

Posicionamento de Câmeras através de Previsão das Simulações Físicas

Daniel Pires Erick Passos Esteban Clua Anselmo Montenegro

Universidade Federal Fluminense, Instituto de Computação, Brasil

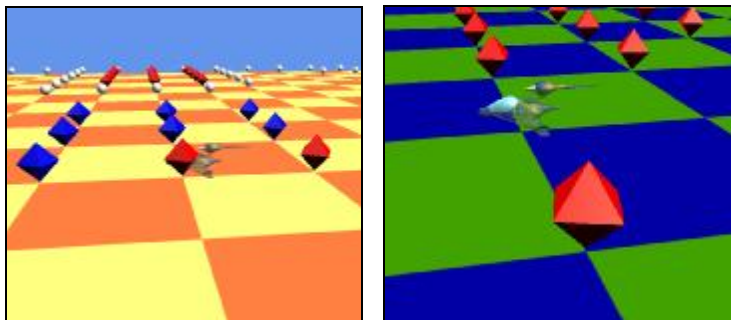


Figura 1: *Replay* gerado através das técnicas de reconhecimento de ações e previsão física.

Resumo

Nos ambientes virtuais 3D, a liberdade para o posicionamento de câmeras vêm permitindo que as aplicações utilizem técnicas de filmagem semelhantes às usadas nas apresentações cinematográficas. Porém, com a existência de interatividade nesses ambientes, não existe uma sequência predeterminada de acontecimentos que permita o posicionamento dinâmico ideal. Faz-se necessária então a utilização de técnicas de reconhecimento e previsão de ações para se posicionar a câmera de forma satisfatória. Este trabalho apresenta uma solução para o posicionamento automático de câmeras baseada em técnicas de reconhecimento e previsão de ações no ambiente virtual interativo com base no estado da simulação física.

Palavras-chave: câmeras virtuais, previsão de ações, cinematografia

Contato:

{dpires, epassos, esteban, anselmo}@ic.uff.br

1. Introdução

Com o advento da tecnologia para geração de gráficos 3D, o conceito de uma câmera virtual tem sido indispensável, estando intimamente ligado a outros como objeto e ambiente 3D. Um objeto 3D existe em um ambiente 3D e é preciso mostrá-lo ao mundo exterior através de uma câmera virtual.

Câmeras virtuais possuem algumas propriedades próprias que as diferem das câmeras do mundo real. A princípio elas são desprovidas de atributos físicos como massa e volume, e podem ser posicionadas e orientadas livremente pelo ambiente 3D, podendo inclusive atravessar outros objetos. Além disso,

câmeras virtuais possuem um limite de visualização bem definido, que dirá ao processador quais objetos, ou polígonos serão exibidos.

Ao se manipular uma câmera virtual sem se considerar propriedades físicas, obtém-se uma apresentação de uma cena 3D um tanto quanto irreal e estilizada. Com o objetivo de aproximar a simulação da realidade, desenvolvedores e *designers* passaram a modelar câmeras virtuais com atributos físicos, permitindo assim que esta possuísse uma aceleração quando se movimenta e levando em consideração colisões com outros objetos da cena.

Na cinematografia atual, a câmera é um dos principais elementos capazes de realçar o que se quer mostrar, além de introduzir subjetividade à cena dependendo do ângulo de visualização, do que está sendo focado e do tempo de tomada da cena [Martin 1985]. Com o passar do tempo, os desenvolvedores de ambientes (jogos) 3D se inspiraram nas produções cinematográficas para adicionar mais efeitos e emoção às suas produções [Hawkins 2005]. Termos como *zoom* e *pan* foram incorporados à interface de movimentação da câmera virtual. De fato, a programação de uma câmera virtual vem se tornando cada vez mais parecida com os comandos que um operador de câmera do mundo real receberia, com instruções de mais alto nível. A experiência da indústria do cinema explica e justifica o uso de cada tipo de tomada de câmera, constituindo assim uma linguagem própria, que também é aproveitada na produção de apresentações e jogos 3D.

No entanto, em aplicações 3D interativas, uma boa utilização de uma câmera virtual é bastante dificultada devido à imprevisibilidade dos acontecimentos. Em aplicações com sequências pré-definidas, já se sabe exatamente o que vai acontecer com os elementos da cena, e isso torna possível o posicionamento e

orientação ideais da câmera, de acordo com os resultados desejados. Porém, quando não se sabe previamente as transformações que podem acontecer com os objetos do ambiente 3D, é necessário se utilizar técnicas alternativas para a manipulação da câmera. É preciso, principalmente, reconhecer as ações e identificar os objetos relevantes para que a cena seja transmitida de forma apropriada, ainda se considerando a probabilidade da ocorrência de algum outro evento. Após o reconhecimento das ações no ambiente interativo, o sistema precisa ainda escolher qual é a mais relevante a cada instante, exibindo-a de forma conveniente com a emoção esperada. Para isso é preciso escolher uma boa posição de câmera, definir o que será focado e determinar o tempo da tomada.

