Reconstrução de Objetos 3D utilizando Estruturas de Indexação Espacial com o Microsoft Kinect

Fernando Akio Araújo Yamada, Luciano Walenty Xavier Cejnog, Renan Augusto Dembogurski,
Marcelo Bernardes Vieira, Rodrigo Luis de Souza da Silva
Departamento de Ciência da Computação
Universidade Federal de Juiz de Fora
Juiz de Fora, Brasil
akio@ice.ufjf.br, luciano.wxc@gmail.com, ad.renan@gmail.com,marcelo.bernardes@ufjf.edu.br, rodrigoluis@ice.ufjf.br

***Resumo*—O registro de superfícies representa ainda hoje um desafio na área de visão computacional. Para auxiliar neste problema diversos equipamentos podem ser utilizados, mas seu preço muitas vezes inviabiliza sua utilização. Neste contexto, um equipamento com preço mais acessível, suporte a imagens de profundidade e câmera RGB a ser considerado é o Microsoft Kinect. O objetivo deste trabalho é propor um método de reconstrução de objetos reais a partir das nuvens de pontos geradas pelo Kinect, otimizado com uso de K-D Trees.**

***Keywords*-Reconstrução geométrica; Microsoft Kinect; Iterative Closest Point; K-D Tree;**

I. Introdução

Na área de Visão Computacional, um problema recorrente é o mapeamento de superfícies. Este pode ser definido como mapear um ou mais objetos reais, gerando um modelo de representação virtual dos mesmos. De forma geral, esse processo pode ser realizado através de *Scanners* 3D, porém o alto custo destes aparelhos pode representar um empecilho à sua utilização. Com o *Kinect*, dispositivo utilizado no trabalho aqui apresentado, tem-se uma alternativa mais econômica para a realização dessa tarefa.

Outro problema muito explorado na visão computacional é o registro de superfícies. Neste problema o objetivo é determinar as transformações espaciais que otimizam os alinhamentos entre conjuntos de pontos que representam um determinado objeto. Um dos algoritmos mais utilizados para fazer registro de superfícies é o ICP (Iterative Closest Point).

O objetivo deste trabalho é desenvolver uma técnica utilizando *K-D Trees*, para otimizar a busca do vizinho mais próximo no algoritmo ICP, utilizando nuvens de pontos 3D obtidas com um *Kinect*. Esta otimização resulta em uma melhoria significativa em relação ao algoritmo de busca original.

II. *Kinect*

O *Kinect* (Figura 1) é um dispositivo de controle baseado em movimentos e fala para o console *Xbox 360*, desenvolvido pela *Microsoft* em colaboração com a empresa israelita *Primesense*. O dispositivo é composto por um

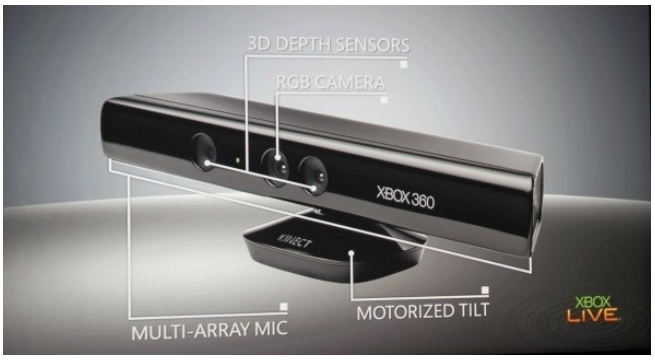

Figura 1. *Microsoft Kinect*

emissor infravermelho, uma câmera *RGB*, um sensor de profundidade (infravermelho) e uma sequência de microfones utilizada para capturar áudio direcional. A câmera e o sensor funcionam na frequência de 30Hz, ambas com resolução de 640x480.

