Refinamento Progressivo da Cena com Traçado de Raios Distribuído

RENATO GONÇALVES DE ARAUJO
MARCELO GATTASS
MARCELO DREUX

TeCGraf-Grupo de Tecnologia em Computação Gráfica, PUC-Rio
Rua Marquês de São Vicente, 255, 22453-900 Rio de Janeiro, RJ, Brasil
{renato,gattass,dreux}@icad.puc-rio.br

Abstract. In the Image Synthesis discipline, the usual Distributed Ray Tracing algorithms are not computationally efficient, which makes the production process tedious and not productive. This work presents an optimization of the Distributed Ray Tracing algorithm that improves the productivity: a preview technique based on the progressive refinement of the scene.

Keywords: Ray Tracing, Progressive Refinement, Preview Technique.

Introdução

A técnica de Traçado de Raios (*Ray Tracing*) [Whitted (1980)] tornou-se rapidamente bastante popular devido à qualidade dos seus resultados e à simplicidade de seus métodos, que simulam o comportamento dos raios de luz provenientes de fontes luminosas, sobre as superfícies dos objetos, de forma bastante natural.

Praticamente, qualquer tipo de objeto pode ser utilizado no método e a adição de outros efeitos, independentes de sua natureza, tais como refração, texturas e movimentação de objetos, também pode ser feita de forma simples, já que todas as etapas envolvidas na geração da cena são facilmente compreensíveis. Até mesmo a adaptação do método para Traçado de Raios Distribuído [Cook (1984)], que produz imagens muito mais realistas, pode ser feita sem muitos problemas.

Porém, à primeira vista, uma característica é desanimadora na técnica de Traçado de Raios: o esforço computacional necessário não é pequeno (em sua grande parte, cálculos de interseção raio/objetos) e isso se reflete no tempo de geração da cena. Muitas vezes, na geração de fotografias virtuais, o usuário precisa fazer várias alterações na cena a ser criada, para que esta se mostre exatamente como foi imaginada. Normalmente, a escolha da iluminação e das características das superfícies não é uma tarefa simples e envolve, até mesmo, uma motivação artística. Se o tempo de cada alteração depender do tempo para a geração completa da cena, certamente há um problema.

Com as implementações atuais, uma cena complexa pode demorar horas de processamento para ser finalizada. O problema do tempo excessivo pode ser minimizado com o refinamento progressivo da imagem, semelhante à proposta de Hollasch [Hollasch (1992)], para permitir a sua exibição à medida que os cálculos vão sendo feitos, oferecendo uma noção com poucos detalhes, mas bastante satisfatória, da imagem completa. Assim, se for necessário, o usuário pode interromper os cálculos muito antes da finalização da cena, economizando tempo de processamento.

No campo da Síntese de Imagens, este trabalho está voltado para a geração de fotografias virtuais e o seu principal objetivo é investigar a otimização da técnica de Traçado de Raios, de forma a torná-la mais eficaz na produção de foto-realismo e animação. Para isso, são adicionados alguns efeitos ao trabalho de Roman Kuchkuda [Kuchkuda (1987)], como refração e luz colorida, e é feita uma adaptação para Traçado de Raios Distribuído. Esta técnica, apresentada por Robert Cook [Cook (1984)], é capaz de adicionar *motion-blur*, penumbra, *gloss*, translucidez e, aqui, *anti-aliasing* e controle sobre a profundidade de campo. A ênfase está na vantagem da pré-visualização (*preview*), através do refinamento progressivo da cena, proposto por Steve Hollasch [Hollasch (1992)], pesquisado e implementado neste trabalho.

Traçado de Raios Tradicional

O Método de Traçado de Raios tradicional (*Ray Tracing*) [Whitted (1980)] foi formulado a partir da observação da realidade. Nesta técnica, os objetos são construídos e suas cores calculadas, através das interseções destes objetos com os "raios de luz" simulados, que partem do ponto de vista do usuário e passam por cada *pixel* da tela. Com isso, deve-se considerar as fontes luminosas presentes na cena, os materiais que compõem as superfícies dos objetos (com todas as suas propriedades), as distâncias dos objetos à

tela de projeção e a posição da tela em relação ao ponto de vista do observador. Todos estes procedimentos de cálculos foram obtidos a partir de observações da natureza e, obviamente, simplificados a ponto de se tornarem factíveis com os equipamentos atuais. Apesar das simplificações, o método apresenta um resultado bastante satisfatório e, muitas vezes, surpreendente.

