Pontifícia Universidade Católica
do Rio de Janeiro

**Djalma Lúcio Soares da Silva**

# Uso de Estruturas Planares Extraídas de Imagens RGB-D em Aplicações de Realidade Aumentada

**Dissertação de Mestrado**

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre pelo Programa de Pós–graduação em Informática do Departamento de Informática do Centro Técnico Científico da PUC-Rio.

Orientador: Prof. Alberto Barbosa Raposo

Rio de Janeiro
Agosto de 2016

Pontifícia Universidade Católica
do Rio de Janeiro

**Djalma Lúcio Soares da Silva**

# Uso de Estruturas Planares Extraídas de Imagens RGB-D em Aplicações de Realidade Aumentada

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre pelo Programa de Pós–graduação em Informática do Departamento de Informática do Centro Técnico Científico da PUC-Rio. Aprovada pela Comissão Examinadora abaixo assinada.

**Prof. Alberto Barbosa Raposo**
Orientador
Departamento de Informática – PUC-Rio

**Prof. Luiz Carlos Pacheco R. Velho**
Instituto Nacional de Matemática Pura e Aplicada – IMPA

**Prof. Leandro Augusto Frata Fernandes**
Universidade Federal Fluminense – UFF

**DSc. Manuel Eduardo Loaiza Fernandez**
TecGraf – PUC-Rio

**Prof. Márcio da Silveira Carvalho**
Coordenador Setorial do Centro Técnico Científico – PUC-Rio

Rio de Janeiro, 1 de Agosto de 2016

**Djalma Lúcio Soares da Silva**

Graduou-se em Ciência da Computação no Centro Universitário SENAC (São Paulo). Já trabalhou nos laboratórios LSI-TEC da USP e TecGraf da PUC-Rio. Atualmente trabalha no laboratório VISGRAF IMPA, onde está envolvido em projetos de computação gráfica, realidade virtual etc.



Ficha Catalográfica

Silva, Djalma Lúcio Soares da

Uso de Estruturas Planares Extraídas de Imagens RGB-D em Aplicações de Realidade Aumentada / Djalma Lúcio Soares da Silva; orientador: Alberto Barbosa Raposo. – 2016.

68 f. : il. ; 30 cm

Dissertação (mestrado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática, 2016.

Inclui bibliografia

1. Informática – Teses. 2. RGB-D. 3. Estruturas Planares. 4. Realidade Aumentada. 5. Computação Gráfica. 6. Processamento de Imagens. 7. Visão Computacional. I. Raposo, Alberto Barbosa. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

Dedico este trabalho às duas pessoas mais importantes da minha vida, minha amada e querida esposa Alexandra e ao meu amado e adorado filho Henrique.
Eles são a força motriz que faz com que eu não desista nunca.
Amo muito vocês!

## Agradecimentos

Agradeço ao meu orientador pelos conselhos, confiança e liberdade para realizar este trabalho.

Gostaria de agradecer imensamente ao Professor Luiz Velho pelo apoio, suporte e principalmente amizade. Também por confiar e permitir que eu relizasse toda minha pesquisa no laboratório VISGRAF.

Não posso deixar de agradecer aos amigos Aldo Zang e Leandro Cruz por todo o tempo que dispensaram para me ajudar com diversos assuntos da pesquisa.

Agradeço a todos os professores e funcionários do Departamento pelos ensinamentos e pela grande ajuda.

Agradeço à PUC-Rio pelo auxílio concedido, sem o qual este trabalho não poderia ter sido realizado.

## Resumo

Silva, Djalma Lúcio Soares da ; Raposo, Alberto Barbosa. **Uso de Estruturas Planares Extraídas de Imagens RGB-D em Aplicações de Realidade Aumentada**. Rio de Janeiro, 2016. 68p. Dissertação de Mestrado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Esta dissertação investiga o uso das estruturas planares extraídas de imagens RGB-Dem aplicações de Realidade Aumentada. Ter o modelo da cena é fundamental para as aplicações de realidade aumentada. O uso de imagens RGB-D auxilia bastante o processo da construção destes modelos, pois elas fornecem a geometria e os aspectos fotométricos da cena. Devido a grande parte das aplicações de realidade aumentada utilizarem superfícies planares como sua principal componente para projeção de objetos virtuais, é fundamental ter um método robusto e eficaz para obter e representar as estruturas que constituem estas superfícies planares. Neste trabalho, apresentaremos um método para identificar, segmentar e representar estruturas planares a partir de imagens RGB-D. Nossa representação das estruturas planares são polígonos bidimensionais triangulados, simplificados e texturizados, que estão no sistema de coordenadas do plano, onde os pontos destes polígonos definem as regiões deste plano. Demonstramos através de diversos experimentos e da implementação de uma aplicação de realidade aumentada, as técnicas e métodos utilizados para extrair as estruturas planares a partir das imagens RGB-D.



## Palavras-chave

RGB-D; Estruturas Planares; Realidade Aumentada; Computação Gráfica; Processamento de Imagens; Visão Computacional;

## Abstract

Silva, Djalma Lúcio Soares da ; Raposo, Alberto Barbosa(Advisor). **Using Planar Structures Extracted from RGB-D Images in Augmented Reality Applications**. Rio de Janeiro, 2016. 68p. MSc. Dissertation – Departmento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

This dissertation investigates the use of planar geometric structures extracted from RGB-D images in Augmented Reality Applications. The model of a scene is essential for augmented reality applications. RGB-D images can greatly help the construction of these models because they provide geometric and photometric information about the scene. Planar structures are prevalent in many 3D scenes and, for this reason, augmented reality applications use planar surfaces as one of the main components for projection of virtual objects. Therefore, it is extremely important to have robust and efficient methods to acquire and represent the structures that compose these planar surfaces. In this work, we will present a method for identifying, targeting and representing planar structures from RGB-D images. Our planar structures representation is triangulated two-dimensional polygons, simplified and textured, forming a triangle mesh intrinsic to the plane that defines regions in this space corresponding to surface of objects in the 3D scene. We have demonstrated through various experiments and implementation of an augmented reality application, the techniques and methods used to extract the planar structures from the RGB-D images.



## Keywords

RGB-D; Planar Structures; Augmented Reality; Computer Graphics; Image Processing; Computer Vision;

# Sumário

# Lista de figuras

# Lista de tabelas

*If debugging is the process of removing software bugs, then programming must be the process of putting them in.*

**Edsger Dijkstra**, .

# 1
# Introdução

Apesar das estruturas planares não cobrirem todas as possibilidades de formas geométricas em uma cena capturada, elas são muito revelantes, pois estão presentes em grande parte dos ambientes feitos pelo homem, conforme pode ser visto na Figura 1.1.

Figura 1.1: Ambientes feitos pelo homem

Ao escanear uma cena com um equipamento RGB-D (45) de baixo custo, consegue-se fazer a reconstrução aproximada desta cena (Figura 1.2), sobre a qual um artista ou designer pode começar a trabalhar através de aplicações de realidade aumentada. Isto pode ser aplicado na decoração de interiores, simulação de ambientes, aluguel de imóveis, loja de móveis, preparação de exposições etc. Sendo assim, a detecção de planos através da análise de imagens RGB-D (45) é muito importante na área de reconstrução de cenas (2, 4, 5, 6).

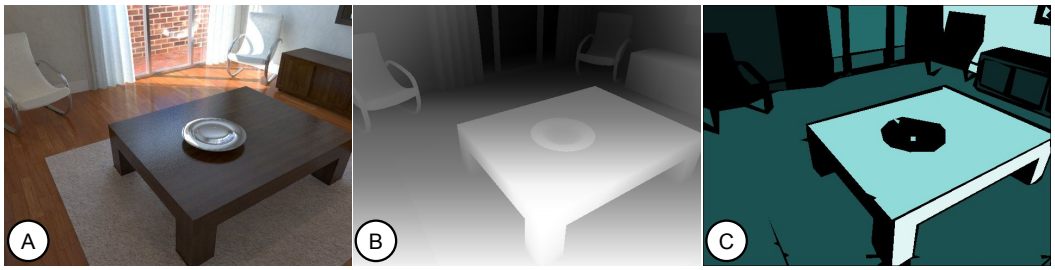



Figura 1.2: Reconstrução aproximada de uma cena capturada por um dispositivo RGB-D de baixo custo. (A) Imagem RGB; (B) Imagem com os dados de profundidade; (C) Cena reconstruída a partir da análise dos planos detectados na imagem RGB-D.

Com o aumento da produção de dispositivos móveis com suporte a imagens RGB-D, tais como o Project Tango da Google (17), Structure Sensor da Occipital (18) e o Real Sense da Intel (19), torna-se cada vez mais importante o desenvolvimento de algoritmos e aplicações que tirem vantagem dos recursos fornecidos por estes dispositivos.

Neste trabalho propomos um método para identificar, segmentar e representar estruturas geométricas planares a partir de imagens RGB-D.

## 1.1 Objetivo

Com a identificação e segmentação dos planos é possível construir uma representação geométrica parcial da cena que poderá auxiliar em algumas tarefas nos sistemas de realidade aumentada (2). Neste caso, as estruturas planares serão utilizados para projetar objetos virtuais, principalmente, nas componentes maiores como paredes, pisos e tetos que são lugares de suporte onde se pode realizar alguma interação.

O objetivo deste trabalho é construir as estruturas geométricas planares a partir dos dados fornecidos por dispositivos RGB-D de baixo custo, e usar estas estruturas em aplicações de realidade aumentada.

## 1.2 Contribuições

A principal contribuição deste trabalho é facilitar o desenvolvimento de aplicações de realidade aumentada. Isso porque, em grande parte destas aplicações, as estruturas planares são utilizadas como o principal suporte de projeção dos objetos virtuais que serão inseridos ao ambiente real.

Contudo, as estruturas geométricas planares extraídas das imagens RGB-D também podem ser utilizadas na compressão de dados. Em geral, quando se trabalha com o dado bruto, o trabalho é feito sobre uma nuvem de pontos na resolução da imagem. Mas se esta nuvem de pontos for substituída por segmentos de planos, isto é, por polígonos com textura, a quantidade de dados utilizados é reduzida. Isto torna-se importante, principalmente, durante a captura das imagens RGB-D em dispositivos móveis, pois armazenar todos os *frames* no dispositivo como nuvem de pontos demanda bastante memória. Esta compressão também é válida durante a transmissão destes *frames* através de um rede, uma vez que somente serão transmitidas as informações de câmera, os pontos que não são planos e a lista de planos da cena.

Algoritmos globais como os utilizados no KinectFusion (36) também podem ser beneficiados com as estruturas geométricas planares. Normalmente,

estes algoritmos trabalham sobre os pontos não planares. Sendo assim, os métodos apresentados neste trabalho, podem servir como um filtro, onde, somente os dados não coplanares, seriam enviados para estes algoritmos.

## 1.3 Organização da dissertação

Os capítulos apresentados neste trabalho estão organizados da seguinte maneira:

- No Capítulo 3 serão apresentados alguns trabalhos relacionados com a representação das estruturas geométricas planares.
- No Capítulo 2 serão apresentados os conceitos que são usados ao longo deste trabalho.
- No Capítulo 4 apresentaremos os métodos e técnicas utilizados para a construção da representação das estruturas geométricas planares existentes em uma cena 3D, onde estas estruturas são extraídas a partir de uma imagem RGB-D.
- No Capítulo 5 apresentaremos os detalhes do método que utilizamos para transformar em dados geométricos as regiões formadas pelos pixels classificados de uma imagem.
- No Capítulo 6 apresentaremos algumas categorias de aplicações de realidade aumentada, assim como a aplicação que implementamos para demonstrar os conceitos e técnicas propostos neste trabalho.
- No Capítulo 7 serão apresentadas as conclusões e possíveis trabalhos futuros.

# 2
# Conceitos básicos

Neste capítulo serão apresentados os conceitos que são usados no decorrer deste trabalho. As definições relacionadas às transformações de câmera são usadas na seção 4.5, as relacionadas às imagens RGB-D estão presentes na seção 4.2, as informações sobre a detecção de planos são a base de conhecimento da seção 4.3 e os conceitos de realidade aumentada são utilizados no Capítulo 6.

## 2.1
## Transformações de câmera

O modelo *pinhole* é o mais simples dos modelos de câmera existentes (31). A imagem formada em uma câmera *pinhole* é definida pela interseção dos raios luminosos com o plano do fundo da câmera, onde estes raios passam por um orifício e são emitidos ou refletidos pelos objetos da cena, como pode ser visto na Figura 2.1.

Apesar de não corresponder exatamente ao que ocorre em câmeras reais, o modelo *pinhole* fornece uma boa aproximação em diversas situações.

O funcionamento geométrico da câmera é determinado pela associação de cada ponto $P$ do espaço tridimensional ao ponto correspondente $p$ no plano de formação da imagem. Esta associação é chamada de projeção perspectiva, no qual o estudo levou ao desenvolvimento da geometria projetiva (31).

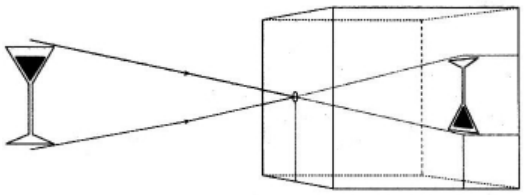


Figura 2.1: Câmera *pinhole*

Com a geometria projetiva podemos trabalhar no espaço Euclidiano, usar as transformações afins, ter uma representação unificada do espaço Euclidiano com as transformações afins e incorporar as transformações de visualização (24).

As transformações geométricas são essenciais na computação gráfica. Em geral, as aplicações usam as transformações geométricas para alterar a posição, orientação e tamanho dos objetos.

Através das transformações de câmera, cada ponto do espaço, que está expresso em um sistema de coordenadas de referência ortogonal, é associado à posição correspondente na imagem capturada por uma câmera (31). Para realizar esta associação é preciso especificar certos parâmetros que definem o comportamento da câmera. Estes parâmetros são conhecidos como extrínsecos e intrínsecos. Os parâmetros extrínsecos descrevem a posição e orientação da câmera. Os parâmetros intrínsecos, tais como a distância focal e o ponto principal, descrevem seu funcionamento interno.

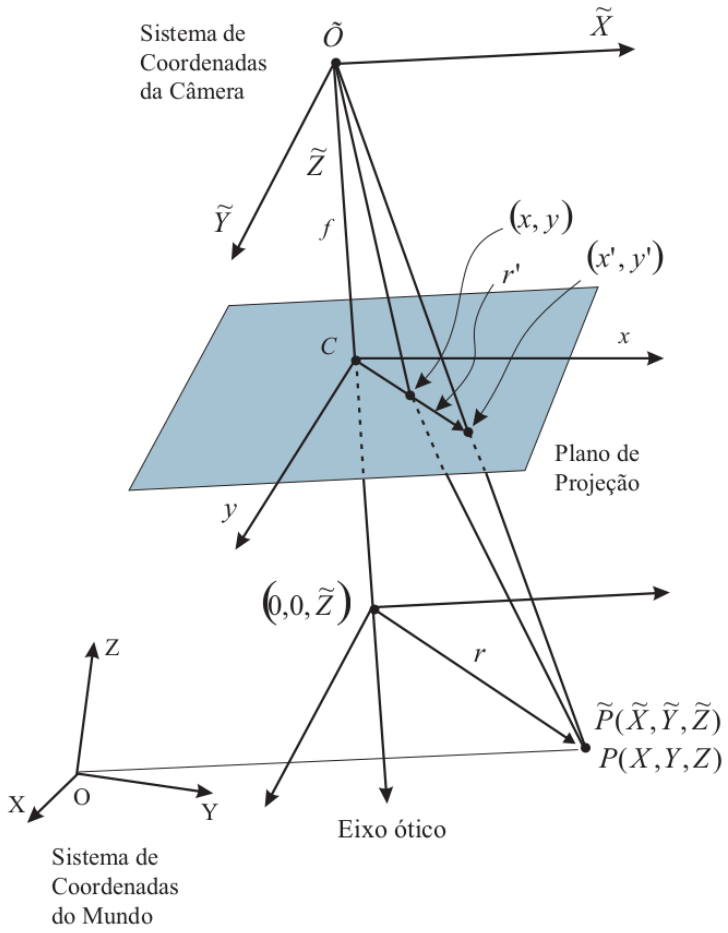



Figura 2.2: Geometria da câmera com projeção perspectiva

A transformação de câmera é expressa através da composição de transformações simples realizadas entre alguns sistemas de coordenadas. Esta composição de transformações faz a correspondência entre os pontos no espaço e pontos na imagem. Os sistemas de coordenadas considerados são os seguintes:

– Sistema de Coordenadas do Mundo (SCM): é um sistema tridimensional

que tem origem no ponto $O$ e as coordenadas com relação a este referencial são representadas por $(X, Y, Z)$, como pode ser visto na Figura 2.2.

– Sistema de Coordenadas da Câmera (SCC): é um sistema tridimensional com origem no centro ótico da câmera. Os eixos $X$ e $Y$ são paralelos às bordas da imagem que será formada sobre o plano de projeção, já o terceiro eixo é perpendicular a este plano. Na Figura 2.2, o sistema de coordenadas da câmera tem origem no ponto $\tilde{O}$ e as coordenadas neste referencial são representadas por $(\tilde{X}, \tilde{Y}, \tilde{Z})$. A distância do plano de projeção $\pi$ ao centro ótico é chamada de distância focal e é representada por $f$. No SCC o plano de projeção é o plano de equação $\tilde{Z} = f$.

– Sistema de Coordenadas de Imagem (SCI): é um sistema bidimensional localizado no plano de projeção. A origem deste sistema é o ponto $C$, que é obtido por meio da projeção ortogonal do centro ótico $\tilde{O}$ sobre o plano de projeção. As coordenadas de um ponto neste referencial são representadas por $(x, y)$.