O *Kinect* apresenta algumas limitações técnicas em seu funcionamento. São mapeados os pontos cuja distância em relação ao *Kinect* está entre aproximadamente 70 centímetros e 6 metros. Pontos fora dessa faixa não são reconhecidos.

A. A nuvem de pontos

Tanto a câmera *RGB* quanto o sensor de profundidade possuem resolução 640x480, ou seja, a cada *frame* é gerada uma nuvem de pontos que possui 307200 pontos. Para cada ponto mapeado, são fornecidas informações de profundidade e de cor (*RGB*). A Figura 2 apresenta um exemplo de nuvem de pontos obtida pelo *Kinect* sem informação de cor.

B. Framework Utilizado

Após o lançamento do *Kinect* a comunidade movimentou-se no sentido de desenvolver um meio de uso do dispositivo em plataformas além do *Xbox 360*. Em pouco tempo, foram desenvolvidos *drivers* para simplificar o acesso às informações obtidas pelo *Kinect* em um ambiente de programação.

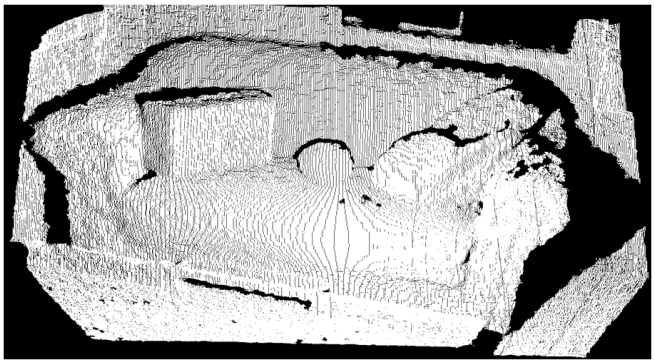

Figura 2. Exemplo de nuvem de pontos gerada pelo Kinect.

Entre os mais conhecidos atualmente, podemos destacar o *OpenNI*, que foi desenvolvido pela *Primesense*, o *Microsoft Kinect SDK*, da própria *Microsoft*, e o *OpenKinect*, driver *open-source* desenvolvido por uma comunidade aberta. Utilizou-se neste trabalho o *OpenKinect*, para a geração das nuvens de pontos.

O *OpenKinect* [1] utiliza uma biblioteca denominada *libfreenect*, que fornece uma *API* capaz de retornar informações de cor, profundidade, motores, acelerômetro e *LED* do *Kinect*.

As imagens fornecidas pela *libfreenect* apresentam o valor da profundidade em cada ponto variando entre 0 e 1024. Para os pontos que não são mapeados (ou seja, que não obtêm resposta do sensor IR), é atribuído o valor 0 para a profundidade. A cada *frame*, o *libfreenect* fornece um *array* com os dados relativos à profundidade dos 307200 pontos, e outro *array* com a informação de cor (*RGB*) relativa a cada ponto.

C. Captura das malhas

Neste trabalho, cada malha capturada foi salva em um arquivo com extensão *.dat* no qual a primeira linha contém o número de pontos da malha e em cada linha que se segue, estão os valores de x, y, z, r, g e b de cada ponto. Os valores de r, g e b foram utilizados somente para exibição e não possuem relevância no alinhamento das malhas.

D. Filtragem das malhas

Devido à densidade dos pontos e a detalhes de outros objetos do cenário foi realizada uma filtragem dos pontos por profundidade, mantendo na nuvem de pontos somente os pontos do objeto de interesse.

Em [11] há exemplos de filtragens de pontos por amostragens, mas neste trabalho utilizou-se um método mais simples. Nesta abordagem é possível realizar filtragens por fatores arbitrários definidos pelo usuário na malha M ou P. Este método reduz o nível de detalhes a fim de diminuir o tempo de execução. Definido um fator x, o algoritmo de filtragem passará por todos os pontos e selecionará um a cada x pontos da malha escolhida; desta forma este novo conjunto possuirá $\frac{1}{x}$ pontos.

