Universidade Federal
do Rio de Janeiro
Escola Politécnica

Geração de banco de dados de funções plenópticas amostradas utilizando *physically based rendering*

Projeto de Graduação apresentado ao Curso de Engenharia Eletrônica e de Computação da Escola Politécnica, Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Engenheiro.

Orientador:
Eduardo Antônio Barros da Silva

DEL

Agosto de 2013

Universidade Federal do Rio de Janeiro

Escola Politécnica

Departamento de Eletrônica e de Computação

Geração de banco de dados de funções plenópticas amostradas utilizando *physically based rendering.*

Autor:

Luiz Gustavo Cardoso Tavares

Orientador:

Prof. Eduardo Antônio Barros da Silva, Ph. D.

Examinador:

Prof. Gelson Vieira Mendonça, Ph. D.

Examinador:

Cassius Duque Estrada, M. Sc.

DEL

Agosto de 2013

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
Escola Politécnica - Departamento de Eletrônica e de Computação
Centro de Tecnologia, bloco H, sala H-217, Cidade Universitária
Rio de Janeiro - RJ CEP 21949-900

Tavares, Luiz Gustavo Cardoso

Geração de banco de dados de funções plenópticas amostradas utilizando *physically based rendering*.

41 páginas

Monografia (Graduação) - Escola Politécnica da Universidade Federal do Rio de Janeiro. Departamento de Engenharia Eletrônica e de Computação.

1. Funções plenópticas

2. *Ray-Tracing*

I. Universidade Federal do Rio de Janeiro. Escola Politécnica. Departamento de Engenharia Eletrônica e de Computação.

Aos meus pais e amigos

AGRADECIMENTOS

Agradeço primeiramente à Deus por guiar meus caminhos e me dar forças para, por muitas vezes escolher as opções mais difíceis, porém mais engrandecedoras.

Aos meus pais pelo apoio incondicional, pela paciência e pelos ensinamentos de uma vida toda. A ajuda, amor e sacrifício de vocês me trouxeram até aqui e à vocês dedico todo o crédito deste trabalho, pois sem os pilares fundamentais não é possível edificar nada sólido.

Ao professor Eduardo pela confiança e por toda a paciência. Sua orientação foi (e continuará sendo) essencial para o meu crescimento não somente durante o curso, mas também para todas as etapas que ainda virão. É uma grande honra conhecê-lo e tê-lo como orientador.

A todos os professores, amigos e colegas do Laboratório de Processamento de Sinais (LPS) em especial a Felipe Ribeiro, Luís Lucas, Jonathan Gois, Tadeu Ferreira, Andreas Ellmauthaler, Maurício Costa por me ensinar o *clearpage*, Gabriel Matos, Markus Lima e Renam Castro. Um muito obrigado ao José Fernando por todo o auxílio e prestatividade em esclarecimento diversos, dentro do escopo do projeto ou não. É um grande prazer poder trabalhar e partilhar alegrias com vocês a cada dia. Não posso me imaginar trabalhando com colegas e amigos melhores.

Aos amigos Felipe Barboza, Leonardo Abdalla e Yoann Léger por cada palavra de força e auxílio. Cada dia ficou mais fácil com a alegria e amizade de vocês. Sem amigos com os quais podemos contar, somos ninguém.

Aos professores do Departamento de Engenharia Eletrônica, por além de instruir, tornar este curso prazeroso e me fazer cada dia mais gostar do que faço.

A todos que não agradeci nesta folha por falta de espaço, haverá espaço para vocês nos trabalhos futuros.

Por último, porém não menos importante, a você leitor que, ao ler este texto, valoriza todo o nosso trabalho.

RESUMO

Este projeto consiste no desenvolvimento e criação de um banco de funções plenópticas amostradas a partir de imagens computacionalmente renderizadas pela técnica de ray-tracing a fim de modelar e estudar imagens tridimensionais realistas.

Funções plenópticas são uma família de funções que armazenam toda a informaçao visual de um determinado ambiente. Com estas funções, é possível armazenar as características de espaços e objetos como posição, componentes cromáticas, assim como posicionamento e orientação da câmera que captura os raios de luz da cena. Para gerar esse tipo de função, é utilizada neste trabalho uma técnica de renderização chamada ray-tracing, cujo algorítmo baseia-se nas características físicas da luz, seu comportamento e interação com objetos.

Dentro desta temática, foi criado um banco de matrizes compostas de imagens a fim de viabilizar estudos na área de processamento de imagens, como a interação de seres humanos com modelos tridimensionais interativos com o objetivo de avaliar conforto e realismo de acordo com a variação de alguns parâmetros.

A síntese de vistas realistas neste projeto está vinculada à dificuldade de trabalhar-se com um número grande de câmaras visto que o processo de sincronização, ajuste de foco e tempo de exposição de cada câmara e suas respectivas posições tornariam o projeto muito dispendioso, demorado e impreciso. Neste quesito, a renderização de imagens, apesar de acarretar em um grande gasto computacional, torna-se uma alternativa viável para a aquisição deste tipo de imagem para fins de pesquisa.

Palavras-Chave: funções plenópticas, *ray-tracing*.

ABSTRACT

This project consists of developing and creating a database of sampled plenoptic functions from rendered computational models. These images are rendered using ray-tracing methods for the purpose of modeling and studying realistic three-dimensional images.

Plenoptic functions are classes of functions that store all visual information from a particular environment. Using these functions, one can store object features such as position, colours as well as camera features like orientation and its position.

In order to generate this kind of functions, a rendering technique called ray-tracing is used. It is based on physical light features, behaviour and interactions with objects.

Due to the difficulties when dealing with a large amount of cameras, light field acquisitions tends to be very costly and slow processes. This is mainly caused by the time spent on synchronization, focus adjustment and exposition of the cameras. However, even though image rendering is computationally complex, it appears to be a good option for light field acquisition.

Key-words: plenoptic functions, *ray-tracing*.

Sumário

Lista de Figuras

Capítulo 1

Introdução

A obtenção de múltiplas vistas como funções plenópticas de uma cena é atualmente uma área largamente estudada devido às mais diversas aplicações em diferentes áreas como entretenimento, na área médica e também na arte, como por exemplo, a fotografia. Este grupo de funções é responsável por armazenar imagens diversas de uma mesma cena, a partir de condições diferentes do observador, ou seja, a imagem a ser vista depende do estado da câmera, definido por seus parâmetros, incluindo nos mesmos, além dos parâmetros intrínsecos, como distância focal, sua posição no espaço, também chamados de parâmetros extrínsecos. Este trabalho explorará esse tema criando um conjunto destas funções e explorando algumas de suas carcterísticas.

1.1 Delimitação

O objeto de estudo é a geração imagens de uma função plenóptica renderizadas a partir da técnica de *ray-tracing* com o *software* PBRT (Physically Based Ray-Tracer) [1]. Estas imagens serão geradas para serem posteriormente estudadas no contexto da tecnologia 3D. Neste contexto, por exemplo, serão avaliadas as melhores disparidades, distâncias entre imagens e o número mínimo de imagens para que, em um contexto de visão estereoscópica com mudança de pontos de vista a partir de um rastreamento de olhos, haja uma boa fluidez durante a troca entre os pares de vistas.

1.2 Localização

Iniciado com os trabalhos de Turner Whitted em 1979 [2], o *ray-tracing* ou traçador de raios, é uma classe de algoritmos desenvolvidos para emular o percurso real de raios de luz em uma determinada cena seguindo as leis da física isto é, a luz é afetada por efeitos de difração, reflexão e dispersão. Desde então, com o aprimoramento da técnica, é possivel atualmente a representação de cenas renderizadas em alta fidelidade principalmente na área do entretenimento (cinema, jogos) e na modelagem para diversos ramos de pesquisa (medicina, engenharia). Esta técnica então será utilizada juntamente com o estudo de funções plenópticas para os fins desejados neste projeto.

Atualmente, o campo de pesquisa em funções plenópticas têm sido desenvolvido principalmente em ajuste e armazenamento de múltiplos planos focais em fotografia. Câmeras especiais foram desenvolvidas como a descrita em [3], utilizando-se de lentes especiais e princípios ópticos, para armazenar estes tipos de imagem e serem ao mesmo tempo utilizáveis pela população. Outra maneira utilizada de se obter funções plenópticas é criar um arranjo uni ou bidimensional de câmeras, como feito por [4]. Porém, estas câmeras possuem limitações de uso, como resolução e quantidade de vistas adquiridas, além de tornar o projeto muito caro. Estes fatores limitariam o campo de pesquisa e por isso faz-se necessário a geração deste tipo de banco de imagens, o qual este projeto se propõe a fazer.

1.3 Justificativa

As funções plenópticas, nome latino formado por *plenus* (completo, total) e *optic* (visão), também conhecidas por *light-fields* ou campos de luz, é uma família de funções que descrevem, em um determinado ponto do espaço, toda informação visual (como cor e iluminação) a partir de um conjunto de parâmetros inicialmente descritos por [5] que contém informações sobre o observador. Na função plenóptica, as coordenadas x e y (ou θ e ϕ) indicam a orientação da visão, as coordenadas V_x, V_y e V_z representam a posição do observador, o instante de tempo t da observação e a densidade espectral de potência $S(\lambda)$ da luz recebida nas células cromáticas de visão do observador (cones).

A síntese de vistas realistas neste projeto está vinculada à dificuldade de trabalhar-se com um número grande de câmeras fotográficas visto que o processo de sincronização, ajuste de foco e tempo de exposição de cada câmera e suas respectivas posições tornariam o projeto muito dispendioso, demorado e impreciso. Neste quesito, a renderização de imagens, apesar de acarretar em um grande gasto computacional, torna-se uma melhor alternativa de aquisição deste tipo de imagem para fins de pesquisa.

1.4 Objetivo

O objetivo geral é, partindo das justificativas mencionadas, criar um banco de funções plenópticas com o intuito de estudar alguns métodos para compressão e tratamento para estes tipos de imagens.

1.5 Metodologia

Os ambientes computacionalmente gerados neste trabalho serão renderizados a partir da técnica de *Ray-tracing*. Esta técnica consiste em sintetizar cenas através de objetos, texturas e modelos de iluminação e ambientes, todos descritos em uma linguagem computacional, projetando raios de visão a partir do ponto de localização do observador(câmera) em direção aos *pixels* da imagem. Ao encontrar o objeto ou plano de fundo nesta direção o traçador computa coeficientes de reflexão, refração, transmissão; incidência de luz ou projeção de sombra nesta região, propriedades de textura do material, etc. Após o cômputo final do *pixel* referido e dos restantes, o processo se repete até a qualidade da imagem alcançar o realismo desejado em número de amostras por *pixel*.

