MINISTÉRIO DA CIÊNCIA E TECNOLOGIA
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

ALGORITMO PARA DETERMINAÇÃO DE COORDENADAS ESPACIAIS DE OBJETOS COM BASE EM IMAGENS

RELATÓRIO FINAL DE PROJETO DE INICIAÇÃO CIENTÍFICA
(PIBIC/CNPq/INPE)

Vivian Dorat Betoni (UNIVAP, Bolsista PIBIC/CNPq)
E-mail: vdb29586@yahoo.com.br

Dr. Valdemir Carrara (DMC/ETE/INPE, Orientador)
E-mail: val@dem.inpe.br

Julho de 2007

RESUMO

Iniciado em fevereiro de 2006, este trabalho tem como objetivo a criação e desenvolvimento de um algoritmo que defina coordenadas espaciais de objetos, com base em um processamento de coordenadas bidimensionais obtidas por meio de imagens (fotografias) do objeto, em diversos pontos de vista, de uma mesma cena, de modo a possibilitar - utilizando os pontos obtidos - a construção tridimensional deste objeto. O processamento de coordenadas consiste em uma minimização do erro entre as projeções das prováveis soluções das coordenadas espaciais, passadas por entradas manuais na imagem pelo usuário, e as respectivas coordenadas reais das imagens. Para a determinação do erro e sua variação com relação aos parâmetros das transformações e sua minimização, o algoritmo realiza aplicações de transformações lineares de rotação, variação de escala e translação como em cálculos estereoscópicos, de maneira a tentar encontrar a diferença entre os diferentes pontos de vista da cena. O algoritmo está sendo desenvolvido em linguagem C++, permitindo uma possível implementação visual tridimensional do cenário e dos objetos por meio das bibliotecas gráficas do OpenGl (*Open Graphics Library*).

ALGORITHM FOR IMAGE-BASED DETERMINATION OF AN OBJECT'S SPACE COORDINATES

ABSTRACT

The main objective of this work, which was started in February of the year 2006, is to create and develop algorithms to define space coordinates of an object, based on the processing of stereoscopic two-dimensional coordinates obtained from images (photographs) in different angles of a scene in order to permit – using the obtained points – the three-dimensional construction of this same object. The processing of coordinates consists in the minimization of the error between the projections of the possible solutions of the space coordinates, obtained by user input, and their respective image coordinates. In order to determine and minimize the error and its variation with respect to the transformation parameters, the algorithm applies linear transformations, such as rotation, scaling and translation like in stereoscopic calculations, in order to try and find the difference between the different points of view of the scene. The algorithm is being developed in C++ language, allowing a possible implementation of a three-dimensional visualization of the scene and objects using OpenGl (Open Graphics Library).

SUMÁRIO

LISTA DE FIGURAS

CAPÍTULO 1

INTRODUÇÃO

A criação de um modelo computacional fotorealístico tridimensional encontra várias aplicações que vão desde planejamentos arquiteturais, reconstruções arqueológicas até a criação de ambientes virtuais e efeitos especiais. Combinando técnicas de Computação Gráfica e Computação Visional torna-se possível a criação de um algoritmo que defina coordenadas espaciais de objetos através de um processamento de coordenadas bidimensionais, utilizando como base algumas imagens (fotografias) de um mesmo objeto em diversos ângulos, de uma mesma cena, gerando assim o objeto desta cena em três dimensões. O processo de modelagem é composto por duas partes:

- na primeira, utilizando-se de técnicas de modelagem fotogramétrica, busca-se a extração de informações nas imagens, a serem usadas na sua reconstrução tridimensional, de maneira interativa, mas com cálculos para detecção de possíveis erros nos pontos indicados pelo usuário e, caso ocorra, da minimização desta distância.

- na segunda parte, por meio de técnicas estereoscópicas, é feita a recuperação do grau de desvio da imagem base (principal) com as outras imagens e seus respectivos pontos de interligação, recuperando a profundidade da extensão espaçada dos pares de imagem (ou seja, a variação de escala das câmeras), rotação e translação.

Portanto, o objetivo deste trabalho é determinar as coordenadas espaciais de um objeto, utilizando-se como base um grupo de imagens e buscando uma modelagem otimizada, com o uso de menos fotos do que utilizam as aproximações de modelagem com base em imagens atuais.

Foram utilizadas, primeiramente, a linguagem C e, mais tarde, testou-se a implementação nas linguagens C++ e C#. Porém para a utilização de recursos do OpenGl, como foi definido previamente, é mais indicado que o resultado final seja com a utilização da linguagem C++.

As principais funções implementadas visando à geração tridimensional do objeto, foram:

1) Correção dos cálculos de minimização, visando um resultado mais próximo do valor real.
2) Modificações no algoritmo visando um desempenho mais otimizado.

