Ian Medeiros Coelho

Avaliação de um Algoritmo de Reconstrução 3D com Sensores RGB-D

DISSERTAÇÃO DE MESTRADO

DEPARTAMENTO DE INFORMÁTICA
Programa de Pós–graduação em Informática

Rio de Janeiro
Setembro de 2012

Pontifícia Universidade Católica
do Rio de Janeiro

Ian Medeiros Coelho

Avaliação de um Algoritmo de Reconstrução 3D com Sensores RGB-D

Dissertação de Mestrado

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

Orientador: Prof. Marcelo Gattass

Rio de Janeiro
Setembro de 2012

Pontifícia Universidade Católica
do Rio de Janeiro

Ian Medeiros Coelho

Avaliação de um Algoritmo de Reconstrução 3D com Sensores RGB-D

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

Prof. Marcelo Gattass
Orientador
Departamento de Informática — PUC–Rio

Prof. Cristina Nader Vasconcelos
UFF

Prof. Manuel Eduardo Loaiza Fernandez
Departamento de Informática — PUC-Rio

Prof. Alberto Barbosa Raposo
Departamento de Informática — PUC-Rio

Prof. José Eugenio Leal
Coordenador Setorial do Centro Técnico Científico — PUC–Rio

Rio de Janeiro, 17 de Setembro de 2012

Ian Medeiros Coelho

Graduou-se em Bacharelado em Ciência da Computação pela UESC. Tem trabalhado como freelancer em projetos de interatividade homem-máquina, com foco em visão computacional e design de interfaces, pela empresa FicTix.

Ficha Catalográfica

Coelho, Ian Medeiros

Avaliação de um algoritmo de reconstrução 3d com sensores rgb-d / Ian Medeiros Coelho; orientador: Marcelo Gattass. — Rio de Janeiro : PUC–Rio, Departamento de Informática, 2012.

v., 71 f: il. ; 29,7 cm

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

Inclui referências bibliográficas.

1. Informática – Tese. 3D Mapping; Reconstrução Densa de Superfície; Renderização de Volumes; KinectFusionI. Gattass, Marcelo. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

Agradecimentos

Ao CNPq e à PUC-Rio, pelos auxílios concedidos, sem os quais este trabalho não poderia ter sido realizado.

À minha mãe, pelo exemplo, dedicação e carinho, me apoiando em cada escolha que fiz em minha vida.

Ao meu pai, pelo apoio durante o começo do Mestrado.

Aos colegas Lucas Teixeira e Manuel Loaiza, pelas criticas e sugestões no desenvolvimento dessa dissertação, sem eles esse trabalho não seria o mesmo.

Ao amigo Adriano Medeiros pelo modelo latex que agilizou a escrita do trabalho e ao Matheus Dahora pela correção de erros ortográficos.

Ao meu orientador Marcelo Gattass, que acreditou em meu trabalho e me inspirou a alcançar meus objetivos.

Aos parceiros da FicTix Theo Ribeiro, Thiago Trindade e Eduardo Melo, pela compreensão nos momentos de ausência. Pelo apoio nos momentos de necessidade e pelo companheirismo ao longo desses ultimos anos.

Aos meus amigos, que fizeram com que esse mestrado no Rio de Janeiro longe da família fosse uma experiência fenomenal.

Resumo

Coelho, Ian Medeiros; Gattass, Marcelo. **Avaliação de um Algoritmo de Reconstrução 3D com Sensores RGB-D**. Rio de Janeiro, 2012. 71p. Dissertação de Mestrado — Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Sensores de profundidade do tipo RGB-D são alternativas interessantes para realizar a reconstrução 3D do ambiente a um baixo custo. Neste trabalho, avaliamos uma pipe-line de reconstrução implementada em GPU que une um sistema de tracking baseado no alinhamento de nuvem de pontos para estimar a posição da câmera e um sistema de reconstrução/visualização volumétrica para suavizar as medidas naturalmente ruidosas do sensor, utilizando o Kinect como entrada RGB-D. Foi feita uma análise técnica do algoritmo, demonstrando o impacto de modificações de parâmetros do sistema, além de um comparativo de tempo e precisão entre a implementação aqui apresentada e uma versão pública disponibilizada pela Point Cloud Library(PCL) durante o desenvolvimento deste trabalho. Algumas modificações em relação ao trabalho original foram feitas e os testes de desempenho demonstram que nossa implementação é mais rápida do que a da PCL sem comprometer significantemente a precisão da reconstrução.

Palavras–chave

3D Mapping; Reconstrução Densa de Superfície; Renderização de Volumes; KinectFusion

Abstract

Coelho, Ian Medeiros; Gattass, Marcelo (Advisor). **Evaluation of a 3D Reconstruction Alghorithm with RGB-D Sensors**. Rio de Janeiro, 2012. 71p. MSc Dissertation — Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Depth sensors of the type RGB-D are interesting alternatives to do a 3D reconstruction of the environment with low cost. In this work, we evaluate a reconstruction pipe-line implemented on GPU that merges a point cloud alignment tracking system to estimate the camera position and a volumetric reconstruction/visualization to smooth the naturally noise measures from the sensor, using the Kinect as RGB-D input. A technical analysis of the algorithm has been made, showing the impact of parameter modifications of the system and a comparative of time and precision between the presented implementation and a public version available by the Point Cloud Library (PCL) during the development of this work. Some modifications to the original work have been made and the performance tests demonstrate that our implementation is faster than the PCL version without compromising the reconstruction precision.

Keywords

3D Mapping; Dense Surface Reconstruction; Volume Rendering; KinectFusion

Sumário

Lista de figuras

Lista de tabelas

"Aventure-se, pois da mais insignificante pista
surgiu toda riqueza que o homem já conheceu."

John Masefield

1
Introdução

1.1
Motivação

A criação de modelos geométricos 3D a partir de sensores de captura de imagens, sem qualquer conhecimento prévio a respeito do ambiente sendo observado e em tempo real é um problema difícil, que envolve várias áreas do conhecimento como computação gráfica, cálculo numérico e computação paralela. Apesar deste assunto ter sido objeto de muitos artigos, ainda é um problema sem uma solução efciente e abrangente. Os trabalhos publicados sempre envolvem algum tipo de limitação, como assumir ambientes bem texturizados, boas condições de iluminação ou áreas pequenas. Muitas vezes sacrificam desempenho em troca de maior precisão e generalidade. Como possui uma grande quantidade de aplicações, tem atraído a atenção de muitos pesquisadores, especialmente nos últimos anos, devido ao aumento no poder computacional e na qualidade dos sensores de captura.

Na Robótica, por exemplo, [1] implementa um sistema de navegação automática de um quadrotor[1] em ambiente desconhecido através de um mapa 3D que é criado durante sua navegação utilizando o Kinect.

Um sistema de reconstrução 3D densa e preciso é capaz de estender a Realidade Aumentada (RA) para interagir diretamente com o ambiente, exatamente como é proposto em [2]. Este trabalho apresenta um sistema de simulação física de partículas que se chocam com a cena, em RA. Além disso, ele utiliza o modelo 3D criado para transformar qualquer objeto, de qualquer formato, em uma superfície multi-touch.

Além dos exemplos descritos, acreditamos que o problema estudado é a base para uma série de aplicações que não foram alvo de publicações acadêmicas, como a reprodução de ambientes reais em jogos ou sistemas de teleconferências, criação de avatares virtuais que reproduzam com fidelidade as características das pessoas, facilitar a criação de modelos CAD, mapeamento de ambientes perigosos para os seres humanos, localização em ambientes sem sinal GPS, dentre tantas outras.

[1] Um quadrotor é um helicoptero de quatro hélices.

1.2 Trabalhos Relacionados

Um fator muito importante para se desenvolver um algoritmo de reconstrução, é analisar as propriedades da entrada do sensor utilizado, e é possível notar que por muito tempo os estudos em *Simultaneous Localization and Mapping* (SLAM) aplicado à reconstrução densa têm sido focados amplamente em sistemas que utilizam sensores de cor comuns, as conhecidas câmeras RGB monoculares. Um dos pilares do SLAM é a capacidade de realizar a estimativa de pose de uma câmera virtual através do processamento de uma imagem gerada por uma câmera real. Quando isso é feito para um fluxo contínuo de imagens, quadro a quadro, damos o nome de *rastreamento de câmera* (do inglês *camera tracking*). Uma solução bastante conhecida é utilizar objetos de dimensões conhecidas pelo sistema e que sejam facilmente identificáveis, como marcadores fiduciais ([3]- figura 1.1) ou objetos de geometria bem definida [4]. Este tamanho conhecido é utilizado para criar uma relação entre a posição da câmera e tais objetos. Apesar de retornar bons resultados, esse tipo de solução é claramente limitada, pois não pode ser aplicada a ambientes desconhecidos.

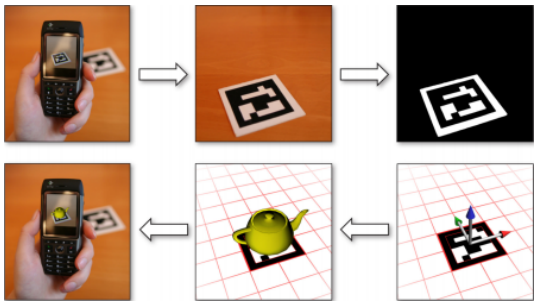

Figura 1.1: Um marcador fiducial é utilizado para realizar a estimativa de pose da câmera em uma aplicação de realidade aumentada. Retirada de [3]

Até pouco tempo atrás, o estado da arte em sistema de rastreamento de câmera sem marcadores foi o trabalho publicado em 2007, junto com uma implementação em código livre, de título *Parallel Tracking and Mapping* (PTAM - [5]). Ele baseia-se em uma técnica conhecida como Estrutura a Partir do Movimento (do inglês Structure-from-Motion (SFM)). Características extraídas de imagens de entrada consecutivas são utilizadas para estimar um modelo de velocidade para extrair a posição da câmera. Sua maior contribuição foi separar o estágio de rastreamento do processo de mapeamento, que até então eram modelados conjuntamente em um filtro probabilístico [6]. A qualidade dos resultados apresentado neste trabalho demonstrou a praticidade em separar as etapas de rastreamento e mapeamento em tarefas independentes, o que abriu caminho para que mais pesquisas fossem feitas neste segmento. Apesar da qua-

lidade dos resultados obtidos pelo PTAM, o mapa gerado pelo sistema serve apenas como marcos de referência para a etapa de rastreamento e não tem como objetivo a representação das superfícies da cena. Tendo isso em vista, [7] estendem o sistema do PTAM para realizar uma reconstrução densa da cena. Neste trabalho, é descrito uma pipeline de reconstrução que utiliza as poses do sistema de rastreamento, além dos marcos da etapa de mapeamento, gerados a partir do PTAM, para gerar um modelo de superfície bruto como base para reconstrução densa. Este modelo é utilizado parar gerar uma previsão de visibilidade que é então comparada às imagens de entrada utilizando um algoritmo de fluxo ótico em GPU. Essa comparação gera um mapa denso de correspondências, de onde são extraídos mapas de profundidade. Fundindo-se tais mapas de profundidade, obtemos a reconstrução densa da cena. Apesar da qualidade do modelo de superfície gerado, o módulo de reconstrução não pode ser executado em tempo real, mesmo quando executado em duas GPUs.

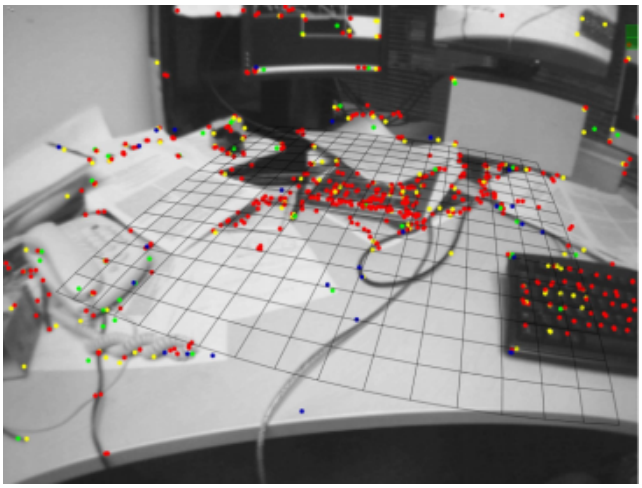

Figura 1.2: Mapa do ambiente gerado pelo PTAM. Sua natureza esparsa não retrata a geometria da cena sendo observada de forma precisa. Retirada de [5]

.

Recentemente foi publicado o Dense Tracking and Mapping (DTAM - [8]). Ele apresenta um novo sistema de rastreamento denso, onde toda a informação de cor é utilizada através de um algoritmo paralelizado e executado em GPU. Este é o primeiro trabalho que temos conhecimento que utiliza uma única câmera RGB e é capaz de alimentar o sistema de rastreamento com toda a imagem, ao invés de selecionar apenas determinados pixels por meio de algum algoritmo de detecção de características, como SIFT [9], FAST[10] ou SURF [11]. Esta abordagem demonstra melhor precisão no sistema de rastreamento, mesmo em situações de rápido movimento ou quando imagens desfocadas são utilizadas como entrada, conseguindo atualizar a posição global da câmera em condições que o PTAM não é capaz de lidar. Ela é capaz de

representar uma área de tamanho bastante limitado de forma densa e apesar do rastreamento ser feito em tempo real, a reconstrução da superfície não é integrada imediatamente ao modelo global.

Um problema comum em sistemas monoculares é o fato de que todos os algoritmos de reconstrução baseados em imagens RGB partem do principio de que é possível estimar um mapa de profundidade a partir da correlação entre duas ou mais imagens de uma mesma cena, vistas de diferentes ângulos e registradas ao mesmo tempo. Quando utilizada apenas uma câmera, em geral isso é feito levando-se em conta de que a cena é estática, permitindo que fotos tiradas em momentos diferentes possam ser consideradas do mesmo instante. Por isso, movimentos de rotação muito bruscos em geral representam um desafio para sistemas de rastreamento monoculares, pois não é possível observar uma correlação entre pixels de imagens diferentes. Essa limitação pode ser superada utilizando sistemas de múltiplas câmeras, acopladas fisicamente e sincronizadas. Este tipo de configuração é chamada de Stereo e simplifica bastante o processo de cálculo do mapa profundidade, pois sabemos exatamente qual a transformação que alinha as imagens, processo que deve ser estimado para quadros consecutivos em soluções monoculares. Apesar de facilitar a derivação do mapa de profundidade, utilizar múltiplas câmeras ainda sofre do problema de necessitar de ambientes bem texturizados para funcionar e são altamente influenciados por variações na iluminação.

O principal problema de soluções stereo é o alto custo computacional em criar o mapa de profundidade a partir do pareamento das múltiplas imagens de entrada. Este problema é atacado por [12], onde é descrito um método para rápida geração de mapas de disparidade a partir de sistemas estéreos binoculares. O algoritmo foi implementado para um único processador e já foi capaz de apresentar bons tempos de execução. O autor deixa claro que o algoritmo é facilmente paralelizável e tem o potencial para ser executado em tempo real se utilizar o poder das placas de vídeo atuais.

Sozinho [12] não ataca o problema de reconstrução 3D, mas é estendido em [13] para realizar esta tarefa. A partir do mapa de disparidades gerado pela técnica descrita em [12], é calculada uma nuvem de pontos e quadros consecutivos são fundidos de forma projetiva, gerando uma representação por pontos global da cena. Pontos estimados a partir de um par de imagens dos quadros anteriores são projetados no par de quadros atuais, resultado em um vetor de coordenadas 2D, que representa uma correspondência entre nuvens geradas por dois quadros consecutivos. Quando dois pontos correspondentes estão muito próximos, eles são fundidos para a média da posição espacial entre eles. O rastreamento da câmera é feito através de odometria calculada a partir

do pareamento de características esparsas.

Ao longo dos anos, além dos avanços nos algoritmos de rastreamento de câmera e reconstrução, houve avanços significativos na tecnologia de captura dos sensores. Câmeras baseadas em profundidade são capazes de retornar uma representação densa das distâncias para a superfície da cena observada, calculada em circuitos integrados embutidos no sensor. Essa estimativa de profundidade é um estágio que é preciso ser feito em software, caso utilizemos uma câmera RGB comum e é essencial para algoritmos de reconstrução. Isso é um grande avanço em relação a soluções fotométricas citadas anteriormente, já que todo o poder computacional pode ser utilizado para realizar tanto o rastreamento quanto a reconstrução da cena.

Sensores de profundidade laser, até pouco tempo atrás, eram as únicas alternativas disponíveis no mercado. A maior desvantagem destes sensores é dificuldade para obtê-los. Em geral eles são caros e só são vendidos por empresas especializadas. Diversas tecnologias são empregadas neste tipo de sensor, dentre a mais baratas e acessíveis citamos as câmeras Time-of-Flight (ToF). Câmeras ToF estimam a profundidade da cena calculando o tempo que um sinal laser demora para ser refletido de volta. Os mapas de profundidade gerados são bastante ruidosos e de baixa resolução quando comparados à saída de outros sensores laser. Este é o maior desafio ao utilizá-los em sistema de reconstrução 3D.

[14] mostra uma pipeline de reconstrução que utiliza um sistema de rastreamento externo para estimar poses iniciais para cada uma das entradas obtidas por um sensor ToF. Como a resolução do sensor utilziado é muito baixa, é aplicado uma super amostragem que gera um mapa de profundidade de maior resolução. Multiplas entradas são alinhadas utilizando as poses estimadas pelo sistema de rastreamento. O alinhamento é então refinado através de um método probabilístico que leva em consideração possíveis deformações não rígidas causadas pela natureza extremamente ruidosa da entrada do sensor. Cada um dos mapas de profundidade gerados são renderizados de forma independente e o problema de fundi-los em um único modelo global não é tratado.

[15] cria um mapa 3D em forma de uma nuvem de pontos, utilizando também um sensor ToF. O problema da entrada de natureza ruidosa é tratado através de uma filtragem que permite que o alinhamento entre duas entradas consecutivas seja feito ignorando-se qualquer deformação não rígida, ao contrário de [14]. Sendo assim, dois mapas de profundidade são alinhados calculando-se uma transformação rígida por meio de uma adaptação do *Iterative Closest Point* (ICP). O mapa é representado através de um grafo, que armazena cada uma das nuvens de ponto já alinhadas. Esta representação

permite que um algoritmo de otimização do mapa seja aplicado para relaxar os erros acumulados pelos alinhamentos consecutivos com ICP. Finalmente, o mapa final é refinado retirando-se pontos que estejam muito dispersos.

O alto custo envolvido nos sensores de profundidade foi resolvido com o lançamento do Microsoft Kinect no mercado de entretenimento. Atualmente podendo ser encontrado pelo valor de aproximadamente $90, o Kinect utiliza uma técnica de luz estruturada para realizar a estimativa de profundidade da cena, possuindo tanto um sensor de profundidade quanto outro de cor. Como ambos estão fisicamente acoplados a uma distância fixa, uma simples calibração estéreo é capaz de alinhar o mapa de profundidade à imagem RGB retornados por ambos os sensores. Por isso o Kinect é classificado como sensor RGB-D (RGB-Depth). Um dos problemas neste tipo de sensor é a natureza ruidosa do mapa de profundidade gerado, em que são encontrados buracos e variações nas medidas de profundidade, dentre outros ruídos. Isso representa uma dificuldade em utilizá-lo como instrumento de reconstrução 3D. A possibilidade de realizar o alinhamento das imagens geradas pela câmera RGB com o mapa de profundidade de forma simples abre espaço para algoritmos de rastreamento que utilizem tanto características geométricas, quanto fotométricas, atraindo bastante atenção de publicações desde o lançamento do Kinect.