Para renderizar cada imagem do plano de visão é então utilizada uma transformação de translação nas coordenadas da câmera, neste caso particular mantendo a direção da observação de cada câmera sempre paralela às demais e também mantendo a cena sempre dentro dos limites da imagem para não haver perda na percepção tridimensional.

Obtidas as imagens teremos então a função plenóptica da cena. Considerando uma malha igualmente espaçada, a função contará no total com cinco dimensões:

duas espaciais dentro de uma imagem, uma de profundidade ao acrescentarmos a imagem da outra vista, e mais duas relativas à posição na malha de observação. A geração de um par de vistas 3D é realizada ao renderizar duas imagens desta cena localizadas na mesma linha horizontal da rede. Para uma pequena percepção de profundidade, imagens próximas serão escolhidas para compor às vistas esquerda e direita assim como imagens mais distantes proporcionarão uma sensação tridimensional maior. Assim, ao fixar uma disparidade (dada em número de imagens de distância), o modelo pode ser interativo com o observador se este ao se deslocar um pouco, os pares de imagens sejam trocados por imagens adjacentes porém mantendo a disparidade original.

1.6 Materiais

Neste trabalho será utilizado o *software* livre para ray-tracing PBRT (*Physically Based Ray-Tracing*) utilizando uma base de modelos computacionais disponibilizados gratuitamente em [6]. Este *software* descreve as cenas em arquivos que neste projeto são alterados. Cópias deste arquivo de configuração são criadas e cada uma destas cópias possui uma posição diferente de câmera. Cada modificação nestes arquivos é feita por *scripts* em *BASH* em ambiente Linux. Também utilizamos os *softwares* de licensas livres OpenEXR e OpenSceneGraph além do pacote OpenCV trabalhado na linguagem C++.

As imagens obtidas são testadas então em computadores do Laboratório de Processamento de Sinais cujos monitores permitem trabalhar com diferentes tipos de estereoscopia. Será utilizado também um monitor 3D com polarização circular do modelo um JVC 463D10U de 46 polegadas para visualização e teste dos modelos obtidos.

1.7 Organização

No capítulo 2 do presente documento, serão apresentados os fundamentos teóricos deste projeto, que consiste na definição de funções plenópticas, da técnica de *Ray-tracing* e alguns conceitos básicos sobre visão tridimensional.

O capítulo 3 descreve toda o procedimento de criação do banco de funções plenópticas e sua descrição matemática.

O capítulo 4 apresenta os resultados do presente trabalho e apresenta sugestões para os trabalhos vindouros.

Capítulo 2

Fundamentos Teóricos

Neste capítulo serão apresentado os conceitos-base e as ferramentas necessárias para a implementação do projeto. Na Seção 2.1, será discutido o algoritmo de *ray-tracing* seus conceitos e os parâmetros utilizados.

A Seção 2.2 descreverá o comportamento das funções plenópticas, sua definição e sua aplicação na área de processamento de imagens

A Seção 2.3 fará uma breve descrição do procedimento para a utilização da tecnologia 3D para a exibição das funções plenópticas.

2.1 *Ray-tracing*

2.1.1 Definição

No contexto tecnológio atual, a criação de cenários ou imagens virtuais podem ser encontradas em diferentes lugares e nas mais diversas aplicações. Entretanto, conforme a capacidade computacional disponível aumenta, cresce a demanda por imagens modeladas mais realistas e precisas, como a imagem na Figura 2.1. Um dos tipos de *software* capaz de cumprir estes requisitos é o *Ray-tracer* ou *Traçador de raios*. Essa classe de algoritmos é caracterizada por transformar um conjunto de parâmetros da cena em uma imagem utilizando-se de princípios da Física e da Óptica. Para isso é necessário de um grande conjunto de especificações como, posição da câmera, dos objetos e das fontes de luz; textura dos objetos e seus coeficientes de refração e difração; características das lentes da câmera e das fontes luminosas; formas de visualização (perspectiva, panorâmica, ...); resolução da imagem final;

número de amostragens por *pixel* assim como outros parâmetros mais específicos do renderizador, como por exemplo, o método de integração e de amostragem.

Figura 2.1: Exemplo de imagem realista renderizada pela técnica de *ray-tracing*.

Os procedimentos usados por essa classe de algoritmos são naturalmente re-

cursivos: considerando um raio de luz emitido do centro da câmera dado um objeto na cena, computa-se as interações do raio com os elementos presentes em cena como descrito na Seção 2.1.2.

Neste projeto, é utilizado um algoritmo de Traçador de raios chamado PBRT - *Phisically Based Ray Tracer* - um *software* de código aberto disponibilizado em [1], que permite o desenvolvimento necessário do banco de imagens. Também todas as cenas das quais foram desenvolvidas as funções plenópticas renderizadas podem ser obtidas na página deste programa (ver Apêndice A).

2.1.2 Algoritmo

A técnica de *ray-tracing* ao buscar um realismo possível na síntese de imagens tridimensionais, utiliza-se das propriedades da luz em seu trajeto pela cena para computar o valor de cada *pixel*. Todavia, uma modelagem exata do trajeto da luz, partindo de uma fonte luminosa e sendo refletida e absorvida por objetos até encontrar a câmera seria extremamente custosa computacionalmente, pois somente uma pequena minoria destes raios encontrariam efetivamente a lente da câmera, o que tornaria o *ray-tracing* um algoritmo inviável. Por isso, é utilizada uma lógica baseada na Teoria da Emissão, criada pelos gregos, que partia da suposição que raios de luz eram emitidos pelos olhos, interagiam com o ambiente e então eram refletidos de volta aos olhos. Uma análise um pouco mais detalhada desta lógica pode ser vista no algoritmo descrito na Figura 2.2.

2.1.3 Amostragem

Como visto anteriormente, o procedimento inicial do *ray-tracer* é computar o vetor que liga o centro de projeção ao *pixel* que será calculado. Porém, este *pixel* possui uma certa área que permite a criação de diversos vetores que ligam o ponto de observação a esta área. O conjunto destes vetores ligam a câmera à cena e pertencem no espaço à uma pirâmide, como visto na Figura 2.4 . Portanto,o número de vetores contidos nesta pirâmide é chamado de número de amostras por *pixel*. Cabe posteriormente ao algoritmo de integração encontrar o melhor vetor RGB que representa os resultados obtidos para cada raio de luz traçado.

Em termos computacionais, quanto maior a densidade da amostragem por

Objetivo

Dado uma cena composta por I objetos descrita por um *software* de modelagem, renderizar a cena em uma imagem $m \times n$ obtendo o valor correspondente de cada *pixel* dessa imagem que representa a cena de modo mais realista.

Algoritmo

1. Selecionar o centro de projeção, bem como o plano de projeção e suas dimensões (distância do centro de projeção, número de *pixels* verticais e horizontais).
2. Para cada *pixel* da imagem no plano de projeção fazer :
   (a) Determinar o raio r de luz que liga o centro de projeção ao *pixel* em questão.
   (b) Para cada objeto S_i em cena:
      i. Determinar se o raio de luz intercepta cada objeto.
         - Caso negativo, retornar ao *pixel* o valor do plano de fundo da cena.
         - Caso afirmativo, determinar se dos objetos interceptados, este é o mais próximo. Caso seja, armazenar i em i_{min} e a posição do ponto P de interseção.
   (c) Para o ponto P interceptado em $S_{i_{min}}$, calcular o vetor $\vec{n}$ normal à superfície deste.
   (d) Para cada fonte luminosa L_j
      - Lançar um raio de luz $\vec{l}$ do ponto P até L_j.
      - Determinar se neste percurso, há interceção com outros objetos S_k, $k \neq i$.
         – Caso afirmativo, aplicar sombreado ao ponto.
         – Caso negativo, aplicar iluminação ao ponto. Esta iluminação dependerá das propriedades do material e do fator $(\vec{n} \cdot \vec{l})$, que representa o produto interno entre o raio de luz $\vec{l}$ e o vetor $\vec{n}$ normal à superfície no ponto P.
            * Se o material de S_i é reflector, gerar um raio reflector e repetir o processo.
            * Se o material de S_i é transparente, gerar um raio refrator e repetir o processo.

Figura 2.2: Algoritmo traçador de raios. Uma ilustração deste algoritmo pode ser vista na Figura

pixel, mais lenta torna-se a renderização. Porém isto torna o modelo muito mais realista aos olhos. Deve-se então utilizar um número suficientemente alto de vetores

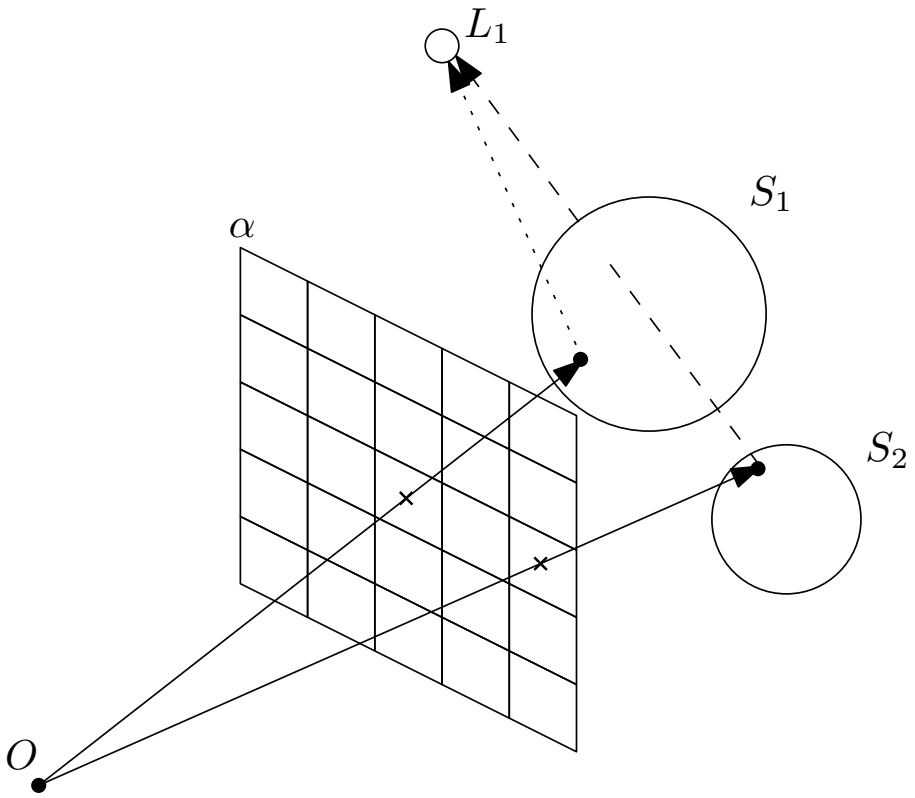


Figura 2.3: Representação do algoritmo de iluminação do *ray-tracing*. O raio pontilhado representa um raio lançado de um ponto iluminado e o raio tracejado de um ponto de sombra.

para tornar a imagem fiel, porém não acima da percepção humana o que só tornaria a renderização mais lenta.