– Sistema de Coordenadas em Pixel (SCP): é um sistema bidimensional que define a posição de um ponto da imagem com relação à grade de pixels. Suas coordenadas são expressas em pixels e, geralmente, sua origem é o canto superior (ou inferior) esquerdo da imagem. As coordenadas neste sistema são representadas por $(u, v)$.

Considerando uma câmera *pinhole*, o processo para mapear um ponto que está no SCC para o SCP consiste na seguinte sequência de transformações:

$$\text{SCM} \rightarrow \text{SCC} \rightarrow \text{SCI} \rightarrow \text{SCP}$$

O primeiro passo para obter a posição na imagem correspondente a um ponto $(X, Y, Z)$ no espaço é representar essas coordenadas tridimensionais no SCC. Esta mudança de sistema de coordenadas é realizada através da matriz abaixo, onde R (rotação) e T (translação) são os parâmetros extrínsecos da câmera. Os detalhes da construção desta matriz são encontrados em (31).

$$\begin{bmatrix} \tilde{X} \\ \tilde{Y} \\ \tilde{Z} \\ 1 \end{bmatrix} = \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

A transformação do SCC para o SCI é realizada através de um projeção perspectiva, com a câmera a uma distância focal $f$ e escrita como:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \simeq \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \tilde{X} \\ \tilde{Y} \\ \tilde{Z} \\ 1 \end{bmatrix} \frac{1}{\tilde{Z}}$$

A transformação afim que leva do SCI para o SCP é modelada da seguinte forma:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & \tau & u_c \\ 0 & s_y & v_c \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Os coeficientes da matriz anterior, juntamente com a distância focal $f$, formam os parâmetros intrínsecos da câmera. Os significados dos parâmetros são os seguintes:

- $s_x$ e $s_y$ representam o número de pixels por unidade de comprimento nas direções horizontal e vertical, respectivamente; onde, na maioria das câmeras, $s_x$ e $s_y$ são iguais, isto é, os pixels são quadrados;
- $u_c$ e $v_c$ fornecem a posição, em pixels, da projeção ortogonal $C$ da origem sobre o plano de projeção. Na maior parte das câmeras, $C$ está no centro da imagem e os valores de $u_c$ e $v_c$ são iguais a metade das dimensões da imagem;
- $\tau$ é a tangente do ângulo que as colunas de pixels formam com a perpendicular às linhas, idealmente, na maioria das câmeras, as colunas são perpendiculares às linhas, isto é, $\tau = 0$.

A transformação que leva um ponto do SCM à sua projeção em SCP, pode ser obtida através da composição das transformações anteriormente descritas. Representando as coordenadas do ponto no SCM como $[P]$ e da imagem no SCP como $[p]$, temos:

$$\big[p\big] \simeq \begin{bmatrix} s_x & \tau & u_c \\ 0 & s_y & v_c \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix} \big[P\big]$$

ou ainda,

$$\left[p\right] \simeq \begin{bmatrix} fs_x & f\tau & u_c \\ 0 & fs_y & v_c \\ 0 & 0 & 1 \end{bmatrix} \left[R \quad T\right] \left[P\right]$$

A matriz $K = \begin{bmatrix} fs_x & f\tau & u_c \\ 0 & fs_y & v_c \\ 0 & 0 & 1 \end{bmatrix}$ é chamada de matriz de calibração e reúne os parâmetros intrínsecos da câmera, enquanto $\left[R \quad T\right]$ representa os extrínsecos.

Na maior parte dos casos é conveniente expressar a matriz de calibração como $K = \begin{bmatrix} f_x & \tau & u_c \\ 0 & f_y & v_c \\ 0 & 0 & 1 \end{bmatrix}$.
Pois, a menos que os fabricantes informem os valores de $f$, $s_x$ e $s_y$; não é possivel estimar seus valores, já que estes parâmetros aparecem na transformação de câmera através de seus produtos $fs_x$ e $fs_y$.

## 2.2 Imagem RGB-D

O Microsoft Kinect foi um dos primeiros dispositivos de fácil acesso a fornecer imagens RGB-D (45). O sucesso nas vendas deste dispositivo, tanto para entretenimento, quanto para uso em pesquisa, provavelmente, motivou o desenvolvimento dos dispositivos Asus Xtion Pro Live (20), o Project Tango do Google (17), o Realsense da Intel (19), Structure Sensor da Occipital (18) dentre outros.

A imagem RGB-D (Figura 2.3) é composta por aspectos fotométricos e informações geométricas da cena; onde, em cada pixel na imagem da cena capturada, os aspectos fotométricos são uma combinação de três canais de cores (um para cada cor primária: vermelho, verde e azul) e as informações geométricas são os dados de profundidade destes pixels (45).

Figura 2.3: Imagens RGB e D capturadas com o dispositivo Asus Xtion Pro Live

Na imagem RGB-D, os aspectos fotométricos da cena, normalmente, são capturados por uma câmera do tipo *pinhole*.

As técnicas de captura de dados geométricos baseadas em imagem podem ser divididas em: estéreo passivo e estéreo ativo. De maneira semelhante ao olho humano as técnicas de estéreo passivo utilizam duas ou mais câmeras calibradas e calculam correspondências entre as projeções de pontos em imagens distintas para medir a profundidade dos pontos na cena. As técnicas de estéreo ativo tem como base os mesmos princípios da estereoscopia passiva, contudo utilizam um par câmera-projetor, que ilumina a cena com um padrão de luz estruturada. O sistema óptico do projetor possui um papel semelhante ao da segunda câmera nas medições estereoscópicas e o padrão projetado facilita a correspondência entre os pontos (31). A maioria dos dispositivos RGB-D de baixo custo utilizam a técnica de estéreo ativo.

## 2.3 Realidade aumentada

Furht (35) define realidade aumentada (RA) como uma visão direta ou indireta em tempo real de um ambiente físico do mundo real que foi aprimorado ou aumentado por informações geradas pelo computador. Sendo assim, RA permite a combinação de objetos reais com objetos virtuais, onde esta combinação pode melhorar para o usuário a percepção do mundo real.

Em realidade aumentada, o alinhamento entre os objetos virtuais e as imagens geradas pela câmera pode ser feito por diferentes métodos de visão computacional. Esses métodos normalmente consistem em dois estágios: rastreamento (*tracking*) e reconstrução.

O rastreamento faz uso da detecção de pontos característicos, detecção de bordas ou algum outro método de processamento de imagens para interpretar as imagens fornecidas pela câmera. A maioria das técnicas de rastreamento pode ser divididas em duas classes: *feature-based* (33) e *model-based* (33). Os métodos de *feature-based* consistem em descobrir a conexão entre *features* em uma imagem 2D e suas coordenadas 3D no mundo (34). Os métodos *model-based* fazem uso do modelo dos objetos que estão sendo rastreados, como os modelos de CAD ( *Computer Aided Design*). Uma vez feita a conexão entre a imagem 2D e suas coordenadas 3D no mundo, é possível encontrar a posição e orientação da câmera através da reprojeção das coordenadas 3D da *feature* sobre as coordenadas da imagem 2D observada. O estágio de reconstrução utiliza as informações do estágio de rastreamento para reconstruir o sistema de coordenadas do mundo real. As aplicações de RA utilizam este sistema de coordenadas para adicionar informações ou objetos 3D à cena real.

Para fazer realidade aumentada *model-based*, isto é, sem a utilização dos marcadores, é preciso ter o modelo da cena. Usar imagens RGB-D ajuda bastante nesse processo, pois elas fornecem a geometria e aspectos fotométricos da cena.

### 2.3.1 Realidade aumentada em dispositivos móveis

Devido ao aumento da capacidade de processamento do *hardware* de dispositivos móveis e a melhoria dos algoritmos de visão computacional e processamento de imagem, tornou-se comum ter aplicações de realidade aumentada nestes dispositivos (14, 15, 16). Recentemente, surgiram os dispositivos móveis que fornecem imagem RGB-D, tais com o Google Tango (17), Structure Sensor da Occipital (18) e o Realsense da Intel (19). Estes dispositivos, além de fornecer a imagem RGB-D que possuem os dados geométricos da cena, também fornecem o dados de *tracking* (posição e orientação da câmera) através dos diversos sensores que possuem.


Figura 2.4: Dispositivos RGB-D de baixo custo

No contexto de RA móvel, com dispositivos como estes, temos tanto a imagem RGB-D como os dados de *tracking* da câmera. Desta maneira o desenvolvimento de aplicações de RA não precisam se preocupar com tarefas como a calibração de câmera, que métodos utilizar para realização de *tracking* etc.

## 2.4 Detecção de planos

Existem diversos algoritmos que têm como propósito a detecção de planos a partir de uma nuvem de pontos. Os métodos que tem sido amplamente utilizados são os baseados em RANSAC (32), *region growing* (37) e transformada de Hough (38).

Neste trabalho, utilizamos os planos detectados através da transformada de Hough em imagens RGB-D, desenvolvida por Souza *et al.* (50) e explicada

na seção 4.3. Sendo assim, esta seção somente aborda a detecção de planos através da transformada de Hough.

A transformada de Hough, desenvolvida por Paul Hough (38), inicialmente era utilizada para a detecção de linhas em imagens. Contudo ela foi estendida para a identificação de círculos, elipses, planos e outras formas analíticas e não-analíticas. A variante da transformada, utilizada atualmente para retas, é a desenvolvida por Duda e Hart (39), também conhecida por SHT (*Standard Hough Transform*), a qual substituiu a parametrização que utiliza a equação reduzida da reta, pela parametrização baseada na equação normal da reta 2-1:

$$\rho = x\cos(\theta) + y\sin(\theta), \tag{2-1}$$

onde $x$ e $y$ são as coordenadas de um ponto pertencente à reta, $\rho$ é a distância da reta até a origem do sistema de coordenadas da imagem, e $\theta$ é o ângulo entre a normal da reta com o eixo $x$. Estender esta parametrização para 3D, faz com que a transformada de Hough suporte a detecção de planos.

$$\rho = x\cos(\theta)\sin(\phi) + y\sin(\phi)\sin(\theta) + z\cos(\phi) \tag{2-2}$$

Na equação 2-2, $x$, $y$ e $z$ são coordenadas cartesianas das amostras na nuvem de pontos, $\theta \in [0°, 360°)$ e $\phi \in [0°, 180°]$ são as coordenadas polares do vetor normal do plano, e $\rho \in \mathbb{R}_{\geq 0}$ é a distância do plano à origem do sistema de coordenadas.

Para a detecção de planos em uma nuvem de pontos, a transformada padrão de Hough ou SHT (do inglês *Standard Hough Transform*) utiliza a equação 2-2 e ao iterar sobre cada amostra desta nuvem, a SHT realiza a votação no acumulador para todos os possíveis planos que passam por esta amostra. Isto é, para cada trio de coordenadas $x$, $y$ e $z$, a iteração é feita para todas as combinações de $\theta$ e $\phi$. Mesmo discretizando os valores de $\theta$ e $\phi$, o custo computacional de uma SHT é da ordem $O(|P|N_\theta N_\phi)$, onde $|P|$ é o número de pontos da nuvem de pontos $P$; e $N_\theta$ e $N_\phi$ são o número de células do acumulador, na discretização dos ângulos de $\theta$ e $\phi$, respectivamente.

Devido ao alto custo computacional da SHT, algumas técnicas foram propostas para acelerar o processo de votação. As estratégias da PHT (*Probabilistic Hough Transform*) (40), APHT (*Adaptive Probabilistic Hough Transform*) (41) e PPHT (*Progressive Probabilistic Hough Transform*) (42); têm como foco a redução do tempo de execução através do uso de um subconjunto dos pontos da nuvem de pontos $P$, em oposição à criação de novos algoritmos que efetivamente venham a reduzir o custo assintótico do processo de votação (1).

Diferente das estratégias anteriores, Limberger e Oliveira (1) utilizam uma abordagem determística e com complexidade computacional $O(n \log n)$,

no número de amostras de entrada, para a detecção de regiões planares em nuvem de pontos não estruturadas. Tendo como base a KHT (*Kernel-based Hough Transform*) desenvolvida por Fernandes e Oliveira (3), eles desenvolveram a 3-D KHT (1), que agrupa conjuntos de pontos aproximadamente coplanares e deposita votos para estes conjuntos (ao invés das amostras individuais) em um acumulador esférico.

A seguir são descritos, de forma sucinta, os estágios da 3-D KHT (1).

- Agrupar as amostras aproximadamente coplanares: o agrupamento é realizado através da subdivisão de uma octree. O processo de subdivisão é iniciado a partir do nó pai que possui todo o conjunto de dados. A cada subdivisão são verificadas as seguintes condições: a quantidade de amostras necessárias para a criação de um *cluster* e as proporções da distribuição dos dados através da PCA (*Principal Component Analysis*), que identifica quando um conjunto de pontos representa um plano. Quando as duas condições forem satisfeitas, a subdivisão para, caso contrário o processo continua até que o número de amostras seja menor que um determinado valor.
- Calcular os núcleos (*kernels*) trivariados para a votação: para que os votos sejam depositados para cada *cluster* no acumulador esférico, é necessário conhecer as variâncias das distribuições dos *clusters* em relação aos parâmetros do acumulador. Para determinar estas variâncias, é preciso propagar para o espaço paramétrico as incertezas em relação ao eixo cartesiano e que foram calculadas por PCA durante o agrupamento das amostras. Para isso, é utilizada uma propagação de primeira ordem, multiplicando a matriz de covarância nas coordenadas $(x, y, z)$ pela matriz Jacobiana à esquerda e sua transposta à direita, propagando assim as variâncias de $(x, y, z)$ para $(\theta, \phi, \rho)$.
- Votar para *clusters* utilizando distribuições gaussianas: com a matriz de variância calculada nas coordenadas corretas, é utilizada uma distribuição gaussiana trivariada para amostrar a quantidade de votos que será depositada em cada célula.
- Detecção de picos: o processo de detecção de picos no acumulador, onde estes picos representam os planos mais importantes na nuvem de pontos. Neste processo é utilizado um vetor para armazenar as células do acumulador que receberam votos durante o processo de votação. Este vetor é convoluído por um filtro gaussiano que utiliza os 6 vizinhos imediatos, isto é, variando um eixo a cada amostragem. Em seguida, os valores do vetor são classificados em ordem decrescente e iterados

linearmente. Os vizinhos de cada célula são examinados em busca de células que já foram inspecionadas neste processo. Caso nenhum vizinho de uma célula tenha sido anteriormente inspecionado, esta célula é escolhida como pico.

A abordagem apresentada por (1) é a base da D-KHT (*Depth Kernel-based Hough Transform*), criada por Souza *et al.* (50) e utilizada neste trabalho para a detecção de planos em imagens RGB-D.

# 3
# Trabalhos relacionados

Como o objetivo deste trabalho é a construção das estruturas geométricas planares a partir de imagens RGB-D, e estas estruturas são iniciadas através da detecção de planos, então é essencial uma revisão dos trabalhos que realizam este processo.

Nos diversos trabalhos mencionados na seção 3.1, o processo de criação da representação das estruturas geométricas planares é intrínsica ao processo de detecção dos planos e há poucas informações sobre esta representação. Na seção 3.2 verificamos as representações das estruturas geométricas utilizadas em alguns trabalhos que utilizam imagens RGB-D como entrada.



## 3.1
## Detecção de planos

A transformada de Hough (38) é um dos métodos mais utilizadas para a detecção de planos em nuvem de pontos, conforme apresentado por Borrmann *et al.* (5), e onde são mostradas algumas das variações da transformada de Hough para este propósito, além de evidenciar que um dos maiores problemas relacionados à performance está na representação do acumulador.

Hulik *et al.* (4) apresentam diversas otimizações na transformada de Hough que aprimoram a extração de planos a partir de uma nuvem de pontos obtidas através de imagens RGB-D fornecidas por dispositivos RGB-D de baixo custo. Dentre as melhorias propostas, estão o tratamento do alto consumo de memória utilizado pelo espaço de parâmetros e tempo de processamento. Entretanto, para estas otimizações, das características exploradas das imagens RGB-D, o aspecto utilizado foi a coerência temporal entre os *frames* fornecidos pelos dispositivos.

Limberger e Oliveira (1) elaboram uma técnica com abordagem determinística, baseada na transformada de Hough, para detectar em tempo real, regiões planares em uma nuvem de pontos não estruturada. A solução apresentada é robusta a ruído, independente da amostragem da distribuição e utiliza uma estratégia eficiente de votação para a detecção dos planos. Esta técnica somente utilizada para nuvens de pontos não estruturadas, então ela não explora

a estrutura fornecida pela imagens RGB-D, quando esta nuvem é construída a partir dispositivos RGB-D, como é feito por Souza *et al.* (50).

A detecção das regiões planares em imagens RGB-D é realizada nos trabalhos de Tang *et al.* (7) e Erdogan *et al.* (6). Todavia, para realizar a detecção, eles utilizam RANSAC (32), que apesar de ser robusto com relação a ruído, é não-determinístico, somente recupera um plano por vez e é iterativo, isto é, pode demorar para fazer a detecção.

No trabalho de Hemmat *et al.* (53) a detecção das regiões planares em imagens RGB-D utiliza o método *region growing* (37) baseado na proposição geométrica (54), onde é indicado que a intersecção de cada par linhas encontra-se em um plano. O algoritmo proposto detecta as curvas de bordo na imagem de profundidade, procura por segmentos de linha que se intersectam nas regiões delimitadas por estas curvas, e a região planar é construída através da combinação destas linhas, pois elas estão sobre o mesmo plano. Esta é uma abordagem interessante. Contudo, para atingir a performance necessária que as aplicações de realidade aumentada necessitam, é preciso realizar a implementação deste algoritmo em GPU.

