2.
Conceitos e Trabalhos Relacionados

Este capítulo apresenta uma resenha sobre os elementos básicos, as questões de aceleração e os métodos e técnicas de gerenciamento para renderização fotorealista utilizada na indústria. Nesta resenha, sempre que possível, procura-se fazer uma comparação com as propostas do presente trabalho. De qualquer maneira, todos os conceitos e elementos mencionados neste capítulo são necessários para o melhor entendimento do trabalho realizado.

2.1 Elementos Básicos para Renderização Fotorealista

Na conjuntura da pesquisa realizada, avaliando a área de renderização, tem-se que os requisitos de *rendering* na indústria de cinema são extremamente severos (Christensen et al., 2006):

1. A geometria de cena é muito grande para caber na memória em forma tecelada;
2. Muitas superfícies são *displacement-mapped*;
3. Podem existir milhares de texturas (o que é excessivo para caber na memória em alta resolução) para controlar parâmetros de reflexão e *displacements*;
4. Podem haver milhares de fontes de luzes (pontuais e de área)
5. Toda a iluminação e as características de reflexão de superfície são controladas por shaders complexos e programáveis;
6. *Aliasing* temporal e *aliasing* espacial não são aceitáveis;
7. Imagens são renderizadas em alta resolução, com efeitos visuais (profundidade de foco, *motion blur*, ...);
8. Inter-reflexões devem ser adequadamente consideradas;
9. Sombras devem ser adequadamente simuladas;
10. Oclusão de ambiente (*ambient occlusion*) deve ser gerada.
11. Iluminação Global deve ser gerada quando necessário.

12. As imagens geradas devem ser em formato *HDR* para composição futura com cenas reais.

Os seis últimos requisitos da lista acima podem ser adequadamente tratados com *ray tracing*, desde que o tempo de rendering não seja excessivo. Entretanto, deve-se entender que *ray tracing* de cenas complexas de Cinema/TV digital é muito mais difícil do que as técnicas de scanline usualmente usadas pela indústria. Primeiramente, não pode-se usar *culling*, porque objetos fora do *frustum* podem jogar sombra sobre os objetos visíveis ou ser refletidos por eles. Segundo, mesmo que os raios primários sejam coerentes, os raios secundários (reflexões e sombras) acessarão outras geometrias e texturas de maneira muito pouco coerente; nesse ponto, entra a necessidade de realizar simulações de Monte Carlo para os disparos de raios. Ademais, tanto os raios traçados para inter-reflexões difusas como os traçados para oclusão de ambiente são completamente incoerentes; e isso é necessário para evitar efeitos de *banding* (formação de bandas de cor).

Resultados expressivos com o uso de *ray tracing* na indústria do cinema são muito recentes (Christensen et al., 2006), (Hou et al., 2010), (Guntury & Narayanan, 2010) e (Parker et al, 2013). Os requisitos 8 a 10 foram eficientemente tratados por Christensen et al. (2006) através da inclusão de *ray tracing* no sistema RenderMan da Pixar – o software mais usado pela indústria do cinema. O RenderMan é baseado na arquitetura de Reyes (Cook et al., 1987) que usa *scanlines* e micropolígonos. Christensen et al. (2006) usaram as técnicas de diferencial de raio e de multiresolução conforme trabalho anterior (Christensen et al., 2003). Uma melhora significativa para este tipo de arquitetura pode ser encontrada no trabalho de Zhou et al. (2009) que propuseram um sistema para rendering de Reyes interativo, no entanto, as limitações presentes nos mesmo o tornaram inviável para cenas reais. Quanto ao requisito 7, Hou et al. (2010) apresentaram uma solução para desfocagem e *motion blur*,

usando micropolígonos (o que facilita uma possível integração com o RenderMan).

