

Laboratório VISGRAF

Instituto de Matemática Pura e Aplicada

**Mundos Virtuais e Jogos por Computador:
PONG – Um Estudo de Caso**

*Antonia Lucinelma Pessoa Albuquerque
Luiz Velho*

Technical Report TR-01-02 Relatório Técnico

September - 2001 - Setembro

The contents of this report are the sole responsibility of the authors.
O conteúdo do presente relatório é de única responsabilidade dos autores.

1 Introdução

Mundos virtuais caracteriza-se por um ambiente gerado por computação gráfica, onde pessoas fisicamente distantes estão compartilhando esse ambiente, conectadas via rede, interagindo entre si e com objetos do ambiente virtual.

Este trabalho descreve a estrutura de mundos virtuais e os elementos técnicos envolvidos. Apresenta a implementação do jogo *Pong Game* em um ambiente distribuído e mostra sua relação com *Mundos Virtuais*.

O jogo foi desenvolvido em uma estrutura Cliente-Servidor, na plataforma Linux, usando recursos de comunicação entre processos IPC. Permite a conexão de dois clientes remotos, como jogadores. A conexão é do tipo orientada com padrão de protocolo TCP/IP.

A linguagem usada foi C, a parte gráfica e a interface foram desenvolvidas usando a GP.

2 Mundos Virtuais

Esses ambientes virtuais *3D* permitem a interação de múltiplos usuários em tempo real, conectados remotamente. Seu desenvolvimento envolve as tecnologias de realidade virtual, sistemas com simulação de vida, as comunidades virtuais, animação, agentes inteligentes, comportamento e sistemas distribuídos.

Mundos Virtuais visam atingir um senso de realismo e proporcionar experiências imersivas convincentes. Suas propriedades variam de acordo com a aplicação. Existem ambientes colaborativos - onde pessoas desenvolvem tarefas em comum e compartilham conhecimentos; existem os ambientes competitivos nos quais estão incluídos os jogos - onde os participantes competem entre si e tem o objetivo de vencer. Outros mundos virtuais são verdadeiras comunidades virtuais, mais abrangentes, incluem de forma acentuada as características de um mundo real, permitindo tarefas e comportamentos diversificados, tais como entretenimento com chats, jogos e trabalho colaborativo, simulação de vida natural, construção de casas, e para isso estabelece regras mais definidas de conduta; esses mundos são mais comumente chamados de *mundos virtuais habitados*. Porém é importante ressaltar que em todos os mundos virtuais existem regras, assim como no mundo real.

Como principais aplicações de mundos virtuais pode-se citar: jogos e entretenimento, educação à distância, chat *3D*, video chat, eventos sociais e culturais como Conferências, e-comércio em *virtual shopping malls* e *showrooms*, suporte remoto a usuários, treinamento militar e industrial, projetos colaborativos.

A presença independente de múltiplos usuários diferencia mundos virtuais dos sistemas padrões de realidade virtual ou sistemas padrões de jogos. A possibilidade de compartilhar objetos nos ambientes virtuais diferencia-os das salas de chat convencionais e a interação em tempo real diferencia da comunicação por e-mail ou web.

2.1 Termos e elementos que compõem mundos virtuais

Cada pessoa participante do mundo virtual tem sua representação virtual, que será visualizada pelos outros participantes e que navegará no ambiente virtual. Essa representação virtual é chamada de *avatar*. O avatar pode ter diferentes formas: humanos, animais, cartoons, formas geométricas, imagem real obtida de uma câmera, etc., podendo portanto ser 3D ou 2D, de acordo com o ambiente. Esse ambiente é também chamado de *cyberspace*.

A figura 1 mostra o *hall* de uma conferência realizada em um ambiente virtual.



Figura 1: Avatars98, a primeira conferência em um Cyberspace [1]

Biota é uma espécie de vida virtual que popula mundos virtuais e pode se assemelhar a plantas, animais ou outras formas de vida. Eles possuem vida independente no sistema e podem ou não ser interativos. Biotas mutam, se reproduzem, crescem e morrem de acordo com regras no mundo virtual. Programas genéticos, redes neurais e teoria da complexidade são importantes tecnologias usadas no desenvolvimento de biotas.

Bot é um programa que se comporta como um usuário em um ambiente de chat ou executa algum tipo de serviço útil aos usuários.

Agentes são programas projetados para atender necessidades dos usuários mais do que para auto-existir, como os biotas. Existe uma discussão em torno de quando um programa é simplesmente um programa ou um agente, e isso segue uma graduação relacionada com automaticidade, orientação para uma tarefa, flexibilidade, comunicação, etc. Os agentes se assemelham aos bots.

