vector é um objecto cujas propriedades não podem depender do sistema de coordenadas utilizado.

Módulo ou Norma do Vector - $\|\vec{a}\|$

Módulo do vector é o comprimento do vector (na figura seria a distância de A a B).

Fórmula de cálculo : $\|\vec{a}\| = \sqrt{x^2 + y^2}$ (dedução a partir do Teorema de Pitágoras)



Operações com vectores

Adição

$$\vec{a} + \vec{b} = (a_x + b_x)\vec{e}_x + (a_y + b_y)\vec{e}_y$$

Subtracção

$$\vec{a} - \vec{b} = (a_x - b_x)\vec{e}_x + (a_y - b_y)\vec{e}_y$$

Ângulo entre dois vectores

$$\cos\theta = \frac{a_x b_x + a_y b_y}{\|\vec{a}\| \|\vec{b}\|}$$

Um vector pode desta forma ser definido pelas suas propriedades sobre diferentes mudanças de sistema coordenadas. Também é possível generalizar esta definição para espaços não euclidianos com varias dimensões. Por exemplo, em geometria diferencial, um vector pode ser definido como uma derivada de uma curva em um variedade e desta forma possui uma definição livre da escolha de um sistema específico de coordenada.

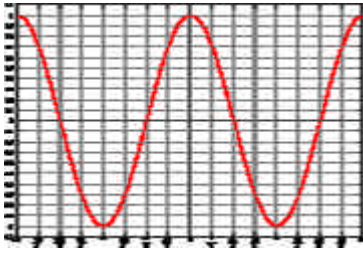
Esta última definição de vectores em geometria diferencial também mostra que um vector é um caso específico de um objeto mais genérico chamado tensor. Vectores são os tijolos com os quais se constrói o Cálculo Vectorial.

Os vectores têm aplicação em várias áreas científicas (física, engenharia e economia, por exemplo, onde facilitam a resolução de alguns problemas).

Na álgebra, vector é um elemento de uma estrutura abstrata chamada espaço vectorial

Co-seno

Origem: Wikipédia



Função co-seno

O co-seno (usam-se ainda as formas coseno e cosseno) é uma função trigonométrica.

Dado um triângulo rectângulo com um de seus ângulos internos igual a θ , define-se $\cos(\theta)$ como sendo a proporção entre o cateto adjacente a θ e a hipotenusa deste triângulo. Ou seja:

$$\cos \theta = \frac{\text{Cateto adjacente}}{\text{Hipotenusa}}$$

Propriedades dos co-senos

Os valores que um co-seno pode obter repetem-se a cada 360 graus, ou 2π radianos. Por exemplo, o co-seno de $\left(\frac{\pi}{2}\right)$ é igual ao co-seno de $\left(2\pi + \frac{\pi}{2}\right)$. Portanto:

$$\cos \theta = \cos (\theta + 2\pi)$$

onde os ângulos estão em radianos. Essa expressão serve para quando se quer saber o co-seno de um ângulo maior que 2π radianos. Na verdade, poderíamos usar qualquer múltiplo inteiro de 2π nessa expressão (incluindo os negativos). Genericamente,

$$\cos \theta = \cos (\theta + 2k\pi), k \in \mathbb{Z}$$

Transformações geométricas a duas dimensões

A criação de imagens de síntese quer seja a 2D quer seja a 3D obriga não apenas a criação de primitivas gráficas (ex: linha, curva, preenchimento) e dos seus atributos (ex: cor, layer) mas também são necessárias operar transformações sobre os elementos gráficos. É destas transformações que vamos falar neste capítulo .

As designadas transformações básicas são:

- translação
- rotação
- escala

Complementarmente a estas usamos muitas outras como reflexão e Cisalhamento.

Translação

Chamamos translação ao acto de levar um objecto de um ponto para outro, num sistema de referência.

Matematicamente podemos definir translação como a operação que permite para cada ponto (x,y) obter um ponto (x',y') que designamos de transladado,

$$\begin{cases} x' = x + t_x \\ y' = y + t_y \end{cases}$$

Onde o par (t_x, t_y) é chamado vector de translação ou vector deslocação: t_x indica quantas unidades a figura é deslocada na direcção horizontal e t_y quantas unidades é deslocado na vertical.

Rotação

Dá-se o nome de rotação ao acto de girar um objecto de um ângulo, num sistema de referência. A rotação em torno da origem do referêncica é dada por:

$$\begin{cases} x' = x * \cos(q) - y * \text{sen}(q) \\ y' = x * \text{sen}(q) + y * \cos(q) \end{cases}$$

Se representarmos matricialmente esta operação obtemos:

$$\begin{bmatrix} x_1' \\ x_2' \end{bmatrix} = \begin{bmatrix} \cos(q) & -\text{sen}(q) \\ \text{sen}(q) & \cos(q) \end{bmatrix} * \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \text{ também se representa como } \mathbf{P}' = \mathbf{R} * \mathbf{P}$$

a operação inversa é dada por: $\mathbf{P} = \mathbf{R}' * \mathbf{P}'$ e em que

$$\mathbf{R}' = \begin{bmatrix} \cos(q) & \text{sen}(q) \\ -\text{sen}(q) & \cos(q) \end{bmatrix}$$

Em que \mathbf{R}' é a transposta de \mathbf{R} : $\mathbf{R}' = \mathbf{R}^T$

Escala

Designamos por escala a transformação cujo resultado é um novo objecto semelhante ao original mas esticado ou encolhido.

As escalas podem ser homogéneas ou não homogéneas (não confundir com escala homogéneas, quer isto dizer, no caso das homogéneas, que os factores de escala são iguais segundo x e y e no segundo caso são diferentes provocando uma distorção na forma geral do objecto.

$$\begin{cases} x' = x * S_x \\ y' = y * S_y \end{cases}$$

Se representarmos matricialmente esta transformação obtemos:

$$\begin{bmatrix} x_1' \\ x_2' \end{bmatrix} = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} * \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \text{ também se representa por } \mathbf{P}' = \mathbf{S} * \mathbf{P}$$

Transformações Compostas

As transformações que vimos até agora são elementares, ou seja são aplicadas independentemente umas das outras. Na prática temos necessidade de aplicar as transformações em grupo. Por exemplo se pretender rodar um objecto que se encontra deslocado da origem do referencial sobre si mesmo (sobre o seu pivot point) temos de aplicar uma sequência de transformações elementares, neste caso:

- 1) deslocar o ponto pivot e objecto para a origem do referencial,
- 2) Aplicar a rotação na origem,
- 3) Transladar novamente o objecto para o local original.

Designamos estas transformações de compostas.

Vamos aprender como executar facilmente estas transformações, mas por necessidade de generalização das operações sobre pontos, vectores e matrizes vamos para já introduzir o conceito de coordenada homogénea.

Nestas coordenadas um ponto 2D passa a ser representado por (x,y,h) e em 3D um ponto passa a ser representado por (x,y,z,h) em que h vale 1.

Ao representarmos as coordenadas dos pontos desta forma todas as transformações geométricas apresentadas passam a poder ser executadas por multiplicações sucessivas de matrizes de dimensão 3x3 elementos. As coordenadas são representadas em coluna nas matrizes e as transformações passam a ser representadas por matrizes de 3x3.

Translação

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Rotação

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\mathbf{q}) & -\text{sen}(\mathbf{q}) & 0 \\ \text{sen}(\mathbf{q}) & \cos(\mathbf{q}) & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Escala

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

As operações inversas destas são representadas por : T(-Tx,-Ty); R(-?) e S(1/Sx,1/Sy).

Concatenação de Transformações genéricas

3 Ambiente de desenvolvimento algorítmico Raptor

Ver a sebenta de introdução a ciência dos computadores 1º de informática da autoria de Jorge Mota.

Funções Matemáticas em Raptor

Cada função em raptor devolve um valor numérico que pode ser usado em qualquer expressão onde um valor numérico é apropriado. A seguinte lista dá exemplos de uso de funções matemáticas em Raptor:

Em atribuições:

```
x_magnitude <- abs(x)
```

```
produto <- e^(log(factor1) + log(factor2))
```

Nas estruturas Select e em expressões de ciclos:

```
log(x) > 0.0
```

```
sqrt(c^2) <= sqrt(a^2 + b^2)
```

```
random > 0.5 and abs(x) < 100.0
```

Podem ainda usar-se em caixas de impressão e argumentos para chamadas de funções (calls).

Lista alfabética de funções.

ABS

```
variavele <- abs(math_expression)
```

abs devolve o valor absoluto de uma expressão matemática. Por exemplo, abs(-3.7) is 3.7, e abs(23) é 23.

CEILING

```
variavel <- ceiling(math_expression)
```

ceiling devolve o menor inteiro maior ou igual ao argumento. Exemplo, ceiling(15.9) é 16, ceiling(3.1) é 4, ceiling(-4.1) é -4, e ceiling(23) é 23.

E

variavel <- e^x

e devolve a base do logaritmo natural (aproximadamente 2.7). esta é uma função constante que não aceita argumentos e devolve sempre o mesmo valor. O programador não pode usar uma variavel de nome e por causa desta função e.

FLOOR

variavel <- floor(math_expression)

floor devolve o maior inteiro menor ou igual ao valor dado. Por exemplo , floor(15.9) é 15, floor(3.1) é 3, floor(-4.1) é -5, e floor(23) é 23.

LOG

variavel <- log(math_expression)

log devolve o logaritmo natural (base e) do valor dado como argumento. UM valor de zero ou negativo dá erro de run-time.

MAX

variavel <- max(math_expression, max_expression)

max devolve o máximo de dois valores dados. Por exemplo, max(5,7) returns 7.

MIN

variavel <- min(math_expression, max_expression)

min devolve o mínimo de dois valores dados. Por exemplo, min(5,7) returns 5.

PI

radians <- degrees * (pi / 180)

pi devolve o valor de PI (aproximadamente 3.14159). Esta é uma função constante.

POWERMOD

```
variavel <- powermod(base, exp, modulus)
```

powermod, Esta função devolve o valor da expressão $((\text{base}^{\text{exp}}) \bmod \text{modulus})$. O Objectivo é o seu uso nos algoritmos de encriptação e deciptação das chaves públicas RSA. powermod é disponibilizado de forma a permitir bases e expoentes grandes que de outra forma não era possível calcular em Raptor.

RANDOM

```
variavel <- random
```

random Gera um numero aleatório entre [0.0,1.0). Ou seja algumas vezes devolve 0 mas nunca devolve 1. Para gerar um numero entre 1 e n usar a função $\text{floor}((\text{random} * n) + 1)$. Por exemplo para gerar os numeros de um dado (random entre 1 e 6) usar $\text{floor}((\text{random} * 6) + 1)$.

SQRT

```
variavel <- sqrt(math_expression)
```

sqrt devolve a raiz quadrada do valor dado como argumento. Se o valor for negativo ocorre um erro.