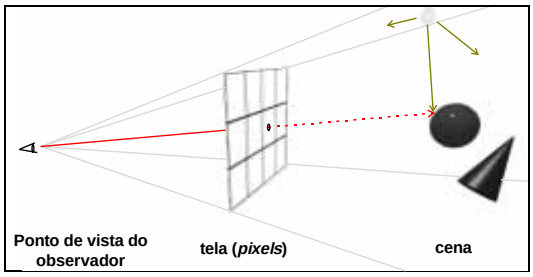


Figura 1 - O Método de Traçado de Raios Tradicional.

A Figura 1 ilustra o modelo descrito acima. De uma forma bem resumida, cada *pixel* fornece a direção de um raio, que parte do ponto de vista do observador e se dirige para a cena composta de objetos modelados pelo usuário. Verifica-se quais objetos fazem interseção com este raio e obtém-se as distâncias destes pontos de interseção até o ponto de vista do observador. A menor distância indica o objeto mais próximo e, conseqüentemente, o que será visível naquele *pixel*. Os demais pontos de interseção ficam ocultos pelo primeiro.

Tendo-se o ponto de interseção e as informações sobre o objeto (forma, material da superfície etc.), calcula-se a sua intensidade luminosa. Esta intensidade é obtida a partir da influência das fontes luminosas da cena neste ponto. O cálculo da influência é feito através do vetor normal à face do objeto e dos raios que são traçados até as fontes de luz, utilizando-se as características do material (transparência, reflexividade, contribuição para a iluminação difusa etc.) [Foley (1990).

Para aumentar o realismo das cenas, são feitas algumas alterações no algoritmo básico, para substituir a intensidade de brilho das fontes de luz por cores (luz colorida) e para representar a refração da luz através de objetos transparentes.

Traçado de Raios Distribuído

A técnica de Traçado de Raios tradicional é, porém, inadequada para o tratamento de outros fenômenos que enriquecem ainda mais o realismo das imagens geradas, como profundidade de campo e, principalmente, na solução do efeito provocado pela discretização da imagem no espaço da tela, conhecido como *aliasing* [Cook (1984)].

O controle sobre a profundidade de campo é um efeito muito usado em fotografia e cinema, quando alguns objetos aparecem em foco, enquanto outros ficam fora de foco (borrados). Normalmente, utiliza-se este recurso para se chamar a atenção do espectador para os objetos em foco, plano da imagem onde está se desenvolvendo o objetivo principal desta, enquanto os outros objetos fazem parte do cenário ou simplesmente não têm importância significativa naquele instante [Lasseter (1987)].

No método tradicional [Whitted (1980)] (Figura 1), todos os objetos da cena ficam em foco, já que é utilizado o modelo de um único ponto de projeção geométrica (o próprio olho humano). A substituição deste modelo por uma câmera virtual mais aprimorada, que considere os efeitos de uma lente esférica delgada biconvexa e de uma função de abertura da mesma (diafragma) já é suficiente para alterar a profundidade de campo da cena. Este modelo foi proposto por Robert Cook [Cook (1984)] e implementado na PUC-Rio por Daniel Mazzuca [Mazzuca (1991)] (Figura 2).

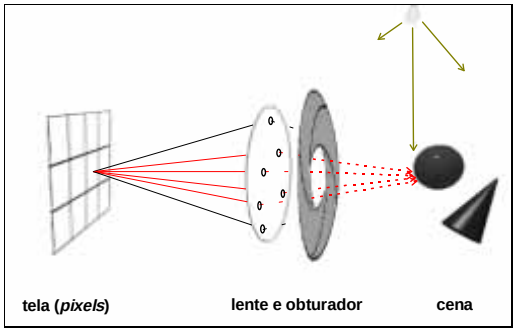


Figura 2 - Modelo de Traçado de Raios utilizando uma câmera virtual mais aprimorada, com lente e obturador (diafragma).

Neste trabalho, não serão apresentadas com detalhes as características ópticas das lentes, mas a Figura 3 ilustra claramente o controle obtido sobre a profundidade de campo.