[16] publicou um dos primeiros trabalhos que explora as facilidades fornecidas por sensores RGB-D para construir um sistema de rastreamento e reconstrução. O algoritmo descrito utiliza um sistema de rastreamento híbrido, onde tanto as informações de cor quanto de profundidade são utilizadas simultaneamente. Primeiro, uma pose inicial é estimada através da correspondência de pixels característicos entre duas imagens RGB consecutivas. Os pixels característicos são extraídos utilizando SIFT. Essa pose é então refinada alinhando as nuvens de pontos geradas a partir dos mapas de profundidade através de uma adaptação do ICP. O design do sistema de rastreamento não foi feito para ser executado em tempo real. Por último, o método de modelagem densa apresentado, baseado em uma representação de segmentos de superfície a que deu o nome de surfels, produz uma representação menos refinada que um procedimento de fusão global.

[1] demonstra uma aplicação bastante interessante para algoritmos de reconstrução/mapeamento 3D com sensores RGB-D, construindo um sistema de navegação autônoma de um quadrotor.

[17] avalia um algoritmo similar a [16], porém utiliza o método SURF para extrair as características da imagem e uma diferente aproximação para o ICP. O interessante desse trabalho é que ele realiza testes de precisão com datasets extraídos do mesmo repositório que o nosso. Inclusive em suas páginas 32

e 33, testes são feitos para um dos datasets que avaliamos, servindo como base de comparação de resultados.

No fim do ano passado foram publicados dois artigos que descrevem um algoritmo de rastreamento e reconstrução densa, chamado KinectFusion [18] e [2]. Durante a execução desta dissertação, foi lançada uma implementação deste algoritmo por uma biblioteca de código livre chamada Point Cloud Library (PCL - [19]. A pipeline de reconstrução descrita nestes trabalhos baseia-se fundamentalmente em um sistema de rastreamento denso. Todas as medidas de profundidade são convertidas para pontos 3D e, através do alinhamento de pontos de quadros consecutivos, é estimada a posição global do sensor com seis graus de liberdade. A partir das poses estimadas, o mapa de profundidade é fundido em um volume de tamanho fixo, que representa implicitamente a superfície da cena sendo reconstruída de forma densa e precisa. Uma estratégia inovadora foi utilizar o modelo sintético extraído do volume como referência de alinhamento durante o estágio de rastreamento. Isso foi demonstrado por [18] ser o grande responsável pela precisão alcançada nas estimativas de posição da câmera. Além disso, a implementação de todas as rotinas do sistema em GPU faz com que ele rode em tempo real. Como a superfície reconstruída é definida por um volume de tamanho fixo, o KinectFusion possui uma limitação no tamanho da cena sendo reconstruída. [20] ataca este problema utilizando um volume móvel, que vai reconstruindo partes da cena de acordo com a posição atual da câmera.

1.3 Objetivo e Contribuições

O objetivo deste trabalho foi utilizar os novos sensores de profundidade de baixo custo, em particular o Microsoft Kinect, para desenvolver um sistema de rastreamento e reconstrução, sem uso de marcadores, capaz de ser executado em tempo real, onde consideramos o mapeamento de ambientes com até sete metros quadrados.

Para atender estes objetivos, nos baseamos no algoritmo apresentado por [18] e [2], chamado KinectFusion. Apresentamos um sistema capaz de criar um modelo geométrico bastante preciso que é extraído enquanto o usuário navega com o Kinect pelo ambiente. Utilizando o poder computacional das placas de vídeo modernas, toda a informação disponível nos mapas de profundidade alimenta um sistema de rastreamento que roda em tempo real, sem marcadores. O sistemas descritos por [12], [14], [15], [16] , [1] e [17] não foram projetados para rodar em taxas interativas.

Como não há a necessidade de extração de características esparsas e não

utilizamos imagens RGB, ambientes com más condições de iluminação não possuem qualquer influência sobre a qualidade do rastreamento, ao contrário de soluções monoculares ou estereo como [5], [7], [21], [8] e [13], ou que utilizem a imagem RGB do Kinect, como [1], [17] ou [16].

O sistema de reconstrução utiliza um modelo volumétrico para fundir as superfícies de forma densa, ao contrário de [16], que representa o mapa 3D gerado através de segmentos de superfície a que deu o nome de surfels. Apesar de conseguir mapear maiores distâncias, [16] não representa a cena com a precisão apresentada em nosso sistema. Implementamos todo o algoritmo em CUDA, e apresentamos cada um dos estágios da pipeline de rastreamento e reconstrução de forma detalhada. Propomos uma modificação no módulo de rastreamento, descrita na seção 2, em relação ao trabalho original que demonstramos aumentar seu desempenho. Ainda não realizamos a integração das informações de cor à superfície reconstruída. A figura 1.3 exibe algumas saídas típicas do sistema.

Utilizamos a implementação disponibilizada pela PCL para realizar uma análise técnica do nosso sistema. Isso foi feito em forma de uma comparação quantitativa de desempenho e precisão, utilizando como base um dataset público [22]. Neste comparativo é possível identificar quais os gargalos e limitações do sistema para futuras contribuições.

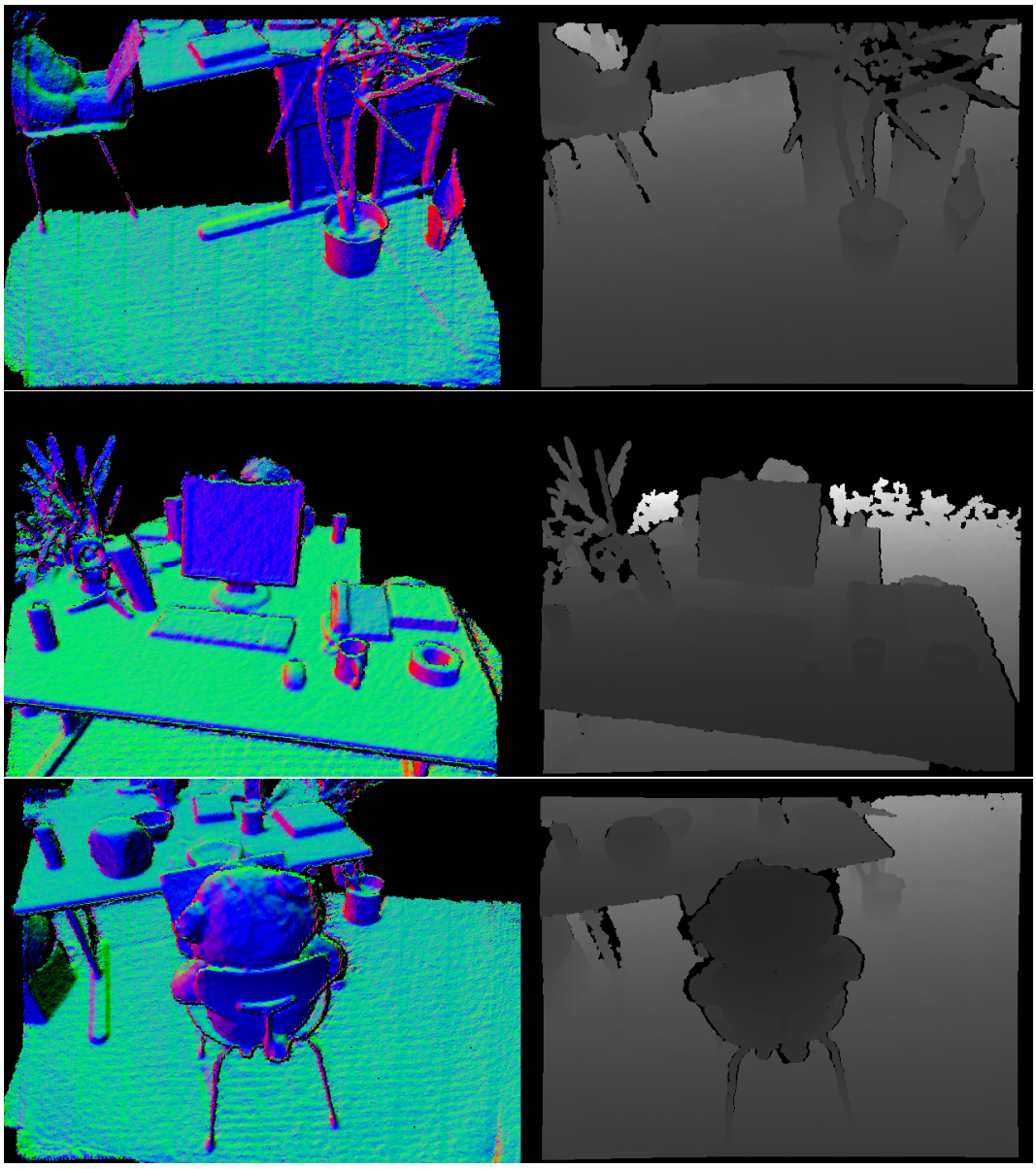

Figura 1.3: Resultados do nosso sistema de reconstrução. Esquerda, superfície da cena reconstruída sendo renderizada utilizando os valores das x,y,z das normais de cada ponto para definir as cores. Direita, mapa de profundidade de entrada.

2
Método Proposto

Nesta seção apresentamos aspectos relevantes para o nosso trabalho sobre o funcionamento do Kinect e detalhamos toda a pipeline de reconstrução.

2.1
Funcionamento do Kinect

O sensor RGB-D utilizado em nosso trabalho foi o Kinect. Ele baseia-se em um emissor infravermelho, um sensor de profundidade e outro de cor, como mostra a figura (2.1).

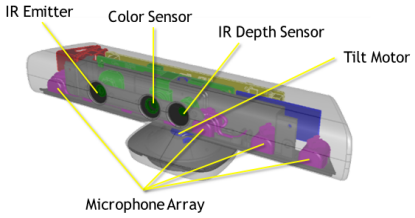

Figura 2.1: Componentes internos do Kinect.

Como descrito em [23], o emissor laser emite um único feixe que é então dividido em múltiplos feixes por uma grade de difração que cria um padrão mosqueado de imagem que é projetado na cena. A partir de uma correlação por triangulação entre este padrão projetado e outro armazenado na memória interna do Kinect, é calculado um mapa de disparidades de resolução constante de 640x480 em uma taxa de trinta quadros por segundo. Essas medidas de disparidades são armazenadas em inteiros de 11 bits, onde 1 bit é reservado para marcar valores onde medidas não puderam ser feitas e identificar o pixel como inválido. A partir deste mapa de disparidades é calculado um mapa de profundidade e como sabemos a posição dos sensores infravermelho e de cor, podemos fazer o alinhamento entre a imagem de profundidade e RGB por meio de calibração estéreo. A figura (2.2) ilustra o mapa de profundidade gerado a partir do padrão mosqueado.

Propriedades do objeto também têm um impacto nas medidas de profundidade. Como pode ser visto na figura (2.2), superfícies que refletem o laser infravermelho de forma muito forte e concentrada, ou que não o reflita com intensidade suficiente, dificultam a identificação do padrão mosqueado na imagem gerada pela câmera infravermelha, gerando pontos em que a profundidade não pode ser estimada.

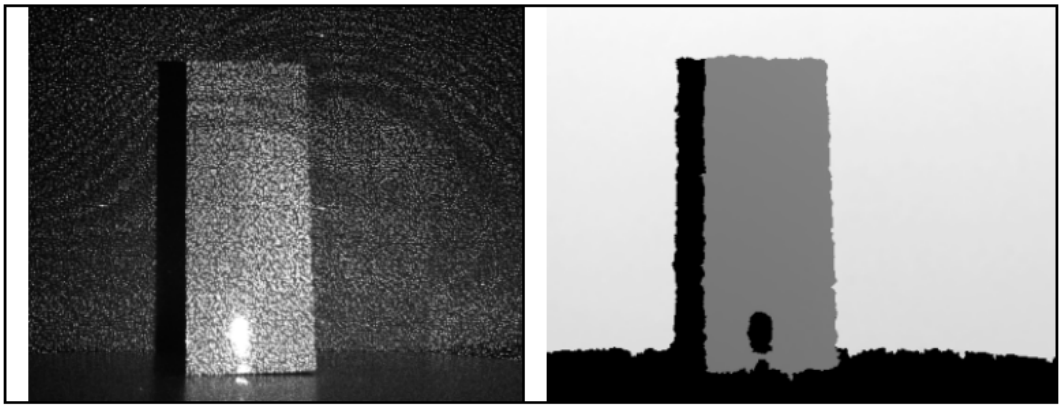

Figura 2.2: Esquerda, imagem infra vermelha do padrão mosqueado projetado em um objeto. Direita, mapa de profundidade gerado.

Note que dependendo da geometria da cena observada, partes dela podem estar oclusas ou sombreadas. Na figura (2.2), o lado direto da caixa está ocluso já que ele não pode ser visto pela câmera infravermelha mesmo podendo estar sendo iluminado pelo laser, enquanto seu lado esquerdo está sombreado pois não está sendo iluminado pelo laser mas pode ser visto pela câmera. Em ambos os casos, evidentemente não é possível se estimar a profundidade e resultam em pixels inválidos.

2.2 Pipeline de Reconstrução

A pipeline de reconstrução segue os trabalhos de [2] e [18]. A espinha dorsal de nosso sistema de reconstrução é composta por dois algoritmos. O primeiro é um algoritmo de alinhamento de nuvem de pontos conhecido como Iterative Closest Point (ICP) [24], que é a base de nosso sistema de rastreamento da câmera. O segundo é um método volumétrico de reconstrução de modelos complexos [25]. Ambos foram escolhidos por serem algoritmos que podem ser executados de forma paralela, tornando possível explorar o poder computacional das GPUs modernas e permitir sua execução em tempo real.

A pipeline de todo o sistema pode ser dividido em quatro partes principais, exibidas na figura (2.3):

(a) **Conversão do mapa de profundidade:** Esta etapa é responsável por processar o mapa de profundidade para ser usado nos próximos estágios da pipeline. Primeiro é aplicado um filtro para reduzir os ruídos da entrada. Em seguida, é realizada uma redução na amostragem na imagem de profundidade, gerando uma pirâmide em que o primeiro nível armazena a imagem em sua resolução original de 640x480 e é reduzida pela metade a cada nível subsequente. Cada mapa de profundidade da pirâmide é então

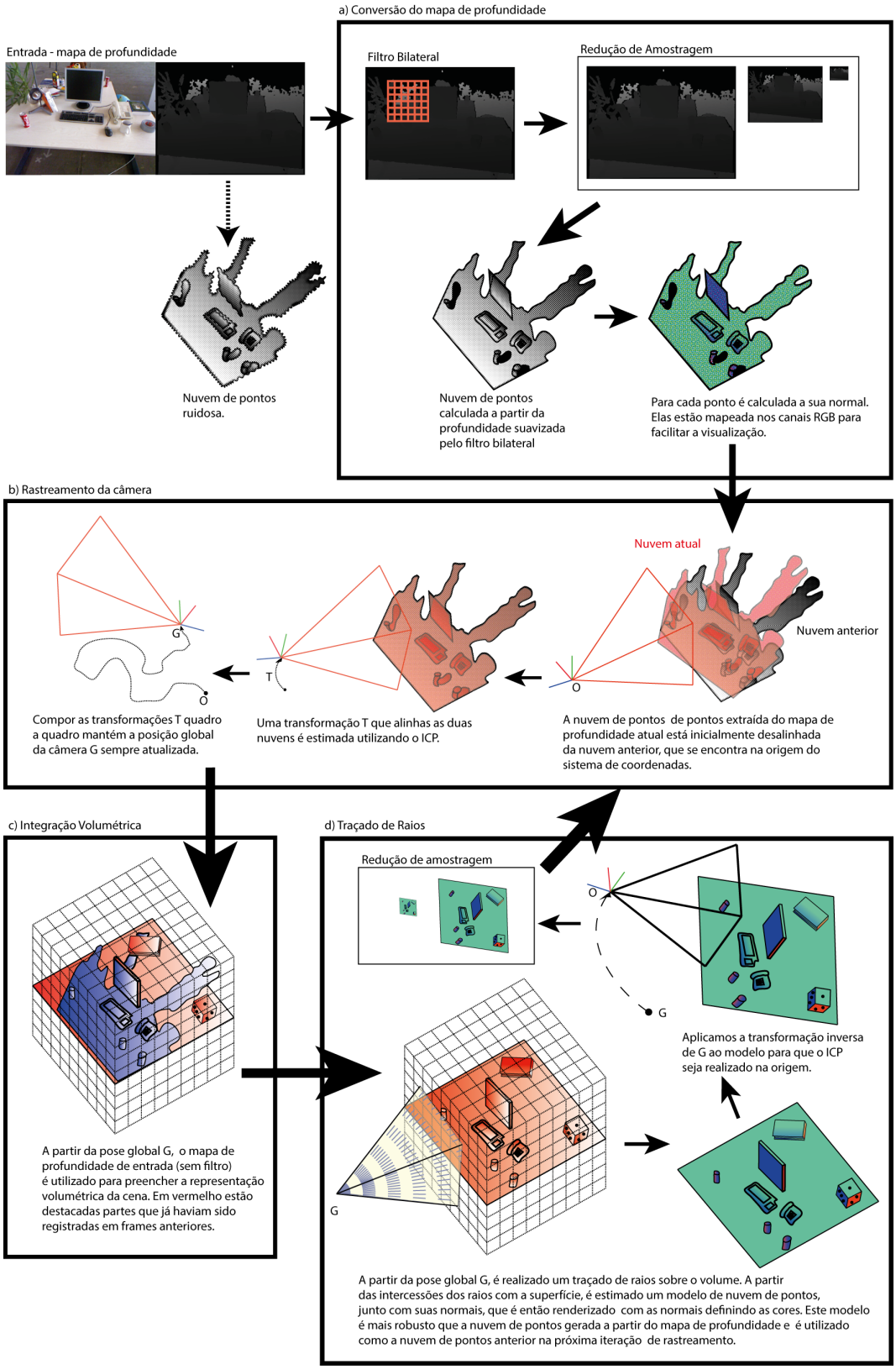

Figura 2.3: Visão geral de toda a pipeline de reconstrução, desde a aquisição do mapa de profundidade até a criação do modelo de renderização.

convertido de coordenadas de imagem para pontos 3D(que chamamos de vértice). Ao conjunto de pontos damos o nome de mapa de vértices ou nuvem de pontos. Por último, para cada ponto, sua normal é calculada, gerando um mapa de normais.

(b) **Rastreamento de câmera:** Na etapa de rastreamento calculamos uma transformação rígida com seis graus de liberdade que alinha a nuvem de pontos atual a uma nuvem de pontos anterior. Para isso, utilizamos uma implementação em GPU do ICP. A transformação encontrada é então composta com a transformação encontrada para o alinhamento do mapa de profundidade anterior, atualizando a posição global do Kinect quadro a quadro.

(c) **Integração Volumétrica:** O mapa de profundidade é utilizado para atualizar um volume que armazena uma representação implícita das superfícies da cena. Esta representação implícita é feita por meio de uma função de distâncias, onde cada voxel armazena a distância para um ponto da superfície representada pelo mapa de profundidade. Nesta abordagem, cada vértice é convertido para coordenadas globais, utilizando a pose calculada durante o estágio de rastreamento da câmera, e é utilizado para atualizar um dos voxels.

(d) **Traçado de Raios:** Por último, é realizado um traçado de raios sobre o volume. As interseções dos raios com a superfície geram uma nuvem de pontos e um mapa de normais, que são então renderizados para o usuário. Esta nuvem de pontos representa um modelo sintético da cena que por ser derivada da média de diversas medidas de profundidade, é menos ruidosa do que uma nuvem extraída a partir da entrada atual, o que a torna mais robusta para ser utilizada durante o rastreamento da câmera. Desta forma, o modelo é utilizado como referência de alinhamento para o próximo quadro.

Como em nossa implementação propomos que o alinhamento durante o ICP seja feito levado em consideração que a nuvem de pontos anterior esteja sempre em coordenadas locais, o que resulta em um ganho de desempenho considerável em relação ao trabalho original, aplicamos a transformação inversa da pose global sobre o modelo antes da próxima iteração da pipeline.