Tendo escolhido uma ação para ser filmada, uma estratégia para posicionar a câmera, de baixo custo computacional, é se basear na simulação física, prevendo o que irá acontecer nos próximos momentos dentro do ambiente virtual. Para exemplificar o uso da técnica foi desenvolvido um jogo, onde o usuário tem a sua visão de jogo, e após sua jogada é exibido um *replay* de toda a ação, o que também representaria a visão de um telespectador.

Neste trabalho, primeiramente serão discutidos os trabalhos da comunidade científica relacionados ao problema. Em seguida serão apresentados alguns conceitos relevantes da área de cinematografia, seguidos pela estrutura de câmeras virtuais na plataforma XNA (www.xna.com). Finalmente será abordado o problema, as soluções propostas e resultados obtidos, explicando todas as particularidades do jogo produzido como exemplo.

2. Trabalhos Relacionados

Desde o surgimento da cinematografia, diversos autores vêm estudando novas técnicas para deixar as produções mais atraentes para o telespectador, e as escolas de cinematografia também se multiplicaram. Hoje existem boas publicações sobre as técnicas cinematográficas, como o trabalho de Martin [1985], além de *websites* especializados, como o Mnemocine [2008]. Mais recentemente, as mesmas técnicas vêm sendo amplamente usadas em jogos, para aumentar a imersão do jogador no ambiente 3D. Além disso, começou-se a pensar na possibilidade de se exibir uma partida de um jogo 3D exatamente como nas transmissões esportivas. O trabalho de Drucker [1994] analisou a viabilidade dessa aplicação das técnicas cinematográficas.

Existem diversos trabalhos que visam facilitar o problema de posicionar a câmera no ambiente virtual. Alguns deles utilizam uma DSL (*Domain-Specific Language* – Linguagem Específica de Domínio). São extremamente úteis por posicionar a câmera através de comandos de bem mais alto nível do que as linguagens de programação convencionais, porém têm a

desvantagem de deixar o posicionamento das câmeras predefinido, apenas ativam uma câmera ou outra dependendo dos eventos que ocorrem no ambiente. Os trabalhos de He et al. [1996] e Amerson & Kime [2000] se baseiam no uso de DSLs para posicionamento dinâmico de câmeras.

Outros trabalhos, como os de Drucker [1994] e Hermann & Celes [2005], posicionam a câmera de forma mais dinâmica, sem a intervenção do usuário. Têm um alto apelo pela automatização de tarefas, mas oferecem um resultado cinematográfico mais pobre se comparado ao uso de DSLs. O sistema de Hermann e Celes [2005] ainda divide o trabalho de posicionar a câmera em módulos roteirista, diretor e cinegrafista. Porém, no XNA, o módulo cinegrafista proposto por eles se confunde com o próprio mecanismo de geração de imagens no XNA. Com isso, o módulo diretor pode dar a sua saída de dados diretamente para o *shader* da aplicação. Este trabalho apresenta uma aplicação capaz de reconhecer as ações do ambiente e posicionar a câmera de tal forma que sejam considerados os eventos do ambiente virtual e a previsão dos próximos acontecimentos, através de cálculos físicos e o aproveitamento de informações passadas pela *engine* do jogo.

3. Cinematografia

Cinematografia é uma ciência que vem se desenvolvendo há muitas décadas, e criou com o tempo o que chamamos de linguagem cinematográfica. Essa linguagem define um conjunto de regras para transmitir ao espectador uma certa emoção, dependendo da organização das tomadas. Isso inclui posicionamento da câmera, iluminação, tempo dos cortes e seqüenciamento das cenas. Por exemplo, em uma filmagem de um diálogo, os posicionamentos e transições de câmera dizem que tipo de emoção se quer transmitir. No caso, a câmera um pouco inclinada (movimento conhecido como *roll*) pode enfatizar uma desorientação do personagem ou um mistério na narrativa [Hawkins 2005].

Devido à popularidade do cinema e ao desenvolvimento da linguagem cinematográfica, hoje as pessoas acham essa forma de comunicação muito natural. Quando aplicada de forma incorreta, porém, pode deixar o telespectador confuso. Às vezes a intenção do diretor em um certo momento pode ser mesmo instaurar a confusão, mas caso contrário, pode trazer uma interpretação errada da cena ou da produção como um todo. O objetivo da cinematografia é deixar as ações mais coerentes ao telespectador, tornando sua experiência mais agradável.