Baseado nesses elementos pode-se identificar genericamente dois tipos de entidades nos mundos virtuais: as *entidades passivas* e as *entidades ativas*. Entidades passivas tem seu comportamento pré-estabelecido no sistema, pois obedecem diretamente a uma programação, como exemplo tem-se o bot. As entidades ativas são tipicamente os avatares, que como representantes dos usuários tem um comportamento imprevisível depen-

dente da ação do usuário, mas obviamente dentro das limitações e regras do mundo. O sistema nunca saberá previamente quando o usuário irá acionar uma tecla ou botão, ou dirigir-se para esta ou aquela direção.

Mundos virtuais devem possuir as seguintes propriedades [2]:

Compartilhamento do espaço - todos os participantes devem ter a ilusão de estarem localizados no mesmo lugar, na mesma sala, etc, portanto este deve apresentar as mesmas características para todos os participantes.

Compartilhamento do senso de presença - isso se dá através do avatar. O avatar não precisa ter a forma de um humano, mas é importante que os participantes percebam a chegada ou saída de um participante através de seu avatar.

Compartilhamento do tempo - deve possibilitar a interação em tempo real, e dar aos participantes o sentimento de que as coisas acontecem simultaneamente para todos.

Ferramenta de comunicação - apesar das formas de visualização, a comunicação é necessária entre os participantes para acrescentar realismo e funcionalidade. Seja ela por texto, voz ou gestos.

2.2 Estrutura de Mundos Virtuais

Primeiramente, antes de definir a estrutura de um mundo virtual, deve-se ter em mente qual será a sua utilização. A partir deste ponto, pensar em como ele será funcional para esta aplicação. No caso do entretenimento, como atrair as pessoas para que elas frequentem esse ambiente e queiram voltar lá. Quais serão as possíveis atividades desempenhadas. Qual será a representação utilizada para os participantes. Esse mundo suportará avatares ou terá outra forma de compartilhamento desse ambiente. Quais tecnologias serão utilizadas para seu desenvolvimento: cenas 2D ou 3D, comunicação por texto ou voz, qual a estrutura de comunicação, existirá *bot* e *biota*, qual plataforma a ser utilizada.

2.2.1 Arquiteturas

Ambientes virtuais conectados em rede, também chamados de *Net-VE*, são difíceis de implementar corretamente ou eficientemente devido à sua complexidade pois envolvem vários tipos de sistemas tradicionais em uma única aplicação. *Net-VE* envolve as áreas de Sistemas Distribuídos, Aplicações gráficas e Aplicações interativas. Por serem aplicações distribuídas necessitam de uma arquitetura de comunicação.

As arquiteturas de comunicação para *Net-VE* são as seguintes:

- Duas pessoas em uma LAN (*Local Area Network*)
As pessoas interagem via PC ou estações de trabalho conectadas em LAN. As mensagens são enviadas entre as duas máquinas.
- Sistema Multiusuário Cliente-Servidor
Em um sistema Cliente-Servidor cada usuário envia dados para outros usuários via servidor.

- **Multiusuário Cliente-Servidor, com Arquiteturas Múltiplo-Servidores**
Nesta arquitetura tem-se vários usuários compartilhando um área de interesse no ambiente virtual mas ligados a diferentes servidores, e estas conexões servidor-servidor transmitem pacotes requisitados pelos usuários.
- **Arquiteturas Peer-to-Peer**
Peer-to-Peer significa que a comunicação vai diretamente de quem envia para quem recebe, um ou vários usuários que devem receber a informação.

2.2.2 Estado Dinâmico Compartilhado

Para obter as propriedades citadas na seção 2.1 é preciso garantir aos usuários uma visão consistente do estado do sistema. Mundos virtuais são sistemas dinâmicos por ser um processamento em tempo real, com usuários alterando frequentemente o estado do sistema. O estado dinâmico compartilhado consiste nesta mudança de informação que as múltiplas máquinas devem manter sobre o sistema, tais como: o que e quem está participando no momento, sua localização e suas ações, e dar a todos os participantes a mesma informação e visualização simultânea.

A precisão deste estado é fundamental para criar ambientes realísticos, sem isso cada usuário poderá agir independentemente e não haverá mais o senso de compartilhamento de lugar e tempo entre os usuários.

