Algoritmo de Visão para Humanos Virtuais

Pedro Semião, Maria Beatriz Carmo, Ana Paula Cláudio

Faculdade de Ciências da Universidade de Lisboa
i21197@alunos.di.fc.ul.pt, {bc, apc}@di.fc.ul.pt

Sumário

A plataforma IViHumans destina-se a suportar a geração de cenas animadas com humanos virtuais inteligentes. Esta plataforma é constituída por dois módulos independentes que comunicam entre si: a componente do processamento gráfico e a componente do processamento inteligente. Neste artigo apresenta-se um algoritmo de visão sintética, baseado em ray casting, integrado nesta plataforma. Através deste algoritmo um humano virtual pode reconhecer o ambiente que o rodeia permitindo a interacção com os elementos da cena.

Palavras Chave

Humanos virtuais, visão sintética, ray casting

1. INTRODUÇÃO

A construção de ambientes virtuais onde se movimentam humanos virtuais é um tema de estudo com aplicação em várias áreas: entretenimento, educação, treino, quer em situações de emergência, quer em aplicações militares ou em ambientes industriais, reconstrução de ambientes históricos, aplicações médicas, tratamento de fobias, planeamento urbanístico entre outras.

Para gerar um agente virtual, além de uma aparência física realista, é necessário dotá-lo também de uma personalidade própria capaz de exprimir emoções e interagir com o meio que o rodeia. Um aspecto chave para a concretização da interacção com o meio é a incorporação de mecanismos que permitam captar o mundo envolvente, simulando a forma como os humanos apreendem o meio ambiente. Embora já haja estudos que se debruçam sobre o desenvolvimento de sensores tácteis e auditivos [Noser95], os sensores de visão são os mais desenvolvidos. Isto porque, por um lado, requerem, normalmente, equipamento menos sofisticado, por outro lado, extraem informação mais completa sobre os objectos que rodeiam o agente.

Neste artigo descreve-se um mecanismo de visão integrado na plataforma IViHumans (**I**ntelligent **Vi**rtual **Humans**) desenvolvida para suportar a geração de cenas animadas com agentes inteligentes [Silvestre05] [Carmo05]. Para simplificar a linguagem, designaremos no texto humanos virtuais inteligentes por agentes.

Há diferentes abordagens para a concretização de algoritmos de visão. Na secção 2, identificam-se os tipos de soluções encontradas na literatura. Na secção 3, descreve-se a plataforma IViHumans onde o algoritmo desenvolvido se enquadra. Na secção 4, apresenta-se o algoritmo implementado. Finalmente na secção 5 conclui-se descrevendo o trabalho em curso e o trabalho futuro.

2. MECANISMOS DE VISÃO

Analisando os algoritmos de visão desenvolvidos, identificam-se dois tipos principais que podem ser designados por omniscientes (*sensory omniscience*) e honestos (*sensory honesty*)

Os algoritmos designados por omniscientes têm acesso directo à descrição da cena. Assim o agente tem informação sobre todos os objectos, incluindo aqueles que não consegue ver, por exemplo, os que estão atrás de uma parede. Apesar deste tipo de visão ser mais fácil de concretizar e ser mais eficiente em termos de tempo, é mais susceptível de produzir resultados irrealistas.

Os algoritmos honestos [Burke01] tentam obter apenas a informação que está no campo de visão do agente. Deste modo, o agente só reage perante os objectos que estão realmente visíveis, atingindo-se uma interacção mais realista. Esta categoria de algoritmos pode subdividir-se em dois grupos: visão artificial e visão sintética.

Designam-se por algoritmos de visão artificial aqueles que se baseiam no reconhecimento de imagens [Noser95]. Neste caso, o agente conhece apenas a informação extraída a partir de imagens 2D. Este processo requer a utilização de algoritmos complexos e que exigem, normalmente, muito tempo de processamento. Esta técnica é sobretudo utilizada na robótica em situações em se pretende reconhecer um ambiente real à custa de imagens capturadas por uma câmara.