1.1 - Organização do documento

Este relatório está dividido em três capítulos a seguir:

- Capítulo 2: DESENVOLVIMENTO – Onde serão descritos os métodos, as pesquisas, formulações e funções abordadas.

- Capítulo 3: ALGORITMO – Explicação do algoritmo que realiza os cálculos para desenvolvimento do objeto em três dimensões.

- Capítulo 4: CONCLUSÃO – Conclusão sobre os métodos estudados e atividades desenvolvidas.

CAPÍTULO 2

DESENVOLVIMENTO

A necessidade de se criar ambientes virtuais que reproduzam ambientes reais está em constante crescimento e vem ganhando interesse cada vez maior nos campos de Computação Gráfica e Visional. Infelizmente o processo tradicional em que o usuário marca a cena tem muitas desvantagens: é extremamente trabalhoso, pois, precisa-se de muitas informações e, sendo assim, é difícil verificar se o resultado é preciso; o mais decepcionante é que os modelos gerados são facilmente reconhecidos como computadorizados, e mesmo os que têm um bom mapeamento de textura falham. Como resultado, é muito mais fácil distinguir uma imagem sintetizada de uma imagem real. (Debevec, 1996).

Para determinar as coordenadas em três dimensões, estas têm que ser de pontos de vistas semelhantes. Ineficiências como a necessidade da inserção de várias fotos para um melhor resultado dificultam a criação de um ambiente virtual com uma navegação otimizada. Para isso foi buscado o desenvolvimento de uma aproximação utilizando-se técnicas novas, o que resulta no uso de um pequeno conjunto de imagens que pode produzir resultados de qualquer ponto de vista.

Visando alcançar os objetivos deste trabalho e as devidas implementações das funções necessárias, está sendo realizado um estudo em cima de informações coletadas no artigo “Modeling and Rendering Architecture from Photographs” (Debevec, 1992), livros sobre Álgebra Linear (Boldrini et al., 1980), e pesquisas na rede mundial de computadores.

2.1 - Método para determinação da posição de pontos no espaço por meio de vistas estereoscópicas.

Mapeamentos de texturas dependem da visão utilizada (do ponto de vista) e detalhes podem ser recuperados automaticamente através da correspondência

estereoscópica baseada em modelos. O resultado final pode ser gerado por mapeamento de texturas (menos novo) ou baseado em imagens (mais novo).

Debevec em 1996 desenvolveu um programa que visava precisão, conveniência e realismo. Utilizando-se de sistemas de sintetização baseado em geometria e novas técnicas baseadas em imagens, gerava uma arquitetura fotogramétrica com tratamento de detalhes recuperados por correspondência estereoscópica que requeria um pequeno conjunto de fotos, produzindo resultados de qualquer ponto de vista. Este programa, o *Façade* (Figura 2.1) seria o precursor do software *Canoma* da MetaCreations (Viewpoint Corporation, 2000).

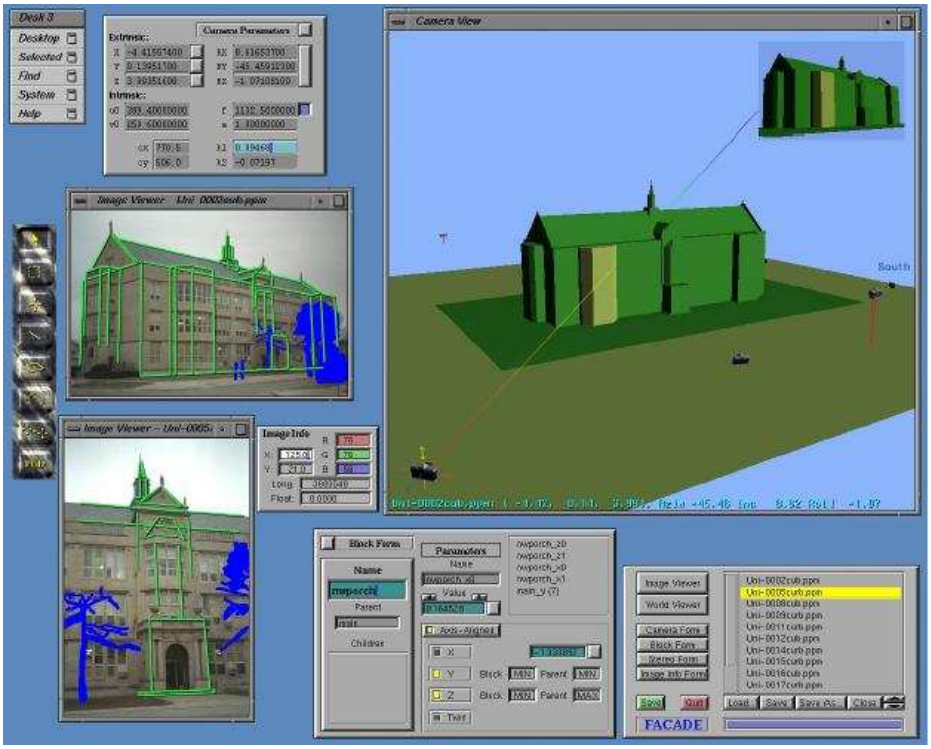

Fig. 2.1 – Interface do programa Façade (Fonte: Debevec, 1996).