Em nosso trabalho, utilizamos a técnica desenvolvida por Souza *et al.* (50) para a detecção das regiões planares, onde a transformada de Hough tem como foco a detecção de planos em imagens RGB-D. Na seção 4.3 são encontradas as informações relacionadas a esta técnica.

## 3.2 Representação geométrica de estruturas planares

Alguns sistemas SLAM (*Simultaneous Localization And Mapping*) (46) utilizam a representação do tipo *point-based* para representar as estruturas planares, como a utilizada por P. Henry *et al* (47) e por Weise et al. (48). Já a representação utilizada pelos sistemas baseados no KinectFusion (36) é uma malha densa e é muito importante para determinados tipos de aplicações.

As estruturas geométricas planares representadas através do tipo *point-based*, além de possuirem um custo de armazenamento alto (49), também adicionam um custo computacional considerável, mesmo quando aplicada uma simples transformação de rotação ou translação, pois estas transformações devem ser aplicadas a cada ponto. Devido ao *hardware* das GPUs atuais possuírem diversas funções voltadas para trabalhar com malhas triangulares (49), as representações que utilizam malhas densas têm seu custo computacional diminuído, entretanto o problema com o alto custo de armazenamento persiste. Então, estas representações não são o que procuramos para as estruturas planares a serem utilizadas em aplicações de realidade aumentada.

Nguyen *et al.* (2) apresentam uma boa representação para estruturas planares constituída por polígonos triangulados e simplificados. Contudo, são empregados somente métodos de geometria computacional para construção e representação destas estruturas geométricas, onde a utilização destes métodos pode gerar alguns problemas numéricos, o que pode tornar a geometria inconsistente com a topologia.

Nossa abordagem para construção e representação das estruturas geométricas planares utiliza polígonos triangulados, simplificados e texturizados no espaço do plano. Uma das principais vantangens da nossa representação é que ela é compacta, isto é, possui poucos triângulos. Além disso, utilizamos basicamente métodos de processamento de imagens para realizar sua construção. Sendo assim, não temos os problemas númericos que podem tornar a geometria inconsistente com a topologia. Nossa representação é apresentada e detalhada na seção 4.5.

# 4
# Estruturas planares em imagens RGB-D

Neste capítulo, apresentaremos um panorama da primeira parte desta dissertação, onde propomos um método para identificar, segmentar e representar estruturas planares em imagens RGB-D.

## 4.1
## Visão geral

Buscamos, neste trabalho, uma boa representação das estruturas planares existentes em uma cena 3D, onde estas estruturas são extraídas a partir de uma imagem RGB-D. A Figura 4.1 ilustra o *pipeline* para a criação da representação da estrutura planar, onde as regiões planares são detectadas a partir das informações da cena capturada, para que enfim sejam representadas por uma malha poligonal bidimensional.

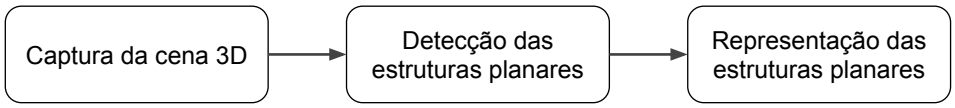



Figura 4.1: *Pipeline* para a criação da representação das estruturas planares.

A estrutura planar é representada pela região de um plano definida através de uma malha poligonal bidimensional. Como o plano pertence a uma cena que possui atributos visuais, então essa malha possui uma textura associada a esta região. É através desta textura que é feito o mapeamento de novos atributos visuais a cena em aplicações de realidade aumentada.

Para determinar as estruturas planares em uma imagem RGB-D de uma cena 3D capturada, primeiramente, necessitamos detectar os planos existentes nesta cena. A transformada de Hough é utilizada para realizar esta detecção, onde, durante este processo, diversos pontos da imagem RGB-D são marcados como pertencentes a um determinado plano. A marcação destes pontos é feita durante o processo de criação da *quadtree* que é a estrutura de dados utilizada para armazenar os pontos pertencentes aos planos detectados. Os detalhes desta detecção são encontrados na seção 4.3.

Apesar da detecção ser eficaz, existem alguns casos em que alguns pontos não puderam ser classificados como pertecentes a alguma região planar na imagem RGB-D. Neste caso, é feita uma análise mais detalhada destes pontos. Esta análise é realizada com o auxílio da *quadtree* e dos planos detectados no processo de detecção dos planos. A análise dos pontos é explicitada na seção 4.4.

Com todos os pontos devidamente classificados, temos definidas regiões no plano formadas por estes pontos, e estas regiões são as estruturas planares que buscamos representar. Elaboramos um método que transforma as regiões formadas pelos pixels classificados de uma imagem em dados geométricos. As tarefas realizadas para fazer esta transformação e ao final termos a representação das estruturas planares, são as seguintes:

- Cálculo das curvas de bordo de cada região definida.
- Simplificação destas curvas de bordo.
- Triangulação dos polígonos delimitados pelas curvas de bordo simplificadas.
- Determinação das coordenadas de textura dos polígonos.
- Mapeamento dos triângulos para o espaço do plano através de um transformação afim.

Para simplificar os polígonos triangulados, os passos anteriormente descritos são executados duas vezes. Na seção 4.5 descrevemos o porquê desta segunda execução e fornecemos mais informações sobre os passos.

Na etapa de geração dos polígonos 2D do processo de construção das estruturas geométricas, levamos estes polígonos para o espaço do plano. Para realizar essa mudança de espaço precisamos ter definido o sistema de coordenadas do plano. A geração deste sistema de coordenadas, bem como as transformações estão descritas na seção 4.6.

Neste momento, temos como resultado a representação das estruturas planares que são polígonos bidimensionais triangulados e simplificados que estão no sistema de coordenadas do plano, onde os pontos destes polígonos definem as regiões deste plano.

Os passos anteriormente descritos são utilizados para a construção da representação das estruturas planares em um *frame*. Mas, e se quisermos que essa representação seja composta com as informações de todos os *frames* capturados ao longo do tempo? Essa informação é dada na seção 4.8.

## 4.2 Captura da cena

Nesse trabalho, é fundamental ter um conjunto de imagens RGB-D de diversas cenas. Contudo, a aquisição destes dados não é uma tarefa simples, pois demanda um tempo considerável para capturar estas informações. Felizmente, o pesquisador Qianyi Zhou (25, 26) fornece um amplo conjunto de imagens RGB-D tanto de cenas sintéticas, quanto de cenas capturadas usando o dispositivo *Asus Xtion Pro Live*. Além das imagens RGB-D, ele também fornece os dados de *tracking* da câmera para os dois tipos de cena. Com isso, dedicamos nossos esforços no desenvolvimento das ferramentas que manipulam estas informações.

A imagem RGB-D é usada como entrada por todos os processos durante a construção das estruturas planares. Contudo, além das informações de profundidade e cor, estes processos também necessitam de outras informações, como os parâmetros intrínsecos e extrínsecos da câmera e das coordenadas locais de um ponto. Para auxiliar estes processos utilizamos uma estrutura que encapsula todas estas informações e que chamamos de RGBD-Frame. A Figura 4.2 ilustra sua construção.

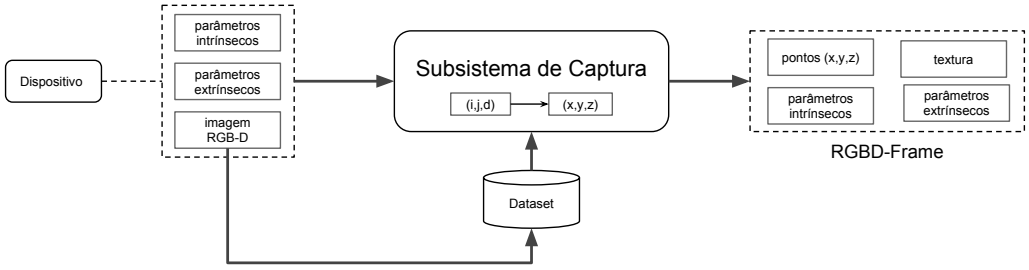



Figura 4.2: Processo de construção do RGBD-Frame, que é a estrutura que encapsula as informações da câmera, assim como a textura e as coordenadas locais dos pontos convertidos a partir dos pixels.

Os parâmetros intrínsecos e extrínsecos da câmera normalmente são obtidos através de algum método de calibração de câmera. Na seção 2.1 são encontrados alguns detalhes do que são estes parâmetros e como são usados. Os parâmetros intrínsecos e extrínsecos da câmera que capturou o conjunto de imagens RGB-D que utilizamos, são fornecidos com este conjunto de imagens.

Nos dispositivos móveis como o Google Tango (17) e Structure Sensor (18), os parâmetros intrínsecos são fornecidos pelos fabricantes e os parâmetros extrínsecos são gerados a partir de um conjunto de sensores destes dispositivos.

Para obter as coordenadas locais de um ponto da imagem RGB-D necessitamos converter cada pixel (i,j,d) no ponto (x,y,z). Esta conversão é

realizada usando os parâmetros intrínsecos ($c_x, c_y, f_x$ e $f_y$) da câmera conforme mostrado na equação 4-1:

$$\begin{aligned} z &= d, \\ x &= \frac{(i - c_x)z}{f_x}, \\ y &= \frac{(j - c_y)z}{f_y}. \end{aligned} \tag{4-1}$$

Ao final do processo de captura, temos a estrutura RGBD-Frame contendo a textura que são os valores de RGB da imagem RGB-D, os parâmetros intrínsecos e extrínsecos da câmera e os pontos em coordenadas locais convertidos do pixel (i,j,d) para o ponto (x,y,z).

## 4.3 Detecção de planos

A transformada de Hough é uma das técnicas mais utilizadas para a detecção de planos, conforme visto na seção 2.4.

Neste trabalho usamos uma variante da transformada de Hough focada na detecção de planos em imagens RGB-D chamada de D-KHT (*Depth Kernel-based Hough Transform*), desenvolvida por Souza *et al.* (50) e que tem como base a 3-D KHT (1). A técnica proposta é bastante robusta à presença de ruídos e, em relação ao tempo de processamento, é bem eficiente. Esta variante consiste, basicamente, em subdivir a imagem RGB-D, agrupar os pixels aproximadamente coplanares e fornecer uma lista contendo os planos mais significativos.

A D-KHT não realiza uma subdivisão sobre o eixo $z$, como é feito por Limberger e Oliveira (1) na 3-D KHT, pois os pontos das imagens RGB-D não possuem valor negativo no eixo $z$ e somente existe um valor armazenado para estes pontos, isto é, não existe sobreposição de pontos ao longo do eixo $z$. Desta maneira, a *octree* utilizada pela 3-D KHT, em uma nuvem de pontos não estruturada, foi substituída por uma *quadtree* sobre os eixos $i$ e $j$ de uma imagem RGB-D.

De forma resumida, a técnica proposta pela D-KHT subdivide a imagem RGB-D como uma *quadtree* contendo nós com pontos aproximadamente coplanares; utiliza a PCA (*Principal Component Analysis*) (43) para identificar o plano que melhor se ajusta aos pontos contidos em um nó da *quadtree* e durante esta etapa, com base neste plano, é calculado o *kernel* gaussiano do nó;

em seguida realiza a votação para cada *kernel* no mapa de acumuladores; e por fim, encontra os máximos locais do mapa de acumuladores, cujas coordenadas correspondem aos parâmetros de cada plano detectado. Este processo pode ser observado na Figura 4.3. Os detalhes de cada etapa, bem como o embasamento matemático, são encontrados em Souza *et al.* (50).

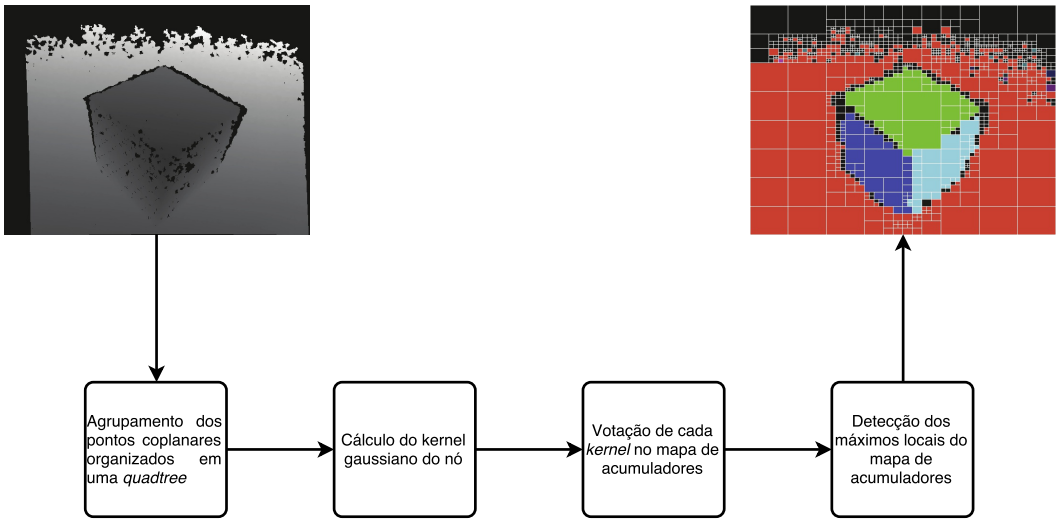



Figura 4.3: Etapas da D-KHT que tem como entrada os dados de profundidade da imagem RGB-D e, como saída, retorna uma *quadtree* contendo os planos mais relevantes da cena capturada.

A *quadtree* gerada e os planos detectados são o que usamos como entrada em nosso trabalho. Durante a construção da *quadtree* são geradas duas listas, uma contendo os nós que foram classificados como coplanares (ou completos) e outra com os nós classificados como não-identificados, como pode ser visto na Figura 4.4. Todos os pontos dos nós coplanares pertecem a um plano. Contudo, para os nós não-identificados, é preciso saber se os seus pontos pertencem ou não a um plano. A análise dos nós do tipo não-identificados é vista na seção 4.4.

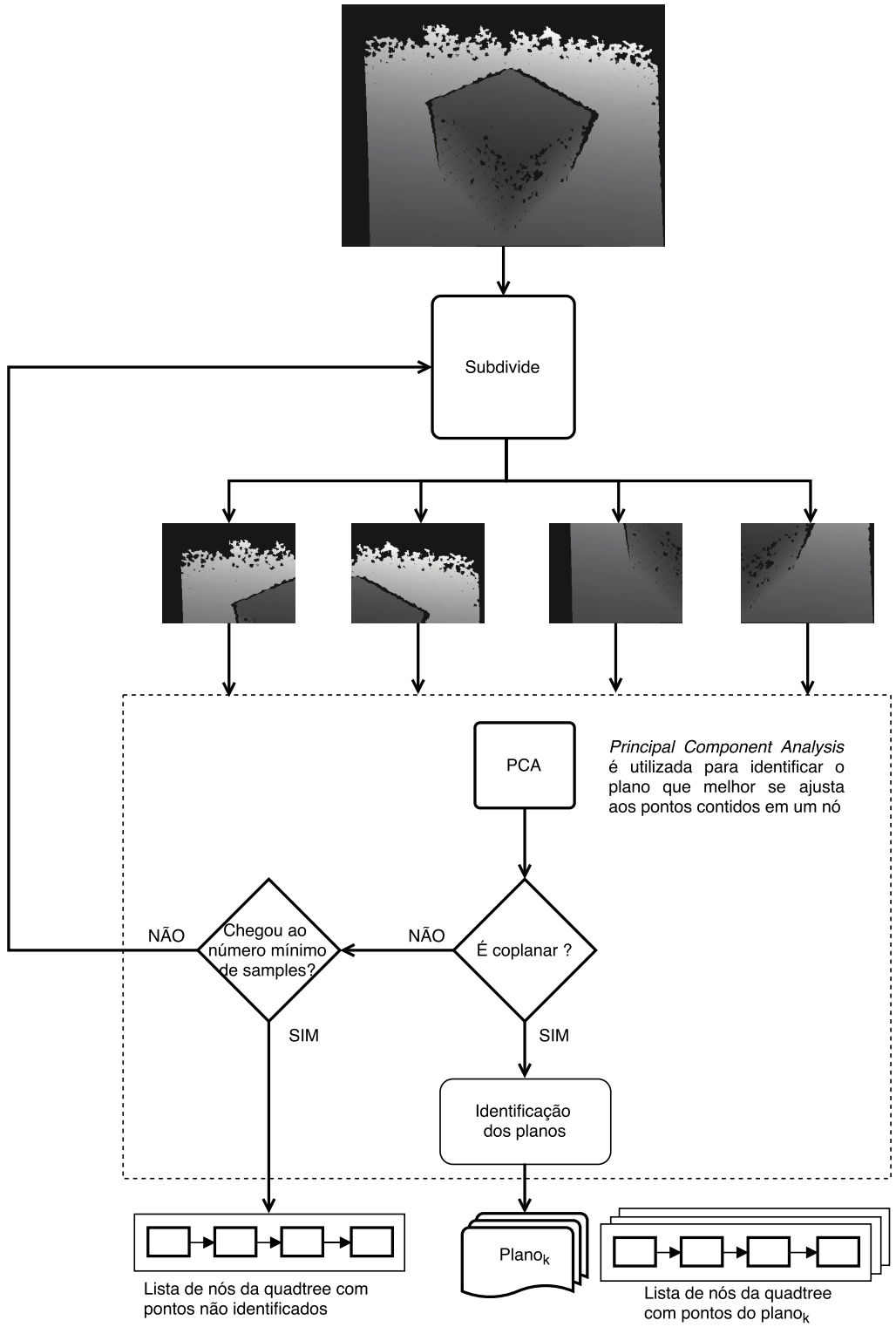



Figura 4.4: Processo de criação da *quadtree*, onde é feito o agrupamento das regiões da imagem que contem pontos aproximadamente coplanares.