Através da comparação da Figura 2.5 podemos notar que uma amostragem de baixa densidade como na Figura 2.5a, torna a cena muito granulosa e falsa aos olhos, enquanto uma amostragem de alta densidade por *pixel*, exemplificada na Figura 2.5b, resulta em uma imagem mais realista.

Este parâmetro está descrito no arquivo de configuração por:

```
Sampler "nomeAmostrador" "integer pixelsamples" [spp]
```

onde *nomeAmostrador* é o algoritmo de escolha dos melhores vetores para amostragem dentro da pirâmide e *spp* é a quantidade destes vetores.

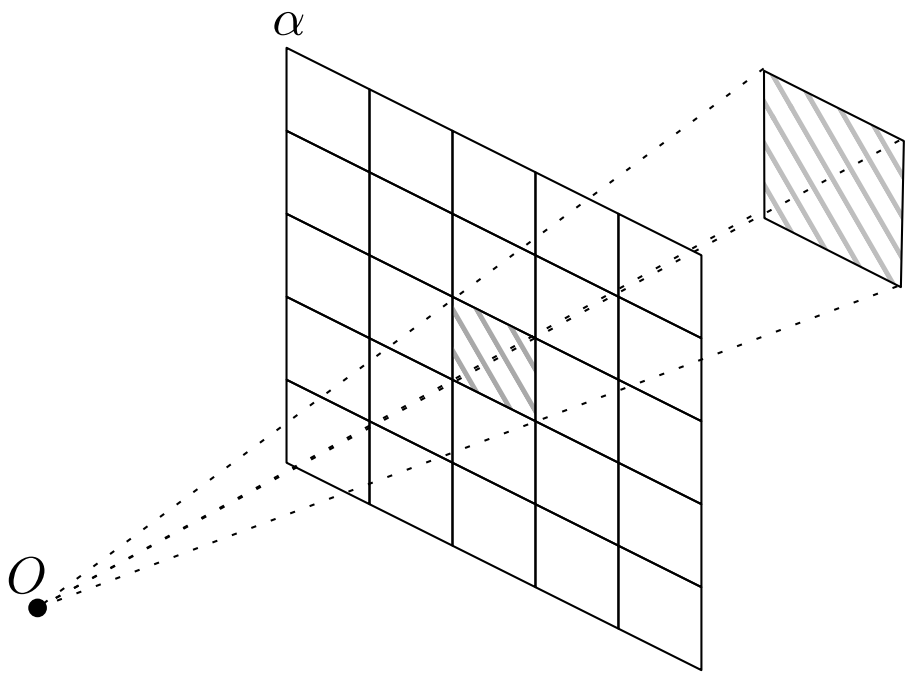
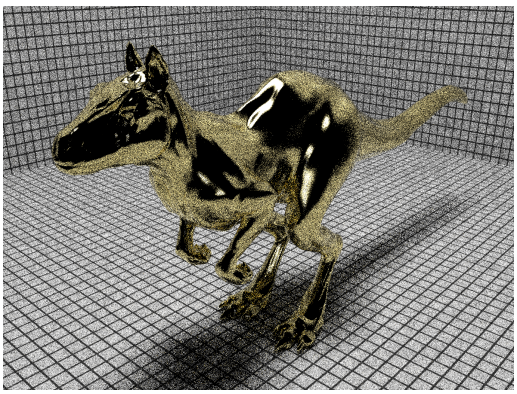
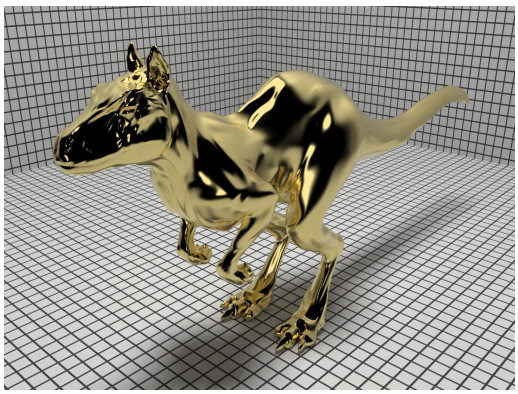


Figura 2.4: Área na cena a ser representada por vetores pertencentes a um *pixel*.



(a) killeroo-gold renderizada com 2 amostras/pixel com tempo de renderização de 25.1s



(b) killeroo-gold renderizada com 1024 amostras/pixel com tempo de renderização de 1663.3s

Figura 2.5: Efeito de diferentes níveis de amostragem e a diferença proporcinada no tempo de renderização.

2.1.4 Resolução

Resolução é a quantidade de *pixels* computados para cada eixo, ou seja, são as dimensões da matriz do plano de projeção do algoritmo traçador. A resolução pode ser representada em um arquivo *.pbrt* por:

```
"integer xresolution" [xres] "integer yresolution" [yres]
```

onde *xres* é o número de colunas e *yres* o número de linhas da matriz do plano de projeção.

2.1.5 Posição e orientação da câmera

A posição e a orientação da câmera são descritas no arquivo *pbrt* ..., pelo comando *LookAt*, como visto a seguir:

```
LookAt Ox Oy Oz Px Py Pz Ux Uy Uz
```

Este comando recebe nove parâmetros em ponto flutuante, ilustrados na Figura 2.6 que indicam:

- A posição tridimensional da câmera no espaço através de O_x,O_y e O_z;
- O ponto para o qual a câmera aponta, dado por P_x,P_y e P_z;
- O vetor de orientação vertical da câmera, que é o vetor perpendicular à borda superior da imagem dado por U_x, U_y e U_z (ver Figura 2.6).

2.2 Funções Plenópticas

2.2.1 Definição

A partir do século XIX, quando a luz passou a ser vista pelo meio científico como onda eletromagnética, mais especificamente depois da publicação *Thoughts on Ray Vibrations* de Michael Faraday em 1846, foi introduzido o conceito de campo de luz para representar o comportamento luminoso no ambiente e suas interações com objetos.

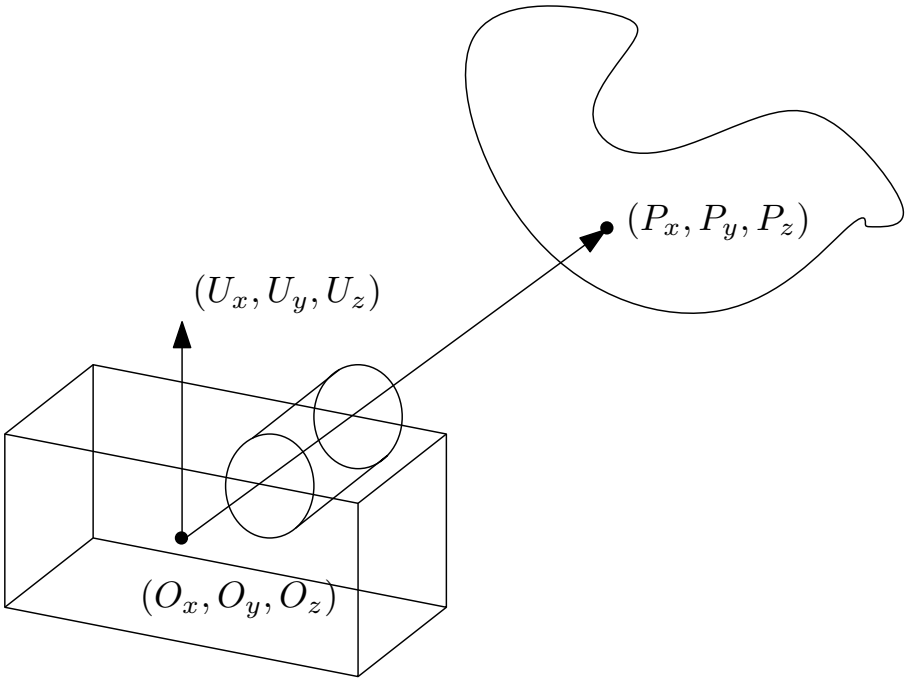


Figura 2.6: Representação dos parâmetros de uma câmera virtual .

Desde então, o modelo da luz e dos campos criados foram aprimorados até a representação descrita por [5] que representa-os pela intitulada **função plenóptica**. Também conhecida como *light field* é uma família de funções multidimensionais que descrevem, em um determinado ponto do espaço, toda informação visual (como cor e iluminação) que pode representar este ponto da cena e o seu observador.

Esta cena quando observada pode, de um modo geral, ser descrita completamente por pelo menos oito variáveis, como mostrado na Figura 2.7.

Estas variáveis são definidas por:

- Três coordenadas para descrever a posição do observador no espaço (O_x, O_y e O_z);
- Três coordenadas para descrever o *voxel (volumetric element)* do espaço observado (P_x, P_y e P_z);
- Uma coordenada que representa o eixo dos tempos na cena;
- Uma coordenada para descrever a densidade espectral de energia luminosa no voxel observado.

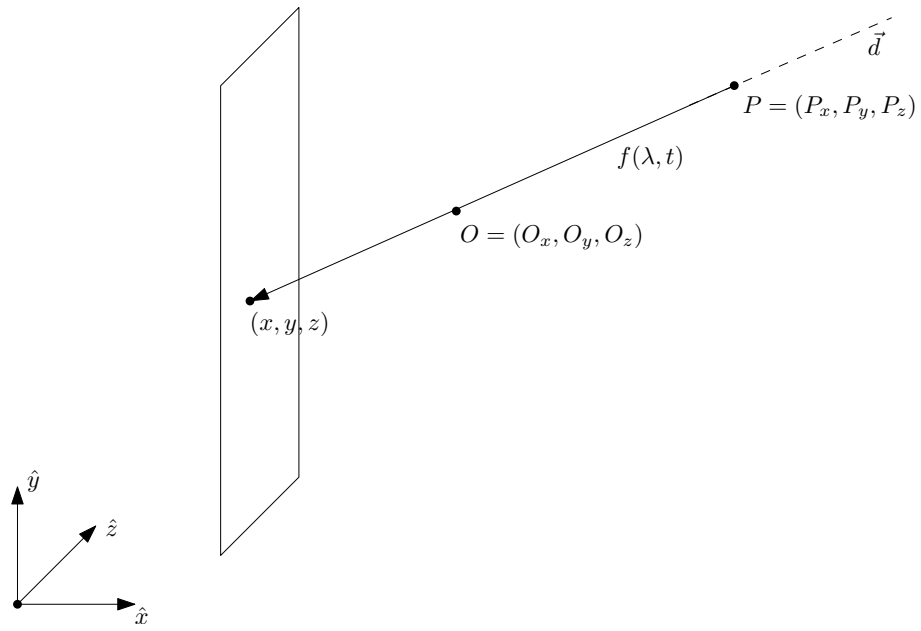


Figura 2.7: Representação de uma função plenóptica completa.