Existem diversas técnicas de extração de coordenadas tridimensionais de imagens, sendo algumas das mais relevantes:

Calibração de câmera – Se dá quando as coordenadas da imagem e as direções do centro da câmera são conhecidas. As dadas posições da câmera no espaço (sua translação e rotação) não são necessárias. Apesar do aumento da busca em se utilizar

imagens geradas com câmeras não calibradas, foi descoberto que a sua calibração pode trazer resultados mais simples e diretos. Este princípio é utilizado em diversos tipos de *scanner 3D* (ver, por exemplo, *Scan 3D*, 1999).

Estrutura a partir de imagens distintas – Utiliza-se de técnicas usadas na fotogrametria para se produzir mapas topográficos: com mais de uma imagem é possível se deduzir matematicamente as localizações tridimensionais dos pontos e as posições originais das câmeras até certo fator de escala desconhecido. O problema desta técnica é que a recuperação da estrutura do objeto é muito sensível a "ruídos" nas medidas das imagens quando estas têm uma translação de suas posições muito pequena. Debevec utilizou como solução a recuperação de modelos geométricos para medir as imagens, buscando corrigir o problema de minimização da distância entre os pontos reais e os pontos estimados e incluindo restrições adicionais em geometrias conhecidas. Neste trabalho será buscada a recuperação das medidas das imagens em um conjunto de pontos no espaço.

Formas de contorno da silhueta – A geometria do objeto é obtida através da intersecção de linhas que cruzam seu contorno. Quanto mais imagens, mais fácil a recuperação dos raios que não intercedem com o objeto, possibilitando a captura de algumas estruturas de formas côncavas simples. Apesar disso, com um pequeno número de imagens é possível a recuperação de um objeto bem próximo do real. Este tipo de modelagem traz resultados melhores para objetos curvos, e, caso exista uma geometria conhecida de câmera e os objetos possam ser automaticamente segmentados, o processo pode ser automatizado. Embora não tenha um resultado preciso em formas com muitas pontas e cavidades, pode ser de grande valia na recuperação de vegetações em cenas arquiteturais.

Correspondência estereoscópica – A teoria geométrica usada na recuperação da estrutura obtida a partir de imagens distintas assume que é possível resolver o problema da correspondência, que é identificar os pontos em duas ou mais imagens que são projeções do mesmo ponto no espaço (Debevec et al., 1996). Na visão dos humanos, ocorre a chamada *Estereoscopia Binocular*, melhor exemplificada pela Figura 2.2. Anos

de pesquisas comprovaram que essa determinação é muito dificultosa. Só em casos como na visão humana, em que as imagens são semelhantes e com pouca distância uma da outra se obtém sucesso. A medida que o *baseline* (distância entre a localização das câmeras) aumenta, torna-se mais difícil a determinação da correspondência correta pelo computador pois a alteração nos *pixels* das imagens confunde o algoritmo que serve-se de técnicas de similaridade, perdendo até detalhes da imagem. A melhor alternativa é a que calcula a profundidade, mas infelizmente é muito sensível a ruídos nas medições das imagens, significando que, à medida que a câmera virtual se afasta do ponto original, a visualização se torna defeituosa.

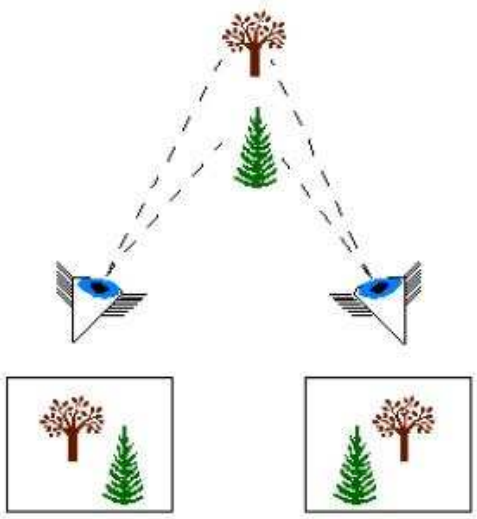

Fig. 2.2 – Visão binocular da mesma cena. (Fonte: Raposo et al., 2004)

Síntese baseada em imagens – Seu modelo consiste em um conjunto de imagens e um mapa de profundidade obtido através do processamento dessas imagens por correspondência estereoscópica. Sendo a profundidade dos pontos conhecida, a imagem é gerada de um ponto de vista próximo ou entre as imagens conhecidas, projetando os *pixels* da imagem nas suas localizações tridimensionais em um novo plano da imagem. Sua vantagem é que pode sintetizar cenas arbitrariamente complexas com uma quantidade constante de cálculos por *pixel*. Porém este método é eficiente apenas quando o mapa de profundidade apresenta pouca variação nas distâncias entre *pixels* vizinhos (como num relevo, por exemplo). Para ambientes lineares com visões próximas funciona bem, mas extrair estimativas de profundidade confiáveis através de estereoscopia é mais complicado. A necessidade de que as imagens devem estar

próximas umas das outras é uma séria limitação na geração de ambientes virtuais plenamente navegáveis.