Posicionar uma câmera real no ponto em que o diretor gostaria pode envolver o uso de braços mecânicos, trilhos e outros equipamentos especializados [Martin 1985]. Dependendo do tempo e do orçamento, pode acontecer que o diretor use uma

posição diferente da ideal. Felizmente esse tipo de problema não existe em ambientes 3D, mas mesmo assim algumas questões devem ser consideradas. Dentro de um ambiente 3D é possível posicionar e mover a câmera livremente. Mas vale lembrar que em ambientes virtuais de tempo real um planejamento das cenas e dos acontecimentos pode ser impossível, e isso pode acarretar alguns problemas. A existência de um obstáculo entre a câmera e o alvo pode representar um problema difícil de detectar e de prevenir. Toda a computação consumida para enquadrar o objeto na tomada pode ter sido em vão se existirem obstáculos que bloqueiam a visão. Deixar os obstáculos invisíveis pode ser uma solução, mas pode se tornar uma armadilha, pois afeta a consistência espacial da cena; deve ser usado com cautela. Outro problema que pode surgir é em relação à movimentação da câmera. Se ela passa por dentro de objetos, torna o ambiente irreal, e conseqüentemente incoerente para o telespectador.

Nos *games* a cinematografia pode ser usada da mesma forma que no cinema para as apresentações e passagens pré-definidas. Mas para o ambiente de jogo é preciso fazer algumas adaptações devido à sua natureza mais dinâmica e para não prejudicar a jogabilidade. Hoje em dia os posicionamentos de câmera, definindo como será a visão do jogador, são uma importante questão de usabilidade a ser considerada. Dependendo do grau de interatividade, os conceitos cinematográficos podem ser suprimidos ou adaptados em um certo grau. Por exemplo, nas primeiras versões do *game* Resident Evil (www.residentevil.com), a câmera era fixa mas colocada em pontos que já traziam alguma expectativa de ação ou dificuldade. Em outros jogos, como God of War (www.godofwar.com), a experiência cinematográfica se dá através da movimentação da câmera seguindo uma trajetória pré-definida dependendo da localização do personagem principal no cenário. Mas em jogos como Super Mario 64 (www.nintendods.com/sm64ds/), maximizou-se a interatividade e a cinematografia foi um pouco sacrificada. Neste jogo é possível controlar a câmera livremente, além do personagem. A Figura 2 mostra os jogos citados.

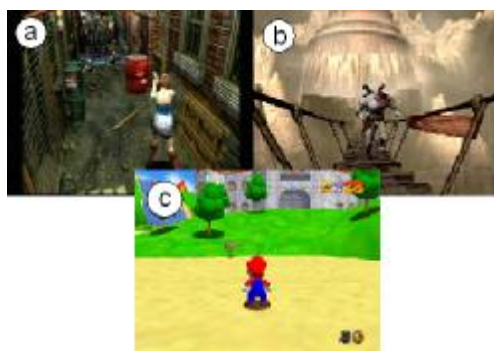


Figura 2: Cinematografia em jogos. (a) Resident Evil. (b) God of War. (c) Super Mario 64

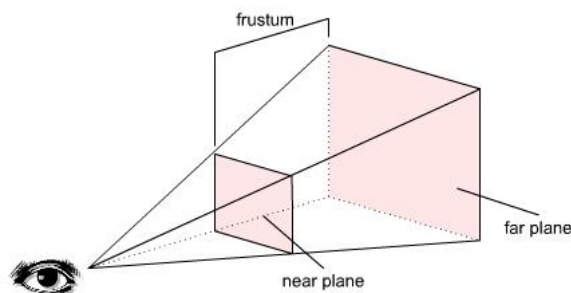
Outra aplicação com alto apelo cinematográfico é a exibição dos acontecimentos do jogo para uma platéia, em tempo-real. Neste caso, os conceitos da cinematografia podem ser aplicados em totalidade, visando mostrar a ação da forma mais atraente possível. Nota-se que as imagens geradas para o telespectador são bem diferentes das exibidas para o jogador; com essas imagens o jogo se torna impossível. As questões que aparecem são o reconhecimento de ações, a escolha da melhor ação e a exibição.

4. Câmeras Virtuais

A plataforma XNA, lançada pela Microsoft há aproximadamente 2 anos, se tornou muito popular no meio acadêmico para produção de jogos para Windows e X-box 360, e a cada dia ganha mais adeptos ao redor do mundo. Com o crescimento, surgiram diversos *websites* especializados na ferramenta, oferecendo desde tutoriais a classes e bibliotecas prontas. Existem hoje diversos trabalhos demonstrando diversas técnicas para implementação de câmeras em XNA. Entre eles destacam-se Riemers [2008] e Fegelein [2008], por explicar todos os fundamentos matemáticos em se tratando de manipulação de câmeras em ambientes 3D. Também surgiram muito bons livros sobre a ferramenta, tais como a publicação de Nitschke [2007].

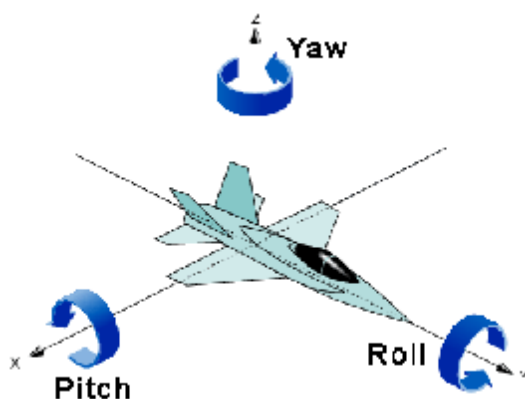
Como o público-alvo do XNA são estudantes e curiosos, sua própria documentação é repleta de exemplos e ainda conta com explicações sobre os fundamentos de processamento gráfico e técnicas de programação.

