Um estudo para visualização de objetos com geometria dinâmica em jogos multi-jogador

Lauro Eduardo Kozovits
Computer Science Department, PUC-Rio
lauro@inf.puc-rio.br

Marcelo Dreux
Department of Mechanical Engineering, PUC - Rio
dreux@mec.puc-rio.br

Bruno Feijó
Computer Science Department, PUC-Rio
bruno@inf.puc-rio.br

PUC-RioInf.MCC34/03 October, 2003

Abstract: In the last years, many techniques have been developed to improve the quality, in terms of graphical effects, of multiplayer games, a specialized field in networked virtual environments (Net-VE). Although new techniques have been developed to provide, also, efficient network use in those games, some challenges in this area still remain. A new technique to distribute objects among game players during a session, based on its geometric representation, is proposed in this work in order to overcome the Internet bandwidth limitations.

Keywords: distributed virtual environment net-VE, multiplayer game design, rendering systems, level of detail algorithms (LOD), streaming, B-Rep, CSG, metaball, progressive transmission.

Resumo: Nos últimos anos, muitas técnicas têm sido desenvolvidas para melhorar a qualidade gráfica de jogos multi-jogador, uma especialização da área de ambientes virtuais ligados por rede. Embora muitas técnicas tenham sido desenvolvidas para também permitir o uso eficiente dos recursos da rede, alguns desafios nesta área ainda persistem. Uma nova técnica para distribuição de objetos entre os jogadores durante uma sessão, explorando uma representação geométrica concisa, é proposta neste trabalho de modo a contornar as limitações de largura de banda na Internet.

Palavras-chave: ambientes virtuais distribuídos, jogos multi-jogador, *rendering*, algoritmos de nível de detalhe (LOD), fluxos de dados pela rede, B-Rep, CSG, metaball, transmissão progressiva.

This work is being sponsored by CNPq and ICAD.

1. Introdução

Jogos multi-jogador são uma área multidisciplinar que tem merecido a atenção tanto de empresas quanto do meio acadêmico. No primeiro caso, este interesse dá-se pelo volume de receitas que podem ser obtidas por assinantes do serviço via Internet. Em contrapartida, a demanda por conteúdo novo e a existência de diversas categorias de jogos demanda enormes investimentos na manutenção de equipes de *design* e de programação bem como em hardware para suportar um número elevado de usuários simultâneos.

Do ponto de vista acadêmico, os desafios nesta área, uma especialização da área de ambientes virtuais distribuídos, bem como sua já dita natureza multidisciplinar permitem, atualamente, uma florescente pesquisa nas áreas de computação gráfica, redes, interface com o usuário, multimídia etc.

Se por um lado, podemos considerar que os avanços tanto de CPUs quanto de processadores gráficos (GPUs) resolveram muitas das questões ligadas aos requisitos visuais destes ambientes, pelo lado da tecnologia de rede muito ainda existe a fazer para contornar as limitações tanto de largura de banda quanto de latência da Internet. É importante ressaltar, neste momento, duas características que diferenciam ambientes virtuais ligados por rede e jogos multi-jogador:

- Em ambientes de jogos, ao contrário de simulações, busca-se o entretenimento, a satisfação dos usuários e não necessariamente um realismo fisico ou uma acuidade visual. A expressão que define a meta do jogo como sistema produto é a obtenção de uma boa "jogabilidade". Isto permite que certos "truques" possam ser usados para contornar limitações tecnológicas o que não seria admissível em certas simulações realistas.

- Em jogos multi-jogador, nem todas as tecnologias utilizadas em ambientes virtuais ligados por rede são usualmente empregadas ou mesmo podem ser empregadas. Por exemplo, pelo fato de a Internet ser a principal, senão a única rede que une todos os jogadores de um dado jogo, não é possível o uso da tecnologia *broadcast* disponível apenas em redes locais. Da mesma forma, a tecnologia *multicast*, apesar de presente, não está disponível na Internet em larga escala. Finalmente, apesar do crescimento contínuo das chamadas tecnologias de banda larga, o equipamento mais comum para permitir a conexão entre os jogadores são modems analógicos. Isto é uma limitação séria que também motiva o presente trabalho.

Cabe ainda ressaltar que muitos trabalhos para contornar as limitações da rede têm sido feitos, principalmente aqueles ligados a tecnologia de *dead-reckoning*, termo original de técnicas de navegação e introduzido na área de ambientes virtuais no simulador militar SIMNET (Pope, 1989). Iremos, também neste trabalho, apresentar outras técnicas gráficas que permitem contornar as limitações da rede.

O foco aqui, portanto, é abordar aspectos geométricos ligados tanto ao compartilhamento de objetos quanto à exploração da geometria do ambiente na diminuição do tráfego da rede.

Assume-se muitas vezes, que toda a geometria do ambiente está disponível no momento de instalação ou num momento anterior ao início de uma determinada sessão, o que seria o caso de uma atualização do ambiente original. No entanto, para certas categorias de simulações ou jogos, pode ser necessária uma atualização da geometria, durante uma sessão ativa, exemplificando:

- um objeto ou terreno pode ter sua geometria alterada em função de uma explosão, colisão com outro objeto ou mesmo por intervenção de um usuário com poderes para tal.

- um novo usuário, com seu avatar (objeto que o representa) e objetos personalizados pode entrar numa sessão em andamento e os demais usuários não têm, ainda, a geometria para representá-lo assim como suas ferramentas.
- novos objetos são criados dinamicamente por outros jogadores ou pelo sistema responsável pela sessão em andamento.

Levando-se em conta o que foi mencionado acima, são analisados trabalhos relacionados ao envio ou modificação de objetos durante uma sessão de modo a não prejudicar a jogabilidade do ambiente.

São descritas técnicas visando explorar características do ambiente e evitar o envio de informações desnecessárias. Além disso, é importante também a análise das formas de modelagem ou representação de objetos o que permite explorar estas características, para otimizar seu compartilhamento em diversas estações participantes da simulação.

Finalmente, o presente trabalho propõem uma estratégia alternativa para transferência de certas classes de objetos entre jogadores, durante o andamento de uma sessão, ao explorar sua representação geométrica.

2. Técnicas correntes de representação geométrica em jogos

Neste item, são analisadas técnicas de otimização correntemente utilizadas com base na representação geométrica do ambiente de um jogo ou simulação. A análise crítica destas tecnologias permite o leitor situar melhor a nova técnica proposta neste trabalho, bem como justificar seu uso em conjunto com estas técnicas.

2.1 Técnicas para se evitar o envio de informações desnecessárias explorando características geométricas do ambiente

Embora sem um registro formal na literatura acadêmica, é comum, em jogos realizados em ambientes fechados, como a maioria dos jogos em primeira pessoa *(FPS-first person shooters*) como Doom, Quake etc, utilizar-se as técnicas de BSP (Fuch, 1980) e PVS (ou simplesmente portais) (Tell, 1991), (Tell, 1992) para, além de eliminar o envio de mensagens de atualização de posição de personagens, definir áreas de interesse do ambiente para evitar a atualização desnecessária de objetos da cena. Exemplificando: se uma ação se desenrola na sala A e outra na sala B, não é necessário que objetos modificados ou introduzidos na sala A, sejam também enviados para jogadores na sala B.

Obviamente, se um jogador sair da sala A e se dirigir a sala B, o jogo deverá dispor de uma estrutura de dados para inventariar objetos já disponíveis ou não e fazer a atualização do banco de dados geométrico deste jogador apenas quando necessário.

É comum o uso de técnicas especiais em jogos para se conseguir efeitos visuais sem onerar o processamento por exemplo, a técnica de *sprites* consiste em se aplicar, a cada *frame*, imagens(em geral de personagens) sobre uma cena já finalizada, evitando-se o custo da criação e do processamento de um modelo geométrico.

Outra técnica comum e semelhante são os *billboards*, muito utilizados, por exemplo, para modelagem de árvores em ambientes 3D. Neste exemplo, é curioso o fato de que o observador pode dar uma volta em torno de uma árvore criada pela técnica de *billboard* e, mesmo assim, ver continuamente sempre a mesma imagem por ela ser sempre exibida projetada num plano perpendicular à visada do observador.