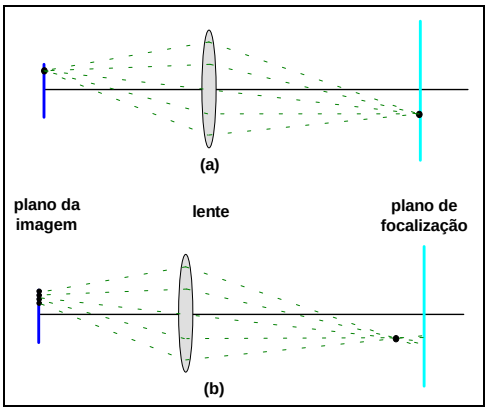


Figura 3 - Efeito da lente na obtenção da profundidade de campo. Em (a), tem-se um ponto no plano de focalização da lente e em (b) isto não acontece.

No plano da imagem, um *pixel* qualquer não fica dependente de um único raio, que após disparado atingirá os objetos no espaço da cena. Como mostra a figura, vários raios são disparados do *pixel* em questão e, dependendo da posição em que atingem a lente, emergem dela em direções diferentes, podendo, até mesmo, atingir objetos distintos. Qualquer lente possui um plano de focalização; isso significa que todos os raios (e somente estes raios) que partem de um mesmo ponto nesse plano e atravessam a lente, incidirão também em um mesmo ponto sobre o plano da imagem - Figura 3(a). Para um ponto fora do plano de focalização, vários raios provenientes de diferentes pontos no plano da imagem (*pixels*) podem atingi-lo. Isso faz com que vários *pixels* tenham a informação de intensidade de luz sobre este ponto, fazendo com que ele apareça borrado na imagem final - Figura 3(b). Esta aparência borrada, chama-se círculo de confusão.

Pode-se dizer que a visibilidade da cena é diferente para cada ponto da lente e, portanto, a intensidade final em um *pixel* da imagem é o resultado da integração das intensidades calculadas sobre toda a superfície da lente. Como é comum em Computação Gráfica, a integração sobre a superfície da lente pode ser resolvida através da utilização de pontos de amostragem. Para isto, a superfície da lente é subdividida em regiões e, para cada região, escolhe-se um ponto representante, de onde será disparado um raio. Os valores de intensidade obtidos são filtrados e o resultado é a intensidade final desejada do *pixel*.

O principal problema das técnicas de amostragem é uma conseqüência da discretização da imagem, conhecido por efeito *aliasing*. Cook mostrou [Glassner (1989)] uma boa maneira de se explicar as causas do efeito *aliasing*, através de processamento de sinais. As conclusões obtidas indicam que a incapacidade de reproduzir corretamente a freqüência de um sinal através de amostras regulares é um erro inerente ao processo, mas o aparecimento de freqüências falsas como *aliasings* é uma conseqüência direta da regularidade com que são colhidas as amostras.

O tamanho de um pixel define o limite superior para as freqüências que podem ser exibidas. Freqüências mais altas causam aliasings, como o aparecimento de linhas serrilhadas (*jaggies*) na imagem, ao invés de linhas contínuas [Glassner (1989)]. Uma solução simples para reduzir este problema é conhecida como "super-amostragem", que consiste em subdividir o *pixel* em vários *subpixels*, fazendo com que se tenha mais informações para se determinar a intensidade do *pixel* em questão. Quanto mais sub-regiões são divididas, melhor o resultado, mas também mais cálculos serão realizados. Esta solução não elimina o problema do *aliasing*, mas pode levá-lo a dimensões insignificantes.

Baseando-se no princípio de que o *aliasing* é uma conseqüência da regularidade do *grid* de amostragem, Cook propôs a amostragem estocástica, na qual se adiciona um deslocamento aleatório às localizações de uma distribuição de amostragem. Este deslocamento é conhecido como *jitter*. Desta forma, o *aliasing* é substituído por ruído, muito mais tolerado pelo olho humano [Glassner (1989)].

Esta amostragem passa a ser adotada, então, para os *subpixels* e para as sub-regiões da lente, fazendo com que se obtenha, ao mesmo tempo, *anti-aliasing* e profundidade de campo (Figura 4).

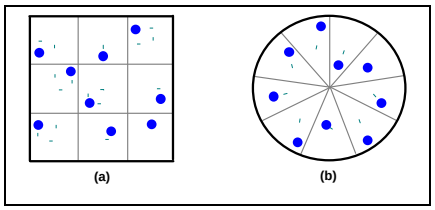


Figura 4 - Distribuição *jitter* sobre a área de um pixel (a) e sobre a superfície da lente (b).