Neste trabalho é dada uma atenção especial ao desenvolvimento de um algoritmo que, servindo-se de pontos oferecidos pelo usuário em cada imagem bidimensional utilizada, minimiza a distância entre os pontos fornecidos e os pontos reais com base em transformações paramétricas do posicionamento e orientação das câmeras.

2.1.1 - Formulação da minimização do erro

Dado um conjunto de n pontos em um espaço vetorial (ou seja, os pontos reais das imagens) denotados por:

$$P_i = (x_i, y_i, z_i),\ i = 1, ..., n,$$

relativos a um sistema de coordenadas retangulares x, y, z, deseja-se estimar estas coordenadas com base no processamento das projeções destes pontos numa seqüência de imagens.

Considera-se agora l outros sistemas de coordenadas x_k, y_k, z_k, com $k = 1, 2, ... l$ (número de imagens) relacionados ao sistema x, y, z por meio de uma matriz de rotação A_k e de um vetor de translação T_k. Os pontos P_i são projetados nos planos x_k, y_k por meio de uma projeção em perspectiva, o que origina o conjunto de pontos $Q_{ki} = (x_{ki}, y_{ki}, 1),\ i = 1, ..., n$, para cada sistema de coordenadas.

Porém, cada conjunto Q_{ki} destes pontos contém um subconjunto distinto de projeções de P_i, ou seja, os pontos fornecidos (ou estimados) pelo usuário que, em geral, não constitui a totalidade dos n pontos. Isto se deve ao fato de que nem todos os pontos são observados em cada imagem. As coordenadas espaciais dos pontos P_i no sistema k são dadas por P_i^k e são obtidas a partir das equações da projeção em perspectiva:

$$P_i^k = z_{ki}\, Q_{ki} = z_{ki} \begin{pmatrix} x_{ki} \\ y_{ki} \\ 1 \end{pmatrix}$$

onde z_{ki} representa a profundidade (desconhecida) P_i^k e o índice superior k indica o sistema de coordenadas no qual o ponto é referido.

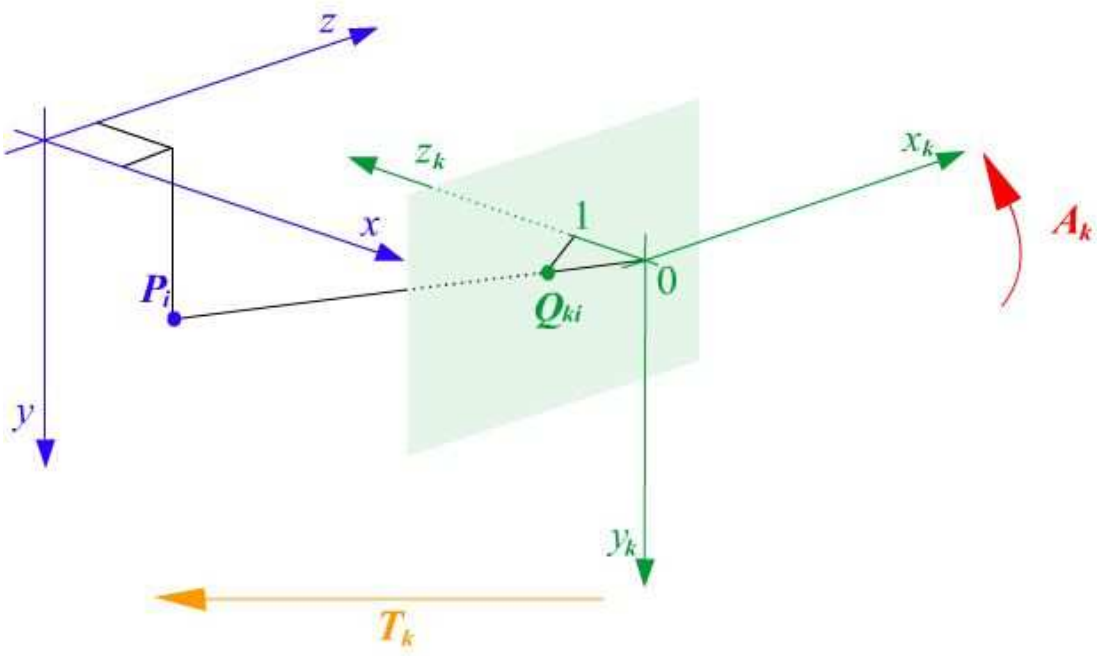

Fig. 2.3 – Relação dos sistemas de coordenadas *x, y, z* e x_k, y_k, z_k, por meio das matrizes de rotação (A_k) e translação (T_k) originando um subconjunto de sistemas de coordenadas (Q_{ki})