4 Estruturas de dados para computação gráfica

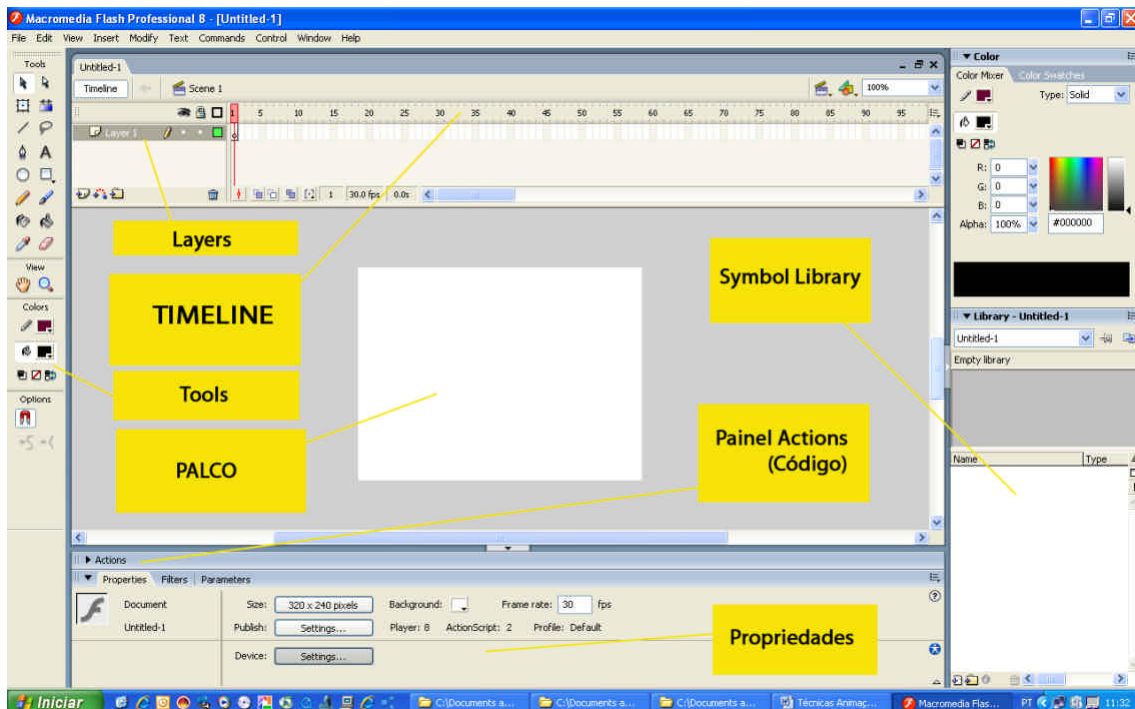
Um aspecto muito importante da computação gráfica é a compreensão das estruturas de dados que lhe servem de suporte.

5 Ambiente de desenvolvimento gráfico Raptor/RaptorGraph

O ambiente de desenvolvimento raptor, possui integrado um ambiente gráfico muito útil para a demonstração de algoritmos aplicados à computação gráfica. O seu maior inconveniente é a sua lentidão.

6 O ambiente de desenvolvimento Flash 8.0

O Adobe Flash (antes Macromedia Flash) é um software de criação de gráficos vectoriais com suporte para imagens bitmap e vídeos - utilizado geralmente para a criação de animações interactivas que funcionam num ambiente de run-time (Flash Player) multiplataforma (PC, Mac OS, Telemóveis, Consolas de Jogos etc), e que pode por exemplo ser vistas num browser Web.



O Flash apresenta um ambiente de desenvolvimento com duas formas distintas de desenvolver conteúdos:

- 1 – Ambiente interactivo
- 2 – Ambiente programável usando uma linguagem de scripting – o ActionScript

Neste curso vamos focar a nossa atenção na criação de conteúdos em flash numa forma programável e numa perspectiva da computação gráfica.

O Introdução ao ActionScript 2.0

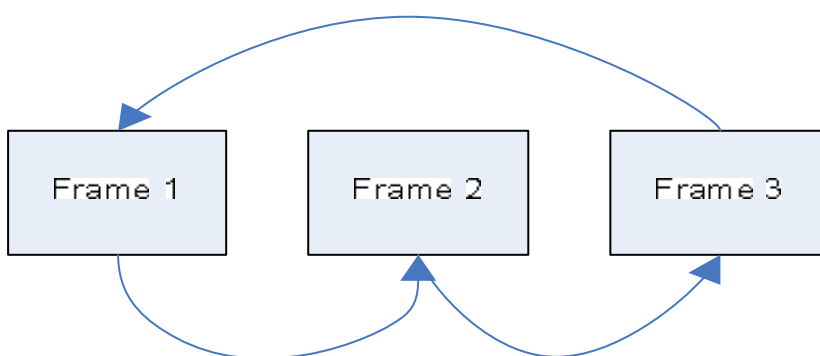
Em versões recentes do Flash (a partir da V5), a Macromedia/Adobe expandiu a utilização do Flash para além das simples animações, criando uma ferramenta de desenvolvimento de aplicações completa. Para o efeito embebeu uma linguagem de programação o ActionScript que pode ser utilizada nos ficheiros flash (.swf). A terceira versão desta linguagem (actionScript 3.0) foi, lançada recentemente, tornando mais fácil e rápido criar conteúdos interactivos multiplataforma, assim como diversificar e sofisticar os mesmos.

Uma nova plataforma, chamada Apollo, está a ser desenvolvida pela Adobe e tem como objetivo solidificar o desenvolvimento da linguagem ActionScript, seja através do Flash, do Flex ou de outros programas.

Em Flash e ActionScript vamos usar 2 modelos de programação:

1. Baseado em Frames.
2. Baseado em Eventos

No **modelo baseado em frames**, a sequência de execução assume o seguinte comportamento:



O código da primeira frame(quadro) é executado, terminado este passa ao código da segunda frame, terminado este passa ao código da terceira frame

e assim por diante, quando chega à última frame volta para a primeira frame e repete a não ser que este ciclo de execução seja interrompido por instrução explícita como gotoAndPlay ou gotoAndStop ou outra. Na versão actual do actionScript, com novas funcionalidades, o habitual é apenas usarmos a frame 1 para colocar o código.

Devemos usar sempre um layer próprio para o código. Assim isolamos o Código em actionScript dos objectos gráficos ficando muito mais fácil de fazer alterações ao programa mais tarde.

No **modelo baseado em eventos**, muito usado até a versão MX e ainda usado nas versão do Flash Lite, o código é associado a cada objecto colocado no palco, e a sua execução é disparada por eventos. O statement usado para este tipo de pré-programação dos eventos é onClipEvent(evento). Vejamos um exemplo:

```
onClipEvent(load){  
  
    //Criar as variáveis que representam o vector velocidade  
  
    var xv=5;  
  
    var yv=5;  
  
}
```

Os dois eventos que mais iremos usar é o **load** e o **enterFrame**. O primeiro é disparado sempre que um objecto é carregado (exemplo carregar um movie), o segundo sempre que a execução entra numa nova frame.

A titulo de exemplo podíamos dizer que o *código de inicialização* poderá aparecer no onClipEvent(load), já que só é executada uma vez, e o código associado ao comportamento do objecto em onClipEvent(enterFrame).

Variáveis em ActionScript.

A linguagem Actionscript 2.0 apresenta duas formas de declaração de variáveis:

- c) Forma Explícita
- d) Forma Implícita

A forma implícita, a única disponível antes da versão 2 do actionScript, pressuponha que o tipo de variável só fosse determinado no momento em que à variável fosse atribuído um valor.

Em contraponto, e como uma forma de evolução desta, a versão 2 da linguagem ActionScript, alias como a generalidade das modernas linguagens de programação, permite a declaração explícita dos tipos no momento da declaração. Vejamos alguns exemplos para visualizar estas distintas formas de declaração:

Declaração Implícita:

// É declarada a variável x, mas o seu tipo só será conhecido quanto um //primeiro valor for atribuído a linguagem.

```
Var x;
```

```
// Neste caso é atribuído o tipo string
```

```
X="Porto";
```

Declaração Explícita:

// É declarada a variável x, e o seu tipo é imediatamente definido no statemente de declaração da variável

```
Var x:Number;
```

```
// Neste caso é atribuído o tipo Numérico
```

É igualmente possível declarar simultaneamente o valor inicial da variável da seguinte forma:

```
Var x:Number=10;
```

De uma forma geral podemos dizer que a linguagem ActionScript, possui os seguintes tipos base para variáveis:

1. STRING ex: "porto"
2. BOOLEAN ex: True
3. NUMBER ex: 5

O primeiro permite conter cadeias de caracteres como por exemplo a palavra "porto".

O segundo tipo assume os valores true ou false (verdadeiro ou falso) e é usada por exemplo em expressões como as condições de um ciclo ou de uma estrutura de decisão (if).

O terceiro tipo define o tipo numérico, por outras palavras permite conter valor numéricos inteiros ou reais.

A atribuição de valores a variáveis em ActionScript é feita usando o operador de atribuição =.

Exemplo:

```
X=1000; // atribui o valor 1000 a variável x
```

Se fizermos:

```
Trace(x)
```

Obtemos na janela de controlo o valor 1000.

Podemos, se pretendermos, inicializar directamente com a declaração da variável o seu valor. A sua sintaxe é:

```
Var x:Number=1000;
```

Uma estrutura de dados muito importante em flash é o array. Outra muito importante é o xml.

Um array é uma estrutura que pode conter outros :

Um array é uma colecção de variáveis partilhando um mesmo nome mas que podem ser acedidas individualmente usando um índice. O índice aparece entre parênteses rectos ([]) depois do nome da variável do tipo array. Ilustrada na figura abaixo está um array unidimensional com nome Notas contendo 4 elementos (numerados de 1 a 4). O que é mostrado são os nomes dos elementos não os seus valores.

Notas[1]	Notas[2]	Notas[3]	Notas[4]
----------	----------	----------	----------

O número entre parênteses recto, como já referido anteriormente é o índice. Este distingue as diferentes "variáveis" agrupadas sobre o nome Notas. Nota Importante: O índice tem que ser um numero inteiro positivo. Em ActionScript o primeiro elemento é o zero. Abaixo é mostrada uma forma diferente de visualizar a estrutura de um array, na primeira linha aparece o índice e na segunda o valor de cada elemento associados a cada índice.

Notas

1	2	3	4
87	93	77	82

O poder dos array nos programas é que nos permite processar inúmeros valor com um simples ciclo contado (loop). O corpo do ciclo processa cada elemento do array através da indexação dos respectivos elementos.

Vamos ver como declarar um array em ActionScript:

```
Var Notas:Array=new Array();
```

Neste caso foi declarado um array chamado pontos. Como podemos agora definir o tipo de dados que pode conter?

Por exemplo num ciclo:

```
For (var i=1;i<10;i++)  
{  
Notas[i]=Math.random()*20;  
}
```

Mas podemos fazê-lo directamente no construtor:

```
Var Notas:Array=new Array(0,1,2,3,4,5,6,7,8,9);
```

Outra Forma é usando o método push da classe array, vejamos um exemplo:

```
Notas.push(1);
```

Outros métodos associados a classe array:

A Ordenação em ordem crescente ou decrescente : **Sort**

Exemplo:

Queremos ordenar a lista "Maria", "Jorge", "Francisco", então:

Declaramos o array

```
Var Lista:array=new array("Maria", "Jorge", "Francisco");
```

Mandamos ordenar

```
Lista.sort(Array.ASCENDING);
```

Vamos verificar se está ordenada:

```
Trace(lista);
```

Array Associativos: Neste tipo de array usamos uma chave não numérica para referenciar os elementos de um array. Vejamos um exemplo.

Vamos criar um array chamado notas:

```
Var Notas:object = New Object();
```

Em seguida vamos associar a nota do "jorge" a este array:

```
Notas.jorge=14;
```

Outra forma de o fazer seria:

```
Notas["jorge"]=14;
```

Em ambos os casos usamos a chave (key) "jorge" para identificar o elemento ao qual pretendemos associar a nota. Se quiséssemos agora verificar a nota do elemento "Jorge", poderíamos usar a notação dot ou parênteses rectos para aceder ao elemento, exemplo:

```
Trace(Notas.jorge);
```

Ou

```
Trace (Notas["jorge"]);
```

Ou ainda,

```
Var chave:string="jorge";
```

```
Trace(Notas[chave]);
```

XML (eXtensible Markup Language)

Xml é uma recomendação da W3C para gerar linguagens de marcação para necessidades especiais.

É um subtipo de SGML (acrónimo de Standard Generalized Markup Language, ou Linguagem Padronizada de Marcação Genérica) capaz de descrever diversos tipos de dados. Ex: (Jorge Mota). O propósito principal é a facilidade de partilha de informações através da Internet. Entre linguagens baseadas em XML incluem-se XHTML (formato para páginas Web), RDF, SMIL, MathML (formato para expressões matemáticas), NCL, XBRL, XSIL e SVG (formato gráfico vectorial).

Objectivos do XML

O W3C desenhou em meados da década de 1990 esta linguagem de marcação para que combinasse a flexibilidade da SGML com a simplicidade da HTML. O princípio do projecto era criar uma linguagem que pudesse ser lida por software, e integrar-se com as demais linguagens. Sua filosofia seria incorporada por vários princípios importantes:

Separação do conteúdo da formatação

Simplicidade e Legibilidade, tanto para humanos quanto para computadores

Possibilidade de criação de tags sem limitação

Criação de arquivos para validação de estrutura (Chamados DTDs)

Interligação de bancos de dados distintos

Concentração na estrutura da informação, e não na sua aparência

O XML é considerado um bom formato para a criação de documentos com dados organizados de forma hierárquica, como se vê frequentemente em documentos de texto formatados, imagens vectoriais ou bancos de dados.