2.3 Conversão do Mapa de Profundidade

Esta etapa é responsável por realizar a captura do mapa de profundidade e processá-lo para ser utilizado nos estágios de rastreamento e integração volumétrica. Primeiro é necessário realizar a captura do mapa de profundidade (seção 2.3.1), em seguida este mapa é filtrado utilizando o filtro bilateral para que haja uma suavização dos ruídos oriundos da captura do sensor (seção 2.3.2), a saída do filtro é um mapa menos ruidoso que é então subamostrado (seção 2.3.3). Finalmente, cada subamostra do mapa de profundidades é convertido em um mapa de vértices (seção 2.3.4) de onde é derivado o mapa de normais (seção 2.3.5).

2.3.1 Captura do Mapa de Profundidade

Em nosso sistema, um mapa de profundidade bruto capturado em um instante i é definido por uma imagem R_i - do inglês raw depth - onde cada pixel **p=(u,v)** representa uma medida de profundidade $R_i(p)$. A imagem de entrada possui uma resolução fixa de largura **w=640** e **h=480**. Cada medida de profundidade é armazenada como um inteiro de 16 bits (short). Como o Kinect possui uma restrição de distância mínima para poder identificar o padrão de luz emitido pelo emissor infravermelho, utilizamos $R_i(p) = 0$ para representar qualquer pixel inválido, onde a estimativa de profundidade não pode ser feita pelos motivos citados na seção 2.1.

2.3.2 Filtro Bilateral

A partir do mapa de profundidades bruto (*raw depth*) R_i é extraído um mapa de profundidade com menor quantidade de ruído D_i (*depth*) utilizando o filtro bilateral.

Considere uma janela de raio $d \in \mathbb{N}^2$ ao redor de um pixel de interesse $\mathbf{p}$ do mapa de profundidade, formando uma região $S \in \mathbb{N}^2$. Um filtro é uma expressão que avalia todos os pixels $q \in S$ para definir o valor de $\mathbf{p}$. A figura (2.4) mostra o formato dessa janela dentro de um mapa de profundidade.

O filtro bilateral é uma extensão de um filtro conhecido como Suavização Gaussiana (Gaussian Blur) que é definido por $GB[R_i]$ [26]:

$$GB[R_i] = \sum_{q \in S} G_\sigma(||p - q||) R_{i_q} \tag{2-1}$$

Onde $||p|| = \sqrt{p_u^2 + p_v^2}$ e $G_\sigma(x)$ denota o núcleo gaussiano bidimensional:

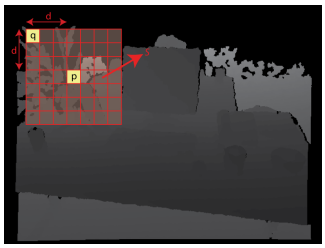

Figura 2.4: Janela de raio d ao redor de um pixel de interesse **p**. O conjunto de pixels dentro dessa janela formam uma região S. Um filtro utiliza todos os pixels $q \in S$ para definir o valor de **p**.

$$G_\sigma(x) = \frac{1}{2\pi\sigma^2} \exp(-\frac{x^2}{2\sigma^2}) \tag{2-2}$$

Dessa forma, a filtragem gaussiana é uma média ponderada de intensidades adjacentes onde há um peso decrescente com a distância espacial para a posição central **p**. Essa distância é definida por $G_\sigma(||p-q||)$, onde σ é um parâmetro que define a extensão da vizinhança, ou seja, na figura (2.4) $\sigma = d$. Como consequência, descontinuidades no mapa de profundidade são suavizadas quando aplicado o filtro gaussiano, como mostra a figura (2.5).

O filtro bilateral estende o filtro gaussiano levando em consideração as variações de intensidade para realizar uma suavização do ruído, sem desfazer as descontinuidades no mapa de profundidade [26]. O conceito por trás do filtro bilateral é que dois pixels só estarão próximos um ao outro caso ocupem posições vizinhas na imagem e, ao mesmo tempo, tenham uma similaridade no valor de profundidade . O filtro bilateral aplicado sobre o mapa de profundidade bruto R_i tem como saída um mapa filtrado D_i e é definido por:

$$D_i = BF[R_{i_p}] = \frac{1}{W_p} \sum_{q \in S} G_{\sigma_s}(||p-q||) G_{\sigma_r}(R_{i_p} - R_{i_q}) R_{i_q} \tag{2-3}$$

Onde W_p é um fator de normalização:

$$W_p = \sum_{q \in S} G_{\sigma_s}(||p-q||) G_{\sigma_r}(R_{i_p} - R_{i_q}) \tag{2-4}$$

Os parâmetros σ_s e σ_r são constantes que medem o quão o mapa de profundidade deverá ser filtrado. A equação (2-3) é uma média ponderada normalizada definida por três parâmetros:

- d: Define o tamanho da janela S a ser ponderada.
- σ_s (Sigma Space): Utilizado para calcular uma gaussiana espacial $G(\sigma_s)$ que diminui a influência de pixels distantes.
- σ_r (Sigma Range): Utilizado para calcular uma gaussiana de alcance $G(\sigma_r)$ que diminui a influência de pixels com profundidades diferentes do pixel central **p**.

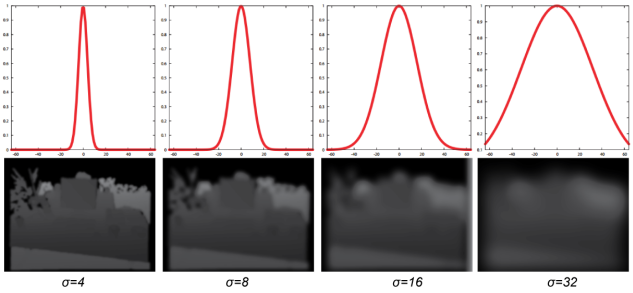

Figura 2.5: Filtro gaussiano sendo utilizado sobre um mapa de profundidade, utilizando diferentes valores de σ. A parte superior mostra o gráfico gerado pelo núcleo gaussiano bidimensional e a parte inferior mostra o borrão causado pelo valor de σ correspondente. Imagem adaptada a partir de [26].

O filtro bilateral evita suavizações de grandes descontinuidades do mapa de profundidade adicionando um peso que diminui a influência de pixels que possuam profundidades muito diferentes do pixel sendo filtrado. Essa característica do filtro selecionado é importante porque não influencia de forma drástica na geometria da cena sendo observada.

Seguindo o modelo de threads e blocos descritos em [27], o processo de paralelização do filtro bilateral para execução em CUDA é direto. Sabendo que o mapa de profundidade possui uma resolução constante de 640x480, disparamos um conjunto de blocos bidimensionais, contendo 32x8 threads, formando um grid de 20x60 blocos, como mostra a figura (2.6). Cada thread do bloco é responsável por realizar a média ponderada, dentro de uma janela de tamanho d, definida na equação (2-3), atribuindo-a a cada um dos pixels do mapa de profundidade de saída D_i.

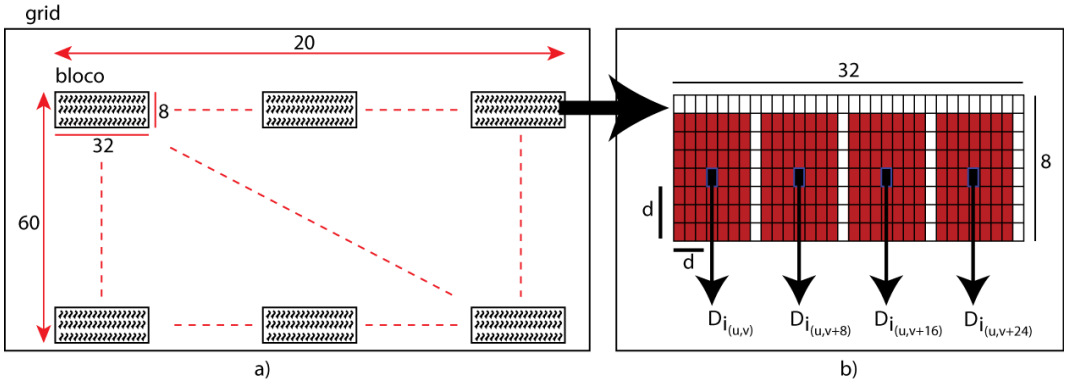

Figura 2.6: Diagrama de threads executadas na GPU utilizando CUDA para o filtro bilateral. a) Relação de número de threads/bloco e blocos/grid, b) Exemplo de execução de algumas threads de um bloco: cada thread calcula a média ponderada dentro da janela definida por d e define um pixel **p=(u,v)** do mapa de profundidade D_i , em paralelo. No exemplo b), ilustramos as threads definindo os valores de **p** com um intervalo de 8 pixels em **v**.

Como em nossa implementação do filtro bilateral sempre utilizamos

valores pequenos para d, evitamos um o cálculo de uma exponencial para cada pixel q pré-calculando todos valores de $G_{\sigma_s}(||p-q||)$ dentro do intervalo $[0, ceil(d\sqrt{2})]$ e os armazenamos em um vetor. Isso é feito pois um acesso à memória é mais rápido do que o cálculo de $G_\sigma(x)$. O mesmo não foi feito para $G_{\sigma_r}(R_{i_p} - R_{i_q})$, pois G_{σ_r}é definido em um intervalo muito grande, necessitando de muito espaço em memória na GPU, que já é um recurso escasso nesse sistema. A aplicação do filtro bilateral sobre o mapa de profundidades aumenta consideravelmente a qualidade do mapa de normais extraído pelo método descrito na seção 2.3.5.

2.3.3 Subamostragem

A subamostragem é feita através de uma representação multi escala na forma de uma pirâmide de mapas de profundidade. Esta pirâmide é armazenada em um vetor, onde cada nível L possui um mapa de resolução diferente, partindo do nível $L = 0$ com a resolução original de 640x480 e é reduzido pela metade em cada nível subsequente, como mostra a figura (2.7). Em particular, em nossa implementação utilizamos uma pirâmide de três níveis.

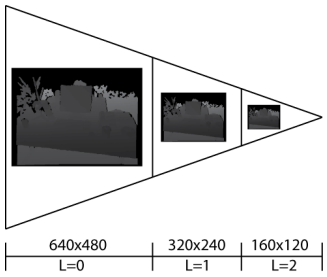

Figura 2.7: Pirâmide de amostragem com três níveis. Em cada nível, a resolução do mapa de profundidade armazenado é igual à metade da resolução do nível anterior, começando do nível L=0 com a resolução original de 640x480.

O processo de subamostragem é feito da seguinte forma: cada pixel $\mathbf{p}^{L+1}$ de um mapa de profundidade D_i^{L+1} assume o valor da média dos vizinhos do pixel $p^L = 2p^{L+1}$ dentro de uma janela de tamanho 2x2 do mapa de profundidade D_i^L, ou seja (um exemplo é exibido na figura (2.8)):

$$p_{u,v}^{L+1} = \frac{1}{4} \sum_{i=2u}^{i=2u+1} \sum_{j=2v}^{2v+1} p_{i,j}^{L} \tag{2-5}$$

A paralelização do código de subamostragem em CUDA foi feita levando-se em consideração o tamanho do mapa de profundidade de menor resolução. A quantidade de threads por bloco é a mesma que na figura (2.6), mudando assim, apenas a quantidade de blocos por grid. O número de blocos em x

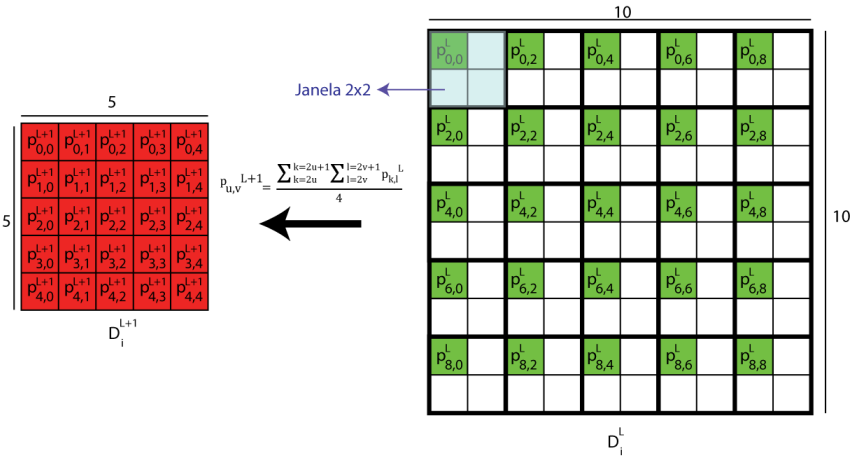

Figura 2.8: Um exemplo de subamostragem sendo realizada em um mapa de profundidade D_i^L de resolução 10x10 para um mapa D_i^{L+1} de resolução 5x5. Cada pixel $\mathbf{p}_{(u,v)}^{L+1} \in D_i^{L+1}$ é resultado da média dos pixels em uma janela de 2x2 em torno de $\mathbf{p}_{(2u,2v)}^{L} \in D_i^{L}$.

é definido por $numBlock.x = \frac{w}{2}\frac{1}{32}$, e o número de blocos em y é definido por $numBlock.y = \frac{h}{2}\frac{1}{8}$, com w e h definindo a largura e altura do mapa de profundidade de entrada, respectivamente. Por fim, cada thread de um bloco calcula um pixel p^{L+1} a partir da fórmula 2-5.

2.3.4 Mapa de Vértices

O mapa de profundidade é convertido em um mapa de vértices através de um modelo de câmera pinhole, que é definido da seguinte forma:

> "Considere um sistema de coordenadas euclidiano em que a origem seja o centro de projeção, e um plano em $Z = f$ que é chamado plano de imagem, ou plano focal. No modelo de câmera pinhole, um ponto no espaço com coordenadas $P = (X, Y, Z)$ é mapeado para o plano da imagem onde uma linha unindo o ponto P ao centro de projeção encontra o plano de imagem. ". [28]

A geometria do modelo de câmera pinhole, adaptada para o nosso problema, é mostrado na figura (2.9).

Por similaridade de triângulos conclui-se que um ponto (X, Y, Z) é mapeado para o ponto $(\frac{fX}{Z} + c_x, \frac{fY}{Z} + c_y, f)$ no plano da imagem, que é exatamente o mapa de profundidade. Ignorando a última coordenada, podemos definir a operação de projeção, que a partir de um ponto em $\mathbb{R}^3$ calculamos seu correspondente em $\mathbb{R}^2$:

$$(X, Y, Z) \mapsto (\frac{fX}{Z} + c_x, \frac{fY}{Z} + c_y) \tag{2-6}$$

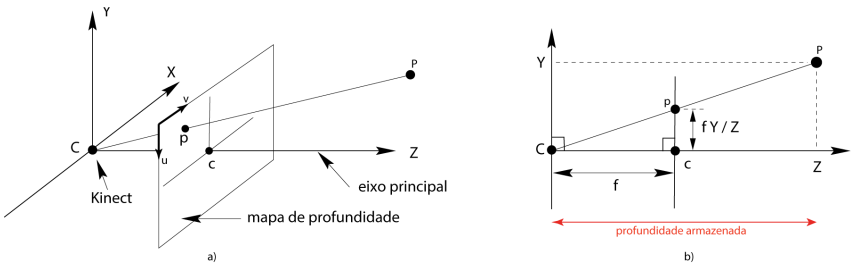

Figura 2.9: Geometria da câmera pinhole. O Kinect é o centro de câmera C, que coincide com a origem do sistema de coordenadas, e c é o ponto principal. O ponto **p** é o pixel no mapa de profundidade que armazena a coordenada Z do ponto P. Adaptado a partir de [28].

O centro de projeção é chamado de centro de câmera, e em nosso sistema representa a posição do Kinect. A projeção de um ponto P da cena sendo observada pelo Kinect sobre o mapa de profundidade nos dá um ponto $p \in \mathbb{R}^2$. Aproximando o ponto p para um número natural temos um pixel em coordenadas $\mathbf{(u,v)} \in \mathbb{N}^2$. Cada pixel armazena um valor de profundidade, que é a distância em Z do Kinect para o ponto P.

Utilizamos as coordenadas **(u,v)** de cada pixel, junto de suas respectivas medidas de profundidade Z, para gerar seus respectivos pontos 3D pelo mapeamento inverso da equação (2-6):

$$(u, v) \mapsto (\frac{1}{f}(Zu - c_y), \frac{1}{f}(Zv - c_x), Z) \tag{2-7}$$

Até o momento, consideramos que os eixos u e v possuem uma escala idêntica, mas, na prática é possível que os pixels de uma câmera calibrada não sejam quadrados. Como as coordenadas de imagem são medidas em pixels, precisamos adicionar essa assimetria em nosso modelo. Isto é feito adicionando um componente em X e outro em Y que levam em consideração o número de pixels por unidade de distância em ambos eixos. Tais componentes podem ser multiplicados pela distância focal, gerando os parâmetros f_x e f_y, resultando nos mapeamentos finais:

$$(X, Y, Z) \mapsto (\frac{f_x X}{Z} + c_x, \frac{f_y Y}{Z} + c_y) \tag{2-8}$$

$$(u, v) \mapsto (\frac{1}{f_y}(Zu - c_y), \frac{1}{f_x}(Zv - c_x), Z) \tag{2-9}$$

Sendo assim, a função de conversão do mapa de profundidade para um mapa de vértices necessita de quatro parâmetros, que recebem o nome de parâmetros intrínsecos da câmera:

– f_x e f_y: Encapsulam a distância focal e a assimetria dos pixels nos eixos

X e Y.

– c_x e c_y: Representam o ponto principal, onde o eixo principal intercepta o mapa de profundidade.

De cada nível da pirâmide de subamostragem é extraído um mapa de vértices. A implementação em CUDA é feita utilizando o mesmo modelo de threads/bloco da figura (2.6) e alterando a quantidade de blocos/threads dependendo da resolução da resolução do mapa de profundidade sendo processado, idêntico ao que é descrito no ultimo parágrafo da seção (2.3.3). Para manter a proporcionalidade no mapa de vértices gerado para cada nível da pirâmide, os valores dos parâmetros intrínsecos são reduzidos pela metade em cada nível, da mesma forma que a resolução do mapa correspondente. Como profundidades iguais a 0 são sempre inválidas, vértices com a coordenada Z igual a zero são considerados inválidos em nosso sistema.

A figura (2.10) exibe um mapa de vértices visto de três ângulos de visão, gerados a partir do mesmo mapa de profundidade.

Figura 2.10: Mapa de Vértices renderizado de três ângulos de visão diferentes. Os pontos foram coloridos utilizando a imagem RGB que vem sincronizada ao mapa de profundidade.

2.3.5 Mapa de Normais

Como cada quadro gerado a partir do sensor de profundidade é uma medida de superfície em um grid regular, é possível calcular um mapa de normais N_i utilizando um simples produto vetorial entre vizinhos de um mapa de vértices V_i. Para cada pixel $\mathbf{p}$ temos:

$$N_i(p) = normalized(V_i(u+1,v) - V_i(u,v)) \times (V_i(u,v+1) - V_i(u,v)) \quad (2\text{-}10)$$

$$normalized(x) = \frac{x}{||x||} \quad (2\text{-}11)$$

Pixels com vértices inválidos, ou vizinhos de vértices inválidos geram normais inválidas que são representadas como (0,0,0). A paralelização segue o mesmo modelo que a criação do mapa de vértices.

Como esse método de se calcular as normais utiliza uma quantidade muito pequena de vértices, qualquer ruído no mapa de vértices causa uma variação muito grande nos valores de pixels vizinhos do mapa de normais. Sendo assim, a característica do filtro bilateral de ajustar pixels semelhantes para valores próximos, sem afetar grandes descontinuidades, tem um grande impacto na qualidade do mapa de normais, como mostra a figura (2.11).

