Simulação e Controlo do Efeito de Profundidade de Campo em Tempo de Interacção

Hugo Miguel Ventura Manuel Próspero dos Santos
CITI, Departamento de Informática
Faculdade de Ciências e Tecnologia
Universidade Nova de Lisboa
2829-516 Caparica
{ hugompventura@gmail.com, ps@di.fct.unl.pt }

Sumário
A geração de imagens foto-realistas tem sido um dos grandes objectivos da Computação Gráfica e, nesse sentido, o conceito óptico da profundidade de campo afigura-se como um dos efeitos mais importantes. Auxilia o sistema visual humano a interpretar as posições dos objectos de uma cena tridimensional virtual, dando ideia das distâncias relativas entre os mesmos. Foram já propostos vários métodos para modelação da profundidade de campo, mas, devido à evolução tecnológica que vai superando limitações do passado, só hoje em dia é possível introduzir correctamente este efeito em programas interactivos, de aplicação imediata em Realidade Virtual ou em Realidade Aumentada. Neste artigo apresenta-se uma metodologia de implementação eficiente duma técnica para simular o efeito de profundidade de campo e introduzi-lo numa vulgar aplicação gráfica interactiva. De forma a garantir simulações com tempo de execução apropriado à interactividade exigida, são consideradas optimizações de diversa ordem, entre as quais o uso das potencialidades de hardware gráfico. Aproveita-se o paralelismo no processamento em GPU através da linguagem de shading GLSL, elevando assim o desempenho global quando manipulados pixels em grande número.

Keywords
Profundidade de Campo, Filtragem, Tempo Real, Shaders, GPU, OpenGL, Frame Buffer Objects

1. INTRODUÇÃO

Uma cena tridimensional normalmente sintetizada por computador tem a particularidade de possuir uma profundidade de campo infinita e, assim, mostrar todos os seus objectos perfeitamente detalhados, situação a que se associa uma notória falta de realismo visual. Para dar um aspecto real a uma imagem de síntese, esta deve ser aproximada o mais possível à imagem que um ser humano captaria com os seus próprios olhos ou com câmaras ópticas. Ora uma das noções mais relevantes para tal realismo é a da profundidade de campo, existente no processo natural da visualização humana e fazendo com que a área focada seja vista com aspecto nítido, enquanto as restantes, na mesma direcção mas com diferentes distâncias de afastamento, se exibem desfocadas. Por conseguinte, este conceito óptico de profundidade de campo foi já estendido às imagens de síntese.

No passado, a profundidade de campo foi gerada em aplicações gráficas segundo métodos variados, dos quais sobressaem, por exemplo, *Ray Tracing* [Coo84] e *Accumulation Buffer* [Hae90]. Porém, praticamente todos estes métodos denunciavam um grave compromisso entre qualidade e rapidez de formação do resultado final.

Os avanços tecnológicos no mundo informático, e especificamente na área da Computação Gráfica, levaram à exigência prática de funcionalidades interactivas nas suas aplicações, apenas eficientes se os respectivos tempos de processamento se aproximassem aos de tempo real. Também as técnicas de implementação da profundidade de campo, até então de utilização proibitiva em aplicações gráficas interactivas devido ao seu imenso custo computacional, conseguiram evoluir e moldar-se às novas exigências. Dos tipos de métodos atrás mencionados, foram os de pós-processamento que melhor conseguiram elevar a rapidez de execução na modelação da profundidade de campo [Ber04, Zho07, Lee08], adaptando-se da melhor forma à contínua evolução do hardware gráfico.

Este artigo apresentará uma proposta de metodologia de implementação que, assentando num método de pós-processamento existente [Zho07], permite uma eficiente simulação e controlo da profundidade de campo num ambiente virtual interactivo.

2. ENQUADRAMENTO

2.1 Motivação

Com uma correcta noção de profundidade, muito beneficiarão aplicações em áreas como as de videojogos, Realidade Virtual (RV) [Rok96, Hil08] ou Realidade Au-

mentada (RA). Nestes domínios, existem hoje aplicações com objectivos tão diversos como são os de entretenimento, simulação, treino militar, arquitectura e urbanismo, entre muitos outros, não havendo praticamente quaisquer limites. Por exemplo, o realismo em RV, ou, ainda com maior relevância, em RA, será muito difícil ou até impossível de se atingir caso não exista nenhuma possibilidade de percepção sobre as distâncias dos objectos no ambiente em que se encontra o utilizador final. A profundidade de campo dará, aí, a naturalidade necessária às interacções do utilizador com o correspondente universo virtual.

2.2 Objectivos

Pretendeu-se simular o efeito de profundidade de campo em tempo de interacção através dos recursos actualmente existentes na programação de gráficos 3D e, ao mesmo tempo, construir uma metodologia que possa ser utilizada por qualquer programador de aplicações interactivas em que a profundidade de campo seja exigida. Para essa finalidade seleccionou-se, como técnica de base, a que foi desenvolvida por T. Zhou, J. X. Chen e M. Pullen [Zho07].

3. O CÍRCULO DE CONFUSÃO

A profundidade de campo define o efeito resultante do processo de focagem de objectos que se encontrem na extensão do campo de visibilidade. Os objectos de uma área focada exibem-se perfeitamente nítidos, o que significa uma alta definição dos seus contornos. Por outro lado, objectos que se distanciem da zona focada apresentam-se naturalmente pouco nítidos, como se pode ver na fotografia da Figura 1. O efeito de profundidade de campo é bem conhecido em Cinema e em Fotografia, tratando-se de um dos efeitos que a Computação Gráfica procura oferecer às imagens de síntese quando se pretende foto-realismo.

Figura 1: Profundidade de campo em Fotografia.

A criação de aplicações gráficas com renderização realista de dados 3D pressupõe a definição de um modelo de câmara fotográfica. Os gráficos de computadores, em geral, carecem da profundidade de campo finita presente nas imagens obtidas por câmaras reais. Tal resulta da adopção do modelo de câmara "buraco de alfinete" (*Pinhole*), em que os raios de luz atravessam um pequeno orifício de uma caixa fechada e projectam a imagem na superfície interior desta, onde se colocará o filme. Devido ao buraco ser infinitesimamente pequeno, em teoria qualquer ponto do filme só poderá ser atingido por um único raio de luz. É, assim, criada uma imagem totalmente nítida, que se pode considerar com uma profundidade de campo infinita [Kos01].

Sabendo-se que o modelo *Pinhole* é incapaz de recriar a profundidade de campo, foi então utilizada a modelação de câmaras com lentes delgadas (*Thin Lens*), cuja espessura pode ser desprezada mas em que se considera ser o índice de refracção superior à unidade. A aproximação a este modelo acaba por ser a mais simples de entre os modelos de simetria circular e abertura óptica finita.