O valor do campo de luz F pode então ser explicitado na forma:

$$F = f(x, y, z, P_x, P_y, P_z, t, \lambda) \tag{2.1}$$

2.2.2 Campos de luz de menor dimensão

Normalmente, uma função plenóptica como a descrita na equação 2.1 implica em uma quantidade muito grande de informação a ser armazenada e é praticamente inviável dada a tecnologia atual. Por este motivo, faz-se necessária uma diminuição na ordem deste campo.

Será considerado um raio de luz emitido do ponto P através de uma abertura infinitesimalmente pequena localizada em O. Este raio de luz incidirá em algum ponto interior à câmera cujas coordenadas são (x, y, z). Dada uma reta r_d que interliga (x, y, z) ao ponto O descrita em sua forma paramétrica por um dado parâmetro γ, no formato

$$r_d = \begin{cases} P_x = x + \gamma(O_x - x) \\ P_y = y + \gamma(O_y - y) \\ P_z = z + \gamma(O_z - z) \end{cases}$$

e que a intensidade de luz I no ponto (x, y, z) depende do valor de F em todos os pontos ao longo de r_d exteriores à câmera, ou seja, com $1 \leq \gamma < \infty$ na forma

$$I = \int_1^\infty r_d(\gamma) d\gamma \tag{2.2}$$

pode ser definida uma função $g(x, y, z, O_x, O_y, O_z, \gamma, t, \lambda)$ que possui a forma

$$g(x, y, z, O_x, O_y, O_z, \gamma, t, \lambda) = f(x, y, z, x+\gamma(O_x-x), y+\gamma(O_y-y), z+\gamma(O_z-z), t, \lambda). \tag{2.3}$$

Como é desejado considerar que a intensidade na imagem multidimensional é uma contribuição da intensidade luminosa em cada *voxel*, considerando esta intensidade por apenas um instante de tempo t_0 e utilizando-nos da equação 2.2, chegamos à

$$I(x, y, z, O_x, O_y, O_z, \lambda, t_0) = \int_1^\infty g(x, y, z, O_x, O_y, O_z, \gamma, t, \lambda) d\gamma. \tag{2.4}$$

Segundo [7], a resposta $\alpha_i(C)$ de uma célula cromática i com resposta espectrais $S_i(\lambda)$ a um estímulo luminoso, entro dos limites de visão humana ($350nm \leq \lambda \leq 780nm$) é dada por

$$\alpha_i(C) = \int_{350nm}^{780nm} S_i(\lambda) C(\lambda)\, d\lambda. \tag{2.5}$$

Por essa razão, um canal de cor primária $\mathcal{P}_i(x, y, z, t)$ descrita na imagem multidimensional pode ser representada como

$$\mathcal{P}_i(x, y, z, t_0) = \int_{350nm}^{780nm} \int_1^\infty g(x, y, z, O_x, O_y, O_z, \gamma, t, \lambda) S_{\mathcal{P}_i}(\lambda)\, d\gamma d\lambda. \tag{2.6}$$

Se representarmos esta função para apenas uma profundidade z_0 e utilizarmos por exemplo, as cores primárias do sistema RGB, cada canal de cor da imagem pode ser descrito como nas equações

$$R(x, y, z_0, O_x, O_y, O_z, t_0) = \int_{350nm}^{780nm} \int_1^{\infty} g(x, y, z_0, O_x, O_y, O_z, \gamma, t, \lambda) S_R(\lambda)\, d\gamma d\lambda; \quad (2.7a)$$

$$G(x, y, z_0, O_x, O_y, O_z, t_0) = \int_{350nm}^{780nm} \int_1^{\infty} g(x, y, z_0, O_x, O_y, O_z, \gamma, t, \lambda) S_G(\lambda)\, d\gamma d\lambda; \quad (2.7b)$$

$$B(x, y, z_0, O_x, O_y, O_z, t_0) = \int_{350nm}^{780nm} \int_1^{\infty} g(x, y, z_0, O_x, O_y, O_z, \gamma, t, \lambda) S_B(\lambda)\, d\gamma d\lambda \quad (2.7c)$$

caso deseje-se modelar estas funções para câmeras convencionais.

2.2.3 Funções plenópticas amostradas

Até o momento, os campos de luz foram tratados como funções contínuas no espaço. Entretanto, para processamento digital é necessário amostrar tal campo suficientemente para bem representá-lo. Por isso, normalmente utilizam-se grades de câmeras para representar as posições mais interessantes de observadores, de modo que se possa utilizar as imagens obtidas para atender os propósitos determinados. Alguns exemplos de funções plenópticas amostradas podem ser encontradas em [4].

2.3 Visão Tridimensonal

2.3.1 Conceito

Visão tridimensional ou estereoscopia é a capacidadede gerar a percepção de profundidade. Isto é possivel graças à visão binocular, a qual utiliza duas imagens simultaneamente. Tendo informação obtidas a partir de duas imagens e utilizando-se de dicas visuais como a paralaxe, a acomodação dos olhos, dentre outras, é possível obter de imagens bidimensionais uma noção de tridimensionalidade.

2.3.2 Dicas Visuais

Existem diversas técnicas que permitem o observador obter informações de distância e profundidade sobre o ambiente no qual ele se encontra mesmo que este use apenas um olho. Abaixo serão descritas as pistas visuais mais relevantes para este trabalho.

- **Paralaxe de movimento**

O movimento relativo do observador ao objeto implica em uma mudança na posição aparente deste. A informação de profundidade é adquirida então comparando conjuntamente a variação da posição do objeto com a mudança de posição e velocidade do observador. Um objeto muito distante não mudará sua posição aparente quando a paralaxe é utilizada, enquanto um objeto muito próximo será visto deslocando-se proporcionalmente ao inverso da sua distância para o observador nas mesmas situações.

- **Disparidade**

  Percepção de distância adquirida através do uso de duas imagens, a do olho direito e a do olho esquerdo, devidamente comparadas pelo cérebro, utilizando-se das diferenças de paralaxe entre elas para estimar a profundidade do ambiente. Esta dica visual é dependente de um parâmetro chamado *distância interocular*. Esta distância é definida como a separação entre o centro de dois globos oculares ou entre as oculares de instrumentos ópticos [8]. Esta técnica é então aproveitada no presente trabalho para criar as câmeras virtuais dentro do renderizador. Estas câmeras estarão dispostas paralelamente e separadas por uma distância fixa. A uma dada separação, múltipla da distância entre duas vistas, é dado o nome de **distância interocular**. Esta distância representa a distância média padrão entre as pupilas humanas e possui o valor de $6.5cm$. Uma distância interocular muito acima da padrão gera uma sensação de desconforto para o espectador quando estas são processadas pelo cérebro, podendo ele perder inclusive a sensação de espacialidade tridimensional. Uma distância muito baixa por sua vez, diminui a sensação 3D, pois equivale a diminuir a amplitude da paralaxe de movimento.

- **Acomodação dos olhos**

  Para pequenas distâncias, o olho humano contrai ou expande seu cristalino com o auxílio de músculos ciliares. O objetivo deste tipo de esforço ocular é colocar o objeto a ser olhado em foco. Desta maneira, uma estimação da distância do objeto dá-se pela medida de esforço dos olhos comparativamente às distâncias anteriormente visualizadas pelo observador. Este método de medição só é válido para distâncias da ordem de grandeza do globo ocular, caso contrário

as distâncias são suficientemente grandes para considerarmos o sistema de lentes focado no infinito.

- **Efeito *zoom***

  Efeito percebido ao realizar-se um movimento na direção do objeto observado. Dependendo do sentido, este pode aumentar seu tamanho aparente ou diminuí-lo. Casa esta variação seja muito lenta para um grande deslocamento do observador, o objeto encontra-se muito distante. Caso contrário, ele encontra-se próximo ao sistema visual.

2.3.3 Exibição

Os modelos tridimensionais utilizados neste projeto são exibidos por meio de um monitor estereoscópico com tecnologia polarizada (Figura 2.8).

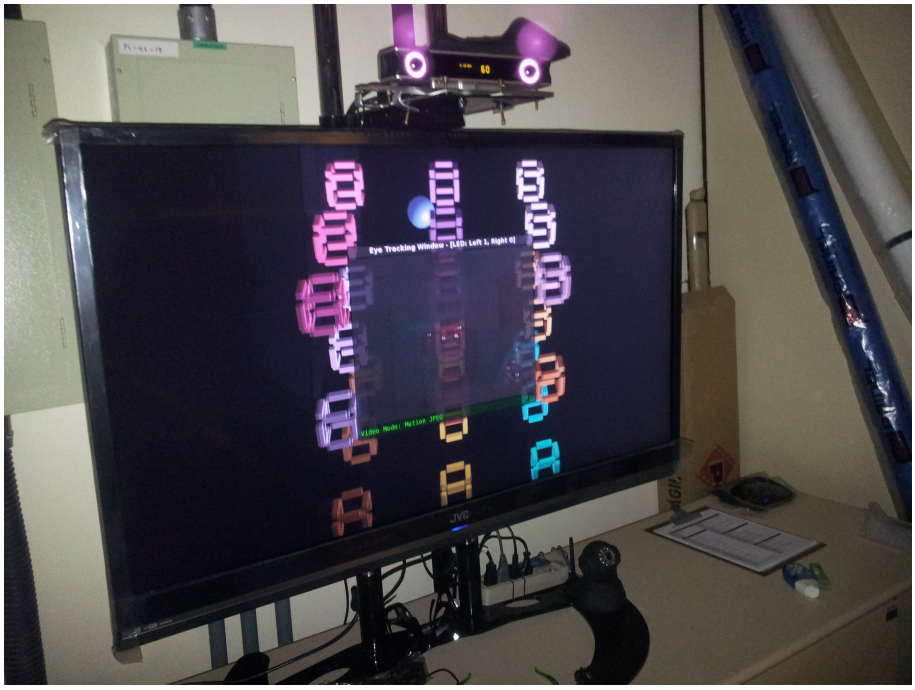


Figura 2.8: Monitor estereoscópico com tecnologia polarizada.