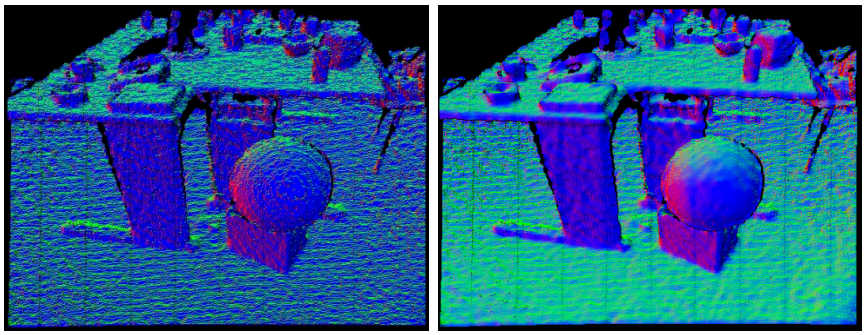

Figura 2.11: Renderização de um mapa de normais através do mapeamento dos valores de X, Y e Z de cada normal para os canais R, G e B respectivamente. Esquerda: mapa de normais gerado sem filtro bilateral. Vértices da mesma superfície geram valores de normais inconstantes. Direita: mapa gerado após aplicação do filtro bilateral com parâmetros $d = 6$, $\sigma_s = 200$ e $\sigma_r = 6$. Vértices da mesma superfície geram valores parecidos e a transição entre normais de pontos diferentes é mais suave.

2.4 Rastreamento da Câmera

O processo de rastreamento de câmera consiste em estimar uma posição da câmera para cada um dos mapas de profundidade de entrada. Nós representamos a posição global da câmera para um quadro no instante i, com seis graus de liberdade, por uma matriz de transformação de corpo rígido:

$$T_{g,i} = \begin{pmatrix} R_{g,i} & t_{g,i} \\ 0 & 1 \end{pmatrix} \in \mathbb{SE}^3 \tag{2-12}$$

Essa transformação transfere um vértice v_i obtido no instante i para o sistema de coordenadas global g através da fórmula $v_g = T_{g,i}v_i$. Uma superfície é representada pelo par formado entre o mapa de vértices e o mapa de normais $S_i = (V_i, N_i)$. A estimativa da posição da câmera é feita pelo alinhamento de superfícies através de um método numérico conhecido como *Iterative Closest Point* (ICP - [24]). Neste método, calculamos uma transformação que minimiza uma função de energia entre pontos correspondentes de forma iterativa, a cada iteração nos aproximamos da transformação ótima. Tanto a função de energia a ser minimizada, quanto a associação de pontos podem ser feitas de diversas maneiras, mas assumir um pequeno movimento de um quadro para o outro (pois garantimos uma rápida taxa de quadros por segundo) nos permite utilizar o algoritmo de associação projetiva de pontos [2] e a métrica de ponto-para-plano [29] como função de energia. Esses dois algoritmos podem ser paralelizados, permitindo que exploremos o poder computacional das placas de vídeo atuais para rodar o sistema em taxas interativas. Para obter maior precisão no sistema de rastreamento realizamos o alinhamento da superfície atual (V_i, N_i) à um modelo de superfície $(\widehat{V}_{i-1}, \widehat{N}_{i-1})$, extraído de nossa representação volumétrica da cena, assim como em [18]. Como vemos na figura (2.3), compor as transformações que alinham as superfícies, quadro a quadro, mantém a posição da câmera sempre atualizada. Na seção (2.4.1) descrevemos como é feita a associação projetiva de pontos, a seção (2.4.2) explica a métrica de ponto-para-plano e a seção (2.4.3) descreve como compor a associação projetiva e a minimização de ponto para plano no fluxo do ICP.

2.4.1 Associação Projetiva de Pontos (APP)

A associação de pontos projetiva é uma forma simples e rápida de encontrar pontos correspondentes entre duas superfícies. O conceito por trás deste algoritmo é que quando dois pontos muito próximos de duas superfícies diferentes são projetados para o mesmo plano de imagem, a coordenada 2D de ambos os pontos também será muito próxima. Se esta coordenada 2D

corresponder ao mesmo pixel e a distância entre os dois pontos, junto com o ângulo formado entre suas normais, respeitem um limite definido pelo usuário dizemos que os pontos são correspondentes. Pontos que sejam projetados para fora do frustum de visão são ignorados.

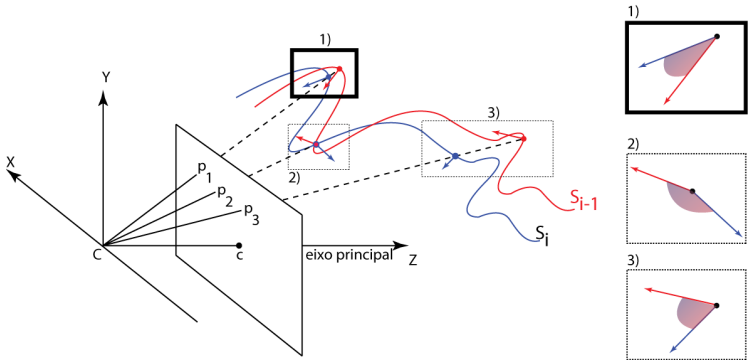

Figura 2.12: Geometria do algoritmo de associação projetiva.

Como mostra a figura (2.12), cada ponto da superfície S_i é projetado em um pixel $\mathbf{p}$ através do mapeamento (2-8). Em seguida verificamos se os pontos das superfícies S_i e S_{i-1} armazenados em $\mathbf{p}$ estão dentro de uma distância máxima e se suas respectivas normais formam um ângulo menor do que uma constante do sistema. 1) A distância e o ângulo entre as normais dos pontos estão dentro de um limite aceitável e eles são considerados correspondentes. 2) Apesar dos pontos estarem sobrepostos, respeitando assim a distância limite, o ângulo entre as normais é muito grande e os pontos não são considerados correspondentes. 3) A distância entre os pontos é muito grande, fazendo com que os pontos não sejam associados.

Em nossa implementação deste algoritmo, realizamos uma alteração em relação ao que é descrito em [2] que resultou em um ganho de desempenho considerável do sistema. Em [2], o modelo de superfície S_{i-1} é armazenado sempre em coordenadas globais. Ao contrário, S_i é armazenado em coordenadas da câmera. Isso significa que toda vez que for necessário realizar a associação projetiva de pontos entre S_{i-1} e S_i, necessitamos aplicar a transformação inversa $T_{g,i-1}^{-1}$ aos vértices do modelo S_{i-1} antes de projetar seus vértices. Como o ICP aplica a associação projetiva de pontos em cada iteração, isso significa que para cada passo do ICP, é necessário realizar o produto de uma matriz 4x4 sobre um vetor 1x4 para cada pixel do mapa de vértices. Além deste problema, o algoritmo descrito em [2] considera que a superfície atual S_i é sempre armazenada em coordenadas locais da câmera. Dessa forma, a transformação T_{opt}^{k-1}, calculada na iteração anterior do ICP (seção 2.4.3), é aplicada a S_i durante a associação projetiva. Isso é feito em cada passo do ICP. Apesar de ser uma operação que não pode ser evitada, quando o algoritmo

é implementado em CUDA esta transformação é feita após um acesso não ordenado à memória, pois os pixels obtidos através da operação de projeção sobre cada um dos vértices de V_i não são garantidos de serem gerados de forma ordenada. Como vemos em [27], isso resulta em uma perda de desempenho no escalonamento das threads durante a execução do código na GPU.

Propomos aqui duas alterações:

- Armazenar o modelo de superfície S_{i-1} em coordenadas locais. Dessa forma, a transformação $T_{g,i-1}$ vai ser aplicada ao modelo uma única vez, logo após o traçado de raios, ao invés de aplicá-la todas as vezes que for necessário fazer a associação projetiva.
- Armazenar a superfície S_i, com os vértices já transformados por T_{opt}^k, aplicando-a logo após a solução da iteração k do ICP. Isso retira esta operação de transformação de dentro da associação projetiva e garante o acesso ordenado à memória quando for aplicada.

2.4.2 Minimização de Ponto-para-Plano

Quando a métrica de ponto-para-plano é utilizada, a função de energia a ser minimizada é definida pelo quadrado da soma das distâncias entre um ponto de origem e o plano tangente ao seu ponto de destino correspondente, como mostra a (2.13). Mais especificamente, se $o_i = (o_{ix}, o_{iy}, o_{iz})$ é um ponto de origem, $d_i = (d_{ix}, d_{iy}, d_{iz})$ é seu ponto correspondente de destino e $n_i = (n_{ix}, n_{iy}, n_{iz})$ é o vetor unitário normal em d_i, então o objetivo em cada iteração do ICP é achar uma transformação T_{opt} tal que:

$$T_{opt} = arming_T(\sum_i (Ts_i - d_i).n_i)^2 \tag{2-13}$$

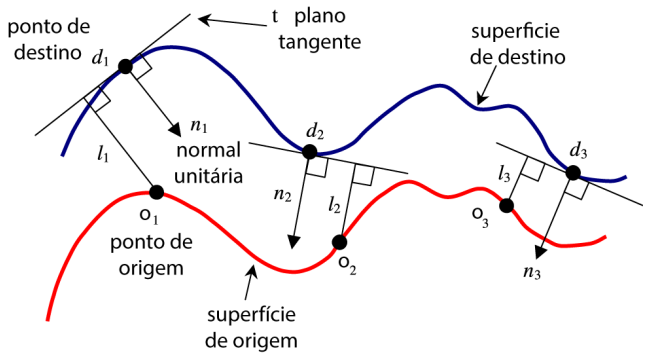

Figura 2.13: Geometria do algoritmo de associação projetiva.

A equação (2-13) é essencialmente um problema de otimização de mínimos-quadrados. Sua solução requer a determinação de seis parâmetros

$\alpha, \beta, \gamma, t_x, t_y$ e t_z onde α,β,γ são os parâmetros que definem a rotação de T_{opt} em torno dos eixos X, Y e Z e t_x, t_y e t_z são os parâmetros de translação.

Assumir uma transformação infinitesimal de uma superfície para a outra nos permite representar a rotação como 3 parâmetros de uma matriz anti-simétrica, resultado da simplificação gerada pela eliminação dos senos e cossenos presentes nas matrizes de rotação (sen(0) = 0 e cos(0) = 1). Isso nos permite solucionar a função (2-13) como o resultado de um sistema de equações lineares [29]:

$$T_{opt} = arming_T(\sum_i (Ts_i - d_i).n_i)^2 \cong argmin_x|Ax - b|^2 \tag{2-14}$$

Onde

$$x = (\alpha, \beta, \gamma, t_x, t_y, t_z) \tag{2-15}$$

$$A = \begin{pmatrix} o_1 \times d_1 & n_1 \\ o_2 \times d_2 & n_2 \\ \vdots & \vdots \\ o_n \times d_n & n_n \end{pmatrix} \tag{2-16}$$

$$b = \begin{pmatrix} n_1(d_1 - o_1) \\ n_2(d_2 - o_2) \\ \vdots \\ n_n(d_n - o_n) \end{pmatrix} \tag{2-17}$$

Como sabemos que 0 é o valor que simplifica a função $|.|$, basta solucionar o sistema $Ax + b = 0$ para achar o valor de x.

2.4.3 ICP

O ICP é um algoritmo iterativo que em cada iteração é calculada uma transformação T_{opt}^k que minimiza um sistema de equações que é gerado pela substituição dos pontos correspondentes em uma função objetivo. A figura (2.14) exibe o fluxo de execução de todo um ciclo do ICP.

O modelo de superfície anterior $(\widehat{V}_{i-1}, \widehat{N}_{i-1})$ é associado à superfície atual (V_i, N_i) utilizando o algoritmo de Associação Projetiva de Pontos, gerando a função objetivo $|Ax+b|_k$, onde cada ponto associado gera uma linha da matriz A e do vetor b. Como o mapa de profundidade possui uma resolução de até 640x480, dependendo do nível da pirâmide de subamostragem, é necessário resolver um sistema de até 307200 linhas. Isso é evitado utilizando a relação

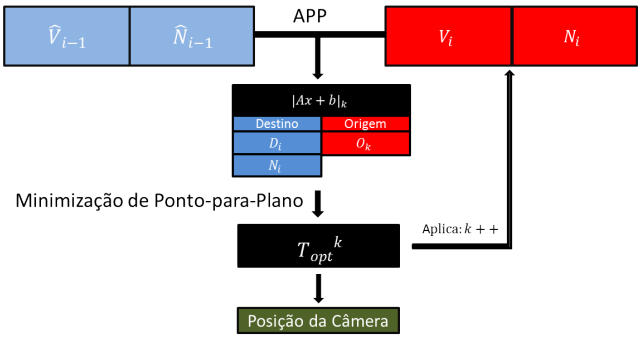

Figura 2.14: Fluxo de execução do ICP entre os quadros capturados nos instantes i e $i-1$.

$A^TAx = A^Tb$, o que faz com que o sistema seja simplificado para 6 linhas, pois, considerando as dimensões das matrizes resultantes a partir do produto de matricial, temos:

$$A^TA \implies (6 \times N)(N \times 6) \implies (6 \times 6) \tag{2-18}$$

$$A^Tb \implies (6 \times N)(N \times 1) \implies (6 \times 1) \tag{2-19}$$

As simplificações (2-18) são implementadas em CUDA utilizando uma redução paralela baseada-em-árvore, também conhecida como primitiva scan [30].

O somatório do produto de cada linha de A por uma coluna de A^T é calculado em paralelo através da primitiva scan. Cada bloco de 256 threads reduz 256 valores desse somatório, que é então armazenado em um vetor com (640*480)/256=1200 somatórios. A primitiva scan é aplicada novamente sobre este vetor, utilizando quatro blocos de 256 threads, gerando cada um dos índices da matriz 6x6 AA^T. Os índices pretos da (figura 2.15) demonstram que é possível economizar memória e tempo de processamento explorando a simetria de AA^T e calculando apenas sua parte triangular superior na GPU que é apenas espelhada para a parte triangular inferior quando passamos o sistema para a CPU, onde o sistema é solucionado utilizando decomposição Cholesky.

A transformação x então é aplicada sobre a superfície (V_i, N_i) e uma nova iteração do ICP é feita. Isso se repete por um número predeterminado de vezes para cada um dos níveis da pirâmide de subamostragem. A transformação da última iteração é composta à posição da câmera atual, atualizando assim a transformação $T_{g,i}$.

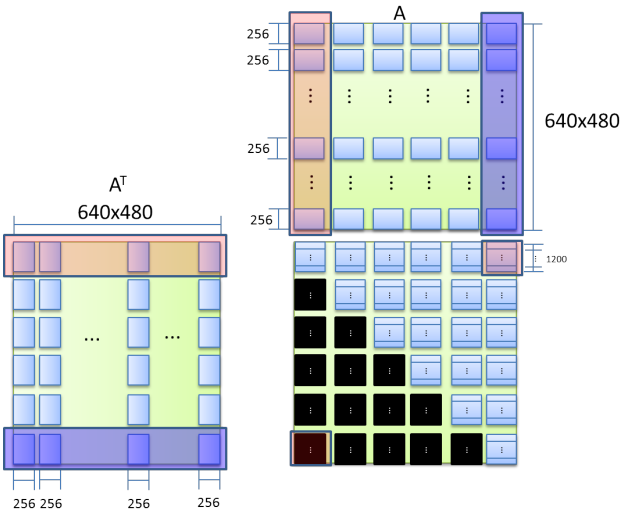

Figura 2.15: Execução de AA^T em CUDA para o nível 0 da pirâmide de subamostragem em blocos unidimensionais de 256 threads.

2.5 Reconstrução Volumétrica

Em uma representação explícita, definimos uma superfície escrevendo explicitamente cada um dos pontos contidos em sua fronteira. Alternativamente, uma representação implícita define a fronteira da superfície como o isocontorno de alguma função [31]. Por exemplo, considerando um sistema unidimensional, o isocontorno zero de $\phi(x) = x^2 - 1$ é o conjunto de pontos onde $\phi(x) = 0$, mais especificamente os pontos -1 e 1.

Utilizando a posição global da câmera estimada através do ICP, todos os mapas de profundidade de entrada são fundidos em um único sistema de coordenadas global. Estes mapas são integrados utilizando uma representação volumétrica e implicita de superfície, formando um campo de distâncias tridimensional. Um volume 3D de resolução fixa e alinhado aos eixos X, Y e Z é definido, que mapeia pontos específicos de um espaço físico. Esse volume é então dividido em um grid de voxels. Em particular, sendo o tamanho e resolução do volume definidos por $size = (size_x, size_y, size_z)$ e $res = (res_x, res_y, res_z)$ respectivamente, mapeamos sempre uma região que tenha exatamente metade do volume à esquerda da posição inicial da câmera, outra metade à sua direita e esteja sempre à sua frente, ou seja ,a conversão de coordenadas do grid g_{xyz} para coordenadas globais v^g_{xyz} e vice versa é feita através das relações:

$$g_{xyz} = (\frac{\frac{size_x}{2} + v_x^g}{\frac{size_x}{res_x}}, \frac{\frac{size_y}{2} + v_y^g}{\frac{size_y}{res_y}}, \frac{v_z^g}{\frac{size_x}{res_x}}) \quad (2\text{-}20)$$

$$v_{xyz}^g = (\frac{size_x}{res_x} g_x - \frac{size_x}{2}, \frac{size_y}{res_y} g_y - \frac{size_y}{2}, \frac{size_z}{res_z} g_z) \quad (2\text{-}21)$$

Os vértices globais são fundidos em voxels utilizando uma variação da *Função de Distâncias com Sinal* (FDS), onde cada voxel armazena um valor $F \in \mathbb{R}$ que representa a distância da posição global do voxel para a superfície sendo reconstruída. Esses valores são positivos fora da superfície, negativos dentro dela e a fronteira da superfície é definida pela travessia em zero (do inglês *zero-crossing*) onde os valores mudam de sinal. Como limitamos os valores de F que são armazenados em torno da superfície a uma distância máxima μ, nossa representação recebe o nome de *Função de Distâncias Truncadas com Sinal* (FDTS). O resultado de acumular as médias de FDTS's de várias nuvens de pontos que estão alinhadas a um sistema de coordenadas global é uma fusão da superfície global. A figura (2.16) mostra como é possível extrair uma superfície arbitrária através desta representação.

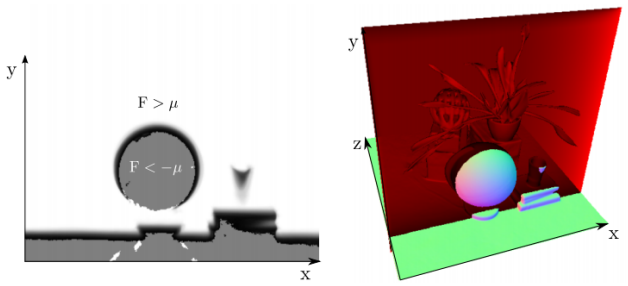

Figura 2.16: Uma fatia vertical sobre o volume de distâncias truncadas com sinal. A superfície é extraída a partir da travessia em zero, onde F muda de sinal. Ela fica armazenada nos voxels que ficam entre a região de valores truncados com $F > \mu$ (branco) e onde medidas ainda não foram feitas (cinza). [18]

O processo de calcular as FDS entre as posições globais dos vértices e cada um dos voxels, integrando-as aos valores já calculados a partir de quadros anteriores, recebe o nome de *Integração Volumétrica*. Para cada voxel, calcula-se o valor de uma função de distância entre sua posição global e um dos pontos do mapa de profundidade. A associação voxel-vértice é feita de forma projetiva. Como o cálculo de cada um dos FDS independe do resultado de voxels vizinhos, o algoritmo pode ser paralelizado para execução eficiente em GPU.