## 4.4 Determinação dos pontos do plano

Durante o processo de identificação dos planos gera-se uma *quadtree* que possui dois tipos de nós: os completos e os não-identificados. Nos nós completos, todos os pontos que estão entre as coordenadas de suas fronteiras estão sobre um plano. Já nos nós não-identificados, estão todos os pontos que o método de detecção dos planos não conseguiu classificar como pontos pertencentes a algum plano. Esta seção mostra como determinar a que planos os pontos dos nós não-identificados pertencem.

A Figura 4.5 apresenta uma cena capturada em que há um jarro e alguns objetos que não são planares, então esses objetos realmente não pertencem a nenhum plano. Nesta cena, também há objetos planares como as esquadrias da porta ou janela que pertencem a algum plano mas não foram classificados devido a definição de um número mínimo de amostras. Neste caso, é necessário definir a que plano os pontos destes nós pertencem.


Figura 4.5: Objetos não planares e não determinados da cena

Usando a equação do plano, basta verificar a distância do ponto ao plano para saber se este ponto pertence ou não ao plano. Se o ponto estiver abaixo de um determinado $\epsilon$ de distância, a qual chamamos de tolerância local, então

este ponto pertence ao plano, caso contrário ele é descartado. O cálculo desta distância é apresentado na equação 4-2 (27).

Seja o ponto $P_a = (x_a, y_a, z_a)$ e a equação do plano $\Pi : Ax+By+Cz+D = 0$, então a distância do ponto $P_a$ ao plano $\Pi$ é dada por:

$$\text{distância}_{P_a,\Pi} = \frac{Ax_a + By_a + Cz_a + D}{\sqrt{A^2 + B^2 + C^2}}. \tag{4-2}$$

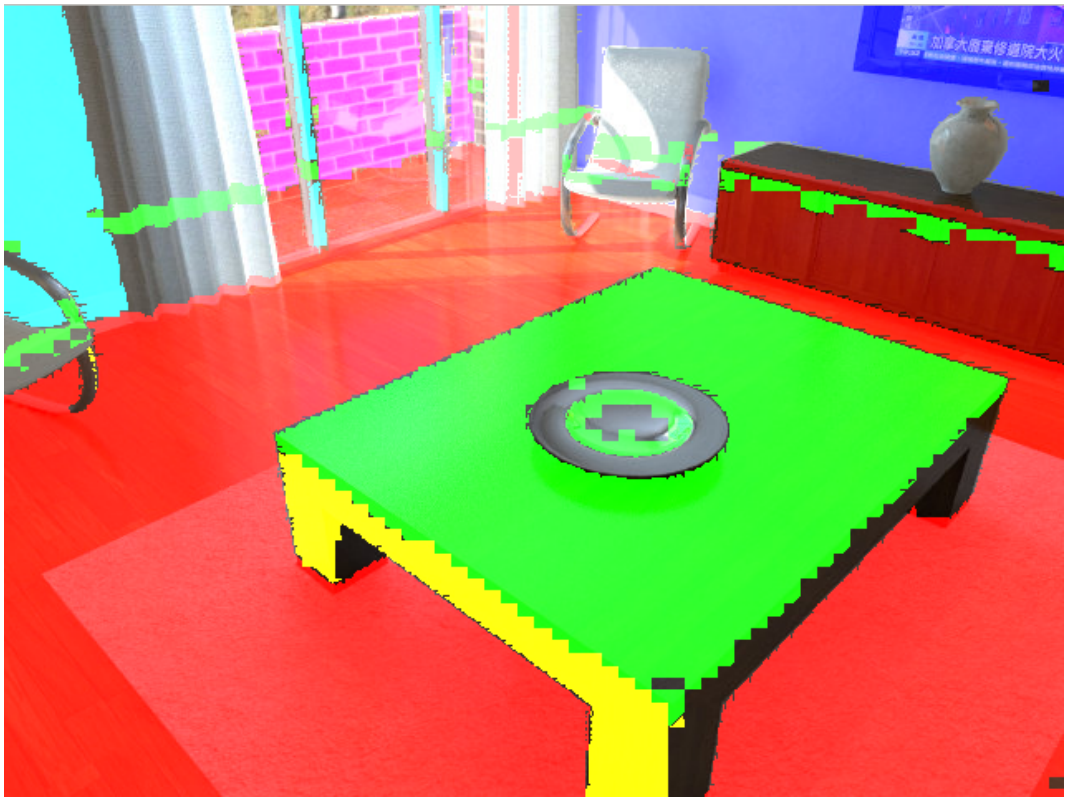


Figura 4.6: Resultado do método de determinação dos pontos do plano, onde é utilizada somente a distância de um ponto ao plano, para definir a que plano os pontos pertencem.

Apesar de eficiente, o método da distância ao plano não é completamente robusto, pois ele vale somente para um ponto e não para um conjunto de pontos que pertence a um objeto que representa parte do plano. Como pode ser visto na Figura 4.6, um ponto pode estar dentro da tolerância local em mais de um plano.

Para saber se esse ponto pertence ou não ao objeto que representa um segmento de um plano, é necessário fazer uma análise mais detalhada, pois tal objeto é composto não somente por um ponto que está sobre o plano, mas sim por ele e sua vizinhança. Essa vizinhança é facilmente obtida, pois ela é fornecida pela estrutura da imagem RGB-D.

Usando a informação da vizinhança, podemos fazer o teste básico da distância com o ponto analisado. Isto é, medindo a distância do ponto ao plano, como informado anteriormente. Caso o ponto esteja a tal distância, então olhamos sua vizinhança. Se todos os pontos ao redor também estiverem sobre o plano, então o ponto é marcado como pertencente ao plano. Mas infelizmente isso também não é robusto o suficiente, pois em imagens com ruído pode ocorrer o mesmo problema anteriormente mencionado e mostrado na Figura 4.6.

Para ser mais robusto mesmo com ruído, além de fazer a verificação da distância do ponto ao plano, verificamos as normais definidas pelos triângulos formados pelos pontos vizinhos do ponto analisado. A formação destes triângulos é mostrada na Figura 4.7, onde cada triângulo define um plano e uma orientação.

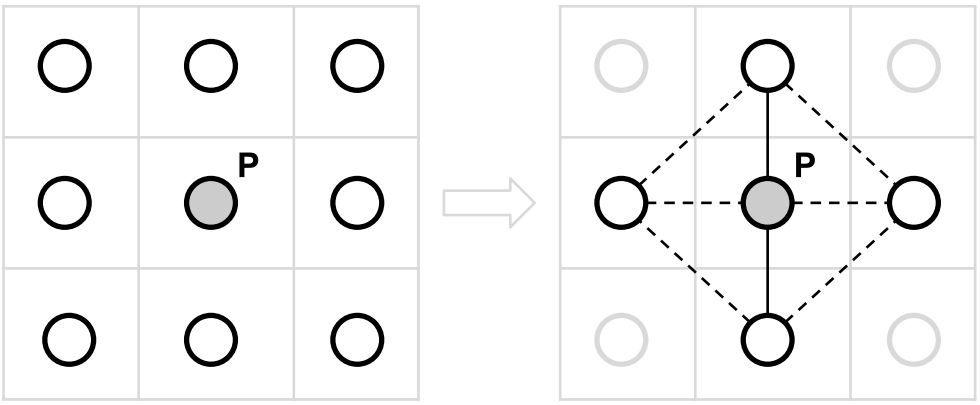



Figura 4.7: Triângulos gerados entre o ponto analisado e seus vizinhos.

Com isso, fazemos uma votação, onde é analisado se a maioria dos triângulos formados pelos pontos vizinhos tem a orientação próxima a da orientação do plano. Caso tenham, o ponto é considerado como pertencente àquele plano. Esta verificação é feita contra cada plano identificado. Este processo é realizado da seguinte forma:

1. Criar os vetores

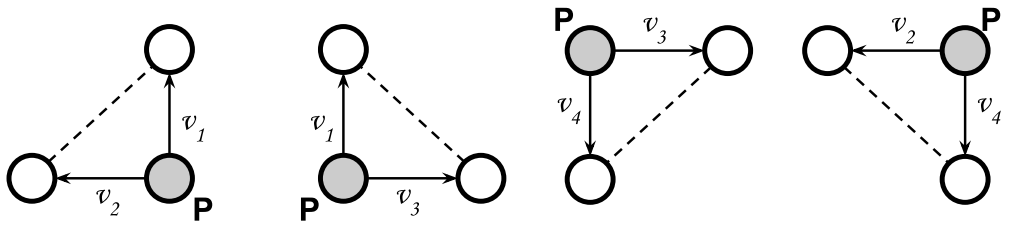



Figura 4.8: Vetores criados a partir das arestas dos triângulos gerados.

2. Normalizar os vetores $v_1, v_2, v_3, v_4$

3. Calcular a normal de cada triângulo
   $\hat{n}_1 = \hat{v}_1 \times \hat{v}_2, \qquad \hat{n}_2 = \hat{v}_1 \times \hat{v}_3, \qquad \hat{n}_3 = \hat{v}_3 \times \hat{v}_4, \qquad \hat{n}_4 = \hat{v}_2 \times \hat{v}_4$

4. Verificar quantos triângulos tem orientação próxima a orientação do $plano_i$
   $acos(\hat{n}_1 \cdot \hat{n}_{plano_i}) \leq$ ângulo, $\qquad acos(\hat{n}_2 \cdot \hat{n}_{plano_i}) \leq$ ângulo,
   $acos(\hat{n}_3 \cdot \hat{n}_{plano_i}) \leq$ ângulo, $\qquad acos(\hat{n}_4 \cdot \hat{n}_{plano_i}) \leq$ ângulo

A Figura 4.9 ilustra o resultado do processo de determinação dos pontos do planos usando as normais.

Todos os pontos classificados como pertecentes a algum plano durante esse processo são pintados nas coordenadas (i,j) da imagem segmentada que é usada como entrada para o método descrito na seção 4.5.

## 4.5 Construção das estruturas geométricas planares

Para gerar os polígonos da estrutura planar, desenvolvemos uma ferramenta que recebe como entrada uma imagem previamente classificada e transforma em dados geométricos as regiões formadas pelos pixels classificados desta imagem. Como saída, temos os dados geométricos que são os polígonos triangulados com textura que representam essas regiões. O método executado por essa ferramenta é usado em dois momentos em contextos diferentes durante a construção das estruturas geométricas planares, como pode ser verificado na Figura 4.10.

Na primeira etapa, este método recebe como entrada somente as imagens geradas pelo processo apresentado na seção 4.4; uma vez que é trivial a triangulação dos nós completos da *quadtree* gerados pelo processo descrito

Figura 4.9: Resultado do método de determinação dos pontos do plano, onde são utilizadas as normais para definir a que plano os pontos pertencem.

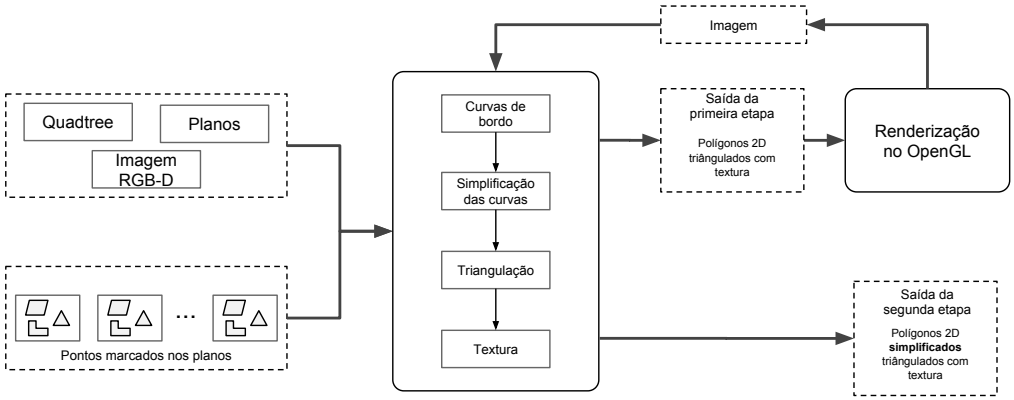



Figura 4.10: Etapas da construção das estruturas geométricas planares. Na primeira etapa os planos detectados são segmentados, triangulados e texturizados; gerando uma imagem que será utilizada pela segunda etapa, onde serão criados os polígonos simplificados, triangulados e com textura.

na seção 4.3, pois basta gerar dois triângulos (12, 13) para cada nó. Os polígonos triangulados de cada plano a partir dos nós completos da *quadtree* e os triangulados na primeira etapa do método são mostrados na Figura 4.11.

Neste trabalho, usamos o OpenGL para realizar a renderização dos

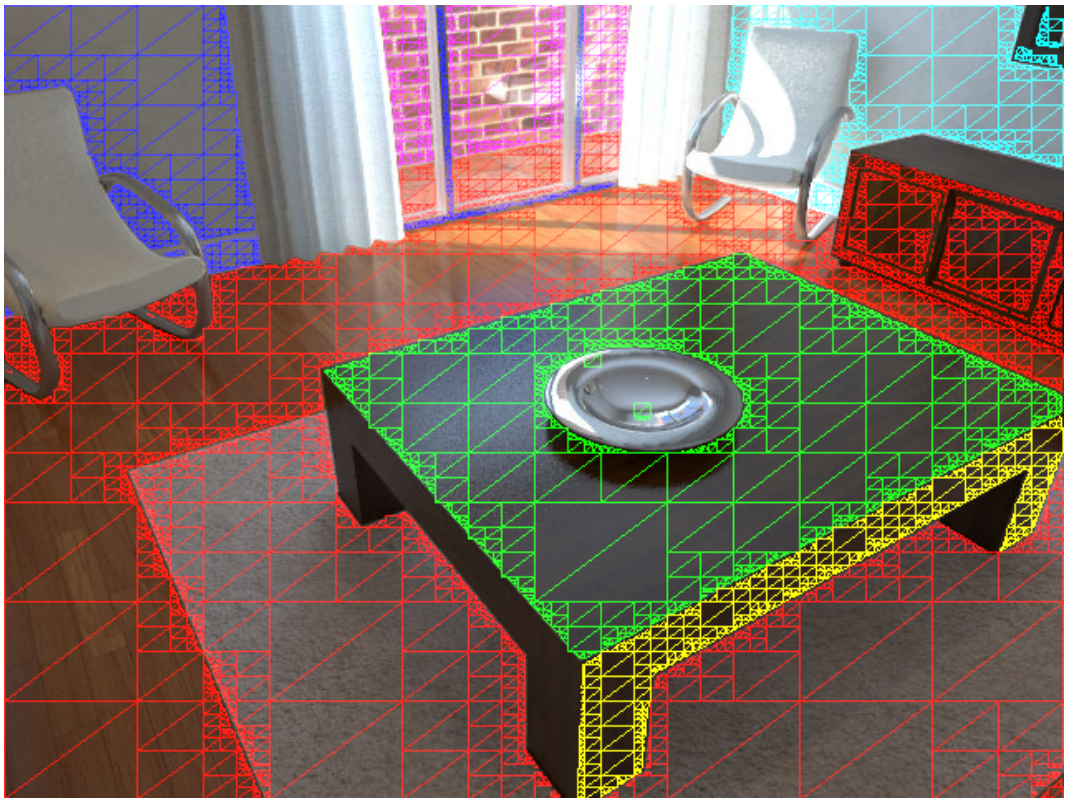


Figura 4.11: Resultado da primeira etapa da construção das estruturas geométricas planares, com os polígonos triangulados a partir dos nós da *quadtree* e dos polígonos formados durante o processo de determinação dos pontos de um plano.

polígonos, pois o OpenGL está presente tanto no *Desktop* como nos dispositivos móveis. A imagem gerada nesta renderização é utilizada como entrada para a segunda etapa. Como todos os pontos desta imagem renderizada pertencem a um plano, nesta segunda etapa não é mais necessário o pré-processamento executado nas seções 4.3 e 4.4. A única condição necessária é que esta imagem seja binária. A saída nesta segunda etapa serão polígonos texturizados com triagulação simplificada, e que podem ser vistas na Figura 4.12. Os detalhes dos algoritmos usados nestas etapas são explicados no capítulo 5.

## 4.6 Sistema de coordenadas e transformações

Para levar os polígonos triangulados do espaço da imagem para o espaço do plano é preciso fazer uma transformação afim. Após essa transformação, os polígonos são desenhados sobre o plano usando uma projeção ortográfica.