Utilizando-se a ferramenta XNA, para renderizar uma cena é preciso uma coleção de objetos 3D que compõem a cena e um *shader* que é o responsável por desenhá-la. Este *shader* toma uma configuração de câmera como parâmetro. Normalmente, para desenhar a cena corretamente, todo objeto 3D deve ser passado ao *shader* usando a mesma configuração de câmera. Uma câmera é composta por 2 elementos principais: as matrizes 4x4 conhecidas como *view* e *projection*. Ambas são processadas pelo *shader* na GPU. A matriz *view* estabelece a orientação da câmera; ela contém informações como sua posição, definida por um ponto no espaço, o alvo, que é o ponto para onde a câmera aponta, e o vetor que representa a orientação vertical da câmera. A matriz *projection* estabelece como a câmera projeta sua visão para a tela; e contém informações como a abertura focal, a relação entre altura e largura da janela de exibição, e os limites (a distância) do que a câmera pode enxergar, conhecidos como *nearplane* e *farplane*. Será efetivamente exibido na tela tudo o que estiver dentro do volume definido pela matriz *projection*, conforme a Figura 3.

Figura 3: *Frustum* de visão

É possível adicionar outros atributos à câmera transformando-a em uma classe. Para que a câmera tenha propriedades físicas, basta adicionar membros como velocidade, massa e aceleração como atributos da classe. Também podem-se adicionar propriedades e métodos que facilitem seu uso, assim a manipulação de camera será mais eficiente e versátil [Fegelein 2008].

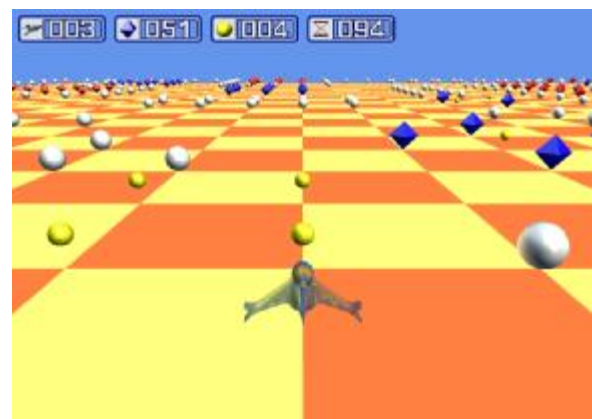
O XNA fornece um método muito útil para a manipulação de câmeras, `Vector3.Transform()`. Ele permite que um vetor seja facilmente rotacionado no espaço de acordo com o quatérnio ou matriz (de rotação) passada como parâmetro. Com este método a implementação de câmeras em primeira pessoa ou de câmeras que tentam manter uma distância fixa dos seus alvos é facilitada. O uso de quatérnios é recomendado quando é preciso armazenar rotações relativas ao objeto; são muito mais eficientes que armazenar rotações alinhadas com os eixos cartesianos [Riemers 2008]. Também são muito úteis para implementar movimentos relativos da câmera, tais como *yaw*, *pitch* e *roll*. Esses movimentos são originários da aviação mas que são muito usados em cinematografia. A Figura 4 ilustra o significado de cada um desses movimentos.

Figura 4: *Yaw*, *pitch* e *roll*

Para usar a câmera durante a programação do jogo, depois de todos os cálculos da lógica da classe câmera, é preciso informar ao *shader* as matrizes *view* e *projection*. O XNA fornece formas para agilizar o cálculo das matrizes *view* e *projection*, com os métodos `Matrix.CreateLookAt()` e `Matrix.CreatePerspectiveFieldOfView()`, respectivamente. Com isso, o programador só precisa se preocupar em passar os parâmetros necessários.

5. O Jogo, o Ambiente e Física

Para ilustrar o uso da técnica de posicionamento de câmeras aproveitando resultados da simulação física do ambiente, foi produzido um jogo em XNA chamado *Crystal Arena*. Este jogo consiste em controlar uma espaçonave que desliza pelo solo, coletando cristais azuis e evitando cristais vermelhos. O jogador tem uma vista do terreno relativamente ampla, permitindo visualizar os objetos que estão na frente da nave. A câmera acompanha o movimento desta, mantendo uma distância fixa. A nave tende a se mover continuamente para a frente, e pode ser rotacionada em 90 graus para a esquerda ou 90 graus para a direita, seguindo as divisões contidas no solo, que é quadriculado. Quando a nave colide com uma bola branca, ela passa a se movimentar na direção contrária. A Figura 5 mostra a tela de jogo.

Figura 5: O jogo *Crystal Arena*

Uma fase de jogo é representada por uma matriz 32x32 contendo todos os elementos (cristais azuis, cristais vermelhos, tesouros e bolas brancas) nas interseções das linhas do cenário. O conteúdo da matriz vai sendo atualizado à medida que o jogo avança, ela representa o que existe no momento. Por exemplo, se um tesouro é coletado, o valor naquela posição da matriz muda para indicar que agora contém um espaço vazio.