Pode-se dividir esse estado dinâmico em dois:

- **Estado Local** - é o que ocorre em cada cliente, ou seja, as alterações que cada cliente gera localmente em sua máquina e manda para o servidor. Essa ação do cliente está diretamente ligada à interação, que é a forma que o usuário tem para integrar ao sistema e expressar as suas ações.
- **Estado Global** - é tratado pelo servidor que atualiza todos os valores recebidos dos clientes e distribui para os mesmos, para garantir a consistência. Na construção de um mundo virtual é fundamental considerar uma forma de manter uma visão consistente do estado dinâmico para os participantes.

2.2.3 Interação

Um ponto forte de mundos virtuais é propiciar a interação não só entre usuários, mas do usuário com os objetos do mundo virtual.

A natureza de mundos virtuais contribui para a dificuldade de descrever e modelar interações. É necessário considerar a separação entre o que é interação controlada pelo sistema e o que é baseado no usuário. Mundos virtuais dão ao usuário recursos de interação que são fundamentalmente diferentes dos sistemas de computação convencionais. Nestes ambientes o usuário é um participante ativo, enquanto que nos sistemas tradicionais existe uma barreira entre a máquina e o usuário. Mundos virtuais visam um novo tipo de interação onde essa separação não seja clara ao usuário.

A interação local do usuário com o sistema se dá fisicamente através de um dispositivo, sejam eles os mais simples como teclado e mouse, e a interface do programa. A outra forma de interação se dá através do avatar, dentro do mundo virtual; existem diversas possibilidades para essa interação, como por exemplo: interação avatar-avatar, avatar-objetos virtuais, avatar-bot, etc.

2.2.4 Visualização e Simulação

Cada participante do mundo virtual deve ter a mesma visão do estado global desse ambiente, em tempo real. Para isso, cada usuário recebe o estado global atualizado e efetua o cálculo do rendering em tempo real para permitir a visualização do mundo.

Em mundos tridimensionais, o usuário tem o controle sobre a sua posição de observador, acarretando com isso uma maior complexidade pois o sistema permite que cada usuário tenha uma visão própria do mundo, que muda dinamicamente em função da sua navegação. Cada avatar se comporta como uma câmera virtual, o que requer o controle da visão de cada avatar e da visão global da cena.

O processo de rendering pode ser responsável pelo delay do sistema, dependendo da complexidade do modelo da cena.

Resolver o problema de rendering em tempo real é uma parte do esforço na obtenção de um mundo virtual mais convincente. Mas ver não é suficiente, é preciso tocar objetos no mundo, tocar o chão. O sistema precisa determinar o momento em que esses “contatos” ocorrem. Para isso usa-se o cálculo de detecção de colisão, em tempo real.

Nos primeiros sistemas de mundos virtuais, a detecção de colisão era ignorada, o que tornava o sistema bastante irreal, mas isso tem evoluído muito, apesar de ainda não ser satisfatório. Existem diferentes métodos para cálculo de detecção de colisão.

O processo de detecção de colisão pode ser separado em dois grupos: colisão de um objeto dinâmico contra um objeto estático e a colisão de um objeto dinâmico contra outro objeto dinâmico. Em um mundo mais complexo com grande número de objetos, é necessário determinar uma vizinhança de objetos que serão testados contra determinado objeto para que o teste de colisão não seja feito com todos os polígonos da cena, o que torna o cálculo bastante inviável. Uma vez determinado que houve colisão, isso deve acarretar uma reação em resposta a essa colisão.

Muitos sistemas calculam a detecção de colisão baseada em leis da física, isto é fundamental principalmente em sistemas que pretendem simular situações reais.

Esse conjunto de ações e subsistemas fazem de mundos virtuais um sistema bastante complexo. Como cada usuário pode ter comportamento independente e em tempo real, esse fato gera uma complexidade adicional para a comunicação. Para que esse resultado seja viável e tenha um tempo rápido de resposta para o cliente, faz-se necessário um tratamento de sincronismo nessa comunicação, feito no servidor.

3 Pong game - uma aplicação de mundos virtuais

O Pong é o primeiro *video-game*. Existe uma história complexa entre a Magnavox e a Atari envolvendo a criação desse jogo. Para melhor conhecer sua história e saber quem realmente inventou o primeiro video-game visite [4]. A figura 2 mostra a console de jogo da Magnavox e a versão do pong da Atari.