A figura (2.17) ilustra como é feito o calculo de FDS para um dos voxels v^g. A coordenada global do voxel é projetada sobre o plano de imagem, obtendo

um pixel **p**. O FDS é então resultado da diferença entre a distância de v^g para a posição atual da câmera $t_{g,i}$ e a profundidade em **p**:

$$fds(v^g) = \lambda R_i(p) - ||t_{g,i} - v^g|| \tag{2-22}$$

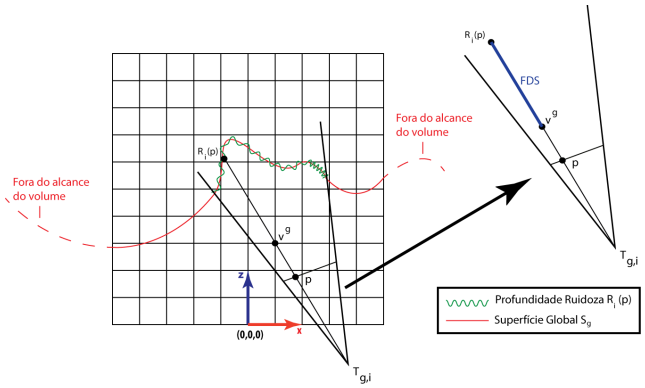

Figura 2.17: Cálculo do FDS para um voxel com coordenada global v^g tendo como ponto de vista a fatia do plano ZX do volume de reconstrução.

Na figura (2.9), ilustramos que o valor armazenado no mapa de profundidade é medido ao longo do eixo principal. Como calculamos o valor da FDS ao longo do raio que aponta a partir do centro da câmera para **p**, precisamos aplicar um fator de escala λ sobre $R_i(p)$ para que este último seja representado ao longo do mesmo raio. Utilizando uma simples relação trigonométrica, temos:

$$\lambda = \sqrt{((\frac{p_v - c_x}{f_x})^2 + (\frac{p_u - c_y}{f_y})^2 + 1)} \tag{2-23}$$

Na prática, apenas distâncias que estejam dentro de um limiar máximo μ são representadas no volume. Voxels fora do frustum de visão ou que possuam $fds < -\mu$ durante a integração são ignorados. Voxels com $fds > \mu$ são truncados para um único valor, resultando na função de distâncias truncada com sinal:

$$fdts(v^g) = \begin{cases} \min(1, \frac{fds}{\lambda}) & \text{se } fds \geqslant -\mu \\ null & \text{caso contrário} \end{cases} \tag{2-24}$$

A fusão global de todos os mapas de profundidade no volume é formada a partir da média ponderada de todas FDTS's individuais calculadas para cada mapa de profundidade. Já que a média entre várias medidas ruidosas tende a reproduzir com maior exatidão a posição real da superfície sendo reconstruída, o acúmulo de várias FDTS's origina uma superfície mais suave do que a extraída dos mapas de profundidade originais, representada em vermelho na figura (2.17). Esse procedimento é feito armazenando um peso $W_i(v^g)$ em

cada um dos voxels, que é incrementado até um limite W_{max} toda vez que o FDTS do voxel correspondente é atualizado em um instante i:

$$fdts_i(v^g) = \frac{(W_{i-1}(v^g)fdts_{i-1}(v^g) + W_i(v^g)fdts_i(v^g))}{(W_{i-1}(v^g) + W_i(v^g))} \tag{2-25}$$

$$W_i(v^g) = min(W_{i-1}(v^g) + W_i(v^g), W_{max}) \tag{2-26}$$

Em nossa implementação em CUDA, representamos o grid regular em um vetor de tamanho fixo. Encapsulamos cada par $(fdts_i, W_i)$ em um único endereço de 32 bits, onde o $fdts_i$ é armazenado como um half float (16 bits) e W_i como um short (16 bits). Como o número de voxels é muito maior do que o limite de threads por kernel, para cada coordenada (x, y) da fatia frontal do volume é atribuída uma thread. Cada thread então realiza a atualização dos voxels desde $z = 0$ até a resolução máxima do volume em Z. A alocação da memória é feita de forma alinhada, garantindo o acesso a partir de threads paralelas de forma aglutinada, aumentando assim o rendimento de memória.

2.6 Extração da Superfície por Traçado de Raios

De posse da reconstrução mais atual, extraímos a superfície codificada no nível zero $fdts_i = 0$. Isso é feito através de um traçado de raios a partir da pose global $T_{g,i}$ sobre o volume global, gerando uma superfície modelo $(\widehat{V}_{i-1}, \widehat{N}_{i-1})$, que é utilizada tanto para visualização, quanto para realizar o rastreamento da câmera para o próximo quadro.

Para cada pixel $\mathbf{p}$ do plano de imagem do frustum de visão atual é disparado um raio que parte da posição atual da câmera até que uma travessia em zero (de um FDTS > 0 para um FDTS < 0) seja encontrada, gerando um vértice $\widehat{v}$, ou até que o raio ultrapasse a dimensão do volume. Quando a travessia não é encontrada, o raio gera um pixel de vértice e normal inválidos. Para pontos muito próximos, ou sobre a superfície em $fdts(\widehat{v}) = 0$, é assumido que o gradiente da FDTS em v^g é ortogonal ao conjunto de nível zero, o que significa que a normal da superfície encontrada para o pixel $\mathbf{p}$ pode ser computada diretamente através da derivada numérica da FTDS. Em particular, utilizamos uma aproximação por diferença central de precisão de segunda ordem:

$$\widehat{N}_x = \frac{\partial fds}{\partial x} \approxeq \frac{fdts(\widehat{v}_{x+\Delta}) - fdts(\widehat{v}_{x-\Delta})}{|\Delta|} \tag{2-27}$$

Na equação (2-27), os valores de $fdts(\widehat{v}_{x+\Delta})$ e $fdts(\widehat{v}_{x-\Delta})$ são calculados através de uma interpolação trilinear. O mesmo princípio é aplicado para

calcular os valores de $\widehat{N}_y$ e $\widehat{N}_z$.

Como sabemos o valor da distância de truncagem, podemos percorrer o raio em passos de tamanho fixos $< \mu$, pois isso nos garante que iremos passar por pelo menos um valor não truncado positivo de tsdf antes de passar para um valor negativo, encontrando assim o conjunto de nível zero.

Realizamos uma aproximação linear para calcular a interseção do raio com a superfície de forma eficiente. Dado que o raio intercepta a FDS onde F_t^+ e $F_{t+\Delta t}^+$ são valores de FDS trilinearmente interpolados de cada lado da travessia em zero nos pontos t e $t + \Delta t$ ao longo do raio a partir de sua posição inicial, encontramos um ponto t^* no qual a interseção acontece mais precisamente:

$$t^* = t - \frac{\Delta F_t^+}{F_{t+\Delta}^+ - F_t^+} \tag{2-28}$$

O mapa de vértices e de normais do modelo são então calculados a partir em t^*.

3
Experimentos e Resultados

Como análise técnica, foram realizados testes comparativos de precisão e desempenho entre nossa implementação e a versão pública disponibilizada pela PCL, utilizando o data-set público [22]. Neste data-set, temos disponíveis 39 sequências de ambientes internos, cada uma contendo as imagens de cor e profundidade obtidas com um Kinect e um ground-truth obtido com um sistema de captura de movimento. Dentre as 39 opções, foram escolhidas 3 sequências que se encaixam bem no contexto do algoritmo proposto, pois não representam áreas muito grandes e são capazes de avaliar a qualidade do sistema de rastreamento da câmera, são elas:

1. freiburg1 xyz
2. freiburg2 xyz
3. freiburg2 desk

As duas primeiras são sequências onde se evita movimentos de rotação e o movimento é feito principalmente nos eixos X, Y e Z. Apesar de parecidas, a segunda sequência possui muito mais frames de entrada para se estimar a pose da câmera, o que torna possível avaliar o desalinhamento causado pelos acúmulos de erros oriundos tanto do método numérico utilizado, quanto das aproximações por ponto flutuante ao longo do tempo. A terceira sequência é uma rotação completa em torno de uma mesa com vários objetos em cima e ao seu redor que garantem a riqueza de detalhes geométricos no mapa de profundidade, necessários para realizar um bom alinhamento entre duas nuvens de pontos subsequentes. A figura (3.1) mostra as trajetórias obtidas com o sistema de captura de movimento e imagens para identificar cada um dos data-sets.

3.1
Precisão do Sistema de Rastreamento da Câmera

O primeiro passo nos testes de precisão, foi comparar a trajetória estimada pela nossa implementação para cada uma das sequências com a trajetória obtida com a PCL. Para realizar o comparativo, todas as variáveis do sistema foram colocadas para valores idênticos, sendo elas:

– Parâmetros de Calibração da Câmera

f_x: 525

f_y: 525

c_x: 319.5

c_y: 239.5

– Parâmetros do filtro bilateral

σ_r (Sigma Range): 30

σ_s (Sigma Space): 4.5

Tamanho da janela d: 6

– Parâmetros do ICP

Número de iterações: 4,5,10

Treshold de distância entre pontos correspondentes: 0.1

Treshold do ângulo entre normais correspondentes: 0.34

– Parâmetros da Integração/Visualização Volumétrica

Distância de truncagem: 0.03

Tamanho do volume (m): 3,3,3 (x,y,z)

Resolução do volume: 512 x 512 x 512

Tamanho do passo no traçado de raios: 0.8 * distância de trucagem

O gráfico das trajetórias estimadas com a PCL e nossa implementação são exibidas na figura (3.1).

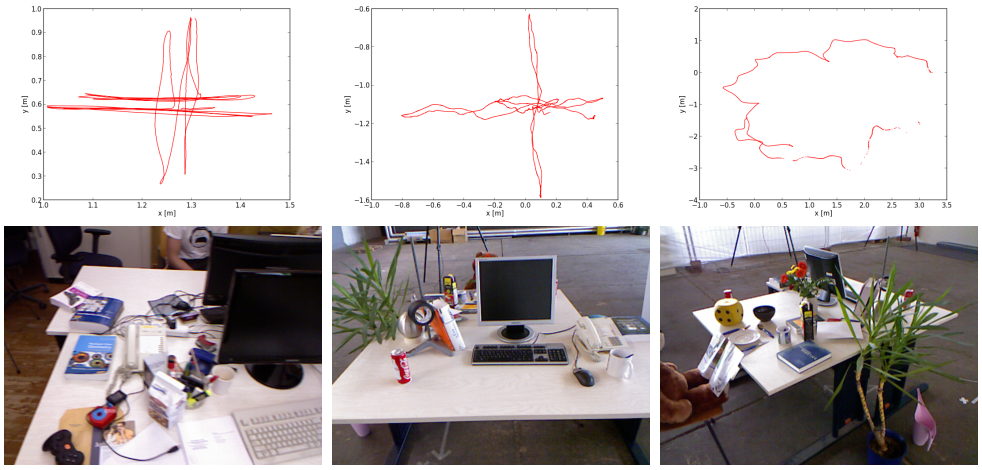

Figura 3.1: Groundtruth e imagens de referência para as sequências 1,2 e 3 respectivamente.

Como vemos na figura (3.1), as trajetórias não são exatamente iguais, mas estão muito próximas, como esperado. Isso acontece porque existem

diferenças entre ambas as implementações que influenciam na pose estimada, como descrito na seção (3.1.1).

Para avaliar a precisão de ambas as implementações, foram feitos testes em relação ao ground-truth utilizando as duas métricas de erro propostas por [22], sendo elas o Erro de Trajetória Absoluto(ETA) e o Erro de Posição Relativo(EPR). Como valor numérico de comparação, utilizamos a Raiz Quadrada da Média dos Erros (RQME).

3.1.1 Erro de Trajetória Absoluto (ETA)

O erro de trajetória absoluto é uma forma de se avaliar a consistência global da trajetória estimada. Essa métrica consiste em comparar a distância absoluta entre uma trajetória de referência e outra trajetória estimada. Como ambas podem estar definidas em sistemas de coordenadas arbitrárias, primeiramente é necessário alinhá-las.

Para realizar essa avaliação utilizamos o script escrito em Python disponibilizado pelos autores de [22] que já faz o alinhamento entre duas trajetórias e desenha os gráficos das diferenças. Para realizar o alinhamento, o script espera como entrada duas trajetórias, uma de referência e outra que será alinhada a ela. Como o sistema de coordenadas do sistema de captura de movimento que gera o ground-truth possui um sistema de coordenadas muito diferente da PCL e da implementação proposta nesse trabalho (a origem e eixos de rotação da trajetória não são definidos pela posição inicial do Kinect) utilizamos como referência a trajetória gerada pelo algoritmo proposto para gerar os gráficos comparativos "Proposto vs ground-truth" e "Proposto vs PCL", enquanto a trajetória gerada pela PCL foi usada como referência para os gráficos "PCL vs ground-truth". Dessa forma, os desenhos ficam mais parecidos e torna mais simples a inspeção visual para realizar os comparativos. Vale notar, que apesar da forma entre gráficos serem parecidas, as coordenadas são diferentes. Isso acontece, pois, sendo (v_x, v_y, v_z) o tamanho definido para o volume, a PCL o define no intervalo $(0, 0, 0) - (v_x, v_y, v_z)$ e utiliza como pose inicial para a trajetória a posição $(v_x/2, v_y/2, 1.2 * v_z/2)$, diferente do que definimos na seção (2.5) para o posicionamento do volume de reconstrução, além de utilizarmos (0,0,0) como posição inicial.

As figuras (3.3) à (3.6) representam as trajetórias estimadas para a sequência 1, que é a menos desafiadora das três sequências. Ela possui uma trajetória curta, com simples translações nos eixos X,Y e Z, com poucas rotações envolvida nas estimativas das poses e é representada por poucos frames de entrada. Como a quantidade de frames é pequena, vemos esses

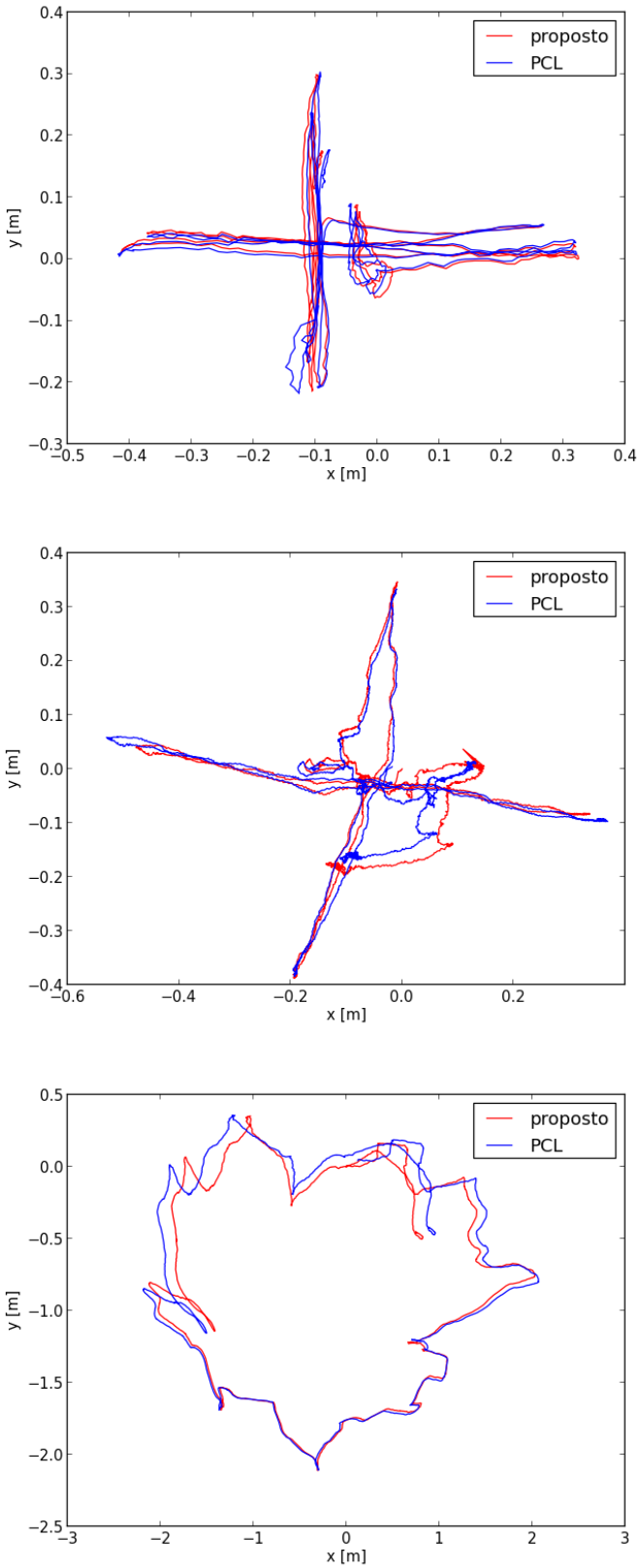

Figura 3.2: Gráfico comparativo entre as trajetórias estimadas com a PCL e com nossa implementação para as sequências 1, 2 e 3 respectivamente (de cima pra baixo).

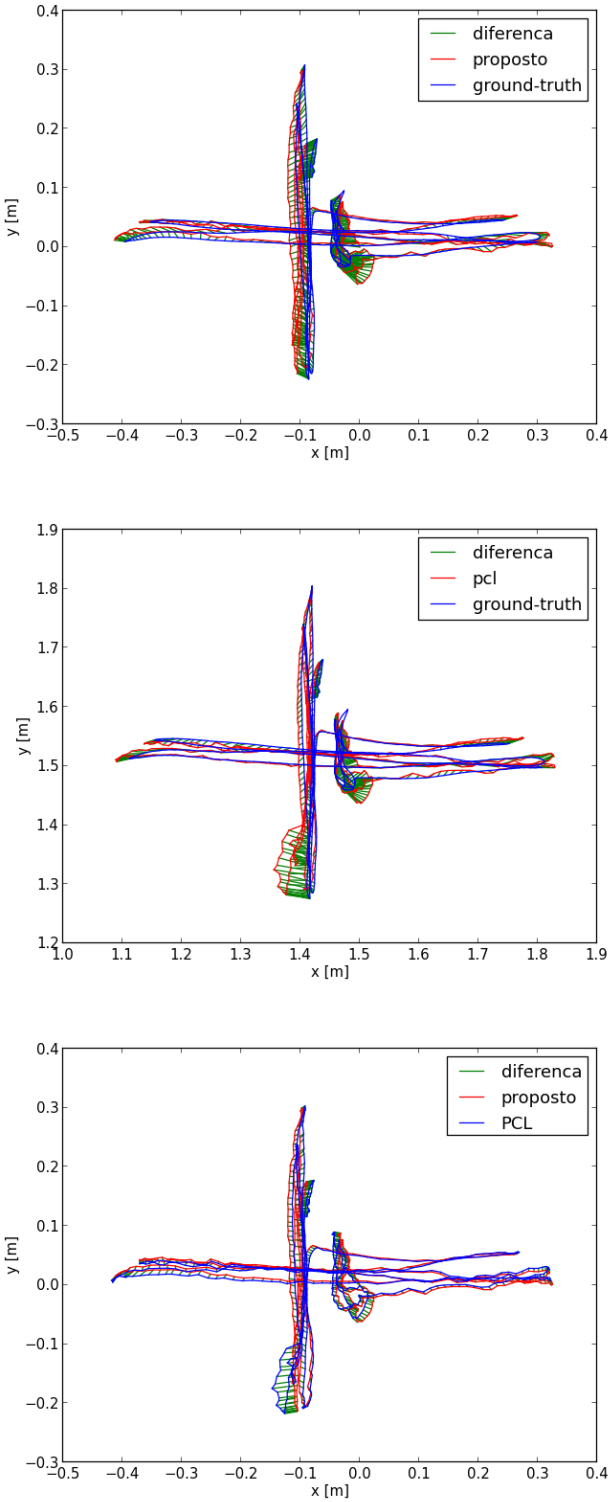

Figura 3.3: Posições x,y da trajetórias estimadas para a sequência 1 e o ETA estimado após o alinhamento para cada uma delas representado em verde. topo) Proposto vs ground-truth, meio) PCL vs ground-truth, baixo) proposto vs PCL.