Mas para que possamos realizar alguma transformação ou projeção sobre um plano, primeiro é necessário definir o sistema de coordenadas deste plano. Para definir o sistema de coordenadas do plano precisamos calcular os vetores

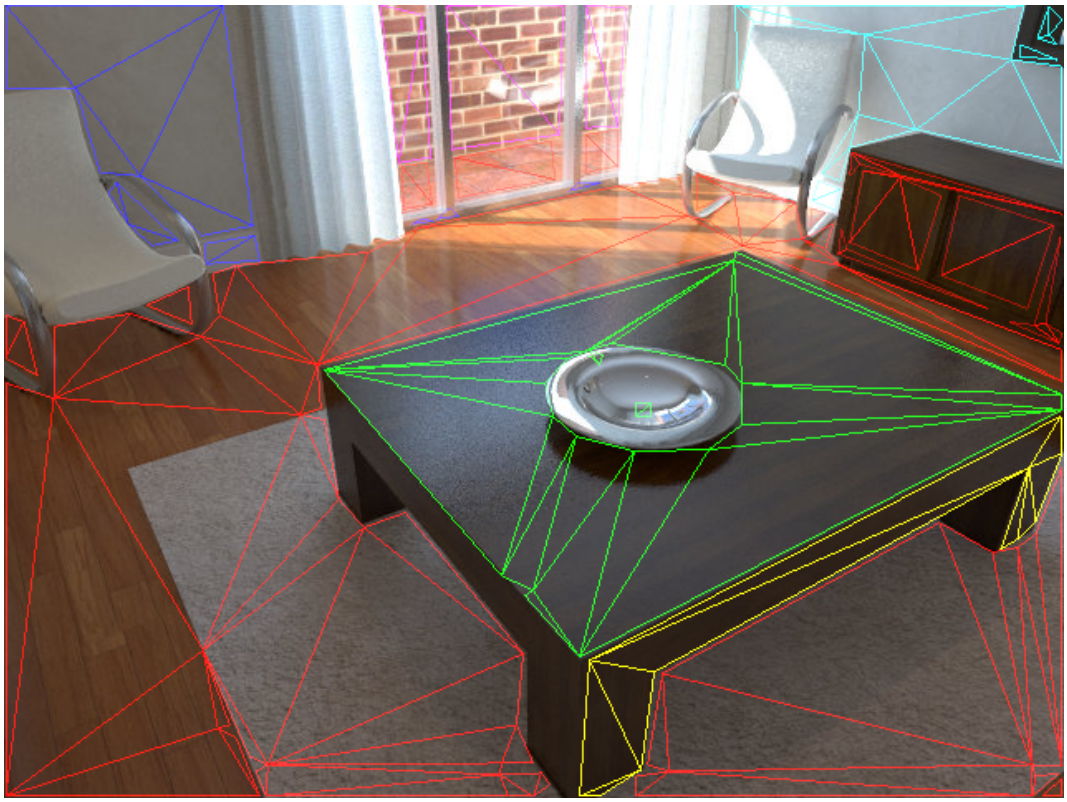


Figura 4.12: Resultado da segunda etapa da construção das estruturas geométricas planares, que teve como entrada os polígonos triangulados na primeira etapa.

deste sistema de referência (24). Para calcular esses vetores usamos o vetor normal do plano ($n$) que já está normalizado e o vetor vertical da câmera ($U$). O cálculo é realizado da seguinte forma:

$$\hat{v} = \frac{U - (U \cdot n)n}{|U - (U \cdot n)n|},$$

$$\hat{u} = n \times \hat{v}.$$

Então, os vetores do sistema de coordenadas são definidos como

$$X = \hat{u}, \;\; Y = \hat{v} \;\text{ e }\; Z = n$$

Com o sistema de coordenadas do plano definido, agora podemos definir

as matrizes de transformação e projeção do plano.

Para construir a matriz que leva os pontos que tem coordenadas locais para o sistema de coordenadas do plano, ou seja, a matriz de transformação do plano, nós usamos a caixa envolvente ($\beta$) que acomoda estes pontos. Esta caixa envolvente é calculada a partir das coordenadas locais dos vértices dos polígonos definidos na seção 5.3. Na construção da matriz de transformação do plano a propriedade desta caixa envolvente que nos interessa é o seu centróide.

Primeiramente determinamos

$$u = X,\ \ v = Y,\ \ n = Z \text{ e } c = \beta_{center}$$

Então, calculamos a translação $t$ com

$$\begin{aligned} t_x &= c \cdot u \\ t_y &= c \cdot v \\ t_z &= c \cdot n \end{aligned}$$

Para enfim definirmos a matriz de transformação do plano

$$M = \begin{bmatrix} u_x & u_y & u_z & -t_x \\ v_x & v_y & v_z & -t_y \\ n_x & n_y & n_z & -t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Construímos a matriz $M$ que projeta os polígonos triangulados e texturizados sobre o plano, também usando as informações da caixa envolvente ($\beta$), mas desta vez são usadas suas dimensões. Primeiro definimos as variáveis $l(left)$, $r(right)$, $b(bottom)$, $t(top)$, $n(near)$ e $t(far)$.

$$\begin{aligned} l = \beta_{xmin} \quad \text{e} \quad r = \beta_{xmax} \\ b = \beta_{ymin} \quad \text{e} \quad t = \beta_{ymax} \\ n = \beta_{zmin} \quad \text{e} \quad f = \beta_{zmax} \end{aligned}$$

Com estas variáveis definidas, construímos a matriz de projeção ortográfica:

$$T = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{-2}{f-n} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

## 4.7 Resultados

Neste trabalho não realizamos uma implementação com o foco na performance. Entretanto, como pode ser visto nas Tabelas 4.1 e 4.2, conseguimos bons resultados em um Notebook DELL XPS L502X que possui um processador Intel(R) Core(TM) i5-2430M CPU @ 2.40GHz com 4GB de RAM e placa gráfica NVIDIA GeForce GT 525M. As etapas indicadas nas Tabelas 4.1 e 4.2, são explicadas na seção 4.5.

Tabela 4.1: Resultados de performance obtidos durante as etapas do processo de construção das estruturas geométricas planares sobre o *dataset* LivingRoom2 (25)

| Frame | Quantidade de planos | Primeira etapa (milisegundos) | Segunda etapa (milisegundos) |
|---|---|---|---|
| 99 | 8 | 5.648 | 2.808 |
| 751 | 7 | 3.137 | 2.341 |
| 1884 | 6 | 6.711 | 2.768 |
| 2068 | 6 | 6.020 | 2.666 |
| 2136 | 8 | 6.134 | 3.000 |

Na Figura 4.13 estão os resultados obtidos da construção das estruturas geométricas planares que teve como entrada as imagens RGB-D do *dataset* sintético LivingRoom2 (25), e na Figura 4.14 estão os resultados sobre o *dataset* CopyRoom (26) que foi capturado com o dispositivo *Asus Xtion Pro Live*. Nestas figuras, além das imagens RGB e profundidade, pode ser vista, a cena capturada sobreposta pelos polígonos triangulados na primeira etapa e os polígonos triangulados e simplificados na segunda etapa do processo de construção das estruturas geométricas planares, também é mostrado que os segmentos de alguns planos desaparecem devido a determinados valores

Tabela 4.2: Resultados de performance obtidos durante as etapas do processo de construção das estruturas geométricas planares sobre o *dataset* CopyRoom (26)

| Frame | Quantidade de planos | Primeira etapa (milisegundos) | Segunda etapa (milisegundos) |
|---|---|---|---|
| 1063 | 11 | 3.873 | 3.142 |
| 1791 | 8 | 3.348 | 2.909 |
| 2297 | 10 | 3.419 | 2.727 |
| 4161 | 4 | 3.476 | 2.823 |

do parâmetro utilizado pelo algoritmo de simplificação da curvas de bordo Douglas-Peucker, que é apresentado na seção 5.2. Isto ilustra o problema de autointersecção gerado pela implementação tradicional do Douglas-Peucker. Alguns dos trabalhos que resolvem este problema estão indicados na seção 5.2.

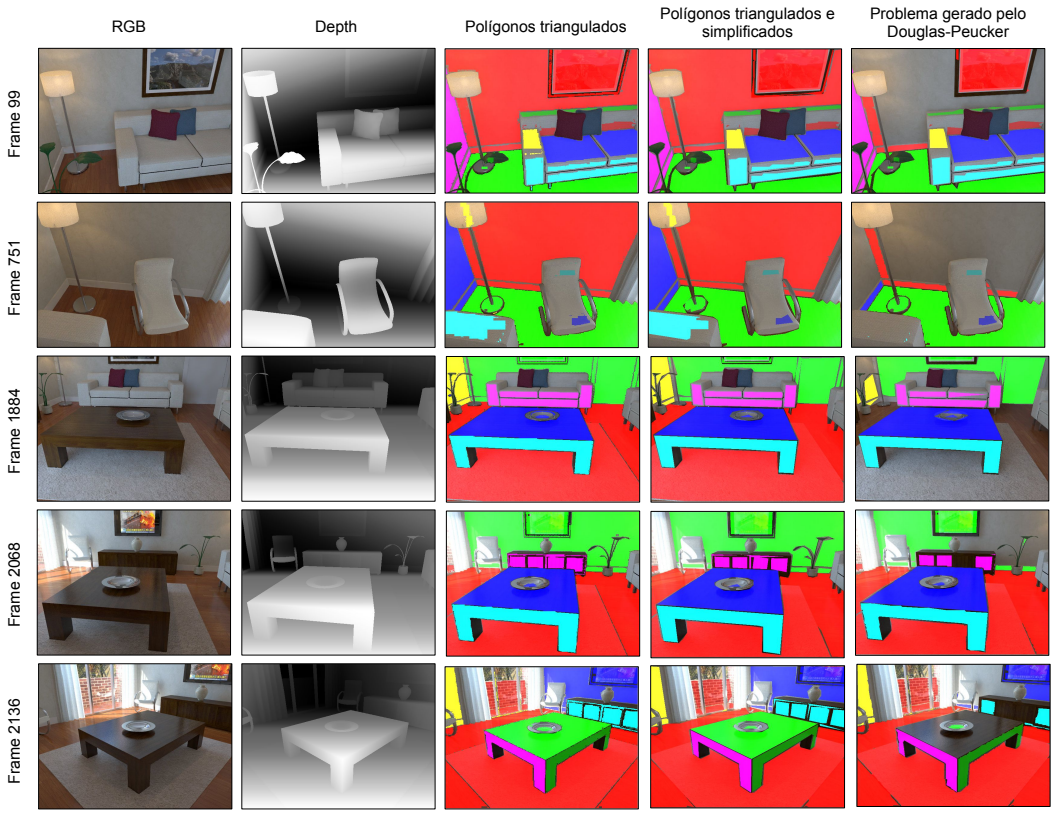



Figura 4.13: Resultados da construção das estruturas geométricas planares sobre o *dataset* LivingRoom2 (25), que são dados sintéticos. Da esquerda para direita temos a imagem RGB, os dados de profundidade, a cena com os polígonos triangulados na primeira etapa, a cena com polígonos triangulados e simplificados da segunda etapa; e a cena com os problemas causados pelo Douglas-Peucker. Os polígonos triangulados possuem uma cor que está associada a um plano, onde não há cor não existe existe um polígono triangulado.

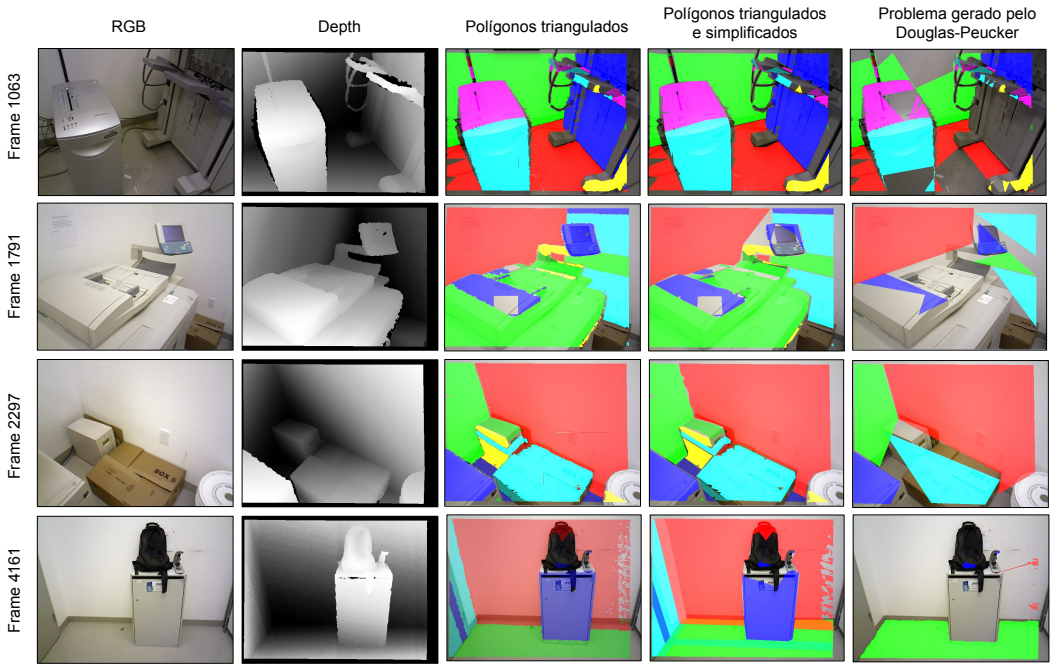



Figura 4.14: Resultados da construção das estruturas geométricas planares sobre o *dataset* CopyRoom (26), obtido através da captura da cena com o dispositivo *Asus Xtion Pro Live*. Da esquerda para direita temos a imagem RGB, os dados de profundidade, a cena com os polígonos triangulados na primeira etapa, a cena com polígonos triangulados e simplificados da segunda etapa; e a cena com os problemas causados pelo Douglas-Peucker. Os polígonos triangulados possuem uma cor que está associada a um plano, onde não há cor não existe existe um polígono triangulado.



## 4.8 Processo incremental da reconstrução da cena

As informações descritas nesta seção não foram implementadas, pois o processo contínuo de detecção dos planos não foi implementado. Este processo é responsável por manter uma lista com o planos mais importantes da cena. Sem esta lista, não é possível realizar a atualização das estruturas planares.

Para realizar a reconstrução de uma cena 3D a partir das imagens de um vídeo RGB-D, é necessário combinar os dados fornecidos por cada *frame* deste vídeo. Contudo, resolver este problema usando somente métodos de geometria computacional, pode acarretar em diversos problemas numéricos, o que tornaria difícil manter a geometria consistente com a topologia.

A maneira que idealizamos para fazer esta reconstrução é através da combinação das estruturas planares obtidas através das imagens do vídeo

RGB-D.

Os planos mais importantes do primeiro *frame* capturado serão detectados pelo processo de detecção dos planos, e nos *frame*s seguintes, muitos desses planos irão continuar, enquanto que outros irão desaparecer e alguns irão aparecer. Então, além da detecção dos planos, também é necessária a monitoração dos planos. Esta monitoração será feita através da atualização de uma lista com os planos mais importantes detectados até o momento. Cada elemento desta lista possuirá a equação do plano e todos os polígonos que estão sobre este plano, juntamente com sua textura.

Esta lista será atualizada em dois momentos: na detecção dos planos e na construção das estruturas geométricas planares. Durante o processo de detecção dos planos, para cada *frame* capturado, na lista serão adicionados os novos planos detectados no *frame* e quais planos continuam em relação ao último *frame* capturado. O processo de construção das estruturas geométricas planares será responsável por atualizar as estruturas planares dos planos que continuam e inicializar as estruturas dos novos planos detectados. O procedimento de inicialização é o mesmo que foi realizado para o primeiro *frame* capturado e detalhado na seção 4.5.

A atualização das estruturas planares é realizada através da junção dos novos polígonos e sua textura aos polígonos e textura existentes do plano. Como a representação das estruturas planares usam como sistema de referência o sistema de coordenadas da primeira câmera, e assumindo que temos os dados *tracking* da câmera, a atualização das estruturas planares é feita da seguinte forma:

- Transformar os novos polígonos para o sistema de coordenadas da primeira câmera.
- Desenhar os polígonos existentes sobre o plano.
- Desenhar os novos polígonos sobre os existentes.
- Executar o passo de simplificação.

Ao desenhar os polígonos novos sobre os existentes, realizamos a combinação entre os polígonos antigos e os novos, contudo esta combinação não é geométrica, mas sim através da fusão das imagens. Após a simplificação, ao final, teremos a geometria atualizada dos planos.

# 5
# Transformando pixels em polígonos

Neste capítulo, apresentaremos os detalhes do método que utilizamos para transformar em dados geométricos as regiões formadas pelos pixels classificados de uma imagem.

Desenvolvemos um método que transforma uma imagem segmentada em regiões, numa representação poligonal com textura. A imagem segmentada recebida pelo método é uma imagem binária comum. Como estamos trabalhando com imagens RGB-D, então temos que segmentar e converter essa imagem RGB-D em uma imagem binária. Isto é feito durante o processo de segmentação da imagem RGB-D em regiões.

A etapa de segmentação da imagem em regiões recebe como entrada uma imagem RGB-D, onde o D é a propriedade utilizada como critério que determina de que região este ponto é pertencente. Esta etapa gera como saída uma imagem binária para cada região planar identificada, conforme mostrado na Figura 5.1.

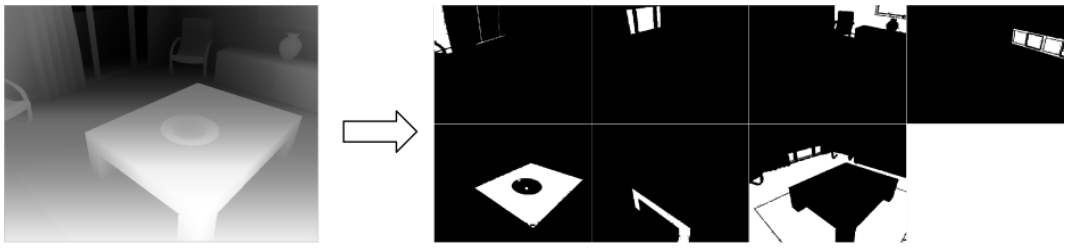


Figura 5.1: Segmentação da imagem de profundidade em regiões, onde estas regiões estão associadas a um plano detectado.

A segmentação da imagem em regiões é realizada em dois momentos. O primeiro é através da classificação dos pontos desta imagem pela transformada de Hough. Em seguida, a segmentação, é complementada pela identificação dos pontos da imagem que a transformada de Hough não conseguiu identificar. Essas etapas são discutidas nas seções 4.3 e 4.4.