O conceito de visão sintética foi introduzido por Renault et al. [Renault90], correspondendo à simulação da visão do agente através de uma câmara localizada ao nível dos seus olhos. A partir da geração da imagem, usando o ponto de vista do agente, obtêm-se os objectos visíveis. Além disso, acedendo à descrição da cena, é possível consultar, nomeadamente, a posição dos objectos e informação semântica disponível. Deste modo, obtém-se

mais informação sobre o ambiente e de forma mais rápida [Peters02].

Para obter informação a partir da imagem gerada, Renault et al. [Renault90] propuseram uma técnica baseada em cores falsas. Isto é, os objectos da cena não são desenhados com a cor definida na descrição da cena, mas usando uma cor que identifica univocamente o objecto. O resultado da geração da cena segundo o ponto de vista do agente é guardado numa matriz bidimensional que contém em cada posição informação sobre o valor do *pixel*, a identificação do objecto a que corresponde o valor do *pixel* e a sua distância aos olhos do agente. A dimensão da matriz tem de ser escolhida de modo a identificar os objectos com uma precisão razoável, mas sem sobrecarregar muito o sistema. Renault et al. usaram uma matriz 30x30 para aplicar esta técnica à detecção de obstáculos num corredor. Noser et al [Noser95] usaram uma matriz 50x50 e estenderam esta técnica juntando um modelo de memória visual baseado numa *octree*. Kuffner e Latombe [Kuffner99] aplicaram esta técnica adoptando uma matriz de 200x200 *pixels*. Neste caso considerou-se que o cenário era composto por elementos de pequenas dimensões. Objectos maiores como, por exemplo, paredes foram subdivididos em vários objectos. A cada objecto foi atribuído um identificador único. Neste caso, a memória visual era simulada através de uma lista que guardava os identificadores dos objectos visíveis.

Peters e Sullivan [Peters02] estenderam a técnica de cores falsas definindo modos de visão diferentes, recorrendo a paletes de cores distintas: modo individual (*distinct mode*) e modo em grupo (*grouped mode*). No modo individual, a cada objecto é atribuída uma cor. No modo em grupo, a mesma cor é partilhada por um grupo de objectos que satisfazem um critério previamente estabelecido. Este último modo é menos preciso porque basta que um dos elementos do grupo esteja visível para que o grupo seja assinalado como visível. Estes autores consideraram uma matriz de 128x128 *pixels*.

Terzopoulos et al. [Terzopoulos96] adoptaram um sensor de visão cobrindo um ângulo de 300º para simular a visão de peixes. A visão de cada peixe é obtida através de quatro câmaras virtuais que geram imagens com diferentes resoluções e campos de visão. Cada peixe é capaz de reconhecer um conjunto de modelos de cor de objectos em que está interessado.

Inspirando-se em modelos sobre o movimento de abelhas, Blumberg [Blumberg96] propôs um modelo de visão sintética baseado no equilíbrio da energia do movimento. A cena é gerada do ponto de vista do agente e o valor de cada *pixel* em *frames* sucessivas é usado para determinar a energia do movimento, bem como outras características do ambiente.

Uma técnica diferente de visão sintética, baseada em lançamento de raios (*ray casting*), é apresentada por Vosinakis e Panayiotopoulos [Vosinakis05]. Um conjunto de raios é lançado para a cena dentro do campo de visão do agente. Para cada raio determina-se a posição, tamanho e tipo dos objectos que são intersectados pelos raios.

O algoritmo de visão descrito na secção 4 baseia-se no lançamento de raios.

3. A PLATAFORMA IVIHUMANS

Para gerar humanos virtuais com aparência e comportamento credíveis, além de reproduzir o aspecto humano de forma realista, é necessário criar mecanismos para simular o comportamento humano. Para atingir este objectivo, a plataforma IViHumans tem duas componentes fundamentais: a componente gráfica e a de inteligência artificial. A componente gráfica suporta a modelação e animação quer do ambiente quer dos humanos virtuais. A componente de inteligência artificial é responsável pelo comportamento do humano virtual.