Este tipo de monitor exibe as vistas esquerda e direita no mesmo quadro, porém atribuindo uma polarização diferente a cada uma delas. Uma das imagens é atribuída às linhas pares da tela com polarização circular (dextrógira ou levógira) e a outra às linhas ímpares com polarização contrária, como mostrado na Figura 2.10. Este procedimento diminui à metade a resolução da imagem exibida. A atribuição

das imagens a cada olho é feita com filtros de polarização luminosa nas lentes dos óculos: cada lente possui um determinado filtro que permite passar apenas uma das polarizações (ver Figura 2.9). Desta maneira, um olho só consegue enxergar a vista correspondente e então proporcionar a percepção tridimensional ao espectador.

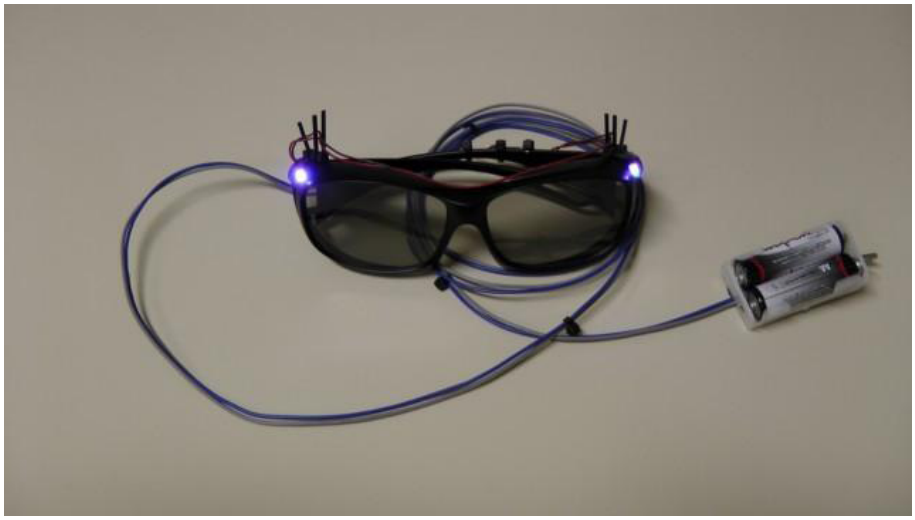


Figura 2.9: Óculos para visão 3D polarizado.

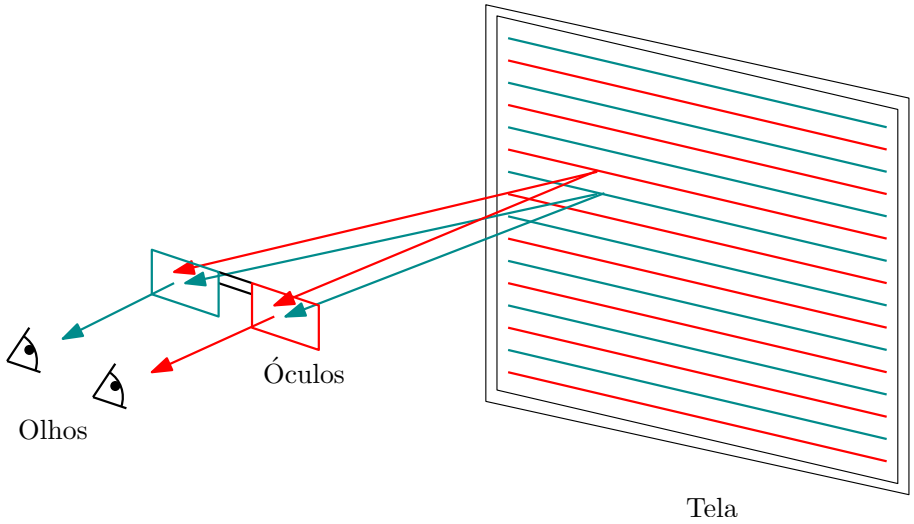


Figura 2.10: Esquema de exibição de um monitor polarizado. Neste imagem, as linhas ímpares (em azul) com uma certa polarização circular, atravessam somente uma lente do óculos que possui um filtro que permite passar somente este tipo de polarização. As linhas pares (em vermelho), de polarização contrária, atravessam somente a outra lente.

Capítulo 3

Metodologia

Este capítulo discursará a respeito da metodologia utilizada para a geração do banco de dados. Cada Seção descreverá uma etapa do processo.

A Seção 3.1, apresenta os procedimentos iniciais a serem tomados para a correta realização da amostragem dos *light-fields*.

A Seção 3.2 mostrará a maneira na qual as funções plenópticas serão obtidas e os algoritmos que permitem o posicionamento da câmera virtual no ambiente modelado.

Por fim a Seção 3.3 apresentará o procedimento aplicado nas imagens renderizadas para a posterior aplicação em 3D.

3.1 Procedimentos iniciais

3.1.1 Descrição de pacotes utilizados

Para este trabalho, o programa renderizador por *ray-tracing* escolhido foi o PBRT [1] por ser código livre e por possuir um banco de cenas pré-modeladas disponíveis.

Em ambiente Linux, é necessária a instalação de alguns pacotes adicionais para o pleno funcionamento do PBRT. Estes devem ser instalados na seguinte ordem:

- Pacotes necessários para a compilação dos programas, disponíveis nos repositórios de distribuições Linux:
  - autoconf;

  - automake;
  - libconf;
  - scons;
  - zlib.

- Programas necessários à compilação do *ray-tracer* PBRT:
  - **Imlbase** Bibliotecas matemáticas otimizadas em C++ para resolução de operações matriciais em ponto flutuante e equações polinomiais de até terceira ordem;
  - **OpenEXR** Formato de imagens com grande faixa dinâmica compatível com a linguagem C++;
  - **OpenEXR Viewers** Visualizador e manipulador de parâmetros de imagens em formato *.exr*.

Todos estes pacotes podem ser obtidos em repositórios Linux e em [9]. Este programa interpreta um arquivo descritivo de extensão *.pbrt*, que contém todas as posições, descrições e algoritmos utilizados para a renderização de uma cena.

Um exemplo de um arquivo de configuração *.pbrt* pode ser visto na Figura 3.1

Nesta cena, por exemplo, a sequência de comandos

```
Film "image"
    "string filename" "bunny.exr"
```

indica que o arquivo de saída será uma imagem de nome "bunny.exr".

Na terceira linha vemos a escolha do algoritmo de amostragem(*low discrepancy*) e o número de amostras por pixel (neste caso 32).

```
Sampler "lowdiscrepancy" "integer pixelsamples" [32]
```

Este número de amostras por pixel também nos modelos utilizados é o suficiente para garantir um bom compromisso entre realismo e tempo de execução.

A linha seguinte apresenta o comando de posicionamento e orientação da câmera já descrito na Subseção 2.1.5. Porém, linha seguinte oferece mais informação

bunny.pbrt

```
Film "image"
    "string filename" "bunny.exr"
Sampler "lowdiscrepancy" "integer pixelsamples" [32]

LookAt 0 .2 .2   -.02 .1 0  0 1 0
Camera "perspective" "float fov" [60]

WorldBegin

AttributeBegin
Rotate 90 1 0 0
LightSource "infinite" "string mapname"
                       ["textures/glacier_latlong.exr"]
        "integer nsamples" [32]
AttributeEnd

AttributeBegin

Translate 0 2 0
Rotate 90 1 0 0

AttributeEnd

Material "matte" "color Kd" [.4 .42 .4]
Shape "trianglemesh" "point P" [ -1 0 -1 1 0 -1 1 0 1 -1 0 1 ]
        "integer indices" [ 0 1 2 2 3 0]
Material "measured" "string filename" "brdfs/mystique.brdf"
Include "geometry/bunny.pbrt"
WorldEnd
```

Figura 3.1: Descrição da cena de exemplo *bunny* em seu arquivo de configuração.

a respeito de características da câmera, como o modo projeção da luz no plano de imagens (*perspective*) e uma variável relativa a este modo, chamada "campo de visão" (fov).

Após a palavra

```
WorldBegin
```

o arquivo passa a descrever os objetos em cena e as suas características geométricas, assunto que está fora do escopo deste trabalho.

3.1.2 Utilização do *ray-tracer*

Para iniciar o traçador de raios, é necessário informar ao renderizador alguns parâmetros importantes: como o nome do arquivo de saída (*outfile*) em formato *.exr*; o número de núcleos de processamento a serem utilizados(*ncores*); e se será feita uma renderização rápida ao invés do algoritmo convencional (*quick*).

Por exemplo, a sequência

```
pbrt  --ncores 4 --outfile saida.exr bunny.pbrt
```

renderiza a cena *bunny.pbrt* utilizando quatro núcleos do processador e escreve a imagem resultado no arquivo *saida.exr*. Porém, caso conste no arquivo de configuração a sequência

```
"string filename" "bunny.exr"
```

o programa sempre escreverá o arquivo de saída como explicitado na linha *"string filename"*, não atendendo a qualquer novo nome recebido através da opção *outfile*.

Caso seja executada, por exemplo, a sequência

```
pbrt  --quick  bunny.pbrt,
```

o modelo será renderizado de forma rápida (poucas amostras por *pixel* e baixa resolução) e escrito no arquivo de saída "bunny.exr".

3.2 Amostragem dos campos de luz

Uma vez definidos os campos de luz, faz-se necessário decidir qual amostragem os representa da melhor forma. Há diversas opções possíveis como, por exemplo, dispor as câmeras ao redor do objeto ou apontadas para cada direção. Neste trabalho a disposição utilizada foi um reticulado uniformemente espaçado com os eixos alinhados à posição inicial da câmera. Um exemplo de amostragem pode ser vista na Figura 3.2.