Com as regiões da imagem determinadas, o método calcula as curvas de bordo dessas regiões. Em seguida as curvas de bordo são simplificadas. Os polígonos formados pelas curvas de bordo são triangulados e então

são definidas as coordenadas de texturas de acordo com os triângulos. Neste momento, temos como resultado a representação poligonal com textura das regiões segmentadas da imagem. A Figura 5.2 ilustra este processo.

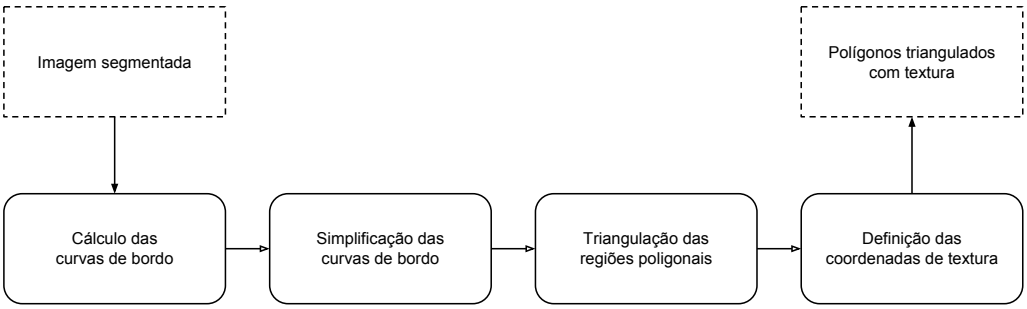



Figura 5.2: *Pipeline* do método que transforma uma imagem segmentada em regiões, em polígonos triangulados com textura.

Dentre os diversos algoritmos apresentados neste capítulo, o algoritmo utilizado na seção 5.1 possui mais detalhes em sua descrição que os demais. Em razão das poucas referências com os detalhes de sua implementação, desenvolvemos nosso próprio conjunto de procedimentos que executam este algoritmo.

Nas seções a seguir são descritos os componentes de nosso método. O resultado de sua execução pode ser visto na Figura 5.3.

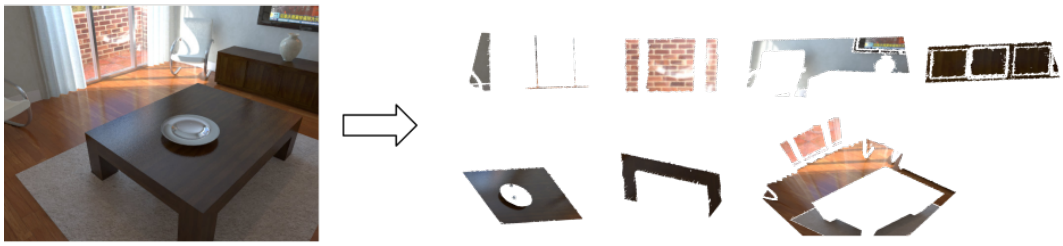


Figura 5.3: Polígonos triangulados e com textura, obtidos a partir da cena capturada.

## 5.1 Identificação das componentes conexas

Várias representações compactas estão disponíveis para determinar as curvas de contorno em uma região de pixels de uma imagem, tais como o Chain Code, Mid Crack e Crack Code (9). Devido aos dados de profundidade das imagens RGB-D possuírem muito ruído, escolhemos como representação o

Crack Code, pois ele é robusto suficiente para determinar as curvas de contorno de um único pixel, além de ser facilmente paralelizável.

Tendo como base a representação do Crack Code, desenvolvemos um algoritmo que calcula as curvas de bordo de uma região e determina se esta curva de bordo é externa ou interna. As curvas de bordo classificadas como externa são os limites de uma componente conexa, já as curvas internas são buracos existentes nesta componente conexa. No pseudo-código do algoritmo 1 há alguns pontos que devem ser destacados, que são a expansão da imagem binária, a variável *estado* e a função *CriarCurvaDeBordo*.

A expansão da imagem binária é necessária para que não seja necessário tratar os pixels de borda. Esta expansão consiste em adicionar na imagem uma linha antes de primeira linha, uma linha depois da última linha, uma coluna antes da primeira coluna e uma coluna depois da última coluna. Este procedimento é ilustrado na Figura 5.4.

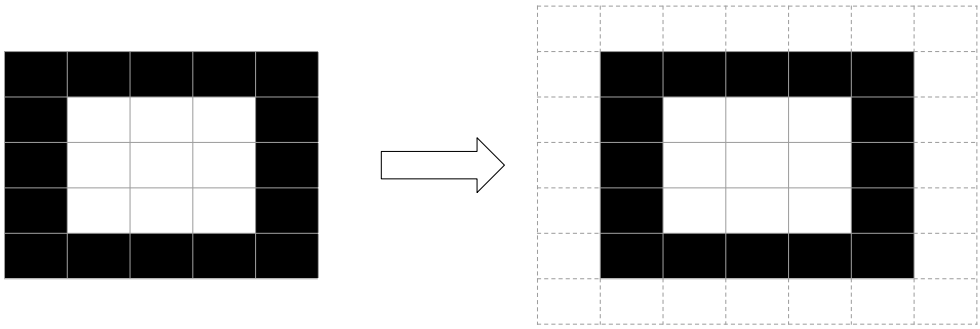


Figura 5.4: Expansão da imagem binária para evitar o tratamento dos pixels de borda. À esquerda está a imagem onde serão identificadas as componentes conexas; à direita mostra esta imagem expandida.

A verificação de cada pixel na imagem é feita usando o método *scanline*. Isto é, a imagem é varrida pixel a pixel e linha por linha. Cada pixel já processado recebe uma marcação que indica se ele pertence a alguma curva de bordo. A variável *estado* é utilizada para evitar que os pixels pertencentes a uma curva de bordo sejam processados novamente, como ser visto no algoritmo 1. O estado OUT significa que o pixel não pertence à região pintada da imagem, então não deve ser processado; o estado IN indica que o pixel pertence ao bordo de uma região pintada e deve ser analisado. O estado ON, por sua vez, indica que o pixel pertence a uma região pintada que pode ser um bordo ou não, contudo já foi processado. Os estados e as possíveis transições são mostrados na Figura 5.5.

A função *CriarCurvaDeBordo*, como o próprio nome diz, é quem cria as curvas de bordo interna e externa começando a partir de um determinado pixel.

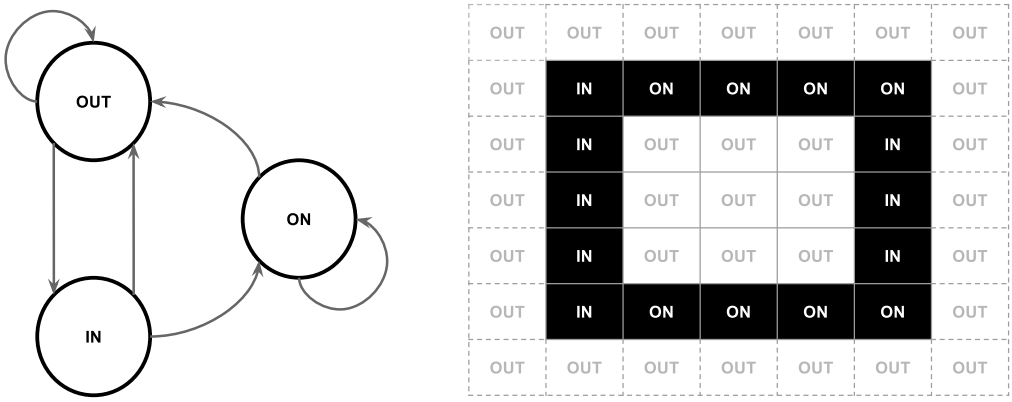



Figura 5.5: Estados e transições possíveis durante a realização da verificação de cada pixel da imagem através do método *scanline*. O estado OUT significa que o pixel não pertence à região pintada da imagem, então não deve ser processado; o estado IN indica que o pixel pertence ao bordo de uma região pintada e deve ser analisado; o estado ON indica que o pixel pertence a uma região pintada que pode ser um bordo ou não, contudo já foi processado.

É nela que os pixels são marcados como pertecentes a uma curva de bordo. O pseudo-código da função *CriarCurvaDeBordo* pode ser visto no algoritmo 2. Na Figura 5.6, há uma visualização do processo de criação de uma curva de bordo externa. Na Figura 5.7 é mostrado o processo de criação de uma curva de bordo interna. Na Figura 5.8 pode ser visto o resultado obtido após a aplicação do processo de identificação das componentes conexas.

## 5.2 Simplificação das linhas poligonais

Com as componentes conexas determinadas é possível gerar polígonos que serão usados como segmentação de regiões dos planos identificados. Contudo, estas componentes conexas possuem muito mais pontos do que o necessário para os polígonos que serão gerados. Sendo assim, usamos o algoritmo Douglas-Peucker (30) que tem como função a simplificação de linhas poligonais. A Figura 5.9 exibe os resultados obtidos após o processo de simplificação das linhas poligonais.

Muitos algoritmos de triangulação necessitam que os polígonos de entrada sejam simples, isto é, sem autointersecção. Infelizmente, o algoritmo Douglas-Peucker apresenta este problema. Ou seja, ele simplifica as linhas poligonais mas não garante que o resultado seja um polígono sem autointersecção.

Há diversos trabalhos que resolvem este problema, como em (28), além

```
   entrada: imagem binária
   saída    : curvas de bordo
 1 estado ← OUT;
 2 Expande(imagem binária);
 3 foreach pixel_{i,j} ∈ imagem binária do
 4     if pixel_{i,j} = Cor da frente then
 5         switch estado do
 6             case OUT
 7                 estado ← IN;
 8                 if condições para criar curva de bordo then
 9                     CriarCurvaDeBordo(Externa, pixel_{i,j});
10                 end
11             case IN
12                 estado ← ON;
13                 continue;
14             case ON
15                 continue;
16             end
17         endsw
18     else if pixel_{i,j} = Cor do fundo then
19         switch estado do
20             case IN
21                 estado ← OUT;
22                 if condições para criar curva de bordo then
23                     CriarCurvaDeBordo(Interna, pixel_{i,j});
24                 end
25             case ON
26                 estado ← OUT;
27                 if condições para criar curva de bordo then
28                     CriarCurvaDeBordo(Interna, pixel_{i,j});
29                 end
30             case OUT
31                 continue;
32             end
33         endsw
34     end
35 end
```

**Algorithm 1:** Cálculo das curvas de bordo

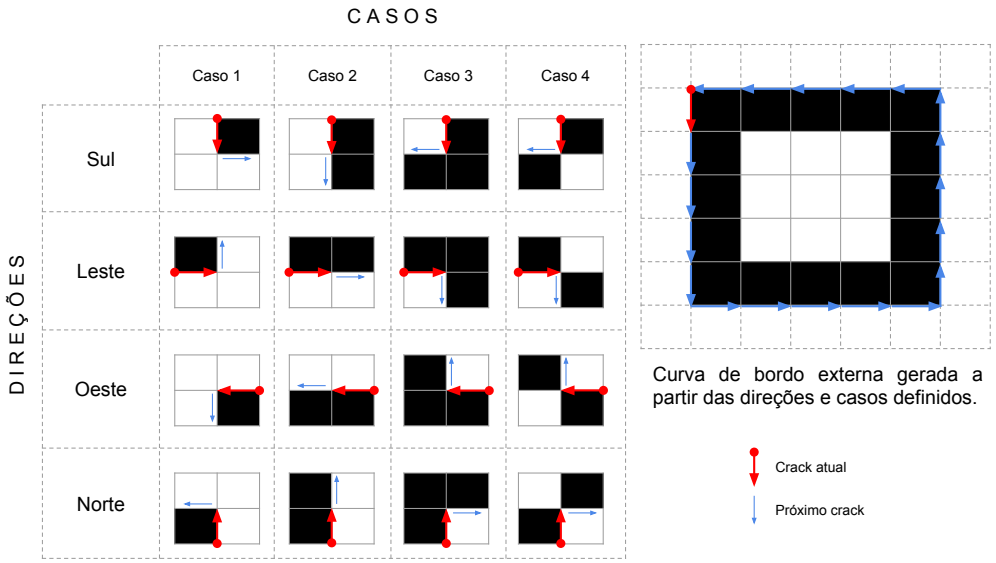



Figura 5.6: Processo de criação de uma curva de bordo externa e a tabela com os possíveis direções que um crack pode seguir.



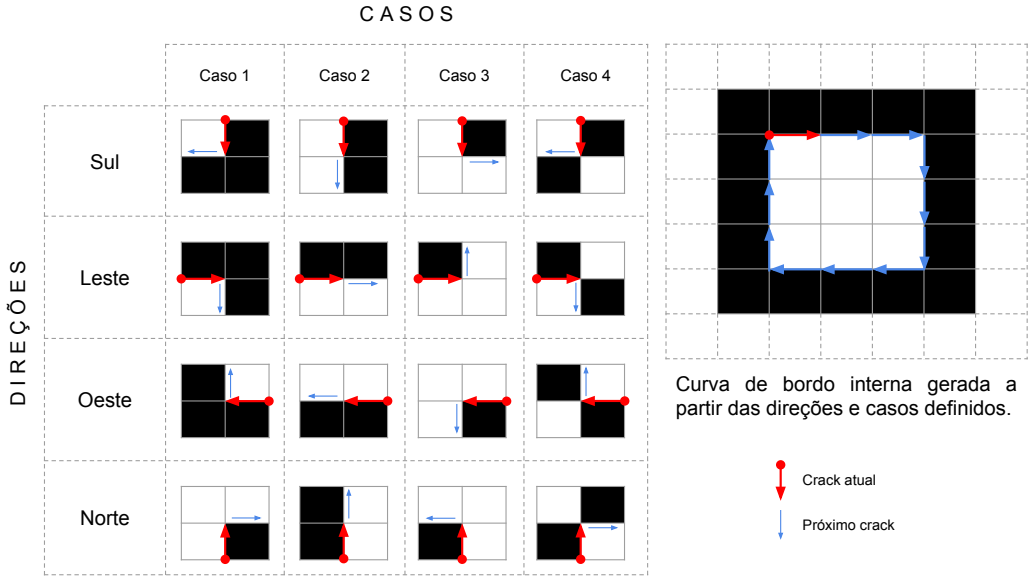



Figura 5.7: Processo de criação de uma curva de bordo interna e a tabela com os possíveis direções que um crack pode seguir.

de outros algoritmos de simplificação de linhas poligonais, como (29). Contudo, apesar deste problema, decidimos usar o Douglas-Peucker em nosso *pipeline* devido a simplicidade de sua implementação. Porém, no futuro iremos substituir a implementação tradicional do Douglas-Peucker por uma variante que trate o problema de autointersecção ou por um outro algoritmo de simplificação de linhas poligonais que não gere esse problema.

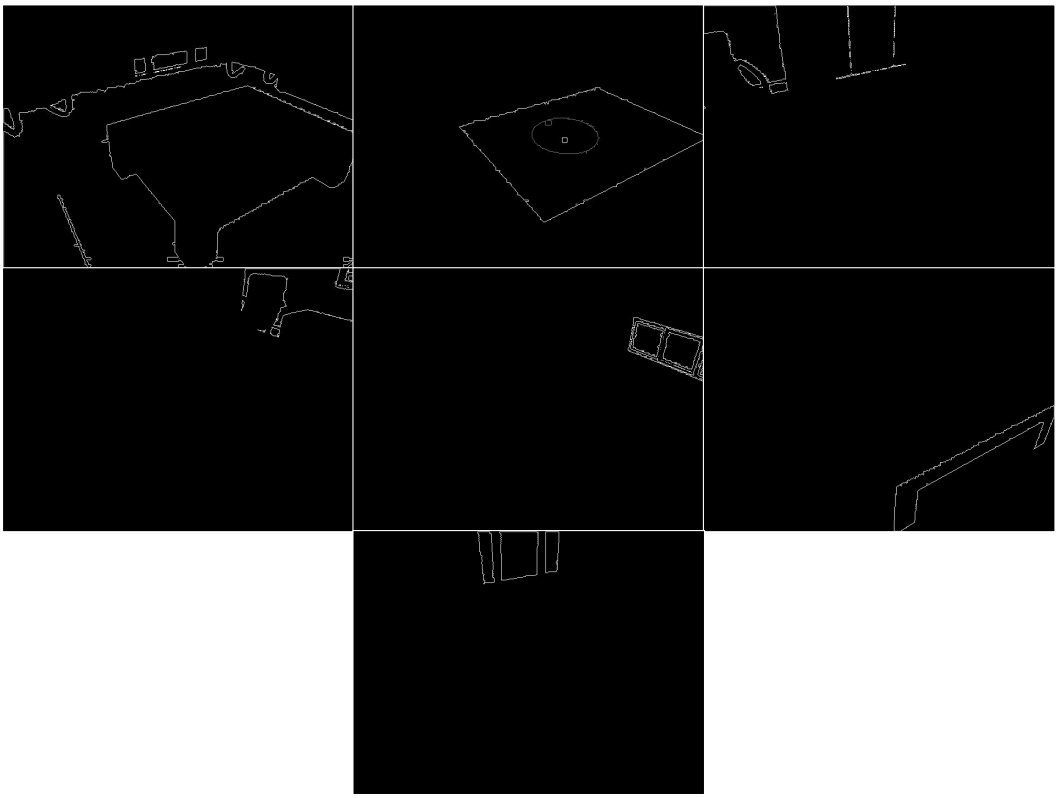
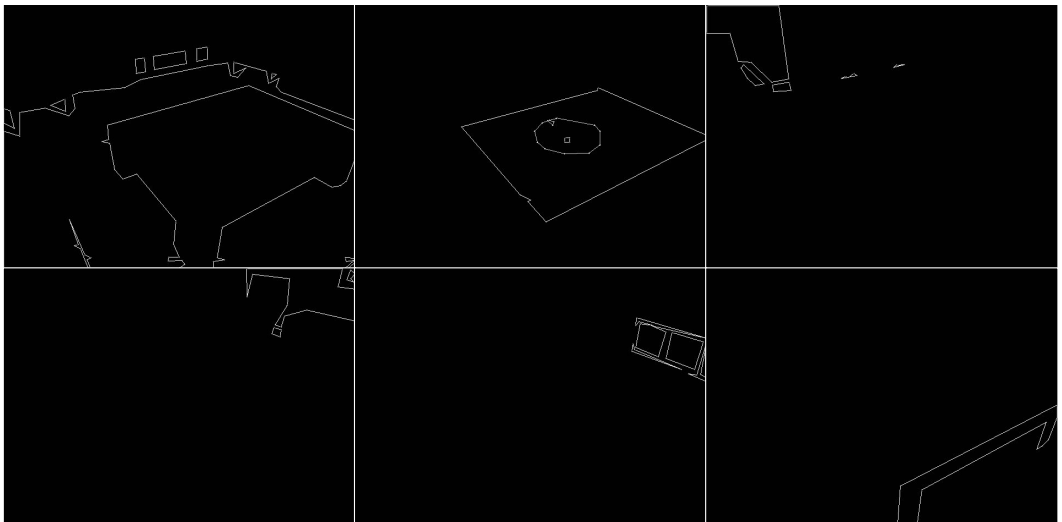