Uma vez que o processo de desenvolvimento de ferramentas gráficas e de inteligência artificial é normalmente moroso e exigente do ponto de vista de recursos humanos, optou-se por incorporar aplicações já existentes e de domínio público. Para a componente de processamento gráfico é necessário considerar as capacidades de modelação e de geração de imagens em tempo-real, bem como, a simulação do comportamento físico dos objectos da cena. Para preencher estes requisitos foram seleccionadas as seguintes aplicações: o Blender [hB] para a modelação 3D, o OGRE [hOG] como motor gráfico e o ODE [hOD] como motor físico. Na realidade a modelação pode ser efectuada por qualquer aplicação que exporte o modelo construído num formato que possa ser lido pelo motor gráfico (OGRE).

Para a componente de inteligência artificial foi adoptada a bancada de agentes JADE (Java Agent Development Framework) [hJ].

Como as aplicações gráficas adoptadas estão implementadas em C++ e o JADE está implementado em Java, a comunicação entre as duas componentes é feita através de *sockets*.

Uma das características principais desta plataforma é a sua modularidade. A integração de um módulo para a simulação de visão é uma peça essencial para a interacção dos agentes entre si e com o ambiente que os rodeia.

4. ALGORITMO DE VISÃO

Considerou-se que seria mais adequado incorporar na plataforma um algoritmo de visão do tipo honesto. Ou seja, um agente vê apenas os objectos que estão no seu campo de visão e identifica-os através da imagem que é gerada a partir de uma câmara colocada ao nível dos seus olhos. Uma forma de garantir que todos os objectos são detectados consiste em varrer todos os *pixels* da imagem e identificar os objectos representados. Mas esta aproximação seria muito lenta. Nos exemplos apresentados por outros autores, o maior número de *pixels* tratados foi 200x200. Podemos considerar como razoável estabelecer um intervalo de tempo dentro do qual um agente deve tomar conhecimento da entrada de um objecto no seu campo de visão. Por exemplo, considerando um segundo como intervalo de tempo razoável para detectar um objecto, o varrimento da

imagem pode ser efectuado ao longo das *frames* que são geradas durante um segundo. Se forem geradas X *frames* por segundo (FPS), o varrimento da imagem é feito em X *frames*. No entanto, se o número de *frames* por segundo diminuir, pode haver objectos que não são detectados no tempo pretendido. Para minorar este problema, o algoritmo de visão deve adaptar o varrimento da imagem de acordo com o número de FPS para manter um bom desempenho.

No motor gráfico escolhido para a plataforma havia duas formas de satisfazer os objectivos acima descritos: consultar o buffer de profundidades (Z-Buffer) ou lançar raios para a cena. Uma vez que a primeira solução ficaria muito dependente do hardware utilizado, entendeu-se que a segunda abordagem seria melhor. Com o lançamento de raios a partir da posição dos olhos do agente para pontos do *viewport* é possível identificar os objectos que são intersectados pelos raios.

Após a realização de alguns testes, verificou-se que o motor gráfico (OGRE) não fazia intersecções precisas entre o raio e objecto. O OGRE faz a intersecção do raio com a caixa envolvente do objecto (*bounding box*). Este volume é normalmente muito maior que o objecto o que conduz a situações de erro: a caixa envolvente pode ser intersectada, mas o objecto não ser efectivamente intersectado pelo raio. Se este objecto estiver à frente de outro, gera-se uma situação de falsa oclusão. Para obter intersecções mais precisas, adoptou-se a biblioteca OpCode [hOC] Optimized Collision Detection, que permite obter caixas envolventes muito próximas da forma do objecto. Utilizou-se a biblioteca OgreOpCode [hOGC] para estabelecer a ligação entre o OGRE e o OpCode.