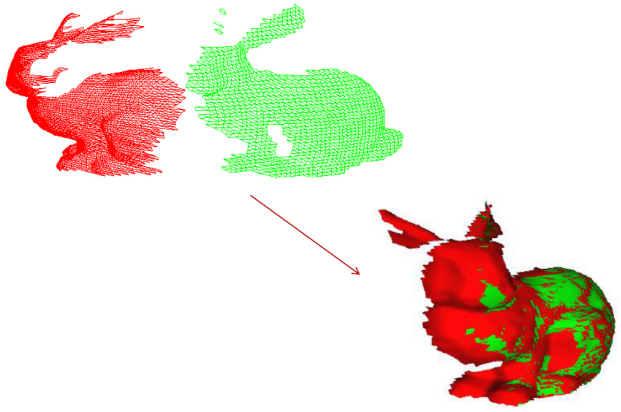

Figura 3. Exemplo do *ICP*

Uma filtragem por profundidade também foi utilizada, semelhante à aplicada em [12]. A fim de capturar apenas os pontos pertencentes ao objeto, uma faixa de profundidade foi determinada, onde pontos fora desta faixa foram descartados.

III. O Algoritmo *ICP*

O *ICP (Iterative Closest Point)* [3] é um processo iterativo utilizado para realizar o alinhamento entre duas nuvens de pontos, minimizando a distância entre elas. As entradas do algoritmo são duas malhas distintas, e como saída, o algoritmo fornece uma transformação rígida, composta por uma rotação e uma translação, que alinha essas duas malhas. Um exemplo de alinhamento realizado através do ICP pode ser observado na Figura 3.

Alternativas para uso de cor foram apresentadas em [4], onde pesos são atribuídos para cor e profundidade dos pontos. Em [10] as cores são contadas como uma dimensão a mais, desta forma os pontos possuem seis dimensões (espaço 3D + RGB).

Utilizou-se neste trabalho apenas a profundidade dos pontos para os cálculos do ICP.

A. Visão Geral

O processo se resume a:

procedure ITERATIVE CLOSEST POINT(M, P, t)
  while $d \leq t$ **do**
    for all $m \in M$ **do**
  Busca vizinho mais próximo em P;
    end for
  Calcula a distância média d;
  Calcula a matriz de transformação T, que aproxima os pontos;
  Aplica T a todos os pontos de M
  end while
end procedure

Enquanto a distância média d entre as duas malhas for menor que um *threshold* t, a cada iteração o ICP buscará, para cada ponto da malha M, o vizinho mais próximo em P, ou seja, o ponto em P que possui menor distância euclidiana. Com este conjunto de pontos é calculada a distância média e estimada uma transformação rígida que aproxima o conjunto P a M.

B. Aspectos Técnicos

Conforme apresentado em [8], o ICP tem como entrada duas malhas, equivalentes a dois pontos de vista distintos de um mesmo objeto, M e P, com N_m e N_p pontos em $\mathbf{R}^3$, respectivamente.

Para cada ponto $m_i \in M$, define-se o **operador de ponto mais próximo** $\mathbf{C}(m_i, P)$, como a operação que retorna o ponto na malha P cuja distância euclidiana até m_i é mínima.

Em cada iteração do algoritmo, é definido o conjunto de pontos $y_i \in Y$, tal que $y_i = \mathbf{C}(m_i, P)$. É importante notar que o conjunto Y e o conjunto P possuem o mesmo número de elementos.

Um **quatérnio** é um vetor em $\mathbf{R}^4$ que representa um arco em $\mathbf{R}^3$. A unidade quatérnio é dada pela quádrupla:

$$\vec{q_r} = [q_0 q_1 q_2 q_3]^t,\ q_0 \geq 0 \text{ e } q_0^2 + q_1^2 + q_2^2 + q_3^2 = 1$$

Associa-se ao vetor $\vec{q_r}$ um vetor $\vec{q_t} = [q_4 q_5 q_6]^t$ de translação, constituindo assim o **vetor completo de registro de estado** $\vec{q} = [\vec{q_r} \mid \vec{q_t}]$

O vetor $\vec{q}$, é tal que:

$$\vec{q} = \mathbf{Q}(P, Y)$$

onde $\mathbf{Q}$ é o operador de quatérnio de mínimos quadrados.