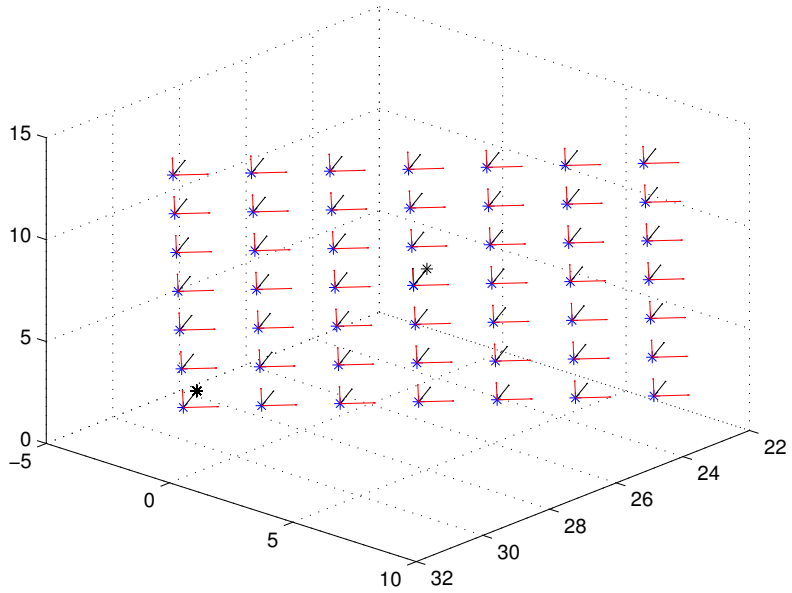


Figura 3.2: Amostragem bidimensional retangular de um campo de luz. Os vetores representados na imagem são os utilizados para posicionamento e orientação da câmera. A posição central marcada com asterisco preto corresponde à posição original da câmera e o outro asterisco preto marca a amostra de índice $(i, j) = (0, 0)$

Outro parâmetro ainda relacionado à amostragem é a densidade de tais reticulados. Primeiramente são definidos os limites da rede de câmeras. Por isso, para cada modelo testado, foi avaliado o deslocamento horizontal máximo para o qual o motivo da cena não fosse descaracterizado. Isto significa que a região onde a câmera pode ser posicionada foi limitada a fim de evitar eventuais oclusões de objetos importantes da cena, o que descaracterizaria o modelo e seu efeito 3D. Por exemplo, para o modelo na Figura A.1, a região escolhida é aquela em que ainda podemos ver

o carro completamente.

Definidos os limites, o número de amostras neste intervalo deve ser suficiente para que a transição de uma vista para outra não seja percebida como brusca. Para isto, a solução é renderizar um grande número de vistas, inicialmente 640, que é o número máximo de vistas utilizado em [10].

3.2.1 Número de dimensões

Como descrito na equação 2.6, as dimensões referentes ao observador de uma função plenóptica para a geração de uma imagem bidimensional são suas três coordenadas de posicionamento espacial do centro da câmera, três de posicionamento espacial da imagem na câmera (que podem ser reduzidas a duas no seu plano da imagem), e uma de posicionamento temporal.

Neste trabalho, foi desenvolvido um *script* em *BASH* capaz de posicionar uma câmera no campo de luz no espaço do modelo, ou seja, utilizando somente as coordenadas espaciais e mantendo a cena estática no tempo. A variação temporal pode ser simulada de acordo com a movimentação do espectador no programa de realidade aumentada.

Contudo, devido ao grande tempo de renderização de cada imagem, para este projeto apenas a variação da posição da câmera em uma direção será simulada. A direção escolhida é a horizontal, paralela ao plano da imagem inicial do modelo. Esta orientação da rede de câmeras foi escolhida por ser a direção do movimento mais comum ao movimento humano ao avaliar uma cena.

3.2.2 Alteração da Resolução

Para adequar as imagens geradas ao monitor de alta-definição utilizado, é necessário alterar a resolução de saída de cada modelo. O monitor utilizado nas simulações foi um JVC $463D10U$ de $46''$, que possui a resolução de 1920×1080. Por isso, o script usado então altera a linha do arquivo descrito na Seção 2.1.4 de

```
“integer xresolution” [xres] “integer yresolution” [yres]
```

para

`"integer xresolution" [1920] "integer yresolution" [1080].`

Desta maneira, o tamanho das imagens geradas enquadrar-se-á perfeitamente no monitor.

3.2.3 Translação de câmeras virtuais

Seja uma câmera virtual descrita por um modelo de computação gráfica em um ambiente do qual deseja-se amostrar uma função plenóptica.

Podemos então definir as bases ortogonais do espaço no qual a câmera está imersa a partir dos parêmetros desta já definidos através do comando "LookAt". Estas bases são:

- Vetor $\hat{d}$, obtido a partir da normalização de $\vec{d} = P - O$, ou seja é o vetor que aponta da posição da câmera até algum ponto da cena desejado;
- Vetor $\hat{u}$, que indica a direção perpendicular à borda superior da câmera;
- Vetor $\hat{v}$, estipulado como terceiro vetor da base deste espaço, é ortogonal a $\hat{u}$ e $\hat{d}$, sendo obtido a partir de $\hat{v} = \hat{d} \times \hat{u}$, como mostrado na Figura 3.3.

Esta câmera deve estar posicionada dentro de um volume limitante em forma de paralelepípedo, alinhado às bases ortogonais, necessário para não ocorrem oclusões indesejadas nos modelos.

Define-se então as dimensões deste paralelepípedo limitante como Δ_d, Δ_v e Δ_u para as direções $\hat{d}$, $\hat{v}$ e $\hat{u}$ respectivamente.

As amostras estão posicionadas neste paralelepípedo uniformemente e de maneira separável ao longo das direções $\hat{d}$, $\hat{u}$ e $\hat{v}$. Cada amostra deste espaço representa uma posição específica deste campo de luz onde a câmera virtual será posicionada.

O número de amostras será então $n_d.n_v.n_u$, onde n_a é o número de amostras na direção a.

Uma vez que cada direção é devidamente amostrada, a distância entre duas amostras, em cada direção, definidas como δ_d, δ_v e δ_u podem ser encontradas por

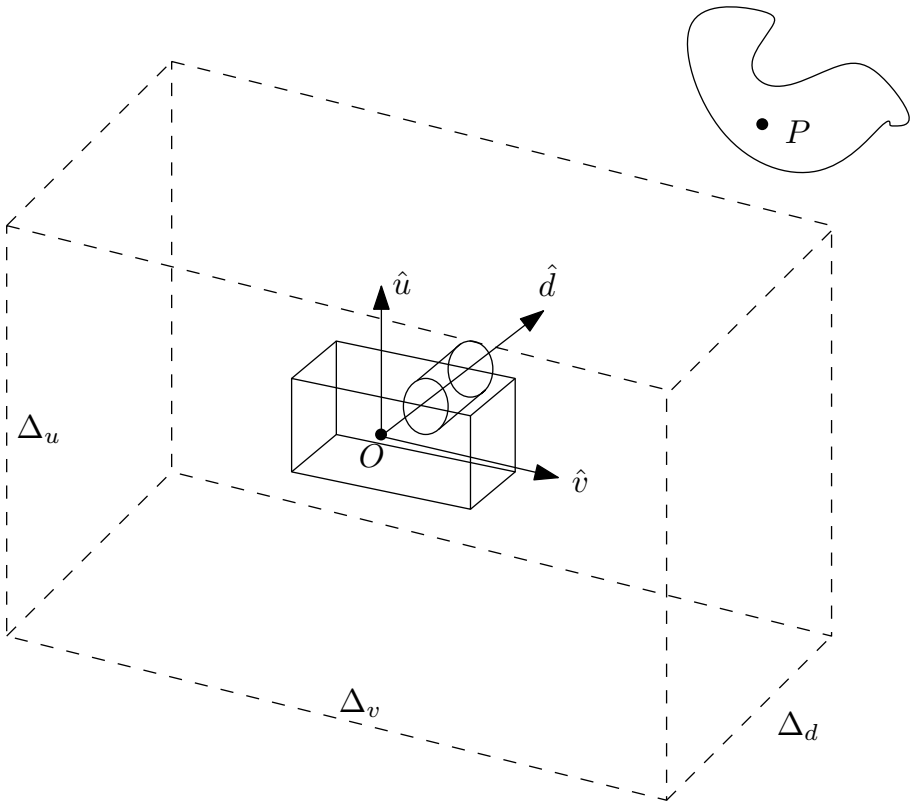


Figura 3.3: Representação da base ortogonal que descreve as posições nas quais a câmera virtual pode assumir.

$$\delta_d = \frac{\Delta_d}{n_d - 1}; \tag{3.1a}$$

$$\delta_v = \frac{\Delta_v}{n_v - 1}; \tag{3.1b}$$

$$\delta_u = \frac{\Delta_u}{n_u - 1}. \tag{3.1c}$$

Estas posições estão ilustradas na Figura 3.4.

Qualquer posição de amostragem no espaço é quantizada com base nos parâmetros definidos acima, podendo ser definida em função dos índices de quantização i, j e k das direções $\vec{v}$, $\vec{u}$ e $\vec{d}$ respectivamente, como

$$s(i,j,k) = i.\delta_v\hat{v} + j.\delta_u\hat{u} + k.\delta_d\hat{d}, \quad (i,j,k) \in \mathbb{N} \quad e \quad 0 \leq (i,j,k) < (n_v, n_u, n_d). \tag{3.2}$$

Assim cada ponto de origem O e cada ponto de orientação P desta câmera, quando expressos em termos de sua posição, serão escritos como $O(i,j,k)$ e $P(i,j,k)$ quando localizados no ponto de amostragem de índice (i,j,k).

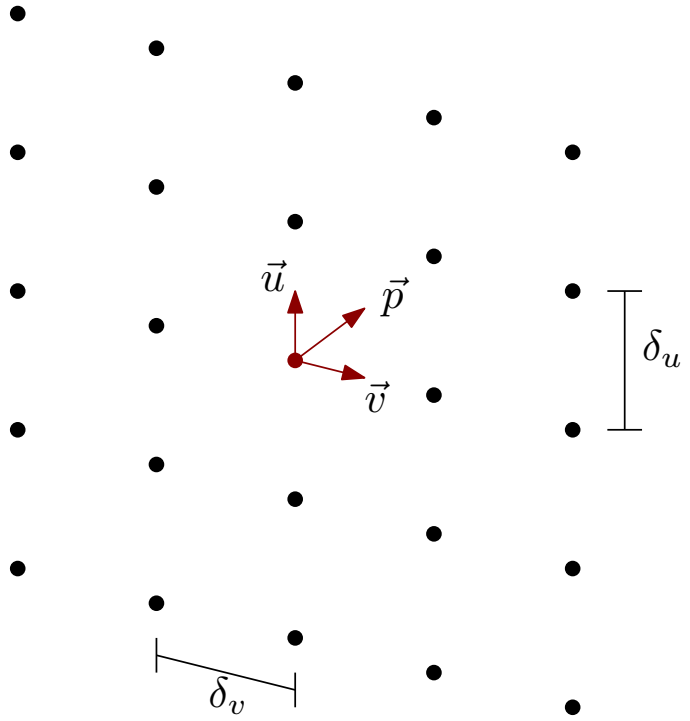


Figura 3.4: Representação da base ortogonal e de suas bases no espaço a ser trabalhado.