O jogador também pode fazer uso de uma jogada especial, que consiste em, enquanto coletando cristais azuis que vão se tornando vermelhos, circundar um grupo de cristais azuis com cristais vermelhos, conforme a Figura 6. Ao concluir esta manobra, todos os cristais neste conjunto se transformam em tesouros. Neste momento a matriz que representa a fase também é atualizada, localizando as posições da matriz que representam aquele grupo de cristais, e mudando seus valores para que representem agora tesouros.

Quando o jogador perde uma vida ou passa de fase, existe a opção de visualizar o *replay* de sua jogada. Neste ponto a câmera pára de seguir a nave e entra no modo cinematográfico, onde os resultados da previsão física no ambiente são aproveitados para posicionar a câmera. A nave volta à posição inicial e a simulação

recomeça, mas desta vez sem a intervenção do usuário para movimentar a nave.

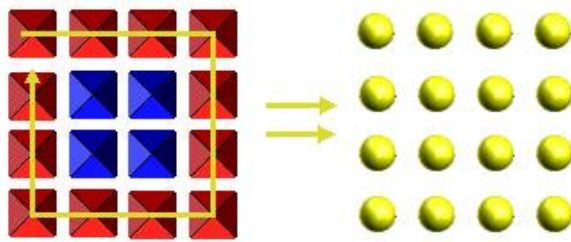


Figura 6: Transformação de cristais em tesouros

Devido à simplificação nos movimentos, a nave só pode colidir com objetos de frente ou de costas, sempre perpendicularmente, mas isso não inviabiliza os cálculos físicos e a técnica utilizada. Um jogador telespectador receberia as mesmas imagens que aquelas produzidas no *replay*. Porém, em um *replay*, os próximos estados do teclado, dos objetos ou do ambiente são conhecidos. Este fato não deve ser considerado nos algoritmos, para não prejudicar o caráter de previsão.

5.1 Reconhecendo Ações

Reconhecer as ações significa determinar corretamente o que está ocorrendo no ambiente virtual no momento. Em um jogo, ações mais objetivas ou pontuais fazem parte da própria *engine* do jogo ou fazem parte das propriedades ou máquina de estados de alguma classe. Neste caso, saber que uma ação ocorreu torna-se uma tarefa trivial. No jogo desenvolvido, os eventos que fazem parte da sua lógica são:

- Início da fase
- Nave se move para a frente ou para trás
- Curva à direita ou à esquerda
- Impacto com bola branca
- Impacto com cristal azul, que se torna vermelho
- Impacto com cristal vermelho, que destrói a nave
- Coleta de tesouro
- Conclusão de jogada especial
- Abrir a saída quando acabam os cristais azuis
- Alcançar a saída da fase
- Tempo esgotado

Às vezes uma modelagem de todas as ações pode ser muito difícil, devido à subjetividade de alguns eventos. Computacionalmente esses tipos de eventos são mais difíceis de reconhecer por causa da necessidade de uma interpretação dos acontecimentos ao longo do tempo. Para o domínio do jogo produzido, são ações interessantes:

- Nave começa a cercar cristais azuis para jogada especial
- Fazer manobras complicadas em espaços apertados
- Se aproximar da saída
- Chegar perto das extremidades da fase

- Nave se move em linha reta

Para reconhecer esses eventos pode ser desejável a utilização de um agente que fique monitorando as ações da nave e o estado da matriz da fase ao longo do tempo, e que avise ao sistema caso o padrão de movimento da nave e os objetos em volta formem um padrão que se assemelhe com os eventos desejados. Por exemplo, quando a nave fizer uma curva depois de tocar 2 cristais azuis nas extremidades de uma área preenchida por outros cristais azuis, então muito provavelmente trata-se de uma jogada especial. Se o jogador falha ao coletar os cristais necessários para a jogada especial, apenas o evento subjetivo “tentativa de jogada especial” será registrado; não existirá, porém, a ocorrência do evento de jogo “conclusão de jogada especial”.

5.2 Previsão da Simulação Física

A previsão é usada para descobrir o que vai acontecer nos próximos instantes. Como no ambiente virtual um certo objeto pode estar sujeito a várias coisas, a previsão ajuda a determinar a probabilidade com que um dado acontecimento pode ocorrer ou como um dado acontecimento pode se concluir.

A previsão dos acontecimentos do ambiente virtual ajuda no reconhecimento de ações subjetivas, e é especialmente útil para posicionamento automático de câmeras. Por exemplo, em um ambiente virtual onde uma motocicleta salta por uma rampa, é possível, através de cálculos de física, descobrir onde e como ela vai cair, tomando assim as devidas providências para mostrar o fato da melhor forma possível. Logicamente, por meio da previsão, seria possível executar outros processos tais como fazer o piloto sair da motocicleta caso o salto se mostre desastroso. Previsão física também é útil para fins de melhoria da inteligência artificial. Por exemplo, na lógica de um lutador inteligente, que calcula seu próximo ataque de acordo com o tempo restante para que seu inimigo aterrisse depois de um salto.