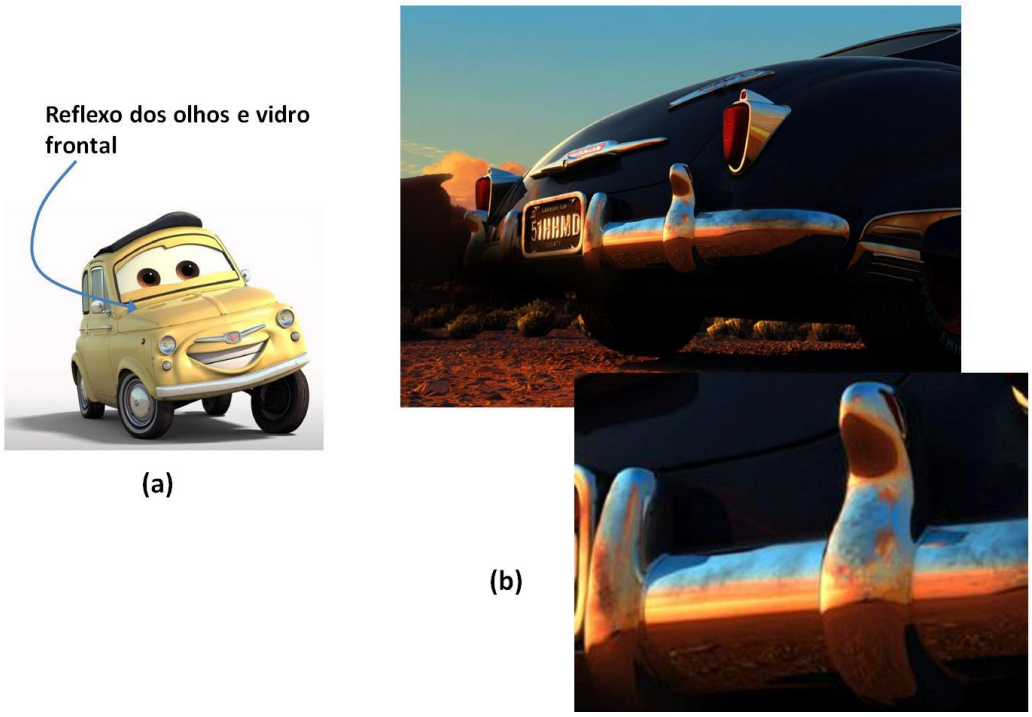


Figura 3: Inter-reflexão com uso de ray tracing no filme “Cars (Pixar, 2006)”: (a) reflexo dos olhos e vidro frontal ; (b) reflexão entre partes cromadas. Figura composta a partir de Christensen et al. (2006).

A figura 3 apresenta duas situações, do filme “Cars (Pixar, 2006)”, onde a inter-reflexão gerada por ray tracing é essencial (o software Pixar RenderMan, padrão no cinema, baseado em scanline rendering, só passou a usar *ray tracing* por ocasião da produção do filme “Cars”, sendo oficialmente lançado em 2007).

Figura 4: Sombras bem definidas e detalhadas geradas por ray tracing, do filme "Cars (Pixar, 2006)" contendo 1000 fontes de luz (figura extraída de Christensen et al. (2006)).

No entanto, quando habilitado, o *ray tracing* do RenderMan causa um enorme aumento do tempo de renderização das cenas, pois toda a cena passa a ser renderizada por *ray tracing*. Outros sistemas como o VRay da Chaos Group (Chaos Group, 2006), são sistemas de renderização que usam como base o *ray tracing* e chegam a ter visualizadores interativos, mas que não são fiéis ao resultado final, assim, servem apenas como referência de iluminação, e, dessa forma, são pouco utilizados pela indústria de cinema.

Para simular diversos efeitos necessários para a composição de imagens reais e virtuais, o modelo de câmera utilizado no cinema deve ser reproduzido na renderização. Assim, o trabalho de Kolb et al (1995) apresenta meios para simular os diversos parâmetros fotográficos de uma câmera, ou seja, lente, entrada da pupila, diafragma (*f-stop*), exposição e sensibilidade *ISO*. Alguns parâmetros são apenas mascarados, como o *ISO*, que interfere na granulação da imagem em regiões de baixa luminosidade.

Observa-se que com o uso desses parâmetros surgem efeitos como profundidade de campo, *motion blur*, distorção radial e outros. O tipo de

obturador também influencia o resultado (mecânico e eletrônico), mas, normalmente não é simulado por gerar efeitos não desejados.

Sombras dão fortes dicas sobre a iluminação e sobre o posicionamento relativo dos objetos. As técnicas de scanline computam sombras usando *shadow maps* o que podem causar os seguintes inconvenientes: problemas de resolução devido à grande quantidade de pequenos detalhes de geometria e problemas de gerenciamento de acervo (*asset management*) para rastrear os arquivos de *shadow map* necessários para considerar as milhares de fontes de luz nas cenas reais. A Figura 4 mostra uma cena do filme “Cars” com sombras bem definidas e detalhadas.

(a) (b)

Figura 5: Oclusão de ambiente gerado por ray tracing em imagens de alta qualidade (figura extraída de Christensen et al. (2006)).

Outro elemento importante é a Oclusão de ambiente, também denominado de “sky lighting”, que é a medida de quanto do céu é bloqueado em qualquer ponto de uma superfície ou, mais precisamente, o quanto de luz alcança um ponto a partir de uma semi-esfera uniformemente acesa envolvendo a cena. Uma imagem de oclusão de ambiente é sempre em tons de cinza e é usualmente referida como uma passada de sujeira (no sentido de adicionar uma aparência “suja” ao que de outra maneira seria excessivamente limpo para ser realista). A oclusão de ambiente é essencial para adicionar detalhes e profundidade às

sombras. A figura 5 ilustra duas imagens de oclusão de ambiente. Wald et al. (2004) desenvolveu um método de realizar *global ilumination* usando *ray tracing* de forma bastante otimizada.

2.2 Aceleração de Renderização em Alta Resolução

Os efeitos visuais em alta resolução (desfocagem, *motion blur*, ...) são normalmente realizados por técnicas de *scanline*. Recentemente, Hou et al. (2010) propuseram um algoritmo de *ray tracing*, rodando em *GPU*, que pode ser integrado a sistemas baseados em micropolígonos (e.g. RenderMan). A Figura 1b, ilustra o resultado desta nova técnica.

Christensen (2008) propôs um método baseado em nuvens de pontos para computar iluminação global difusa (*color bleeding*), onde os *surfels* na nuvem de pontos são organizados em uma *octree*. O método de Christensen (2008) chega a ser 10 vezes mais rápido do que *ray tracing*. No entanto, para cenas animadas sofre com problemas de coerência de raios, causando artefatos de iluminação, sendo raramente usado.