Porém, não é necessário disparar um raio para cada *subpixel* e, para cada *subpixel*, um raio para cada sub-região da lente, já que o número de raios traçados no total seria proibitivo. A idéia do Traçado de Raios Distribuído é exatamente "distribuir" os raios adicionais do espaço da tela para as outras dimensões envolvidas (no caso, a lente). Assim, cada raio traçado é proveniente de um diferente *subpixel* e atinge um diferente ponto da lente. Esta associação é feita de forma aleatória. Para cada raio obtido, aplica-se o processo de Traçado de Raios Tradicional.

O poder da técnica de Traçado de Raios Distribuído pode ser percebido com a quantidade de efeitos que podem ser obtidos, como *anti-aliasing* e controle da profundidade de campo, descritos anteriormente, *motion-blur*, penumbra, reflexões borradas (*gloss*) e translucidez. Para se obter estes últimos, é necessário "distribuir" os raios do espaço da tela para a dimensão do tempo, "distribuir" os raios de sombra para as superfícies das fontes de luz, "distribuir" os raios refletidos ao redor do raio de reflexão e "distribuir" os raios refratados ao redor do raio de refração, respectivamente.

Refinamento Progressivo da Cena

Os métodos usualmente utilizados para cálculo das equações de Traçado de Raios não utilizam nenhuma regra para escolha da ordem em que os *pixels* serão obtidos. O que normalmente se observa é que, em um sistema com visualização da cena, cada linha do dispositivo é inteiramente calculada para depois ser exibida, uma após a outra, de cima para baixo ou de baixo para cima.

Quando se utiliza subdivisões de cada pixel então, deve-se aguardar um tempo muito longo para que se possa garantir que a imagem foi definida com os parâmetros corretos. O objetivo aqui é utilizar a proposta de Hollasch [Hollasch (1992)], para visualização progressiva da cena e adaptar o algoritmo de Traçado de Raios para que a imagem seja refinada progressivamente, enquanto vai sendo visualizada. Dessa forma, o usuário pode visualizar a figura sendo montada (a princípio, por *pixels* bem grandes) e tem noções cada vez mais detalhadas de sua aparência à medida que vai sendo calculada, até o nível final de refinamento. O efeito obtido passa a impressão de que os *pixels* diminuem de tamanho, o que pode ser comparado a um mosaico no sentido inverso.

A Figura 5 descreve como o processo de refinamento progressivo utilizado neste trabalho funciona. Inicialmente, a imagem é dividida em quatro partes e escolhe-se um *pixel* em cada quadrante. No nosso caso, para simplificar, escolhemos os *pixels* na mesma posição dentro de cada quadrante - 5(a). A cor calculada para aquele *pixel* será utilizada para todo o quadrante, o que também será feito nos outros três - 5(b). A seguir, cada quadrante será dividido em quatro novos quadrantes e repete-se a escolha dos *pixels* e cálculos - 5(c) e 5(d), até que todos os *pixels* já tenham sido escolhidos e calculados.

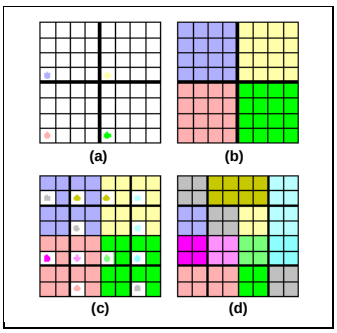


Figura 5 - O processo do refinamento progressivo.

Para não ter que controlar qual pixel já foi previamente calculado, utiliza-se sempre o mesmo pixel de cada quadrante, por exemplo, o mais inferior à esquerda. Então, o quadrante mais inferior à esquerda de cada subdivisão nunca precisa ser calculado, pois já se conhece a sua cor (resultado do cálculo anterior, do quadrante que os envolve).

Além da escolha dos pixels ser baseada na subdivisão de quadrantes, a ordem de escolha dos quadrantes também tem grande importância no refinamento progressivo. A ordem de escolha seqüencial pode provocar um efeito desagradável, à medida que o refinamento avança. Se for escolhido, inicialmente, o quadrante inferior esquerdo, a seguir o direito, depois o quadrante superior esquerdo e, finalmente, o superior direito, a impressão passada ao usuário é claramente de que o refinamento se faz da parte inferior da imagem para a superior, através da ilusão de "linhas". Este efeito é ainda mais observável quando o refinamento está no final, já que os *pixels* tornam-se menores e as linhas demoram mais tempo para "subirem" pela imagem.