Como trata-se de um formato texto-puro, XML pode ser criado e editado em qualquer editor de textos moderno, suportando ainda a maioria das codificações de caractere (como ISO-8859-1 e UTF-8). Conhecendo-se a sintaxe de XML e o DTD, pode-se escrever documentos XML "à mão" tão válidos quanto os gerados por programas automatizados.

Este exemplo demonstra a sintaxe flexível do XML sendo usada para descrever uma receita de pão:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Receita nome="pão" tempo_de_preparo="5 minutos" tempo_de_cozimento="1 hora">
  <título>Pão simples</título>
  <ingrediente quantidade="3" unidade="xícaras">Farinha</ingrediente>
  <ingrediente quantidade="7" unidade="gramas">Fermento</ingrediente>
  <ingrediente quantidade="1.5" unidade="xícaras" estado="morna">Água</ingrediente>
  <ingrediente quantidade="1" unidade="colheres de chá">Sal</ingrediente>
  <Instruções>
    <passo>Misture todos os ingredientes, e dissolva bem.</passo>
    <passo>Cubra com um pano e deixe por uma hora em um local morno.</passo>
    <passo>Misture novamente, coloque numa bandeja e asse num forno.</passo>
  </Instruções>
</Receita>
```

Temos na 1ª linha

```
<Receita nome="pão" tempo_de_preparo="5 minutos" tempo_de_cozimento="1 hora">
```

"Receita" é o nome principal para o seu documento. Note que a semelhança entre XML e HTML é grande, na 1ª linha abrimos a tag Receita e na última linha fechamos a mesma, como em HTML e assim se estende por todo o exemplo (exemplo tirado de pt.wikipedia.org).

Em ActionScript podemos declarar uma variável do tipo xml da seguinte forma:

```
var xml:XML = new XML();
```

Estruturas de Controlo de Fluxo em ActionScript

Estruturas de Sequência

Para construirmos um bloco de instruções com comportamento sequência, agruparmos conjuntos de instruções com abrir e fechar chavetas, como no exemplo:

```
{  
... Corpo  
}
```

Sempre que tivermos de agrupar grupos de instruções no nosso programa, como no corpo de uma instrução if ou no corpo de um ciclo usamos esta forma de agrupamento.

Estrutura de controlo de fluxo tipo Decisão : Se .. Então .. Senão

Este tipo de estrutura pode ser construída usando duas construções diferentes o **If** ou o **Switch**:

Construção do if na situação de tratamento apenas para o caso da condição ser verdadeira.

```
If (cond) {  
// instruções a executar se a condição for verdadeira  
}
```

Construção para o caso de pretendermos tratar a situação se ser verdadeira ou falsa a condição:

```
If(cond) {  
}  
Else  
{  
// Instruções a executar se a condição é falsa  
}
```

Se pretendermos encadear condições, por exemplo testar se é igual a 5, senão testar se é igual a 6 senão mostrar mensagem a dizer que não coincidem, usamos a construção `if (cond) { ... } else if (cond) {...} else {...}`.

// Ciclos imbricados

```
If (x==5) {  
Trace("foi encontrado o 5 ");  
}  
Else if (x==6) {  
Trace("foi encontrado o 6");  
}  
Else  
{  
Trace("Não foi encontrado");  
}
```

Para além do if o actionScript suporta a estrutura **switch**. A sintaxe para esta intrusão é:

```
Switch(Variavel){  
    case n1:  
        corpo1;  
    case n2  
        corpo 2;  
    default :  
        Corpo do default  
}
```

Estruturas de repetição

O actionscript suporta como estruturas de repetição ou loop os ciclos **for**, **while** e **do...while**.

Ciclos for

O ciclo for permite repetir um bloco de instruções n vezes, é uma estrutura de repetição que designamos por ciclo contado, porque existe uma variável que é incrementada num determinado intervalo (até que se verifique a condição que faz parar o ciclo) de um valor (incremento do contador).

A sintaxe do ciclo for é muito semelhante ao Javascript e C:

```
for (contador=inicial;condição;incremento)
{
    Corpo do ciclo
}
```

Vamos ver um exemplo:

```
Var i:number=10;
for(i=1;i<=10;i++){
    trace(i);
}
```

Neste exemplo é imprimido os numeros entre 1 e 10.

Ciclos while

O ciclo do tipo while permite repetir o corpo do ciclo enquanto a condição for verdadeira. Termina o ciclo quando a condição dá um valor falso. O corpo do ciclo só é executado se a condição for verdadeira na primeira passagem. No exemplo a seguir vamos obter a soma dos cem primeiros números:

```
var i:Number=1;
var soma:Number=0;
While (i<=100) {
    soma=soma+i;
    i=i+1;
```

```
}
```

```
trace(soma);
```

Ciclos do...while

O ciclo do tipo do...while executa um ciclo enquanto a condição for verdadeira, mas o corpo do ciclo é sempre executada pelo menos uma vez.

O mesmo exemplo anterior expresso com um ciclo do...while seria:

```
var i:Number=1;
```

```
var soma:Number=0;
```

```
do
```

```
{
```

```
    soma=soma+i;
```

```
    i=i+1;
```

```
}while (i>=100);
```

```
trace(soma);
```

Expressões em ActionScript

Operadores em ActionScript

Booleanos

And	verdadeiro	falso
verdadeiro	verdadeiro	falso
falso	falso	falso
Or		
Verdadeiro	verdadeiro	verdadeiro
falso	verdadeiro	falso

Relacionais

Tabela de operadores relacionais

Operador	Descrição	Exemplo de uso
$a == b$	devolve verdadeiro se a igual a b	$x == 23$
$a > b$	devolve verdadeiro se a maior que b	$x > 12$
$a < b$	devolve verdadeiro se a é menor que b	$x < -22.3$
$a >= b$	devolve verdadeiro se a é maior ou igual a b	$x >= y$
$a <= b$	devolve verdadeiro se a é menor ou igual a b	$x <= -y$
$a != b$	devolve verdadeiro se a é diferente de b	$x != 1$

Aritméticos e matemáticos

Como executar teste de colisão entre dois movieclips ou com um ponto particular do palco em ActionScript?

Usamos a função hitTest dos movieClips:

```
meu_mc.hitTest(x, y, shapeFlag)
```

neste caso testa a colisão de um movieclip com um ponto de coordenadas (x,y).

```
meu_mc.hitTest(target)
```

neste outro testa a colisão entre dois movieClips o meu_mc e o target.

Parâmetros

x - coordenada x do ponto de colisão do palco.

y - coordenada y do ponto de colisão do palco.

target o nome do movieClip com o qual se pretende verificar a colisão.

my_mc. Uma instância de um movieClip.

shapeFlag Um valor booleano que especifica se pretendemos testar todo o movieClip(true) ou apenas uma boundingBox particular(false).

DEVOLVE

Um valor booleano verdadeiro se ocorre uma colisão (sobreposição) com a área a testar ou falso noutro caso.

Implementação do Jogo Pong em ActionScript Lite

// Programação do **Comportamento da Bola**.

```
onClipEvent(load){
    //Criar as variaveis que representam o vector velocidade
    var xv=5;
    var yv=5;
}
//Este código é executado sempre que entra numa frame

onClipEvent(enterFrame){
    //Verificar colisões com os limites do campo

    if((this._y+this._height)>Stage.height){
        //Recolocar a bola no fim do campo
        this._y=Stage.height-this._height;
        //Activar movimento da Bola para cima
        yv=yv*(-1);
    }
    // Verificar colisões com os limites do campo

    if(this._y<0){
        //Colocar a bola no canto superior
        this._y=0;
        //Activar o movimento da bola para cima
        yv=yv*(-1);
    }
    //Verificar colisões com os limite lateral

    if((this._x+this._width)>Stage.width){
        //Recolocar a bola no fim do campo
        this._x=Stage.width-this._width;
        //Activar movimento da Bola para cima
        xv=xv*(-1);
    }

    //verificar se ultrapassa os limites laterais do campo

    if((this._x+this._width<0) || (this._x>Stage.width)){
        //Somar um ponto ao jogador
        if(this._x+this._width<0){_root.score++;}
        //Colocar a bola a meio campo
        this._x=Stage.width/2;
        //Alterar a direcção da bola
        xv=xv*(-1);
    }
    //Incrementar a posição da Bola
    _x=_x+xv;
    _y=_y+yv;
}
```

// Programação do **Comportamento da Raquete**

```
onClipEvent(enterFrame){  
  
    // Se a tecla UP é premida move a raquete da esquerda para cima  
  
    if(Key.isDown(Key.UP)){  
        this._y= this._y -5;  
    }  
    // Se a tecla DOWN é premida move a raquete da esquerda para  
Baixo  
    if(Key.isDown(Key.DOWN)){  
        this._y= this._y +5;  
    }  
    //Se a Raquete ultrapassa a área do Campo, então é colocada  
novamente no fundo do campo  
    if((this._y+this._height)>Stage.height){  
        this._y=Stage.height-this._height;  
    }  
    //Se ultrapassa o campo parte superior então move para a parte  
superior  
    if(this._y<0){  
        this._y=0;  
    }  
    //Verificar a colisão da bola com a raquete  
    if(this.hitTest(_root.Bola)){  
        //Altera a direcção da bola  
        _root.Bola.xv=_root.Bola.xv*(-1);  
        //Move a bola para os limites da raquete  
        _root.Bola._x=this._x+this._width;  
    }  
}
```

7 Representação matricial e vectorial de imagens

Introdução ao processamento de imagem

O processamento de imagem envolve processar ou alterar uma imagem existente de uma determinada forma.

O primeiro passo é obter uma imagem num formato que se consiga ler. Com a abundante quantidade de informação disponível na internet, hoje, não é difícil obter toda a informação para ultrapassar esta dificuldade. Os ambientes modernos de desenvolvimento como o Flash suportam já a leitura de uma quantidade razoável de formatos como o jpeg ou bmp.

Vamos a título de exemplo apresentar dois formatos standard de imagens muito usados em diversos tipos de trabalhos os formatos TIFF e BMP.

O primeiro passo como já foi referido é ler a imagem depois á que aplicar o processamento pretendido.

Algumas características das imagens

Uma imagem é composta por um array bidimensional de números, números estes que representam a cor. A cada elemento da imagem chamamos pixel (picture element) e é representado por estes ditos "números". A forma mais simples de representação é a dita imagem a preto e branco (black&white) em que só tenho necessidade de para cada pixel por ou um 0 ou um 1 (preto e Branco).

O segundo tipo mais simples de imagem é a designada “grayscale” com escala de cinzentos. Nesta cada pixel contém informação entre 0 e o número de escalas de cinzento suportadas – exemplo : [0,255), ou seja 256 níveis de cinzentos. Estas imagens podem ser só preto e branco ou possuir níveis de cinzento. As pessoas conseguem distinguir cerca de 40 níveis de cinzento, pelo que uma escala de 256 níveis é mais que aceitável [12].

O terceiro tipo de representação de imagem é a designada imagem a cores. Este tipo de imagem usa três canais de cor primárias o RGB – red, green e blue - (também designadas bandas) para cada pixel. É comum que cada destes canais de cor tenham escalas de cor com 256 níveis, o que na prática significa que podem representar aproximadamente 16 milhões de cores.

Para obter uma imagem digital existem inúmeras formas: executar o scanner de uma fotografia em papel, tirar uma foto com uma máquina digital, descarregar uma foto de repositório da internet. Por exemplo neste último caso bastaria carregar no botão do lado direito do rato quando o apontar estiver sobre a imagem e guardar a mesma. Nestes casos o formato mais comum é o jpeg.

Especificações de “file I/O”

Na maioria dos actuais ambientes de desenvolvimento já existem componentes, classes ou bibliotecas que permitam a leitura escrita de ficheiros standard de imagem. Não quero no entanto deixar de abordar este tema pelo que vamos falar de dois formatos e das suas principais características.

A forma geral de uma programa de leitura/processamento/escrita de uma imagem digital é (exemplo em linguagem C):