Para prever o que pode acontecer no ambiente virtual podem-se usar vários artifícios, tais como cálculos de física ou outros cálculos, às vezes agilizados pelas próprias estruturas de dados usadas no sistema. Sempre é preciso considerar o que o objeto tende a fazer (como gravidade e outras velocidades) e os objetos que estão em volta, além de considerar como o usuário ou outros componentes podem intervir na cena. No jogo produzido, como a nave tende sempre a ir para a frente, a probabilidade de colisão à frente se torna maior conforme o tempo passa, já que sempre existirá algo à frente para colidir. Com isso o sistema estará cada vez mais preparado para filmar a colisão, que pode eventualmente não acontecer, devido à intervenção do usuário. Também, para a jogada especial, é muito provável que o jogador consiga concluí-la se já coletou mais de $\frac{3}{4}$ dos cristais

necessários para tal; a câmera estará preparada para filmar a jogada bem-sucedida.

5.3 Posicionando a Câmera

A partir das ações que foram reconhecidas e pelas inferências obtidas pela manipulação física, é possível posicionar a câmera e fazê-la se comportar segundo os ensinamentos de cinematografia. Todos os dados obtidos nas etapas anteriores serão processados por um agente diretor, que possui seu próprio estilo de filmagem. Como resultado o agente diretor altera diretamente as propriedades da classe câmera, que passa ao *shader* da aplicação a visualização a ser renderizada. A Figura 7 mostra um esquema de funcionamento.

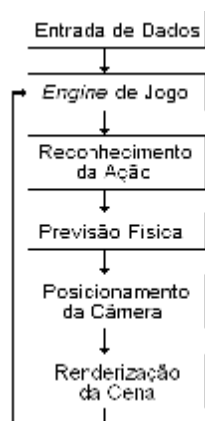


Figura 7: Processo de posicionamento de câmera

Como mostrado na Figura 7, o posicionamento de câmera é resultado de todas as etapas anteriores. A *engine* de jogo informa os eventos relativos à lógica do jogo ou as propriedades e máquinas de estados dos objetos. Com base nessas informações, é possível que eventos subjetivos sejam reconhecidos, e que próximos eventos sejam previstos. Então o agente diretor pode posicionar a câmera de acordo com a linguagem cinematográfica utilizada, pois ele já tem acesso a todas as informações que ele necessita. Também existe um fator de aleatoriedade na escolha da posição da câmera, para que o processo não se torne totalmente determinístico. Dadas de uma a cinco posições aceitáveis, uma é escolhida. Com isso o *replay* pode ser visto diversas vezes, cada uma com uma configuração de câmera diferente. Após isso, a cena é renderizada, e a partir daí o processo se repete até que aconteça outro evento ou o agente diretor entenda que agora existe uma melhor posição de câmera. Finalmente, como o usuário tem o controle de parte do ambiente virtual (o personagem que ele controla), outros eventos podem ocorrer devido à sua intervenção.

Segundo Hawkins [2005], as principais questões que devem ser consideradas ao posicionar a câmera no ambiente virtual são:

- **Enquadramento:** Com o uso da plataforma XNA, é possível definir o alvo da câmera como sendo o centro da nave, por exemplo, para fazer a câmera segui-la automaticamente, independente da posição da câmera. Da mesma forma, pode-se programar a câmera para focar em qualquer ponto ou objeto do cenário. No entanto, enquadrar o objeto de forma que ele fique muito grande ou muito pequeno na cena tem seu impacto para o telespectador. Métodos tradicionais consistem em calcular a razão entre a área da tela e a área da projeção da *bounding box* ou *bounding sphere* do objeto focado. No ambiente virtual produzido, como os tamanhos de todos os objetos já são conhecidos, para simplificar os cálculos basta afastar ou aproximar a câmera em fatores já conhecidos.
- **Oclusão:** Em ambientes virtuais, pode ser muito difícil detectar quando um objeto está bloqueando a visão. Métodos tradicionais compreendem lançar raios até o alvo da câmera e avaliar se os raios interceptam algum outro objeto que está entre a câmera e o alvo. No jogo produzido, porém, sabe-se que não existe nenhum objeto acima de certa altura, e como o espaço foi todo discretizado, é possível, consultando-se a matriz da fase, saber se em uma posição (x, z) do cenário existe algum objeto. Como solução pode-se mover a câmera para um outro local, ou deixar todos os objetos que atrapalham a visão invisíveis. Essa segunda opção pode ser perigosa pois leva à inconsistência espacial.
- **Consistência espacial:** Existem algumas regras cinematográficas para manter a consistência espacial, isto é, não deixar o telespectador ficar confuso quanto ao posicionamento dos componentes da cena em função de um mal posicionamento de câmera. A Figura 8 mostra um exemplo.