Figura 5.8: Resultados obtidos após a aplicação do processo de identificação das componentes conexas.

```
1 Function CriarCurvaDeBordo (tipo, pixel_inicial)
      saída: curva de bordo
2     repeat
3         if tipo = Externa then
4             Consultar tabela de casos de bordo externo e definir
              direção do novo crack
5         else
6             Consultar tabela de casos de bordo interno e definir
              direção do novo crack
7         end
8         crack_atual = novo crack;
9     until crack_atual estar no pixel_inicial e número de cracks > 4;
```

**Algorithm 2:** Criação de uma curva de bordo



## 5.3
## Triangulação dos polígonos

Com os polígonos tendo suas linhas simplificadas, então realizamos sua triangulação. Para realizar a triangulação dos polígonos usamos a biblioteca de triangulação *Triangle* (23). Esta biblioteca foi desenvolvida para a geração de malhas bidimensionais. É extramente robusta e leve, além de possuir um grande número de opções para a geração dos triângulos. Devido ao fato das imagens RGB-D possuirem muito ruído e como nosso algoritmo de Crack Code indentifica os buracos nas componentes conexas, uma das características que nos auxiliou bastante é de tratamento de buracos nos polígonos. A Figura 5.10 ilustra os resultados da triangulação dos polígonos simplificados.

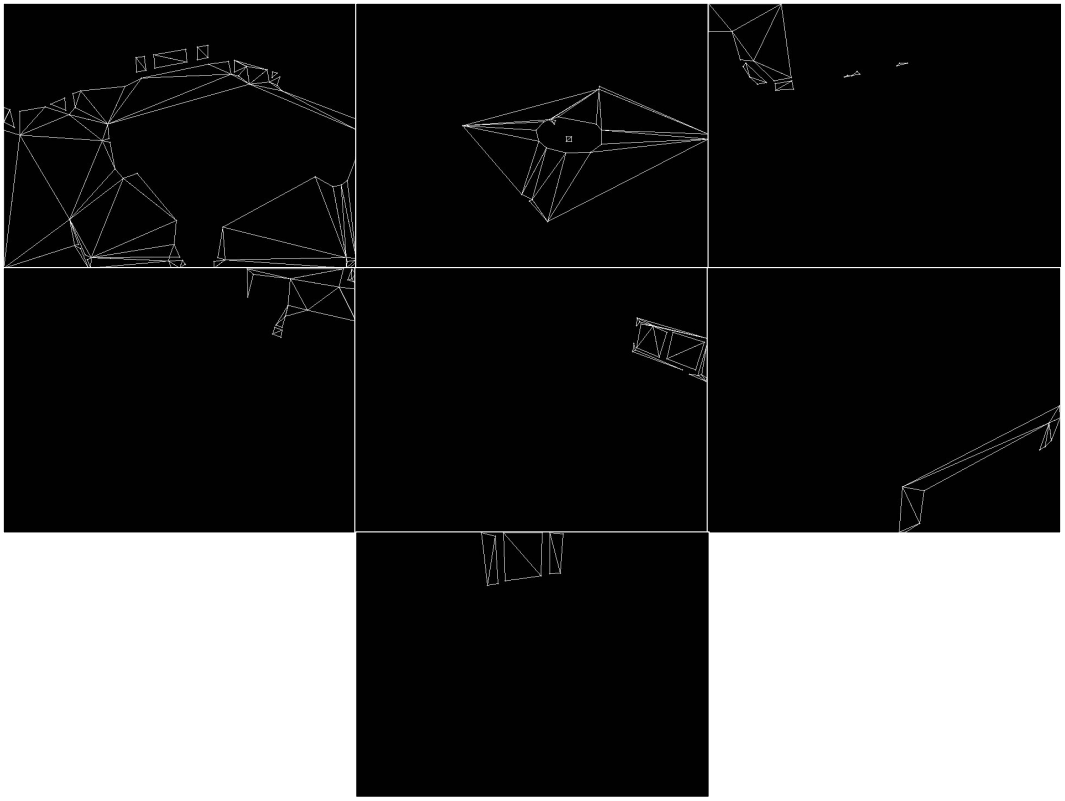


Figura 5.10: Resultados obtidos após a triangulação dos polígonos simplificados.

## 5.4
## Geração das coordenadas de textura

Utilizando os vértices dos triângulos gerados durante a triangulação e simplificação dos polígonos, são geradas as coordenadas de textura. A Figura 5.11 mostra todas as texturas geradas a partir da projeção ortográfica dos

polígonos sobre o plano. Já a Figura 5.12, ilustra a projeção perspectiva dos polígonos com textura.

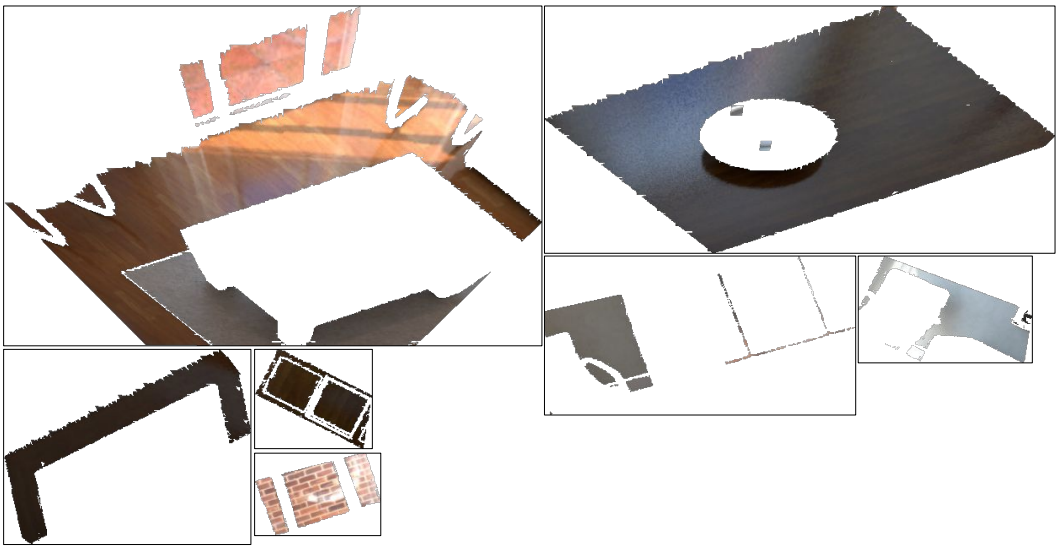


Figura 5.11: Mapa com todas as texturas associadas aos planos e extraídas da cena capturada.

Figura 5.12: Projeção perspectiva dos polígonos triangulados e texturizados por nosso método.

# 6
# Aplicações em realidade aumentada

Neste capítulo, apresentaremos algumas categorias de aplicações de realidade aumentada, assim como a aplicação que implementamos para demonstrar os conceitos e técnicas propostos neste trabalho.

Grande parte das aplicações de realidade aumentada utilizam superfícies planares como sua principal componente para projeção de objetos virtuais (2). Tais aplicações costumam utilizar marcadores para identificar estas superfícies (21).

Com as estruturas geométricas planares detectadas e utilizando os recursos fornecidos pelos dispositivos RGB-D de baixo custo, o desenvolvimento de apliçacões de realidade aumentada sem a necessidade de marcadores é bastante beneficiado, facilitando, assim, a criação de aplicações mais simples como as de pintar paredes (22) e também aplicações mais complexas, conforme as apresentadas na seção 6.1.

Para validar os métodos e técnicas propostos neste trabalho foi desenvolvida a aplicação Arte Sobre Planos apresentada na seção 6.2.

## 6.1
## Tipos de aplicações

As aplicações de realidade aumentada enriquecem o ambiente real com determinada quantidade de informações sintéticas, preferencialmente, apenas o suficiente para superar as limitações do mundo real para uma deteminada aplicação. Azuma *et al.* (44) afirmam que, além da combinação das imagens reais e virtuais, as aplicações de realidade aumentada devem ser executadas em tempo real e os objetos virtuais têm de ser alinhados com as estruturas do mundo real.

As informações passadas através de um objeto virtual, em uma aplicação de realidade aumentada, podem auxiliar o usuário durante a realização das tarefas diárias, como por exemplo:

– Na construção civil, um engenheiro marca a parte elétrica ou hidráulica dos cômodos. Depois o responsável por realizar a obra, visualiza onde exatamente deve ser passada a fiação ou encanamento.

– Na restauração de prédios históricos, um engenheiro informa onde devem ficar os andaimes; um arquiteto determina como ficará a iluminação e onde ficarão os móveis etc.
– Na criação de efeitos visuais durante a gravação de um vídeo. Onde a textura seria modificada para gerar buracos, amassar paredes e enfim, deformar superfícies como feito por Felinto *et al.* (11).

Quando consideramos ambientes internos (*indoor*), há aplicações que utilizam estes ambientes como cenários, onde os objetos virtuais serão adicionados mas não serão persistidos (Figura 6.1). Existem outras aplicações onde a captura da cena deve ser persistida (Figura 6.2); Também são encontradas aplicações que armazenam em algum servidor o modelo do lugar onde o usuário está, e através dos dados de geolocalização este modelo é obtido, para que seja comparado com o modelo que está sendo capturado.

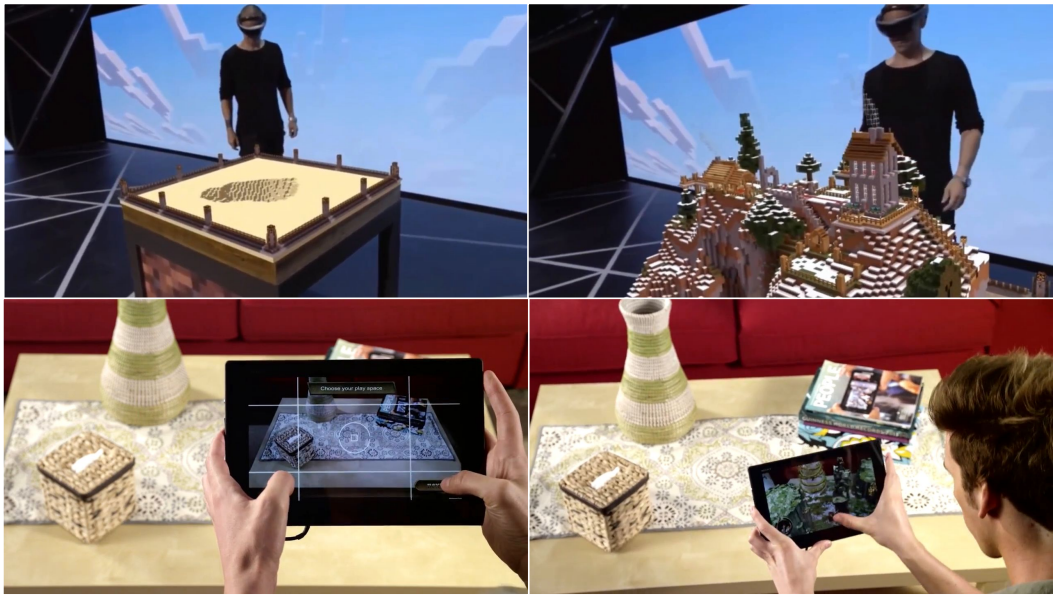


Figura 6.1: Exemplos de aplicações de realidade aumentada que não possuem a necessidade de persitência do modelo para posterior utilização. As imagens da parte superior são do jogo Minecraft (51) sendo manipulado através do dispositivo Hololens (52). Já as imagens da parte inferior são da aplicação Smart Terrain (21) que utiliza o sistema Vuforia (21).

Estes tipos de aplicação associados com áreas como jogos de computador, engenharia, arte digital, arquitetura etc; são beneficiados ao terem as estruturas geométricas planares para realizar a inserção dos objetos virtuais no mundo real, pois elas, normalmente, são o suporte necessário para a projeção destes objetos.

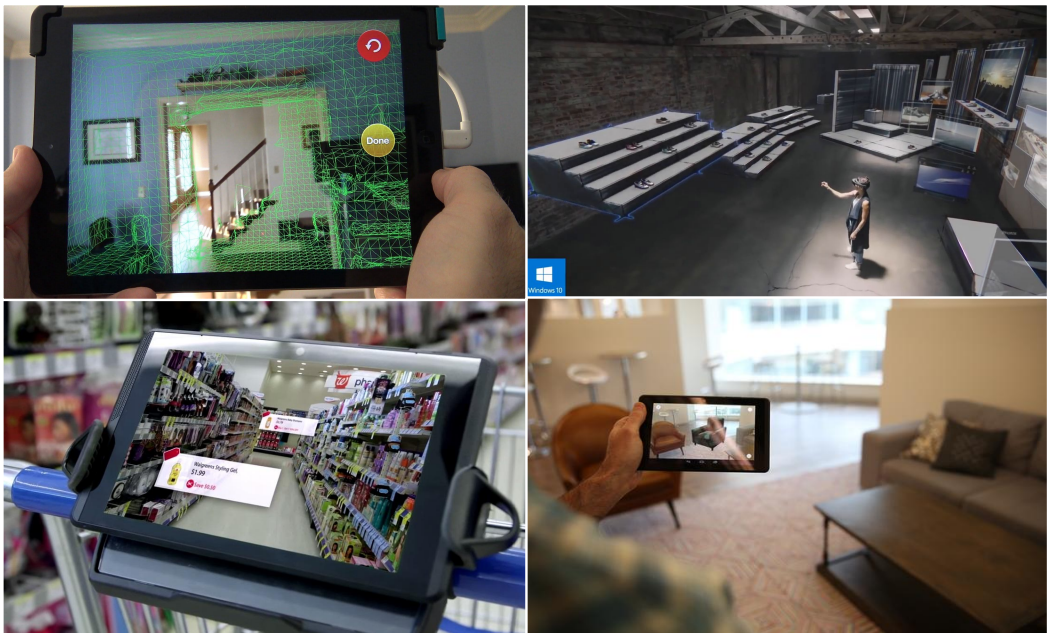



Figura 6.2: Exemplos de aplicações de realidade aumentada que podem fazer uso dos dados capturados. Da esquerda para direita e de cima para baixo, estão as aplicações de captura do ambiente usando o dispositivo Structure Sensor, de decoração de ambientes usando o dispositivo Hololens, de navegação em um supermercado e de decoração, ambas usando o dispositivo Google Tango.

## 6.2 Demonstração de conceito

A fim de demonstrar os métodos e técnicas propostos neste trabalho, desenvolvemos a aplicação chamada Arte Sobre Planos. Esta aplicação permite que o usuário modifique as texturas dos planos identificados e segmentados.

Primeiramente o usuário faz o escaneamento do ambiente para a criação das estruturas geométricas planares. Em seguida ele seleciona um dos planos e, usando como guia a textura associada ao plano selecionado, ele pode pintar, escrever um texto ou timbrar algumas figuras geométricas disponíveis sobre o plano escolhido. Após modificar o plano, o usuário pode voltar a navegar pelo ambiente e irá ver o plano com sua arte aplicada. Esta sequência de ações está ilustrada na Figura 6.3.