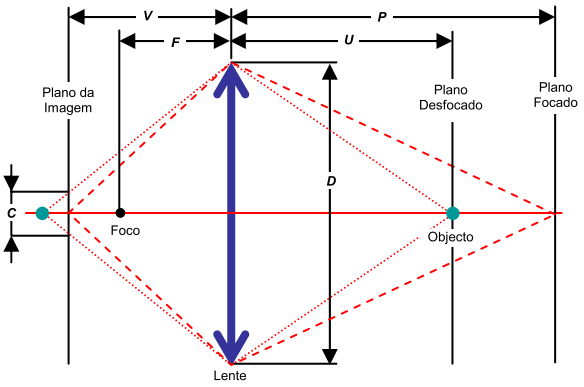

Figura 2: Círculo de Confusão.

A partir da informação apresentada na Figura 2, é então possível entender como a imagem de uma cena é criada num plano de filme com um afastamento ***V*** do plano da lente, dada a abertura óptica ***D*** e a distância focal ***F***. Para renderização computacional de imagens simulando o modelo *Thin Lens*, revela-se imprescindível a especificação da distância ***P*** do plano de focagem. Esta distância pode ser obtida facilmente através da relação (1), que é a equação fundamental da óptica geométrica [Kos01].

$$\frac{1}{P}+\frac{1}{V}=\frac{1}{F} \qquad (1)$$

Considerem-se agora os múltiplos raios de luz dispersos que, a partir de um ponto de um objecto, passam pela lente da Figura 2. Caso todos estes raios convirjam para um único ponto no plano de imagem a uma distância ***V*** da lente, o respectivo ponto do objecto define-se como focado. Os raios de luz provenientes de um ponto não focado intersectam o plano de imagem numa área circular, tecnicamente conhecida por Círculo de Confusão. O diâmetro deste círculo, ***C***, pode ser calculado através da equação (2).

$$C=\left| D\frac{F\,(P-U)}{U\,(P-F)} \right| \qquad (2)$$

Existem ainda outros modelos ópticos que, embora mais realistas, são, também, muito mais complexos, como é o caso das lentes espessas (*Thick Lens*), em que se deixa de

ignorar a espessura da lente. Contudo, a capacidade que esses modelos têm de simular até as aberrações das câmaras ópticas reais só acontece em troca de uma implementação complexa e uma execução demorada. Face à exigência de se obter taxas de execução compatíveis com uma forte interactividade aplicacional com o utilizador, o modelo *Thin Lens* foi o único que se demonstrou perfeitamente adaptado a tais objectivos, sendo, por isso, considerado para consequente implementação.

4. MODELAÇÃO E TÉCNICAS EXISTENTES

A profundidade de campo foi já objecto de trabalho de muitos autores no passado, com a implementação de inúmeras aproximações a renderizações de imagens com profundidade de campo finita. Entre estas, serão resumidas de seguida as técnicas de *Ray Tracing*, de *Accumulation Buffer* e algumas de Pós-Processamento, todas elas habilitadas a simular o efeito de profundidade em imagens sintéticas, a partir da descrição de uma cena gráfica tridimensional, com diversos graus de qualidade.

4.1 *Ray Tracing* Distribuído

Na técnica de renderização avançada de *Ray Tracing* Distribuído [Coo84] é tida em conta, para cada pixel, a possível reflexão especular envolvendo todos os objectos em cena, bem como a refracção da luz nos materiais transparentes. Em vez de cada pixel ser atravessado por um raio proveniente de um único ponto, são gerados e distribuídos vários raios para simulação do atravessamento da lente pela luz. A cor final de cada pixel é calculada através da média de intensidades dos raios distribuídos [Dem04]. O processamento na técnica de *Ray Tracing* é um excelente exemplo de uma relação qualidade/eficiência, pois garante a melhor qualidade de imagem em troca de um maior número de passos algorítmicos executados sobre uma cena tridimensional. Mas as suas velocidades de execução estão ainda muito afastadas das exigidas em simulações interactivas.

4.2 Accumulation Buffer

Tendo por base o tradicional *z-buffer* (contendo os dados de profundidade) e usando-se um *buffer* suplementar, designado por *accumulation buffer*, é possível simular-se o efeito da profundidade de campo [Hae90]. Para uma abertura óptica de tamanho finito é renderizada uma cena múltiplas vezes, segundo o modelo *Pinhole*, a partir de posições de câmara ligeiramente diferentes. É efectuada a amostragem necessária por toda a lente. Todas as imagens renderizadas são, assim, combinadas e acumuladas no referido *accumulation buffer*. A profundidade de campo neste tipo de técnica é tanto mais realista quanto mais passos de renderização forem efectuados, constatando-se que o número de passos necessários é proporcional à área do círculo de confusão em causa. De momento, infelizmente, e para além da tendência quanto ao aparecimento de artefactos visuais, a capacidade desta técnica para oferecer bons resultados exige ainda um elevado número de passos que a afasta das frequências de tempo real desejadas no âmbito deste trabalho.

4.3 Pós-Processamento

As implementações da profundidade de campo através dos conotados métodos de pós-processamento seguem um procedimento comum. Primariamente é renderizada uma cena para o *frame buffer*, e só posteriormente é processada a imagem da cena resultante, exibindo-se finalmente a imagem com o efeito de profundidade de campo. Geralmente, as técnicas de pós-processamento são conduzidas por meio de métodos de filtragem digital, razão de serem também conhecidas como técnicas de pós-filtragem. A essência da metodologia adoptada em pós-processamento, juntamente com um reduzido custo computacional aproveitando as crescentes capacidades de hardware gráfico, fazem da mesma a mais qualificada para a obtenção de profundidade de campo em taxas de refrescamento compatíveis com tempo real.

Os primeiros trabalhos importantes devem-se a Potmesil e Chakravarty [Pot82], que criaram um processamento posterior à geração da cena tridimensional com a função de guardar cada amostra de ponto, retendo informação relativa à posição, visibilidade e intensidade de cor. Cada ponto amostrado é convertido para um círculo de confusão, cujas dimensão e distribuição de intensidade são determinadas pela profundidade do ponto e pelas características do modelo de câmara utilizado. Assim, cada pixel na rasterização da imagem final é calculado pela acumulação das distribuições de intensidade dos círculos de confusão sobrepostos, isto para todos os pontos amostrados.

Embora praticamente todas as técnicas de pós-processamento sigam um procedimento global idêntico, existem ainda inúmeras variantes neste tipo de técnica. A maioria das variações ocorre na fase de filtragem. Foram já publicados diversos algoritmos para amostrar e pesar intensidades de pixels, baseados na profundidade ou informação relacionada com a profundidade obtida aquando da renderização inicial. Podem citar-se exemplos recentes com técnicas por aplicação de filtro separável, como em [Zho07], a base deste trabalho, ou técnicas por filtragens anisotrópicas, como em [Ber04] e [Kri03].