```
0 char *in_nome, *out_nome;
```

```
1 short **a_imagem;
```

```
2 long altura, largura;
```

```
3 criar_imagem_file(in_nome, out_nome);  
4 get_imagem_tamanho(in_nome, &altura, &largura);  
5 a_imagem = alocar_imagem_array(altura, largura);  
6 ler_imagem_array(in_nome, a_imagem);  
7 call uma_rotina_processamento_imagem  
8 escrever_imagem_array(out_nome, a_image);  
9 libertar_imagem_array(a_image, altura);
```

Os passos do algoritmo são:

- 1 – Ler o ficheiro da imagem (TIFF, BMP, JPEG ou outros)
- 2 – Chamar a rotina de processamento (exemplo converter sepia)
- 3 - Escrever no disco (ou outro dispositivo) a imagem
- 4 – libertar o espaço anteriormente alocado

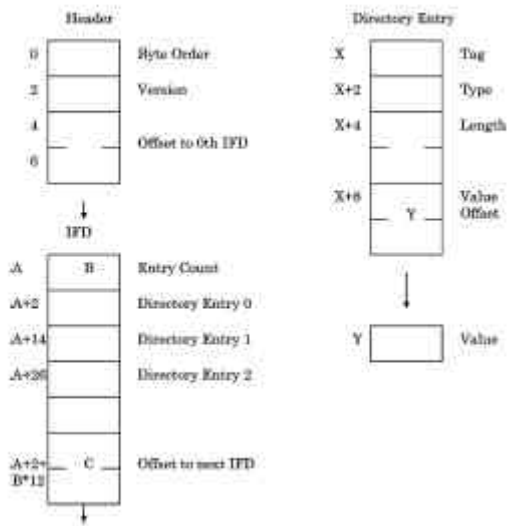
A estrutura de dados usado neste caso para conter a imagem foi um array e as rotinas de leitura e escrita escondem os pormenores de I/O relativo aos formatos.

TIFF (Tag Image File Format)

Vários fabricantes de computadores e scanners criaram um formato considerado " industry standard" o formato TIFF. Este formato é suportado por muitos fabricantes e produtores de sw para PC e Mac . Nas premissas base deste formato estavam:

- Extensibilidade, portabilidade e alterabilidade. Ou seja pretendia-se um formato que permiti-se a extensibilidade no futuro, a portabilidade entre computadores, processadores e Sistemas operativos e um formato escrita e leitura. O formato usa uma estrutura de tags organizadas em directorias de tags que contêm informação como largura e altura e numero de pixels. As

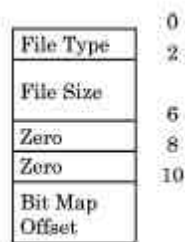
directorias não tem comprimento fixo o que permite uma alteração flexível das necessidades.



BMP

O formato Microsoft Windows Bitmap (BMP) é um formato básico para imagens digitais no Microsoft Windows world. Este formato muito mais simples e menos versátil que o TIFF tem como único objectivo guardar imagens digitais. Simplicidade é a palavra chave para este formato e q sua simplicidade de leitura e escrita.

É o formato nativo no Windows world, pelo que a maioria do software Windows o usa. Foi criado apenas para processadores tipo INTEL.



O programa Microsoft's Paint (gratuito com o windows) trabalha com BMP.

A imagem é guardada de baixo para cima.

O BMP suporta tabelas de cor ou seja tabelas que fazem a correspondência entre números e cores. Ou seja embora suponhamos o número da cor seja 12 a cor pode ser 200, 250 ou outra qualquer. Este sistema é uma oportunidade para poupar espaço na imagem já que só temos que armazenar especificamente a informação das cores que a imagem usa.

Header	0
Size	4
Image Width	8
Image Height	12
Color Planes	14
Bits Per Pixel	16
Compression	20
Size of Bitmap	24
Horizontal Resolution	28
Vertical Resolution	32
Colors	36
Important Colors	

No caso do BMP a tabela de cores (color table) tem quatro bytes para cada cor três usadas para o red, green e blue e o quarto é sempre o. Ou seja para uma tabela com 256 cores a tabela de cores ocupa 4x256 bytes. No bmp a imagem é gravada linha a linha com uma marca de fim de linha para facilitar a leitura blocada.

Como Usar imagens bitmap em Flash

O objecto BitmapData, disponível a partir da versão 8.0 do Flash permite a manipulação de imagens bitmap em ActionScript.

Como criar um objecto BitmapData?

New BitmapData (width:Number, height:Number, Transparent: Boolean, fillColor:Number);

O package em que está incluído este objecto é o: *flash.Display.BitmapData*, devemos pois fazer a sua importação no início do nosso programa.

Os parâmetros requeridos são os seguintes:

- width e height respectivamente largura e altura da imagem.
- Transparent : Diz que a imagem contém uma canal Alfa.
- fillColor: é a cor inicial com que é criado o bitmap.

Nota com cores a 32 bits :

0xFFFFFFFF = Branco puro

0x00FFFFFF = Cor 100% transparente

0x80FFFFFF = Cor 50% transparente de branco

O método attachBitmap permite fazer o attach em run time de um bitmap:

Sintaxe:

attachBitmap(bitmapData,depth,pixelSnapping, smoothing)

BitmapData é um bitmap

Depth é um inteiro que nos dá em que plano fica colocado o bitmap

pixelSnapping diz-nos se é feito ou não o snapping para os pixels(auto, always ou none)

smoothing se é aplicado amaciamento (true ou false)

vamos ver um exemplo:

```
Import flash.display.BitmapData;
```

```
Var bitmap:bitmapData=New bitmapData(100,100,false,0xffff0000);
```

```
_root.attachBitmap(bitmap,0)
```

Exemplo de um programa que faz spray sobre um bitmap (pagina 95 [2])?

```
Import flash.display.BitmapData;

Var densidade: Number= 100;

Var raio: Number=50;

Var bitmapData=new (bitmapData(Stage.width,
Stage.height,false,0xFFFFFFFF);

_root.attachBitmap(bitmap,0);

Function onMouseDown():Void{
    sprayColor=Math.random()*0xfffff;
    //no caso de este evento ocorrer executar a função spray
    onEnterFrame=spray;
}

Function onMouseUp():Void{
Delete onEnterFrame;
}

Function spary():Void{
For (var i: Number=0; i<densidade;i++){
Var angulo: Number=math.random()*Math.PI*2;
Var ranRaio: Number=Math.random()*raio;
Var ranX: Number=Math.cos(angulo)*ranRaio;
Var ranY: Number=Math.sin(angulo)*ranRaio;
Bitmap.setPixel32(_xmouse+ranX,Ymouse+ranY,sprayColor);
}
}
```

Como Carregar uma imagem num bitmap em ActionScript?

```
import flash.display.BitmapData

//criar um movie clip onde colocar a imagem

//this.createEmptyMovieClip("Editor",this.getNextHighestDepth());

/*

Usar a classe MovieClipLoader de forma que possamos controlar quanto
ocorre um erro ao carregar a imagem.

*/

loader = new MovieClipLoader()

//registar o objecto de forma que possa tratar eventos
loader.addListener(this)

// Carregar o ficheiro no nosso movieClip

loader.loadClip("fotol.jpg",Editor)

//Esta função é chamada pelo objecto 'loader' quando o ficheiro for
carregado e estiver pronto a ser usado.

function onLoadInit()
{

/*

Criar um novo objecto em memoria que é do mesmo tamanho da imagem
carregada, mas preenchido com pixels transparente.

*/

meuBitmap = new BitmapData(Editor._width,
Editor._height,true,0x00FFFFFF)

//Snapshot do movie clip que contem a imagem carregada

    meuBitmap.draw(Editor)

    //descartar o movie clip Original
    Editor.removeMovieClip()
}

//Esta função é chamada se ocorrer um erro no carregamento da imagem

function onLoadError()
```

```
{  
    /*  
    Remover o contentor do movie clip, por não ser necessario  
    */  
    Editor.removeMovieClip()  
}
```

Como Carregar uma imagem do disco usando XML?

Criar um documento XML.

Vamos ver como carregar dados de uma página em xml e usar esta informação em Flash.

Primeiro vamos criar um documento xml com o nome da imagem que queremos carregar e vamos chamar-lhe *imagens.xml*. O código será este:

```
<images>
  <image image="imagens/foto1.jpg" caption="Londres"/>
</images>
```

Neste documento vamos manter a estrutura do xml simples pelo que só vamos usar um nó. Vamos gravar este documento numa directoria chamada xml.

Criar o documento Flash.

Criar um documento flash e chamar-lhe main (estilo linguagem C). renomear o nome do layer para Graficos. Colocar um campo de texto dinamico no palco (stage), com o nome de instancia Texto, Assegurar que são embebidos (botão embed) a font, maiusculas, minusculas e pontuação. Depois desenhar um rectangulo e usar F8 para criar um movieClip e chamar-lhe Editor. Neste movie colocar o x e o y respectivamente a (0,0).

Finalmente criar um layer chamado codigo e colocar lá estas instruções:

```
// criar a variavel XML
var xml:XML = new XML();
// Vamos configurar de forma a ignorar espaços
xml.ignoreWhite = true;
// definir a função que é chamada quando o xml é carregado
```

```
xml.onLoad = function() {  
  // indicar quantos nodos existem no xml  
  var nodes = this.firstChild.childNodes;  
  // Quantos itens existem  
  numDeItens = nodes.length;  
  // Fazer o attach  
  for (var i = 0; i < numDeItens; i++) {  
    // fazer o attach da imagem para o MovieClip Editor  
    Editor.loadMovie(nodes[i].attributes.image);  
    // Carregar o texto  
    Texto.text = nodes[i].attributes.caption;  
  }  
};  
// Carregar o XML  
xml.load("xml/imagens.xml");
```

Projecto 1: Tratamento de imagem em ActionScript (Editor de imagem).

Neste projecto pretende-se construir um pequeno editor de imagens bitmap (exemplo Fotografias), que permita aplicar pequenos efeitos sobre os(as) mesmas. Exemplos:

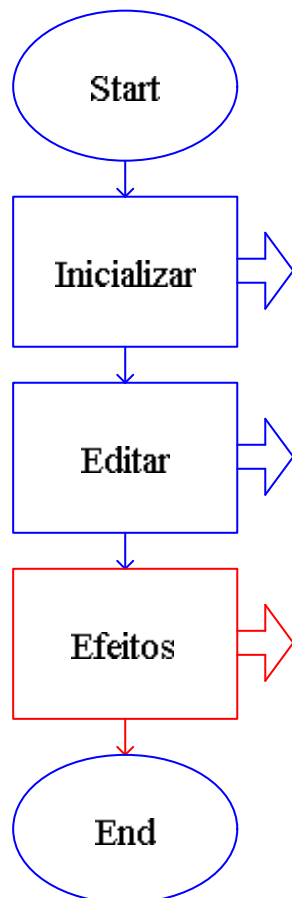
Fazer Zoom (in) e Zoom (out)

Converter a imagem para preto e branco

Converter a imagem para Sépia

Calcular o Histograma

Eliminar Olhos vermelhos



Estrutura geral de um pequeno Editor de bitmaps para exemplo:

```
// Este pequeno exemplo usa um ficheiro xml para definir a fotografia/imagem que se  
// pretende editar.  
// Autor: Jorge Mota, ISTECAbril de 2007  
// incluir a API de tratamento de bitmaps do ActionScript 2.0  
import flash.display.BitmapData  
// Carrega o ficheiro a partir do XML
```