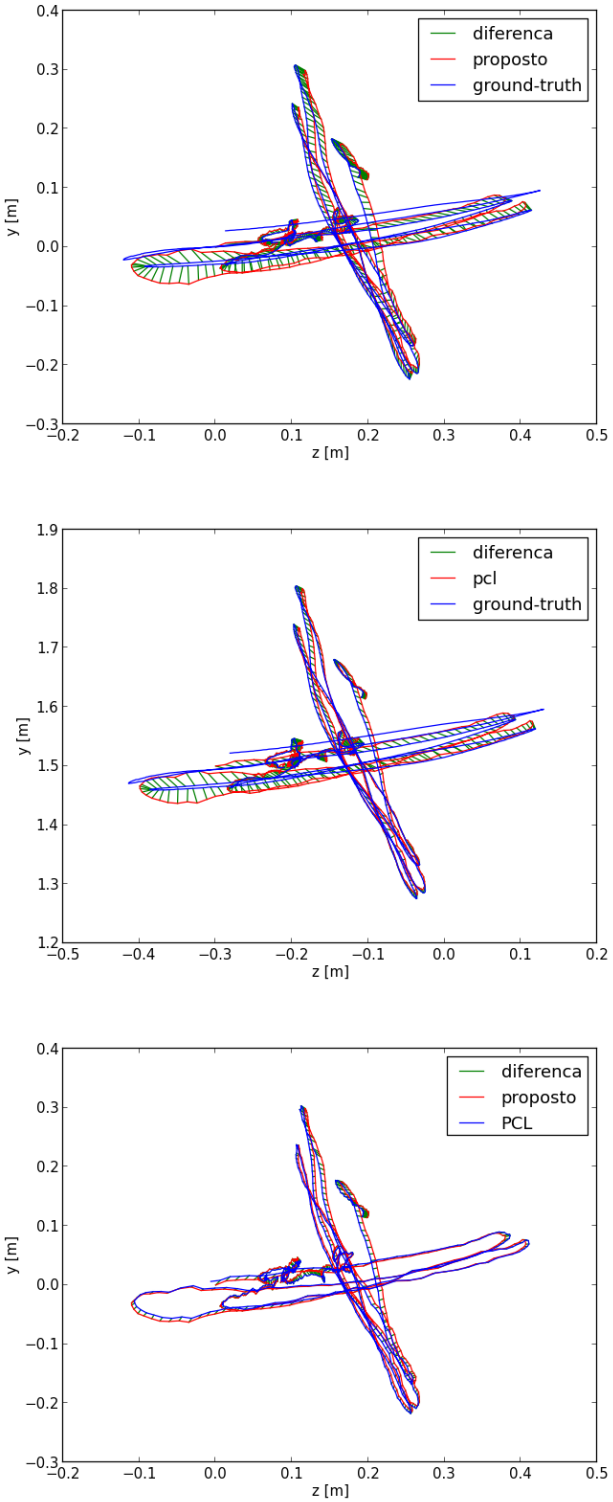

Figura 3.4: Posições z,y da trajetórias estimadas para a sequência 1 e o ETA estimado após o alinhamento para cada uma delas representado em verde. topo) Proposto vs ground-truth, meio) PCL vs ground-truth, baixo) proposto vs PCL.

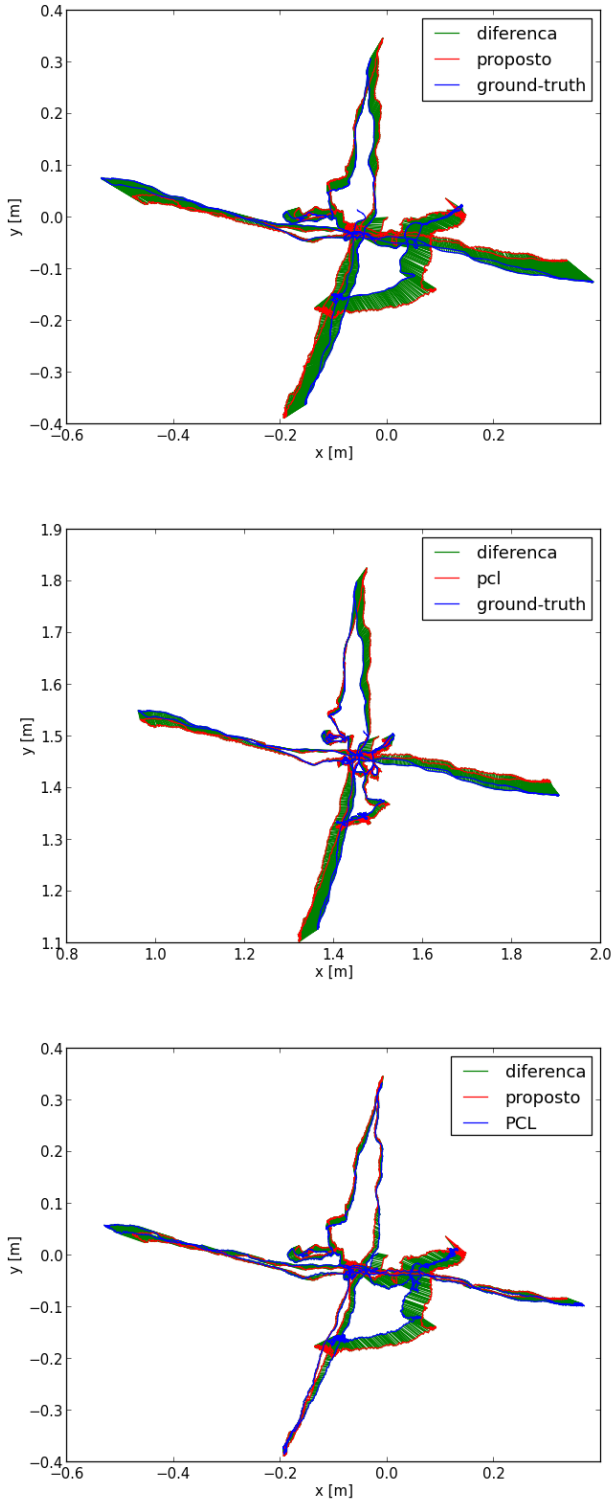

Figura 3.5: Posições x,y da trajetórias estimadas para a sequência 2 e o ETA estimado após o alinhamento para cada uma delas representado em verde. topo) Proposto vs ground-truth, meio) PCL vs ground-truth, baixo) proposto vs PCL.

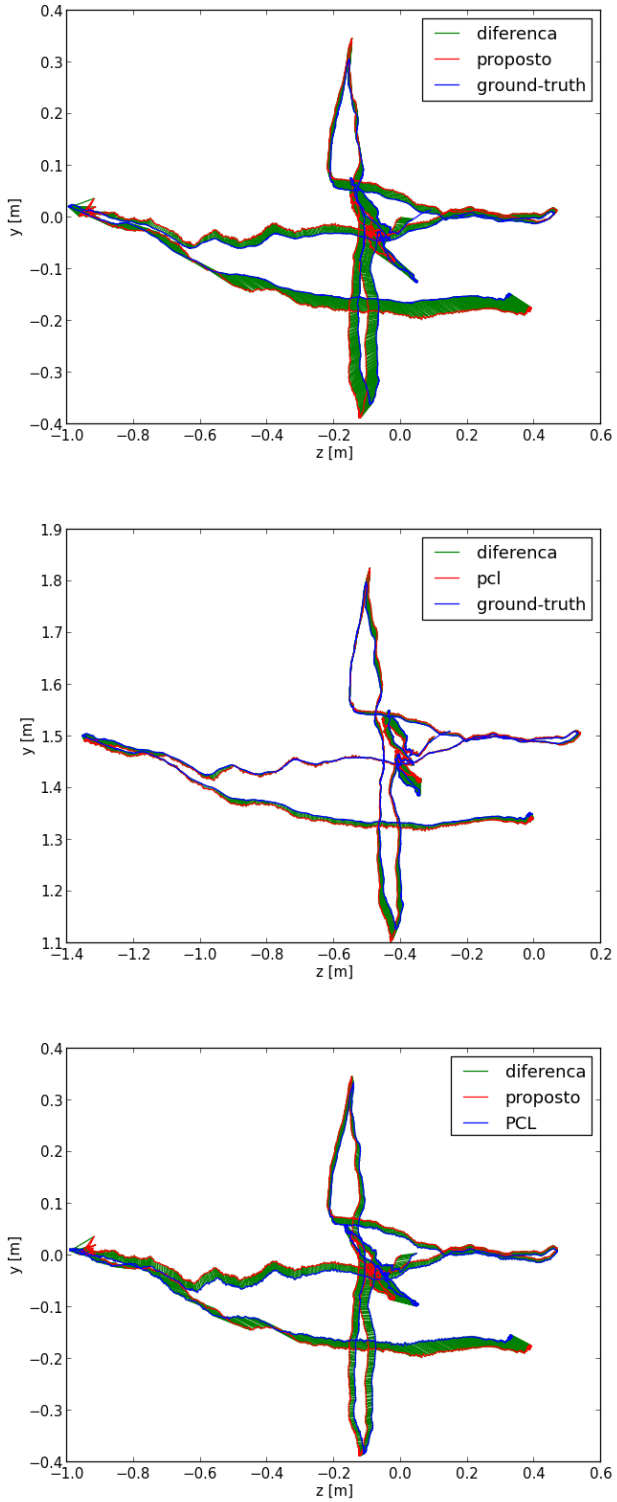

Figura 3.6: Posições z,y da trajetórias estimadas para a sequência 2 e o ETA estimado após o alinhamento para cada uma delas representado em verde. topo) Proposto vs ground-truth, meio) PCL vs ground-truth, baixo) proposto vs PCL.

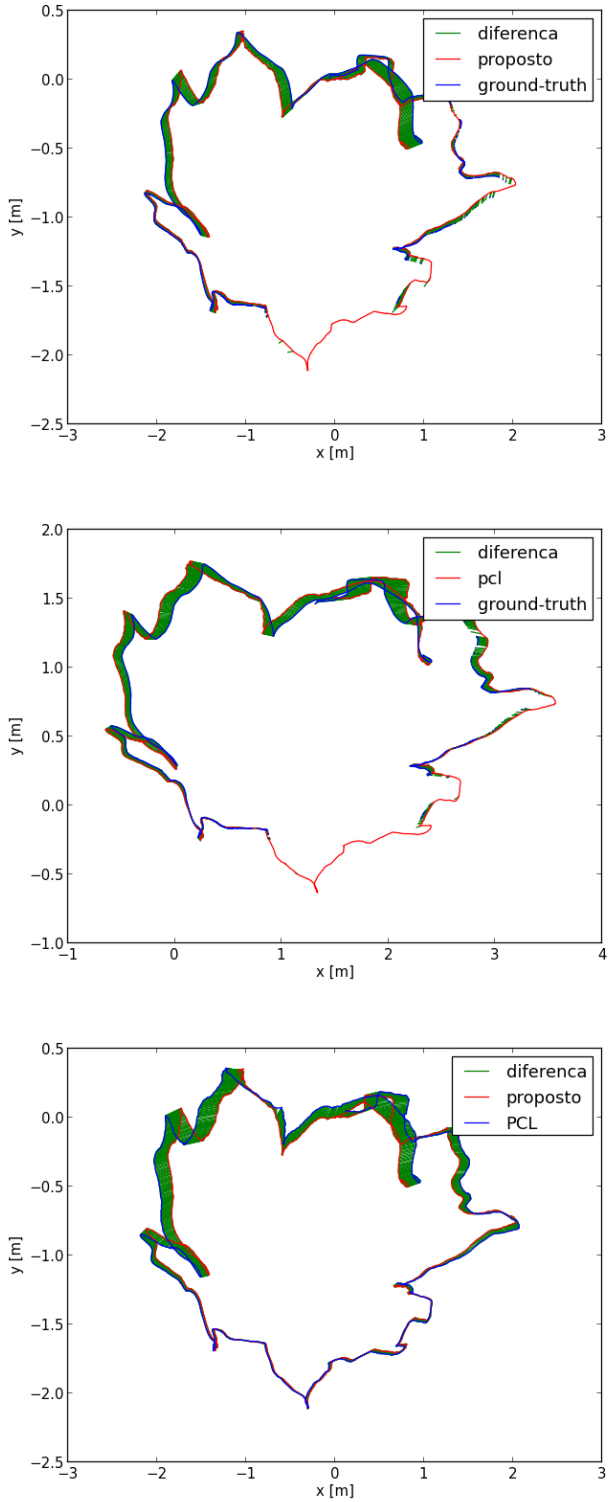

Figura 3.7: Posições x,y da trajetórias estimadas para a sequência 3 e o ETA estimado após o alinhamento para cada uma delas representado em verde. topo) Proposto vs ground-truth, meio) PCL vs ground-truth, baixo) proposto vs PCL.

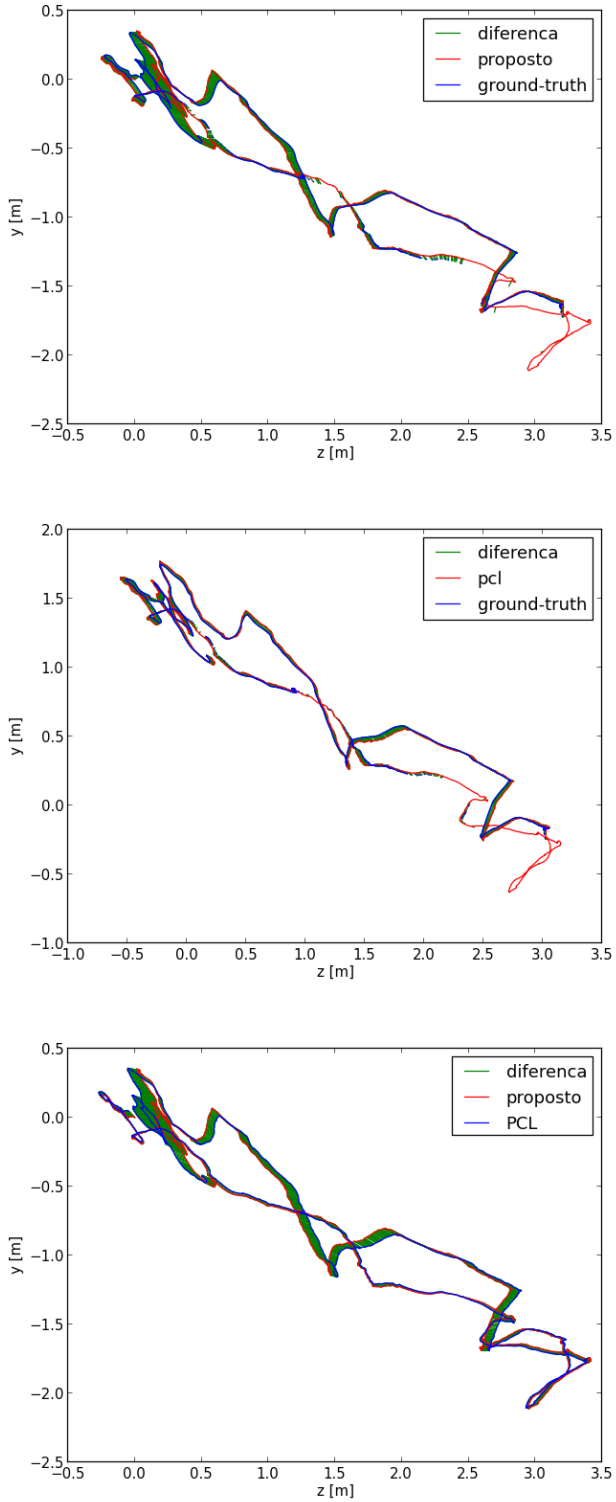

Figura 3.8: Posições z,y da trajetórias estimadas para a sequência 3 e o ETA estimado após o alinhamento para cada uma delas representado em verde. topo) Proposto vs ground-truth, meio) PCL vs ground-truth, baixo) proposto vs PCL.

espaçamentos entre cada uma das medidas.

Analisando os gráficos "Proposto vs PCL" das figuras (3.3) e (3.4), o ETA frame a frame entre a implementação proposta e a PCL é pequeno. Isso ocorre porque todas as simplificações que fizemos em relação ao algoritmo original em busca de melhor desempenho não chegam a ser relevantes para sequências tão pequenas, gerando ETAs semelhantes em relação ao ground-truth nos gráficos "Proposto vs ground-truth" e "PCL vs ground-truth".

As figuras (3.5) e (3.6) representam a trajetória da sequência 2. Esta sequência possui as mesmas características da sequência 1, porém, possui muito mais frames de entrada, o que faz com que as simplificações propostas tenham um impacto maior ao longo do tempo. Essa sequência possui uma diferença mais significativa entre a trajetória estimada pela PCL e pelo implementação proposta neste trabalho. Podemos citar alguns pontos que causam essa diferença de resultados.

Em primeiro lugar, o cálculo de normais da PCL utiliza um método aparentemente mais robusto, porém mais lento, para o cálculo das normais a partir do mapa de vértice. O método implementado por eles não é o mesmo descrito nos artigos originais do KinectFusion.

Além disso, as distâncias de cada voxel à superfície, calculadas durante a etapa de integração volumétrica, são feitas em relação ao centro do voxel, enquanto em nossa versão, a distância é calculada em relação ao canto inferior esquerdo de cada voxel. Como o modelo gerado durante o traçado de raios depende diretamente da interpolação tri-linear destes valores, tanto a posição dos vértices, quanto a normal estimadas a partir do volume são diferentes em ambas versões, gerando sistemas de equações diferentes durante a minimização de ponto pra plano e obtendo soluções diferentes.

Outro ponto importante é que durante o traçado de raios abrimos mão da interpolação tri-linear para calcular f_t e $f_{t+\Delta t}$. Fizemos isso para verificar qual o impacto em abrir mão de maior precisão na representação volumétrica em troca de um menor número de operações. Dessa forma diminuímos a precisão com que a posição dos vértices e direção das normais são calculadas em troca de maior desempenho.

Por último, a PCL implementa uma heurística que evita que o volume seja integrado caso a transformação entre dois frames consecutivos não seja maior do que um treshold. Como a sequência 2 possui muitas imagens geradas a partir de posições muito próximas, acreditamos que tal heurística evite que medidas ruidosas dentro de um mesmo voxel sejam adicionadas ao volume, melhorando assim a qualidade do modelo utilizado durante o ICP.

Vale destacar que não acreditamos que a alteração feita no sistema de

rastreamento da câmera em relação ao KinectFusion, onde evitamos transformações repetidas dos vértices durante a associação de pontos projetiva, seja responsável pela diferença em relação à trajetória estimada com a PCL. Em nossa implementação, após o modelo ser gerado com o traçado de raios a partir da posição da câmera estimada sobre o volume, aplicamos a transformação inversa calculada pelo ICP à tal modelo, levando-o para o sistema local da câmera, como descrito na seção (2.4.1). Tal transformação não é feita no algoritmo proposto em [2]. Ao contrário, o modelo é armazenado em coordenadas globais e é preciso levar os vértices para coordenadas locais em cada uma das iterações do ICP durante o estágio de associação de pontos. Fundamentalmente, não modificamos o algoritmo, apenas retiramos a ambiguidade de aplicar a mesma transformação múltiplas vezes para cada um dos vértices. Essa modificação tem um impacto significante no desempenho, como mostramos na seção (3.2).

As figuras (3.7) e (3.8) foram geradas a partir da sequência 3, que representa a sequência mais complexa dentre as três, onde há forte influência de rotações sobre a estimativa das poses. Analisando a diferença absoluta entre as trajetórias, a PCL possui uma ligeira vantagem na precisão em relação ao algoritmo proposto, mas menos significativa do que os resultados para a sequência 2. Tal afirmação é confirmada quando analisamos os resultados numéricos da tabela (3.1), que possuem resultados bastante similares. Acreditamos que a diferença seja menos perceptiva nesta sequência porque o impacto da heurística que evita que o volume seja reintegrado caso o movimento entre dois frames seja pequeno é menor, já que nela o Kinect foi movido de forma mais rápida durante a gravação, assim como na sequência 1, que gera resultados semelhantes. Tal diferença de velocidade pode ser inspecionada visualmente assistindo-se os vídeos das sequências disponíveis no site dos autores do data-set.