Por sua vez as posições relativas ao sistema de coordenadas das projeções (Q_{ki}) relacionam-se com o sistema *x, y, z* dos pontos P_i por meio de uma matriz de rotação A_k e de uma translação de um vetor T_k:

$$P_i = A_k\, (P_i^k - T_k)$$

Procura-se reconstituir os pontos reais (P_i) utilizando-se exclusivamente os diversos conjuntos Q_{ki}. Para isto, não só as coordenadas dos pontos P_i devem ser calculadas, mas também a rotação e a translação efetuadas em cada sistema de coordenadas, bem como as profundidades z_{ki} de cada ponto.

A solução adotada aqui consiste na minimização da distância entre uma estimativa S_i dos pontos P_i, com base nas diversas projeções destes pontos nos sistemas k, ou seja:

$$f = \min \sum_{k=1}^{l} \left[\sum_{i=1}^{n_k} (P_i - S_i)^2 \right]$$

Este problema apresenta duas indeterminações:

- a indeterminação da localização do sistema de coordenadas dos pontos P_i;
- a escala dos sistemas de coordenadas.

De fato, seja, por exemplo, uma translação de um vetor T no sistema de coordenadas dos pontos P_i. Neste caso, a estimativa S_i será também transladada e a função de minimização fica:

$$f = \min \sum_{k=1}^{l} \left[\sum_{i=1}^{n_k} (P_i + T - S_i - T)^2 \right] = \min \sum_{k=1}^{l} \left[\sum_{i=1}^{n_k} (P_i - S_i)^2 \right],$$

o que faz com que a função de minimização não consiga determinar qual foi esta translação.

Seja agora uma rotação neste mesmo sistema causada pela matriz A:

$$f = \min \sum_{k=1}^{l} \left[\sum_{i=1}^{n_k} (AP - AS_i)^2 \right] = \min \sum_{k=1}^{l} \left[\sum_{i=1}^{n_k} A\,A^T (P_i - S_i)^2 \right],$$

e, uma vez que a matriz de rotação é ortogonal, então $A\,A^T = I$, e novamente a função de minimização não foi alterada.

Isto significa, portanto, que o sistema de coordenadas x, y, z não pode ser determinado por este processo, mas apenas as relações entre os diversos sistemas x_k, y_k,

z_k. Logo, deve-se adotar um sistema qualquer para a solução do problema. Adota-se para *x, y, z* um sistema paralelo ao da primeira imagem, x_1, y_1, z_1, aplicando apenas uma translação de h_1 no eixo *z*. Com isso tem-se que $A_1 = I$ e $T_1 = (0, 0, h_1)^T$. Este resultado indica que é igualmente válido resolver o problema no sistema de coordenadas de qualquer uma das imagens, e portanto a função de minimização fica:

$$f = \min \sum_{k=1}^{l} \left[\sum_{i=1}^{n_k} (P_i^k - S_i^k)^2 \right].$$

Porém se S_i for uma solução válida, então aS_i e aP_i, onde *a* é uma constante, também será, pois:

$$f = a^2 \min \sum_{k=1}^{l} \left[\sum_{i=1}^{n_k} (P_i - S_i)^2 \right],$$

o que significa que também a escala do problema não pode ser determinada por meio deste processo.

Assim, deve-se adotar uma dada medida como sendo a unidade. Adota-se, por conveniência, $|T_1| = h_1 = 1$. Contudo deve-se mencionar que os vetores de translação das demais imagens não serão unitários.

Dado um ponto P_i no sistema *xyz*, este ponto pode ser transformado em coordenadas relativas ao sistema $x_k\ y_k\ z_k$ com a (conforme mostra a Figura 2.5):

- aplicação de 3 rotações em direções distintas, de ângulos α_k, β_k e θ_k, nos eixos cartesianos *y, x* e *z*, respectivamente.
- aplicação de uma translação de um escalar h_k na direção de *z*
- aplicação de uma translação de u_k e v_k nas direções x_k e y_k, respectivamente (ou seja, $T_k = (u_k\ v_k\ h_k)^T$)