Os autores J. Krivanek e J. Zara [Kri03] desenvolvem um rápido algoritmo que renderiza a profundidade assentando na técnica de filtragem *surface splatting*, o qual permite renderizar superfícies, opacas ou transparentes, de agrupamento de pontos sem qualquer conectividade. Esta implementação distingue-se por cada ponto ser desfocado antes da formação da imagem final, evitando o processo de focagem em separado, como acontece regularmente na maioria dos métodos de Pós-Processamento.

Bertalmio *et al.*, em [Ber04], introduz uma nova equação diferencial parcial (EDP) para difusão anisotrópica, aplicada a imagens 2D renderizadas por modelo de câmara *Pinhole*. Isto é conseguido desfocando as imagens segundo a informação de profundidade presente no *z-buffer* da cena 3D correspondente. A sua metodologia de profundidade de campo implementada é inteiramente executada em GPU, em conjunto com um esquema numérico adequado à EDP referida. Adaptado às características da

linguagem de hardware gráfico programável, é assim possibilitada uma renderização em tempo-real.

Ainda outras técnicas de simulação de profundidade de campo foram recentemente implementadas atendendo a requisitos específicos aplicacionais, como [Kas06] e [Ham07] para animações cinematográficas e jogos computorizados, respectivamente.

M. Kass, A. Lefohn e J.Owens, em [Kas06], desenvolveram uma aproximação à profundidade de campo numa tentativa de criar uma ferramenta interactiva para criação de animação cinematográfica. O seu intuito seria o de atingir, interactivamente, imagens sintéticas computorizadas de acordo com a visão de realizadores cinematográficos ou directores de fotografia. Depois de parametrizada cada cena da animação na aplicação sugerida, é então utilizada a técnica de *Ray Tracing* para computação das imagens finais exibidas no filme em questão. A profundidade de campo é modelada por Kass *et al.* como uma difusão de calor, onde o tamanho do Círculo de Confusão corresponde ao calor do pixel aplicado à imagem criada originalmente. A solução em estudo direcciona-se para uma série de sistemas lineares tridimensionais, que implementam efectivamente filtros recursivos independentes, computados em tempo constante por pixel, autónomo do tamanho dos círculos de confusão processados [Kas06].

Por sua vez, Hammon *et al.* [Ham07] direccionam a sua implementação para o universo dos jogos de computadores e consolas, principalmente para os de estilo "Primeira-Pessoa". Neste tipo de jogos o jogador controla a visão da personagem no jogo, por isso é essencial a profundidade de campo para que a mesma se torne tão real quanto possível. A solução de Hammon *et al.* descarta, em situações particulares, a simulação física correcta da profundidade de campo em prol de uma melhor experiência de entretenimento proporcionada ao jogador, devido às características únicas associadas ao respectivo jogo e à sua jogabilidade. Quaisquer artefactos visuais presentes na jogabilidade, em fase de implementação, ficam corrigidos ao serem desfocadas, especificamente, as descontinuidades visíveis por manipulação de Círculos de Confusão, calculados a partir de funções Gaussianas.

5. FILTRAGEM DUPLA

A modelação do efeito de profundidade de campo foi realizada através da aplicação de um filtro de passagem dupla sobre uma imagem 2D renderizada. Trata-se de um procedimento de pós-processamento baseado no algoritmo de Zhou *et al*. [Zho07].

A primeira passagem para filtragem da imagem original é efectuada numa direcção pré-determinada (horizontal ou vertical). A imagem resultante será a imagem de entrada para a segunda passagem. Esta segunda fase do processo de filtragem será feita na direcção perpendicular à da primeira passagem. Cada pixel é processado de forma idêntica em ambas as passagens. Os pesos atribuídos à contribuição dos pixels vizinhos amostrados, aquando do processo de filtragem, regem-se segundo três factores de controlo: o factor da sobreposição, o da intensidade luminosa, e o da fuga de intensidade (*intensity leakage*). Cada um dos três factores origina um peso com valor entre zero e um.

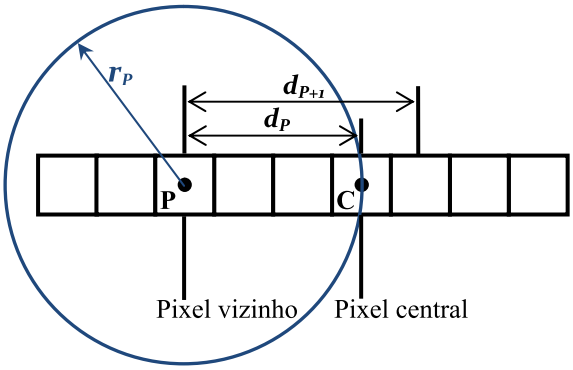

Figura 3: Contribuição do pixel P para o pixel com centro em C, de acordo com o círculo de confusão.

O factor de sobreposição, dado pela função (3) e com a geometria indicada na Figura 3, define a contribuição que o círculo de confusão de um pixel amostrado P dá ao pixel central C a ser calculado. Um pixel vizinho P contribuirá totalmente para o pixel C se este último estiver totalmente contido no círculo de confusão, de raio r_p, do pixel P. O mesmo pixel P não contribuirá de forma alguma para C se não houver qualquer sobreposição. Nos casos restantes, a diferença entre a distância dos dois pixels e o círculo de confusão de P definirá a contribuição deste último para C.

$$O(r_P) = \begin{cases} 0, & r_P < d_P \\ r_P - d_P, & d_P \le r_P < d_{P+1} \\ 1, & r_P \ge d_{P+1} \end{cases} \quad (3)$$

O segundo factor, calculado a partir de (4), refere-se à intensidade da luz sobre cada pixel. Seguindo a opção simplificada de Zhou *et al.* [Zho07], no círculo de confusão do pixel amostrado é usada aqui uma intensidade luminosa que diminuirá na proporção do inverso do quadrado do respectivo raio.

$$I(r_P)\ \alpha\ \frac{1}{r_P^2} \quad (4)$$

Por fim, o terceiro e último factor a pesar na filtragem de cada pixel da cena sintética tridimensional é o de dispersão de cor, isto é, a fuga de intensidade de pixels em situações de oclusão entre objectos. A contribuição segundo este factor é determinada pela função definida em (5), que ajusta o peso do pixel amostrado P por um factor baseado no tamanho do círculo de confusão do pixel central se P se encontrar mais longe da câmara que o plano de focagem. Garante-se, assim, que objectos para além do plano focado não contribuem para objectos focados, evitando artefactos por fuga de intensidade. Por outro lado, P contribui com a sua cor por inteiro se estiver a uma distância da câmara menor ou igual que o plano focado. Isto garante que os pixels amostrados contribuem

sempre na desfocagem de pixels mais distantes em profundidade se estiverem à frente do plano em foco.