```
iniciar();
//Cria um bitmap através do API do ActionScript
Editar();
// Define as funções para os botões
botoes();
function Editar():Void{
    //Criar um novo objecto em memoria que é do mesmo tamanho da
    imagem carregada, mas preenchido com pixels transparente.
    meuBitmap = new BitmapData(Editor._width,
    Editor._height,true,0x00FFFFFF)
    //Snapshot do movie clip que contem a imagem carregada
    meuBitmap.draw(Editor)
    //descartar o movie clip Original
    Editor.removeMovieClip()
}
function botoes():Void{
    this.B_menos.addEventListener("click", diminuir);
    this.B_mais.addEventListener("click", aumentar);
}
function aumentar()
{
    trace("aumenta")
}
function diminuir()
{
    trace("Diminui")
}
function iniciar():Void{
    // criar a variavel XML
    var xml:XML = new XML();
    // Vamos configurar de forma a ignorar espaços
    xml.ignoreWhite = true;
    // definir a função que é chamada quando o xml é carregado
    xml.onLoad = function() {
    // indicar quantos nodos existem no xml
    var nodes = this.firstChild.childNodes;
    // Quantos itens existem
    numDeItens = nodes.length;
    // Fazer o attach
    for (var i = 0; i<numDeItens; i++) {
    // fazer o attach da imagem para o MovieClip Editor
    Editor.loadMovie(nodes[i].attributes.image);
    // Carregar o texto
    Texto.text = nodes[i].attributes.caption;
    }
    };
    // Carregar o XML
    xml.load("xml/imagens.xml");
}
```

Conceitos sobre física do movimento

Neste capítulo vamos falar um pouco de física do movimento e física para jogos. O objectivo é falar de grandezas como a velocidade, aceleração, inércia, gravidade e atrito.

Tweening in Code

Animação refere-se ao processo segundo o qual cada fotograma de um filme é produzido individualmente, podendo ser gerado quer por computação gráfica quer fotografando uma imagem desenhada, quer repetidamente fazendo-se pequenas mudanças a um modelo (claymation e stop motion), fotografando o resultado. Quando os fotogramas são ligados entre si e o filme resultante é visto a uma velocidade de 16 ou mais imagens por segundo, há uma ilusão de movimento contínuo (por causa da persistência de visão). A construção de um filme torna-se assim um trabalho muito intensivo e por vezes entediante. O desenvolvimento da animação digital aumentou muito a velocidade do processo, eliminando tarefas mecânicas e repetitivas

A animação tradicional, até a bem pouco tempo, era feita usando uma câmara montada verticalmente num suporte apontando para baixo. A este setup era dado o nome de Trunca ou "rostrum". Os únicos movimentos permitidos eram a de mover a câmara de forma rígida de cima para baixo.

Os desenhos eram colocados num suporte adequado sob a câmara que podia mover-se para a direita e esquerda, cima e baixo e muitas ainda tinha a possibilidade de rodar 360°.

Desta forma era possível mover os desenhos sob a câmara criando movimentos, junto com os movimentos de zoom da própria câmara.

Para criar o movimento o operador da câmara de animação possuía tabelas que lhe permitiam calcular a deslocação da câmara ou célula de animação de forma a obter o movimento desejado. Estas tabelas permitiam interpolar o movimento imprimindo uma aceleração no início do movimento e desacelerando até parar.

Algumas das curvas usadas eram baseadas em senos (função trigonométrica) . Hoje o uso de computadores veio facilitar muito o trabalho de operação da animação [8].

Easing in e Easing out

Quando em flash movemos um clip de uma posição para outra ao longo da linha de tempo, temos várias hipóteses de “conformar” o movimento. Cada intervalo da linha de tempo pode ser usado para mover o clip de uma mesma distância, o que soa artificial, (designada interpolação linear do movimento e é o tipo por defeito do flash) ou podemos acelerar ou desacelerar o movimento por exemplo no início e fim do movimento (ease in e ease out) e obter um movimento mais natural. Este rampeamento do movimento está muito mais próximo da física do movimento (já pensaram se os carros instantaneamente chegassem aos 100 Km/h).

O controlo do movimento conforme descrito pode ser obtido por matemática simples e é o que iremos descrever a seguir.

Interpolação Linear

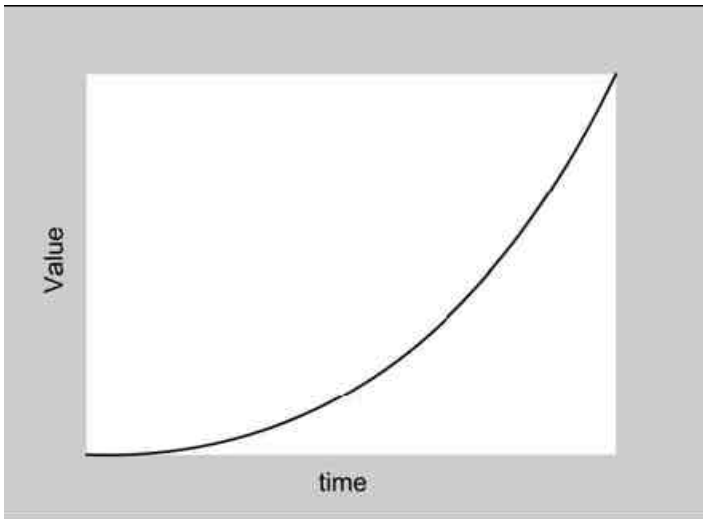
Se um clip se move de uma posição A para uma B no tempo T, então a interpolação linear do movimento pode ser obtida da seguinte forma:

```
var dt=tempo/duracao;  
_x=offset.x*dt+inicio.x;  
_y=offset.y*dt+inicio.y;
```

Usamos um objecto da classe point para guardar a posição inicial e final necessárias. A variável time dá-nos o tempo da timeline a variável duração é a duração total da animação. Quando iniciamos a posição é igual ao ponto de início, no final coincide com o ponto final (tempo/duracao=1) [8].

Interpolação Quadrática

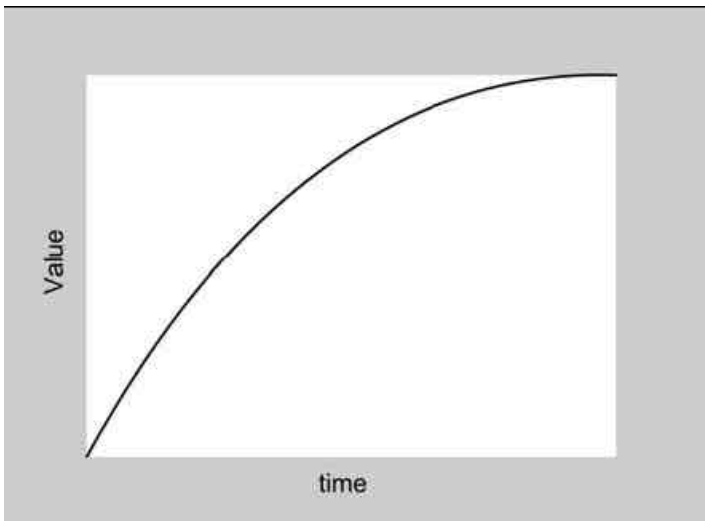
O gráfico $y = x^2$ é mostrado na figura :



O gráfico como se verifica não é linear mas sim uma curva. Se consideramos que x pode assumir valores entre $[0,1]$, e representar o tempo proporcional com a duração total da animação então y representa a distância percorrida. Neste caso temos inicialmente uma pequena distância percorrida sofrendo progressivamente uma aceleração. A representação do movimento em actionscript seria dado por:

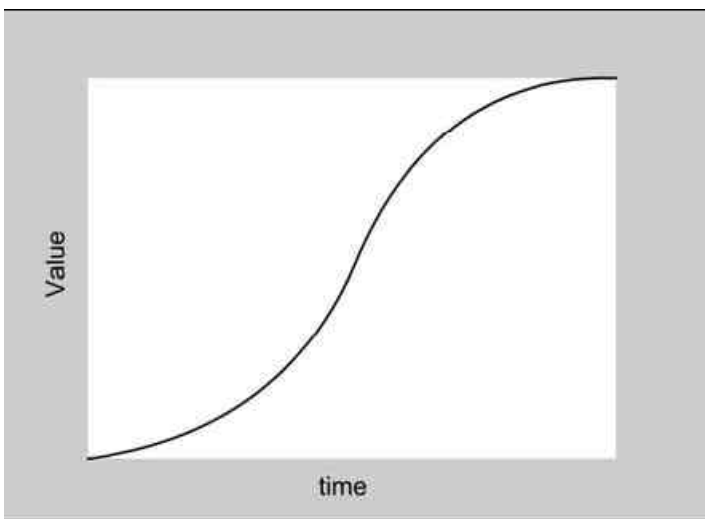
```
var dt=tempo/duracao;  
_x=offset.x*dt*dt+inicio.x;  
_y=offset.y*dt*dt+inicio.y;
```

Nesta situação conviria obter uma forma de ease out ou seja obter uma curva conforme representado na figura:



Para tal invertemos a curva e para tal usamos para o tempo $(1-dt)$ (com dt a ser um valor entre $[0,1]$). A função quadrática desta será $(1-dt) \cdot (1-dt)$, se desenvolvermos a função obtemos $1-(1-2dt+dt^2)$ que podemos simplificar para $2dt-dt^2$. Ou seja a equação será $y=2x-x^2$

Gráfico da função:



A função é criada com o seguinte código em actionscript:

```
Var dt=tempo/duracao;  
_x=offset.x*dt*(2-dt)+inicio.x;
```

```
_y=offset.y*dt*(2-dt)+inicio.y;
```

Para fazermos o ease in e ease out juntando as duas curvas vamos fazer uso das duas curvas até metade da duração usamos uma curva na segunda metade usamos a outra curva:

```
Var dt=tempo/duracao;
```

```
Dt *=2
```

```
If (dt<1){
```

```
    _x=offset.x/2*dt*dt+inicio.x;
```

```
    _y=offset.y/2*dt*dt+inicio.y;
```

```
}
```

```
Var dt=tempo/duracao;
```

```
Dt *=2
```

```
If (dt>=1){
```

```
    Dt--;
```

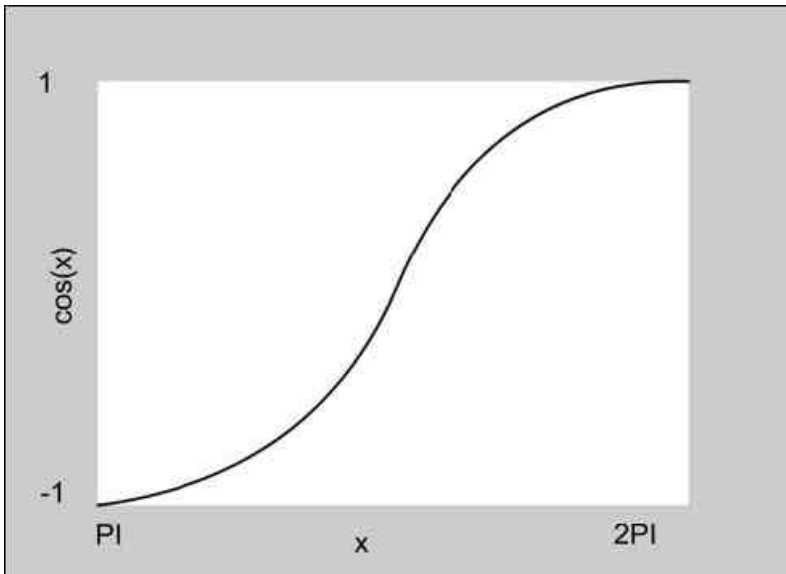
```
    _x=offset.x/2*dt*(2-dt)+0.5+inicio.x;
```

