Refração e Reflexo para Sistemas de Visualização em Tempo Real Utilizando Portais

Esteban W. Gonzalez Clua
Computer Science Department, PUC-Rio
esteban@inf.puc-rio.br

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

Waldemar Celes
Computer Science Department, PUC-Rio
celes@inf.puc-rio.br

PUC-RioInf.MCC27/03 August, 2003

Abstract: This work presents an efficient method for simulating refraction and reflection phenomena for flat surfaces in real time, using portals and texture mapping. The method is easily extended to encompass more complex objects. This paper also claims that the combination of the proposed method with multi-texturing leads to a complete lighting framework for real-time rendering.

Keywords: real-time rendering, portals, lighting model, texture mapping, refraction, reflection.

Resumo: O trabalho apresentado descreve uma eficiente forma de obter o fenômeno de transparência com refração para aplicações que exigem visualização em tempo real, utilizando para isto o algoritmo de portais modificado. Este método, somado ao algoritmo de portais para objetos espelhados consiste numa tentativa de obter efeitos de rendering similares ao ray-tracing, em tempo real, para cenas com um número limitado de portais.

Palavras-chave: Visualização em tempo real, portais, modelo de iluminação, mapeamento de textura, refração, reflexo.

This work is being sponsored by CNPq and ICAD.

1 Introdução

O avanço dos dispositivos de aceleração gráfica se deve especialmente ao poder de manipular um grande número de polígonos e texturas, com bastante velocidade. Isto permite que cenas cada vez maiores possam ser visualizadas em tempo real, possibilitando a modelagem de ambientes mais complexos e realistas. Atualmente estes dispositivos permitem também que parte do pipeline seja programável, podendo customizar alguns modelos de iluminação que antes não eram possíveis. Entretanto, esta programação possibilita que apenas sejam manipulados vértices da malha 3D ou pixels do resultado da projeção e rasterização. Assim sendo, alguns modelos físicos não podem ser tratados com precisão por esta recurso, como é o caso de reflexo, transparência e caustics, uma vez que um cálculo preciso destes fenômenos exige que se trate cada ponto da superfície individualmente.

Assim sendo, tradicionalmente associa-se ao fenômeno de transparência e refração aos algoritmos de Ray-tracing [Glassner 89]. De fato, este algoritmo permite que de forma trivial se aplique a lei de Snell para um raio que está passando de um meio transmissivo para outro, assim como o lançamento de outros raios recursivos para visualizar o que está por trás de um objeto. De fato, neste caso clássico, a situação difusa não pode ser considerada (Figura 1). Métodos de iluminação global podem gerar imagens extremamente realistas, mas são totalmente inadequados para se obter resultados em tempo real. Métodos para aproximação empírica da refração foram recentemente propostos, mas os resultados são questionáveis [Vlachos 01]. Outros trabalhos chegam a propor soluções para reflexo e refração em tempo real [Möller and Haines 99], sem no entanto chegar a implementá-los para o caso de refração.

Embora o poder de processamento gráfico tenha crescido bastante, em nenhum momento o ray-tracing pretende servir como algoritmo de visualização para tempo real, já que o seu pipeline trata individualmente a iluminação para cada ponto de uma superfície sendo projetado num pixel da imagem que se está visualizando, além de criar uma árvore recursiva para reflexos e transmissões múltiplas, que em alguns casos pode vir a ser enorme.

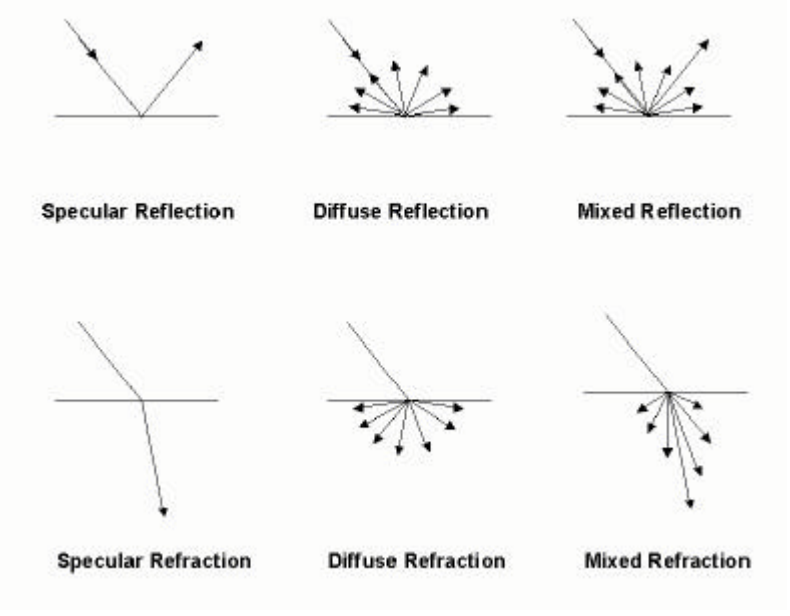


Figura 1 – Tipos de reflexo e refração (este artigo considera apenas o caso de especular)

Por outro lado, têm se tornado um recurso cada vez mais comum para as placas gráficas o tratamento de render passes [Peercy et al. 00, Heidrich et al. 99], o que permite que sejam feitas várias camadas de rendering, ou seja, um objeto é renderizado várias vezes, cada um com alguma propriedade de material. Assim sendo, pode-se renderizar um objeto com uma textura, a seguir

renderizá-lo com um enviroment-map e a seguir com outra textura, fazendo posteriormente uma junção adequada dos resultados obtidos, através de alguma função de *blending*. Este recurso permite que o objeto ganhe bastante realismo mesmo tendo sido gerado em tempo real.