Resolvidos os problemas de detecção correcta de intersecções, foi necessário definir quantos raios lançar, uma vez que não é aceitável em tempo-real lançar raios para todos *pixels* da imagem. Tendo em linha de conta que as alterações detectadas em *frames* sucessivas são, normalmente, pequenas, concebeu-se uma solução que divide os raios a enviar para a cena em *frames* sucessivas [Semião06]. Na subsecção 4.1 descreve-se o algoritmo de visão baseado em lançamento de raios e nas subsecções 4.2 e 4.3 apresentam-se soluções para a criação, respectivamente, de pontos alvo e de vectores de deslocamento. Na subsecção 4.4 indicam-se alguns detalhes de implementação.

4.1 Algoritmo de Visão Baseado em Lançamento de Raios

O algoritmo de visão proposto divide o lançamento de raios por *frames* sucessivas. Para este efeito considerou-se uma matriz, M, que em cada linha contém os pontos a utilizar numa *frame*. Ou seja, se, por exemplo, a matriz tiver dimensão p x q, então tem os pontos a utilizar em p *frames*, e em cada *frame* serão lançados q raios na direcção dos q pontos definidos na respectiva linha da matriz. Definiu-se também um vector AΔ que contém os deslocamentos a aplicar aos pontos de M em sucessivas iterações. O primeiro deslocamento de AΔ é (0, 0) de modo a usar sem alterações os pontos da matriz M.

De uma maneira mais formal, consideremos

M [L, C] uma matriz de pontos 2D com L linhas e C colunas, ou seja, relativa a L *frames* com lançamento de C raios em cada *frame*

e AΔ[S] um vector com S vectores de deslocamento

A matriz M define L*C pontos alvo a atingir em L frames. Combinando os pontos da matriz M com os deslocamentos do vector de deslocamentos AΔ, haverá L*C*S pontos que serão atingidos por raios em L*S *frames*, uma vez que cada vector de AΔ será aplicado a todos os pontos da matriz M.

Para uma dada *frame* F, o conjunto de pontos atingidos será também usado na *frame* F+(L*S*K), onde K é um valor inteiro positivo. Isto é, ao fim de L*S *frames* volta a repetir-se o mesmo conjunto de pontos alvo.

Assim, o conjunto de pontos a usar na *frame* F é

$$\{M [H, i] + A\Delta[D], i=1,2,...,C\}$$

onde

$$H= ((F-1) \bmod L) + 1,$$

$$D= (((F-1) /L) \bmod S) + 1 \text{ e}$$

/ representa a divisão inteira

Consideremos o seguinte exemplo para ilustrar o método adoptado.

Dada a matriz

$$M [2,2] = \{ \{ (0.4, 0.4) , (0.6, 0.6) \} , \{ (0.4, 0.6) , (0.6, 0.4) \} \}$$

e o vector de deslocamentos

$$A\Delta = \{ (0, 0) , (0.01, 0.01) , (-0.02, 0.02) \}$$

O conjunto de pontos para onde são lançados raios em cada *frame* é mostrado na tabela 1. Quando o último ponto é atingido, o processo repete-se novamente a partir do primeiro.

***Frame* id**	**Conjunto de pontos por *frame***
1	(0.4, 0.4), (0.6, 0.6)
2	(0.4, 0.6), (0.6, 0.4)
3	(0.41, 0.41), (0.61, 0.61)
4	(0.41, 0.61), (0.61, 0.41)
5	(0.38, 0.42), (0.58, 0.62)
6	(0.38, 0.62), (0.58, 0.42)

Tabela 1

Falta ainda estabelecer a forma de gerar os pontos que constituem a matriz M e os vectores de deslocamento do vector AΔ. Nas subsecções seguintes apresenta-se uma solução possível.

4.2 Geração de Pontos Alvo

O sistema de visão humano capta mais informação na zona central do campo de visão e tem uma visão periférica muito limitada [Peters02]. Para simular a visão humana através de raios lançados do ponto de vista do agente, interessa lançar um maior número de raios na zona central da imagem. Para atingir este objectivo, vai definir-se uma matriz de pontos alvo (MPA) com uma distribuição simétrica nos quatro quadrantes da imagem, considerando distâncias menores entre pontos adjacentes no centro da imagem.