$$L(z_P) = \begin{cases} \alpha\, r_C, & z_P > z_F \\ 1, & z_P \le z_F \end{cases} \quad (5)$$

Sintetizando, a cor de cada pixel (central) é misturada segundo a soma das contribuições de todos os seus pixels vizinhos, e para cada um destes é calculada a sua contribuição pelo produto dos pesos atribuídos por cada um dos três factores revistos, como representado em (6). De notar que o peso dado ao pixel central se resume simplesmente à sua intensidade luminosa, não incluindo nem o primeiro nem o terceiro factores anteriormente referidos. Depois de somadas todas as contribuições dos pixels vizinhos amostrados ao pixel central, é finalmente renormalizado o seu valor, ou seja, é dividido pelo total de contribuições somadas. Finalmente, garante-se que o resultado respeita os limites definidos e aceites pelo hardware gráfico, remetendo então a cor final do pixel central para o *frame buffer* [Zho07].

$$W(P) = O(r_P) \times I(r_P) \times L(z_P) \quad (6)$$

6. TECNOLOGIAS

6.1 OpenGL

O algoritmo para renderização da profundidade de campo foi implementado usando OpenGL [Shr05, Sob03]. Com esta escolha, as aplicações posteriormente criadas poderão ser executadas em múltiplas plataformas de hardware gráfico sem quaisquer alterações de código.

6.2 FBO

A extensão *Frame Buffer Object* de OpenGL, também conhecida por FBO, representa a técnica de programação usada para definir o destino de desenho de uma renderização que não o *frame buffer* [FBO06]. Este processo funciona como uma renderização *offscreen*. Um objecto FBO necessita de ter definida a imagem ou imagens a si anexadas, ou por meio de texturas ou de um outro tipo de objecto OpenGL, o *Renderbuffer*.

Esta arquitectura oferece flexibilidade de programação controlando o que renderizar, e para que *buffers*, durante a execução da aplicação gráfica. Um único objecto FBO consegue guardar, se exigido, toda a informação associada aos diferentes *buffers* lógicos de OpenGL no acto da renderização 3D. A grande vantagem na utilização de FBO prende-se com uma natural melhoria de desempenho de aplicações que necessitem de aceder à informação da imagem renderizada. Devido à necessidade de processar a profundidade de campo nas imagens renderizadas, evita-se, assim, a escrita da imagem no *frame buffer* e posterior cópia para uma textura. As imagens são renderizadas directamente para uma textura através de objectos FBO, de acordo com o conceito de programação "Renderizar para Textura", resultando no aumento da eficiência global da aplicação.

6.3 *Shading* por GLSL

Sem taxas de tempo de execução ao nível de tempo real perde-se a interactividade pretendida, e quantas mais tarefas forem simultaneamente efectuadas, maior a dificuldade na procura de uma simulação visualmente correcta e eficiente.

A programação em GPU de uma simulação visual é suficiente para garantir que a mesma será executada nos tempos mais baixos permitidos pela tecnologia gráfica utilizada, tornando-se um aspecto quase imprescindível ao sucesso final da mesma.

Aproveitando os recursos oferecidos por um GPU actual, e tendo em conta a metodologia de pós-processamento adoptada, considerou-se a técnica de *shading* como a mais apropriada para se renderizar o efeito de profundidade de campo sobre imagens sintéticas. A programação em GPU foi executada, neste trabalho, através da linguagem de alto nível OpenGL Shading Language (GLSL). Esta linguagem para programação de gráficos é, na verdade, o conjunto de duas linguagens próximas, permitindo a criação de *shaders* para os processadores programáveis contidos no fluxo de processamento do OpenGL. As duas linguagens referidas para programação de *shaders* distinguem-se segundo o processador a que se destina: *vertex* ou *fragment*.

OpenGL oferece os mecanismos necessários à criação e à compilação dos respectivos *shaders*, bem como à ligação dos mesmos formando o código executável denominado Programa. Respeitando algumas limitações relativas ao tipo de *shaders* em causa, estes podem receber dados de entrada enviados pela própria aplicação OpenGL.

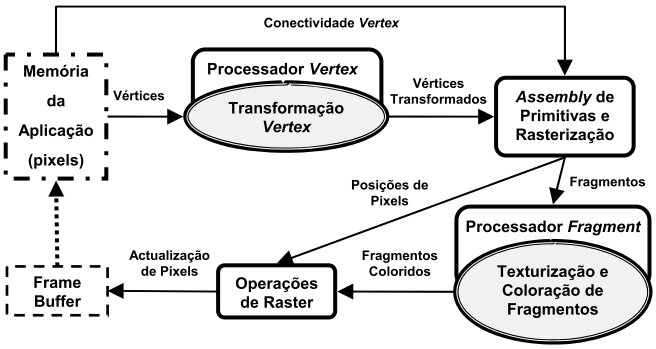

Figura 4: Fluxo de processamento gráfico.

Para uma qualquer programação dos *vertex* e *fragment shaders*, o programador deverá compreender o fluxo de processamento gráfico (Figura 4) e verificar os contextos em que se inserem os diversos *shaders*, as suas capacidades e limitações globais. Um *vertex shader* é executado no processador *Vertex* que representa a unidade programável que opera sobre os vértices e todos os dados a estes associados. Executam tarefas relacionadas com, por exemplo, transformação de posição ou do vector normal, iluminação por *vertex* ou mesmo a transformação e geração de coordenadas de texturas. De forma análoga, um *fragment shader* é executado no processador *Fragment* e realiza tarefas como, por exemplo, a computação de cor,

profundidade e coordenadas de texturas por pixel, ou a manipulação e aplicação de texturas.

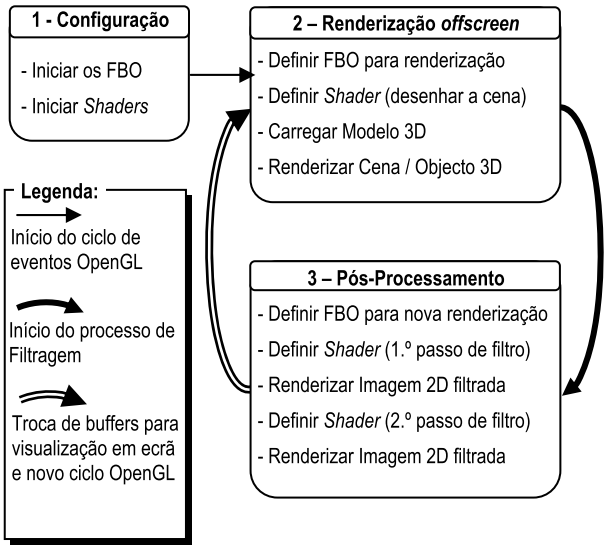

Figura 5: Fluxo de processamento da simulação.