Estas imagens são aplicadas na cena utilizando-se composição baseada em canal alfa ou mesmo definindo-se regiões totalmente transparentes por meio de uma certa cor chave presente nestas imagens.

Em ambientes abertos, uma estratégia que pode ser adotada é o uso de *impostors* (Schaufler, 1995) e *nailboards* (Schaufler, 1997). A idéia que está por trás dos impostors ou nailboards consiste em representar um objeto tridimensional através de um *sprite*, porém diferentemente do conceito padrão de *sprites* ou *billboards*, estes objetos estão realmente definidos como modelos e são submetidos ao processo de *rendering* durante a aplicação corrente e projetados num plano como textura com alpha. (Schaufler, 1995) e (Schaufler, 1997) descrevem estes métodos como adequados para minimizar o número de vezes em que se deve fazer o *rendering* de algum determinado objeto, que possui certa complexidade geométrica (Forsyth, 2001) e, portanto, cara para o *pipeline* de tempo real.

Os impostors serviriam então, como uma espécie de buffer para estes objetos, de forma a permitir seu reaproveitamento enquanto ainda "servem" para uma determinada posição do observador.

Além disso, a imagem gerada para estes objetos pode ser proporcional ao tamanho que ocupam na imagem final. Assim, se estiverem muito distantes, terão seu *rendering* numa resolução pequena, na medida em que se aproximam do observador e, portanto, aumentam em tamanho no plano de visualização, estas imagens são refinadas para resoluções maiores.

Em (Damon, 2003) descreve-se uma forma de implementar impostors utilizando técnicas de *rendering* em memória de vídeo, recurso que está disponível nas placas gráficas mais recentes, podendo aumentar o número de objetos sendo representados com esta técnica, desde que haja memória de vídeo suficiente.

As técnicas de impostors e nailboard, usadas localmente, podem também ser exploradas em ambientes ligados por rede: um objeto introduzido no ambiente por um jogador, pode ter sua geometria transferida para participantes próximos e ter apenas uma imagem calculada a ser transferida para os jogadores distantes. Compara-se obviamente, o tamanho da imagem a ser transferida com o tamanho da representação da mesma estrutura 3D para se saber quando é vantajosa esta abordagem em relação ao envio do modelo completo.

Em jogos com número elevado de jogadores (*massive multiplayer*) empregando arquitetura cliente-servidor, é comum a divisão do ambiente em áreas com fronteiras fixas ou variáveis (Beardsley, 2003) de modo a explorar áreas de interesse e agrupamentos de jogadores. Por exemplo, cada vilarejo numa planície onde se passa a ação poderia ser atendido por um servidor e a informação do ambiente como um todo compartilhada entre os servidores.

De acordo com (Beardsley, 2003), a divisão do mundo é algo extremamente específico e irá depender da natureza do jogo. Por exemplo, num jogo em um terreno aberto, algo típico em RPG (*role playing game*) *online*, uma divisão possível do ambiente pode ser empreendida utilizando-se uma estrutura 2D, mesmo sendo o jogo modelado em 3D.

Já no caso de uma simulação espacial ou combate aéreo, tornar-se-ia necessário pensar em estruturas como Octree (Meagher, 1980) por exemplo, para a divisão deste ambiente entre servidores.

Um problema comum é o gerenciamento da migração de avatars entre as áreas ou ainda sua permanência na região de fronteira das áreas atendidas pelos servidores de jogos.

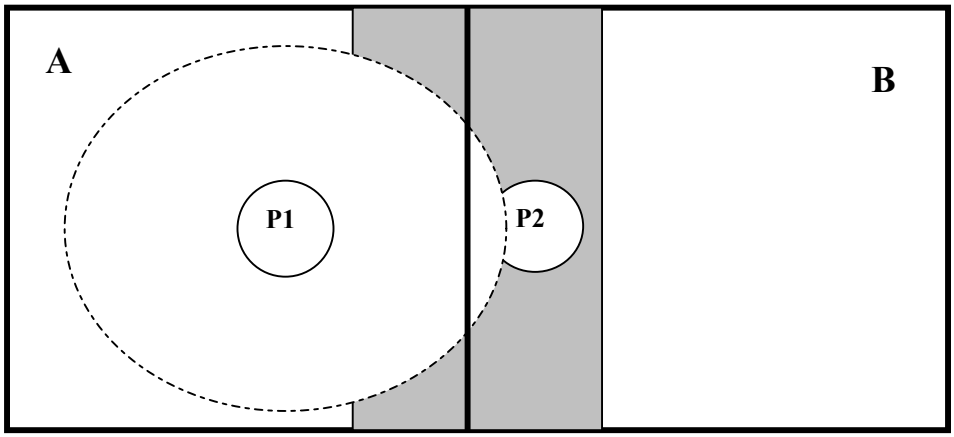

figura 1: jogadores P1(e sua zona de visibilidade) e P2 atendidos por dois servidores diferentes(A e B)

A figura 1 ilustra uma situação problemática comum a implementações utilizando fronteira fixa: os servidores A e B compartilham uma área de borda (mais escura) onde objetos presentes nesta região são tratados por ambos os servidores A e B. A elipse tracejada da figura mostra que o jogador P1 consegue observar P2 (representado também em A por estar numa região de fronteira). No entanto, mesmo que P2 tenha a possibilidade física de observar P1, o avatar P1 não será visto (por P2) por não estar sendo tratado pelo servidor B. Isto coloca o jogador P2 em desvantagem em relação a P1 que pode atacar sem ser visto.

2.2 Otimização baseada na representação geométrica do objeto

Para podermos propor uma nova técnica de compartilhamento de objetos num ambiente ligado por rede, é necessário analisar as principais formas de modelagem e representação geométrica bem como técnicas já comumente empregadas para sua exibição otimizada em ambiente de rede. As formas mais comumente estudadas e utilizadas são: a representação baseada em malhas poligonais (B-Rep) (Baumgart, 1975) e a geometria sólida construtiva (CSG) (Goldstein, 1971). Em adição a estas duas técnicas, devemos mencionar que, nos últimos anos, alguns trabalhos têm sido publicados sobre *metaballs* (Blinn, 1982) ou *soft objects* (Wyvill, 1986), cujo interesse dá-se por ser esta uma ferramenta útil para representação de formas orgânicas.

2.2.1 Objetos criados por Constructive Solid Geometry CSG

Geometria sólida construtiva ou CSG baseia-se no uso de primitivas geométricas e operações de conjunto para representação de objetos complexos. É uma representação bastante concisa e particularmente útil para objetos usináveis. Para compreendê-la, considere as operações a seguir sobre dois sólidos A e B e observe na figura 2 a construção de um sólido C utilizando estas operações:

• UNIÃO: Inside(A) || Inside(B)
— possibilita a criação de um objeto pela união do sólido A com o sólido B
• INTERSEÇÃO: Inside(A) && Inside(B)
— remove qualquer parte de A ou de B que não seja comum aos dois.
• DIFERENÇA: Inside(A) && (! Inside(B))
— remove do objeto A qualquer interseção existente entre A e B

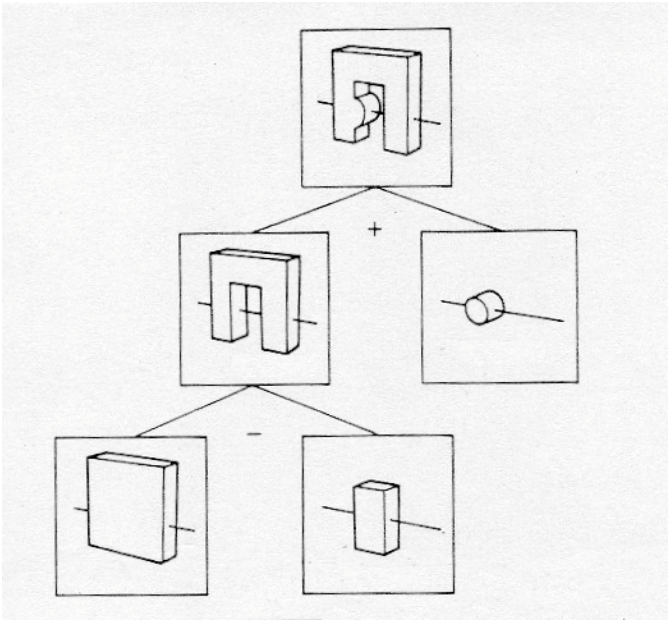

figura 2: uso de uma árvore de composição por CSG descrevendo a construção de uma peça (Roth, 1982)

CSG teve um aumento de interesse ao ser apresentada em conjunto com a técnica de ray-casting para sua visualização (Roth, 1982) possibilitando a construção de modeladores semi-interativos vide figura 3. No entanto, como a maior parte do hardware gráfico atualmente disponível em larga escala e moderado custo, faz o *rendering* de objetos representados como malhas poligonais, torna-se necessário o uso de algoritmos para conversão de modelos CSG para malhas.

