# 4
# Técnicas para Ambientes Virtuais Distribuídos

Ambientes Virtuais Distribuídos trazem o problema do gerenciamento do estado compartilhado. Como demora um certo tempo para uma mensagem chegar ao seu destino, quando a mensagem chega, ela tratará de um estado antigo do computador que enviou a mensagem.

Existem três técnicas básicas para gerenciar o estado compartilhado:

- Repositórios de informação centralizados: Essa técnica permite que todos os participantes da simulação vejam o mesmo estado, ficando em completo sincronismo. Porém há um custo elevado nessa maneira de funcionamento e o nível de interatividade conseguido é baixo. Nessa técnica, as atualizações dos jogadores são feitas em turnos, e a cada *frame*, a base de dados é lida.
- Notificação frequente de atualização de estado: Nessa técnica, o estado de alguma informação é enviado constantemente para algum destino e, com isso, o destino possui o estado um pouco atrasado da informação enviada. O *lag* e a taxa de reenvio de estado definem o quão atrasado é o estado de alguma informação.
- *Dead-reckoning*: É a maneira que permite o menor custo em troca de mensagens em comparação com as outras duas formas mencionadas, e é explicado mais abaixo.

Para mais informação sobre gerência de estado compartilhado veja [24].

Nesse capítulo são explicadas algumas técnicas úteis para uso em ambientes virtuais distribuídos.

PUC-Rio - Certificação Digital Nº 0410842/CA

## 4.1 Dead-Reckoning

*Dead-Reckoning* é uma técnica onde tenta-se estimar a posição de um objeto baseado em alguma informação no passado sobre esse objeto. Por exemplo:

Considere-se que, no passado, um objeto estava na posição (0,0) e se movia a 10 m/s na direção (1,0). Então, após 2 segundos, pode-se estimar que a posição do objeto é:

```
pos.x = pos.x + (vel * dir.x) * deltat;
pos.y = pos.y + (vel * dir.y) * deltat;
```

onde pos é a posição do objeto, vel é a velocidade, dir é a direção e deltat é a variação em segundos da última posição calculada.

*Dead-Reckoning* tem suas origens em simulações militares, onde se tenta prever as trajetórias de aviões e veículos. Polinômios, como os apresentados acima, são as técnicas mais comuns para estimar as posições futuras de um objeto.

*Dead-Reckoning* é uma excelente forma de reduzir o número de mensagens enviadas entre clientes. Enquanto um cliente souber que os outros clientes são capazes de calcular a posição de um objeto sob seu controle, não há necessidade de enviar uma nova mensagem sobre o objeto. Só quando o cliente faz alguma alteração no objeto que torna impossível os outros clientes estimarem corretamente a sua posição é que há a necessidade do envio de uma nova mensagem sobre o objeto.

Ao receber uma mensagem de atualização, o objeto remoto recupera a consistência de estado convergindo (ou simplesmente "pulando") para o estado físico transmitido. Há problemas de descontinuidade (principalmente visual) que podem ser razoavelmente solucionadas com ajuste de curvas [24] e protocolos baseados na história das posições [23]. Estas soluções, entretanto, não conseguem evitar colisões indesejáveis que ocorrem quando a trajetória prevista difere da real.

Uma maneira de se reduzir ainda mais o número de mensagens trocadas e evitar colisões indesejáveis, é através de um *Dead-Reckoning* baseado em objetivos [29]. Para esse *Dead-Reckoning* funcionar é necessário que os clientes e servidores possuam inteligência que os tornem capazes de decidir sobre a movimentação de um objeto baseado em seu objetivo. Esses objetivos podem ser qualquer coisa, atacar um determinado alvo, passar por uma porta, chegar a determinada posição, etc.

PUC-Rio - Certificação Digital Nº 0410842/CA

Nessa forma de *Dead-Reckoning*, os objetivos são enviados pela rede. Quando um cliente recebe um novo objetivo de um determinado objeto, ele passa a calcular a movimentação de maneira que o objeto alcance o objetivo desejado. Isso pode causar variações significativas na movimentação de um mesmo objeto entre clientes e pode não ser uma boa solução em todos os jogos onde *Dead-Reckoning* é aplicável.