(a) Odyssey da Magnavox, a primeira console do jogo



(b) PONG da Atari: primeiro passo

Figura 2: O Primeiro Pong Game

Jogos multiusuários em um ambiente distribuído são um exemplo de mundos virtuais. Esta seção discute a implementação do *Pong Game* como um estudo de caso de mundos virtuais. A escolha do Pong-game para este trabalho deve-se ao fato do mesmo ser um jogo simples mas com todos os elementos de um jogo completo e com isso permite demonstrar os conceitos de mundos virtuais. O jogo é um exemplo de um mundo virtual competitivo, onde existem regras e o objetivo é competir e ganhar. A relação do jogo com mundos virtuais está na interação entre usuários conectados remotamente, comunicando-se através de uma ferramenta gráfica, que passa ao usuário o sentimento de compartilhamento do mesmo ambiente. Portanto contendo todos os subsistemas inerentes a mundos virtuais.

3.1 Estrutura do Programa

O Pong Game com dois jogadores é uma versão bidimensional do Ping-pong, onde cada jogador controla um goleiro que move-se apenas verticalmente. A bola percorre entre os jogadores, e o jogador sofre um goal quando a bola passa por ele e toca a parede de fundo.

Esta aplicação possui duas entidades ativas, que são as representações dos goleiros - aqui representados por dois retângulos, como mostra a figura 3. Como cada goleiro é controlado pelo usuário, o seu movimento é imprevisível, e o computador remoto não pode prever quando o goleiro irá tocar a bola. Esse retângulo aqui chamado de goleiro, é

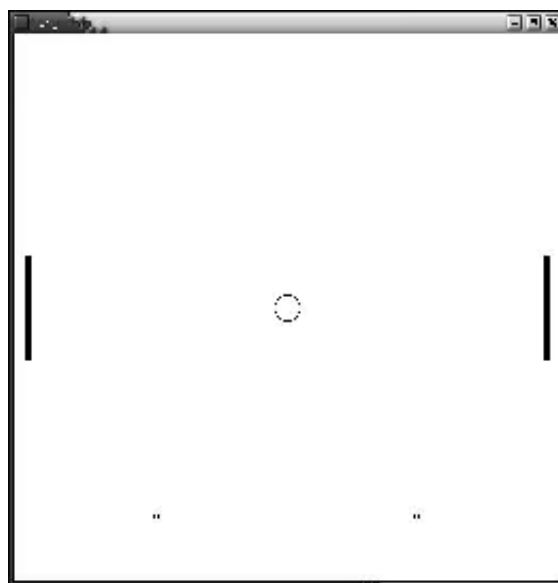


Figura 3: Pong Game

o avatar de cada usuário. A bola é uma entidade passiva, uma vez que quando ela toca o jogador ou uma parede, o seu movimento é previsível.

O cliente move o goleiro usando as teclas de ponto (para cima) e vírgula (para baixo), e envia a informação de movimento para o servidor. O servidor efetua o cálculo de teste das coordenadas do goleiro com a fronteira da janela, calcula a posição da bola e devolve aos clientes como coordenadas, para ser desenhado. Quando um jogador sofre um goal o servidor atualiza o placar e envia como texto aos clientes.

O *Pong game* apresentado tem duas limitações importantes quando comparado com mundos virtuais. Cada jogador pode mover-se em apenas um eixo, quando em mundos virtuais é esperado que este possa mover-se livremente em um ambiente tridimensional. A segunda limitação é que o sistema suporta apenas dois objetos ativos, ao contrário de um mundo virtual mais complexo que deve suportar muitos objetos ativos.

O jogo utiliza a arquitetura de um Sistema Multiusuário Cliente-Servidor. Cada participante, cliente, é conectado ao servidor que é responsável por distribuir a informação aos participantes. O servidor aceita dois tipos de informação: texto e coordenadas do jogador. O texto é usado pelos participantes como chat, o servidor apenas distribui a mensagem.

A figura 4 ilustra a conexão entre servidor e clientes, a transmissão dos dados e a saída apresentada pelos clientes.

Na parte de sistema distribuído tem-se a comunicação entre cliente e servidor e a virtual comunicação entre clientes. No jogo, a comunicação utiliza a estrutura cliente-servidor, através de sockets.