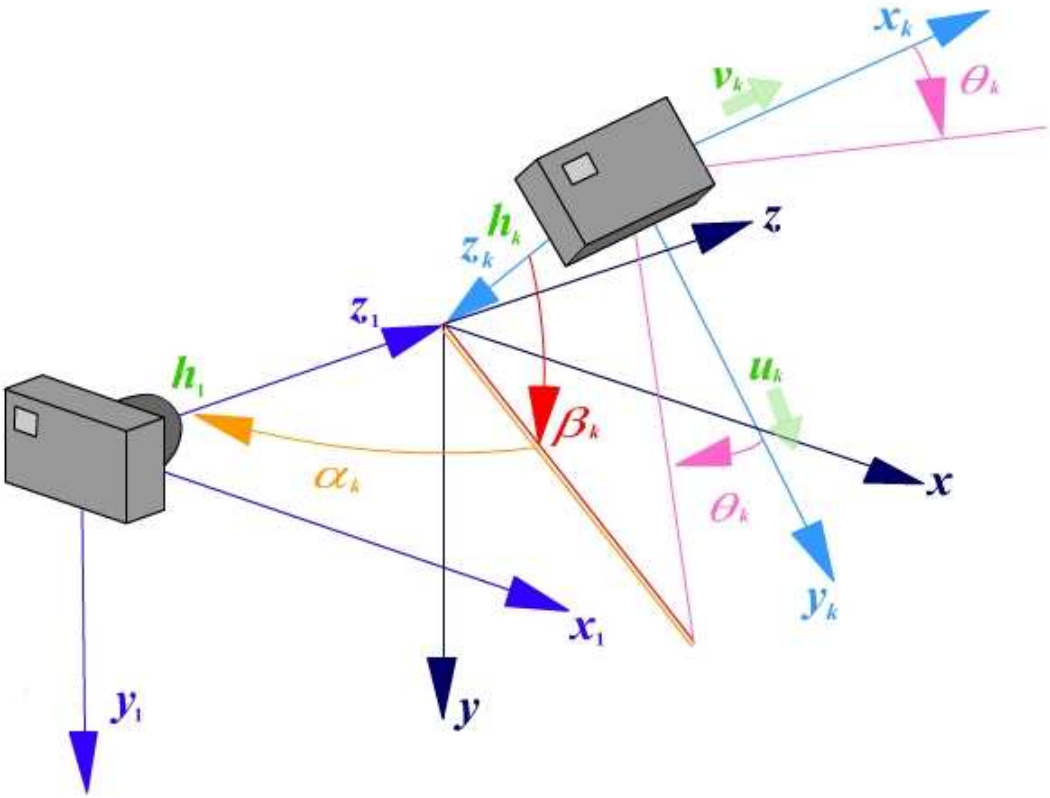

Fig. 2.4 - Sistemas de coordenadas da câmera em duas orientações distintas

Traduzindo-se em termos de equações, tem-se:

$$P_i^k = A_k^T \; P_i + T_k$$

onde a matriz de coordenadas da rotação é dada por:

$$A_k = R_y(\alpha_k)\; R_x(\beta_k)\; R_z(\theta_k)\,,$$

tal que:

$$R_y(\alpha_k) = \begin{pmatrix} \cos\alpha_k & 0 & -\operatorname{sen}\alpha_k \\ 0 & 1 & 0 \\ \operatorname{sen}\alpha_k & 0 & \cos\alpha_k \end{pmatrix},$$

$$R_x(\beta_k) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\beta_k & \text{sen}\,\beta_k \\ 0 & -\text{sen}\,\beta_k & \cos\beta_k \end{pmatrix},$$

e

$$R_z(\theta_k) = \begin{pmatrix} \cos\theta_k & \text{sen}\,\theta_k & 0 \\ -\text{sen}\,\theta_k & \cos\theta_k & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Como o objetivo é comparar as coordenadas dos pontos das diversas imagens entre si, deve-se então adotar um sistema de coordenadas único. Embora o sistema verdadeiro dos pontos P_i não possa ser determinado, é possível adotar o sistema de coordenadas normalizado, como indicado na Figura 2.3, onde $h_1 = 1$. Neste caso deve-se expressar as diversas projeções de P_i neste sistema, e, portanto, é necessário inverter a relação de transformação, ou seja:

$$P_i = A_k\ (P_i^k - T_k)$$

e substituindo-se a transformação de coordenadas e da equação da projeção em perspectiva na função de minimização tem-se:

$$f = \min \sum_{k=1}^{l} \left[\sum_{i=1}^{n_k} (z_{ki}\ A_k\ Q_{ki} - A_k\ T_k - S_i)^2 \right],$$

onde a primeira somatória constitui o conjunto de imagens e a segunda é constituída pelos pontos de cada uma das imagens.