Definida a ordenação das amostras no espaço, é conveniente realizar um deslocamento inicial da câmera para que as coordenadas da câmera que está no centro do arranjo sejam $(i, j, k) = (0, 0, 0)$.

Estipulando que, a posição inicial da câmera, obtida através do comando "LookAt", está localizada precisamente no centro do paralelepípedo, então o deslocamento necessário para levar a câmera deste ponto inicial até a amostra de índice $(i, j, k) = (0, 0, 0)$ será como o exemplo da Figura 3.5.

Para então deslocar esta câmera de sua posição original O, apontada para o ponto da cena P, até o ponto $O(i, j, k) = O(0, 0, 0) = O'$ apontado para $P(i, j, k) = P(0, 0, 0) = P'$, deve-se aplicar a transformação descrita na Equação 3.3.

$$(O'_v, O'_u, O'_d) = (O_v, O_u, O_d) - (\frac{\Delta_v}{2}, \frac{\Delta_u}{2}, \frac{\Delta_d}{2}); \tag{3.3a}$$

$$(P'_v, P'_u, P'_d) = (P_v, P_u, P_d) - (\frac{\Delta_v}{2}, \frac{\Delta_u}{2}, \frac{\Delta_d}{2}), \tag{3.3b}$$

ou em termos do número de amostras por direção, utilizando a Equação 3.1:

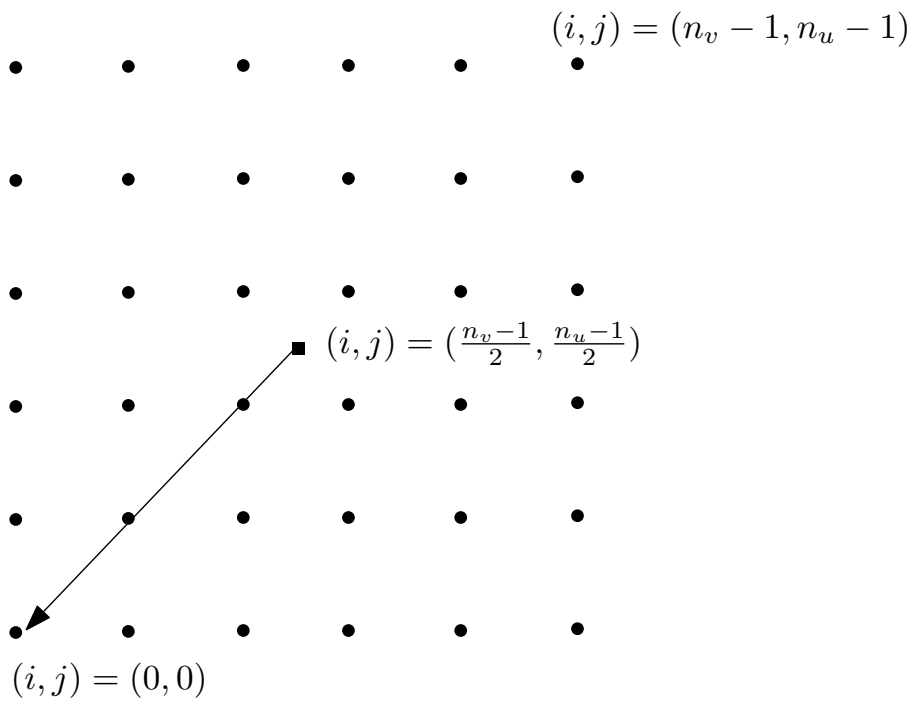


Figura 3.5: Translação do ponto inicial do modelo, considerando apenas as dimensões compostas por $\hat{v}$ e $\hat{u}$.

$$(O'_v, O'_u, O'_d) = (O_v, O_u, O_d) - \left(\frac{(n_v-1)\delta_v}{2}, \frac{(n_u-1)\delta_u}{2}, \frac{(n_d-1)\delta_d}{2}\right); \tag{3.4a}$$

$$(P'_v, P'_u, P'_d) = (P_v, P_u, P_d) - \left(\frac{(n_v-1)\delta_v}{2}, \frac{(n_u-1)\delta_u}{2}, \frac{(n_d-1)\delta_d}{2}\right). \tag{3.4b}$$

Uma vez deslocado ao ponto inicial O' e P', faz-se necessário produzir cada translação que levará a câmera destes pontos até a amostra desejada (i, j, k). Como os deslocamentos δ_v, δ_u e δ_d produzem o deslocamento de uma amostra cada, nas direções correspondentes, o valor final de $O(i, j, k)$ e de $P(i, j, k)$ podem ser expressos por:

$$O(i,j,k) = O' + i(\delta_v.\hat{v}) + j(\delta_u.\hat{u}) + k(\delta_d.\hat{d}); \tag{3.5a}$$

$$P(i,j,k) = P' + i(\delta_v.\hat{v}) + j(\delta_u.\hat{u}) + k(\delta_d.\hat{d}). \tag{3.5b}$$

Uma outra forma de exprimir o deslocamento em função da sua posição inicial é:

$$O(i,j,k) = O - \left(\frac{\Delta_v}{2}, \frac{\Delta_u}{2}, \frac{\Delta_d}{2}\right) + i\,(\delta_v.\hat{v}) + j\,(\delta_u.\hat{u}) + k(\delta_d.\hat{d}); \tag{3.6a}$$

$$P(i,j,k) = P - \left(\frac{\Delta_v}{2}, \frac{\Delta_u}{2}, \frac{\Delta_d}{2}\right) + i\,(\delta_v.\hat{v}) + j\,(\delta_u.\hat{u}) + k(\delta_d.\hat{d}). \tag{3.6b}$$

3.2.4 Algoritmo utilizado

Utilizando o conceito de translação descrito na Seção 3.2.3, o script "renderViews.sh" (ver Figura 3.6) foi desenvolvido em BASH para executar esta operação e renderizar por fim o modelo com a câmera posicionada nos locais devidos.

3.3 Pós-processamento das imagens

Após gerado o banco de vistas, alguns procedimentos devem ser feitos para prepará-lo para a visualização tridimensional. Isto inclui conversão de formato e determinação da diferença de índices que equivale a uma disparidade unitária.

3.3.1 Conversão de formato

O programa traçador de raios, produz todas as imagens de saída em formato ".exr" suportado por [9]. Esta extensão armazena informações da imagem em canais diferentes e por isso permite operações como por exemplo, ajuste de luminosidade.

A biblioteca *OpenEXR* permite uma operação, utilizada neste projeto, chamada *exrtopng*. Esta operação faz a conversão de todos os arquivos obtidos para imagem no formato ".png".

Para este processo também foi desenvolvido um arquivo em BASH, com a finalidade de converter todos os arquivos do banco chamado "conversion.sh".

3.3.2 Testes Preliminares de Disparidade

Após a geração das vistas, um teste para análise da percepção 3D é feito. Para isso foi desenvolvido um programa em linguagem C++, com o auxílio da biblioteca OPENCV [11]. Este concatena horizontalmente duas imagens (Figuras 3.8a e 3.8b) em posições diferentes no campo de luz (separados por um determinado número

Utilização `./renderViews` n_d n_u n_v `modelo` step **Parâmetros**

- n_d - Número de vistas na direção de $\hat{d}$;
- n_v - Número de vistas na direção de $\hat{v}$;
- n_u - Número de vistas na direção de $\hat{u}$;
- **modelo** - Nome do modelo em formato ".pbrt" a ser renderizado;
- step - Distância entre duas amostras.

Algoritmo Para um arquivo de configuração escolhido, o *script* "renderViews.sh" executa a seguinte sequência de passos:

1. **Pré-processamento do arquivo de modelo**
   - Remove linha que atribui nome ao arquivo de imagem de saída;
   - altera a resolução da imagem de saída para o perfeito enquadramento com o monitor;
2. **Translação das câmeras**
   - Obtém os atributos O, P e $\vec{u}$ de posição e orientação da câmera localizados no comando LookAt;
   - Calcula-se o vetor $\vec{d} = P - O$;
   - Normaliza-se $\vec{d}$ e $\vec{u}$;
   - Computa-se o vetor $\vec{v}$ como o produto vetorial $\vec{v} = \vec{d} \times \vec{u}$ em coordenadas cartesianas (não-homogêneas);
   - Deslocam-se os pontos O e P da câmera para seu ponto de início através de $O_s = O - \frac{(N_d-1).\text{step}}{2} - \frac{(N_v-1).\text{step}}{2}$;
   - Para i de 0 a $n_v - 1$
     - Para j de 0 a $n_u - 1$
       - Para k de 0 a $n_d - 1$
         - $O_{ijk} = O_s + i.\text{step} + \text{j.step} + \text{k.step}$;
         - $P_{ijk} = O_s + \vec{d}$;
         - Substituir os valores de P e O no comando LookAt ;
         - Renderizar o arquivo nomeando-o "modelo_i_j_k.exr'.

Figura 3.6: Algoritmo para geração de vistas de uma função plenóptica

de vistas, como descrito na Figura 3.7) subamostrando suas colunas e gera uma imagem resultante semelhante à da Figura 3.8c. A imagem resultante possui as mesmas dimensões de uma das vistas originais. Perde-se assim metade da resolução horizontal inicial.

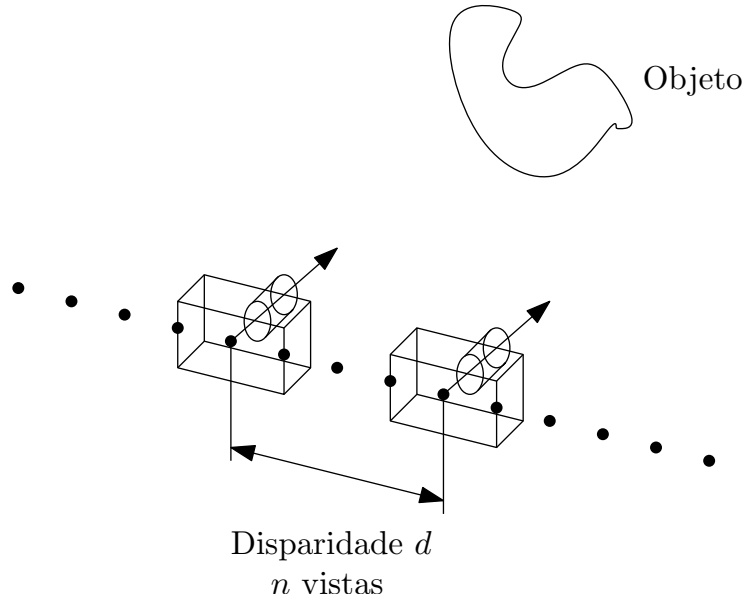


Figura 3.7: Obtenção das vistas esquerda e direita em funções plenópticas amostradas .