Para definir as coordenadas dos pontos alvo dos raios a lançar a partir do ponto de vista do agente, considera-se um sistema de coordenadas intermédio independente da dimensão do *viewport*. Neste sistema de coordenadas a imagem é um quadrado unitário com origem no extremo superior esquerdo (Fig. 1).

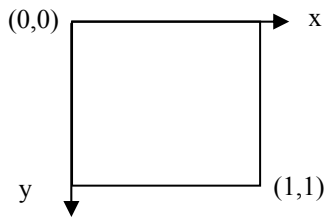

Fig. 1 –Sistema de coordenadas intermédio

Deste modo, para definir a matriz de pontos alvo com coordenadas simétricas relativamente a dois eixos ortogonais com origem no centro do quadrado, basta considerar o mesmo vector, P, para as componentes em xx e em yy, gerando a matriz de pontos como uma grelha rectilinear [Speray90]. Por exemplo, para gerar uma matriz MPA 7x7, consideremos o seguinte vector

$$P=[0.025, 0.25, 0.4, 0.5, 0.6, 0.75, 0.975]$$

obtendo cada ponto da matriz à custa da expressão

$$MPA_{ij}= (P_i,P_j) , i,j=1, 2, .., 7$$

Assumindo que as coordenadas são simétricas relativamente ao centro da imagem, como acontece no exemplo anterior, basta definir metade das coordenadas do vector P para obter todos os pontos.

Para que a distância entre pontos adjacentes aumente para pontos mais afastados do centro da imagem, vamos considerar que a distância entre os pontos aumenta em progressão geométrica com razão superior a 1. Na fase actual de testes, considerou-se o valor 2 como limite superior da razão da progressão, para garantir que na periferia da imagem também sejam lançados raios.

Como interessa adaptar o número de pontos da matriz de modo a obter um número razoável de FPS, a geração da matriz MPA vai ficar condicionada, não só pela razão, k, da progressão geométrica a aplicar à distância entre os pontos, mas também pelo número de pontos que se pretende gerar. Para gerar um número ímpar de pontos por linha, vamos considerar que o centro da imagem, (0.5, 0.5), pertence à matriz MPA. Como foi referido anteriormente, no vector P há simetria relativamente à origem. Assim para criar um vector P com um número ímpar de coordenadas, n, uma vez que é conhecida a coordenada central, 0.5, basta gerar m=(n-1)/2 coordenadas na metade direita da imagem, ou seja, com valores a variar entre 0.5 e 1. As coordenadas na metade esquerda da imagem são obtidas por simetria relativamente ao ponto central.

Para garantir a geração de m coordenadas em metade da dimensão da imagem, é necessário que a soma das distâncias entre os pontos não ultrapasse 0.5. Além disso, como se pretende aplicar deslocamentos aos pontos da matriz inicial, considera-se uma distância δ entre o último ponto gerado e o lado da janela de modo a que o deslocamento não coloque os pontos fora da imagem. Atendendo que as distâncias formam uma progressão geométrica, determina-se a distância entre dois pontos adjacentes no centro da imagem, d_1, resolvendo a seguinte inequação

$$d_1*(1-k^m)/(1-k)< 0.5-\delta \quad (1)$$

As distâncias entre coordenadas sucessivas são obtidas pela equação

$$d_{i+1}= d_i*k, \quad i=1, 2, \ldots, m \quad (2)$$

Sendo $x_{m+1}= 0.5$ o centro da imagem, as coordenadas à direita do ponto médio são dadas por

$$x_{m+1+j}= x_{m+j}+d_j, \quad j=1,2,\ldots, m \quad (3)$$

e as coordenadas à esquerda do ponto médio são dadas por

$$x_{m+1-j} = x_{m+2-j}-d_j, \quad j=1,2,\ldots, m \quad (4)$$

No caso do número de coordenadas, n, de P ser par, é necessário criar m=n/2 coordenadas em cada metade. Neste caso, o ponto médio da imagem, 0.5, corresponderá ao ponto médio entre os pontos com menor afastamento, d_1. Ou seja, é necessário garantir que em metade da dimensão da imagem cabe a soma de $d_1/2$ com os termos sucessivos da progressão geométrica das distâncias entre os pontos. Assim, determina-se d_1 resolvendo a seguinte inequação