Para resolver este problema, foi definida neste trabalho uma nova ordem para o desenho dos quatro subquadrantes na tela, baseada no hábito de leitura dos nossos olhos. Uma informação bastante utilizada em programação visual e citada por Giacomantonio [Giacomantonio (1976)], é que os nossos olhos estão habituados a ler da esquerda para a direita, de cima para baixo. Assim, as regiões mais valorizadas em uma imagem, para uma rápida observação, são os quadrantes superior esquerdo e inferior direito, tendo os outros dois menos importância relativa. Portanto, a melhor ordem para o refinamento se faz quando são escolhidos os subquadrantes desta forma, como indica a Figura 6, já que o efeito de refinamento é rápido e o tempo de observação, neste instante, é curto.

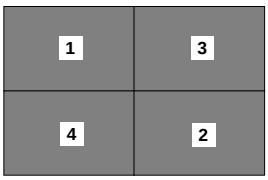


Figura 6 - Regiões mais valorizadas na imagem e ordem dos subquadrantes.

É importante lembrar que não se perde nenhum cálculo, pois todos os *pixels* são escolhidos para obtenção da cor apenas uma única vez. O que acontece diferente dos processos convencionais, é que somente a ordem desta escolha é alterada.

À primeira vista, pode-se pensar que a melhor forma de implementar este método é através de recursividade, mas, na verdade, deixar o controle da escolha da ordem dos *pixels* para a própria pilha de execução poderia causar dois problemas. O primeiro ocorreria se a imagem tivesse um tamanho grande o suficiente para ultrapassar o limite da pilha de execução, mas que poderia ser resolvido com um controle desta.

O outro, mais complexo, provocado pela subdivisão do quadrante dentro da própria rotina recursiva, faria com que sempre o primeiro quadrante subdividido fosse novamente subdividido, antes dos cálculos dos demais. Assim, não haveria mais refinamento progressivo em toda a imagem, mas somente dentro de cada quadrante. Para resolver este problema, a solução ficaria com vários testes de nível de recursividade, o que tornaria o cálculo bem mais lento, como ocorre no clássico problema da busca em profundidade *versus* busca em largura.

A solução encontrada neste trabalho foi utilizar uma fila seqüencial dos quadrantes subdivididos. Cada novo quadrante dividido tem a sua cor calculada e é colocado no final da fila. Para se calcular as cores do próximo quadrante, utiliza-se o primeiro da fila. Novamente, divide-se o novo quadrante em quatro subquadrantes e calcula-se as cores dos três que não contêm o *pixel* previamente calculado. A seguir, os quatro são colocados no final da fila, e assim por diante até a finalização de todos os pixels. Um trecho de código que ilustra a utilização dessa fila foi colocado no Apêndice.

Um ponto crítico deste algoritmo é o tamanho da fila. Pode-se perceber que a fila seqüencial cresce de acordo com o tamanho da imagem, já que os subquadrantes são inseridos nesta estrutura à medida que vão sendo calculados.

Se uma imagem tiver suas dimensões de altura e largura como potências de 2 (para facilitar as divisões), os subquadrantes deixam de ser reinseridos na fila no momento que contiverem 4 *pixels* somente, como ilustra a Figura 7. Nesta situação, cada novo subquadrante conterá apenas um *pixel*, que não precisa ser reinserido na fila. Assim, para o caso de potência de 2, o tamanho da fila terá, no máximo, um quarto do total do número de pixels da imagem, ou seja, $\frac{N}{4}$.

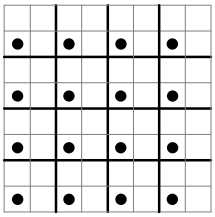


Figura 7 - Subquadrantes com 4 *pixels*.

Se a imagem não tem suas dimensões como potência de 2, não se pode garantir, ao final, subquadrantes com 4 *pixels* na fila. O pior caso ocorre quando todos os subquadrantes contiverem 9 *pixels*, conforme mostra a Figura 8(a). Neste caso, o total de subquadrantes na fila será de N / 9 e, na próxima iteração, será retirado um elemento da fila para cada três que são inseridos, gerando-se então um acréscimo de 2 elementos. A Figura 8(b) mostra este caso. Portanto, teremos um total de:

$$\frac{N}{9} + \frac{2 \cdot N}{9} = \frac{N}{3}$$

o que significa que o limite superior de elementos da fila é de um terço do total de *pixels* da imagem.