Este procedimento é necessário pois um dos modos de exibição do monitor utilizado transforma esse formato de representação de uma vista tridimensional em vistas entrelaçadas (ver Seção 2.3.3).

Para realizar a junção das imagens, foi desenvolvido um *script* em linguagem BASH, juntamente com a biblioteca OPENCV [11] para C++. Este script, nomeado como "mergeViews.sh" , possui a seguinte forma de utilização:

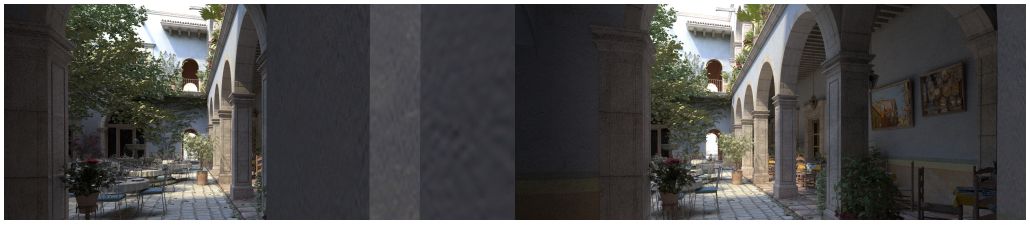
```
./mergeViews.sh   num_v     num_h     modelo   dist
```

sendo "num_v" e "num_h" o numero total de amostras na direção vertical e horizontal respectivamente. A variável "modelo" informa o nome do modelo de imagem a ser trabalhado e "dist" a distância (em número de imagens horizontalmente separadas) da vista esquerda e direita.

Ao combinar uma vista esquerda à outra direita, separadas por "dist", há uma perda intrínseca do processo de "dist" posições horizontais possíveis para o observador que assiste à cena tridimensional.

(a) Vista esquerda do modelo *sanmiguel*

(b) Vista esquerda do modelo *sanmiguel*



(c) Vistas concatenadas (disparidade de 1.4)

Figura 3.8: Concatenação de duas vistas obtidas no campo de luz de *sanmiguel*

3.4 Formato de armazenamento do banco de imagens

Após pós-processadas, o banco de imagens está pronto para ser utilizado. Todas as imagens estão no formato ".png", armazenadas em um diretório nomeado como "modelo_png/ ", onde "modelo" é o nome que representa o modelo utilizado.

Neste diretório, os arquivos estão dispostos no formato

`modelo_i_j_k.png`

onde i, j e k indicam o número da amostra referente a cada dimensão a partir de uma amostra inicial com índices $(i, j, k) = (0, 0, 0)$.

Como os modelos gerados neste trabalho exploram apenas uma dimensão, o nome dos arquivos estão na forma

`modelo_i.png`.

Capítulo 4

Conclusão

4.1 Resultados

Utilizando as técnicas descritas no Capítulo 3, alguns testes foram feitos com as imagens geradas até o momento.

O banco relativo ao modelo TT mostrado em A.1, foi gerado com as dimensões de 1×640, com a separação de 242 imagens para a disparidade unitária. Os limites destas foram atribuídos a imagens cujos deslocamentos são quase grandes suficientemente para causar o oclusão de alguma parte do carro. A partir destes limites, o espaço no qual a câmera virtual estava restrita, foi dividido em 640 posições. A partir deste banco de imagens gerado, foram exibidas no monitor algumas imagens em formato tridimensional com disparidades variadas e foi possível notar que obteve-se o efeito 3D.

Atualmente, estão sendo gerados novos bancos derivados dos modelos A.2 e A.3. Para o modelo *sanmiguel*, os limites escolhidos são aqueles no qual ainda pode-se ver o jardim enquanto para o modelo *killeroo-gold* são os que não permitem a oclusão de parte da miniatura de ouro.

Cada modelo, dependendo da complexidade do ambiente descrito, demandou uma quantidade de tempo necessária para alcançar o número de amostras por pixel estipulada de 32 amostras/pixel. Para o modelo TT, mais simples computacionalmente, foram necessários $530s$ para a renderização de uma imagem. O modelo *killeroo-gold*, também bastante simples, porém com texturas mais complexas necessita de $1700s$ para cada figura. Entretanto, o ambiente *sanmiguel*, por possuir

texturas complexas e um número muito alto de objetos demanda por volta de 20000s para cada imagem gerada.

4.2 Conclusões

A técnica de amostragem de funções plenópticas utilizando renderização de modelos, embora seja um processo que demanda uma considerável quantidade de tempo e poder computacional muito alto, é uma boa alternativa na aquisição de funções plenópticas, uma vez que a utilização de um grande número de câmeras físicas tornaria o trabalho muito complexo e exigiria total precisão e sincronismo.

Com o desenvolvimento deste método de amostragem, é possível gerar vistas tão próximas quanto o desejado e em posições quaisquer. Assim, o processo de síntese de *light fields* e da geração do par estereoscópico para a visualização 3D torna-se mais simples e automático, permitindo estudos posteriores no assunto.

4.3 Trabalhos Futuros

Ainda sobre o assunto de funções plenópticas amostradas, alguns assuntos podem vir a ser estudados ou melhorias a este trabalho são:

- Testar outros programas de renderização com o fim de avaliar vantagens e desvantagens entre eles. Assim será possível portar o *script* gerador de banco de funções plenópticas para programas diferentes;
- Utilizar modelos mais realistas, ainda que custem um tempo maior para serem gerados, e estudar a interação destes com o usuário;
- Renderizar funções plenópticas de dimensões maiores e estudar suas características.
- Realizar séries de testes subjetivos para avaliação da qualidade de experiência interativa com os modelos gerados. Estes testes determinariam a quantidade de vistas necessárias para manter a interação do usuário com o modelo bem fluida, sem a troca brusca de vistas;

- Estudar modelos de interpolação de imagens e de *motion-blur* para avaliar a quantidade de vistas que podem ser descartadas sem comprometer a fidelidade e fluidez do modelo;
- Estudar as características e comparar possíveis formas de codificação e compressão deste tipo de funções.

Apêndice A

Lista de Modelos

Figura A.1: Modelo de um Audi TT.

Figura A.2: Modelo do cenário de San Miguel.

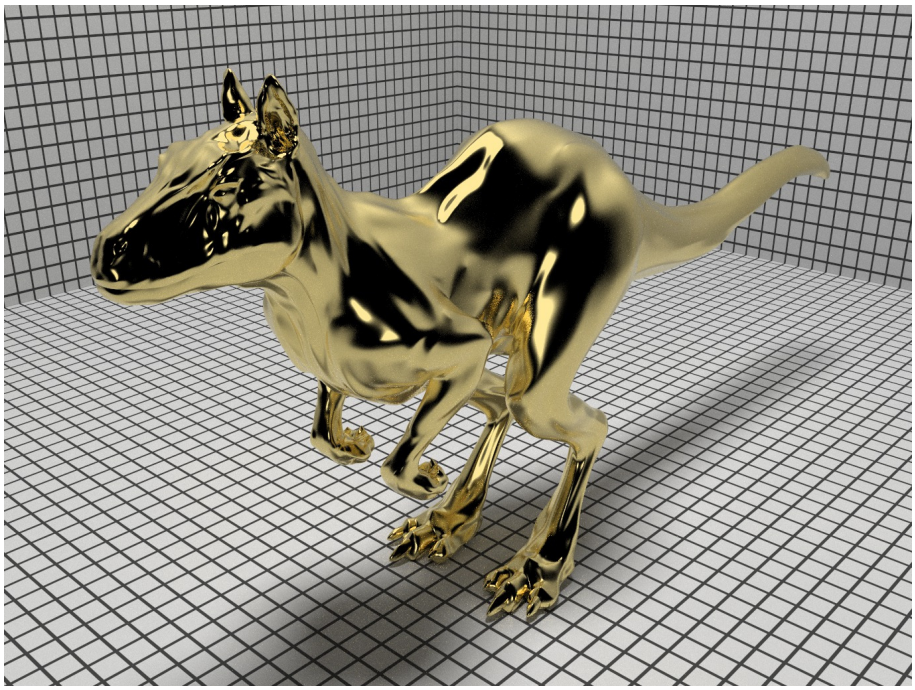


Figura A.3: Modelo Killeroo Gold.

Referências Bibliográficas

[1] PHARR, M., HUMPHREYS, G., "Physically Based Rendering", `www.pbrt.org`, acessado em fevereiro de 2013, 2012.

[2] FOLEY, J. D., WHITTED, T., "An Improved Illumination Model for Shaded Display", Proceedings of the 6th annual conference on Computer graphics and interactive techniques, 1979.

[3] NG, R., *Digital Light Field Photography*. Ph.D. thesis, Stanford University, CA, USA, July 2006.

[4] COMPUTER GRAPHICS LABORATORY, S. U., "The (New) Stanford Light Field Archive", `www.lightfield.stanford.edu/index.html`, 2008, Created by Vaibhav Vaish. Updated by Andrew Adams, acessado em julho de 2013.

[5] ADELSON, E. H., BERGEN, J. R., "The Plenoptic Function and the Elements of Early Vision". In: Landy, M. S., Movshon, A. J. (eds.), *Computational Models of Visual Processing*, Cambridge, MA, MIT Press, pp. 3–20, 1991.

[6] PHARR, M., HUMPHREYS, G., "PBRT scenes", `www.pbrt.org/scenes.php`, acessado em abril de 2013, 2012, updated on 26 May 2012.

[7] JAIN, A. K., *Fundamentals of Digital Image Processing*. Prentice Hall, Englewood Cliffs, 1989.

[8] "Dictionary of Military and Associated Terms, US Department of Defense", `www.thefreedictionary.com/interocular+distance`, 2005, acessado em agosto de 2013.

[9] LIGHT, I., MAGIC, "OpenEXR: A high dynamic-range (HDR) image file format", `http://www.openexr.com/`, 2012, Latest release: 1.7.1 / 31 July 2012, acessado em maio de 2013.

[10] CIANCIO, A., OLIVEIRA, J. F. D., RIBEIRO, F. M. L., *et al.*, "Quality perception in 3D interactive environments", *Circuits and Systems (ISCAS), 2013 IEEE International Symposium on*, pp. 9 – 12, 2013.

[11] OPENCV, "OpenCV: Open Source Computer Vision Library", `sourceforge.net/projects/opencvlibrary/`, 2011, acessado em abril de 2013.