## 4.2
## Sincronização baseada em tempo de comando

Em alguns jogos de estratégia, por exemplo, no momento em que um jogador clica no mapa em um destino para um objeto, esse objeto começa a se mover para a posição de destino. Para que os outros jogadores vejam o mesmo movimento, uma mensagem é enviada para o servidor pelo cliente que requisitou o movimento e, depois, o servidor reenvia a mensagem de movimento para os outros clientes.

Se os clientes e o servidor sempre moverem os objetos na mesma velocidade, aqueles clientes com maior latência com o servidor podem ficar bastante dessincronizados com o resto dos clientes.

É possível reduzir a dessincronização entre clientes fazendo alterações na velocidade dos objetos, baseadas no momento em que as mensagens são recebidas. Um exemplo disso é mostrado a seguir:

Um cliente clica no destino (30,0) fazendo um objeto na posição (0,0) começar a se mover numa velocidade de 10 m/s(fig. 4.1). Em princípio, o objeto deveria chegar em sua posição final no tempo 3 segundos. A mensagem enviada para o servidor diz "O objeto está se movendo para a posição (30,0)". O servidor sabe a velocidade do objeto. Suponha-se que a mensagem demora 1 segundo para chegar ao servidor. Quando o servidor receber essa mensagem, ele começa a mover o objeto em sua própria simulação na velocidade de 10 m/s(fig. 4.2), afinal é no servidor que está o estado mais recente do jogo. Sendo assim, no servidor, o objeto chegará na posição (30,0) no tempo 4 segundos. O servidor então envia para todos os clientes, inclusive para aquele que fez o movimento, a mensagem "O objeto chegará na posição (30,0) no tempo 4 segundos". Todos os clientes, ao receberem a mensagem, calculam a velocidade do objeto necessária para que ele chegue ao seu destino no tempo especificado. Suponha-se então, que todos os clientes recebem essa mensagem no tempo 2 segundos. Nos clientes onde o objeto está na posição (0,0), a velocidade do objeto é calculada em 15 m/s, o que fará o objeto alcançar a posição (30,0) em 2 segundos. Já

PUC-Rio - Certificação Digital Nº 0410842/CA

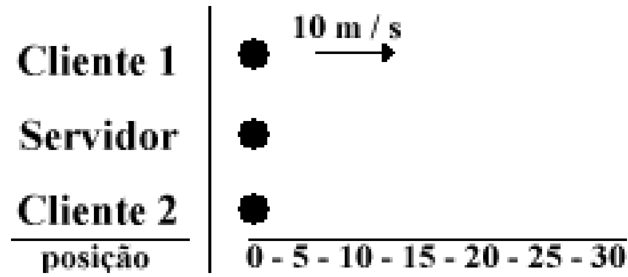


Figura 4.1: Tempo 0 segundos. O Cliente 1 começa a se mover na velocidade 10 m/s

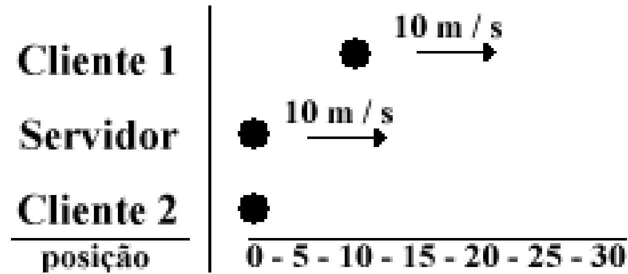


Figura 4.2: Tempo 1 segundo. O Servidor começa a se mover. O Cliente 1 já se moveu 10 metros.

PUC-Rio - Certificação Digital Nº 0410842/CA

no cliente que gerou o movimento, o objeto já tinha começado a se mover numa velocidade de 10 m/s e no tempo 2 segundos o objeto está na posição (20,0). Neste caso, a velocidade é reduzida para 5 m/s, o que fará o objeto também alcançar a posição (30,0) em 2 segundos(fig. 4.3).