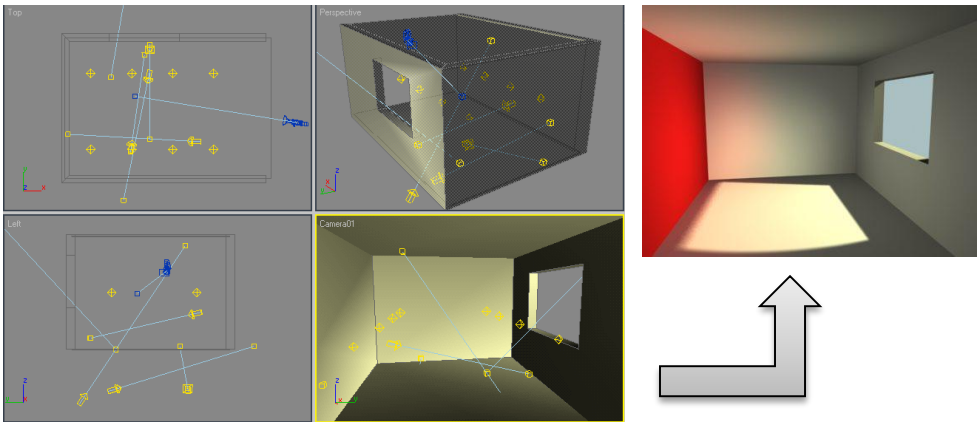
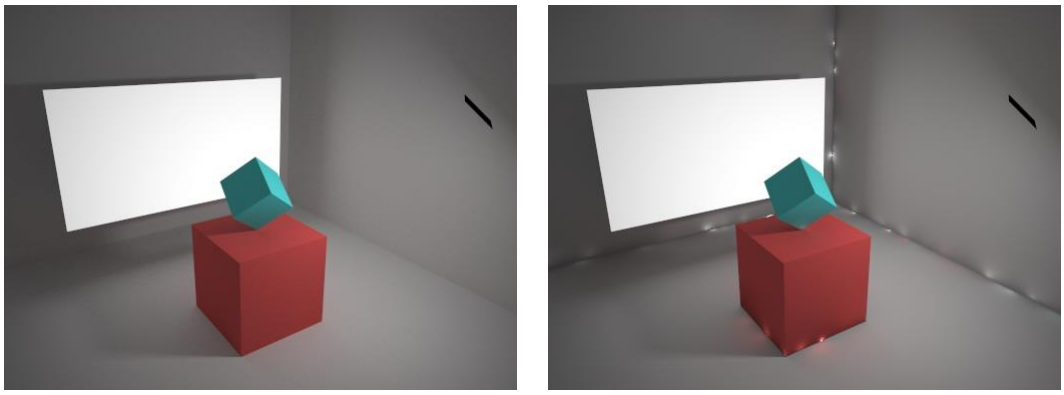


Figura 6: Exemplo da utilização do *Fakeosity* (exemplo produzido)

Dammertz et. al (2010) e Kollig & Keller (2004) apresentaram outra técnica para iluminação baseada em pontos que utiliza como base uma antiga técnica artística de iluminação, o "*fakeosity*" (Figura 6), que

consiste em criar luzes de propagação e utilizar uma técnica de *scanline* convencional. O *Virtual Point Light* é um algoritmo de geração automática de luzes que representem o rebatimento, ou raios secundários de luzes principais. A grande vantagem dessa técnica é que a velocidade de renderização é altíssima, no entanto, o método além de gerar uma iluminação global não muito precisa (mesmo com muitas luzes virtuais), não consegue reproduzir fenômenos como a refração, além de gerar influências errôneas de luzes. A Figura 7 ilustra esse resultado.



a) b)

Figura 7: Comparação entre o resultado do *RayTracing* e *Virtual Point Light* a) resultado obtido com *ray tracing* com iluminação global. b) resultado obtido com VPL. (Imagens extraídas de Walter et al (2012))

A fim de melhorar o resultado do *VPL*, Walter et al (2012) apresentou uma melhoria significativa da técnica usando *Lightcuts*, que são atenuadores e barreiras para as luzes geradas no *VPL*, removendo influencias errôneas melhorando o resultado para cenas animadas. No entanto, isso produz um custo de renderização e ainda não consegue reproduzir efeitos como a refração e atmosfera.

Novàk et al (2012) apresentou uma outra evolução sobre a *VPL*, permitindo a simulação de efeitos atmosféricos e de refração, mesmo em meios difusos. A técnica de *Virtual Ray Light* consiste em uma ampliação da *VPL*, criando caminhos entres as fontes luminosas geradas pelo *VPL*.