Neste artigo propõem-se uma nova técnica, capaz de simular com o realismo necessário para aplicações de realidade virtual e jogos 3D, o efeito de reflexo especular somado ao efeito de transparência com refração, para superfícies planas, em tempo real, sem a necessidade de recorrer ao algoritmo do ray-tracing. A idéia básica deste método consiste primeiramente em transformar a superfície plana num portal e mover a posição da camera para o ponto por onde olhando a cena pelo portal corresponderia à imagem correta da refração. A mesma coisa é feita para o caso do reflexo, conforme se descreve em [Luebke et al. 96]. As duas imagens geradas são armazenadas como texturas e aplicadas sobre o portal com a utilização de *multi-passes*.

Este artigo esta organizado da seguinte forma: o capítulo 2 apresenta o recurso de *multi-texturing* como uma forma de simular com realismo efeitos para tempo real. O capítulo 3 descreve o tradicional recurso de portais, bem como aplicações semelhantes que podem servir como referências para integrações com trabalhos futuros. O capítulo 4 apresenta o método proposto pelo artigo em questão, seguido de um capítulo sobre possíveis otimizações e testes realizados. Finalmente descrevem-se alguns apontamentos sobre trabalhos futuros e a conclusão deste trabalho.

2 Multi-texturing

Em rendering por *multi-passes*, as diversas partes da equação do modelo de iluminação sendo utilizada são calculadas separadamente. Os resultados podem ir sendo compostos a cada *pass* gerado, somando-se ao resultado acumulado previamente, ou podem ir sendo armazenados para serem enviados juntos em alguma etapa do pipeline gráfico. Rendering por *multi-passes* tem sido apontado como um possível modelo de iluminação completo para inúmeras aplicações de visualização em tempo real [Watt & Policarpo 01]. Entretanto, um método mais geral pode ser conseguido, estendendo o modelo de *multi-passes* simples para um modelo de *multi-pass* com *multi-texturing*. Neste caso, modelos complexos podem ser definidos por expressões semelhantes à que se descreve em Eq. 1.

$$C = C_1 \otimes C_2 \otimes \cdots \otimes C_k + C_{k+1} \otimes C_{k+2} \cdots + \cdots \qquad (1)$$

Onde cada variável C_i (uma camada) representa uma textura específica, $\otimes$ é uma operação de combinação de *multi-texturing* e + é uma operação de *multi-pass*.

Atualmente, grande parte dos aceleradores gráficos suportam 4 ou 8 camadas de *multi-texturing* para uma única superfície (k = 4 ou 8). Desta forma, um modelo de iluminação com 4 texturas (Eq.2) é eficientemente calculado numa placa que suporte 4 operações de *multi-texturing*.

$$C = C_1 \otimes C_2 \otimes C_3 \otimes C_4 \qquad (2)$$

Se o número de camadas de texturas superar o número suportado pelo hardware, deve-se realizar utilizar uma operação de *multi-pass* (Eq. 3).

$$C = C_1 \otimes C_2 + C_3 \otimes C_4 \qquad (3)$$

Deve-se notar que um algoritmo de *multi-pass* não pode calcular a Eq.1 por si só, no caso da ordem da combinação afetar o resultado final:

$$(C_1 \otimes C_2 + C_3 \otimes C_4) \neq (C_1 + C_2 + C_3 + C_4) \qquad (4)$$

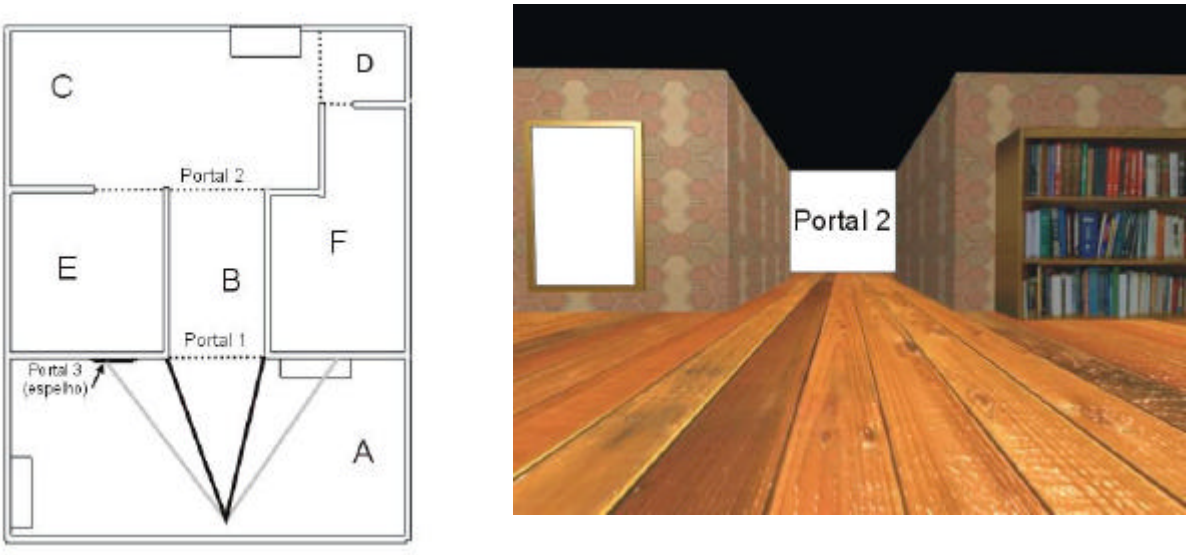
Assim, na Eq.3, se não for utilizado *multi-texturing*, o cálculo de *multi-pass* irá combinar C_1 e C_2 em dois *passes* e C_3 no *pass* seguinte, deixando C_4 sozinho.