A interação é distribuída e em tempo real permitindo que um usuário veja a atividade de cada participante e expresse sua reação em tempo real. No jogo podemos identificar os

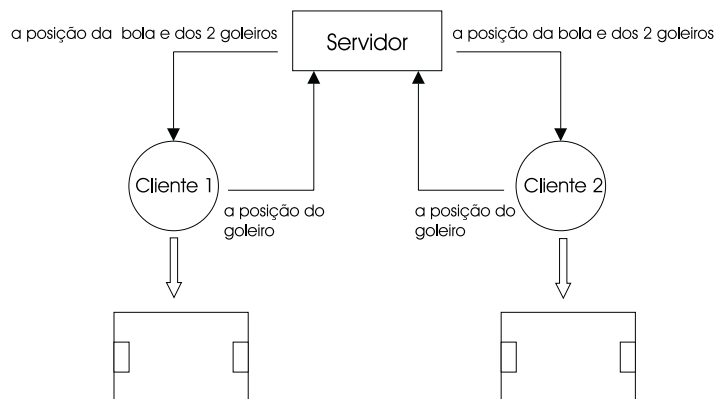


Figura 4: Arquitetura do Sistema para o Pong Game

dois estados:

- Estado local - o estado local é alterado pelo cliente, e é independente para cada usuário, para permitir liberdade de ação dentro daquele mundo, segundo suas regras. No jogo, o estado local é o comportamento do goleiro, comandado pelo jogador que altera a sua posição livremente. Esses valores são enviados ao servidor que efetuará a distribuição, isto é, dará conhecimento desse estado aos demais participantes, no caso do pong, o outro jogador.
- Estado global - o estado global é gerenciado pelo servidor e diz respeito ao que é comum para todos os usuários, e tem os mesmos valores numéricos. O servidor recebe o estado local do cliente, faz a atualização do estado global e distribui para os clientes. No pong, o estado global refere-se ao comportamento da bola, atualização da posição dos jogadores e score do jogo.

O rendering em tempo real ocorre localmente em cada cliente, e permite a visualização do jogo desenhado na tela de cada cliente. O cliente recebe o estado global do servidor, faz o render e exibe na tela.

A detecção de colisão no pong é feita para a bola, quando esta toca as paredes ou o goleiro.

3.2 Pseudocódigo

A seguir tem-se a descrição do pseudo-código do pong-game, para o cliente e o servidor.

Em linhas gerais: o Servidor entra em operação, fica escutando. Se um cliente conectou, manda estado inicial. Quando tem 2 conectados: recebe informação sobre o goleiro, atualiza estado e devolve coordenadas para todos. Quando recebe texto, envia para todos. O Cliente conecta com o servidor, recebe a posição da bola e dos goleiros, desenha; move o goleiro, manda a posição do goleiro para o servidor. Digita texto e manda pro servidor.

3.2.1 Servidor

Na rotina Main, o servidor entra no ar e fica em loop infinito.

Rotina Main

testa se foi fornecido o nível do jogo
põe o servidor no ar (start_server)
inicializa as posições do goleiro, da bola e score
loop
 Processamento do servidor (server_loop)
fim main

A rotina *start_server* põe o servidor no ar e prepara para receber conexões.

Rotina start_server

recebe o número da porta como parâmetro
inicializa o servidor
fim start_server

A rotina *server_loop* gerencia a comunicação cliente-servidor.

Rotina server_loop

declara struct para controle do tempo (t0 t1 td tv)
lê um descritor
pega a hora e põe em *t0*
espera na select - agora de acordo com timeout
for toda a lista de descritores **do**
 if recebeu um sinal de comunicação: **then**
 testa o número de conexões
 if tem 1 **then**
 Manda estado inicial e aviso para esperar o segundo jogador
 if tem 2 **then**
 Manda o estado inicial e espera 2 segundos para começar o jogo
 if o sinal já é de alguém que já está conectado **then**
 Processa mensagem do cliente
pega a hora novamente
calcula o intervalo de tempo entre as tomadas de tempo
Atualiza estado
Manda para o cliente
fim do server_loop

O servidor recebe o estado local do cliente pela rotina *parse_message*, faz a atualização do estado global e distribui para os clientes.

Rotina parse_message

```
if recebeu estado local do cliente  $\leq 0$  then
  if o cliente desconectou then
    Retira da fila de descritores
    Subtrai o contador de conexões
    Seta o estado para "não jogando"
    Executa o estado inicial do jogo
  else
    if o cliente teclou 's' then
      altere o estado do jogo: pára se está jogando, joga se está parado
    if o sender é left then
      testa a sua coordenada y, adicionada à velocidade recebida, com os limites superior e inferior do campo e atribui o menor a y
    else {idem para o right}
      testa a sua coordenada y, adicionada à velocidade recebida, com os limites superior e inferior do campo e atribui o menor a y
  fim parse_message
```

A atualização do estado global é feita na rotina *update_state*, que por sua vez chama outras rotinas necessárias para a atualização desse estado.