$$d_1*(1-k^m)/(1-k) - d_1/2 < 0.5-\delta \quad (5)$$

As coordenadas do vector P, correspondentes às coordenadas à direita do ponto médio da imagem, são dadas por

$$x_{m+1}= 0.5 + d_1/2 \quad (6)$$

$$x_{m+1+j}=x_{m+j}+ d_{j+1}, \quad j=1,2,\ldots, m-1 \quad (7)$$

e as coordenadas do vector P, correspondentes às coordenadas à esquerda do ponto médio da imagem, são dadas por

$$x_m= 0.5 - d_1/2 \quad (8)$$

$$x_{m-j}=x_{m+1-j} - d_{j+1}, \quad j=1,2,\ldots, m-1 \quad (9)$$

A Fig. 2 mostra o resultado obtido com a geração de uma matriz 7x7 considerando k=1.5.

Como relacionar a matriz MPA com a matriz M? Se não houver degradação de desempenho ao tratar numa só *frame* todos os pontos da matriz MPA, pode considerar-se que a matriz M tem apenas uma linha que

contém todos os pontos da matriz MPA. Caso contrário, será necessário repartir os pontos da matriz MPA por várias linhas da matriz M, de modo a serem tratados em diferentes *frames*.

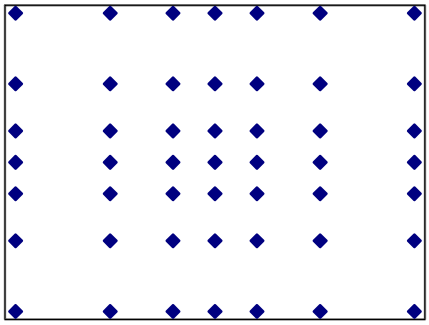

Fig. 2 – Distribuição dos pontos da matriz de pontos alvo, MPA, usando n=7 e k=1.5

4.3 Vector de Deslocamentos

A missão do vector de deslocamentos é provocar a variação dos pontos alvo em iterações sucessivas de modo a atingir o maior número de pontos da imagem.

Como já foi mencionado, a primeira componente deste vector é o vector (0, 0) para permitir em primeiro lugar lançar raios para os pontos calculados.

Os elementos seguintes do vector de deslocamentos devem ser escolhidos de modo a evitar que elementos sucessivos tenham a mesma direcção, para diminuir a probabilidade de o vector de deslocamentos seguir a direcção do movimento do agente em passos sucessivos. Devem também tentar cobrir de uma forma uniforme o espaço circundante aos pontos definidos na matriz MPA.

De modo a preencher estes requisitos foi delineado um algoritmo que tem por base a variação em espiral dos extremos dos vectores de deslocamento.

A geração dos vectores de deslocamento depende de algumas variáveis que podem ser ajustadas para controlar a distribuição dos pontos.

Sejam

$\Delta max = d_1$ – Norma do maior vector de deslocamento, corresponde à menor distância entre quaisquer dois pontos da matriz MPA

N – Número de vectores de deslocamento a calcular

$\Delta = \Delta max/N$ – Incremento da norma entre dois vectores de deslocamento consecutivos

α – Ângulo do vector de deslocamento

αInc – Incremento ao ângulo do vector de deslocamento

R – Factor para a redução de αInc ($0 < R < 1$)

Para gerar um novo vector, precisamos de saber a norma do último gerado. Esta informação será guardada na variável Δtotal, que é inicializada a zero.

O algoritmo consiste em iterar as seguintes acções N vezes:

Passo 1 - Guardar vector de deslocamento (Cos(α) * Δtotal , Sin(α) * Δtotal).

Passo 2 - Incrementar α em αInc graus.