No método proposto neste artigo, a imagem gerada pelo algoritmo de portais é transformada numa textura que será mapeada numa superfície plana. Desta forma, a combinação do método proposto com o conceito de *multi-texturing* mencionado anteriormente, cria um poderoso *framework* para visualização em tempo real completo e eficiente.

A idéia de combinar o uso de *multi-texturing* com *multi-passes* na visualização é sugerida por uma comunicação pessoal de Jason Mitchell apontada em [Möller and Haines 99]. Este trabalho estende sua idéia tendo em vista a tentativa de utilizá-la dentro de um contexto de um framework para iluminação. A possibilidade de mesclar reflexo, refração e outros tipos de efeitos através de texturas reforça a importância do uso de *multi-texturing*. A proposta de usar *multi-texturing* combinado com algoritmo de portais para criar um modelo de iluminação não pode ser encontrado na literatura.

3 Trabalhos Relacionados

O método descrito neste trabalho lança mão do algoritmo de portais [Airey et al. 90, Teller et al. 91, Eberly 00]. Os portais são um eficiente método para culling para ambientes que podem ser divididos em várias células. Este método procura otimizar a visualização fazendo com que apenas as células visíveis num determinado instante sejam processadas. A idéia básica dos portais consiste em substituir parte da geometria de uma cena por um polígono especial (portal). Neste polígono será mapeada a imagem resultante da visualização de uma camera posicionada no mesmo local do observador inicial, mas com um view frustrum menor que o original e limitada pelo próprio polígono corretamente clipado (Fig. 2). O algoritmo de simulação de reflexo e refração que será descrito lança mão desta idéia, acrescentando, entretanto, uma transformação de posição sobre a camera original.



(a) (b)

Figura 2 – Em (a) pode-se ver um mapa de um ambiente fechado. Cada célula está indicada por uma letra maiúscula e os portais estão representados com uma linha tracejada. Em (b) pode-se ver o resultado parcial da visualização da cena: o portal 1 já está sendo mostrado e os portais 2 e 3 ainda estão em branco.

O algoritmo de portais inicia-se dividindo a cena em células convexas, o que garante a propriedade de que um observador posicionado em qualquer ponto do seu interior pode virtualmente ver qualquer outra parte da mesma (não se está considerando a existência de objetos oclusivos, mas que por não pertencerem a árvore BSP da cena não irão influenciar na visualização). Outras células poderão ser vistas apenas através de polígonos convexos especiais, que serão definidos como portais (figura. 3). Assim, diz-se que todas as células estão separadas por polígonos normais (o que impede que o observador veja a célula adjacente) ou por portais. Diz-se que um portal é bidirecional, quando o mesmo existe nas duas células adjacentes. Na realidade, o portal é composto por 2 polígonos convexos, posicionados no mesmo local. Diferem, no entanto, porque cada um aponta para uma célula diferente. Para a representação de células não convexas, pode-se criar paredes invisíveis, onde os portais funcionam como estas paredes.

Os portais serão implementados através da construção de um grafo direcionado, onde as células são os nós e os portais são os *directed graph edges* (figura 3).

O pipeline de rendering utilizando portais deve garantir que os polígonos sejam plotados em *back to front order*, de maneira a garantir a ordem correta de polígonos visíveis. A visualização pode ser resumido pelo seguinte algoritmo:

Faça o Clipping e Culling da sala onde se encontra o observador
Renderize a cena utilizando o obervador na sua posição original
Se um polígono da cena é um portal então
Recalcule o view-frustrum do observador
Renderize a sala que está sendo vista pelo portal
Chame recursivamente o algoritmo de visualização

Em [McReynolds et al. 98] apresenta-se um algoritmo para criar superfícies planas reflexivas utilizando texturas, mas não apontam o uso de portais para tanto. Em [Watt et al. 00] os autores descrevem o uso de portais para criar espelhos e David P. Luebke and Chris Georges entram em detalhes sobre o problema dos objetos que estão atrás dos espelhos implementados por portais, sem no entanto entrar em detalhes de implementação [Luebke et al. 96].

Tomas Möller e Eric Haines [Möller and Haines 99] apresentam a idéia de utilizar diferentes modos de visualização quando portais são atravessados. Propõem usar transformações na posição do observador para criar efeitos especiais em tempo real, tais como reflexo e refração, como se apresenta neste artigo. Entretanto nenhum detalhamento sobre este tratamento é realizado pelos autores.

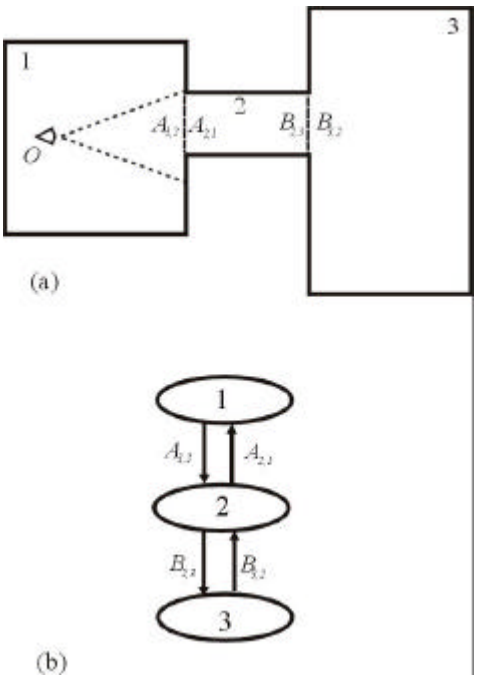


Figura 3 - (a) Exemplo de um conjunto de células e seus respectivos portais bidirecionais. (b) grafo direcionado que representa a estrutura do exemplo da figura.