Para encontrar o operador $\mathbf{Q}$, é necessário definirmos os centróides $\vec{\mu}_p$ de P e $\vec{\mu}_y$ de Y como:

$$\vec{\mu}_p = \frac{1}{N_p}\sum_{i=0}^{N_p} \vec{p}_i \qquad \vec{\mu}_y = \frac{1}{N_y}\sum_{i=0}^{N_p} \vec{y}_i$$

A fim de evitar que a transformação caia em um ótimo local, ou seja, estimar uma transformação que não alinha corretamente as malhas, aplica-se uma pequena translação de 10 unidades no eixo y, em $\vec{\mu}_p$.

Definidos os centróides, calculamos a **matriz de covariância cruzada** de P e Y:

$$\Sigma_{py} = \frac{1}{N_p}\sum_{i=0}^{N_p} (\vec{p}_i \vec{y}_i^t) - (\vec{\mu}_p \vec{\mu}_y^t)$$

A partir da equação anterior, definimos a matriz $A = \Sigma_{py} - \Sigma_{py}^t$, e utilizamos os componentes cíclicos da matriz A para compôr um vetor $\Delta = [A_{23} A_{31} A_{12}]^t$. Também é importante a definição do traço da matriz Σ:

$$tr(\Sigma_{py}) = \Sigma_{11} + \Sigma_{22} + \Sigma_{33}$$

Utilizando as definições anteriores, finalmente definimos o operador de quatérnio de mínimos quadrados $Q(\Sigma_{py})$, como:

$$Q(\Sigma_{py}) = \begin{vmatrix} tr(\Sigma_{py}) & \Delta^t \\ \Delta & \Sigma_{py} + \Sigma_{py}^t - tr(\Sigma_{py}) I_3 \end{vmatrix}.$$

onde I_3 é a matriz identidade 3x3. A dimensão de Q é 4x4.

O quatérnio de rotação $\vec{q_r}$ é descrito como o autovetor associado ao maior autovalor de $Q(\Sigma_{py})$. Utilizando o quatérnio, construímos a matriz de rotação $\mathbf{R}(\vec{q_r})$:

$$\begin{vmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2(q_1 q_2 - q_0 q_3) & 2(q_1 q_3 + q_0 q_2) \\ 2(q_1 q_2 + q_0 q_3) & q_0^2 + q_2^2 - q_1^2 - q_3^2 & 2(q_2 q_3 - q_0 q_1) \\ 2(q_1 q_3 - q_0 q_2) & 2(q_2 q_3 + q_0 q_1) & q_0^2 + q_3^2 - q_1^2 - q_2^2 \end{vmatrix}$$

A partir da matriz de rotação calculada, e dos centróides definidos anteriormente, definimos o vetor de translação:

$$\vec{q_t} = \vec{\mu}_y - \mathbf{R}(\vec{q_r})\vec{\mu}_p$$

Por fim, montamos a matriz final de transformação 4x4:

$$T = \begin{vmatrix} R(\vec{q_r})_{11} & R(\vec{q_r})_{12} & R(\vec{q_r})_{13} & \vec{q_t}_1 \\ R(\vec{q_r})_{21} & R(\vec{q_r})_{22} & R(\vec{q_r})_{23} & \vec{q_t}_2 \\ R(\vec{q_r})_{31} & R(\vec{q_r})_{32} & R(\vec{q_r})_{33} & \vec{q_t}_3 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

Essa matriz T deve ser aplicada a todos os pontos do conjunto P. Como os pontos de P são tridimensionais, são usadas as coordenadas homogêneas $(x_i, y_i, z_i, 1)$ e normaliza-se o ponto pela quarta coordenada após a transformação. Mais detalhes podem ser encontrados em [6].