As velocidades nos clientes são calculadas tendo influência da latência e isso contribui para a redução da dessincronização entre eles. No entanto, um problema com essa técnica é que todos os pontos devem compartilhar o mesmo tempo global. Isso não é algo trivial de se conseguir, mas é possível conseguir com que os clientes adquiram um tempo estimado do servidor calculando uma estimativa de RTT(*Round Trip Time*) entre o servidor e o cliente e fazendo o servidor enviar o seu tempo. Mesmo havendo uma variação de tempos, se essa for pequena, a diferença visual do momento de chegada dos objetos entre os clientes será pequena e possivelmente imperceptível.

## 4.3 Eliminando informações desnecessárias

Durante a renderização de ambientes virtuais 3D, um estágio geralmente encontrado é o estágio de *culling*, onde são eliminadas todas aquelas partes do mundo que não estão visíveis e, portanto, não contribuem para a

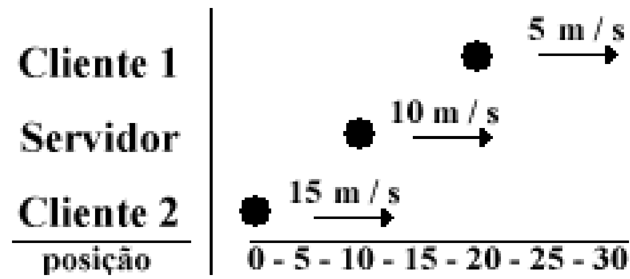


Figura 4.3: Tempo 2 segundos. O Cliente 1 reduz a velocidade para 5 m/s. O Cliente 2 coloca a velocidade 15 m/s

imagem final. Isso é feito com o objetivo de reduzir o número de triângulos enviados para a placa gráfica.

O mesmo pode ser feito para distribuição. Em geral, não há necessidade de enviar uma informação para um cliente que o jogador não possa perceber. É claro que há exceções. Mas enviar uma mensagem da movimentação de um personagem do outro lado do mundo, onde é impossível para o jogador percebê-lo ou interagir com ele, é uma ação desnecessária.

O que se deseja, geralmente, é enviar para um cliente apenas informações que o jogador pode ver. Por exemplo: informações de outros jogadores, monstros e itens visíveis. Diversas estruturas hierárquicas utilizadas para *culling* em computação gráfica podem ser utilizadas também para fazer a eliminação das informações desnecessárias em distribuição. Para saber mais sobre elas recomenda-se a leitura de [18].

Uma maneira de se descobrir rapidamente os objetos próximos de um jogador é usando um *grid*. Um *grid* divide o mundo em vários retângulos de tamanho igual. Cada retângulo armazena informações sobre os objetos que se encontram em seu espaço.

Para pegar os objetos próximos a um jogador, basta pegar os objetos no retângulo do jogador e também os objetos nos retângulos considerados próximos. Essa região que define quais objetos são próximos e possivelmente perceptíveis pelo jogador é chamada de área de interesse.

Pode-se encontrar o retângulo em que se encontra um jogador através das seguintes expressões:

```
grid.x = pos.x / larguraRetangulo;
grid.y = pos.y / alturaRetangulo;
```

onde grid.x e grid.y armazenam a posição no *grid* do jogador, pos.x e pos.y armazena a posição do jogador e larguraRetangulo e alturaRetangulo armazenam as dimensões dos retângulos do *grid*.