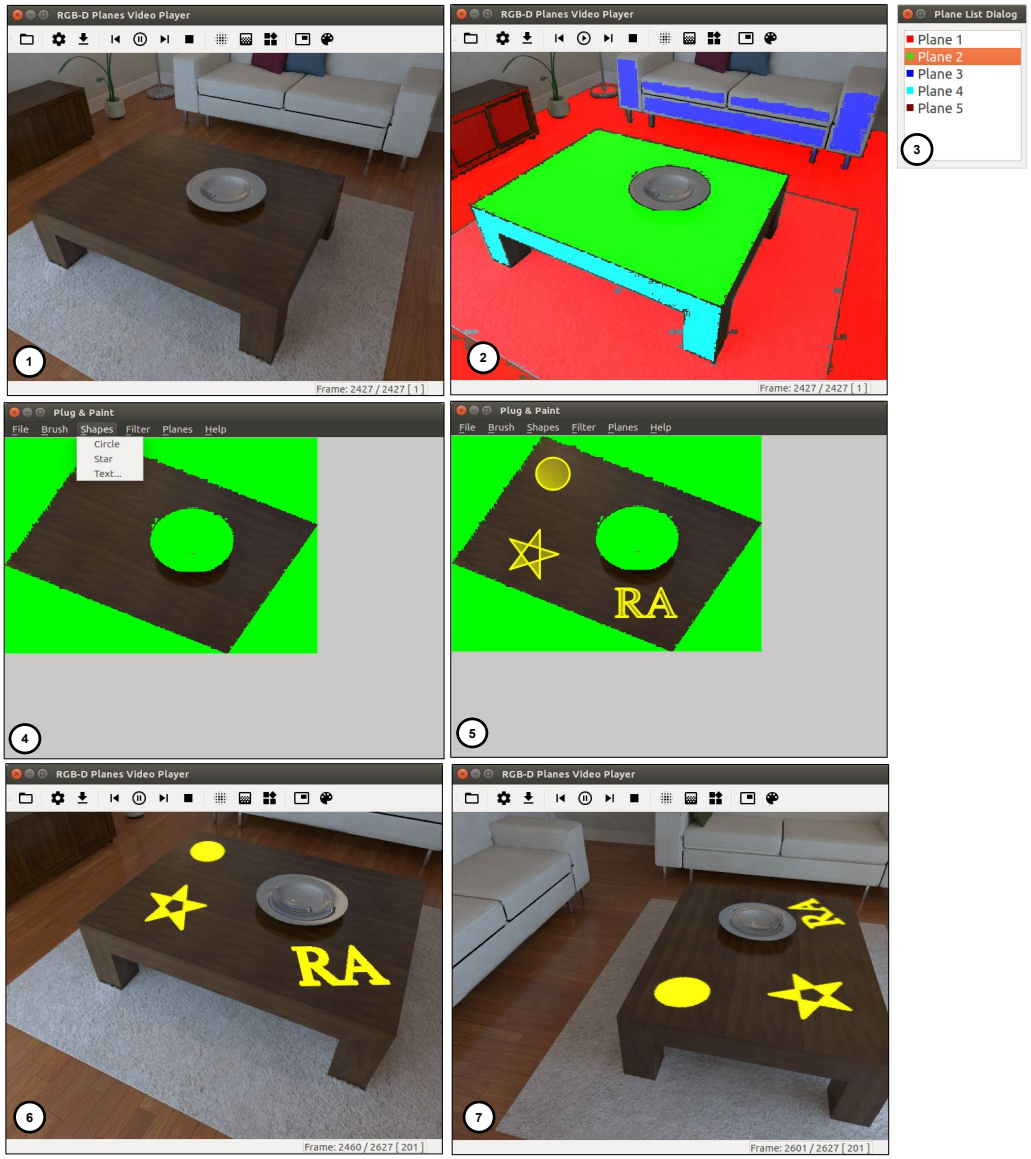



Figura 6.3: Modificando a textura de uma cena capturada. Em (1) está a cena que será modificada; em (2) estão as regiões planares identificadas; em (3) um plano é selecionado; em (4) a textura do plano escolhido é carregada para ser usada como guia; em (5) é realizada a edição; em (6) e (7) é visualizada a textura modificada.

Nossa representação das estruturas geométricas planares facilita a alteração das texturas para este tipo de aplicação, pois a textura é apresentada sem as distorções de perspectiva. Sem essa característica, o usuário teria que realizar a pintura na perspectiva correta em cada *frame* do vídeo. Outro fator a ser considerado é a quantidade de espaço necessária para guardar o modelo reconstruído. Como o modelo reconstruído a partir da nossa representação é compacto, então será necessário pouco espaço para seu armazenamento.

# 7
# Conclusão e trabalhos futuros

Neste trabalho, apresentamos um método para identificar, segmentar e representar estruturas geométricas planares em imagens RGB-D. Nossa representação das estruturas geométricas planares são polígonos bidimensionais triangulados, simplificados e texturizados, que estão no sistema de coordenadas do plano, onde os pontos destes polígonos definem as regiões deste plano.

Para determinar as estruturas planares em uma imagem RGB-D de uma cena 3D capturada, primeiramente, detectamos os planos existentes nesta cena. Nossa abordagem utiliza uma variante da transformada de Hough para a detecção destes planos. Complementamos a detecção através da classificação dos pontos que não puderam ser definidos como pertecentes a alguma região planar durante o processo de detecção.

Após a detecção dos planos e da identificação dos pontos temos a imagem segmentada em regiões. Para representarmos as estruturas planares associadas a cada uma destas regiões, desenvolvemos um método que transforma em dados geométricos as áreas (regiões) formadas pelos pixels classificados de uma imagem.

Durante a transformação realizamos as operações de cálculo e simplificação das curvas de bordo destas áreas; é feita a triangulação dos polígonos delimitados por estas curvas; os polígonos triangulados são simplificados e texturizados.

No final temos as estruturas geométricas planares representadas pelos polígonos bidimensionais no espaço do plano.

Diferente do tratamento realizado no trabalho de Nguyen *et al.* (2), que utiliza somente métodos de geometria computacional para a criação das estruturas geométricas planares, nossa construção destas estruturas é realizada, basicamente, através de métodos de processamento de imagem. Com isso evitamos diversos problemas numéricos, tornando, assim, a geometria consistente com a topologia.

Nosso trabalho traz alguns benefícios para as aplicações de realidade aumentada, pois além de grande parte destas aplicações utilizarem as estruturas planares como o principal suporte de projeção dos objetos virtuais que serão inseridos ao ambiente real, elas também efetuam constantes mudanças nas

texturas existentes da cena capturada. Nossa representação das estruturas geométricas planares facilita a alteração destas texturas, uma vez que a textura é apresentada sem as distorções de perspectiva.

Outro fator a ser considerado é a quantidade de espaço necessária para guardar o modelo reconstruído. Como o modelo reconstruído a partir da nossa representação é compacto, então será necessário pouco espaço para seu armazenamento.

Apesar de termos obtido bons resultados, algumas melhorias devem ser realizadas. Uma otimização a ser feita é a reimplementação do algoritmo de cálculo de curvas de bordo, já que este algoritmo é facilmente paralelizável.

Com relação a robustez, a implementação tradicional do Douglas-Peucker deve ser substituída por uma de suas variantes, a fim de evitar os problemas autointersecção.

Para realizar a reconstrução total da cena é necessária a implementação da detecção contínua dos planos e a atualização incremental da representação das estruturas geométricos planares.

A fim de demonstrar as facilidades que nosso método pode oferecer para as aplicações de realidade aumentada móvel, a implementação dos algoritmos que utilizamos deve ser portada para este novo ambiente, com principal foco nos dispositivos móveis Google Tango (17) e Structure Sensor (18).

# Referências Bibliográficas


[1] LIMBERGER, F. A.; OLIVEIRA, M. M.. **Real-time detection of planar regions in unorganized point clouds**. Pattern Recognition, 48(6):2043–2053, 2015.

[2] NGUYEN, T.; REITMAYR, G. ; SCHMALSTIEG, D.. **Structural modeling from depth images**. Visualization and Computer Graphics, IEEE Transactions on, 21(11):1230–1240, 2015.

[3] FERNANDES, L. A.; OLIVEIRA, M. M.. **Real-time line detection through an improved hough transform voting scheme**. Pattern Recognition, 41(1):299–314, 2008.

[4] HULIK, R.; SPANEL, M.; SMRZ, P. ; MATERNA, Z.. **Continuous plane detection in point-cloud data based on 3d hough transform**. Journal of visual communication and image representation, 25(1):86–97, 2014.

[5] BORRMANN, D.; ELSEBERG, J.; LINGEMANN, K. ; NÜCHTER, A.. **The 3d hough transform for plane detection in point clouds: A review and a new accumulator design**. 3D Research, 2(2):1–13, 2011.

[6] ERDOGAN, C.; PALURI, M. ; DELLAERT, F.. **Planar segmentation of rgbd images using fast linear fitting and markov chain monte carlo**. In: COMPUTER AND ROBOT VISION (CRV), 2012 NINTH CONFERENCE ON, p. 32–39. IEEE, 2012.

[7] TANG, T. J. J.; LUI, W. L. D. ; LI, W. H.. **A lightweight approach to 6-dof plane-based egomotion estimation using inverse depth**. In: AUSTRALASIAN CONFERENCE ON ROBOTICS AND AUTOMATION, 2011.

[8] WU, A. Y.; BHASKAR, S. ; ROSENFELD, A.. **Parallel processing of region boundaries**. Pattern Recognition, 22(2):165–172, 1989.

[9] SHIH, F. Y.. **Image processing and pattern recognition: fundamentals and techniques**. John Wiley & Sons, 2010.

[10] SHIRLEY, P.; ASHIKHMIN, M. ; MARSCHNER, S.. **Fundamentals of computer graphics**. CRC Press, 2009.

[11] FELINTO, D. Q.; ZANG, A. R. ; VELHO, L.. **Production framework for full panoramic scenes with photorealistic augmented reality**. CLEI Electronic Journal, 16(3):8–8, 2013.

[12] DE BERG, M.; VAN KREVELD, M.; OVERMARS, M. ; SCHWARZKOPF, O. C.. **Computational geometry**. Springer, 2000.

[13] SKIENA, S. S.. **The algorithm design manual: Text**, volumen 1. Springer Science & Business Media, 1998.

[14] OLSSON, T.; KÄRKKÄINEN, T.; LAGERSTAM, E. ; VENTÄ-OLKKONEN, L.. **User evaluation of mobile augmented reality scenarios**. Journal of Ambient Intelligence and Smart Environments, 4(1):29–47, 2012.

[15] ARTH, C.; GRASSET, R.; GRUBER, L.; LANGLOTZ, T.; MULLONI, A. ; WAGNER, D.. **The history of mobile augmented reality**. arXiv preprint arXiv:1505.01319, 2015.

[16] JAIN, P.; MANWEILER, J. ; ROY CHOUDHURY, R.. **Overlay: Practical mobile augmented reality**. In: PROCEEDINGS OF THE 13TH ANNUAL INTERNATIONAL CONFERENCE ON MOBILE SYSTEMS, APPLICATIONS, AND SERVICES, p. 331–344. ACM, 2015.

[17] GOOGLE, G. T.. **Google Tango** , 2016. Acesso em: Julho de 2016.

[18] OCCIPITAL, O. S. S.. **Structure Sensor**, 2016. Acesso em: Janeiro de 2016.

[19] INTEL, I. R. S.. **Intel Real Sense**, 2016. Acesso em: Janeiro de 2016.

[20] ASUS, A. X. P. L.. **Asus Xtion Pro Live**, 2016. Acesso em: Janeiro de 2016.

[21] QUALCOMM, Q. V.. **Qualcomm Vuforia**, 2015. Acesso em: Novembro de 2015.

[22] CORAL, C. V.. **Coral Visualizer**, 2015. Acesso em: Outubro de 2015.

[23] SHEWCHUK, J. R.. **Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator**. In: Lin, M. C.; Manocha, D., editors, APPLIED COMPUTATIONAL GEOMETRY: TOWARDS GEOMETRIC ENGINEERING, volumen 1148 de **Lecture Notes in Computer Science**,

p. 203–222. Springer-Verlag, May 1996. From the First ACM Workshop on Applied Computational Geometry.

[24] GOMES, J.; VELHO, L.. **Sistemas gráficos 3d**. IMPA, Rio de Janeiro, 2001.

[25] CHOI, S.; ZHOU, Q.-Y. ; KOLTUN, V.. **Robust reconstruction of indoor scenes**. In: COMPUTER VISION AND PATTERN RECOGNITION (CVPR), 2015 IEEE CONFERENCE ON, p. 5556–5565. IEEE, 2015.

[26] ZHOU, Q.-Y.; KOLTUN, V.. **Dense scene reconstruction with points of interest**. ACM Transactions on Graphics (TOG), 32(4):112, 2013.

[27] LIMA, E. L.. **Geometria analítica e álgebra linear**. IMPA, 2015.

[28] WU, S.-T.; MARQUEZ, M. R. G.. **A non-self-intersection douglas-peucker algorithm**. In: COMPUTER GRAPHICS AND IMAGE PROCESSING, 2003. SIBGRAPI 2003. XVI BRAZILIAN SYMPOSIUM ON, p. 60–66. IEEE, 2003.

[29] VISVALINGAM, M.; WILLIAMSON, P. J.. **Simplification and generalization of large scale data for roads: a comparison of two filtering algorithms**. Cartography and Geographic Information Systems, 22(4):264–275, 1995.

[30] DOUGLAS, D. H.; PEUCKER, T. K.. **Algorithms for the reduction of the number of points required to represent a digitized line or its caricature**. Cartographica: The International Journal for Geographic Information and Geovisualization, 10(2):112–122, 1973.

[31] CARVALHO, P. C.; VELHO, L.; SÁ, A.; MEDEIROS, E.; MONTENGRO, A. A.; PEIXOTO, A. ; ESCRIBA, L. A. R.. **Fotografia 3d–25 colóquio brasileiro de matemática**, 2005.

[32] SCHNABEL, R.; WAHL, R. ; KLEIN, R.. **Efficient ransac for point-cloud shape detection**. In: COMPUTER GRAPHICS FORUM, volumen 26, p. 214–226. Wiley Online Library, 2007.

[33] ZHOU, F.; DUH, H. B.-L. ; BILLINGHURST, M.. **Trends in augmented reality tracking, interaction and display: A review of ten years of ismar**. In: PROCEEDINGS OF THE 7TH IEEE/ACM INTERNATIONAL SYMPOSIUM ON MIXED AND AUGMENTED REALITY, p. 193–202. IEEE Computer Society, 2008.

[34] LEE, B.; CHUN, J.. **Interactive manipulation of augmented objects in marker-less ar using vision-based hand interaction**. In: INFORMATION TECHNOLOGY: NEW GENERATIONS (ITNG), 2010 SEVENTH INTERNATIONAL CONFERENCE ON, p. 398–403. IEEE, 2010.

[35] FURHT, B.. **Handbook of augmented reality**. Springer Science & Business Media, 2011.

[36] IZADI, S.; KIM, D.; HILLIGES, O.; MOLYNEAUX, D.; NEWCOMBE, R.; KOHLI, P.; SHOTTON, J.; HODGES, S.; FREEMAN, D.; DAVISON, A. ; OTHERS. **Kinectfusion: real-time 3d reconstruction and interaction using a moving depth camera**. In: PROCEEDINGS OF THE 24TH ANNUAL ACM SYMPOSIUM ON USER INTERFACE SOFTWARE AND TECHNOLOGY, p. 559–568. ACM, 2011.

[37] ADAMS, R.; BISCHOF, L.. **Seeded region growing**. IEEE Transactions on pattern analysis and machine intelligence, 16(6):641–647, 1994.

[38] HOUGH, P. V.. **Method and means for recognizing complex patterns**. Technical report, 1962.

[39] DUDA, R. O.; HART, P. E.. **Use of the hough transformation to detect lines and curves in pictures**. Communications of the ACM, 15(1):11–15, 1972.

[40] KIRYATI, N.; ELDAR, Y. ; BRUCKSTEIN, A. M.. **A probabilistic hough transform**. Pattern recognition, 24(4):303–316, 1991.

[41] YLA-JAASKI, A.; KIRYATI, N.. **Adaptive termination of voting in the probabilistic circular hough transform**. IEEE Transactions on Pattern Analysis and Machine Intelligence, 16(9):911–915, 1994.

[42] MATAS, J.; GALAMBOS, C.; KITTLER, J. ; OTHERS. **Progressive probabilistic hough transform**. 1998.

[43] JOLLIFFE, I.. **Principal component analysis**. Wiley Online Library, 2002.

[44] AZUMA, R. T.. **A survey of augmented reality**. Presence: Teleoperators and virtual environments, 6(4):355–385, 1997.

[45] CRUZ, L.; LUCIO, D. ; VELHO, L.. **Kinect and rgbd images: Challenges and applications**. In: GRAPHICS, PATTERNS AND IMAGES TUTORIALS (SIBGRAPI-T), 2012 25TH SIBGRAPI CONFERENCE ON, p. 36–49. IEEE, 2012.

[46] AULINAS, J.; PETILLOT, Y. R.; SALVI, J. ; LLADÓ, X.. **The slam problem: a survey.** In: CCIA, p. 363–371. Citeseer, 2008.

[47] HENRY, P.; KRAININ, M.; HERBST, E.; REN, X. ; FOX, D.. **Rgb-d mapping: Using depth cameras for dense 3d modeling of indoor environments**. In: EXPERIMENTAL ROBOTICS, p. 477–491. Springer, 2014.

[48] WEISE, T.; WISMER, T.; LEIBE, B. ; VAN GOOL, L.. **In-hand scanning with online loop closure**. In: COMPUTER VISION WORKSHOPS (ICCV WORKSHOPS), 2009 IEEE 12TH INTERNATIONAL CONFERENCE ON, p. 1630–1637. IEEE, 2009.

[49] KOBBELT, L.; BOTSCH, M.. **A survey of point-based techniques in computer graphics**. Computers & Graphics, 28(6):801–814, 2004.

[50] SOUSA, E. V.. **D-KHT: Detecção em tempo real de planos em imagens de profundidade**. Master's thesis, Instituto de Computação, Universidade Federal Fluminense, Niterói, Rio de Janeiro, Brasil, 2016.

[51] MINECRAFT. **Minecraft**, 2016. Acesso em: Julho de 2016.

[52] HOLOLENS, M.. **Microsoft Hololens**, 2016. Acesso em: Julho de 2016.

[53] HEMMAT, H. J.; POURTAHERIAN, A.; BONDAREV, E. ; OTHERS. **Fast planar segmentation of depth images**. In: SPIE/IS&T ELECTRONIC IMAGING, p. 93990I–93990I. International Society for Optics and Photonics, 2015.

[54] LASKER, E.. **A geometric proposition**. American Journal of Mathematics, 26(2):177–179, 1904.