Passo 3 - Δtotal toma o valor de Δtotal + Δ

Passo 4 - Se α é maior que 360º:

α toma o valor de (α – 360)

αInc toma o valor de αInc * R

Passo 5 - Voltar ao passo 1

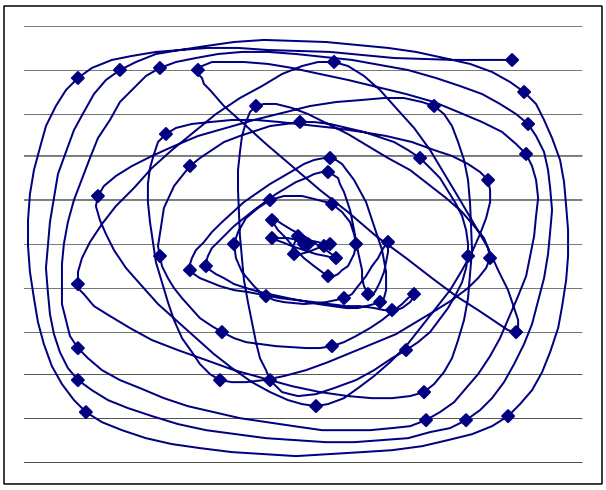

Fig. 3 – Vectores de deslocamento: sequência de extremos finais gerados com R=0.8, N=60, α = 0ºe αInc=90º

A Fig. 3 mostra os extremos dos vectores de deslocamento gerados considerando os seguintes valores iniciais: R= 0.8, N=60, α = 0ºe αInc=90º. Cada pequeno losango da figura representa o extremo final de um vector de deslocamento cuja origem é o ponto (0,0), que corresponde ao centro da figura. A linha une os pontos que representam os extremos dos vectores de deslocamento pela ordem em que foram gerados.

Como já foi mencionado, a primeira componente do vector de deslocamentos é o vector (0, 0) para permitir que em primeiro lugar sejam lançados raios para os pontos calculados para a matriz MPA.

No passo 4, o ângulo αInc é actualizado ao fim de cada volta percorrida pelo ângulo α. Deste modo, garante-se que, por um lado, em voltas sucessivas do ângulo α não sejam gerados vectores de deslocamento alinhados sobre semi-rectas com origem nos pontos alvo, e por outro lado, sejam gerados mais vectores em cada volta, à medida que o número de voltas aumenta, para compensar o alargamento da espiral que este gerador tenta aproximar.

Consideraram-se apenas valores positivos para α e αInc, uma vez que valores negativos destes ângulos apenas modificariam a ordem de geração dos vectores. No entanto, basta alterar, no passo 4, o teste de conclusão de uma volta, para permitir a utilização de valores negativos para qualquer destes ângulos.

4.4 Implementação

Foram realizados alguns testes iniciais para verificar o desempenho do algoritmo, variando o número de objectos na cena e o número de raios lançados para a cena. Como exemplo, mostramos os resultados da tabela 2 com o número de FPS obtidas, fazendo variar o número de objectos e o número de raios. A máquina usada no teste é um Pentium 4 3.40GHz com 1GB de DDR2 RAM e uma placa gráfica NVIDIA GeForce 7800 GTX (Gainward Golden Seal). Foi usado Direct3D9 com uma resolução de 800x600 em *windowed mode*. A cena tem em média 10000 triângulos.

Apesar de os testes terem sido feitos com poucos objectos e lançando poucos raios, o número de FPS é muito elevado, permitindo por isso aumentar, quer o número de raios, quer o número de objectos.

		Número de Raios	
		30	50
Número de Objectos	6	284	264
	12	279	261
	18	267	249

Tabela 2 – resultados em FPS

5. CONCLUSÕES E TRABALHO FUTURO

Apresentou-se um algoritmo de visão sintética baseado em *ray casting*. Este algoritmo divide os raios a lançar por sucessivas *frames*. Deste modo pretende-se optimizar a detecção de objectos sem degradar o desempenho.

Este algoritmo apresenta soluções em termos do cálculo, quer dos pontos a atingir pelos raios, quer dos deslocamentos a aplicar a estes pontos em sucessivas *frames*.