O resultado do alinhamento entre duas malhas é salvo e em seguida uma nova malha representando um novo ponto de vista do objeto é alinhada a esta, compondo uma representação completa do objeto.

C. Otimização do processo

Devido ao tamanho das malhas geradas pelo *Kinect*, o algoritmo original de busca do vizinho mais próximo muitas vezes se torna inviável. Desse modo, é possível otimizar essa etapa do algoritmo utilizando uma estrutura eficiente de indexação. Nesse trabalho, utilizou-se a *K-D Tree*, por sua simplicidade e eficiência [12] [7].

IV. K-D TREE

A *K-D Tree* é uma estrutura de indexação espacial, para pontos de K dimensões. É uma árvore binária facilmente balanceável, que realiza buscas em aproximadamente $log(n)$ [2].

A. Estrutura da K-D Tree

Cada nível de profundidade da *K-D Tree* está associado a uma dimensão do espaço. Cada nó não-folha da *K-D Tree* pode ser pensado como um hiperplano que divide o espaço em dois sub-espaços. Pontos à esquerda desse hiperplano representam a sub-árvore à esquerda do nó. De forma equivalente, os pontos à direita representam a sub-árvore à direita. O hiperplano representado por um nó é perpendicular à dimensão associada ao nível de profundidade do ponto. Um exemplo de espaço 2D indexado pode ser visto na Figura 4 e a representação da *K-D Tree* resultante na Figura 5.

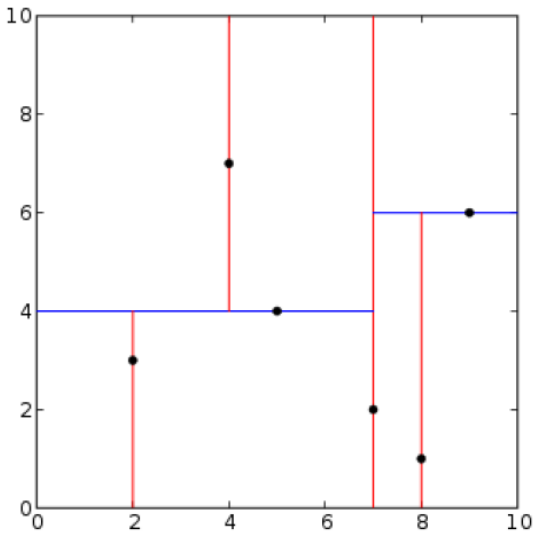

Figura 4. Exemplo de espaço em $\mathbf{R}^2$ indexado por uma *K-D Tree*

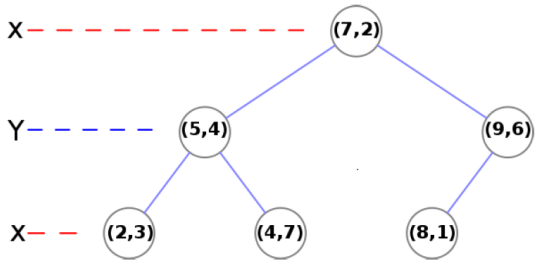

Figura 5. Representação em Árvore correspondente ao exemplo anterior

B. Algoritmo de construção da K-D Tree

Dado um conjunto de pontos P em $\mathbf{R}^k$, inicia-se a construção da árvore ordenando P em relação à coordenada associada à profundidade atual. Para o nó raiz inicia-se, por conveniência, com a coordenada x.