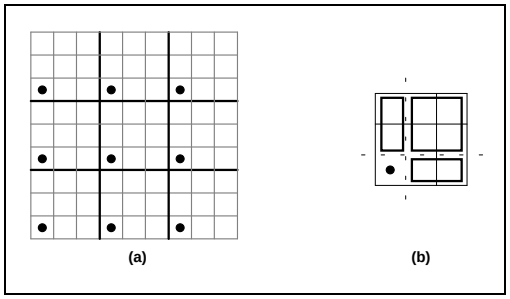


Figura 8 - Subquadrantes com 9 *pixels* (a) e 3 novos subquadrantes a serem incluídos (b).

Com a utilização do método de Traçado de Raios Distribuído, são feitas duas “passagens” de refinamento progressivo. A primeira delas utiliza apenas o primeiro *subpixel* de cada *pixel*, para a obtenção da sua

intensidade. O efeito para o usuário é de que está sendo realizado o Traçado de Raios tradicional, em uma primeira etapa. Isto se faz para que o objetivo proposto se mantenha: a velocidade na síntese de uma imagem satisfatória. Quanto mais tempo o usuário esperar, mais a imagem vai sendo refinada.

Ao fim da primeira "passagem" de refinamento, inicia-se a segunda que, para cada *pixel*, obtém as intensidades dos outros *subpixels* ainda não calculados. A ordem de escolha dos *pixels* é exatamente a mesma usada na primeira "passagem" do refinamento (reproduzindo-se a mesma fila), sendo que não é mais necessário desenhar blocos na tela, pois todos os *pixels* já foram desenhados com uma intensidade próxima da final (resultado de uma amostra apenas). O efeito para o usuário é de que os *pixels* vão mudando de cor, para se aproximarem da cor ideal e correta. Da mesma forma, nenhum cálculo realizado é perdido e o usuário pode interromper o processo no momento que estiver satisfeito.

Resultados

A Figura 9 mostra três momentos do refinamento progressivo. Em 9(a), temos o cálculo em torno de 1% do total. Em 9(b), em torno de 5% e, finalmente, em 9(c), já temos a cena finalizada.

O tempo para a execução de todo o processo é dependente do equipamento ter capacidade de aceleração gráfica, já que para cada cálculo é necessário desenhar um conjunto de pixels em uma determinada cor. Em uma máquina sem capacidade de aceleração gráfica, observou-se um acréscimo de tempo da ordem de 30%, comparando-se quando não é utilizado o refinamento progressivo (para uma imagem de pouca complexidade).

Além disso, o controle do processo principal de cálculo foi passado para a interface, que somente realiza os procedimentos necessários para geração da cena enquanto estiver inerte (estado *idle*). Com isso, o processador não fica "preso" durante os cálculos e o usuário é capaz de mover o mouse, solicitar interrupção e continuação do cálculo, gravar imagens ainda não finalizadas etc. Conseqüentemente, o tempo total para obtenção da cena pode aumentar.

Porém, esta diferença pode ser considerada pequena, quando se percebe as vantagens oferecidas com a visualização por esta técnica. Para imagens com muitos objetos, refrações e reflexões (imagem complexa), este acréscimo total no tempo pode cair para níveis de 10 a 15% acima do tempo de geração da cena em um ambiente sem interface de visualização.

Conclusão

O método de Traçado de Raios Distribuído aliado ao Refinamento Progressivo torna-se uma técnica bastante útil para a síntese de imagens, tanto na área de visualização científica, quanto na área artística, através de fotografias virtuais. A adaptação da proposta de Hollasch para utilização do Refinamento Progressivo com a técnica de Traçado de Raios Distribuído foi realizada incluindo-se a mudança na ordem de escolha dos subquadrantes para o refinamento, o que proporcionou um resultado muito bom.

A primeira sugestão para trabalhos futuros é otimizar o cálculo, através da construção de caixas envolventes (*bounding-boxes*) delimitando as fronteiras dos objetos. Os raios que não fizerem interseção com as caixas, cujos cálculos são bem mais rápidos, são desprezados, otimizando o método.

Outra otimização é alterar a técnica de redução de *jaggies* (superamostragem) para torná-la adaptativa, ou seja, somente subdividir o *pixel* em mais *subpixels* quando for realmente necessário. Quando todos os raios traçados para um *pixel* não interceptarem nenhum objeto, por exemplo, a integração deste resultado já é suficiente para proporcionar uma boa descrição da região sendo calculada. Quando o *pixel* sendo analisado indicar a fronteira de um objeto, mais *subpixels* podem ser necessários para representar corretamente a intensidade luminosa ideal.