Em [Aliaga et al. 97] apresenta-se uma possível otimização para os portais, fazendo com que o mesmo corresponda a uma textura pré-calculada. Quando o observador se aproxima do portal, este passa a ser tratado como geometria, utilizando-se um algoritmo de warping para a transição. Com esta idéia torna-se necessário calcular a visualização apenas da célula onde o observador se encontra, uma vez que as demais células serão tratadas como texturas. Se por um lado este método traz uma grande aceleração, por outro lado traz problemas relacionados a falta de sensação espacial, já que as imagens das células anexas ficam estáticas. Em [Rafferty et al. 98a] o autor procura solucionar este problema sugerindo que cada portal possua um conjunto de imagens pré-calculadas, com vistas da célula anexa à partir de ângulos diferentes (Fig. 4). Dependendo de onde o observador se encontre, escolhe-se a vista que lhe é mais adequada. Este método aumenta a sensação de imersão, mas ainda gera transições descontínuas ao intercalar as imagens. Neste mesmo artigo, assim como em [Rafferty et al. 98b], apresenta-se como uma possível solução para este problema a utilização de warping de imagens 3D, baseado na equação de warping de McMillan [McMillan et al. 95, McMillan 97]. Desta maneira, além de eliminar o efeito de transição descontínua, possibilita-se que haja um número menor de imagens para cada portal. Surgem, no entanto, alguns problemas relacionados a falta de informação para algumas posições em que o observador se encontra (buracos). Em [Popescu et al. 98] apresenta-se uma solução para este mesmo problema utilizando a técnica de Layered Depth Images, que consiste basicamente numa imagem com várias camadas de cores para cada pixel [Gortler et al. 97, Max et al. 95]. Além disto, neste artigo também se apresenta um elegante algoritmo para paralelizar o warping das imagens dos portais. Como será exposto no capítulo 5 deste artigo, todos estes métodos podem ser utilizados para otimizar os cálculos de portais de reflexo e refração.

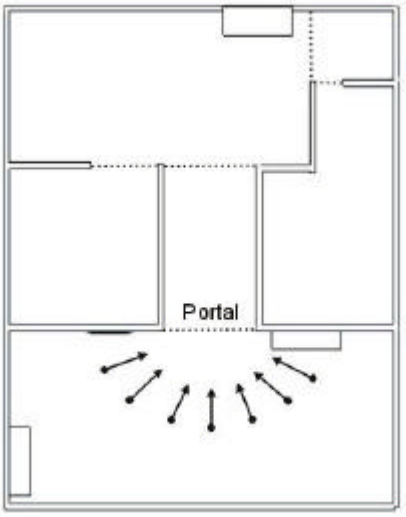


Figura 4 – Em [Aliaga et al. 97] sugere-se que várias possíveis vistas de um portal sejam pré-renderizadas e armazenadas na forma de textura. Dependendo da posição do observador, escolhe-se a imagem mais adequada desde conjunto.

Em [Hurley 00] o autor descreve um método para obter efeito de refração em tempo real por um caminho diferente que o abordado neste trabalho. O autor utiliza uma técnica semelhante aos enviroment mappings [Blinn et al. 76], utilizando a normal de cada polígono para determinar a direção do ponto de um enviroment cube que está atrás do objeto e que será utilizado na composição do material. Este método tem como inconveniente o fato de que se deve fazer um pré-processamento das normais de um objeto. Este cálculo deverá ser refeito sempre que houverem mudanças dos objetos posicionados na cena, uma vez que será necessário recalcular o enviroment cube. Além disso, este método requer um hardware gráfico que tenha suporte para fragment programming para manter coerência com o tempo real.

4 Portais aplicados à reflexo, transmissão e refração sobre um objeto plano

Como foi mencionado anteriormente, cada portal pode ter seu próprio modo de visualização. No método proposto, superfícies espelhadas e refrativas são definidos como portais planos e retangulares com um modo de visualização específico associado a si. Este modo de visualização possui associado um conjunto de transformações *T* para o observador *F*,

$$F' = T(F) \qquad (5)$$

De forma que o vetor da posição do observador (O) até o centro do portal (P) é rotacionado em torno de P para a mesma direção do raio de luz propagado (V_2). A situação é dada por:

$$|P - O| = |P - O'| = h$$

$$\vec{V}_d = \frac{P - O}{h}$$
$$\vec{V}'_d = \frac{P - O'}{h}$$
$$\vec{V}_d = -\vec{V}_1$$
$$\vec{V}'_d = \vec{V}_2$$

(6)

onde V_d, V'_d, V_1, e V_2 são vetores unitários.

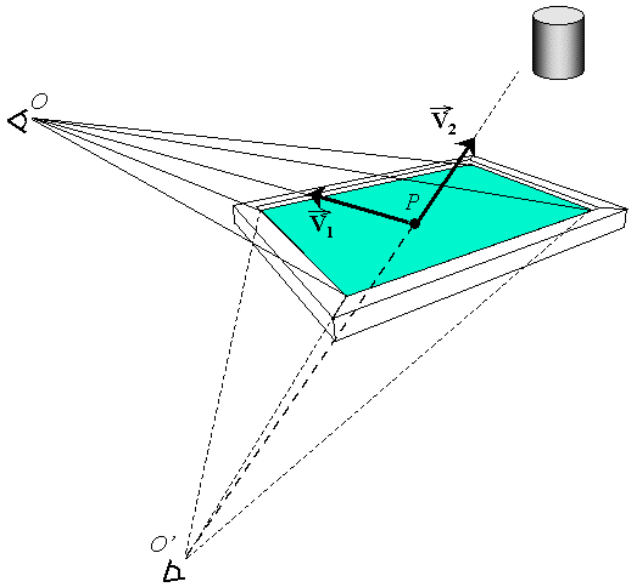


Figura 5 – Neste caso, a imagem a ser mapeada no portal de refração corresponde ao observador refletido.