7. ESTRUTURA DA APLICAÇÃO

Vemos, na Figura 5, como é integrada numa aplicação OpenGL o processamento de *shaders* através da tecnologia GLSL, e como são efectuadas renderizações directas para objectos FBO na execução do algoritmo gerador do efeito de profundidade de campo numa cena sintética 3D. De seguida, referem-se os aspectos mais importantes de cada estágio do fluxo aí ilustrado.

7.1 Configurações

A primeira fase do processo de integração numa aplicação passou pela configuração de todas as tecnologias envolvidas na execução da simulação da profundidade de campo. As configurações englobaram a criação do chamado "contexto OpenGL", como a definição de janelas com duplo *buffer* (renderização *offscreen*), o sistema de cores RGBA e a activação de teste de profundidade para obtenção de informação de profundidade por pixel. Foram inicializados os objectos FBO juntamente com as texturas, ou objectos *Renderbuffer* associados, para guardar o mapa de cor da cena 3D original captada.

Todas as texturas foram geradas com idênticos parâmetros e propriedades. Por razões de optimização, as texturas utilizadas foram definidas com o formato RGBA16F. Cada pixel é, assim, guardado na textura num grupo de quatro valores reais, correspondendo aos quatro canais do sistema de cor RGBA, cada componente do pixel dispondo de 16 bits, num total de 64 bits por pixel. A execução ficou optimizada pela utilização do canal alfa nas texturas. Normalmente este canal está associado à informação relativa a transparências, mas se se prescindir deste efeito, aproveitam-se os respectivos bits para fixar o valor de profundidade de cada pixel renderizado, comprimindo numa única textura toda a informação.

Por fim, neste estágio foram igualmente inicializados os *shaders* a serem utilizados durante todo o processo de renderização do efeito de profundidade de campo.

7.2 Renderização *Offscreen*

Depois de iniciado o ciclo infinito para eventos de OpenGL, a função definida para renderização de imagem será também repetidamente invocada até à terminação do programa. Nesta função inicia-se o estágio de processamento da simulação implementada, realizando-se a renderização prévia àquela a ser exibida em ecrã e que irá oferecer a imagem ao pós-processamento do estágio seguinte.

O objectivo principal nesta fase é renderizar, através de *shaders*, a cena ou objecto 3D e, por uso de objectos FBO, guardar a respectiva informação nos seus *buffers* para ser posteriormente processada na criação de profundidade de campo.

Neste processo deve ser indicado qual o objecto FBO a ser usado para guardar a informação de renderização ou indicado o *buffer* para onde a escrita do *fragment shader* deverá ser direccionada, condizendo com o mesmo *buffer* anexado durante as configurações de objectos FBO. É igualmente necessário alterar, para activo, o estado do *shader* encarregado desta primeira renderização, ou seja, aquele que irá ser usado durante o desenho da cena 3D. A imagem desenhada é, naturalmente, desprovida de profundidade de campo, apresentando-se como uma típica imagem de câmara *Pinhole* e, por isso, será processada no estágio seguinte para obtenção do efeito de profundidade desejado segundo as propriedade ópticas definidas.

Na renderização desta fase, o valor de profundidade real de cada ponto da imagem, em relação ao ponto de vista corrente, é obtido pelo *vertex shader*. Neste é multiplicada a matriz *modelview* de OpenGL com a posição *vertex* calculada. Finalmente, no *fragment shader*, os valores de cor e profundidade são escritos na textura determinada nas configurações de objectos FBO.

7.3 Uso de *Fragment Shader*

Nesta fase, depois de renderizados todos os objectos 3D, perfeitamente detalhados quaisquer que sejam as suas distâncias à câmara, é então dado início à técnica de pós-processamento para criação de profundidade de campo.

Com o mesmo tipo de procedimentos executados no estágio anterior na manipulação de tecnologias *shading* e FBO, são processadas as duas passagens de filtragem da metodologia apresentada na secção 5 deste artigo. A primeira filtragem será realizada sobre a imagem/textura original, resultante da fase anterior (Renderização *Offscreen*), e a nova imagem originada será a informação de entrada para a segunda, e última, filtragem. Por conseguinte, o resultado desta renderização é que será guardado no *frame buffer* de OpenGL para visualização em ecrã.

O *fragment shader* aplicado nos dois passos de filtragem irá processar da mesma forma todos os pixels da imagem previamente renderizada em OpenGL, produzindo a mesma imagem mas agora com o efeito de profundidade de campo. Esta operação leva em linha de conta as propriedades da câmara óptica, como veremos já de seguida.

Através de variáveis uniformes da linguagem GLSL [Ros06], são utilizados os seguintes dados de entrada, indispensáveis à simulação pretendida:

- Textura com o mapa de cor da imagem 2D a filtrar;
- Distância à câmara do plano focado na cena 3D;
- Valor de um pixel escalado para um texel, para cálculo de coordenadas da textura a filtrar;
- Factor de multiplicação para determinação dos círculos de confusão de cada ponto da imagem visualizada;
- Tamanho do filtro.

8. INTEGRAR A PROFUNDIDADE DE CAMPO

No que se refere à introdução do efeito de profundidade de campo numa aplicação gráfica existente, a grande vantagem do método apresentado prende-se com o facto de ser realmente fiel à noção de pós-processamento devido a processar a imagem virtual somente depois da sua normal renderização. Desta forma, a integração da metodologia para profundidade de campo numa aplicação gráfica poderá ser conseguida sem modificar o código previamente realizado. Apenas duas questões se levantam ao implementar a metodologia em causa: a utilização de *shaders* em GPU e a renderização directa para texturas.

8.1 Renderizar com Profundidade de Campo

Sobre a renderização original, a profundidade de campo é obtida segundo uma sequência de renderizações com aplicação do efeito exigido através da técnica de *shading*. Veremos, de seguida, a contextualização da utilização de *shaders* numa renderização.

Uma imagem é renderizada a partir de um conjunto de instruções, ou primitivas, operadas em pleno hardware gráfico respeitando o seu próprio fluxo de processamento. Este fluxo, denominado por *pipeline* gráfico, tipicamente aceita como entrada os valores referidos como representação de uma cena tridimensional, donde resulta, como saída, uma imagem 2D rasterizada.