Outra sugestão é acrescentar texturas às superfícies dos objetos. As texturas, sem dúvida alguma, contribuem muito para aumentar o realismo das cenas. No momento de se obter a cor de um pixel (*shading*), além das características do material, pode-se utilizar uma imagem previamente definida (digitalizada, por exemplo) e devidamente mapeada para as formas do objeto que será representado.

Finalmente, pode ser incluída a possibilidade de gerar uma seqüência de imagens para animação, adicionar *motion-blur* e, até mesmo, otimizar o cálculo de animações, através de caixas envolventes (*bounding-boxes*) no espaço-tempo, como sugerido por Glassner [Glassner (1988)].

Agradecimentos

Este trabalho foi desenvolvido no TeCGraf, Grupo de Tecnologia em Computação Gráfica da PUC-Rio. O TeCGraf é suportado financeiramente através de projetos principalmente com a PETROBRÁS/CENPES. O MCT e CAPES também financiaram este trabalho.

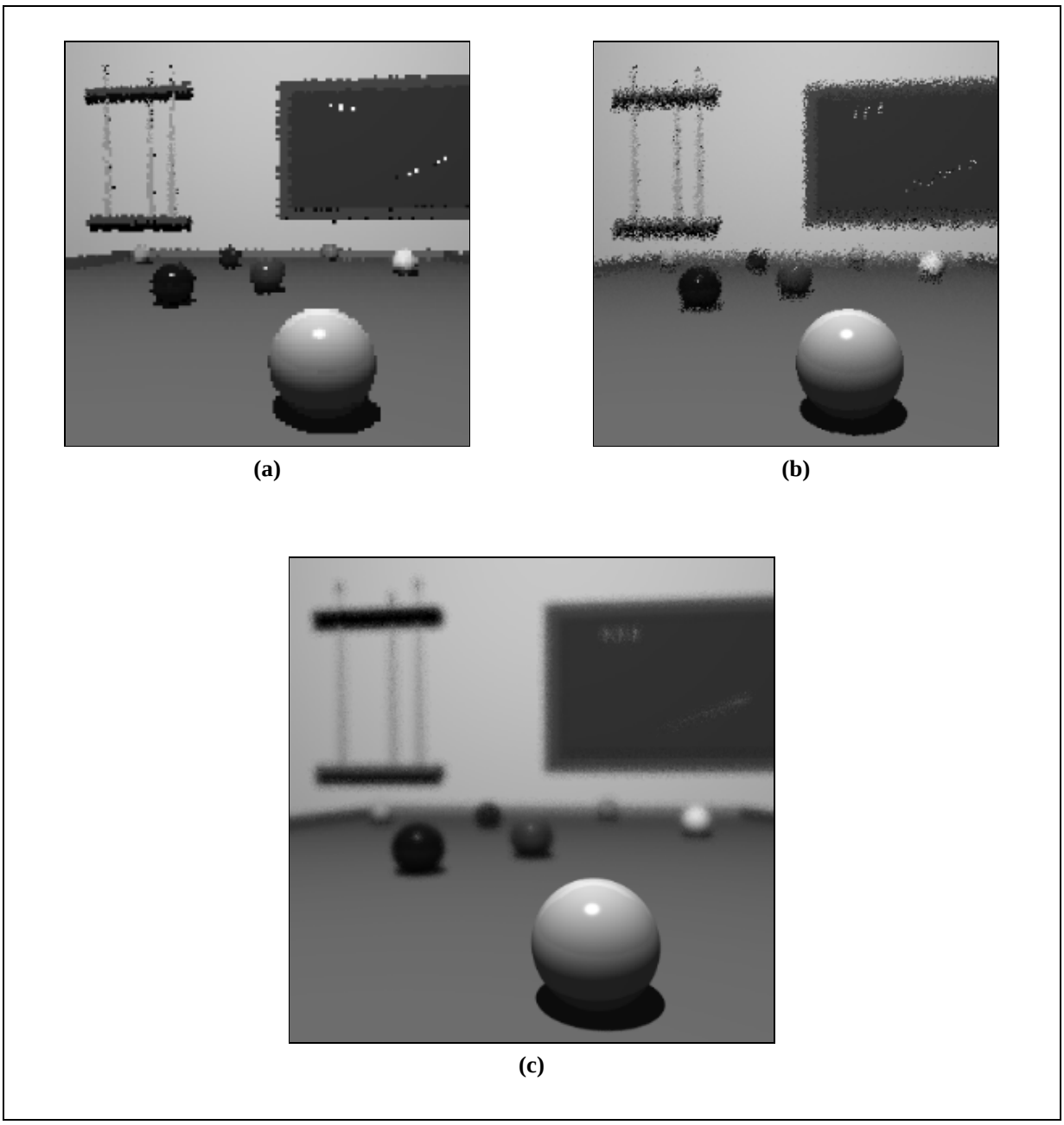


Figura 9 - Refinamento progressivo de uma cena com (a) 1%, (b) 5% e (c) 100%.