O observador *F* é representado pelo *frustrum* que parte do olho e vai em direção ao portal de refração / reflexo. Na prática, o observador *F* é representado por uma camera definida pela tupla

$$\left\langle O, \vec{V}_d, \vec{V}_{up}, FOV \right\rangle$$

(7)

onde O é a posição do observador, V_d é o vetor de direção da camera direction vector, V_{up} é o vetor que indica a onde está a direção vertical da camera (*up vector*) e *FOV* = é o ângulo de abertura da camera.

Para o portal de reflexo, a transformação é

$$F' = M(F) \qquad (8)$$

Onde *M* é a matriz de espelhamento aplicada ao observador (i.e., aos parâmetros da camera), como está ilustrado pela figura 5.

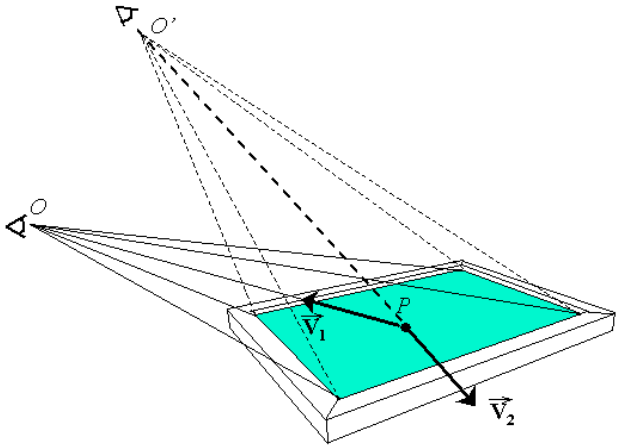


Figura 6 – A imagem a ser mapeada no portal de refração corresponde a um observador com o vetor de direção equivalente ao do vetor do raio refratado.

Para o portal de refração, a transformação *T* é a rotação determinada por V_2 e de acordo com a Lei de Snell (figura 6). Com a camera nesta posição, gera-se uma imagem com o novo ponto de vista e com o frustum definido pelo contorno do polígono do portal. Caso o portal não seja retangular, será necessário criar um bounding-square, que servirá como frustum da camera.

O cálculo da posição de camera a ser utilizado pelo portal de refração inicia-se determinando o ponto *P*, correspondente à intercessão do vetor de direção da camera V_i com o plano definido pelo portal (figura 6). Este cálculo deverá ser efetuado para cada portal que compõe um objeto.

Observando-se a figura 6, pode-se ver que:

$$\vec{V}_{ref} = \vec{N}'' + \vec{S}'' \tag{9}$$

Pela lei de Snell chega-se que:

$$\vec{V}_{ref} = n\left\|\vec{V}_i\right\| + (n\cos q_1 - n\cos q_2)\,\vec{N} \tag{10}$$

Onde

$$\cos q_2 = \sqrt{1 - \left(\frac{n_1}{n_2}\operatorname{sen} q_1\right)^2} \tag{11}$$

e

$$n = \frac{n_1}{n_2} \tag{12}$$

Uma vez determinado o V_{ref} deve-se utilizar este vetor para determinar a nova posição da camera a ser utilizada pelo portal. Isto será feito pela equação 13:

$$O' = -|V_i|\,V_{ref} + P$$

O resultado desta operação pode ser visto também como uma rotação do vetor de incidência em relação ao ponto de intercessão com o portal e pode-se interpretar como o processo inverso do fenômeno da refração: ao invés do raio de luz ser alterado, como acontece no ray-tracing, o observador é que passa a ser alterado.

Ao criar objetos refrativos mais complexos, cada polígono que o representa deverá ser definido por um portal bidirecional: um apontado para o interior do objeto e outro para o seu lado externo.

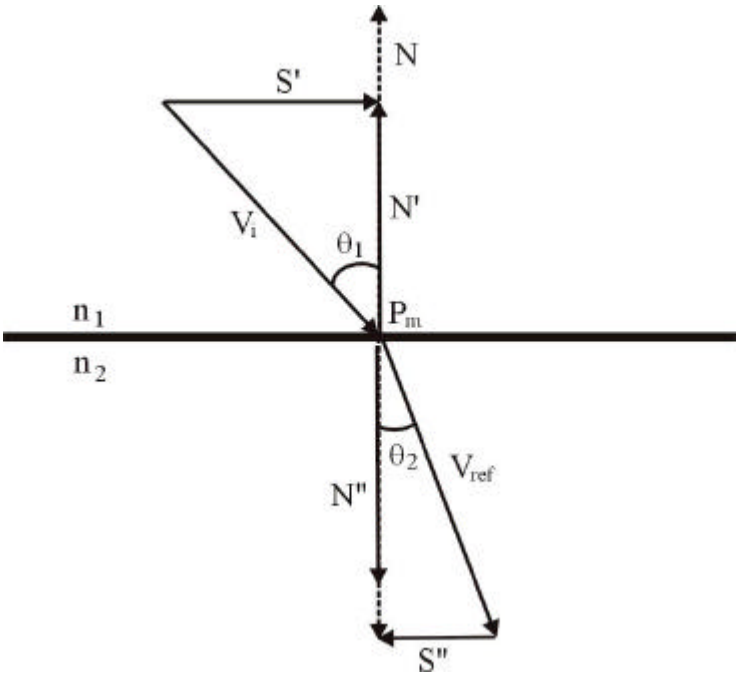


Figura 7 - Para obter a nova posição da camera, é necessário antes determinar o vetor de refração para o raio de incidência sobre o portal. Isto será feito através da lei de Snell.