O *pipeline* gráfico é muito apropriado ao processo de renderização por permitir ao GPU tratar vértices e fragmentos de modo perfeitamente independente, manipulados respectivamente pelos *vertex* e *fragment shaders*. Diferentes vértices e fragmentos podem estar a ser processados em estágios distintos do *pipeline* gráfico num dado momento, como também múltiplos vértices ou fragmentos podem ser processados de forma paralela no seu correspondente processador (*vertex* ou *fragment*). Estas propriedades dos *shaders* sintetizam a grande vantagem da utilização de *shaders* por parte da metodologia apresentada neste artigo, promovendo o aumento de eficiência em conjunto com a obtenção de um efeito mais realista. A flexibilidade de manipulação de vértices e fragmentos durante uma renderização e a capacidade de o efeito gerado pelos *shaders*, neste caso a profundidade de campo, aproveitar o paralelismo oferecido pelo próprio GPU ajudam a compreender o alcance generalista da integração da técnica numa qualquer aplicação gráfica.

8.2 Renderizar para Textura

Devido à necessidade de serem efectuadas diversas renderizações posteriores sobre a renderização da cena 3D original, foi utilizada na metodologia apresentada a técnica de programação de Renderização para Textura implementada através dos objectos FBO (*Frame Buffer Objects*). É evitada a repetição de processos de renderização para *frame buffer* e consequente cópia para textura, que são evidentemente prejudiciais ao desempenho global da aplicação. Com a Renderização para Textura, todas as renderizações necessárias são cumpridas directamente para texturas, incluindo a renderização inicial da cena 3D, sendo que apenas a última é renderizada para o *frame buffer* com o objectivo de ser visualizada em ecrã pelo utilizador.

8.3 Integração em Aplicações Gráficas

A aplicação deste método para geração de profundidade de campo num qualquer programa gráfico pode acontecer durante a criação do próprio programa ou por inserção depois de o programa estar já criado e operacional. As duas modalidades foram experienciadas durante a realização deste trabalho. Se a metodologia de profundidade for integrada durante a criação de uma aplicação gráfica, benéfico por evitar posteriores alterações, esta deverá seguir todo o procedimento apresentado neste artigo tendo em conta as diversas renderizações directas para texturas através de processamentos por uso de *shaders*.

Note-se que quaisquer instruções ao nível de processamento *vertex* (p.ex., iluminação) ou *fragment* (p.ex., cor) devem ser obrigatoriamente escritas nos programas dos *shaders*, pois sendo executados *shaders*, apenas os seus códigos de programação serão efectivamente considerados no *pipeline* gráfico. Por este motivo, no caso em que a profundidade de campo seja integrada numa aplicação já existente, a única imposição será a de transferir quaisquer instruções ao nível de processamento *vertex* ou *fragment* do programa original para os novos *shaders* de processamento da profundidade de campo. Caso contrário, um ambiente virtual criado originalmente com, por exemplo, iluminação realista, ao ser-lhe introduzido o efeito da profundidade de campo segundo a metodologia proposta e não considerando a questão discutida neste ponto, irá exibir um distinto e incorrecto resultado por ficar desprovido da iluminação realista anteriormente presente. Isto explica-se por ser, a iluminação, um dos mais importantes e utilizados processamentos *vertex* nas aplicações gráficas de hoje. Nada mais é alterado relativamente ao código original, faltando apenas a inserção do processamento desta metodologia no mesmo código, posicionando a renderização da cena tridimensional no módulo para renderização directa para textura. Este método permite, assim, manter o processo de renderização original e somente aplicar o efeito de profundidade sobre a imagem renderizada.

8.4 Controlo em Tempo Real

Com o objectivo de testar, da melhor forma possível, a aplicação da proposta para implementação do efeito de profundidade de campo, foi construído um simulador

interactivo com uma interface gráfica simples, criada a partir da biblioteca GLUI [Bad08], permitindo realizar de forma efectiva diversos testes e análises de resultados sobre todo o trabalho desenvolvido. A interface compreende duas janelas gráficas: a janela de visualização da simulação e uma outra para controlo de propriedades de simulação, como é bem visível na Figura 6.

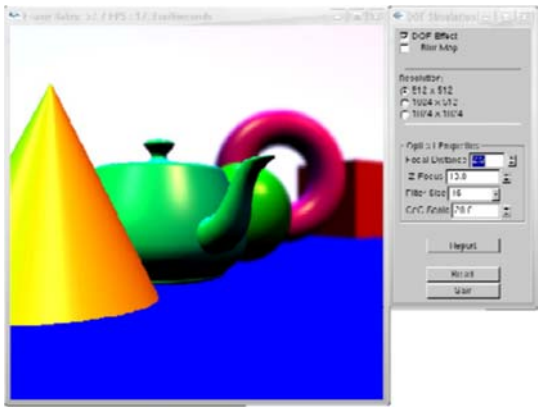

Figura 6: Interface gráfica do simulador.

Figura 7: Mapa de processamento de filtragem.

Oferece-se ao utilizador, desta forma, um conjunto de opções para poder controlar as propriedades ópticas da simulação da profundidade de campo, considerando-se como principais funcionalidades as seguintes:

- Visualização da cena 3D com ou sem efeito de profundidade de campo;
- Visualização do mapa de processamento em profundidade. Zonas a preto correspondem a áreas totalmente nítidas, incluídas no plano focado e, por isso, com menor processamento de filtragem. As restantes zonas serão tanto mais claras quanto mais afastadas do plano focado (Figura 7, nas condições da cena da Figura 6);
- Escolha da resolução da janela principal;
- Definição da distância focal e do plano focado;
- Definição do tamanho do filtro a utilizar nos cálculos de profundidade de campo, correspondendo ao tamanho máximo do círculo de confusão calculado para cada pixel.
- Variação de escala do círculo de confusão, permitindo obter efeitos de profundidade peculiares e personalizados ao gosto do utilizador, embora não precisos em termos das leis da óptica.

9. RESULTADOS

Durante o desenvolvimento dos trabalhos apresentados neste artigo, a profundidade de campo foi integrada em diversas aplicações OpenGL com as respectivas cenas tridimensionais de diferentes características, principalmente ao nível da quantidade, pormenorização e posicionamento de objectos em profundidade, com ou sem animação, e diversificando cores e texturas.