PUC-Rio - Certificação Digital Nº 0410842/CA

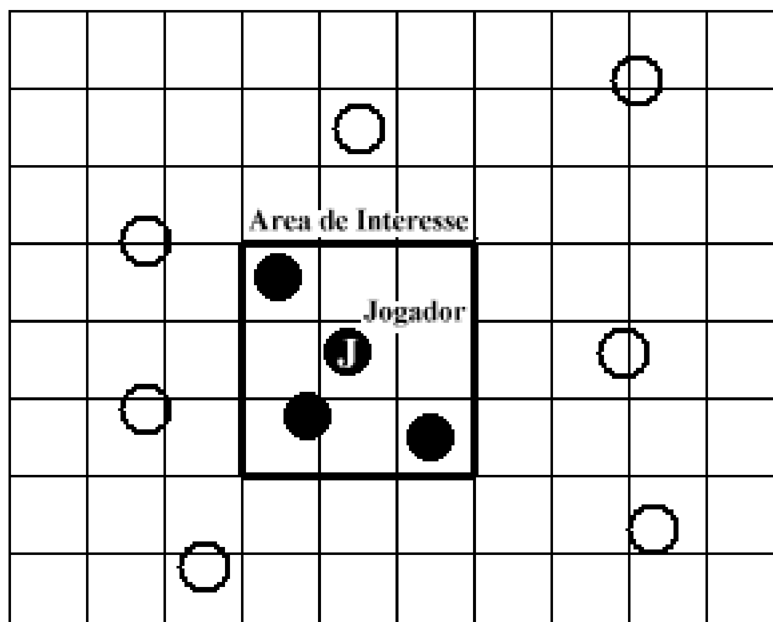


Figura 4.4: Os círculos preenchidos são os personagens que estão na área de interesse do jogador. O círculos não preenchidos são os personagens fora da área de interesse do jogador.O jogador recebe informações de todos os personagens em sua área de interesse.

PUC-Rio - Certificação Digital Nº 0410842/CA

A figura 4.4 mostra o *culling* usando um *grid*. Todos os personagens próximos ao jogador (os círculos preenchidos) são enviados para o cliente. Já os personagens que estão longe do jogador(círculos não preenchidos) não têm suas informações enviadas para o jogador.

A vantagem do uso do *grid* está na sua simplicidade e eficiência. Tanto achar os objetos próximos a um jogador, quanto achar a nova posição no *grid* de um jogador que se movimentou são operações de baixo custo computacional. Porém deve-se tomar cuidado com o uso de objetos grandes, pois eles podem ocupar grande parte de regiões vizinhas e não serem considerados parte dessas regiões. Jogadores movimentando-se nessas regiões podem ver o objeto "surgir do nada", ao moverem-se para uma posição no *grid* próxima à posição do objeto.

Uma outra técnica que pode ser interessante para MMORPGs, onde a movimentação é geralmente 2D, é a *Quadtree*. Nessa técnica, o mundo é dividido hierarquicamente de 4 em 4 regiões. Primeiro o mundo é divido em 4 regiões, depois cada um dessas regiões é dividida em 4 regiões, e esse processo se repete até que algum critério seja atingido.

Nessa técnica, achar os objetos próximos a um jogador tende a ser mais custoso, pois há a necessidade de caminhar hierarquicamente pela *Quadtree* até achar os nós próximos do nó do jogador. No entanto, a *Quadtree* pode ser uma solução adequada quando se possui uma grande variação no tamanho

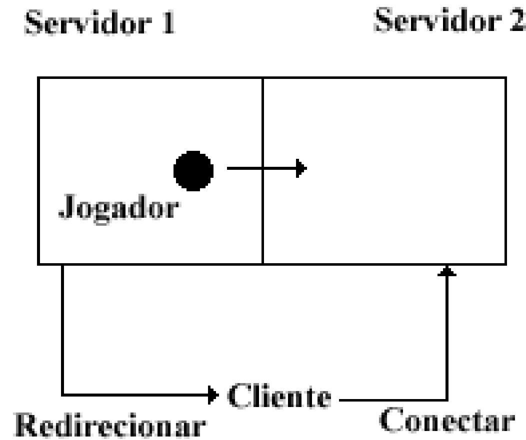


Figura 4.5: Quando um jogador passa de um retângulo para outro, ele se conecta à um novo servidor

dos objetos. Um objeto grande, por exemplo, poderia ocupar um nó superior enquanto um objeto menor ocuparia um nó abaixo na hierarquia.

A movimentação de objetos também tende a ser mais custosa, devido à complexidade maior da estrutura.

PUC-Rio - Certificação Digital Nº 0410842/CA

## 4.4 Balanceamento

Para conseguir que milhares de jogadores estejam conectados simultaneamente a um mesmo jogo é necessário que o processamento do jogo seja quebrado entre diversos servidores, onde cada servidor é responsável por processar uma parte do jogo.