```
    _y=offset.y/2*dt*(2-dt)+0.5+inicio.y;
```

```
}
```

Interpolação usando senos e cos-senos

Um método completamente diferente de obter curvas de movimento é usarmos a função seno ou cosseno. Estudemos o gráfico seguinte:



Para obtermos uma curva semelhante a quadrática já exposta basta representar a curva dada pelo cosseno entre os valores $[PI, 3PI/2]$, que nos dá valores entre $[-1,0]$.

Para obtermos em actionscript esta curva podemos fazer:

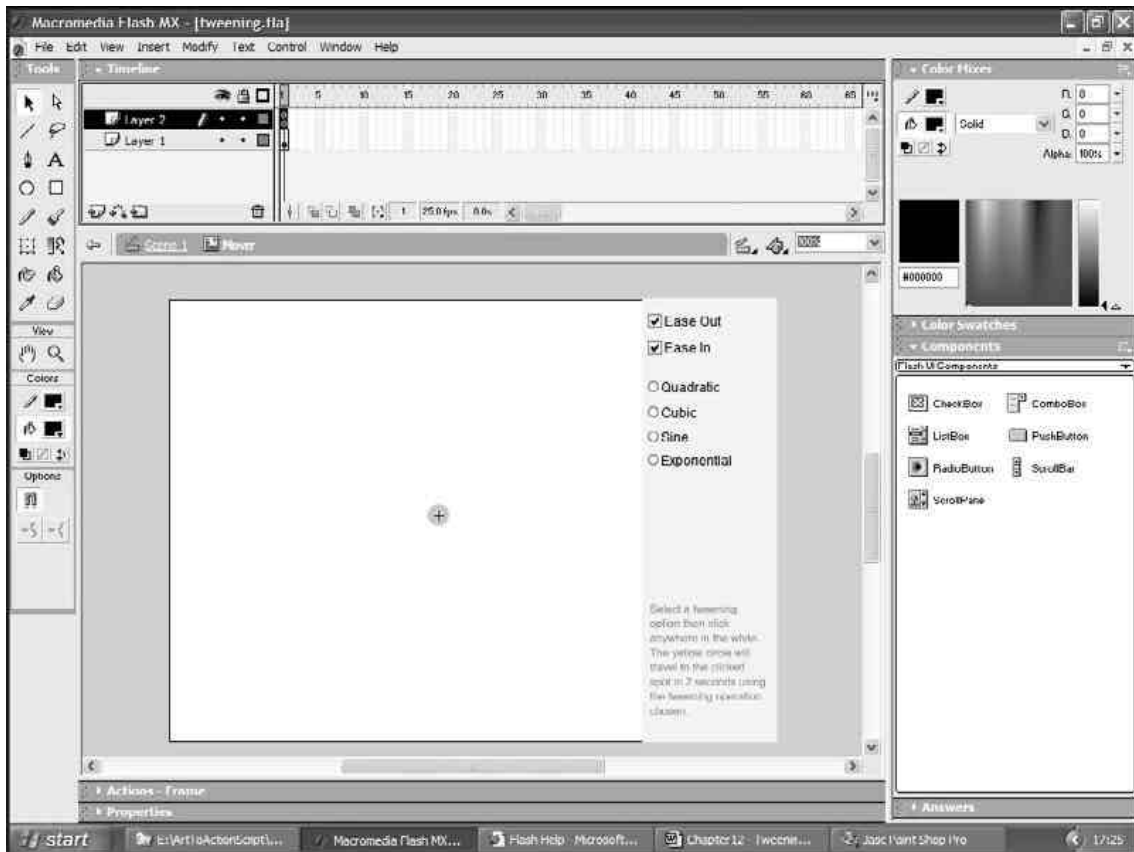
```
var n, dt = time/duration;  
n = (Math.cos(Math.PI * dt + Math.PI) + 1)/2;  
_x = offset.x * n + start.x;  
_y = offset.y * n + start.y;
```

Interpolação usando a curva exponencial

Uma outra curva que dá um rampeamento rápido é a curva $y=2^x$. Qualquer numero elevado a zero dá 1. Ora pretendemos um intervalo para os valores de dt entre $[0,1]$, mas como as curvas precisam de ser escaladas e transladadas vamos considerar a seguinte implementação [8]:

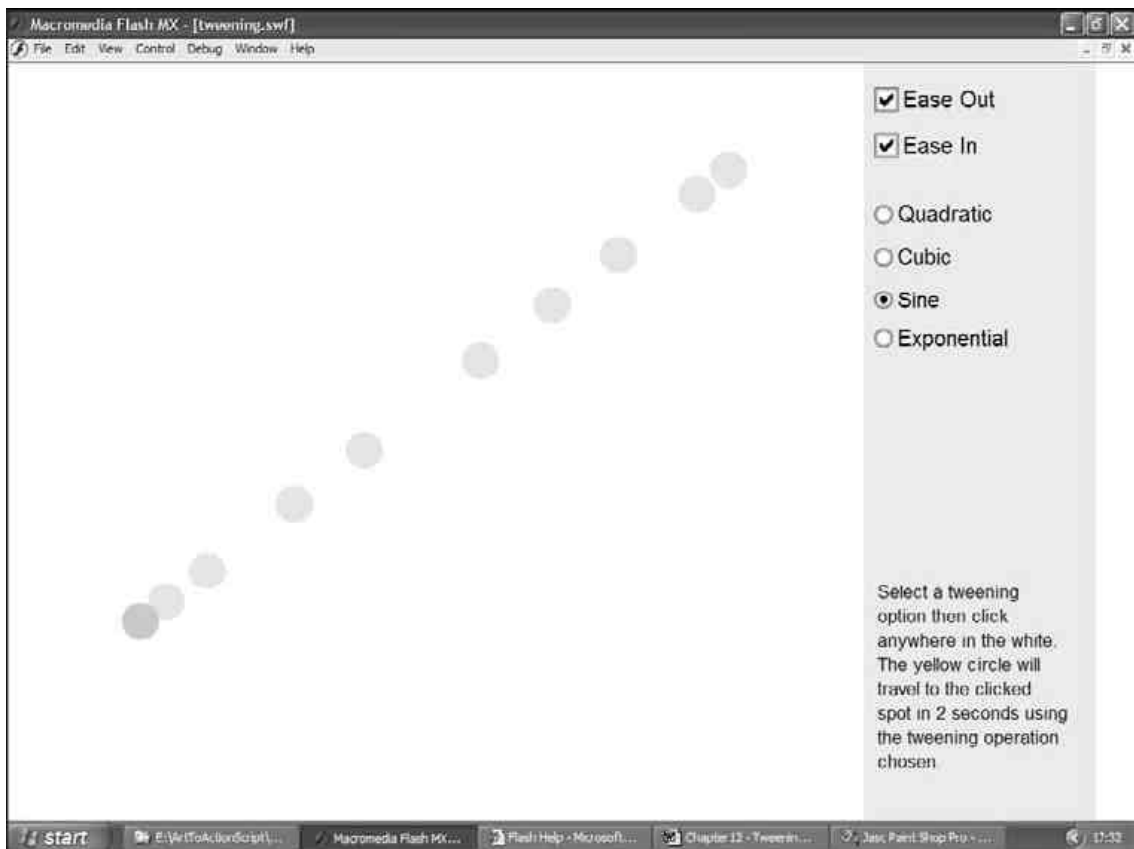
```
var n, dt = time/duration;  
dt *= 2;  
if (dt < 1){  
n = Math.pow( 2, 10 * (dt - 1))/2;  
_x = offset.x * n + start.x;  
_y = offset.y * n + start.y;  
}else{  
dt--;  
n = -(Math.pow( 2, -10 * dt) - 2 )/2;  
_x = offset.x * n + start.x;  
_y = offset.y * n + start.y;  
}
```

Vamos agora implementar um exemplo de tweening usando os elementos estudados:



```
start = new point();
end = new point();
offset = new point();
start.set(12, 34);
end.set(10, 52);
offset.copy(start);
offset.sub(end);
function point(){
this.x = 0;
this.y = 0;
this.set = setPoint;
this.add = addPoint;
this.sub = subPoint;
this.copy = copyPoint;
}
function setPoint(nx, ny){
this.x = nx;
this.y = ny;
}
```