A linha tracejada na Figura 8 é conhecida como linha de ação. Para que a cena fique consistente, é desejável que todas as tomadas de câmera fiquem do mesmo lado do espaço definido pela linha de ação [Hawkins 2005]. A violação desta regra (Figura 8.b) dá a impressão que os personagens trocam de lugar durante a conversa, e pode deixar o telespectador confuso. Para o jogo produzido, para manter a consistência espacial, a câmera tende a ser posicionada de tal forma que sua componente Z no espaço fique sempre entre a componente Z da nave e zero, que é a componente Z do ponto de início da fase. Isto é, a posição Z da câmera é sempre maior que a da nave. Também, não existem cortes ou movimentos de câmera enquanto a nave está fazendo uma curva, colidindo com uma bola branca, ou está muito próximo da saída da fase (o único momento que a componente Z da câmera pode ser maior que a da nave). Se a

câmera está se movendo nesse momento, ela pára imediatamente.

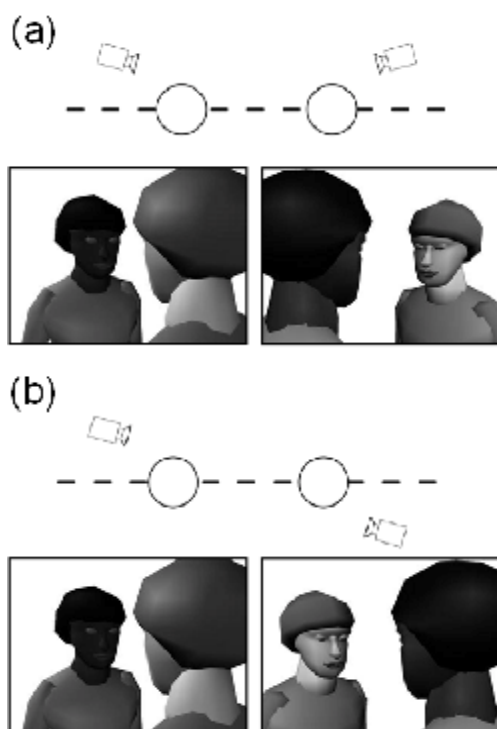


Figura 8: (a) Cena com consistência espacial
(b) Cena inconsistente.

Usando conceitos da linguagem cinematográfica, segundo Martin [1985], também existem outras restrições para dar mais dramacidade à ação, aplicadas ao contexto do jogo:

- Se a nave está muito próxima de uma região onde é possível executar a jogada especial, a câmera tende a ir para um ponto em que pode enquadrar todos os cristais do grupo e mostrar sua transformação.
- Se o usuário erra a jogada especial, a situação pode se tornar perigosa, e a câmera deve se aproximar.
- Se a nave está numa situação segura (evento reconhecido quando existem poucos cristais ou bolas brancas próximas da nave), a câmera fica mais longe e mostra todo o cenário.
- Quando existem muitos obstáculos em espaços apertados, existe muito perigo de colisão. Neste caso a câmera fica sempre perto da nave, e o tempo da tomada é bastante curto.
- Quando a nave está rodeada de muitos cristais vermelhos, existe muito perigo. Neste caso a câmera fica sempre perto da nave, e o tempo da tomada é bastante curto. Pode-se executar então o movimento de sutilmente inclinar a câmera (movimento de *roll*) para enfatizar o perigo [Hawkins 2005].

Exceto o fato de a saída se abrir, nenhum outro evento acontece muito longe do objetivo principal da

ação, que é a nave em movimento. Portanto, em quase todas as tomadas a nave passará na frente da câmera em algum momento, mas isso não significa que o alvo será sempre fixo na nave. Principalmente nas situações de perigo, a câmera pode ter como alvo uma bola branca em que existe alta probabilidade de colisão. Neste momento, provavelmente a nave irá passar, colidindo ou desviando a tempo.

Outro fator que deve ser considerado é o tempo das tomadas. Hoje em dia, o tempo em que uma tomada dura, para cenas de ação e/ou reflexo é de 2 segundos [Hawkins 2005]. Isso ajuda a criar no telespectador a mesma sensação de tensão vivida pelo protagonista ou jogador. Quanto mais confortável for a situação da nave e quanto mais afastada estiver a câmera do solo, mais longa será a tomada, indicando tranquilidade.

5.4 Resultados

No jogo produzido, o *replay* é construído por meio de uma estrutura de dados que armazena todos os comandos do jogador ao longo do tempo. Existe uma variável responsável por contar o tempo de jogo restante ao jogador, que é atualizada a cada *frame*. Uma entrada do jogador é armazenada juntamente com o número de *game-loops* passados desde o início da fase. Esta abordagem é mais robusta do que armazenar o tempo passado em milissegundos, pois o *replay* pode se tornar defasado em função da execução de outros processos no sistema, como a própria técnica para geração do *replay*.