O ponto na posição referente à mediana da lista de pontos é inserido no nó atual, garantindo dessa forma uma árvore balanceada [5]. Todos os pontos antes da mediana serão incluídos na sub-árvore à esquerda, e todos os pontos depois serão incluídos na sub-árvore à direita.

function CONSTROIKDTREE(pontos, eixo)
$eixo \leftarrow eixo \ \mathbf{mod}\ K$;
$\mathbf{ordena}(pontos, eixo)$;
$mediana \leftarrow pontos[tamanho/2]$;
$no.localizacao \leftarrow mediana$;
$no.esquerda \leftarrow \mathbf{constroiKDTree}\ (pontos[0 : mediana],\ eixo + 1)$;
$no.direita \leftarrow \mathbf{constroiKDTree}\ (pontos[mediana : tamanho],\ eixo + 1)$;
end function

C. Método de busca

Como visto em [9], dado um ponto p, deseja-se saber seu correspondente mais próximo na *K-D Tree* K e a distância d_m entre eles.

A busca inicia-se percorrendo K até achar a folha que seria pai de p. Define-se o valor inicial de d_m como a distância euclidiana dessa folha até p. Caso $p \in K$, retorna-se p.

Na sequência, passa-se a checar em K quais sub-árvores podem conter um ponto mais próximo que o atual. Começando da raiz de K, verifica-se se a distância d_p (distância de p ao nó atual) é menor que d_m. Se sim, atualiza-se qual o ponto mais próximo e sua respectiva distância.

Para identificar em qual sub-árvore a busca deve prosseguir, deve-se fazer a interseção entre o hiperespaço do nó atual com uma hiperesfera de raio igual à menor distância, centrada em p. Para tal, checa-se se a distância paralela ao eixo atual entre o ponto em questão e o ponto procurado é menor que d_m. Se esta distância não for menor, não se pode afirmar em qual sub-árvore está o ponto mais próximo e a checagem é feita em ambas. Caso seja menor, pode-se afirmar que um dos hiperespaços não contém pontos que sejam mais próximos que o atual. A seguir faz-se um teste, verificando se a coordenada do ponto no eixo atual é maior que a coordenada de p também no eixo atual. Se sim, a busca prossegue na sub-árvore à esquerda, caso contrário, na sub-árvore à direita.

A busca termina quando cada uma das checagens chegar a uma folha.

V. RESULTADOS

Foram testados os alinhamentos entre dois pontos de vista diferentes de uma caneca (Figura 6) e de um bichinho de pelúcia (Figura 7). O resultado dos alinhamentos pode ser observado na Figura 8. As Tabelas I, II, III apresentam os resultados numéricos referentes aos tempos gastos para o

processamento do *ICP* com e sem o uso da *K-D Tree* e diferentes fatores de redução.

Tabela I
RESULTADOS DO ICP COM E SEM A K-D TREE

Exemplo	N_M	N_P	$T_{original}$	$T_{otimizado}$
Caneca	6820	7621	7.738s	0.109s
Pelúcia	10243	11202	15.740s	0.100s

Tabela II
RESULTADOS DO ICP COM E SEM A K-D TREE, COM FATOR DE REDUÇÃO DE 3X

Redução 3x	N_M	N_P	$T_{original}$	$T_{otimizado}$
Caneca	2274	2541	0.873s	0.047s
Pelúcia	3415	3734	1.778s	0.030s

Tabela III
RESULTADOS DO ICP COM E SEM A K-D TREE, COM FATOR DE REDUÇÃO DE 5X

Redução 5x	N_M	N_P	$T_{original}$	$T_{otimizado}$
Caneca	1364	1525	0.343s	0.031s
Pelúcia	2049	2241	0.655s	0.010s