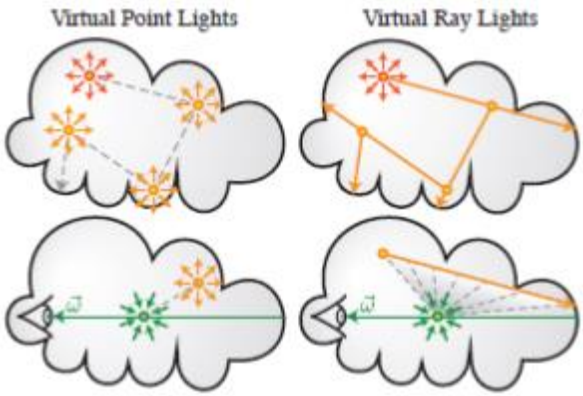


Figura 8: Virtual Point Light simula a influencia de uma luz através de pontos adicionais de luz, já o Virtual Ray Light utiliza uma interpolação de todo o trajeto entre os pontos para simular a interação, sendo assim, muito mais denso. (imagem de Novàk et al (2012))

Nesse contexto, percebe-se que existem diversas formas se obter resultados semelhantes ao *ray tracing* usando técnicas menos onerosas. Contudo, a escolha da técnica é confusa e depende de diversos elementos da cena, os quais o usuário não dispõe de meios para julgar e escolher.

Ainda, a renderização de animações exige que haja uma coerência de raios nas imagens, a fim de evitar artefatos como sombras que centelham, e que realcem a natureza virtual da imagem. Nesse sentido, Cohen & Greenberg (1985) já possuía meios para gerar uma forma de précomputar a iluminação através da solução da radiosidade de uma cena, assim, mantendo a coerência. No entanto, isso é insuficiente para as necessidades atuais de imagem. Krivanek et al (2009) apresenta uma metodologia de *caching* para irradiância, tornando mais eficiente a renderização e reduzindo os ruídos das simulações. Krivanek et al (2009) discursa sobre as diversas técnicas de caching de irradiância, apresentando a metodologia necessária para gerar estruturas reduzidas e eficientes.

Com o crescente aumento do poder de computação das *GPUs*, Szirmay-Kalos et al (2008) descreve técnicas sobre a iluminação global usando *GPU*, usando *ray tracing* e simplificações para rasterização.

O *Ray tracing* em tempo real ou interativo para cenas dinâmicas pode se tornar viável não apenas por causa do uso de hardware com processadores paralelos (*CPU* e *GPU*) de alto desempenho, mas principalmente por causa do tipo de estrutura de dados e da estratégia de atualização dos mesmos. Um apanhado do estado-da-arte em *ray tracing* em tempo real para cenas dinâmicas pode ser encontrado em (Wald et al., 2007).

Encontrar o objeto atingido por um raio é essencialmente um problema de busca que, por sua vez, depende da estrutura de dados usada para acelerar esta busca. As estruturas de aceleração mais utilizadas são as seguintes[1]:

Estruturas por Subdivisão Espacial:

- **Grade Regular ou Grade Uniforme**
  - Consiste em uma subdivisão regular do espaço, ou seja, com seguimentos igualmente espaçados, formando uma matriz, na qual cada célula representa uma porção deste (figura. 9a). (Purcell et. al, 2002), (Ivson et. al, 2009), (Ivson, P., 2009), (Wald et al., 2003)

- **Grade Hierárquica ou Grade Adaptativa**
  - Consiste em uma ampliação da grade regular, na qual se subdivide, recursivamente, as células de acordo com um critério heurístico. Essa estrutura difere da *octree*, pois uma célula subdividida pode possuir muito mais que oito células filhas. (Reinhard et al., 2000), (Jevans, 1989), (Klimaszewski, 1995), (Klimaszewski et al., 1997)

- **kd-trees**

[1] Para uma análise de métodos de subdivisão, recomenda-se (Havram, 2007), (Nam e Sussman, 2004) e (Akenine-Möller e Haines, 2008, Cap. 9).

- A *kd-tree* é uma estrutura hierárquica similar a uma árvore binária de pesquisa. No entanto, cada nível da *kd-tree* se ramifica baseado numa pesquisa de chave para o nível, chamado discriminador. Esse discriminador subdivide o espaço em uma determinada dimensão que, no caso do *Ray tracing*, pode ser um dos eixos de coordenadas (figura. 9b). (Hou et al., 2010), (Shevtsov et al., 2007), (Foley et al., 2005), (Horn et al., 2007), (Popov et al., 2007), (Zhou et al., 2008)

- **Octrees**
  - A *Octree* é uma árvore, na qual cada nó que não é folha possui interligação com mais outros oitos nós da estrutura de dados. A *Octree* é uma técnica de modelagem bastante comum no uso de tratamento de colisões, como o caso do *ray tracing*. (Madeira et al, 2009), (Machado, 2010) (figura. 9c).

(a) (b) (c)

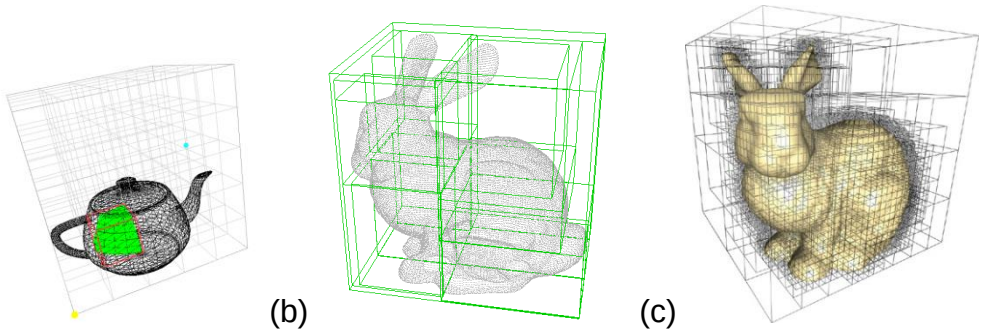


Figura 9: Estruturas Hierárquicas:
a) Grade Regular (Fernando, 2004)
b) Kd-Tree (exemplo produzido)
c) Octree (Pharr, 2005)