Diversos algoritmos foram publicados para a conversão da representação B-Rep (baseada em malhas poligonais) para CSG e de CSG para B-Rep, sendo os últimos consideravelmente mais simples e eficientes. Três exemplos de algoritmos CSG para B-Rep são (Shankar, 1997), onde são discutidos algoritmos para computação de operações boleanas entre sólidos definidos por splines paramétricas; (Wiegand, 1996), que trata da visualização interativa de sólidos obtidos por CSG em estações gráficas utilizando primitivas OpenGL e (O'Loughlin, 1999), que apresenta uma técnica para visualização em tempo real em PCs de objetos modelados por CSG.

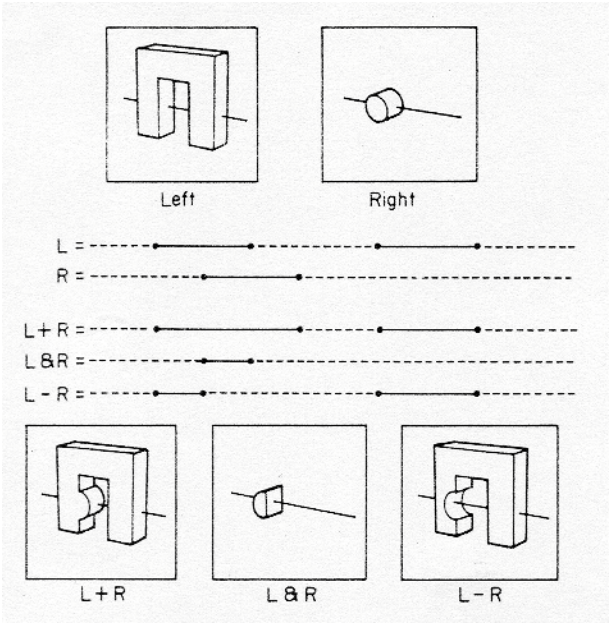

figura 3: uso de ray-casting permitindo uma solução unidimensional (ao longo da trajetória de um raio) para um problema de combinação 3D (Roth, 1982)

2.2.2 Objetos criados por Metaballs ou Soft Objects

Metaballs, também conhecido como *blobby objects*, são um tipo de técnica de modelagem utilizando superfícies implícitas. Podemos imaginar um metaball como uma partícula envolta por um campo (ou função) de densidade, onde a densidade atribuída à partícula (sua influência) decresce com a distância de sua localização.

Uma superfície pode então ser definida tomando-se uma isodensidade neste campo criado. Quanto maior o valor desta isodensidade mais perto ela estará da partícula geradora.

O aspecto poderoso de metaballs como ferramenta de modelagem é a maneira como se pode combinar estes campos de densidade. Simplesmente somando-se as influências de cada metaball, podemos obter fusões suaves destes campos de influência esféricos(veja figura 4).

A chave para o uso de metaballs consiste na definição de uma equação para especificar a influência de uma determinada partícula sobre um determinado ponto no espaço. (Blinn, 1982) usou campos com decaimento exponencial para cada partícula combinado a uma curva Gaussiana de altura b e desvio padrão a. Se r é a distância da partícula para uma certa localização do campo, a influência da partícula é b*exp(-ar). Por razões de eficiência, pode-se trocar a distância pelo quadrado da distância para evitar-se o cálculo da raiz quadrada. O resultado da densidade de uma certa localização no campo é simplesmente o somatório das contribuições de todas as partículas.

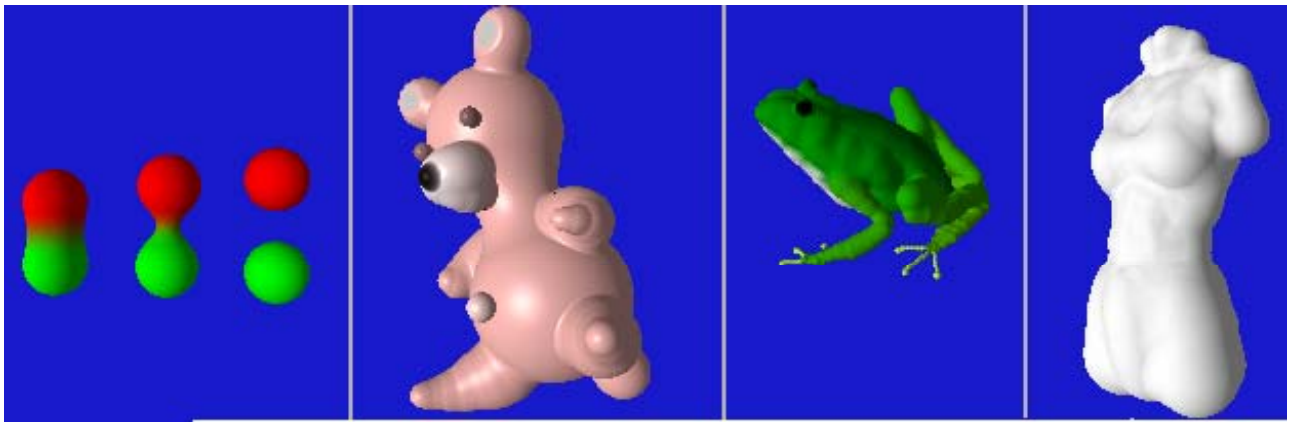

figura 4: exemplos de objetos modelados pela técnica de metaball utilizando um editor escrito em Java (Yoshizu, 1997)

(Wyvill, 1986) simplificou os cálculos definindo um polinômio cúbico baseado no raio de influência para uma partícula combinado à distância do centro da partícula a um dado local do campo em questão. O valor da influência de uma partícula é dado pela fórmula a seguir:

$$D(r) = \begin{cases} a\left(1 - \frac{4r^6}{9b^6} + \frac{17r^4}{9b^4} - \frac{22r^2}{9b^2}\right) \\ 0 \end{cases}$$

É importante ressaltar duas coisas na formulação acima:

- como aparecem as potências 6, 4 e 2, evita-se o cálculo da raiz quadrada ao se calcular a distância "r" de um ponto do espaço à partícula; podemos portanto, pensar nesta fórmula como uma função cúbica da distância ao quadrado.
- ao contrário da curva gaussiana, a função acima é zero para distâncias superiores a "b" o que permite eliminar o cálculo das contribuições de partículas distantes.

Uma vez que o campo, por eles agora chamado de "*soft objects*", é gerado, qualquer técnica para visualização de campos escalares pode ser usada. Em (Wyvill, 1986) uma versão eficiente do algoritmo "*Marching Cubes*" (Lorensen, 1987) para construção de superfícies 3D é também apresentada.

O ponto chave para o interesse de *metaballs* ou *soft objects* neste trabalho é o fato de a representação de um objeto relativamente complexo poder ser feita a partir de uma fórmula e um número relativamente pequeno de partículas.