Ao gerar a imagem a partir do ponto de vista da camera deslocada, o resultado será um quadrilátero, que deverá ser mapeado sobre outro quadrilátero, que é a projeção do portal visto pelo observador na posição original. Esta projeção será feita mapeando-se cada coordenada dos vértices do portal refratado para os vértices correspondentes do portal original.

Na implementação da visualização utilizou-se o OpenGL. Nesta API, a interpolação de coordenadas de texturas é feita linearmente, o que traz uma deformação inconveniente no resultado, conforme mostra a figura 8.

De forma a corrigir esta deformação, é necessário criar coordenadas de textura intermediárias para ambos os polígonos. Quanto maior for a deformação perspectiva, maior será a incorretude da projeção da imagem no portal, e portanto devem haver mais coordenadas de texturas. Com o aumento das coordenadas de texturas o problema ainda persiste, pois a deformação continuará ocorrendo entre uma coordenada e outra. Entretanto, este problema irá tornando-se cada vez mais imperceptível.

Outro problema que poderá ocorrer é o aparecimento de objetos entre a camera deslocada e o portal de refração. Como a camera é movida para outra posição da cena, pode ocorrer de que um objeto que antes não era visto pelo observador passe a obstruir o campo de visão. Na imagem a ser gerada e mapeada sobre o portal este objeto não pode aparecer, pois não está atrás do objeto transmissor. Para solucionar isto deve-se criar um clipping plane que coincide com o plano do portal e calcular a visualização apenas para os polígonos que estejam depois deste plano.

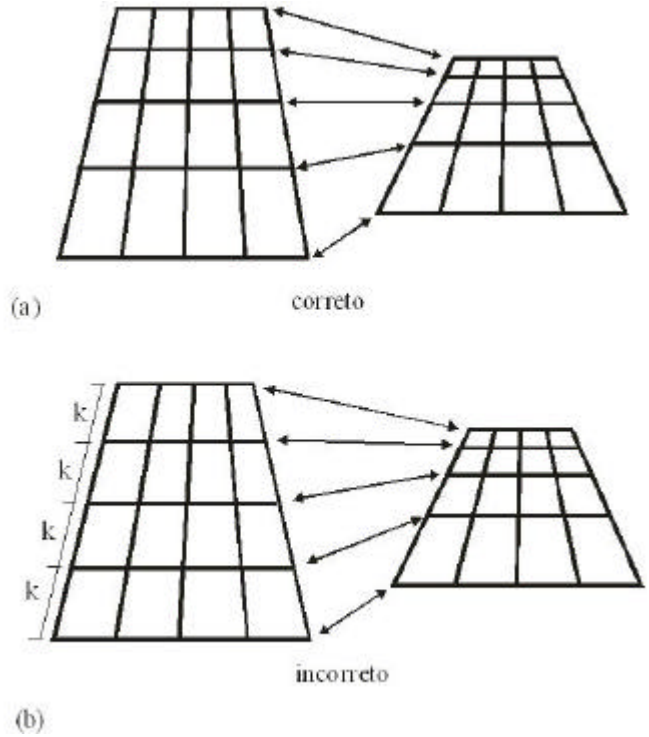


Figura 8 - A malha da esquerda representa o portal visto pela camera deslocada pela lei de Snell e a malha da direita o portal visto pela camera na posição original. Em (a) o mapeamento está correto (perceba-se a não-linearidade que ocorre entre os diversos vértices auxiliares). Em (b) o mapeamento está incorreto (a linearidade pode ser evidenciada pelo fato de cada vértice tem uma distância constante k.

5 Otimização

É conveniente para os portais reflexivos e refrativos que haja um valor de alcance máximo para seu cálculo, de forma que objetos distantes do observador dispensem a utilização destes portais. Para esta otimização pode-se substituir o portal por um polígono com uma textura (pode ser uma figura equivalente a última imagem gerada para este portal) ou simplesmente torná-lo transparente, sem fenômeno de refração.

Irá influenciar também na performance a resolução da imagem gerada pela camera deslocada. Caso o portal represente uma área grande, esta resolução deverá ser relativamente grande. Entretanto, ao representar-se objetos compostos por vários portais, estes serão relativamente pequenos em relação à cena e portanto podem gerar imagens numa resolução baixa.

Outra possível otimização consiste em acumular o resultado do cálculo de um portal num cachê, na forma de textura [Aliaga et al. 97]. Sempre que o observador se movimentar menos do que um determinado valor limite, ao invés de ser calculado novamente a cena refratada ou refletida, pode-se simplesmente usar a imagem que fora utilizada anteriormente. Para que a transição de uma imagem antiga para uma nova não tenha saltos de descontinuidade, pode-se realizar uma transição simples entre uma e outra, utilizando por exemplo uma interpolação linear entre as imagens. Havendo disponibilidade de memória no cachê, pode-se acumular os portais antigos juntamente com a posição que lhes corresponde ser gerados. Assim, com uma simples consulta antes de se calcular a visualização efetivamente, pode-se verificar se a imagem desejada já não havia sido utilizada há algum tempo atrás e ainda está disponível.