Estruturas por Hierarquia de Objetos:

- Bounding Volume Hierarchies (BVH) (Wald, I., Boulos, S. et al., 2007), (Günther et al., 2007)
  - As *BVHs* são hierarquias de volumes envolventes, de acordo com o próprio nome, ou seja, cria-se uma estrutura na qual a cada nível possui um volume envolvente que engloba outros subvolumes. Os volumes envolventes mais utilizados são:
    - *Bounding Sphere*
    - *Bounding Elipsis*
    - *Axis Aligned Bounding Box – AABB*
    - *Oriented Bounding Box – OBB*
    - *Convex-Hull*

Além disso, para cenas dinâmicas, as estratégias de atualização podem ou não alterar a **topologia** e podem envolver atualizações **completas** ou **parciais**. Uma atualização completa que também altera a topologia significa reconstruir completamente a estrutura a partir do zero a cada frame. Esta opção lida muito bem com cenas muito dinâmicas, mas tem o custo elevado de reconstrução. Ivson (2009) descreve uma metodologia de reconstrução rápida de grades uniformes para cenas dinâmicas.

Por outro lado, a atualização parcial, tipicamente, concentra-se na coerência espacial (Havram, 2000), (Wald et al., 2006) da cena. Outro ponto importante de investigação é a exploração de coerência temporal (entre frames), tentando estabelecer uma metodologia de Bounding Volumes 4D, ou seja, considerando a dimensão tempo para realizar atualizações parciais.

A estrutura de aceleração usada por Hou et al. (2010) é um *BVH* baseado em hiper-trapézios OBBs no 4D que se projetam no 3D como *Oriented Bounding Boxes (OBBs).* Este tipo de solução é explorada por este trabalho através das bibliotecas Intel Embree (Intel, 2012) (Wald et al, 2014) e Nvidia Optix (NVidia, 2011) (Parker et al, 2010).

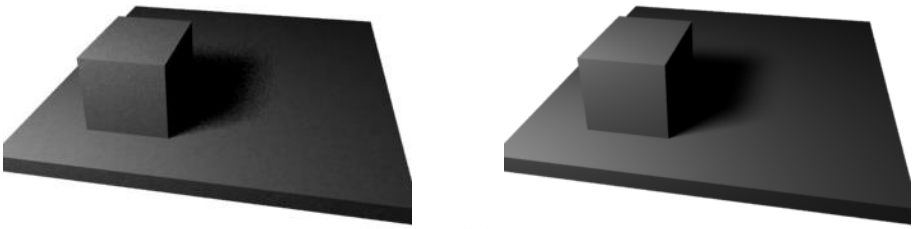
Soluções híbridas com mais de um tipo de estrutura e mais de um tipo de estratégia de atualização representam uma linha de trabalho e investigação da presente tese, de forma geral, a combinação de técnicas para elementos estáticos e dinâmicos é fundamental para otimizar os ganhos de desempenho.

Outro aspecto importante é a questão da multiresolução, visto que as cenas de qualidade cinematográfica envolvem muitos objetos, e cada objeto tem uma quantidade de triângulos muito além do encontrado em games (ordem de milhões). O uso de metodologias que usam nível de detalhe para definir a malha dependendo da distância de visão (Luebke et

al., 2002) também são importantes, mas não foram empregados neste trabalho.

Além disso, outro ponto importante é a questão do *streaming* (carregamento dinâmico) de texturas e dados para a placa de vídeo e para a memória principal, visto que, as placas atuais ainda possuem pouca memória RAM disponível, em relação ao utilizado pela indústria de cinema, chegando a cerca de 8 Gbytes, quando sistemas de *ray tracing* em CPU utilizam cerca de 128 Gbytes de memória. Isso é necessário, pois, a maioria das texturas utilizadas em produção possuem resolução de 10 Mpixels, para texturas geradas por artista, e de 25 Mpixels, para texturas captadas por câmeras digitais. Ainda, uma cena pode possuir muitas texturas. Dessa forma, a utilização de streaming e de utilização de texturas compactadas na placa de vídeos, como exposto por Akenine-Möller et al.(2008) é fundamental.

Ainda, para o aumento do realismo de uma cena são necessários diversos efeitos sofisticados que, entretanto, comprometem o desempenho do sistema em termos de tempo de processamento. Esses efeitos, além de muitas vezes serem fisicamente baseados (Pharr & Humphreys, 2010), e.g., desfocagem e cáusticas, eles devem possuir um rigoroso sistema de *antialiasing*.