Isto faz com que a representação desta categoria de objetos, de modo semelhante ao que ocorre com técnica de CSG, possa ser, ao mesmo tempo, relativamente concisa e independente do número final de polígonos de sua representação.

Além disso, se CSG é uma técnica ideal para objetos usináveis, a técnica de metaball, por outro lado é ótima para modelagem de formas orgânicas.

2.2.3 Objetos definidos por malhas poligonais

O fato de a maioria das placas gráficas fazer o *rendering* de objetos representados por malhas poligonais, faz desta representação a mais importante no desenvolvimento de jogos 3D. Se, por um lado, os requisitos de qualidade visual cresceram junto com a capacidade gráfica dos PCs comuns, a capacidade de transmissão de dados por unidade de tempo ou largura de banda da Internet por outro lado, ainda é muito limitada se comparada com uma rede local por exemplo.

O volume de informação correspondente a um modelo geométrico dos jogos atuais impõe portanto, o uso de técnicas especiais para permitir uma transmissão eficiente de dados tanto no momento de instalação de um jogo via Internet("*download*") quanto e, principalmente, durante uma partida em andamento.

É importante ressaltar que o principal problema está não apenas na primeira instalação, mas sim no fato de que muitos ambientes virtuais estão *online* permanentemente e em constante transformação. Assim, quando um usuário entra numa sessão ele deverá receber, de forma eficiente, as modificações relevantes ocorridas desde sua última presença ou mesmo transmitir aos demais usuários sua geometria específica ou customizada.

Uso de LOD (level of detail) em malhas poligonais

Nos últimos anos, muitos trabalhos (Garland, 1999) têm sido publicados sobre LOD que é uma forma de representar um determinado objeto poligonal, numa resolução menor. Esta representação é calculada de modo a preservar, razoavelmente, a aparência do objeto (vide figura 5), sem contudo comprometer recursos computacionais como memória e capacidade do pipeline gráfico, sobretudo em aplicações interativas.

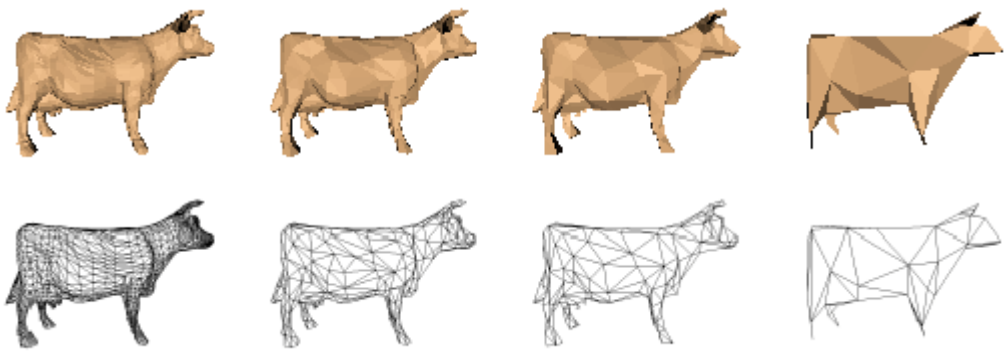

figura 5: exemplos de um objeto em multiresolução (Garland, 1999)

São diversas as técnicas mas, em geral, utilizam alguma métrica para eliminar ou fundir polígonos de modo a alcançar uma estrutura mais simples. Usualmente são técnicas progressivas que expressam visualmente uma representação ainda acurada do objeto sendo tratado.

Tipicamente, objetos situados à distância são candidatos a serem representados com um número menor de polígonos. À medida que nos aproximamos destes objetos, ocorre um aumento da resolução de forma a manter a acurácia visual.

Em um número grande de simulações e jogos, devemos tratar o nível de detalhe de dois grupos distintos: os objetos representando os jogadores e elementos discretos do ambiente e o terreno onde se passa a ação. Em (Ulrich, 2000) é apresentado, em detalhes, um algoritmo para LOD contínuo de terrenos baseado em quad trees (Finkel, 1974) adaptativas. Dependendo da

alteração da posição do observador, o algoritmo altera a resolução da malha. Este algoritmo é divido em dois passos básicos: update() e render().

Durante o update(), faz-se uma subdivisão recursiva de cada quadrante (vide figura 6) para testar se a coordenada dos novos pontos criados, calculados como média da vizinhança dos pontos adjacentes, difere acima de um certo erro do que é definido pelo mapa de alturas (*height field*). Caso isto ocorra, a recursão prossegue gerando novos pontos e quadrantes.

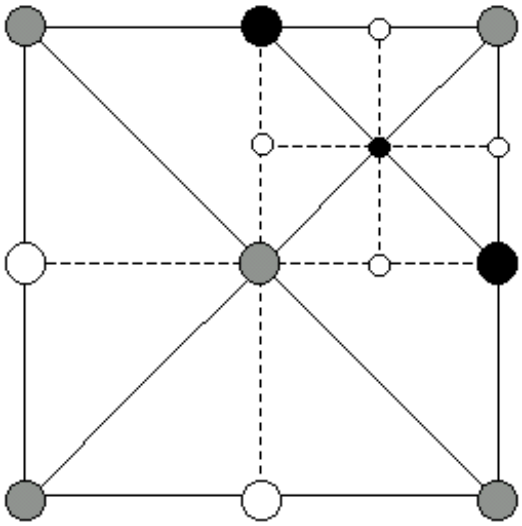

figura 6: quadtrees adaptativas: passo 1

Eventualmente, a necessidade de geração de novos pontos (em preto nas figuras 6 e 7) implica na subdivisão do quadrante vizinho: no exemplo, o quadrante inferior direito da figura 7. Após chamadas recursivas nos dois quadrantes, a malha pode, dependendo da posição do observador ser agora apresentada, na etapa render( ), com os novos vértices como aparece na figura 8. Se o observador voltar a se distanciar, o processo inverso irá ocorrer.

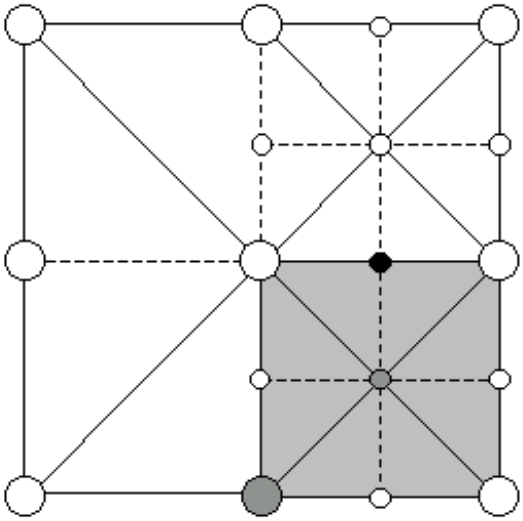

figura 7: quadtrees adaptativas: passo 2

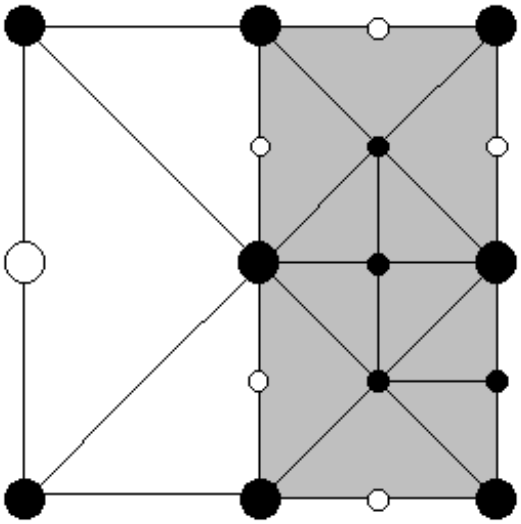

figura 8: quadtrees adaptativas: situação final

Embora, a motivação original que conduziu aos primeiros trabalhos sobre LOD tenha sido obter uma forma de representar eficientemente terrenos ou objetos obtidos por scanners 3D ricos em polígonos, a utilidade de LOD em ambientes ligados por rede também se faz presente. Por exemplo:

- suponha que um determinado jogador seja "teletransportado" para o ambiente; a estação de trabalho deste jogador pode calcular a melhor representação dada a distância dos demais observadores e enviar uma resolução mais econômica baseada em alguma das técnicas de LOD disponíveis.