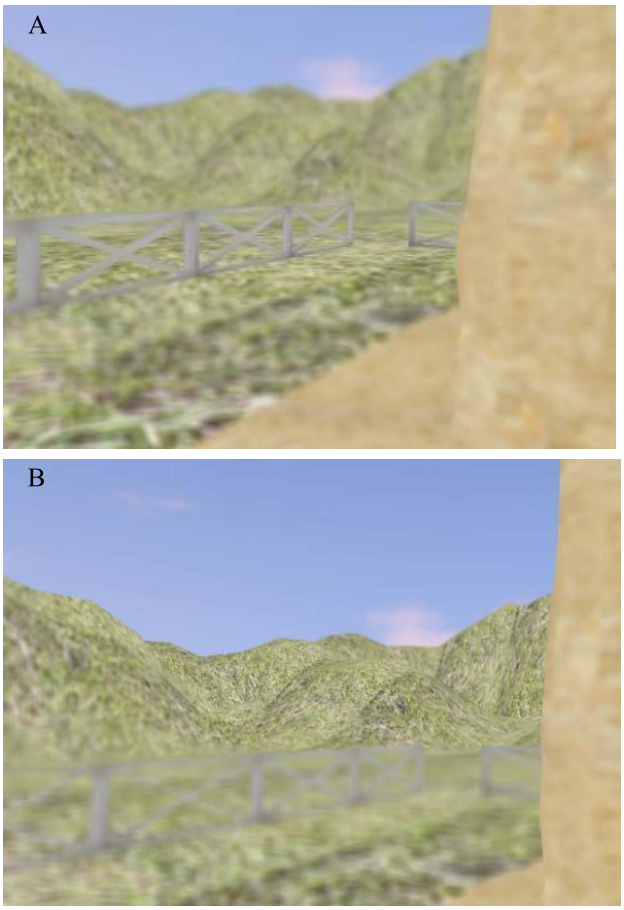

Figura 8: Resultados numa navegação espacial.

Como caso de estudo de adaptação foi utilizada uma aplicação gráfica concebida e gentilmente cedida por Tudor Carean [Car09]. Esta aplicação, ilustrada pelas imagens da Figura 8, apresenta-se com relativo grau de complexidade em termos do modelo 3D, consistindo numas ruínas entre montanhas, envolvendo gravidade, colisões, sombras realistas e livre controlo da navegação por teclado e rato, em tudo idêntico ao estilo de jogo "Primeira-Pessoa". Para incorporar o efeito de profundidade de campo, foi introduzido na aplicação o código correspondente aos passos da metodologia apresentada. Refira-se que, para esta aplicação específica, o plano focado foi automaticamente determinado pelo centro da imagem, ou seja, os objectos focados são aqueles para os quais a câmara está apontada. Na Figura 8-A isso corresponde à

vedação, encontrando-se desfocados tanto os objectos mais próximos como as montanhas ao fundo. Mas orientando a câmara um pouco mais para cima, como na Figura 8-B, já as montanhas passarão a estar focadas. Para o efeito em causa se tornar bem notório nas imagens, os parâmetros foram escolhidos de forma a produzir-se uma profundidade de campo relativamente reduzida.

Analisaram-se os resultados segundo as perspectivas da qualidade, do desempenho e da possibilidade de integração da metodologia em si.

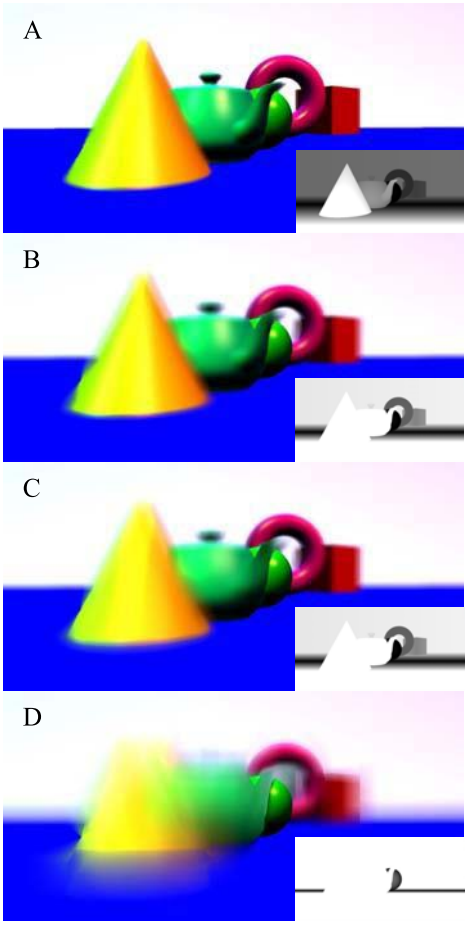

Figura 9: Efeitos de profundidade de campo consoante a parametrização.

Considerando a qualidade do efeito de profundidade de campo sobre as cenas tridimensionais, e desde que os círculos de confusão sejam de tamanho reduzido, esta equipara-se perfeitamente à qualidade do mesmo efeito em Fotografia, indo ao encontro dos objectivos propostos inicialmente. A Figura 9 mostra ainda alguns dos efeitos que a metodologia aqui apresentada é capaz de recriar sobre uma cena desprovida da noção de profundidade. Sobre a mesma cena 3D e mantendo sempre o mesmo plano focado, neste caso correspondendo à esfera por detrás do bule, foi captada uma sequência de imagens com distintos efeitos de profundidade resultantes da variação dos diversos parâmetros disponíveis para controlo interactivo da simulação da profundidade de campo, vistos na secção anterior. A Figura 9-A apresenta um efeito de profundidade suave, possuindo valores baixos de distância focal e de tamanho de círculo de confusão. Um grande aumento da distância focal provoca uma desfocagem proporcional dos planos não focados, como se pode ver na Figura 9-B. Se apenas for aumentado o tamanho do filtro, mais correctos serão os respectivos resultados, verificando-se isto mesmo com a transição da imagem B para C da Figura 9. O parâmetro para variação de escala dos círculos de confusão torna possível a obtenção de efeitos específicos ao contexto da aplicação onde efeitos como o visualizado na Figura 9-D pudessem ser permitidos.

Os testes de desempenho realizados focaram o grau de rapidez de execução, com a alteração do plano de focagem definido pelo utilizador, para se poder concluir se a aplicação está efectivamente a executar o efeito de profundidade correctamente adaptado à rapidez da percepção humana. Verificaram-se desempenhos que se aceitam perfeitamente em qualquer aplicação de tempo real, fruto da combinação da execução de *shaders* e da renderização por FBO. Apenas o tamanho do filtro e a profundidade da cena 3D podem afectar o desempenho global das aplicações. O tamanho do filtro limita o cálculo dos círculos de confusão e, assim, quanto maior for esse valor melhor e mais correcto será o resultado final. Em contrapartida, o processamento será aumentado e o desempenho diminuirá.

Por fim, a animação de objectos presente nos ambientes virtuais revelou-se com uma excelente qualidade, não sendo notada qualquer espécie de inconsistência visual no sucessivo redesenhar das várias imagens afixadas por segundo.

	Simulação da Profundidade de Campo			
	Desligada		Ligada	
Resolução	FPS	Tempo/frame	FPS	Tempo/frame
512x512	96	10,3	58	17,1
1024x512	54	18,5	31	32,1
1024x1024	26	37,6	16	62,9

Tabela 1: Desempenho relativo à cena da Figura 6.