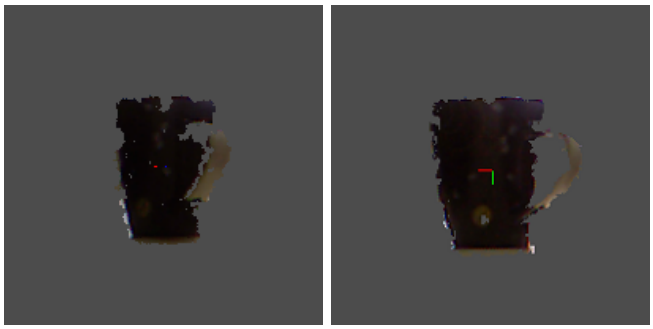

Figura 6. Diferentes pontos de vista da caneca

Figura 7. Diferentes pontos de vista do bicho de pelúcia

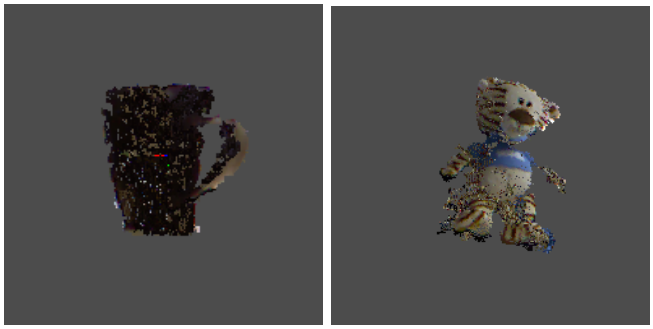

Figura 8. Resultado do alinhamento da caneca e do bicho de pelúcia

O algoritmo com a K-D Tree apresentou, como esperado, um funcionamento muito mais rápido em relação ao algoritmo original do ICP, apresentando mais de uma ordem de grandeza de diferença.

Conclui-se que a utilização do algoritmo *ICP* e a *K-D Tree* fazem com que o uso do *Kinect* para captura de geometria seja viável.

VI. TRABALHOS FUTUROS

A próxima etapa deste projeto é implementar métodos de triangulação para gerar malhas triangulares a partir das nuvens de pontos geradas.

Pretende-se implementar o *ICP* considerando as cores dos pontos, o que possibilitaria um melhor alinhamento das nuvens de pontos, especialmente em objetos simétricos.

Realizar alinhamentos em tempo real sucessivos entre as nuvens, possibilitando a captura da geometria completa de um determinado objeto a partir do *Kinect* também é um dos projetos futuros deste trabalho.

REFERÊNCIAS

[1] Openkinect project, 2010.

[2] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, Sept. 1975.

[3] P. Besl and H. McKay. A method for registration of 3-d shapes. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 14(2):239 –256, feb 1992.

[4] S. Druon, M. Aldon, and A. Crosnier. Color constrained icp for registration of large unstructured 3d color data sets. In *Information Acquisition, 2006 IEEE International Conference on*, pages 249 –255, aug. 2006.

[5] J. H. Friedman, J. L. Bentley, and R. A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Math. Softw.*, 3(3):209–226, Sept. 1977.

[6] B. K. P. Horn. Closed-form solution of absolute orientation using unit quaternions. volume 4, pages 629–642. Journal of the Optical Society of America A, Apr 1987.

[7] 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*, UIST '11, pages 559–568, New York, NY, USA, 2011. ACM.

[8] M. B. V. J. L. R. DE SOUZA FILHO. Alinhamento de nuvens de pontos adquiridos através de digitalizador câmera-projetor com luz estruturada, 2010.

[9] A. Milev. Kd tree - searching in n-dimensions, part i, 2007.

[10] D. Neumann, F. Lugauer, S. Bauer, J. Wasza, and J. Hornegger. Real-time rgb-d mapping and 3-d modeling on the gpu using the random ball cover data structure. In *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, pages 1161 –1167, nov. 2011.

[11] S. Rusinkiewicz and M. Levoy. Efficient variants of the icp algorithm. In *3-D Digital Imaging and Modeling, 2001. Proceedings. Third International Conference on*, pages 145 –152, 2001.

[12] D. A. Simon, M. Hebert, and T. Kanade. Techniques for fast and accurate intra-surgical registration. *Journal of image guided surgery*, 1:17–29, 1995.