Uma adaptação importante da técnica de LOD, diz respeito à transmissão do nível de detalhe (*mesh* ou *LOD streaming*) (Guéziec, 1999). A especificação MPEG-4 ver 2 (MPEG-4, 1998) já contempla a transmissão progressiva de malhas 3D.

Em (Guéziec, 1999) é proposto um framework para minimizar o tempo de espera do envio do modelo completo com excelentes resultados, pois o objeto pode ser manipulado durante a transmissão do modelo evitando maiores atrasos.

O framework é composto de três partes:

1. um processo para gerar múltiplos níveis de detalhe
2. um processo responsável pela transmissão(preferencialmente utilizando compressão)
3. uma estrutura de dados para receber e explorar o LOD recebido gerado na primeira etapa e transmitido na segunda

Este framework, apesar de geral e independente, também foi implementado como um novo nó VRML (VRML, 1997) utilizando o mecanismo PROTO (VRML97, 1997b) para sua extensão.

Finalmente, é importante mencionar que o uso de algoritmos que alteram o nível de detalhe de forma dinâmica, principalmente no caso de terrenos, pode causar alguns problemas em jogos multi-jogador.

Sabendo-se que os algoritmos de LOD exibem detalhes de forma dependente da visão do usuário e, considerando que, obviamente é possível haver avatars de jogadores distribuídos em diversas posições do terreno, é possível então a ocorrência de alguns problemas de visualização, exemplificando:

- um jogador A pode julgar-se seguro num jogo de combate, ocultando-se atrás de uma elevação de terreno que ele, por estar próximo, vê com detalhes. Para outro jogador B distante, no entanto, o terreno não próximo apresenta-se com menos detalhes; muito possivelmente o trecho será representado por polígonos maiores com vértices com uma altura menor. Assim sendo, o jogador A, encontra-se, para este outro jogador B, total ou

parcialmente visível afetando a jogabilidade do ambiente 3D, pois ocorre uma vantagem para um jogador em detrimento de outro.

- um caso extremo também decorrente da dependência da posição do observador no algoritmo LOD é o fato de que um jogador pode aparecer para outros com os pés enterrados num vale do terreno ou mesmo flutuando sobre o topo de uma montanha; isto seria uma inconsistência visual crítica e desagradável influindo negativamente no realismo do ambiente.

2.2.4 Objetos procedimentais

Como pode ser facilmente observado, o nível de detalhes em jogos tem crescido muito nos últimos anos. Não é raro pois, encontrar jogos no mercado cujo pacote de distribuição contenha 2 ou mais CDs.

Por este fato, torna-se, cada vez mais comum, o uso de objetos procedimentais. Objetos, incluindo-se aí principalmente terrenos, têm sido fornecidos como plug-ins ou utilitários complementares de softwares como 3DS Max (3DSMAX, 2003), Lightwave (LightWaveWorld, 2003), TerraGen (Terragen, 2003) e MojoWorld (MojoWorld, 2003). Os fabricantes de jogos têm utilizado estes componentes para a geração do terreno e objetos da natureza em geral, seja na etapa de instalação ou mesmo em tempo de execução.

No caso de ambientes multi-jogador, a situação pode ser ainda mais crítica pelo fato de, a princípio, cada jogador poder ter um acervo individual e personalizado de objetos ou representações de seu avatar.

Obviamente, como estamos lidando com um universo virtual dinâmico, torna-se importante levar em conta de que modo os dados de um jogador que entra numa partida podem ser, de forma eficiente, distribuídos entre os demais jogadores que podem ser dezenas ou mesmo centenas num ambiente multi-jogador. Uma estratégia para lidar com esta potencial explosão de dados é o uso de objetos procedimentais em tempo de execução (O'Neil, 2003).

Há vantagens e desvantagens em se empregar objetos procedimentais gerados em tempo de execução. Para o caso de jogos *massive multiplayer* ou MMP as vantagens superam em muito as desvantagens que iremos citar. Uma observação a fazer é que as técnicas a serem descritas, são válidas mesmo quando a geração em tempo de execução se torna impraticável: - neste caso, elas ficam restritas à utilização numa etapa de inicialização do ambiente.

A primeira e óbvia vantagem de geração procedimental é o tamanho dos dados a serem gerados. Com uma única semente de números randômicos junto à disponibilidade de uma função geradora, torna-se possível a criação e reprodução de um mundo virtual completo. Desde que todos os jogadores tenham acesso a esta mesma função e a esta mesma semente, todos poderão gerar, localmente, por exemplo, um mesmo terreno composto por milhares de polígonos atendendo ao requisito de consistência visual de ambientes virtuais 3D.

O jogo Elite (Bell, 1984) foi um clássico exemplo de efetividade da geração procedimental. Era um ambiente gráfico combinando combate espacial e aventura formado por um universo virtualmente ilimitado, isto realizado numa plataforma com apenas 32Kb de memória.

Embora, como já dito, o hardware tenha evoluído consideravelmente desde então, da mesma forma ampliaram-se os requisitos de qualidade visual e a conseqüente necessidade de detalhamento dos objetos a serem criados. Estes objetos devem então ser gerados, seja numa etapa inicial ou mesmo durante a execução do jogo, armazenados e distribuídos de algum modo. Isto constitui portanto uma outra face das vantagens que é a diminuição dos custos da distribuição *online* (armazenamento, disponibilidade de servidores para múltiplos usuários, tempo de transferência etc.).

Indiretamente, há um benefício de marketing ao estimularmos a criação de novos jogadores ao fornecer um jogo com um tempo menor de *download* . Também, dependendo da implementação,

é possível que a geração do ambiente seja feita sob demanda, parte por parte, de modo a economizar os recursos de memória e capacidade de processamento gráfico do equipamento do jogador.

Uma desvantagem da geração em tempo de execução é a performance dos algoritmos empregados e sua complexidade o que pode limitar, no primeiro caso, o seu uso generalizado em computadores mais antigos e, no último caso, no custo de desenvolvimento destes softwares.

Outro problema é que a maioria dos algoritmos para geração procedimental de dados, como subrotinas baseadas em fractais, são difíceis de se controlar.

Há uma grande variedade de algoritmos para lidar com a geração de terrenos, mas, basicamente, há três grandes categorias: algoritmos estáticos, algoritmos de subdivisão e algoritmos dinâmicos.

Algoritmos estáticos

O que caracteriza esta classe de algoritmos para construção de terrenos, é o fato de ele utilizar uma malha gerada inicialmente com tamanho fixo (daí o nome) e, a partir disto, efetuar operações sobre toda esta malha. Um ponto positivo é a facilidade de se controlar a aparência básica da malha original que, posteriormente, é complementada com um algoritmo dinâmico.

Um bom exemplo de algoritmo estático é o algoritmo *fault line* (Elias, 2001). Este algoritmo parte de uma malha de tamanho fixo e a divide randomicamente em duas partes. No caso de um terreno 2D uma linha randômica irá fazer a divisão do terreno, já no caso de um terreno 3D, um plano randômico é gerado para dividir uma esfera (vide figura 9). Os valores de altura são somados de um lado e subtraídos do outro lado deste plano. Após milhares de iterações o terreno terá uma aparência realista.