Os resultados apresentados na Tabela 1 foram obtidos num computador com processador Intel 1.86 GHz Core2Duo, 2 GB RAM e uma placa gráfica NVIDIA GeForce 8500GT, executando o sistema operativo Windows XP.

10. CONCLUSÕES E TRABALHO FUTURO

Com o trabalho apresentado neste artigo comprova-se que se torna possível, através de técnicas e tecnologias gráficas recentes, a apresentação de uma metodologia para simulação da profundidade de campo capaz de ser integrada por qualquer programador nas suas aplicações gráficas interactivas.

A metodologia apresentada baseou-se no algoritmo de filtragem de Zhou *et al.* [Zho07]. Foi melhorada e optimizada a sua implementação por via de sequências de processamento de imagem com técnicas de *shading* em GPU, segundo a linguagem GLSL, e renderizações direc-

tamente para texturas por meio de objectos FBO, juntamente com a utilização do canal alfa.

Desta forma, é dado um contributo na área da Computação Gráfica pela implementação de uma metodologia capaz de recriar a noção de profundidade de campo em ambientes gráficos virtuais. A sua qualidade, a eficácia e a facilidade de integração permitem naturais desenvolvimentos em áreas como Realidade Virtual ou Realidade Aumentada, quando pretendidas visualizações verdadeiramente foto-realistas.

A qualidade do efeito da profundidade de campo poderia ser melhorada com um trabalho futuro dirigido no sentido de se tratar a simulação de transparências e as situações de oclusão parcial nas cenas 3D, ainda que com um natural acréscimo no desempenho global.

Em alternativa às tecnologias utilizadas, seria interessante implementar totalmente a profundidade de campo usando outras soluções gráficas, como é o caso de CUDA da NVIDIA [CUD08] ou Compute Shader [Boy08] da Microsoft. CUDA é hoje uma referência em termos de GPGPU (ou seja, uso de GPU para programação geral e não exclusivamente gráfica), e Compute Shader é anunciado para breve como uma das grandes inovações na utilização das capacidades de GPU, fazendo parte integrante do pacote DirectX 11.

Também seria um desafio futuro adaptar a metodologia apresentada neste artigo à técnica de interpolação de mipmaps filtrados anisotropicamente, evoluindo talvez para a geração de um novo algoritmo para simulação do efeito de profundidade. A interpolação de mipmaps e respectivas filtragens anisotrópicas, que bons resultados proporcionaram noutras técnicas, deram já origem a uma das mais correctas e eficientes simulações de profundidade de campo por pós-processamento [Lee08].

11. REFERÊNCIAS

[Bad08] Ali BaderEddin, GLUI Window Template, The Code Project Open License, 31 Jan 2008. <http://www.codeproject.com/KB/openGL/GLUI_Window_Template.aspx>

[Ber04] M. Bertalmio, P. Fort, D. Sanchez-Crespo: Real-time, accurate depth of field using anisotropic diffusion and programmable graphics cards. Em Proceedings of Second International Symposium on 3D data Processing, Visualization and Transmission 2004. p. 767–773, 2004.

[Boy08] Chas Boyd: Direct3D 11 Compute Shader, More Generality for Advanced Techniques. Gamefest 2008. Microsoft Game Technology Conference. 2008.

[Car09] Tudor Carean. 3D Coding Tutorials. 2009. < http://www.3dcodingtutorial.com >

[Coo84] R. Cook, T. Porter, L. Carpenter: Distributed Ray Tracing. ACM SIGGRAPH Computer Graphics 18(3): p. 137–145, 1984.

[CUD08] NVIDIA CUDA Programming Guide 2.1. <http://developer.download.nvidia.com/compute/cuda/2_1/toolkit/docs/NVIDIA_CUDA_Programming_Guide_2.1.pdf>

[Dem04] J. Demers: Depth of Field: A Survey of Techniques. GPU Gems: Programming Techniques, Tips, and Tricks for Real-Time Graphics. R. Fernando. Addison-Wesley. p. 375-390. 2004.

[FBO06] Framebuffer object technical paper. 2006. <http://oss.sgi.com/projects/ogl-sample/registry/EXT/framebuffer_object.txt>

[Hae90] P. Haeberli, K.Akeley: The accumulation buffer: hardware support for high-quality rendering. Computer Graphics (SIGGRAPH'90 Proceedings). Vol. 24 (4). p. 309–318. 1990.

[Ham07] Jr. Earl Hammon: Practical Post-Process Depth of Field. GPU Gems 3. Hubert Nguyen. Addison Wesley Professional. cap. 28. 2007.

[Hil08] S. Hillaire, A. Lecuyer, R. Cozot, G. Casiez: Using an Eye-Tracking System to Improve Camera Motions and Depth-of-Field Blur Effects in Virtual Environments. Virtual Reality Conference, 2008. VR '08. IEEE. p. 47-50. 2008.

[Kas06] M. Kass, A. Lefohn, J. Owens: Interactive Depth of Field Using Simulated Diffusion on a GPU. Relatório técnico. Pixar Animation Studios. 2006.

[Kos01] R. Kosara. Semantic Depth of Field – Using Blur for Focus+Context Visualization. Tese de Doutoramento, Vienna University of Technology. Áustria. p. 14-22. 2001.

[Kri03] J. Krivanek, J. Zara: Fast depth-of-field rendering with surface splatting. In Proceedings of Computer Graphics International, p. 196–201, 2003.

[Lee08] S. Lee, G. J. Kim, S. Choi: Real-Time Depth-of-Field Rendering Using Anisotropically Filtered Mipmap Interpolation. IEEE Transactions on Visualization and Computer Graphics. September/October 2008.

[Pot82] M. Potmesil, L. Chakravarty: Synthetic image generation with a lens and aperture camera model. ACM Transactions on Graphics 1(2): 85–108, 1982.

[Rok96] P. Rokita: Generating depth-of-field effects in virtual reality applications. IEEE Computer Graphics and Applications 16(2): p. 18–21, 1996.

[Ros06] Randi J. Rost: OpenGL® Shading Language, Second Edition. Addison-Wesley. 2006.

[Shr05] D. Shreiner, M. Woo, J. Neider, T. Davis: OpenGL Programming Guide (Fifth Edition). Addison-Wesley, Boston, Massachusetts, 2005.

[Sob03] Marcionílio Barbosa Sobrinho: Tutorial de Utilização de OpenGL. Centro Universitário de Belo Horizonte. 2003.

[Zho07] T. Zhou, J. X. Chen, M. Pullen: Accurate Depth of Field Simulation in Real Time. Computer Graphics Forum. Blackwell Publishing. Vol. 26. p.15-23. 2007.