(a) (b)

Figura 10: Sombras suaves utilizando amostragem de Monte Carlo (a) baixa amostragem (b) alta amostragem (imagens produzidas)

Outro ponto de alto processamento são as luzes, pois essas, além de gerarem sombras, normalmente, não são consideradas pontuais, ou

seja, são luzes com área (*area light*). Portanto, geram sombras mais suaves (*soft shadows*) (figura 10). Esse comportamento é simulado através de sistema de amostragem de Monte Carlo (Crane, 2006), (Veach, E., 1997), (Talbot, J. F., 2005), (J., H. W. et al, 2003), gerando muitos raios, os quais devem se manter coerentes entre *frames*, caso contrário, artefatos como *flickering*, que é uma variação brusca de amostragem entre *frames*, podem ocorrer. Isso pode ser corrigido com o aumento da amostragem (aumentando o custo) ou o uso de *cache* (*irradiance caching*). Outra abordagem é apresentada por Pharr & Humphreys (2010), num estudo sobre a renderização baseada em física usando *ray tracing* e *path tracing*. Esse amplo estudo permite entender o processo de simulação física da luz e a propagação no meio, no entanto, o consumo de processamento atingido é incompatível com o aceito em produções reais.

No que tange aos desafios apresentados na presente tese, nem *CPUs* e tão pouco *GPUs* podem resolver tais desafios, se usados de forma isolada. Sendo assim, tendo em vista a grande diferença de arquitetura entre processador e placa de vídeo, utilizar os recursos de cada um deles de forma inteligente, ou seja, utilizando o recurso para realizar funções na qual o mesmo possui uma arquitetura mais adequada é a solução mais viável. Temos por exemplo as *GPUs* realizando tarefas extremamente paralelas (*ambient occlusion*) e *CPUs* realizando tarefas com alto grau de recursão (cáusticas).

Diversos trabalhos abordam o uso híbrido de *GPU/CPU* como forma de ampliar a capacidade de renderização de cenas complexas, assim, Guntury (2010) explora a *GPU* e a *CPU* para a construção de *Kd-Trees* para renderização usando *Ray tracing*, no entanto, a técnica é muito limitada para cenas animadas devido ao custo de criação da *Kd-Tree* e a dificuldade de se reaproveitar uma mesma estrutura entre quadros. Roccia et al (2012) também explora o uso de *Kd-tree*, no

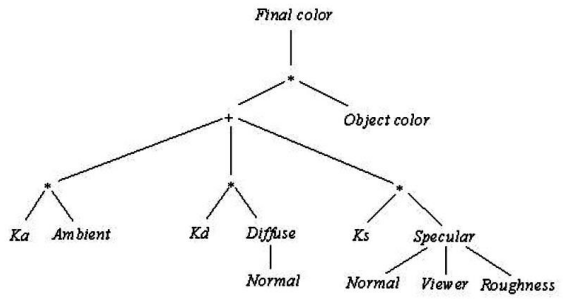
entanto, os resultados ainda não são suficientes para o uso em Cinema/TV, pois os limites de dados não comportam a demanda exigida.

Ainda na área de aceleração de cenas animadas, Garanzha (2011) e Pantaleoni (2010) apresentam diversas técnicas para otimizar a renderização usando hierarquias de volumes envolventes leves (*HLBVH*) que se adaptam entre quadros, reduzindo o custo de reconstrução e funcionando adequadamente para animações.

2.3 Metodologia e Gerenciamento de Renderização

A metodologia de renderização ainda é uma área pouco explorada e não apresenta muitos trabalhos correlatos, no entanto, SABINO et al (2012) apresenta uma técnica de renderização híbrida usando *Ray tracing* (com 1 nível de raio secundário) combinada com rasterização de *GPU* (OpenGL) para jogos em tempo real (*fps* > 50). Para isso, a cena é filtrada e os objetos que necessitam do uso de *ray trace* são separados e renderizados sob o resultado da rasterização, dessa forma, a composição das técnicas é feita pela soma das imagens resultantes, sendo o *ray tracing* predominante.

Assim, em uma solução multitécnica, a descrição dos materiais deve ser feita de forma genérica, pois essa deve ser compatível com *CPU*, *GPU* e Cloud, diferente do apresentado por SABINO et al (2012). Uma forma adequada para descrever materiais é o uso de uma linguagem própria, e uma das primeiras linguagens de *shading* foi o *RenderMan Shading Language* da Pixar, que utilizou a ideia de *shade trees* de Robert Cook (1984), cuja finalidade é organizar operações de *shading* (definidas como *atoms*) como nós em uma estrutura de árvore. Nesta estrutura, as folhas são os dados de entrada, os nós não-folha são operações simples de *shading* e a estrutura é uma representação do código, como exemplificado na figura 11.




```
surface
plastic (float Ka=1, Kd=0.5, Ks=0.5,Roughness=0.1;)
{
   point Normal=faceforward(normalize(N),I);
   point Viewer = normalize(-I);
   Oi=Os;
   Ci=Os*(Cs*(Ka*ambient()+Kd*diffuse(Normal))+
      Ks*specular(Normal,Viewer,Rougness));
}
```

Figura 11: Shade tree e código correspondente na RenderMan (Pixar, 1970)

As atuais linguagens de *shading* não possuem mecanismos próprios de abstração que permitam encapsulamento, modularidade e abstração (McGuire et al, 2006). Tal tarefa é delegada aos editores visuais de *shade trees*.