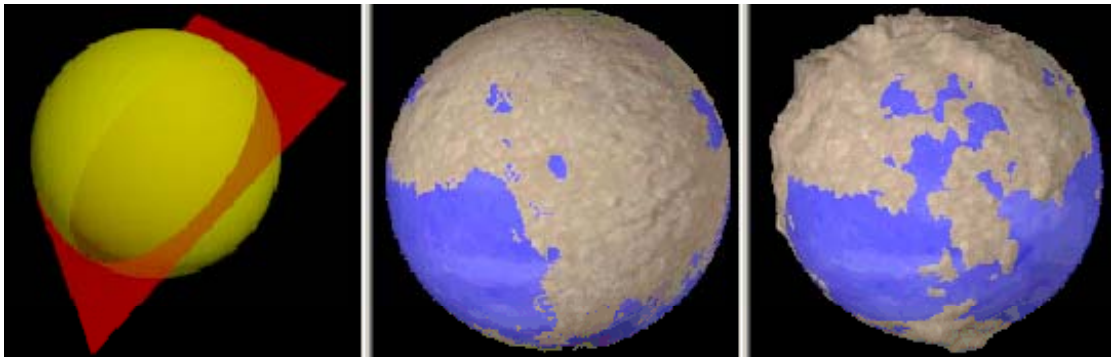

figura 9: algoritmo *fault line* (Elias, 2001)

Algoritmos de subdivisão

O algoritmo de subdivisão mais básico e mais comum é o chamado MPD (*mid point displacement*) ou simplesmente algoritmo do *plasma* (Terrain, 2003). Este algoritmo começa com um quadrado ou triângulo e então progride dividindo este polígono recursivamente. A cada passo, cada novo vértice tem sua altura elevada ou reduzida por um certo número randômico (figura 10). Também a cada nível da recursão, a variância destes vértices é reduzida por um fator que irá permitir o controle da rugosidade da superfície. A maioria das implementações usa um fator fixo, o que confere uma certa aparência monótona aos terrenos criados. É possível, no entanto, fazer o fator depender da posição do terreno em relação à origem das coordenadas para produzir regiões do terreno mais suaves que outras.

Algoritmos de subdivisão são essencialmente estáticos. Os pontos em cada nível da subdivisão devem ser gerados na mesma ordem para obtenção da mesma malha para uma dada semente randômica.

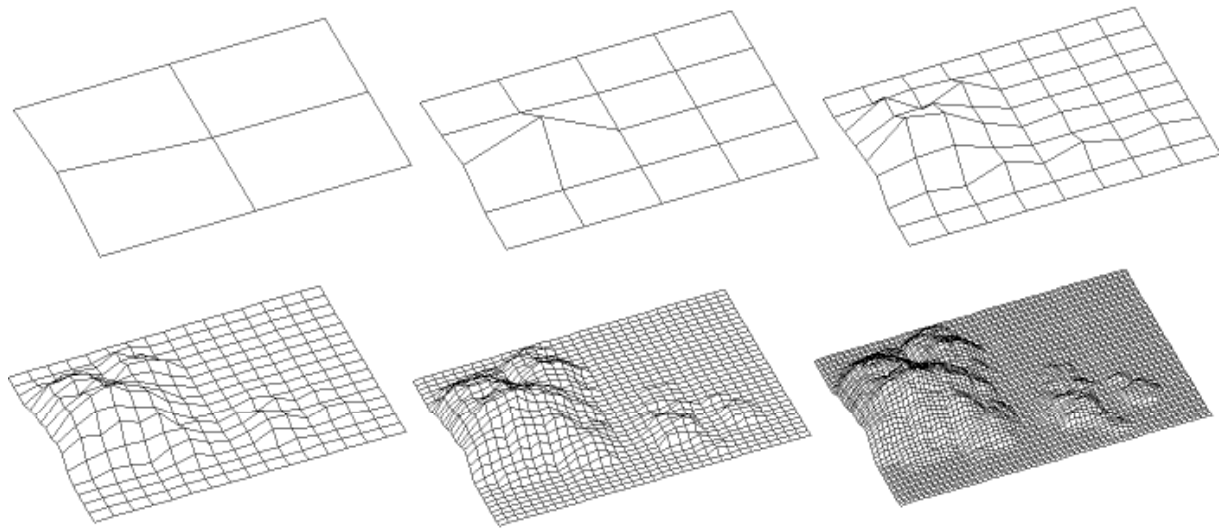

figura 10: algoritmo de subdivisão

Algoritmos dinâmicos

Algoritmos dinâmicos são aqueles que, dadas as coordenadas da malha a ser gerada, pode-se obter a altura da região sem que se tenha nenhum armazenamento prévio. A maioria dos algoritmos dinâmicos usados atualmente são baseados na técnica "*Perlin noise*" (Perlin, 1999). Esta técnica é tão popular que os fabricantes de placas gráficas 3D planejam implementá-la em adaptadores gráficos.

Perlin noise permite criar, rapidamente, ruídos randômicos suavizados usando uma tabela de números gerados aleatoriamente. Cada entrada na tabela corresponde ao valor inteiro de localizações no mapa e interpolações lineares são feitas para encontrar valores entre os pontos inteiros.

O mais conhecido algoritmo procedimental para geração de terrenos utilizando *Perlin noise* é chamado de fBm (*fractal Brownian motion*). A idéia é, dada uma função *noise*, gerar um somatório de termos como a seguir:
noise(p) + 1/2 * noise(2*p) + 1/4 * noise(4*p) + ... 1/n*noise(n*p) .

figura 11: efeito gerado utilizando, respectivamente, 3, 4 e 5 oitavas (O'Neil, 2003)

Cada um dos n termos é chamado de oitava, por sua analogia com o efeito produzido pela combinação de comprimentos de onda diferentes. Os efeitos do somatório podem ser vistos na figura 11.

O que é importante, e também possível de ser observado na figura, é que o aspecto da malha que pode ser gerada com esta equação varia, com cada termo adicionado, apenas na resolução de detalhes e não no aspecto geral do relevo o que é muito útil na modelagem do terreno.

Também importante é o fato de ocorrer uma rápida convergência, ou seja, tomando-se poucos termos (ou poucas oitavas), é possível gerar-se uma função que retorne a altura dada uma coordenada da região, caracterizando a categoria de objetos dinâmicos conforme mencionado.

2.3 Objetos customizáveis por meio de texturas

Dada a natureza social de jogos com um número elevado de jogadores, não é surpresa o interesse dos jogadores em imitar certos comportamentos da vida real. O mais expressivo exemplo disto é o desejo de se gerar personagens com aparência única para destacá-los da multidão.

Do ponto de vista do jogador, quanto maior o número de opções melhor. Já do ponto de vista do desenvolvedor, um maior número de escolhas geralmente implica em mais testes, mais peças de arte, mais elementos a serem compartilhados em rede, caso estas possibilidades de customização não possam ser definidas no momento de instalação.

Se, por um lado, é possível criarmos e compartilharmos personagens e objetos dinâmicos usando algumas das técnicas descritas até aqui, por outro temos uma interessante abordagem que é o uso de customização de aparências baseada na composição de texturas como proposta em (Hayes, 2003). Usando-se uma mesma malha poligonal e diferentes arquivos de texturas é possível gerar, apenas com as técnicas de composição de imagens, um número grande de personagens com diferentes aparências (figura 12).

figura 12: exemplos de arquivos de textura e geração de personagens diferentes por combinação (Hayes, 2003)

3. Técnica proposta

O uso de *stencil buffers* (O'Loughlin, 1999) na visualização interativa de objetos modelados por CSG e a produção de efeitos gráficos como proposto por (Clua, 2003) combinando recursos de CPU e da placa gráfica, aponta para a possibilidade de explorarmos um aumento da carga de processamento local dos PCs.

Desta forma, torna-se também possível propor que objetos customizados ou criados por um jogador sejam distribuídos de forma extremamente eficiente do ponto de vista de economia de banda de rede durante uma sessão em andamento. Ao invés de se fazer a transferência de uma malha poligonal já calculada sugere-se, sempre que possível, utilizar a transmissão da descrição concisa e analítica do objeto a ser sintetizado localmente e, em muitos casos, em tempo real no PC receptor.

Na verdade, o tempo real não é tão importante, mas sim um tempo que seja bem mais razoável do que aquele necessário para o envio de uma descrição poligonal completa.

Uma instância desta técnica é a construção de objetos a partir do envio de mensagens com a árvore de construção de objetos usando CSG. Outra variação da mesma técnica proposta é o envio da descrição de um objeto a ser modelado localmente utilizando metaball. Nos dois casos, temos que a descrição do objeto é concisa o suficiente para ser enviada pela rede.