6 Resultados

O algoritmo foi implementado utilizando-se o engine gráfico Fly3D [Fly3D], próprio para desenvolvimento de jogos 3D, sendo que a visualização deste sistema é implementada utilizando o

OpenGL. A tabela 1 mostra o desempenho da aplicação, para diversos números de portais colocados numa cena. Para estes testes foi utilizado um Pentium IV 1.6 GHz, com uma placa de vídeo Gforce III. Diversas imagens obtidas como resultado podem ser vistas no final do artigo (figuras 9 a 12)

Nº de portais	Frames / segundo (Portais com resolução de 256x256)	Frames / segundo Portais com resolução de 128x128
0	50	50
1	49	49
3	20	30
6	16	24
9	12	18
12	3	4

Tabela 1 – Comparação de performance para rendering de portais de refração

7 Conclusão e trabalhos futuros

Este trabalho apresenta uma eficiente generalização para o algoritmo de portais para calcular reflexos e refrações em superfícies planas, sem a necessidade de um pré-processamento e que simplesmente consiste num rendering recursivo numa parte da cena. Para superfícies não planas, propõem-se utilizar vários portais, um para cada polígono que compõe o objeto. Deve-se, entretanto, criar alguma forma de interpolação baseada em técnicas de image-based rendering para que a borda dos polígonos tenha uma texturização homogênea. A Fig. 12 mostra uma imagem gerada com vários portais para um cilindro.

Neste trabalho não se abordou o caso de refração múltipla, onde um raio de luz muda de meios transmissivos mais de uma vez. Entretanto, o método proposto não apresenta obstáculos para este tipo de fenômeno, bastando para isto permitir que a imagem gerada por um portal de transmissão possa conter outro portal de transmissão dentro dele e fazer com que os portais sejam bidirecionais. Deve-se ter, no entanto, o cuidado para não permitir um número grande de recursões, pois se comprometeria a interatividade do sistema.

O framework de *multi-texturing* pode ser facilmente usado para acrescentar outros efeitos especiais além do reflexo e da refração, como por exemplo caustics, para simular raios de luz desviados por alguma superfície transmissora. Este efeito de caustics pode inclusive ser dinâmico, podendo-se simular por exemplo efeitos de iluminação provocados por movimentos da superfície da água.

8 Agradecimentos

Os autores estão agradecidos à Paralelo Computação pelos recursos que foram disponibilizados para a pesquisa. O primeiro autor agradece também ao CNPq, pelo amparo dado à pesquisa.

9 Referências Bibliográficas

[Airey et al. 90] Airey, J.; Rohlf, J.; Brooks, F. Toward Image Realism with Interactive Update Rates in Complex Virtual Building Environments. Symp. On Interactive 3D Graphics, 41-50, 1990.

[Aliaga et al. 97] Aliaga, Daniel G.; Lastra, Anselmo A. Architectural Walkthroughs Using Portal Textures. IEEE Visualization ’97, pp 355-362, October 1997.

[Blinn et al. 76] Blinn, J. F., Newel, M. E. Texture and Reflection in Computer Generated Images. Comm. ACM, 19 (10), 362-367. 1976.

[Eberly 00] Eberly, D. H. 3D Game Engine Design - A Practical Approach to Real-Time Computer Graphics. Morgan Kaufmann Publishers. 2000.

[Fly3D] www.fly3d.com

[Glassner 89] Glassner, A., S. An Introduction to Ray Tracing. Academic Press. 1989.

[Gortler et al. 97] Gortler, Steven J.; He, Li-wei; Cohen, Michael F. Rendering Layered Depth Images. Technical Report, MSTR-TR-97-09. http://www.research.microsoft.com/pub/tr/tr-97-09.ps

[Heidrich et al. 99] Heidrich, W.; Seidel, H. P. Realistic, hardware-accelerated shading and lighting. Computer Graphics (Proceedings of SIGGRAPH 1999), August 1999.

[Hurley 00] Hurley, K. Approximating Refraction. Nvidia white paper. 2000.

[Luebke et al. 96] Luebke, D.; Georges, C. Portals and Mirrors: Simple, Fast Evaluation of Potentially Visible Sets. Department of Computer Science. University of North Carolina at Chapel Hill. Department publication. 1996.

[Max et al. 95] Max, Nelson; Ohsaki, Keiichi. Rendering Trees from Precomputed Z-Buffer Views. In Patrck M. Hanrahan and Werner Purgathofer editors, Rendering Techniques ’95: Proceedings of the 6th Eurographics Workshop on Rendering, pp 45-54, Dubiln, Ireland, June 1995.

[McMillan et al. 95] McMillan, L.; Bishop G. Plenoptic Modeling: An Image-Based Rendering System. Computer Graphics (Proceedings of SIGGRAPH 95), pp 39-46. August 1995.

[McMillan 97] McMillan, Leonard. An Image-Based Approach to Three-Dimensional Computer Graphics, Ph.D. Dissertation, University of North Carolina, April 1997.

[McReynolds et al. 98] McReynolds,T.; Blythe,D.; Grantham,B.; and Nelson,S. Programming with OpenGL: Advanced Techniques, Course Notes 17 at SIGGRAPH’98, 1998. [Also in: http://www.sgi.com/software/ opengl/advanced98/notes/]

[Möller and Haines 99] Möller,T. and Haines,E. Real-Time Rendering, A K Peters, Natick, Massachusetts, 1999.

[Peercy et al. 00] Peercy, M. S.; Olano, M.; Airey, J.; Ungar, J. Interactive Multi-Pass Programmable Shading. Computer Graphics (Proceedings of SIGGRAPH 2000), August 2000.

[Popescu et al. 98] Popescu, Voicu; Lastra, Anselmo; Aliaga, Daniel; Oliveira, Manuel. Efficient Warping for Architectural Walkthroughs Using Layered Depth Images. IEEE Visualization ’98, October 1998.

[Rafferty et al. 98a] Rafferty, Matthew M.; Aliaga, Daniel G.; Popescu, Voicu; Lastra, Anselmo. Images for Accelerating Architectural Walkthroughs. Computer Graphics & Applications, November/December 1998.