A minimização de f requer que o gradiente seja nulo na solução do problema, isto é:

$$\nabla f(\alpha_k, \beta_k, \theta_k, T_k, S_i, z_{ki}) = \left(\frac{\partial f}{\partial \alpha_k}, \frac{\partial f}{\partial \beta_k}, \frac{\partial f}{\partial \theta_k}, \frac{\partial f}{\partial T_k}, \frac{\partial f}{\partial S_i}, \frac{\partial f}{\partial z_{ki}} \right) = 0 ,$$

para k = 1, 2, ... l , i = 1, 2, ..., n (lembrando que $h_1 = 1$ e $\alpha_1 = \beta_1 = \theta_1 = u_1 = v_1 = 0$). Nota-se que o número de valores (parâmetros de transformação, coordenadas e profundidades) a serem estimados no total é dado por

$$N = 3l + 3l + 3n + \sum_{k=1}^{l} n_k ,$$

que pode ser um número relativamente grande se houverem muitos pontos e muitas imagens.

Uma vez que a função de minimização é quadrática contendo apenas funções lineares ou trigonométricas, pode-se obter este gradiente efetuando-se derivadas analíticas de cada um destes parâmetros. Evita-se, desta forma, a necessidade de se utilizar funções de otimização exclusivamente numéricas. Além disso, a derivada analítica permite obter a solução ótima num único cálculo, a menos, é claro, que as demais variáveis não sejam corretas.

CAPÍTULO 3

ALGORITMO

O método de otimização empregado inicialmente consistiu num algoritmo interativo onde era admitida inicialmente uma solução, e depois se procurava atingir o resultado ótimo com base no cálculo consecutivo das soluções analíticas de cada uma das variáveis. Supôs-se que tal procedimento levaria à solução ótima, pois individualmente cada uma das soluções analíticas apresentava uma variação lenta com os demais parâmetros. Contudo verificou-se rapidamente não ser este o caso, pois o método somente convergia caso a solução admitida estivesse muito próxima da solução real. Em vista disso, partiu-se para uma análise de quais circunstâncias garantiriam a convergência, e, ao mesmo tempo, uma busca exaustiva por soluções que estivessem suficientemente próximas da solução real.

O principal obstáculo é obter algoritmos de busca rápida, uma vez que ela pode ser bastante lenta, principalmente se a região de convergência for pequena. Desconhece-se, ainda, qual é o tamanho desta região para cada um dos parâmetros de otimização.

O algoritmo foi desenvolvido inicialmente na linguagem C, utilizando-se o compilador *lccwin*, por possuir uma interface mais compacta, ideal para o desenvolvimento de pequenos algoritmos. É uma ferramenta padrão ANSI C (não compila programas em C++) disponível gratuitamente para uso não-comercial.

Em seguida, foi testada sua implementação na linguagem C++, utilizando o compilador *Microsoft Visual C++ 6.0* (Microsoft Corporation, 2005), por ser indicada para a utilização de bibliotecas *OpenGl* para o desenvolvimento da parte gráfica.

Atualmente foi feita uma nova versão em C#, utilizando o compilador Microsoft *Visual C# Express 2005* (Microsoft Corporation, 2005), porque é uma linguagem mais nova e mais fácil de utilizar por ser orientada a objetos, além de ser mais fácil encontrar e solucionar erros em suas linhas de código.

3.1 – Estrutura do programa

Nas primeiras correções, as coordenadas dos pontos no espaço foram geradas aleatoriamente, portanto sem a interação com o usuário. O algoritmo para geração destes pontos também gerava as coordenadas bidimensionais das imagens, admitindo-se para isso rotações e translações para cada câmera, além da eliminação (também aleatoriamente) de alguns destes pontos.

Um vetor adicional K_i^k foi implementado para informa se o ponto P_i^k pode ser ou não observado na imagem *k*. O programa conta ainda com estruturas destinadas a realizar cálculos com matrizes (*matrices*) e funções para as matrizes de rotação (*attaux*). As seguintes funções já se encontram implementadas:

- *f_init*: define todos os vetores e matrizes a serem utilizados e atribui valores aos vetores e matrizes do sistema de referência da primeira imagem. Atribui também valores iniciais de rotação e translação aos demais sistemas.

- *gerapontos*: função com finalidade de gerar um conjunto de pontos no espaço para serem determinados por meio do algoritmo. Embora as coordenadas destes pontos sejam conhecidas, elas não serão utilizadas nos processamentos, mas tão somente suas projeções nas diversas imagens. A convergência do algoritmo, porém, é determinada com base na diferença entre as coordenadas os pontos estimados e os reais..

- *prod_transpose*: recebe dois vetores e realiza o produto do primeiro pelo transposto do segundo, resultando em uma matriz quadrada de dimensão 3.

Foi também implementado o cálculo da função de minimização *f*, bem como funções para cálculo das derivadas (analíticas) desta função com relação aos parâmetros a serem determinados, como os ângulos α_k β_k e θ_k, a translação h_k, u_k, v_k, e a profundidade z_{ki} dos pontos.