Para que o usuário crie seus próprios objetos, quando for o caso, é necessário apenas que ele possa dispor de uma ferramenta para modelagem de objetos e que todos os demais disponham de um interpretador da string CSG ou metaball enviada, bem como das primitivas utilizadas em sua composição, o que é disponível na literatura.

O sistema base empregado é o mesmo descrito em (Kozovits, 2003). Foi utilizado Microsoft Visual C/C++ 6.0 para implementar o ambiente gráfico utilizando bibliotecas OpenGL-glut para fazer o *rendering* de objetos de cada sala. Os modelos utilizados têm a extensão MD2 e estão disponíveis na Internet, assim como bibliotecas para sua carga e *rendering*. Criou-se uma biblioteca integrada ao glut para encapsular as complexidades de uso do Microsoft DirectPlay responsável pelas conexões *peer-to-peer* entre os jogadores.

3.1 Formato de Mensagens para Distribuição de Objetos utilizando a técnica proposta

Representamos os números como strings hexadecimais, desta forma a mensagem será texto puro e cada nó terá tamanho fixo. Ex: 255 é representado como "000000FF"

Cada nó da árvore é composto de um identificador de sub-objeto SO (4 bytes totalizando 8 hexadecimais), e duas matrizes de coordenadas homogêneas MTx (4 linhas x 3 colunas x 4 bytes totalizando 96 hexadecimais) representando as transformações a serem aplicadas às sub-árvores da esquerda (MTL) e da direita (MTR), um código de operação OP (1 byte) e, finalmente, dois códigos apontadores para os sub-objetos componentes à esquerda L (4 bytes ou 8 hexadecimais) e à direita R (4 bytes ou 8 hexadecimais).

As operações são: U união, M diferença e I interseção

As primitivas empregadas neste exemplo são quatro, todas elas centradas na origem e têm Y como eixo de simetria:

1. a esfera de raio unitário, código hexa 00000001
2. o cubo unitário código, hexa 00000002
3. o cilindro de raio e altura unitários, código hexa 00000003
4. o cone de raio e altura unitários, código hexa 00000004

O tratamento de primitiva é o mesmo que o dado a sub-objetos, pois cada novo objeto criado torna-se uma primitiva para o sistema de modelagem permitindo a composição de objetos mais complexos. Os primeiros 255 números são reservados a primitivas fornecidas na instalação.

Cada objeto criado é registrado, sob número único (até 64k objetos), no servidor SQL do sistema na tabela "CSG" e pode ser consultado tanto pelo modelador (não implementado) quanto em tempo de execução pelos jogadores que recebem a mensagem de notificação de objeto criado.

Exemplos de mensagens correspondentes à montagem da figura 13:

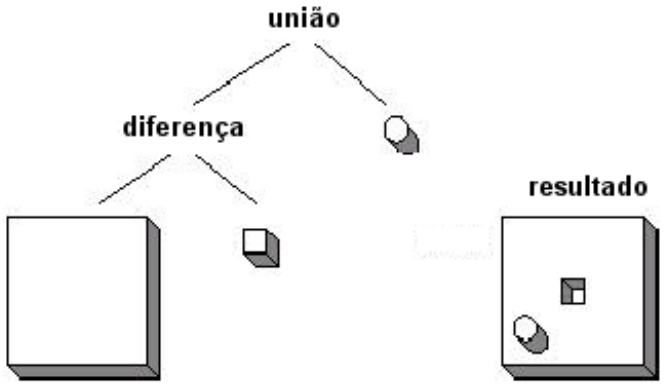

figura 13: exemplo de modelagem de objeto

registro no servidor SQL por meio da chamada de página ASPX (união de dois cubos)
st.aspx?cmd=$OP=U$L=00000001$R=00000001$MTL=96hex$$MTR=96hex$

OBS: cmd é uma string cujo parsing se dá no componente .NET escrito em CSharp; o caractere $ é o separador de cada parâmetro

valor de retorno esperado correspondente à criação do objeto 256
<ok>00000100</ok>

registro no servidor SQL (diferença entre objeto anterior e um cilindro)
st.aspx?cmd=$OP=M$L=00000100$R=00000003$MTL=96hex$$MTR=96hex$

valor de retorno esperado correspondente à criação do objeto 257
<ok>00000101</ok>

Exemplo de mensagem para outros jogadores para comunicação do novo objeto formado pelo código do objeto e a matriz de transformação:
<SO>00000101<MT>96hex</MT></SO>

Exemplo de consulta (operação Q) ao servidor SQL para obter a descrição do objeto 00000100 e, a seguir, objeto 00000101 caso já não esteja presente localmente.

st.aspx?cmd=$OP=Q$SO=00000101$

Resultado esperado:
<ok><OP>M</OP><L>00000100</L><R>00000003</R><MTL>96hex</MTL><MTR>96hex</MTR></ok>

st.aspx?cmd=$OP=Q$SO=00000100$

Resultado esperado:
<ok><OP>M</OP><L>00000002</L><R>00000002</R><MTL>96hex</MTL><MTR>96hex</MTR></ok>

4. Conclusão e trabalhos futuros

O uso da técnica descrita referente ao envio de árvores CSG é imediato, pois mesmo em PCs com pequeno poder de processamento, o máximo que pode ocorrer será restringir-se o uso da técnica proposta a objetos criados a partir de combinações bem simples de primitivas o que, ainda sim, é útil e mais eficiente do que o envio, pela rede, de malhas poligonais do objeto final.

Todavia, apesar de termos citado que conceitualmente, a técnica proposta aplica-se tanto a CSG quanto a Metaballs, será interessante observar uma análise da performance de metaballs em PCs (não implementado), bem como uma análise da performance da utilização das duas técnicas combinadas (BlobTrees descrita em (Wyvill, 1999)).

Qualquer outra representação utilizando fórmulas ou representações concisas em geral (ex.: envio de pontos de controle de splines, superfícies implícitas etc.) podem ser exploradas, pois o que se está fazendo é substituir uma representação próxima do *pipeline* gráfico por algo de mais alto nível e mais compacto. Nesta técnica, está-se aumentando a carga de processamento local considerando o fato de que, para o usuário de jogos, a rede ainda é o gargalo e não mais o desempenho de CPU e GPU dos atuais PCs.

É importante ressaltar que uma análise quantitativa da eficiência da técnica proposta é complexa e dependente da aplicação. Isto pode ser observado por meio de dois exemplos simples, a seguir, para ilustrar situações opostas:

- se desejamos compartilhar um objeto formado por apenas um paralelepípedo, é muito mais barato, em termos de largura de banda de rede, enviar os seis polígonos componentes em sua posição final, do que enviar uma estrutura CSG, que inclui matrizes para transformar a primitiva cubo unitário no objeto paralelepípedo desejado.
- por outro lado, se o objeto a ser representado for uma esfera ou elipse teremos a necessidade de enviar apenas o código da primitiva e uma matriz de transformação e não as dezenas ou milhares de polígonos para representar a esfera.

Um trabalho futuro é, portanto, um otimizador de representação para permitir a decisão automática da melhor forma de representação de um certo objeto, possibilitando seu envio, de modo ótimo, pela rede.

Seguindo a linha de envio pela rede das instruções de composição e não do objeto montado final, surge a possibilidade de aplicar-se a mesma idéia também para imagens, estendendo a idéia proposta em (Hayes, 2003), descrita anteriormente.

Por exemplo, a fotografia de um jogador, pode servir como textura da cabeça de seu avatar. Isto, principalmente num jogo *massive multiplayer*, é algo desejável pelo fato de o jogador desejar se destacar entre os demais. Embora o arquivo de imagem possa não ser muito grande, é algo que tem que ser enviado a dezenas ou centenas de outros jogadores.

É conhecida a técnica de retrato falado que possibilita construir a imagem de um rosto e é utilizada pela polícia para encontrar criminosos.

Esta técnica, se aplicada de forma semelhante à idéia de representação concisa, à partir de primitivas, descrita neste trabalho, pode não apenas servir para economia dos recursos de rede, mas também permitir uma busca automatizada de jogadores.