Referências Bibliográficas

R. L. Cook et alii. "Distributed Ray Tracing". *SIGGRAPH'84*.18(3), July 1984.

J. D. Foley. et alii. *Computer Graphics: Principles and Practice*. Addison-Wesley, 1990.

M. Giacomantonio. *Os Meios Audiovisuais*. Edições 70, 1976.

A. S. Glassner. "Spacetime Ray Tracing for Animation". *IEEE Computer Graphics & Applications*. 8(2), March 1988.

A. S. Glassner. *An Introduction to Ray Tracing*. Academic Press Inc, 1989.

S. Hollasch. "Progressive Image Refinement Via Gridded Sampling". *Graphics Gems III*. Academic Press Inc,1992.

R. Kuchkuda. “An Introduction to Ray-Tracing”. *Theoretical Foundations of Computer Graphics and CAD*. Nato International Advanced Study Institute. Italy, 1987.

J. Lasseter. “Principles of Traditional Animation applied to 3D Computer Animation”. *SIGGRAPH '87*. 21(4), July 1987.

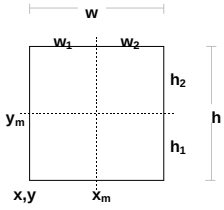
D. A. Mazzuca. *Profundidade de Campo e "Motion Blur" no Método de "Ray Tracing" Distribuído.* Dissertação de Mestrado. PUC-Rio, Dept. de Informática. Julho 1991.

T. Whitted. “An Improved Illumination Model for Shaded Display”. *Communications of the ACM*. 23(6), June 1980.

Apêndice

Trecho de código ilustrando o uso da fila de subquadrantes:

```
raytrace ()
{
   /*  inicializa a FILA  */
  InsertQuad (0, 0, sizex, sizey);

  /* calcula a cor do primeiro quadrante */
  /* (pixel inferior esquerdo) */
  compute (0., 0., sizex, sizey);

  /* faz enquanto existir elemento na fila */
  while (queue)
  {
    /* retira o primeiro elemento da fila */
    RemoveQuad (&x, &y, &w, &h);

    /* neste momento, já sabemos que o quadrante */
    /* inferior esquerdo já foi calculado        */

    /* divide o quadrante em 4 novos subquadrantes*/
    w1 = w/2;
    h1 = h/2;

    xm = x + w1;
    ym = y + h1;
    w2 = w - w1;
    h2 = h - h1;
    if (w1 == 0)
      w1 = 1;
    if (h1 == 0)
      h1 = 1;
```




```
    /* trabalha com o superior esquerdo */
    if (h > 1)
    {
      /* verifica se precisa inseri-lo na fila (posição e tamanho) */
      if (w1 > 1 || h2 > 1)
        InsertQuad (x, ym, w1, h2);

      /* calcula a sua cor  (pixel inferior esquerdo) */
      compute (x, ym, w1, h2);
    }

    /* trabalha com o inferior direito */
    if (w > 1)
    {
      /* verifica se precisa inseri-lo na fila (posição e tamanho) */
      if (w2 > 1 || h1 > 1)
        InsertQuad  (xm, y, w2, h1);

      /* calcula a sua cor  (pixel inferior esquerdo) */
      compute (xm, y, w2, h1);
    }

    /* trabalha com o superior direito */
    if (w > 1 && h > 1)
    {
     /* verifica se precisa inseri-lo na fila (posição e tamanho) */
     if (w2 > 1 || h2 > 1)
       InsertQuad (xm,ym,w2,h2);

     /* calcula a sua cor  (pixel inferior esquerdo) */
     compute (xm,ym,w2,h2);
    }

    /* trabalha com o inferior esquerdo */

    /* verifica se precisa inseri-lo na fila (posição e tamanho) */
    if (w1 > 1 || h1 > 1)
      InsertQuad  (x, y, w1, h1);

    /* não precisa calcular a sua cor (já foi calculada) */

  }   /* fim do while */
}  /* fim da função */
```