	Sequência 1		Sequência 2		Sequência 3		media		
	Proposto	PCL	Proposto	PCL	Proposto	PCL	Proposto	PCL	%
RQME	0.02	0.02	0.05	0.02	0.10	0.09	0.06	0.04	130.30
Média	0.02	0.02	0.05	0.02	0.08	0.08	0.05	0.04	134.62
Mediana	0.02	0.01	0.04	0.01	0.06	0.06	0.04	0.03	141.20
Desvio Padrão	0.01	0.01	0.02	0.01	0.05	0.05	0.03	0.02	117.98
Erro Mínimo	0.00	0.00	0.01	0.00	0.01	0.01	0.01	0.00	198.73
Erro Máximo	0.05	0.06	0.10	0.05	0.20	0.26	0.11	0.12	93.57

Tabela 3.1: Valores numéricos dos ETAs obtidos para cada uma das sequências analisadas, seguido da média entre todas elas. O campo % representa a porcentagem de erro da média do algoritmo proposto em relação à PCL.

Analisando a tabela (3.1), percebemos que na média, a RQME de nossa

versão obteve um valor 30% maior do que a da PCL. Esse erro pode ser evitado se utilizarmos uma heurística para evitar que movimentações muito pequenas sejam acumuladas na posição global da câmera, evitando a propagação de erro entre quadros intermediários. Vale ressaltar que apesar da diferença em porcentagem ser significativa, a diferença absoluta entre os valores de ambas implementações é em media de 2cms. Diante da natureza ruidosa do mapa de profundidade de entrada, essa é uma diferença bastante aceitável.

3.1.2 Erro de Posição Relativo (EPR)

O erro de posição relativo mede a precisão da trajetória estimada dentro de um intervalo de tempo fixo Δ. Levando-se em conta apenas a translação, por exemplo, podemos entender o EPR da seguinte forma: Sejam as poses $Q_1, ..., Q_n$ e $P_1, ..., P_n \in \mathbb{SE}^3$, posições que definam uma trajetória de referência e outra estimada, respectivamente. Sabendo que a translação da pose Q_i para a pose $Q_{i+\Delta}$ possui uma magnitude x, e uma magnitude de $x + \sigma$ entre as poses P_i e $P_{i+\Delta}$, podemos afirmar que o erro de posição relativa é equivalente a σ.

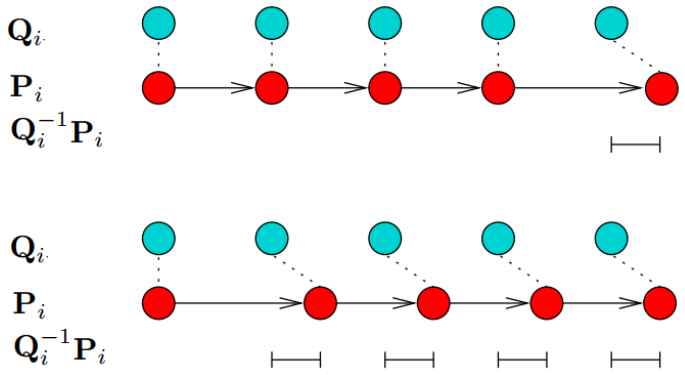

Figura 3.9: Ilustração de um caso bem simples que demonstra a influência de um possível desvio ao longo da trajetória na estimativa de erro. [32]

Ao contrário do ETA, onde analisamos a diferença da posição Q_i para P_i, o EPR analisa a diferença entre a magnitude de movimento de Q_i para $Q_{i+\Delta}$ contra a magnitude de movimento de P_i para $P_{i+\Delta}$. Dessa forma, evitamos que um possível desvio na trajetória estimada comprometa as estimativas de erro de poses subsequentes, como mostra a figura (3.9). Os círculos azuis mostram poses de referência Q_i, enquanto os círculos vermelhos mostram as poses estimadas pelo sistema P_i. As correspondências entre posições estimadas e o ground-truth são exibidas em linhas pontilhadas, e a direção do movimento é destacada pelas setas. No caso demonstrado na parte de cima, o sistema calcula a pose com um pequeno engano no final do caminho, produzindo assim,

um pequeno erro global. Reciprocamente, na situação ilustrada na parte de baixo da figura, o sistema produz um pequeno erro de mesmo tamanho, mas no começo da trajetória, o que causa um erro global muito maior.

As figuras (3.10), (3.11) e (3.12) exibem gráficos comparativos dos EPRs entre a PCL (em azul) e a implementação proposta (vermelho). Os gráficos foram gerados utilizando os scripts disponibilizados pelos autores do data-set e comparamos trajetórias com $\Delta = 1$.

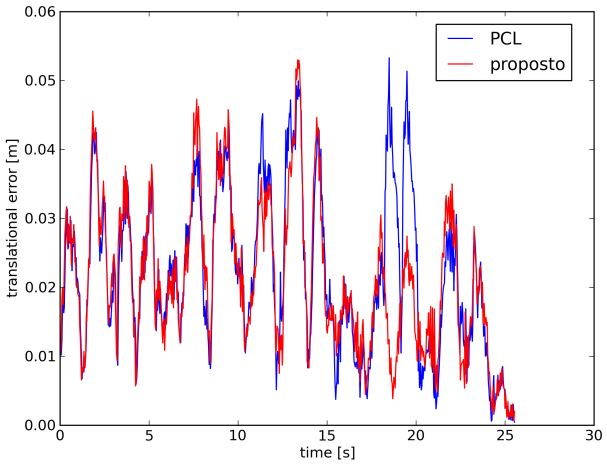

Figura 3.10: Comparativo do erro de posição relativo (EPR) ao longo do tempo entre a implementação proposta e da PCL para a sequência 1.

Analisando a figura (3.10), observamos um comportamente similar entre ambas implementações, inclusive com um pequeno trexo entre 15 e 20 segundos onde nossa versão possui um erro menor que a PCL. Na figura (3.11), notamos um erro máximo maior por volta dos 100 segundos, porém, notamos mais uma vez um comportamento próximo entre ambas implementações durante a maior parte do trajeto. Na figura (3.12), mais uma vez, os resultados são bem parecidos. Esse comportamento é reproduzido nos valores numéricos das tabelas (3.2) e (3.3).

Analisando os valores numéricos do RPE de cada sequência, listados nas tabelas (3.2) e (3.3), é possível perceber uma grande diferença em relação aos valores obtidos para os ETAs. Na média, a diferença entre os RQMEs do erro translacional foi pouco maior do que 4%, enquanto que para o erro rotacional tivemos um valor aproximadamente 10% inferior. Acreditamos que um dos principais fatores para essa diferença é que para calcular o ETA, precisamos que as duas trajetórias sendo comparadas estejam alinhadas. Como não conhecemos o sistema de coordenadas em que foi estimada a trajetória de referência, tanto as trajetórias geradas pela nossa implementação, quanto da PCL, são alinhadas à referência utilizando um método numérico. Sendo assim,

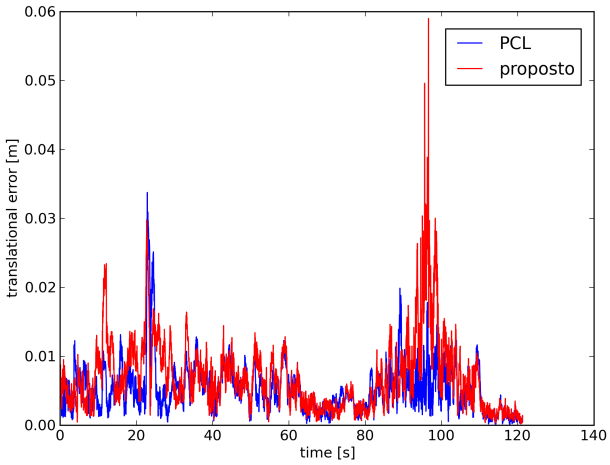

Figura 3.11: Comparativo do erro de posição relativo (EPR) ao longo do tempo entre a implementação proposta e da PCL para a sequência 2.

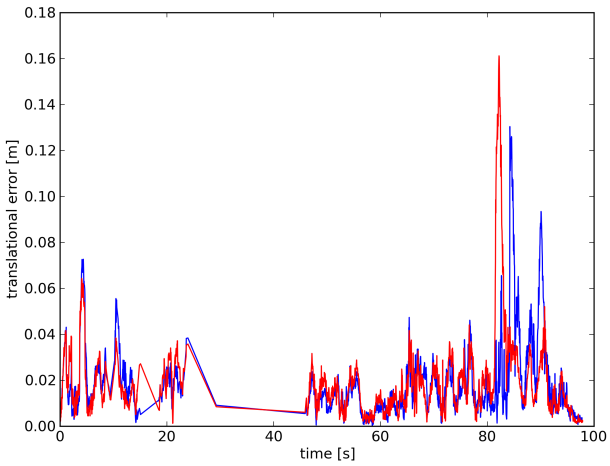

Figura 3.12: Comparativo do erro de posição relativo (EPR) ao longo do tempo entre a implementação proposta e da PCL para a sequência 3.

	Sequência 1		Sequência 2		Sequência 3		Média		
	PCL	Proposto	PCL	Proposto	PCL	Proposto	PCL	Proposto	%
RQME	0.03	0.02	0.01	0.01	0.03	0.03	0.02	0.02	104.53
Média	0.02	0.02	0.01	0.01	0.02	0.02	0.02	0.02	103.01
Mediana	0.02	0.02	0.00	0.01	0.01	0.02	0.01	0.01	104.50
Desvio Padrão	0.01	0.01	0.00	0.01	0.02	0.02	0.01	0.01	106.86
Erro Mínimo	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	212.78
Erro Máximo	0.05	0.05	0.03	0.06	0.13	0.16	0.07	0.09	125.62

Tabela 3.2: Valores numéricos dos erros de translação relativos (em metros) obtidos para cada uma das sequências analisadas seguido da média entre todas elas. O campo % representa a porcentagem de erro da média do algoritmo proposto em relação à PCL.

	Sequência 1		Sequência 2		Sequência 3		Média		
	PCL	Proposto	PCL	Proposto	PCL	Proposto	PCL	Proposto	%
RQME	1.55	1.57	0.39	0.46	1.22	0.85	1.05	0.96	90.84
Média	1.37	1.40	0.35	0.40	0.92	0.76	0.88	0.85	96.78
Mediana	0.02	0.02	0.01	0.01	0.01	0.01	0.01	0.01	101.14
Desvio Padrão	0.71	0.71	0.19	0.23	0.80	0.38	0.57	0.44	76.97
Erro Mínimo	0.10	0.11	0.02	0.02	0.07	0.04	0.06	0.06	91.03
Erro Máximo	3.70	4.03	1.53	1.85	6.13	2.22	3.79	2.70	71.33

Tabela 3.3: Valores numéricos dos erros de rotação (em graus) relativos obtidos para cada uma das sequências analisadas seguido da média entre todas elas. O campo % representa a porcentagem de erro da média do algoritmo proposto em relação à PCL.

durante o cálculo do ETA, não só o erro da diferença entre as posições da trajetória são acumulados, mas também os erros do método numérico envolvido no alinhamento. Como o RPE não necessita que seja feito o alinhamento entre as trajetórias, esse erro não impacta no resultado final. Além disso, no cálculo do ETA, erros ao longo da trajetória acabam impactando o resultado de posições posteriores, como mostra a figura (3.9), aumentando o erro total global de forma mais drástica que no RPE.

3.2 Análise de desempenho

Todos os testes foram realizados em uma máquina de configuração mediana para os padrões atuais. O nosso objetivo era demonstrar a viabilidade financeira e técnica do algoritmo proposto. Os componentes principais do equipamento utilizado são:

- Processador: Intel i7 Quad-Core 920 2.4Ghz
- Placa Mãe: Asus P6T Deluxe V2
- Memória: 3x2Gb a 1333 Mhz (Triple Channel)
- Placa de Vídeo: Geforce GTX 460 1GB SE

A figura (3.13) exibe os gráficos das tomadas de tempo frame a frame para cada uma das sequências avaliadas.

Analisando a figura (3.13), nossa implementação é consistentemente mais rápida do que a da PCL em todos os frames. Vale notar que os gráficos das sequências 1 e 2 possuem medidas de tempo relativamente constantes, enquanto na sequência 3 há uma variação maior entre as estimativas. Apesar de teoricamente o tempo de execução do algoritmo ser sempre constante, já que sempre temos uma matriz 640 x 480 de entrada e processamos um volume

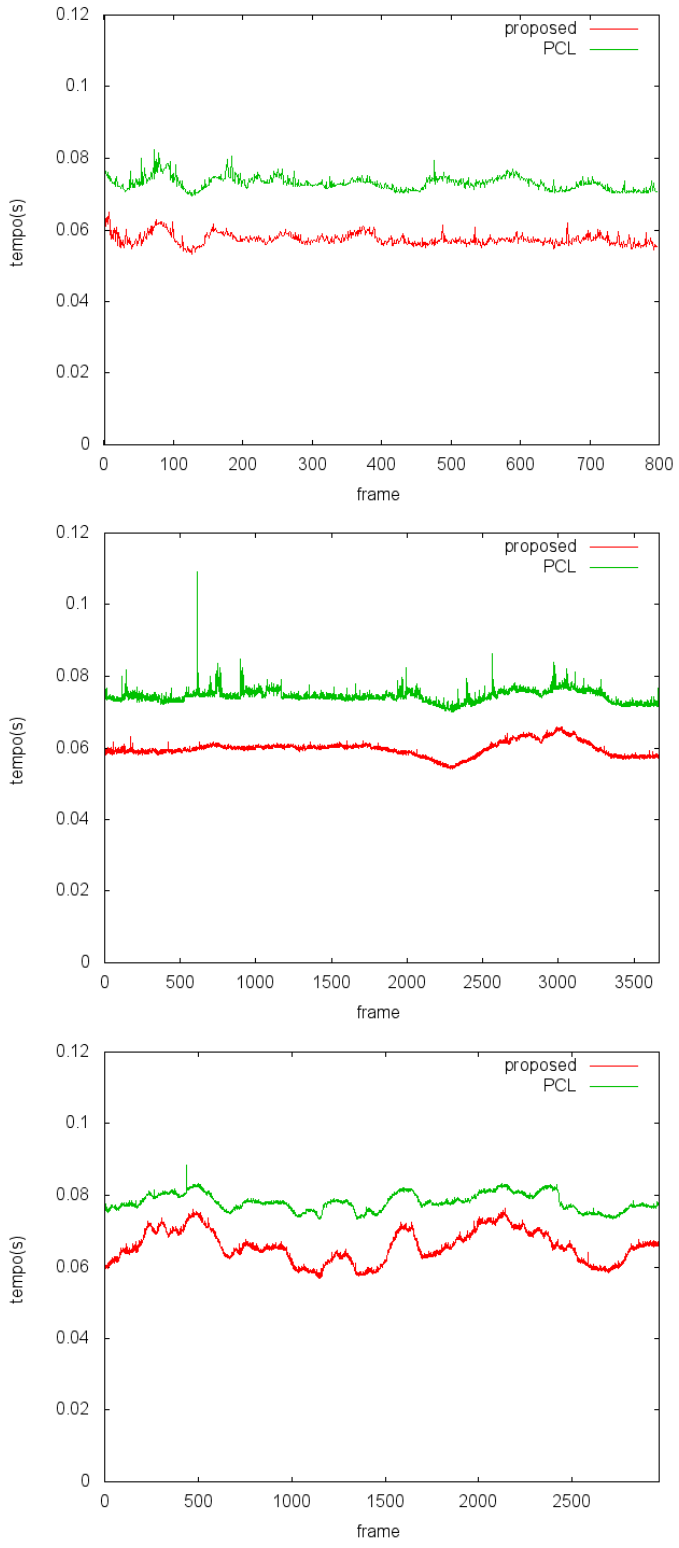

Figura 3.13: Gráficos de tempo de execução de toda a pipeline de reconstrução para cada frame de entrada das sequências 1, 2 e 3 respectivamente (de cima pra baixo).

de resolução de 512 x 512 x 512 em todos os frames, na prática há sempre uma variação no tempo estimado, tanto pelos atrasos na execução do código pelo hardware, quanto pelos condicionais que causam fluxos alternativos no código, como:

- Pixels inválidos, onde o Kinect não retornou uma medida de profundidade válida, são ignorados durante toda a pipeline.
- Normais inválidas, que podem surgir quando um pixel não possui vizinhos válidos ou por não ser possível estimá-la durante o traçado de raios, são ignoradas.
- Durante o traçado de raios, percorremos o raio em saltos constantes. Isso faz com que várias posições ao longo do raio caiam fora da área definida pelo volume de distâncias. Essas posições são ignoradas.

Dessa forma, a quantidade de pixels inválidos e a posição em que se inicia o traçado de raios são os dois principais fatores que impactam no tempo de execução do código.

Na tabela (3.4) mostramos as médias obtidas entre a sequência e a média de tempo entre todas as sequências. Analisando-a percebemos que nossa implementação possui um tempo de execução, em média, 23.38% menor do que da PCL

	Sequência 1	Sequência 2	Sequência 3
Proposto	0.05740s	0.05973s	0.06567s
PCL	0.07306s	0.07438s	0.07811s

Tabela 3.4: Tempo(em segundos) de execução médio das sequências 1, 2 e 3, respectivamente, seguido da média de tempo entre todas as sequências.

Para compreendermos o impacto de cada uma das etapas do algoritmo no tempo de processamento e identificar os possíveis gargalos, foi feita uma análise de tempo de cada uma das etapas da pipeline. Os resultados obtidos são mostrados na figura (3.14) e na tabela (3.5).

Da análise da figura (3.14) percebe-se que os maiores gargalos do sistema são as três principais etapas da pipeline: ICP, integração volumétrica e traçado de raios. Mais detalhes podem ser vistos na figura (3.15), onde são exibidas as cargas de trabalho das etapas do algoritmo.

Tradicionalmente o filtro bilateral exige o cálculo de duas exponenciais para cada um dos pixels dentro da janela pré-definida ao redor do pixel sendo suavizado. Uma exponencial é calculada para a distância entre os pixels, enquanto outra leva em conta a diferença de intensidade entre eles. Como o

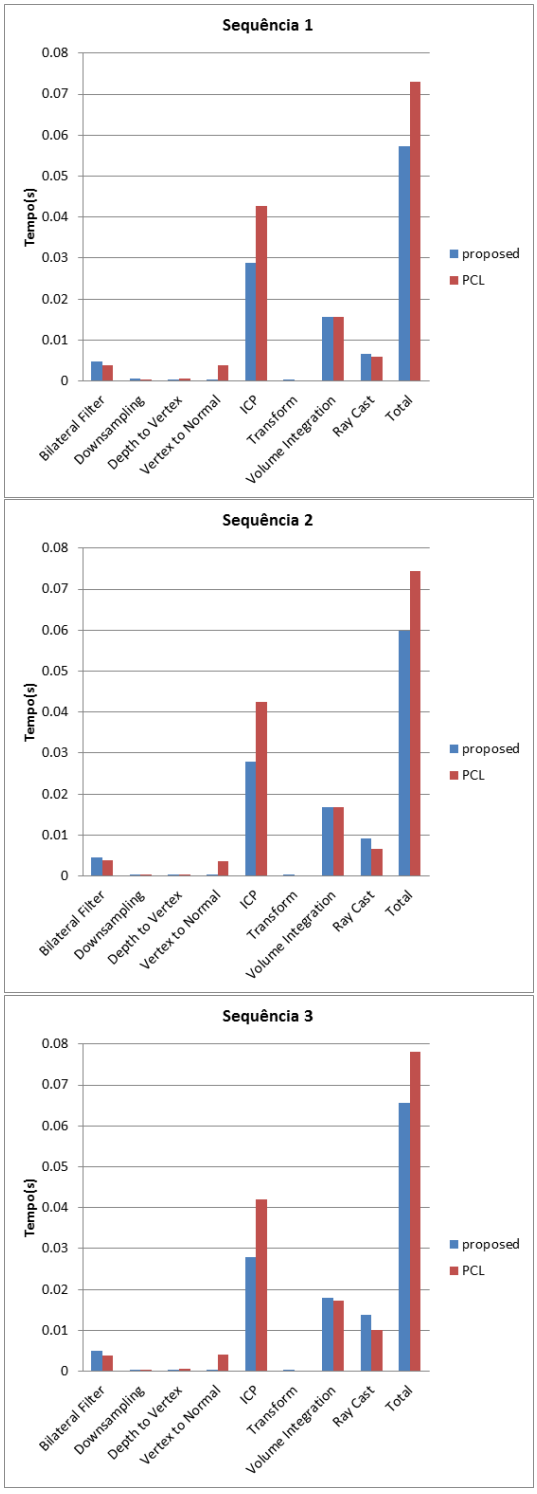

Figura 3.14: Gráfico de barras exibindo o tempo médio de cada uma das etapas da pipeline de reconstrução para cada uma das sequências.