Cook (1984) propôs uma organização para os cálculos de *shading* na forma de uma árvore, cujo nó raiz é a cor final das operações, representadas por nós na árvore. Cada nó produz um ou mais parâmetros de aparência como saída e recebe zero ou mais parâmetros de aparência como entrada. Isso permite com que shade trees sejam combinadas para formar outras shade trees, usando nós raízes como entradas de aparência para nós de operação de outras árvores. Os cálculos são realizados determinando as dependências dos nós a partir do nó raiz (percorrimento no grafo em pós-ordem).

Dessa forma, ao invés de tentar descrever qualquer superfície possível com uma única equação, o *shader* orquestra um conjunto de operações básicas, como produto e normalização de vetores, organizados em uma árvore, para permitir uma maior customização. A saída final da *shade tree* é a exitância, isto é, a densidade do fluxo luminoso emitido por uma superfície luminosa ou iluminada, a cor e a intensidade da luz que saem da superfície descrita pelo *shader*.

Árvores de iluminação (*light trees*) são separadas das *shade trees*, de forma que cada uma possa ser inserida em diferentes *shade trees*. Os cálculos dos diferentes tipos de iluminação (direcionada, ambiente, etc.) são tão específicos para o tipo particular de fonte de luz que são isolados do resto do *shading*, comunicando-se somente através de parâmetros de aparência. As *light trees* não foram implementadas no sistema desenvolvido, apenas luzes pontuais e de área retangular foram utilizadas.

A vantagem imediata das *shade trees* sobre escrever o *shader* diretamente em código-fonte é que essas encorajam experimentação, como mencionado por McGuire et al. (2006), e são melhor recebidas por não-programadores. Dessa maneira, cabe aos programadores construir os blocos básicos da *shade tree*, os quais os não programadores combinam visualmente, e não textualmente, para construir os *shaders*.

A primeira implementação da *shade tree* de Cook foi a linguagem RenderMan da Pixar (Akenine-Möller, Haines &Hoffman, 2008), ainda em uso.

O conceito de *shade trees* foi posteriormente reabordado por McGuire et al. (2006) em um modelo de abstração (*abstract shade trees*) que visava corrigir alguns dos problemas das *shade trees*: a falta de abstração dificultava os artistas a criarem novos efeitos gráficos sem ter que aprender a programar e as *shade trees* estavam crescendo em complexidade, tornando-se verdadeiros labirintos.

Foley & Hanrahan (2011) propõe uma metodologia de composição de *shaders* para permitir uma melhor construção das *shade trees*. O conceito é elaborado de forma ao sistema se encarregar em criar os *shaders* finais do processo, assim, o usuário descreve de forma textual ou gráfica o que o material deve fazer e o compilador se encarrega de produzir os diversos shaders necessários, desde tecelamento até *shaders*

de fragmento. Esse trabalho serve de base para o sistema de *shaders*, no entanto, o objetivo é criar um modelo base que sirva para as diversas técnicas empregadas, e não somente para a rasterização. A figura 12 apresenta o sistema de criação.

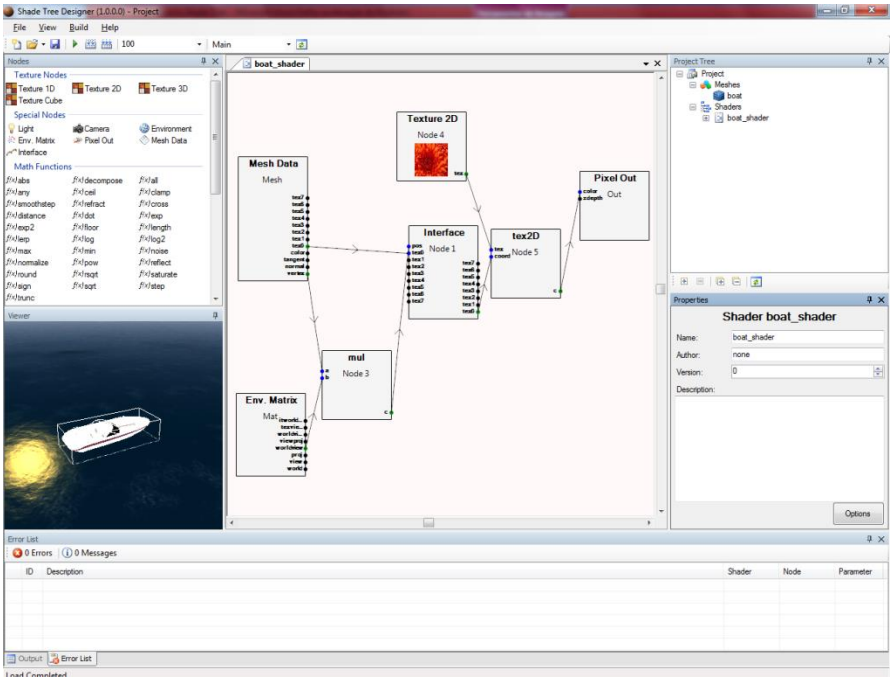


Figura 12: Sistema de criação de Shade Trees desenvolvido nesta pesquisa

Outro aspecto importante é a forma de representação dos elementos que compõem a cena. Essa representação pode ser através de descrições paramétricas, malhas triangulares, nuvens de pontos, *voxels* e outros. Cada elemento possui qualidades e objetivos próprios. Na renderização para cinema, as estruturas mais usadas são as malhas triangulares, no entanto, nuvens de pontos e *voxels* são extremamente usadas para renderização e simulação de partículas e fogo/fumaça. Nesse trabalho, o objeto de estudo se concentrará nas malhas triangulares.