Um dos problemas que normalmente se encontra neste tipo de algoritmos de visão é a não detecção de alguns objectos mais pequenos ou mais distantes. Os algoritmos para geração de pontos alvo e de vectores de deslocamentos têm vários parâmetros ajustáveis. Como trabalho futuro pretende-se realizar testes de modo a adequar estes parâmetros para resolver o problema acima descrito.

Outro aspecto a considerar prende-se com a realização de testes para aprofundar o conhecimento sobre a relação entre o número de FPS com o número de objectos em cena e o número de raios lançados.

O resultado dos testes permitirá conjugar todo o trabalho realizado até ao momento de forma a conseguir um algoritmo adaptável, que consiga manter o número de FPS estável, perto de valores definidos pelo utilizador.

6. REFERÊNCIAS

[Blumberg96] Blumberg, B.: Old Tricks, New Dogs: Ethology And Interactive Creatures. PhD Dissertation, MIT Media Lab, 1996

[Burke01] Burke, R., Isla, D., Downie, M., Ivanov, Y., Blumberg, B.: Creature Smarts: The Art and Architecture of a Virtual Brain, The Game Developers Conference, pp 147-166, 2001

[Carmo05] Carmo, M. B., Cláudio, A. P., Cunha, J. D., Coelho, H., Silvestre, M., Pinto-Albuquerque, M.: Plataforma de Suporte à Geração de Cenas Animadas com Agentes Inteligentes. In Actas do 13º Encontro Português de Computação Gráfica, pp. 79-84, 2005

[hB] http://www.blender3d.org/About/?sub=Features

[hJ] http://jade.tilab.com/

[hOC] http://www.codercorner.com/Opcode.htm

[hOD] http://ODE.org/

[hOG] http://www.ogre3d.org

[hOGC]http://conglomerate.berlios.de/wiki/index.php/OgreOpcode_Introduction

[Kuffner99] Kuffner, J.J., Latombe, J.-C., Fast Synthetic Vision, Memory and Learning Models for Virtual Humans, Proceedings Computer Animation 1999, IEEE International Conference on Computer Animation, pp 118-127, 1999

[Noser95] Noser, H., Thalmann, D.: Synthetic Vision and Audition for Digital Actors. Proc. Eurographics '95, Maastricht, pp 325-336, 1995

[Peters02] Peters, C., O'Sullivan, C.: Synthetic Vision and Memory for Autonomous Virtual Humans, Computer Graphics Forum, vol. 21, issue 4, pp 743-752, Dez. 2002

[Renault90] Renault, O., Magnenat-Thalmann, N., Thalmann, D.: A Vision-based Approach to Behavioural Animation. Journal of Visualization and Computer Animation, vol. 1, nº 1, pp.18-21, 1990

[Semião06], Semião, P. M., Carmo, M. B., Cláudio, A. P., Implementing Vision in the IViHumans Platform, SIACG 2006, Ibero-American Symposium in Computer Graphics, *short paper*, pp 56-59, 2006

[Speray90] Speray, D., Kennon, S., Volume Probes: Interactive Data Exploration on Arbitrary Grids, Computer Graphics, vol. 24, nº5, Nov. 1990

[Silvestre05] Silvestre, M., Pinto-Albuquerque, M., Carmo, M. B., Cláudio, A. P., Cunha, J. D., Coelho, H: Platform for the Generation of Virtual Environments Inhabited by Intelligent Virtual Humans. In Proc. ACM ITiCSE'05, *student posters*, pp 402, 2005.

[Terzopoulos96] Terzopoulos, D., Rabie, T., Grzeszczuk, R.: Perception and Learning in Artificial Animals. Artificial Life V: Proc. Fifth International Conference on the Synthesis and Simulation of Living Systems, Nara, Japan, May, pp 313-320, 1996

[Vosinakis05] Vosinakis, S., Panayiotopoulos, T.: A Tool for Constructing 3D Environments with Virtual Agents. Multimedia Tools and Applications, 25, pp 253-279, 2005