Rotina update_state

```
if a bola não está em jogo then
  Retorna
  b1 recebe a nova posição da bola, escalada pelo intervalo dt {chama a colide recursivamente}
while colidir mas não for goal do
  atualiza a posição da bola
  reflete a bola
  atualiza a posição da bola
if goleiro não defendeu, e a bola ultrapassou os limites das laterais then
  atualiza score
fim update_state
```

A rotina *send_to_clients* manda esse estado para os clientes.

Rotina send_to_clients

```
percorre os descritores
manda o estado global
fim send_to_clients
```

A rotina *collide* faz o cálculo de colisão da bola com os limites do campo e com o goleiro. Esta rotina é recursiva e executa 4 vezes para os planos top, bottom, left e right.

Rotina Collide

recebe a última posição da bola p0 e sua próxima posição projetada pela velocidade da bola p1
calcula a distância de cada ponto (p0 e p1) ao plano, pela função ppl_dist
if uma das distancias for negativa **then** {verifica se um dos pontos ultrapassou o plano}
 calcula o ponto em que houve a interseção, e atribui a p1, como próxima posição para a bola
 testa se houve goal ou não - pela função out_of_kpr
chama collide novamente
fim da collide

A rotina *reflect* calcula a reação da bola após a colisão.

Rotina reflect

if a bola ultrapassou uma das laterais **then** {quando colide e não é goal, reflete a bola}
 calcula sua nova coordenada x e inverte a coordenada da velocidade em x
if a bola ultrapassou uma top ou bottom **then**
 calcula sua nova coordenada y e inverte a coordenada da velocidade em y
fim da reflect

A rotina *game_score* atualiza o score do jogo

Rotina game_score

calcula um valor randômico
if o jogador que sofreu o goal foi o right **then**
 posiciona a bola na sua frente
 calcula a nova direção da bola, usando esse valor randômico para y , e -1 para x .
else
 if o jogador que sofreu o goal foi o left **then**
 posiciona a bola na sua frente
 calcula a nova direção da bola, usando esse valor randômico para y , e 1 para x .
escala novamente a velocidade da bola pelo nível do jogo
fim game_score

3.2.2 Cliente

Na rotina Main o cliente conecta com o servidor, efetua inicializações e entra em loop infinito.

Rotina Main

```
conecta com o servidor
inicializa parte gráfica
loop
  Processa cliente (client_loop)
fim main
```

No cliente, a alteração do estado local ocorre na rotina *client_loop*.

Rotina client_loop

```
if recebeu evento do servidor then
  atualiza display
  pega evento do teclado e coloca o char no buffer
  caso 'q': termina o programa
  caso '.' : coloca em mov a velocidade positiva
  caso ',' : coloca em mov a velocidade negativa
  manda estado local para o servidor
fim client_loop
```

Essa rotina efetua o rendering da cena e desenha na tela.

Rotina Atualiza display

```
if tem texto no buffer que veio do servidor then
  Imprime texto
  Limpa tela
  Desenha bola
  Desenha goleiro direito
  Desenha goleiro esquerdo
  Pinta o score na janela
fim Atualiza display
```

Referências

- [1] Avatars98. <http://www.digitalspace.com/papers/av98ars.html>. [visitado em agosto de 2001].
- [2] Sanddep Singhal e Michael Zyda. *Networked Virtual Environments*, Addison Wesley, 1999.
- [3] Jan de Bruin, Dirk-Jan de Bruin, Bruce Damer e Stuart Gold. *Conferences and Trade Shows in Inhabited Virtual Worlds: A Case Study*. ACM Multimedia Conference, Nov 1999.
- [4] Bienvenue sur PONG-Story. <http://home.worldnet.fr/winter/pong/intro.htm> ou <http://www.pong-story.com/>. [visitado em agosto de 2001].
- [5] *Virtual Worlds: First International Conference, VW '98, Paris, France, July 1-3, 1998: Proceedings*, Vol. 143. Springer-Verlag New York, Incorporated.
- [6] Shamus Smith and David Duke. *The Hybrid World of Virtual Environments*. Eurographics 1999. Volume 18, number 3.
- [7] Beej. <http://www.ecst.csuchico.edu/~beej/guide/net/>. Beej's Guide to Network Programming. [visitado em agosto de 2001].
- [8] W. Richard Stevens. *Unix Network Programming*. Prentice-Hall PTR, 1990.