Para não comprometer a intenção da previsão, apenas a posição atual da lista de comandos deve ser observada, simulando uma situação de jogo em tempo-real. Elementos que representam as próximas entradas do jogador não serão considerados. Para o reconhecimento de ações, não considera-se os comandos anteriores, mas sim os eventos anteriores, tais como virar, colidir etc. Para isso, durante o *replay*, uma outra estrutura de dados é utilizada. Ela armazena os últimos 20 eventos da fase, juntamente com a posição da nave naquele momento. A partir daí o programa faz suas previsões e posiciona a câmera. Com a adição dessas variáveis, o consumo de memória aumenta em média 5 Kb, o que representa uma porcentagem muito pequena do consumo do jogo inteiro.

A diferença de desempenho entre o jogo e o *replay* está relacionada com os processos que são executados a mais para posicionar a câmera corretamente. O único processo que não está presente na hora do *replay* é a exibição dos painéis informativos (HUD – *Head Up Display*), o que representa aproximadamente 10% de todo o trabalho de renderização. A entrada de dados é substituída pela leitura da lista de comandos, que é mais custosa para o sistema. A Figura 7 mostra um ciclo completo do *game-loop*, que é executado no momento do *replay*. Durante a interação com o jogador, as etapas de reconhecimento de ação, previsão

física e posicionamento da câmera não estão presentes. Todas essas etapas podem representar um acréscimo de até 50% no processamento da aplicação como um todo. A diferença foi medida através do *frame rate* do jogo em execução.

Outro fato observado foi que, durante o estado em que o jogador pode controlar a nave, existe um processamento muito maior quando a nave colide com um cristal azul, pois é preciso testar se a jogada especial foi concluída. Em caso positivo, ainda existe o trabalho de transformar os itens da matriz da fase. Porém, durante o *replay*, a colisão com um cristal azul não aumenta tanto o trabalho de processamento, pois já existem muitos outros processos acontecendo nesse momento. O gargalo de processamento se transforma da renderização durante o jogo, para a lógica de posicionar a câmera dinamicamente durante o *replay*.

6. Conclusões

Este trabalho mostrou uma solução para posicionamento de câmeras, a partir de informações passadas pela *engine* de jogo, usando as técnicas de reconhecimento de ações e previsão física dentro do ambiente virtual, além de conceitos de cinematografia. Foi mostrado como funciona a plataforma XNA, com a qual o jogo para exemplo foi construído, e algumas técnicas cinematográficas que foram seguidas.

Todas as adaptações no ambiente virtual e na dinâmica do jogo foram mostradas de forma que a técnica pudesse ser aplicada corretamente e de forma otimizada. Outro fator fundamental para o sucesso da técnica foi a adição de um fator de aleatoriedade, que evita o determinismo e deixa o resultado final mais interessante para o telespectador.

De acordo com as experiências tomadas e resultados alcançados, conclui-se que a utilização da técnica é viável, tanto para a geração de *replays*, quanto para a transmissão em tempo-real para um grupo de telespectadores, como explicado em [Drucker 2005].

Referências Bibliográficas

- AMERSON, D. E KIME, S., 2000. *Real-time Cinematic Camera Control for Interactive Narratives*.
- DRUCKER, S., 1994. *Intelligent Camera Control for Graphical Environments*. Tese de Doutorado, Massachusetts Institute of Technology.
- ERLEBEN, K., 2002. *Module Based Design for Rigid Body Simulators*.
- FEGELEIN, 2008. *Microsoft XNA Framework; Creating a Freelook Camera* [online]. Disponível em: <http://www.fegelein.com/?p=18> [Acessado em 8 de agosto de 2008].
- HAWKINS, B., 2005. *Real-Time Cinematography for Games*. Editora Charles River Media, 2005.
- HE, L. ET AL., 1996. *The Virtual Cinematographer: A Paradigm for Automatic Real-Time Camera Control and Directing*.
- HERMANN, R. E CELES, W., 2005. *Posicionamento Automático de Câmeras em Ambientes Virtuais Dinâmicos*.
- MARCHAND, É. E COURTY, N., 2002. *Controlling a Camera in a Virtual Environment*.
- MARTIN, M., 1985. *A Linguagem Cinematográfica*. Editora Brasiliense, 1985.
- MNEMOCINE, 2008. *Linguagem e técnica cinematográfica* [online]. Disponível em: <http://www.mnemocine.com.br> [Acessado em 8 de agosto de 2008].
- NITSCHKE, B., 2007. *Professional XNA Game Programming for Xbox 360 and Windows*. Editora Wrox, 2007.
- PINHANEZ, C., 1999. *Representation and Recognition of Action in Interactive Spaces*. Tese de Doutorado, Massachusetts Institute of Technology.
- RIEMERS, 2008. *Quaternion Camera* [online]. Disponível em: <http://www.riemers.net/eng/Tutorials/XNA/Csharp/Serie s2/Quaternions.php> [Acessado em 8 de agosto de 2008].
- SEUGLING, A. E RÖLIN, M., 2006. *Evaluation of Physics Engines and Implementation of a Physics Module in a 3D Authoring Tool*. Tese de Mestrado, Umea University.