Ainda, com a melhoria dos sensores e capacidade de processamento das câmeras modernas, o espaço de cor é cada vez mais explorado, a figura 12 mostra a evolução e o estado da arte em termos de

cobertura de *gamut*. Nela observa-se que os televisores CRT e a grande parte dos televisores atuais (norma ITU-709) cobrem apenas uma pequena parte do espectro, enquanto as câmeras modernas como a Sony F65 conseguem uma cobertura de cerca de 80% do espectro visível (figura 13).

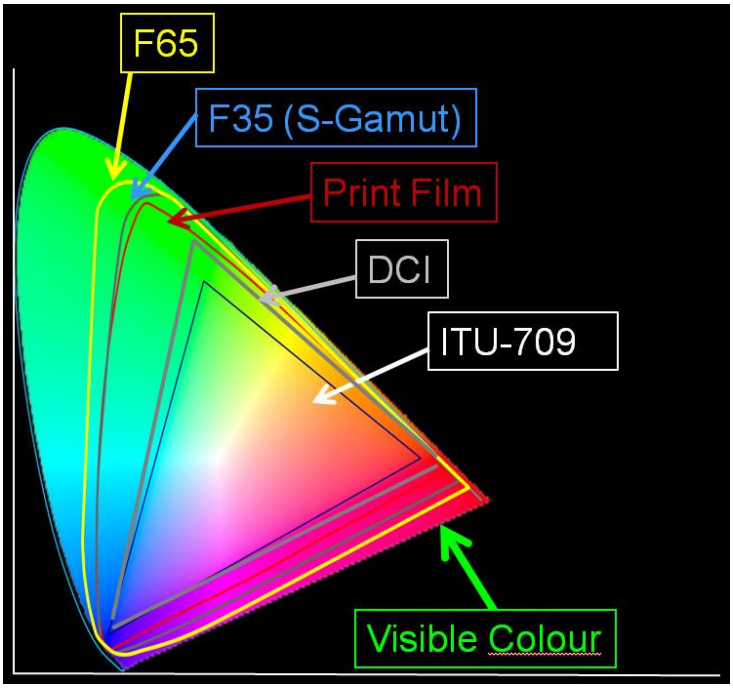


Figura 13: Espectro de cor visível e o gamut de câmeras e televisores (Gaggioni et al, 2012).

Tendo em vista esse cenário, a renderização precisa ser mais precisa ainda para que seja possível combinar, ou compor, cenas com elementos virtuais e elementos reais.

Esse desafio reside no espaço de cor e a latitude dos dados representados. Uma câmera como a Sony F65 chega a oferecer 16 bits por canal em resolução 8K, e ainda utiliza remapeamentos nos espaços de cor através de curvas logarítmicas como a S-Log e S-Log2 (Gaggioni et al, 2012).

Ridolfi (2012) apresenta a utilização de um espaço de cor alternativo, usando o CIELAB, que possui diversos benefícios no tratamento das imagens, permitindo um melhor mapeamento das cores dentro do espaço de cor de uma determinada câmera. Porém, a maioria dos renderizadores utiliza o espaço de cor RGB, dessa forma, este trabalho o suporta.

Outro aspecto pouco explorado é a forma de processamento de frames em sistema multiprocessados. Normalmente, a paralelização utilizada é alocando cada processador em um *pixel*, ou em uma região retangular fixa. Molnar et al (1994) descreve um conjunto de técnicas para renderização paralela. Allard & Raffin (2005) apresentam uma forma alternativa de paralelização usando como parâmetro os *shaders* da cena, assim, a paralelização segue por material. Abraham et al (2004) descreve uma forma não fixa de subdivisão de renderização, no entanto, no contexto de renderização em *realtime*.

Todas essas técnicas se concentram em ambientes *realtime* e com a rasterização por *hardware*, no entanto, em cenas de alta complexidade para cinema, não foi encontrado um estudo que altere a forma de subdivisão local (dada dentro de um *frame* ou *subframe*) e muito menos no âmbito da animação, considerando que um frame em renderização pode não possui dados do *frame* anterior.

Por fim, o processo de gerenciar a renderização de uma cena é um dos tópicos mais importantes na indústria de cinema. Tal fato levou a um *software* receber o *Oscar* de *Scientific & Technical Awards* (Pixar, 2011). No entanto, mesmo um excelente gerenciador não consegue aproveitar toda a capacidade de processamento de uma *Render Farm*. Apesar da grande quantidade de sistemas que atendem essa demanda, todos se concentram em processos de divisão de tarefas, tendo como granularidade o frame e a desconsideração das características temporais da renderização. Assim, o uso de soluções com o OpenMPI (Squyres,

2008) é grande, por realizar o trabalho de forma similar aos industriais Tractor (Pixar, 2010), Qube! (PipelineFX, 2012), Backburner (Autodesk, 2003) e Deadline (Thinkbox, 2007).