A partir de um banco de imagens, componentes de formatos de olhos, queixo, nariz, formato de cabeça e, eventualmente, características de todo o corpo como altura, cor de pele etc., é possível enviar apenas os códigos correspondentes ao banco de imagens que seriam distribuídos a todos os jogadores no momento da instalação do ambiente.

Pode-se assim, localmente, montar a textura (e todo o modelo) correspondente ao personagem que se deseja distribuir. Para exemplificar o funcionamento da tecnologia de retrato falado digital correspondente ao grupo étnico brasileiro, consulte o endereço e o programa disponível em (TslBrasil, 2003).

5. Referências

3DS MAX website: "http://www.discreet.com/products/3dsmax/", 2003

Baumgart, B. "*A Polyhedron Representation for Computer Vision*", AFIPS Conference Proceedings, pp 589-596, 1975.

Beardsley, Jason, *Seamless Servers: The Case For and Against*. In Thor Alexander, ed., Massively Multiplayer Game Develoment, Charles River Media, pp. 213-227, 2003.

Bell, Ian, "The Elite Home Page", disponível em http://www.iancgbell.clara.net/elite/, 1994

Blinn, Jim, "A Generalization of Algebraic Surface Drawing", ACM Transactions on Graphics, Vol. 1, No. 3, pp. 235-256, July, 1982.

Clua, Esteban W. G., "Utilization of Nailboards for optimization in rendering distribution between CPU / GPU using Relief Texture Mapping". Monografias em Ciência da Computação, Departamento de Informática, PUC-Rio, 2003 (em preparação).

Damon, William. *Impostors Made Easy*. Intel Technical Report., http://cedar.intel.com, 2003.

Elias, Hugo, "Spherical Landscapes", disponível online em http://freespace.virgin.net/hugo.elias/models/m_landsp.htm, 2001.

Finkel, R.A., et Bentley, J.L., "*Quad Trees: A Data Structure for Retrieval of Composite Keys*," Acta Informatica, Vol. 4, No. 1, pp. 1-9, 1974

Forsyth, Tom. *Impostors :Adding Clutter*. In Mark DeLoura, ed., Game Programming Gems 2, Charles River Media, pp. 488-496, 2001.

H. Fuchs, Z. M. Kedem and B. F. Naylor. "*On Visible Surface Generation by A Priori Tree Structures*." ACM Computer Graphics, pp 124–133. July 1980.

Garland M., "*Multiresolution Modeling: Survey & future opportunities*." Eurographics '99, State of the Art Report Disponível em http://graphics.cs.uiuc.edu/~garland/papers/STAR99.pdf , September 1999.

Goldstein, R.A., and Nagel, R., "3-D Visual Simulation", Simulation, pp. 25-31, January 1971.

Guéziec, André et al. "A Framework for Streaming Geometry in VRML", IEEE COMPUTER GRAPHICS (Vol. 19, No. 2) pp. 68-78. Disponível em http://researchweb.watson.ibm.com/people/t/taubin/pdfs/Gueziec-etal-cga99.pdf, March/April 1999

Hayes, Todd, *"Texture-Based 3D Character Customization"*. In Thor Alexander, ed., Massively Multiplayer Game Develoment, Charles River Media, pp. 341-354, 2003.

Kozovits, Lauro E., "Arquiteturas para Jogos *Massive Multiplayer*". Monografias em Ciência da Computação, Departamento de Informática, PUC-Rio, 2003 (em preparação).

LightWaveWorld website: http://www.lightwaveworld.com/, 2003

Lorensen, William E. and Cline, Harvey E. "*Marching Cubes: A High Resolution 3D Surface Construction Algorithm*", ACM SIGGRAPH Proceedings, pp.163–169, 1987.

Meagher, D., "*Octree Encoding: A New technique for the Representation, Manipulation and Display of Arbitrary Three-Dimensional Objects by Computer*", Technical Report IPL-TR-80, 111, Image Processing Laboratory, Rensselaer Polytechnic Institute, Troy, New York, October 1980.

MojoWorld Generator, Pandromeda's website: http://www.pandromeda.com/, 2003

MPEG-4, ISO/IEC 14496-2 Visual Working Draft Version 2 Rev. 5.0, SC29/WG11 document number W2473, 16 Oct.1998.

O'Loughlin, J and O'Sullivan, C., "Real-Time Animation of Objects Modelled using Constructive Solid Geometry", Technical Report, Trinity College Dublin's Computer Science Dept., www.cs.tcd.ie/publications/tech-reports/reports.99/TCD-CS-1999-09.pdf, 1999

O'Neil, Sean, *Procedural Worlds: Avoiding the Data Explosion*. In Thor Alexander, ed., Massively Multiplayer Game Develoment, Charles River Media, pp. 314-332, 2003.

Perlin, Ken, "Making Noise", tutorial disponível online em http://www.noisemachine.com/talk1/, 1999

Pope, A., "*The SIMNET network and protocols*", Technical Report 7102, BBN Systems and Technologies, Cambridge, MA, 1989

Roth S. D. , "*Ray Casting for Modelling Solids*", Computer Graphics and Image Processing, Feb., vol 18, pp. 109-144, 1982

Schaufler, G. *Dynamically Generated Impostors*. GI Workshop on Modeling – Virtual Worlds – Distributed Graphics, D. W. Fellner, ed., Infix Verlag, pp. 129-135, November 1995.

Schaufler, G. *Nailboards: A Rendering Primitive for Image Caching in Dynamic Scenes. Proceedings of the 8th Eurographics Workshop on Rendering*. St. Ettiene, France, June 16-18, 1997. *Rendering Techniques '97*, Springer-Verlag, pp. 151-162, June 16-18, 1997.

Shankar, Krishnan, "*Efficient and Accurate Boundary Evaluation Algorithms for Sculptured Solids*", PhD Thesis, University of North Carolina at Chapel Hill, 1997

Teller, S. J. and Sequin, C. H., "*Visibility Pre-processing for Interactive Walkthroughs*." ACM Computer Graphics, pp 61–69. July 1991.

Teller, S. J., "*Visibility Computations in Densely Occluded Polyhedral Environments.*", PhD Thesis. University of California at Berkeley. 1992.

Terragen™ photorealistic scenery rendering software website: http://www.planetside.co.uk/terragen/, 2003

Terrain Tutorial - Mid Point Displacement, disponível online em: http://www.lighthouse3d.com/opengl/terrain/index.php3?mpd, 2003

Tlslbrasil website: http://www.tslbrasil.com.br/downloads.htm, 2003

Ulrich, Thatcher, "Continuous LOD Terrain Meshing Using Adaptive Quadtrees", gamasutra.com, disponível em http://www.gamasutra.com/features/20000228/ulrich_pfv.htm, 28 feb. 2000.

VRML97 The Virtual Reality Modeling Language, International Standard ISO/IEC 14772-1:1997 disponível em http://www.web3d.org/technicalinfo/specifications/vrml97/index.htm, 1997.

VRML97 The Annotated VRML 97 Reference, Prototype semantics, disponível em http://www.web3d.org/resources/vrml_ref_manual/ch2-28.htm, 1997b.

Wiegand, T.F., “Interactive Rendering of CSG Models", Computer Graphics Forum, Volume 15 No. 4, PP. 249-261, 1996.

Wyvill, G. et al., "Data Structure for Soft Objects", The Visual Computer, Vol. 2, pp. 227-234, 1986.

Wyvill, Brian et al., "Extending The CSG Tree. Warping, Blending and Boolean Operations in an Implicit Surface Modeling System", Computer Graphics Forum, vol 18, No 2, PP.149-158, (http://pages.cpsc.ucalgary.ca/~blob/ps/blobtree.pdf), Jun, 1999.

Yoshizu, k., & Nishita, T., "Metaball Editor", disponível em http://nis-lab.is.s.u-tokyo.ac.jp/~nis/javaexampl/MetaJava/EmetaJava.htm, 1997