[Rafferty et al. 98b] Rafferty, Matthew M.; Aliaga, Daniel G.; Lastra, Anselmo. VRAIS’98, pp. 228-233, March 1998.

[Teller et al. 91] Teller, S.; Séquin, C. H. Visibility Preprocessing For Interactive Walkthroughs. Computer Graphics (Proceedings of SIGGRAPH 91), 1991.

[Watt et al. 00] Watt, Allan and Policarpo, Fabio. 3D games, Real-time Rendering and Software Technology. ACM Press, 2001.

[Vlachos 01] Vlachos, A. Approximating fish tank refractions. In: DeLoura, M. (ed.),Game Programming Gems 2, Charles River Media, p.402-405, 2001.

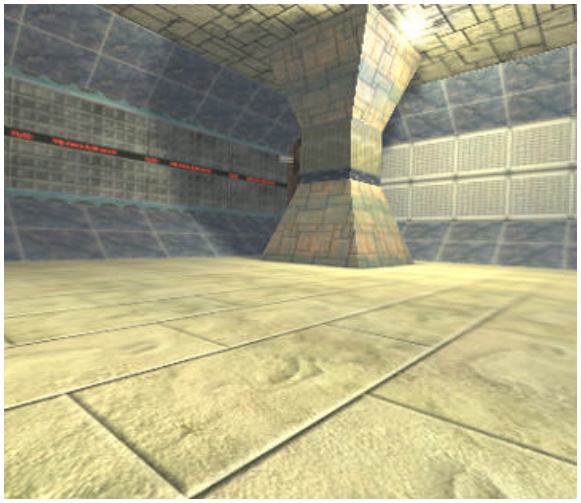
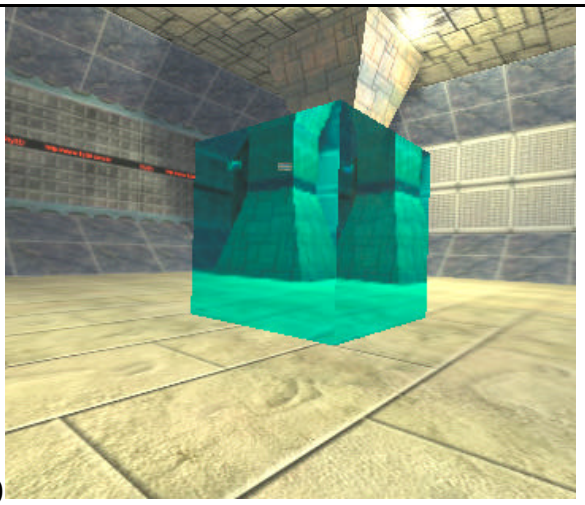
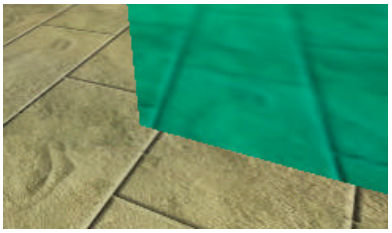
(a)  (b) 

Figura 9 - (a) Cena a ser utilizada pelos exemplos, sem nenhum objeto refrativo. (b) A mesma cena com um cubo com refração.



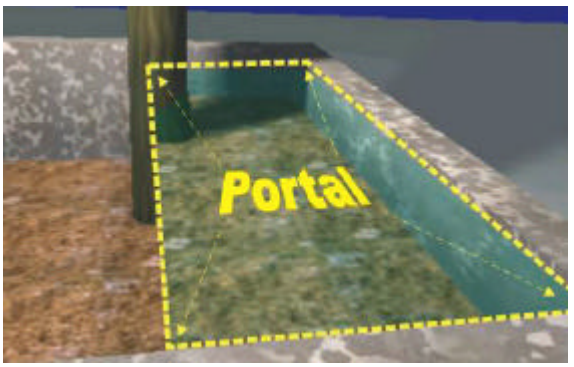
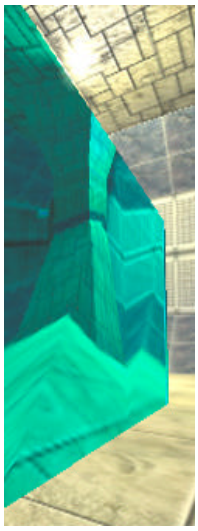


Figura 10 - Detalhes do fenômeno de refração em tempo real.

(a) 

(b) 

(c) 

(d) 

Figura 11 - Exemplo de deformação da projeção do portal. Em (a) não há nenhum refinamento de coordendas de texturas, em (b) o portal é dividido em 2x2, em (c) o portal é dividido em 4x4 e em (d) a divisão é de 10x10.

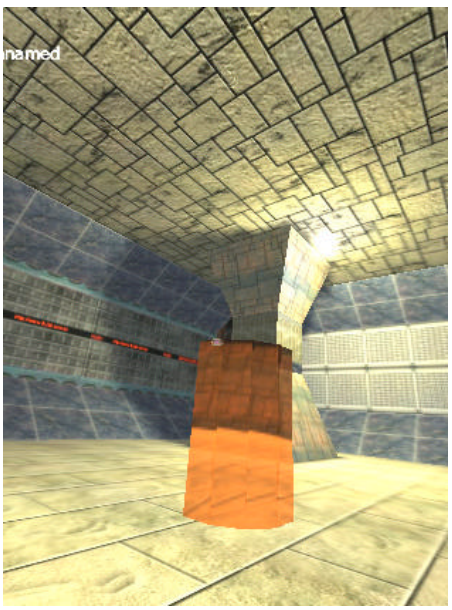


Figura 12 – Exemplo de um objeto composto por vários portais, um para cada polígono da malha. Note-se a descontinuidade das imagens refratadas entre as bordas.