Uma maneira de se fazer isso é quebrar o mundo todo em retângulos, deixando cada servidor responsável por um retângulo e pelo processamento de tudo o que ocorre dentro desse retângulo. Quando um cliente então vai se conectar ao jogo, ele se conecta ao servidor do retângulo em que se encontra e fica se comunicando com esse servidor enquanto estiver nesse retângulo. No momento em que o jogador se mudar para um retângulo de outro servidor, ele se conecta ao novo servidor e esse passa a se tornar o responsável pelo processamento do jogador. Esse redirecionamento do jogador é mostrado na figura 4.5.

Existe um problema, no entanto, quando esses retângulos dividindo o mundo são estáticos. Muitos jogadores podem se aglomerar em um único retângulo, deixando o servidor responsável sobrecarregado, enquanto outros

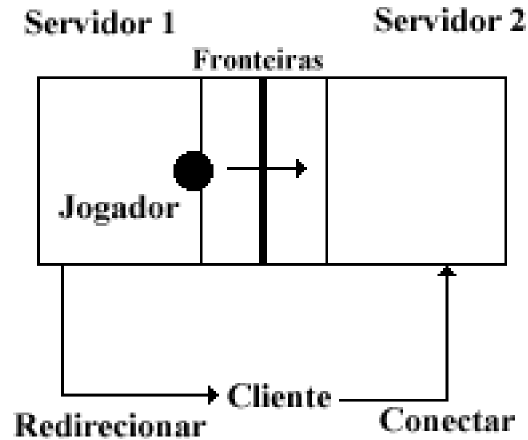


Figura 4.6: Retângulos e suas fronteiras. Quando um jogador cruza uma fronteira ele já se conecta ao novo servidor.

PUC-Rio - Certificação Digital Nº 0410842/CA

servidores ficam com pouco processamento. Uma forma de tratar isso é permitindo que as fronteiras dos retângulos possam se mover de maneira a equilibrar o processamento entre os servidores.

Como estabelecer uma conexão demora um certo tempo (o estabelecimento de uma conexão TCP envolve a troca de 3 mensagens), pode-se tentar estabelecer a conexão com o novo servidor antes do jogador de fato entrar em seu retângulo. Para fazer isso, basta expandir as fronteiras dos retângulos, como mostrado na figura 4.6.

A implementação de um balanceamento usando essa conexão antecipada deve permitir que os clientes fiquem conectados a mais de um servidor simultaneamente (quando por exemplo o cliente está em uma fronteira, ele fica conectado a pelo menos 2 servidores).

Uma alternativa ao uso de retângulos é o uso de hexágonos para o balanceamento. Nesse caso, o mundo todo é dividido em hexágonos e cada servidor é responsável pelo processamento de tudo o que ocorre dentro de um desses hexágonos. A vantagem do uso de hexágonos sobre retângulos é que um cliente ao ser aproximar da fronteira, fica próximo de até 2 servidores no máximo (fig. 4.7), enquanto ao usar retângulos, um cliente pode ficar próximo de até 3 servidores (fig. 4.8). Ou seja, ao usar retângulos, um cliente pode ficar conectado a até 4 servidores ao mesmo tempo (quando está se movendo por uma fronteira), enquanto ao usar hexágonos um cliente só pode ficar conectado a até 3 servidores. A desvantagem do uso de hexágonos é que existe um custo maior na verificação se um cliente ultrapassou uma das fronteiras.

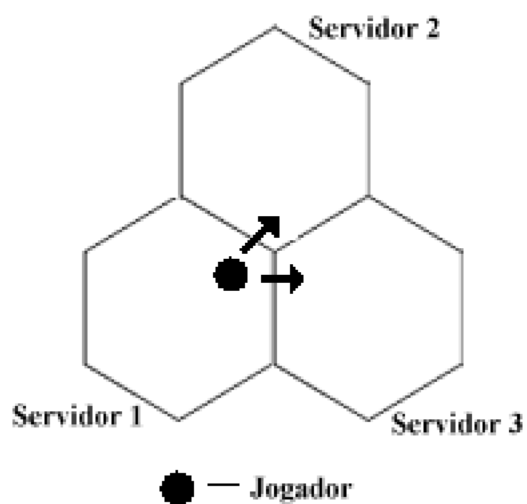


Figura 4.7: Jogador se aproximando de dois servidores em mundo dividido em hexágonos.