	Sequência 1		Sequência 2		Sequência 3		Média	
	Proposto	PCL	Proposto	PCL	Proposto	PCL	Proposto	PCL
Bilateral Filter	0.0047	0.0038	0.0045	0.0038	0.0049	0.0038	0.0047	0.0038
Downsampling	0.0006	0.0005	0.0004	0.0004	0.0004	0.0003	0.0005	0.0004
Depth to Vertex	0.0003	0.0006	0.0003	0.0005	0.0003	0.0005	0.0003	0.0005
Vertex to Normal	0.0004	0.0039	0.0003	0.0037	0.0003	0.0041	0.0003	0.0039
ICP	0.0288	0.0427	0.0278	0.0425	0.0278	0.0421	0.0281	0.0425
Transform	0.0004	0.0000	0.0003	0.0000	0.0003	0.0000	0.0003	0.0000
Volume Integration	0.0156	0.0156	0.0169	0.0168	0.0179	0.0172	0.0168	0.0165
Ray Cast	0.0067	0.0060	0.0092	0.0066	0.0137	0.0100	0.0099	0.0076
Total	0.0574	0.0731	0.0597	0.0744	0.0657	0.0781	0.0609	0.0752

Tabela 3.5: Tempo de execução(em segundos) de cada uma das etapas da pipeline de reconstrução.

cálculo de uma exponencial é custoso e a janela utilizada é pequena e constante, nossa estratégia para acelerar esta etapa foi utilizar um vetor que armazena os valores das exponenciais para todas as possíveis distâncias, trocando assim o cálculo de uma das exponenciais por um acesso à memória. Já a PCL realiza uma simplificação no filtro. Utilizando as definições da seção (2.3.2), a fórmula implementada por ela pode ser escrita da seguinte forma:

$$space = (x - c_x)^2 + (y - c_y)^2 \tag{3-1}$$

$$color = (D_{yx} - D_{c_y c_x})^2 \tag{3-2}$$

$$w = \exp(-(space\frac{1}{2\sigma_s^2} + color\frac{1}{2\sigma_r^2})) \tag{3-3}$$

Como vemos na tabela (3.5), esta simplificação resulta em um ganho de desempenho de alguns milissegundos em relação à nossa versão.

O campo Downsample da tabela (3.5) e da figura (3.14) refere-se à etapa de criação dos níveis da pirâmide de simplificação, e, junto com as etapas de criação do mapa de vértices representam uma porcentagem desprezível no tempo total do algoritmo.

Há uma diferença significativa no tempo para se calcular o mapa de normais entre ambas implementações. Nós implementamos o algoritmo exatamente como proposto em [2], enquanto a PCL utiliza uma forma mais pesada, onde vários vértices vizinhos são levados em consideração para o cálculo de cada uma das normais. Isso resulta em uma diferença de mais de 10 vezes no tempo de execução, mas, avaliando os resultados da seção (3.1.1), aparentemente é um custo que é compensado em forma de maior precisão na estimativa da posição da câmera.

A etapa denominada de Transform equivale ao passo em que o modelo estimado durante o traçado de raios é transformado para coordenadas locais aplicando-se a inversa da transformação calculada durante o ICP, como explicado na seção (2.4.1) . Como a PCL não utiliza essa técnica, para ela, este campo é sempre igual a zero. Podemos observar que o impacto de retirar essa etapa da associação de vértices, realizando-o apenas uma vez em cada iteração, resulta em um ganho de desempenho, em média, de aproximadamente 50% durante o ICP e, por ser o maior gargalo do sistema, é o responsável pelo ganho de desempenho total da nossa versão.

Por último, percebe-se que a PCL consegue melhores resultados durante a integração volumétrica e o traçado de raios. Analisando ambas as implementações, as principais diferenças vêm da forma como os índices são calculados para cada voxel. Além disso, há também diferenças na interpolação tri-linear e na aproximação que fizemos para calcular os valores de f_t e $f_{t+\Delta t}$ durante o traçado de raios. Esperávamos que esta última diferença resultasse em um maior desempenho para a nossa versão, mas os valores obtidos não foram como esperado.

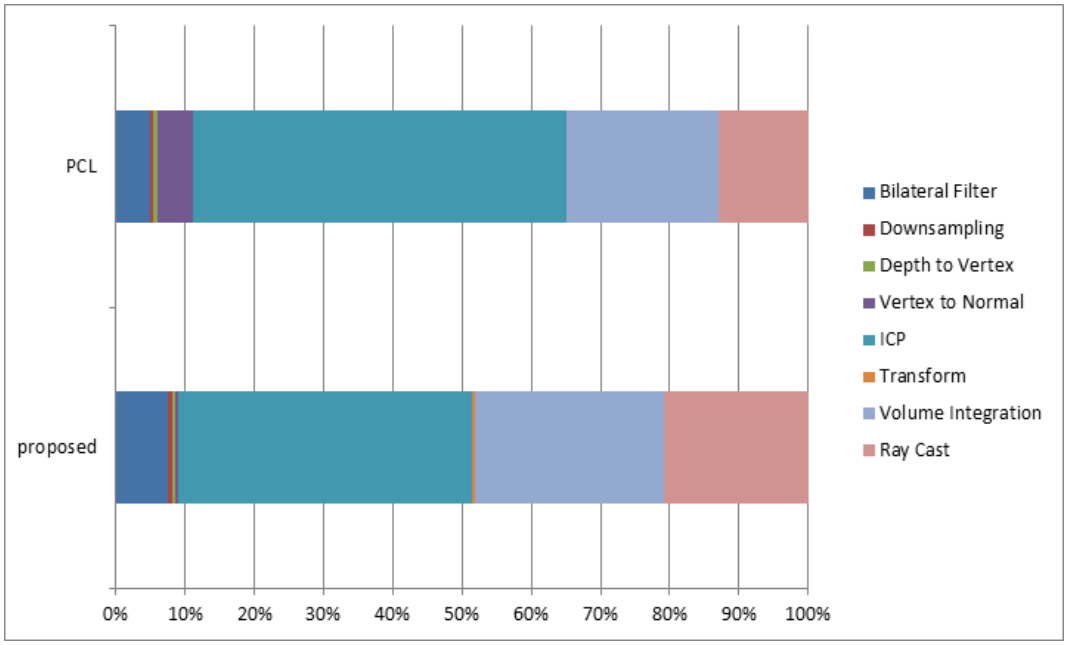

Figura 3.15: Gráfico de barras indicando a carga de trabalho de cada uma das etapas da pipeline, levando-se em conta a média de todas as sequências.

	Proposto	PCL
Bilateral Filter	7.74	6.28
Downsampling	0.76	0.66
Depth to Vertex	0.48	0.85
Vertex to Normal	0.53	6.35
ICP	46.16	69.69
Transform	0.55	0.00
Volume Integration	27.52	27.13
Ray Cast	16.25	12.40

Tabela 3.6: Porcentagens que cada uma das etapas da pipeline ocupam no tempo total do algoritmo, levando-se em conta a média de todas as sequências.

4
Conclusão e Trabalhos Futuros

Foi apresentado aqui um estudo técnico de um algoritmo de rastreamento e reconstrução densa 3D, em tempo real, chamado KinectFusion, como apresentado nos trabalhos publicados por Newcomb et. Al. (2011) e Izadi et. Al. (2011). Através do método estudado, foi implementado um sistema SLAM 3D, que é capaz de realizar o rastreamento da câmera a partir do alinhamento de múltiplos mapas de profundidade. Esses mapas de profundidade são fundidos em uma representação volumétrica global, que integra todos os pontos 3D do ambiente sendo observado por um sensor de profundidade e representa a cena de forma densa e fiel. Utilizar o modelo de superfície derivado a partir deste volume durante o estágio de rastreamento, junto com a nossa implementação em CUDA do ICP que permite que toda a informação do mapa de profundidade seja utilizada para realizar a atualização da posição global da câmera, resulta em um sistema de rastreamento robusto, que funciona nos mais variados ambientes, pois não há necessidade de realizar a detecção de quaisquer características esparsas ou de utilizar marcadores para atualizar a posição do sensor. Como não utilizamos informações fotométricas, variações na iluminação não afetam na qualidade do rastreamento ou da reconstrução, porém, ambientes sem características geométricas distintas representam um desafio, pois não são capazes de restringir a função objetivo utilizada durante o ICP para estimar a posição da câmera. Acreditamos que os algoritmos de SLAM que serão mais bem sucedidos, vão integrar tanto as cores extraídas da imagem RGB, quanto características geométricas extraídas do mapa de profundidade para criar um sistema de localização que compense a falta de informações fotométricas por informações geométricas, e vice-versa.

Realizamos um estudo comparativo quantitativo, analisando os resultados obtidos por nossa implementação contra os resultados obtidos com uma implementação do mesmo algoritmo, disponibilizado pela PCL durante a execução deste trabalho.

Quando ignorando os outros estágios da pipeline, a nossa implementação do ICP representou um ganho de tempo de aproximadamente 50% em relação à versão da PCL. Acreditamos que isso tenha ocorrido porque em nossa implementação nos livramos de uma operação de custo elevado durante o estágio de associação de pontos, como descrevemos na sessão (2.4.1). Esta operação seria aplicada em cada um dos pixels, em cada iteração do ICP.

Toda a nossa pipeline teve uma execução 23% mais rápida do que a versão

da PCL. Como trabalhos futuros queremos investigar a fundo os motivos da integração volumétrica e traçado de raios terem um desempenho inferior à PCL.

Para os testes de precisão quantitativos, foram utilizadas duas métricas de comparação: o ETA e o EPR. Ambas são descritas nas sessões (3.1.1) e (3.1.2), respectivamente. Analisando a raiz quadrada da média dos erros (RQME), métrica de comparação proposta por [22], observamos que nossa versão teve um ETA 30% superior à PCL, porém, tivemos um EPR muito similar, apresentando um erro translacional em média 4% superior, e um erro rotacional em média 10% inferior. Acreditamos que a grande diferença obtida dos dois resultados acontece pois para o ETA, é preciso que as trajetórias estejam no mesmo sistema de coordenadas. Como não sabemos exatamente qual a posição do sensor que realiza o rastreamento que gera o groundtruth, é necessário utilizar um método numérico de decomposição singular para alinhar a trajetória gerada pelo nosso sistema e o groundtruth. O mesmo deve ser feito para a trajetória da PCL. Acreditamos que os erros gerados pelo método numérico elevem os valores de ETA estimados. Como o EPR não leva em consideração a posição absoluta de cada pose observada, mas sim a diferença do tamanho das trajetórias percorridas de uma pose para outra, não há necessidade de alinhamento das trajetórias, gerando resultados até superiores para o RQME de erro rotacional. Sendo assim, concluímos que nossa implementação é mais rápida que a versão da PCL, sem comprometer significantemente na qualidade do sistema de rastreamento.

Acreditamos que este trabalho deixe aberto uma série de possibilidades para trabalhos futuros.

Até o momento, nós apenas extraímos o modelo de pontos de vista específicos através do ray-tracing. Futuramente pretendemos explorar o Marching Cubes [33] como alternativa de rendering, pois ele é capaz gerar uma malha de triângulos global que poderia ser utilizada para outras finalidades, como criação de avatares virtuais 3D.

Além disso, podemos explorar o volume da cena para criar soluções interativas de realidade aumentada e multi-toque, como em [2]. Através do modelo extraído a partir da representação volumétrica, temos uma informação geométrica que permite que componentes virtuais interajam com a cena e vice-versa. Também é possível utilizar o campo de distâncias para realizar uma segmentação de plano de fundo mais robusta do que técnicas baseadas apenas em informações de cor, permitindo a criação de superfícies multi-toque em qualquer objeto da cena.

Outro ponto que pode ser atacado é o fato de que a representação

do volume da cena em um grid regular apresenta um grande cosumo de memória, limitando o tamanho espaço que pode ser representado pelo sistema. Acreditamos que utilizar uma estrutura espacial dinâmica, como uma octree, permitiria um melhor consumo de memória na GPU e poderíamos representar um ambiente mais amplo.

Por ultimo, gostaríamos de avaliar alternativas no sistema de rastreamento que utilizem também as informações fotométricas da imagem RGB, em conjunto com as informações geométricas do mapa de profundidade. Utilizar uma função de energia que utilize tanto as cores, quanto as coordenadas dos pontos da cena, pode tornar o sistema mais robusto em situações onde uma função exclusivamente geométrica ou fotométrica falharia, permitindo o mapeamento de longas distâncias sem que o sistema de rastreamento se perca.

5
Referências Bibliográficas

[1] A. S. Huang, A. Bachrach, P. Henry, M. Krainin, D. Maturana, D. Fox, and N. Roy, "Visual odometry and mapping for autonomous flight using an rgb-d camera," in *Int. Symposium on Robotics Research (ISRR)*, 2011. 1.1, 1.2, 1.3

[2] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, A. Davison, and A. Fitzgibbon, "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*, 2011. 1.1, 1.2, 1.3, 2.2, 2.4, 2.4.1, 3.1.1, 3.2, 4

[3] S. D. W. Daniel, "Artoolkitplus for pose tracking on mobile devices," in *Proceedings of 12th Computer Vision Winter Workshop (CVWW'07)*, 2007. (document), 1.2, 1.1

[4] G. Bleser, H. Wuest, and D. Stricker, "Online camera pose estimation in partially known and dynamic scenes," in *Proceedings of the 5th IEEE and ACM International Symposium on Mixed and Augmented Reality*, ser. ISMAR '06, 2006. 1.2

[5] G. Klein and D. Murray, "Parallel tracking and mapping for small AR workspaces," in *Proc. Sixth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'07)*, 2007. (document), 1.2, 1.2, 1.3

[6] A. J. Davison, "Real-time simultaneous localisation and mapping with a single camera," in *Proceedings of the Ninth IEEE International Conference on Computer Vision - Volume 2*, 2003. 1.2

[7] R. A. Newcombe and A. J. Davison, "Live dense reconstruction with a single moving camera," in *IEEE Conference on Computer Vision and pattern Recognition*, 2010. 1.2, 1.3

[8] R. Newcombe, S. Lovegrove, and A. Davison, "Dtam: Dense tracking and mapping in real-time," in *Proc. of the Intl. Conf. on Computer Vision (ICCV), Barcelona, Spain*, 2011. 1.2, 1.3

[9] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Internetional Journal of Computer Vision*, 2004. 1.2

[10] J. Shi and C. Tomasi, "Good features to track," 1994. 1.2

[11] H. Bay, A. Ess, T. Tuytelaars, and L. V. Gool, "Speeded-up robust features (surf)," *Computer Vision Image Understanding*, 2008. 1.2

[12] A. Geiger, M. Roser, and R. Urtasun, "Efficient large-scale stereo matching," in *Proceedings of the 10th Asian conference on Computer vision - Volume Part I*, 2011. 1.2, 1.3

[13] A. Geiger, J. Ziegler, and C. Stiller, "Stereoscan: Dense 3d reconstruction in real-time," in *IEEE Intelligent Vehicles Symposium*, 2011. 1.2, 1.3

[14] Y. Cui, S. Schuon, D. Chan, S. Thrun, and C. Theobalt, "3d shape scanning with a time-of-flight camera," in *The Twenty-Third IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2010, San Francisco, CA, USA, 13-18 June 2010*, 2010. 1.2, 1.3

[15] S. May, S. Fuchs, D. Droeschel, D. Holz, and A. Nüchter, "Robust 3d-mapping with time-of-flight cameras," in *Proceedings of the 2009 IEEE/RSJ international conference on Intelligent robots and systems*, 2009. 1.2, 1.3

[16] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox, "Rgbd mapping: Using depth cameras for dense 3d modeling of indoor environments," in *In RGB-D: Advanced Reasoning with Depth Cameras Workshop in conjunction with RSS*, 2010. 1.2, 1.3

[17] M. Paton and J. Kosecka, Ph.D. dissertation. 1.2, 1.3

[18] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohli, J. Shotton, S. Hodges, and A. Fitzgibbon, "Kinectfusion: Real-time dense surface mapping and tracking," in *Proceedings of the 2011 10th IEEE International Symposium on Mixed and Augmented Reality*, 2011. (document), 1.2, 1.3, 2.2, 2.4, 2.16

[19] R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2011. 1.2

[20] T. Whelan, J. McDonald, M. Kaess, M. Fallon, H. Johannsson, and J. Leonard, "Kintinuous: Spatially extended KinectFusion," in *RSS Workshop on RGB-D: Advanced Reasoning with Depth Cameras*, 2012. 1.2

[21] J. Stühmer, S. Gumhold, and D. Cremers, "Real-time dense geometry from a handheld camera," in *Proceedings of the 32nd DAGM conference on Pattern recognition*, 2010. 1.3

[22] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of rgb-d slam systems," in *Proc. of the International Conference on Intelligent Robot Systems (IROS)*, 2012. 1.3, 3, 3.1, 3.1.1, 4

[23] K. Khoshelham and S. O. Elberink, "Accuracy and resolution of kinect depth data for indoor mapping applications," *Sensors*, vol. 12, 2012. 2.1

[24] P. J. Besl and N. D. McKay, "A method for registration of 3-d shapes," *IEEE Trans. Pattern Anal. Mach. Intell.*, 1992. 2.2, 2.4

[25] B. Curless and M. Levoy, "A volumetric method for building complex models from range images," in *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, 1996. 2.2

[26] S. Paris, P. Kornprobst, J. Tumblin, and F. Durand, "A gentle introduction to bilateral filtering and its applications," in *ACM SIGGRAPH 2008 classes*, 2008. (document), 2.3.2, 2.3.2, 2.5

[27] NVIDIA, *NVIDIA CUDA Programming Guide 4.1*, 2012. 2.3.2, 2.4.1

[28] R. I. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed. Cambridge University Press, 2004. (document), 2.3.4, 2.9

[29] K.-L. Low, "Linear least-squares optimization for point-to-plane icp surface registration," Department of Computer Science, University of North Carolina at Chapel Hill, Tech. Rep. TR04-004, 2004. 2.4, 2.4.2

[30] M. Harris, S. Sengupta, and J. D. Owens, "Parallel prefix sum (scan) with CUDA," in *GPU Gems 3*, 2007. 2.4.3

[31] S. Osher and R. Fedkiw, *Level Set Methods and Dynamic Implicit Surfaces*. Springer Verlag, 2003. 2.5

[32] R. Kümmerle, B. Steder, C. Dornhege, M. Ruhnke, G. Grisetti, C. Stachniss, and A. Kleiner, "On measuring the accuracy of slam algorithms," *Auton. Robots*, 2009. (document), 3.9

[33] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3d surface construction algorithm," *SIGGRAPH Comput. Graph.*, 1987. 4