```
function addPoint(pt){
this.x += pt.x;
this.y += pt.y;
}
function subPoint(pt){
this.x -= pt.x;
this.y -= pt.y;
}
function copyPoint(pt){
this.x = pt.x;
this.y = pt.y;
}
```



8 Usabilidade e interacção (ActionScript 2.0)

Neste capítulo vamos falar um pouco de ergonomia computacional e usabilidade.

9 Luzes

Neste capítulo vamos falar de luzes e iluminação. Princípios básicos, aplicações em jogos e implementação em ActionScript.

10 Oclusão: Ordem de visualização (Backface culling e Depth Sorting)

Neste Capítulo pretende-se falar dos algoritmos de oclusão e a sua importância na computação gráfica moderna.

11 Projecto 2: Jogo clássico para PSP em Flash Lite

Neste projecto pretende-se desenvolver um pequeno jogo tipo arcade(plataformas) ou para Playstation portátil). Exemplos de tipos de Jogos arcade:

- Space Invaders (clássico)
- Arcanóide
- Asteróides

O jogo a desenvolver deve obrigatoriamente ter um fim didáctico, ou seja destina-se a aprendizagem. O destinatário pode ser o mais variado mas deve ser claro qual o objectivo pedagógico do jogo.

A engenharia de software estabelece que devemos considerar as seguintes elementos no desenvolvimento de um jogo [13]:

- 1-scope,
- 2-arquitectura,
- 3-planeamento,
- 4-testes,
- 5-implementação.

Scope. Requisitos que definem o produto. Claramente deve ser a expressão do que faz e o que é obtido.

Arquitectura. Define como e qual software que compõem o produto e como está organizado física e logicamente (desenho lógico e físico)

Planeamento. O Scope e Desenho permitem planear como e por quem vai ser desenvolvido o software do produto. Nesta fase são especificadas as actividades para desenvolver o produto e como relacionam entre si , por quem e por que tempo.

Teste. Os desenvolvedores de software testam o seu trabalho, logo que saí do desenvolvimento, em várias vertentes de forma a verificar a validade do

mesmo a luz dos requisitos e qualidade em objectivos como a performance, fiabilidade e manutenção.

Implementação. Esta fase corresponde ao desenvolvimento do produto em si de acordo com as especificações do scope, o desenho (arquitectura) e de acordo com o plano estabelecidos. O desenvolvimento de um produto é constituído por muitas actividades que se relacionam de forma complexa [13].

12 ActionScript – Implementação do Jogo PONG

Pseudocódigo é uma forma genérica de escrever um algoritmo, utilizando uma linguagem simples (nativa a quem o escreve, de forma a ser entendido por qualquer pessoa) sem necessidade de conhecer a sintaxe de nenhuma linguagem de programação. É, como o nome indica, um pseudocódigo e, portanto, não pode ser executado num sistema real (computador) — de outra forma deixaria de ser pseudo.

13 História e aplicações da Computação gráfica [1]

Retirado do Livro – Computação gráfica Teoria e Prática

ORIGENS DA COMPUTAÇÃO GRÁFICA

Conhecer a origem é saber se posicionar na escala da evolução, descobrindo as necessidades, motivos e personalidades que alavancaram o desenvolvimento, para só, então, se projetar para um futuro real e imaginário. Parece existir um consenso entre os pesquisadores de que o primeiro computador a possuir recursos gráficos de visualização de dados numéricos foi o *Whirlwind I*, desenvolvido pelo MIT. Esse equipamento foi desenvolvido, em 1950, com finalidades acadêmicas e militares. Em 1955, o comando de defesa aérea dos Estados Unidos desenvolveu um sistema de monitoramento e controle de vôos (*SAGE – Semi- Automatic Ground Environment*) utilizando o *Whirlwind I* como plataforma. O sistema convertia as informações capturadas pelo radar em imagem de um tubo de raios catódicos (na época, uma invenção recente), no qual o usuário podia apontar com uma caneta ótica para os pontos suspeitos. Em 1959, surgiu o termo *Computer Graphics*, criado por Verne Hudson, enquanto o mesmo coordenava um projeto para a Boeing de simulação de fatores humanos em aviões. Em 1962, surgiu uma das mais importantes publicações da computação gráfica de todos os tempos, a tese de Ivan Sutherland (*Sketchpad – A Man-Machine Graphical Communication System*), introduzindo as estruturas de dados para o armazenamento de hierarquias construídas através da replicação de componentes básicos, bem como as técnicas de interação que usavam o teclado e a caneta ótica para desenhar, apontar e escolher alternativas. Essa publicação chamou a atenção das indústrias automobilísticas e aeroespaciais americanas. Os conceitos de estruturação de dados, bem como o núcleo da noção de computação gráfica interativa, levaram a General Motors a desenvolver em 1965 o precursor dos programas de CAD (Computer Aided Design). Logo depois, diversas outras grandes corporações americanas seguiram esse exemplo sendo que, no final da década de 1960, praticamente toda a indústria automobilística e aeroespacial utilizava softwares de CAD.

Na década de 1970, vários pesquisadores desenvolveram novas técnicas e algoritmos que são utilizados até hoje, tais como os métodos de sombreamento e o algoritmo de z-buffer. Nessa mesma época, surgiu a tecnologia dos circuitos integrados permitindo o barateamento das máquinas e o lançamento, em 1975, do primeiro computador com interface visual, o predecessor do Macintosh. Outros fatos importantes dessa década foram: o reconhecimento da computação gráfica como área específica da ciência da computação, o surgimento dos congressos específicos em computação gráfica (SIGGRAPH), a publicação do primeiro livro sobre computação gráfica interativa e o lançamento em 1977 do livro *Fractals: Form, Chance and Dimension*, onde o autor, Benoit Mandelbrot, matemático e, na época, pesquisador da IBM, conseguiu mostrar com imagens geradas em computador a incrível complexidade das equações fractais.

Em janeiro de 1980, a *Scientific American* publicou uma extraordinária imagem chamada Plume 2, a primeira imagem de uma erupção vulcânica no espaço, na lua Jovian tirada pela nave espacial Voyager 1. A foto era ao mesmo tempo um marco e um triunfo para a computação gráfica e mais especificamente o processamento de imagens, onde a imagem do telescópio recebeu um processamento utilizando técnicas da computação gráfica para permitir a visualização da erupção. Imagens de satélite e de explorações interplanetárias são grandes usuárias das técnicas de processamento de imagem, que numericamente manipulam os pixels das imagens para reduzir ruídos, melhorar o contraste ou produzir um efeito desejado, como, por exemplo, salientar um aspecto particular que mereça maior atenção. A década de 1980 viu surgir diversas técnicas novas de iluminação global como o *ray-tracing* (em 1980) e a radiação (em 1984), aproximando as imagens geradas por computador do fotorrealismo. Outro fato marcante dessa década foi a estranha criação, em 1987, da cabeça falante Max Headroom, utilizada em um programa de TV da Inglaterra para simular expressões faciais humanas e apresentar o programa. A década de 1990 marcou o amadurecimento da computação

gráfica com imagens impressionantes como no filme *Jurassic Park*, em 1993. O filme marca a perfeição do fotorrealismo, nas cenas de movimentos dos dinossauros. Não poderíamos deixar de citar o filme *Terminator 2* com a utilização de um personagem computadorizado, e *Toy Story*, o primeiro longa metragem 3D, em 1995. Na área de sistemas, surge a linguagem de programação Open GL em 1992 e as primeiras placas gráficas para PC da NVIDIA, em 1999. Com a virada para o ano 2000, a plataforma mais comum para actividades em computação deixa de ser as estações SGJ passando para PC. Em 2001, são lançados diversos sucessos de bilheteria, como *Shrek* (Dreamworks), com novos métodos de síntese e animação de personagens e *Final Fantasy*, o triunfo da modelagem de personagens 3D; também não poderíamos deixar de citar *Matrix Reloaded*, com personagens virtuais sendo usados, dentre outras coisas, para cenas de risco.

Escala Temporal

A escala temporal nos ajuda a identificar oportunidades e direcções de investigação e aplicação. Algumas das fundações que merecem destaque são: **Euclides [300-250a.C.]** – desenvolveu toda a geometria que norteou seu desenvolvimento até o século XVIII.

Brunelleschi [1377-1446] – arquiteto e escultor italiano que usou de forma criativa a noção de percepção visual, e criou em 1425 a perspectiva. **Descartes [1596-1650]** – matemático e filósofo francês que formulou a geometria analítica e os sistemas de coordenadas 2D e 3D. **Euler [1707-1783]** – o mais produtivo matemático do século XVIII, que, entre outros, criou o conceito de senos, tangentes, a expressão que relaciona o número de vértices, arestas e faces de poliedros etc.

Monge [1746-1818] – matemático francês que desenvolveu a geometria descritiva como um ramo da geometria. **Sylvester [1814-1897]** – matemático inglês que inventou as matrizes e a notação matricial, uma das ferramentas mais comuns da computação gráfica.

Hermite [1822-1901] – matemático francês que provou a transcendência do número e (usado como base para os logaritmos naturais) desenvolveu funções elípticas e curvas, entre outros. Continuando nossa escala temporal, podemos identificar os aspectos de mudança que são considerados marcos da investigação científica e suas principais aplicações nas indústrias e na sociedade.

- Em 1885, iniciou-se o desenvolvimento da tecnologia do tubo de raios catódicos;
- Em 1927, a indústria cinematográfica define o padrão de 24 imagens/segundo;
- Em 1930, P. e W. Mauchly constroem o primeiro computador chamado ENIAC;
- Em 1938, Valensi propõe o tubo de raios catódicos colorido;
- Em 1947, os Bell Labs inventam o transistor;
- Em 1950, Laposky cria as primeiras obras de arte com bases tecnológicas, usando um efeito de um osciloscópio;
- Em 1955, surge o sistema Sage de monitoramento aéreo;
- Em 1956, o MIT constrói o primeiro computador totalmente transistorizado;
- Em 1959, surge o termo *Computer Graphics*, criado por L. Hudson da Boeing;
- No final da década de 1950, as universidades e empresas americanas, como a Boeing, começam a usar computadores para testar idéias e novas aplicações; — Em 1960, é lançado o primeiro computador comercial DEC PDP-1; — Em 1961, no MIT é criado o primeiro jogo de computador (*Spacewars*) para o computador DEC PDP-1; — Em 1963, Sutherland apresenta um sistema de desenho interativo de primitivas gráficas 2D baseado em caneta luminosa;
- Em 1963, Englebart inventa o dispositivo de interação “mouse”;
- Em 1963, Zajac produz nos laboratórios da Bell o primeiro filme gerado por computador (imagens formadas de linhas e texto);
- Em 1963, surge o primeiro sistema comercial de CAD (*DAC-1*);
- Em 1963, Coons inventa a teoria de representação de superfícies curvas através de “retalhos” baseados em aproximações polinomial;

- Em 1965, Roberts cria um algoritmo de remoção de partes invisíveis de segmentos de reta e introduz a noção de coordenadas homogêneas na representação geométrica de objetos;
 - Em 1966, é lançado no mercado o primeiro console caseiro de jogos Odyssey;
 - Em 1966, surge a primeira empresa de produção computacional de animações e efeitos especiais, a MAGI;
 - Em 1967, Rougelet cria o primeiro simulador de voo interativo da NASA;
 - Em 1968, é fundada a Intel;
 - Em 1969, a MAGI produz para a IBM o primeiro comercial baseado em técnicas de computação gráfica;
 - Em 1969, é criado entre os grupos da ACM o Special Interest Group on Graphics SIGGRAPH;
 - Em 1969, nasce a ARPANET, rede percussora da Internet;
 - Em 1969, nos laboratórios da Bell, é construída a primeira matriz de pixels (cada pixel representado por 3 bits);
 - Em 1972, A. Kay, no Xerox PARC, produz o computador gráfico Alto;
 - Em 1972, Bushnell funda a empresa ATARI;
 - Em 1973, Metcalf desenvolve a tecnologia Ethernet e é editado o primeiro livro que aborda detalhadamente os algoritmos e métodos da computação gráfica (autores Newman e Sproull);
 - Em 1977, a Academia de Artes e Ciências Cinematográficas de Hollywood cria a categoria de Oscar de Efeitos Especiais;
 - Em 1979, G. Lucas contrata Catmull, Ray Smith e outros para uma nova empresa denominada Lucas Film;
 - Em 1974, Catmull desenvolve o algoritmo Z-Buffer;
- A partir do algoritmo de Z-Buffer e com o lançamento do PC no início da década de 1980, surge uma infinidade de aplicações e filmes baseados em computador. O mercado da computação gráfica atinge seu estágio de maturidade apresentando um grande crescimento com produções realistas e técnicas avançadas de iluminação e modelagem. São exploradas outras possibilidades de geometrias além do espaço tridimensional, que são utilizadas com uma frequência cada vez maior pelas pessoas que trabalham com artes, computação e visualização científica.

ÁREAS

A computação gráfica atualmente é uma área que engloba, para melhor descrição didática, pelo menos três grandes subáreas: a **Síntese de Imagens**, o **Processamento de Imagens** e a **Análise de Imagens**.

A **Síntese de Imagens** considera a criação sintética das imagens, ou seja, as representações visuais de objetos criados pelo computador a partir das especificações geométricas e visuais de seus componentes. Pode também ser descrita como **Visualização Científica ou Computacional**, principalmente quando se preocupa com a **representação gráfica** da informação, de forma a facilitar o entendimento de conjuntos de dados de alta complexidade, como, por exemplo, os dados de dinâmica dos fluidos, ou simulações espaciais.

O **Processamento de Imagens** considera o processamento das imagens na forma digital e suas transformações, por exemplo, para melhorar ou realçar suas características visuais.

A **Análise de Imagens** considera as imagens digitais e as analisa para obtenção de características desejadas, como, por exemplo, a especificação dos componentes de uma imagem a partir de sua representação visual.

MERCADO

Se você puder imaginar algo, isso pode ser gerado com a computação gráfica. A computação gráfica está a um passo de um mundo novo, repleto de aplicações, ainda desconhecidas, e muitas oportunidades de trabalho para designers, modeladores, animadores, iluminadores e programadores.