PUC-Rio - Certificação Digital Nº 0410842/CA

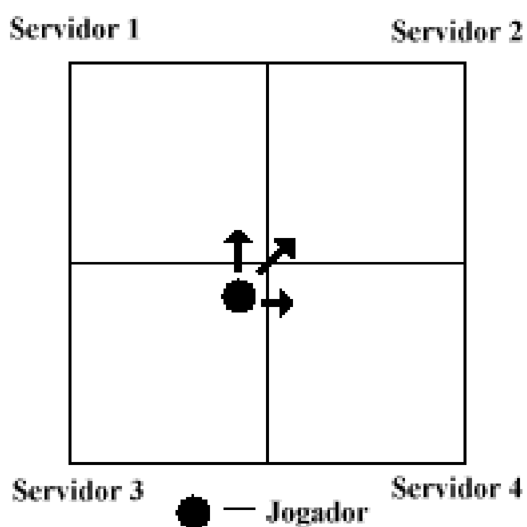


Figura 4.8: Jogador se aproximando de três servidores em mundo dividido em retângulos.

Uma outra alternativa para o balanceamento é fazer o redirecionamento de clientes baseado no tempo que demora uma mensagem para ir e voltar entre o cliente e o servidor, que chamaremos aqui de RTT (*Round Trip Time*). Apenas esse critério não é suficiente, pois um servidor com o RTT bom para um grande número de clientes pode acabar sobrecarregado. Deve ser levado em consideração também o número de clientes e o uso da CPU. Mas a principal limitação dessa forma de balanceamento é que um mesmo servidor pode acabar tratando de clientes próximos a todos os outros clientes e objetos do mundo e isso pode necessitar de mais memória que um servidor pode suportar. Ao usar esse tipo de balanceamento, deve-se aceitar que um servidor pode acabar tendo que armazenar informações sobre todos os jogadores do mundo, mesmo que seja responsável apenas por uma fração desses jogadores. Os outros balanceamentos por retângulos e hexágonos permitem a construção de mundos maiores, pois, em geral, um servidor armazenará informação sobre uma fração de jogadores e objetos do mundo.

PUC-Rio - Certificação Digital Nº 0410842/CA

## 4.5 Nível de detalhe

Essa é uma outra técnica que se inspira no nível de detalhe da computação gráfica [24]. A idéia é que não há necessidade de se receber muita informação de objetos que não influenciam em muito a experiência do jogador. Em geral, os objetos que mais influenciam a experiência do jogador são aqueles que estão próximos. Logo a taxa de atualização desses objetos deve ser maior. Objetos longe do jogador podem adquirir uma taxa de atualização menor.

Essa técnica pode ser usada em complemento à técnica de eliminação de informações desnecessárias (*culling*) mostrada na seção 4.3, por exemplo: Após jogadores distantes e imperceptíveis serem removidos do processamento, pode-se aplicar a técnica de nível de detalhe naqueles jogadores que sobraram, fornecendo menos informações dos jogadores mais distantes e mais informações dos jogadores mais próximos.

Estruturas como o grid mencionado na seção 4.3 podem ajudar a selecionar quais objetos merecem receber mais atualizações e quais merecem receber menos. Por exemplo, aqueles objetos dentro do grid do jogador e vizinhos recebem a maior taxa de atualização, enquanto que os vizinhos dos vizinhos recebem uma taxa de atualização menor.

## 4.6
## Agregação de Mensagens

Outra técnica que permite reduzir o número de bytes enviados em qualquer comunicação é a agregação de mensagens [24]. Caso existam uma série de mensagens a serem enviadas para um determinado cliente, é melhor juntá-las numa única mensagem e enviá-la do que enviar uma série de mensagens pequenas. Isso porque mensagens enviadas por UDP adicionam às mensagens 28 bytes de cabeçalho (20 bytes para IP, 8 bytes para UDP) e mensagens TCP adicionam 40 bytes em cabeçalho (20 bytes para IP, 20 bytes para TCP). Logo, quanto menos mensagens são enviadas, menor o desperdício em bytes de cabeçalho para enviar as mensagens.

PUC-Rio - Certificação Digital Nº 0410842/CA