Toda essa realidade impulsiona a computação gráfica para o desenvolvimento de diversas aplicações 3D, hoje restritas a jogos. Para isso, foram criadas em diversos países, inclusive no Brasil, ferramentas SDK (*Software Development Kit*) capazes de simular fenômenos físicos, facilitar a criação de cenários, entre outros infinitos recursos que podem ser criados por qualquer pessoa e acoplados à engrenagem do SDK sob a forma de *plug-in*.

Algumas das aplicações mais evidentes no momento são os mercados em realidade virtual e *walk through* para projetos arquitetônicos. Na comunidade científica, é consenso que em breve os ambientes 3D modificarão os atuais sistemas operacionais, os bancos de dados e todos os componentes de interface que deverão ser recriados para esse novo mundo.

Programar para esse novo ambiente pode parecer um tanto hostil, exigindo dos programadores o conhecimento tanto de bibliotecas gráficas como *OpenGL* quanto da teoria da computação gráfica. Para designers, esse novo ambiente exigirá uma noção dos conceitos da computação gráfica e conhecimento de técnicas de modelagem, utilizando-se poucos polígonos ou operações *booleanas* e mapeamento com limitações no tamanho das imagens.

Diversos projetos já estão em desenvolvimento no Brasil, onde a mão-de-obra especializada é praticamente inexistente. A teoria da computação gráfica pode impulsionar sua carreira!

No mercado atual, a computação gráfica está presente em diversos segmentos, alguns dos quais são resumidos neste quadro:

Arte Efeitos especiais, modelagens criativas, esculturas e pinturas
Medicina Exames, diagnósticos, estudo, planejamento de procedimentos
Arquitetura Perspectivas, projetos de interiores e paisagismo
Engenharia Em todas as suas áreas (mecânica, civil, aeronáutica etc.)
Geografia Cartografia, GIS, georreferenciamento, previsão de colheitas
Meteorologia Previsão do tempo, reconhecimento de poluição
Astronomia Tratamento de imagens, modelagem de superfícies
Marketing Efeitos especiais, tratamento de imagens, projetos de criação
Segurança Pública Definição de estratégias, treinamento, reconhecimento
Indústria Treinamento, controle de qualidade, projetos
Turismo Visitas virtuais, mapas, divulgação e reservas
Moda Padronagem, estamparias, criação, modelagens, gradeamentos
Lazer Jogos, efeitos em filmes, desenhos animados, propaganda
Processamento de Dados Interface, projeto de sistemas, mineração de dados
Psicologia Terapias de fobia e dor, reabilitação
Educação Aprendizado, desenvolvimento motor, reabilitação

Além dessas aplicações que citamos, ainda existe uma série de fenômenos que só podem ser vistos com o auxílio da computação gráfica. Há algum tempo, era praticamente impossível estudar certos tipos de comportamentos que fugiam à percepção humana. Com o advento da computação gráfica e o aperfeiçoamento de técnicas e métodos computacionais, vários desses fenômenos podem agora ser visualizados, modelados e estudados. Muito antes de se conseguir visualizar um segmento de DNA humano, seu estudo já era possível através da computação gráfica. Em 1983, foi possível visualizar o vírus da AIDS e simular o seu comportamento. Através de dados científicos, foi criada uma imagem sintética que representava o vírus de forma adequada aos propósitos do estudo. Em um outro extremo, a computação gráfica nos permite estudar elementos que

possuem extrema complexidade como os Buracos Negros, objetos extremamente maciços que aprisionam tudo o que está próximo deles, inclusive a luz; ou objetos só realizáveis em dimensões superiores à quarta, como a garrafa de Klein; ou ainda as imagens adquiridas por sinais de galáxias distantes. Quando a imagem real não é suficiente ou mesmo inviável, a imagem sintética toma o seu lugar. Por exemplo, podemos corrigir imperfeições causadas por ruídos, falta de luz ou distorções nas imagens transmitidas por satélites, visualizar uma nuvem radioativa, como a de Chernobyl, ou enxergar o que ocorre no interior dos profundos poços de petróleo. A imagem sintética pode mesmo transformar qualquer dado em imagem, como os

sinais de radar ou calor dos corpos no interior de um prédio. Esse tipo de “visão” só pode ser realizada com o auxílio da computação gráfica. A área médica encontra na computação gráfica uma poderosa aliada. É possível simular o corpo humano e obter conclusões a partir disso. Utilizando uma combinação de dados, como os de ressonância magnética, ultra-som, ou tomográficos, é possível reconstituir tridimensionalmente qualquer parte do corpo, focalizando seus elementos e possíveis doenças ou distúrbios. Na meteorologia, ciclones, tufões, deslocamentos de massas de ar, além do estudo do aquecimento global e da camada de ozônio, podem ser representados para estudos e previsões.

14 Anexo B – Programa da disciplina

PROGRAMA

NO LECTIVO: 2006/2007

CURSO: Licenciatura em Multimédia

ANO CURRICULAR: 1.º

DISCIPLINA: TÉCNICAS DE ANIMAÇÃO GRÁFICA I

CRÉDITOS: 6

CÓDIGO:

DOCENTE(S): Jorge Mota, Engº

Objectivos Gerais:

- Aquisição de conhecimentos teóricos e práticos no desenvolvimento de projectos envolvendo técnicas de computação e animação gráfica.

Competências a desenvolver:

Desenvolver competências conceptuais, procedimentais e atitudinais de planeamento e desenvolvimento de aplicações gráficas em projectos multimédia, incluindo metodologias de desenvolvimento de aplicações, conceitos básicos de matemática e física aplicadas a algoritmos e programas gráficos, ActionScript/Flash 8, Raptor e RaptorGraph. Saber desenvolver e programar aplicações gráficas e jogos por computador em ambiente Flash8/ActionScript.

- Desenvolver competências básicas em termos de matemática e física aplicadas à computação gráfica, nomeadamente : trigonometria, vectores, matrizes, funções, estática, cinemática e dinâmica.
- Desenvolver competências no domínio do design de algoritmos e programas, em computação gráfica.
- Reconhecer e saber aplicar diferentes estruturas de dados em algoritmos gráficos.
- Saber trabalhar em Raptor e RaptorGraph no desenho de algoritmos, estruturas de dados e programas gráficos.
- Planear e desenvolver pequenos jogos a 2D para dispositivos de secretária ou móveis (PC, consolas ou dispositivos móveis).
- Saber trabalhar programáticamente com imagens gráficas, nomeadamente em operações de análise, edição e retoque de imagem.
- Conhecer algoritmos básicos de desenho e renderização gráficos (eq. Breshman, equações paramétricas de cónicas, transformações de cor, preenchimento de áreas, ordem de visualização, clipping)
- Esta disciplina articula de forma muito próxima com Web e Multimédia II , com a qual é partilhado a aprendizagem de ferramentas gráficas como Flash, ActionScript, Flex e proximidade no método e prática de desenvolvimento de programas gráficos.

Conteúdos Programáticos:

1. Introdução à computação gráfica:
 - a. História da computação gráfica
 - b. Áreas de aplicação
 - c. Tecnologias e arquitecturas de hardware
 - d. API gráficos modernos
2. Matemática aplicada a computação gráfica
 - a. Sistemas de coordenadas
 - b. Trigonometria
 - c. Representação de Funções (curvas)
 - i. Pontos
 - ii. Representação analítica: não paramétrica e paramétrica
 - iii. Curvas paramétricas de 3ª ordem
 - d. Vectores e Matrizes
 - e. Transformações geométricas
 - f. Matemática da animação (quadros, tempo e interpolação do movimento)
3. Ambiente de desenvolvimento algorítmico Raptor.
4. Estruturas de dados para computação gráfica.
5. Ambiente de desenvolvimento gráfico Raptor/RaptorGraph
6. O ambiente de desenvolvimento Flash 8.0
7. Introdução ao ActionScript 2.0
 - a. Forma geral de um programa em ActionScript (Frames loops e Clip events, Actions, Classes)
 - b. OOP em ActionScript (Classes, Construtores, herança, subClasses)
 - c. Tipos de dados simples
 - d. Estruturas de controlo de fluxo (sequência, decisão e repetição)
 - e. Técnicas básicas de Render
 - i. Cor
 - ii. API gráfica
 - iii. Transformações (de Cor e geométricas)
 - iv. Filtros
 - v. Bitmaps
8. Representação matricial e vectorial de imagens
9. Projecto 1: Tratamento de imagem em ActionScript (Editor de imagem).
10. Conceitos sobre física do movimento
 - a. Velocidade e aceleração
 - b. Forças, Fronteiras, fricção e inércia
 - c. Gravidade e massa
 - d. Elasticidade e plasticidade
 - e. Detecção de colisões
 - f. Saltos e ricochete
 - g. Cinemática directa e inversa
 - h. Física da Imagem estereoscópica
11. Usabilidade e interacção (ActionScript 2.0)
12. Luzes

Apontamentos de Apoio às aulas de Técnicas de Animação I, do Professor Jorge Mota 2007

13. Oclusão: Ordem de visualização (Backface culling e Depth Sorting)

14. Projecto 2: Jogo clássico para PSP em Flash Lite.

Critérios de Avaliação:

**Classificação final da disciplina = Classificação da prova final teórica (50%)
+
Classificação da prova final prática (50%)**

Duração média das provas: teórica (60 minutos) e prática (60 minutos)

Trabalhos práticos **individuais (1º trabalho pode ser feito em grupos de dois)** a serem realizados durante o semestre:

1º trabalho prático:

Tratamento de imagem em ActionScript 2.0 (Editor de imagem bitmap) –
classes e algoritmos e implementação em ActionScript 2.0.
Data de Entrega: 03 de Maio de 2007

2º trabalho prático

Implementação Jogo clássico para PSP em Flash Lite .
Data de Entrega: 20 de Junho de 2007

Os trabalhos tem a seguinte ponderação para a nota da componente prática final : **1º trabalho – 15% ; 2º Trabalho 35%**).

Os trabalhos são entregues **obrigatoriamente** na secretaria, contendo o relatório em papel e em formato digital e o código fonte.

O aluno que realize os trabalhos propostos nas condições aprovadas podrá optar pela não realização da prova prática final. A nota dos trabalhos, caso o aluno opte por esta modalidade será conservada até ao final do ano lectivo (ou seja será mantida em todas as épocas de exames do ano lectivo corrente).

Bibliografia / Sítios da Internet/Recursos Educativos :

Bibliografia:

Fundamental

- Peters, Keith; "Actionscript Animation", Editora Friends of Apress, 2006
- Conci, ; Computação gráfica Teoria e Prática, Editora Campus, 2003

Complementar

- Mota, Jorge, " Elementos de Algoritmia – ISTECS", Edição ISTECS (PDF), 2006
- David, Eberly, "3D Game engine", Morgan Kaufman, 2002
- ISTECS, COMPUTAÇÃO GRÁFICA, ISTECS- ACADEMIA DE SOFTWARE, 2002
- Harold, Santo, " Métodos Gráficos e Geometria Computacionais", DinaLivro, 1985

Recursos Educativos:

- Portfólio "Computação gráfica" da Academia de Software
- Site do professor : <http://www.jorgemota.com>
- Documentação do professor
- Conteúdos on-line disponíveis em www.wikipedia.com
- Sistema de ajuda electrónica da Adobe

Software e Equipamento necessário para a leccionação:

- Laboratório com computadores pessoais ligados em rede e com acesso à Internet com sistema operativo Windows XP ou equivalente, incluindo browser IE.
- Quadro Branco de grande dimensão
- Videoprojector
- Flash Studio 8.0
- Raptor e RaptorGraph
- PhotoShop ou outro programa de tratamento de imagem.

15 Anexo C – Teste Tipo

Engenharia Multimédia Curso de Informática	2º Ano - 1º Semestre
Técnicas de Animação Gráfica	Tipo
Data : 20/6/2007	Duração : 60 Minutos
Parte Teórica	Prof. : Jorge Mota

Numero : _____ Nome : _____

Pergunta 1

Considere que se encontra a desenvolver um pequeno jogo de bilhar livre para uma PSP.

- Implemente em actionScript um programa que permite detectar a colisão entre duas bolas A e B com o máximo realismo em termos de física do movimento (as bolas são objectos rígidos). (2 valores)
- Apresente uma alternativa ao método usado na alínea a). Justifique porque escolheu esta alternativa. (2 valores)

Pergunta 2

O flash suporta o traçado de splines.

- Explique o que são Splines? Inclua um pequeno desenho explicativo.(1 valor)
- Como se criam splines programáticamente em ActionScript. (1 valor)
- Dê um exemplo de aplicação de splines num trabalho prático real. (1 valor)

Pergunta 3

Explique como converter uma imagem bitmap colorida usando um padrão 24 bits (RGB) numa imagem em tons de cinza (escala de cinzentos). (2 Valores)

Pergunta 4

a)Analise o seguinte algoritmo em raptor e reescreva-o substituindo o ciclo for por um REPETIR ... ATÉ QUE. (2 valores)

Pergunta 5

Represente a matriz genérica de transformação de rotação (MTRX(?)), de forma que esta permita por uma simples operação de transformação rodar o ponto $P(x,y,z,1)$ de ? graus em torno de Z (2 valores):

$$P'(X',Y',Z',1) = P(X,Y,Z,1) \times \text{MTRX}(?)$$

Pergunta 6

Compare as características da API gráfica avançada como DirectX 9 e o Flash/ActionScript_ em termos de aplicações gráficas no domínio dos videoJogos 2D. Enumere vantagens e inconvenientes nas diversas vertentes que abordar.(2 valores)

Pergunta 7

Explique o algoritmo de traçado de rectas conhecido por DDA. Faça um esquema explicativo para a sua explicação.(2 valores)

Pergunta 8

Crie um procedimento em actionscript que permita carregar um textura Jpeg da pasta .\imagens e a mostre no palco.(3 valores)

16 Bibliografia

- [1] Conci, ; Computação gráfica Teoria e Prática, Editora Campus, 2003
- [2] Peters, Keith; "Actionscript Animation", Editora Friends of/Apress, 2006
- [3] Mota, Jorge, " Elementos de Algoritmia – ISTECS", Edição ISTECS, 2006
- [4] ISTECS, Computação Gráfica, ISTECS- ACADEMIA DE SOFTWARE, 2002
- [5] Harold, Santo, " Métodos Gráficos e Geometria Computacionais", Dina Livro, 1985
- [6] Mook, Collin, " Essencial ActionScript 2.0", O'Reilly, 2004
- [7] Junior, Annibal Hetem, "Computação Gráfica", LTC, 2006
- [8] Lever, Nik, "FlashMX 2004 Games – Art to ActionScript", Editora Focal Press, 2004
- [9] Leggett, Richard, Foundations Flash Applications for Mobile Devices, Editora Friends of/Apress, 2006
- [10] Mota, Jorge, Desenvolvimento em Flash para PSP, ISTECS, 2007
- [11] Salomon, David, Curves and Surfaces for Computer Graphics, Editora Springer, 2006
- [12] Phillips, Dwayne, Image Processing in C, Editora R & D Publications, 2000
- [13] Flynt, John, Software Engineering for game developers, Editora Thomson, 2005