Por fim, implementou-se um laço onde todos os parâmetros foram variados dentro de certos limites (exceto o cálculo de S_i e z_{ki}, feitos analiticamente), com incrementos previamente definidos, buscando, dentre todas as combinações, aquela que apresentasse o menor valor para a função de minimização *f*.

Buscou-se a otimização do laço dos parâmetros, visando reduzir o excessivo tempo de processamento demandado. Como exemplo, se o problema fosse composto de duas imagens apenas e considerando-se que seriam necessários 100 pontos para cada parâmetro a ser estimado, dentre α_k β_k, θ_k, h_k, u_k e v_k, então seria necessário calcular testar a função de minimização em 10^{12} pontos. Percebe-se, portanto, que mesmo com os computadores atuais esta tarefa é bastante demorada, o que viabiliza a busca por soluções otimizadas e algoritmos mais velozes.

Continuando as correções, foram feitas várias análises no loop de cálculos implementado. Inicialmente foram verificados os parâmetros de rotação e translação dos pontos e, em seguida, se os valores das coordenadas dos pontos estimados estariam bem próximos ao das coordenadas dos pontos reais, o que não aconteceu. Acreditou-se que um dos possíveis motivos de os valores não estarem corretos poderia ser o intervalo das interações do loop. Para fins de teste, os cálculos de rotação e translação feitos na classe de geração aleatória dos pontos foram colocados para fora da função e foram definidos valores fixos de intervalo no loop para alfa, beta e teta (inicial, final e seu passo).

Posteriormente, esses parâmetros fixos com certo intervalo entre seu início e fim (por exemplo, a variação de alfa de 0 a 360) receberam os valores corretos dos parâmetros dos pontos reais (feito com base na classe da geração aleatória dos pontos), pois, fixando cada parâmetro por vez com seu valor real, seria possível o teste nos outros parâmetros (fixando o valor de alfa, por exemplo, era possível testar o beta, dado que foi verificado no método de otimização empregado anteriormente que com somente um valor duvidoso o programa continuava a calcular corretamente). Após estes testes, o intervalo do passo seria aumentado aos poucos, forçando o loop a achar valores corretos, ou seja, até os resultados finais ficarem próximos aos da classe da geração de pontos. Posteriormente este passo seria variado a fim de se detectar onde o erro ocorre.

Os valores das coordenadas dos pontos reais e dos pontos estimados foram impressos em um arquivo "*.txt" e a partir daí puderam ser visualizados no espaço vetorial utilizando o programa *Grapher 1.06* (Golden Software Inc., 1992). Isto tornou mais clara a análise do algoritmo. No entanto, como os pontos eram gerados aleatoriamente, a visualização de um "objeto" era confusa, como um rabisco. A solução adotada foi criar um vetor com valores definidos às coordenadas dos pontos, criando assim uma melhor visualização de um "objeto" na tela. Para fins de teste, adotaram-se duas "imagens", ou seja, dois conjuntos de pontos reais e seus respectivos conjuntos de pontos estimados.

Assim como para as coordenadas dos pontos, foram definidos valores aos parâmetros de rotação e translação. Os valores aplicados aos parâmetros funcionaram, mas foi notado um erro na escala dos pontos estimados da segunda imagem. Detectou-se um possível erro nos valores dos parâmetros de translação *uk*, *vk* e *hk* que, no fim do loop, não atualizavam seus valores como deveriam, possivelmente por um erro no próprio compilador lcc.

Estes erros nos parâmetros de translação foram analisados, tanto na hipótese de erro no algoritmo quanto de erro do compilador. Era necessário que os valores de *uk*, *vk* e *hk* saíssem do loop valendo 0, 0 e 1, respectivamente, e o valor da função o mais baixo possível. Foi feita uma verificação de quanto a função f valia em cada uma das duas imagens, sendo que em uma delas este valor teria que ser o mais próximo possível de 0, o que também não aconteceu da maneira esperada.

Foram mantidos os valores de *uk* e *vk* fixos, variando somente o *hk*, com um passo de incremento menor (0,005) e feita então a plotagem dos pontos por meio de outro arquivo ".dat", onde eram impressos os valores de *f* e *hk*. Mais tarde, foram impressos outros arquivos ".dat", para cada um dos valores e o *f*, onde os outros valores que não estavam sendo utilizados no dado cálculo eram fixados. Como ainda assim os gráficos resultantes não foram os esperados, pensou-se em um possível erro em

conversão de unidade em alguma das funções de cálculo, já que o loop aparentemente estava correto.

A partir desta suspeita, foi recuperado o algoritmo anterior à inserção do loop, onde as classes não haviam ainda sofrido modificações, e feito um novo loop, de forma que fossem feitos apenas cálculos mais limitados, variando somente um parâmetro de rotação em cada teste. Se todos os parâmetros estivessem corretos, o valor do parâmetro em teste teria que ser igual ao seu valor real.

Se a função de cálculo dos pontos estimados (*Si*) e/ou a função de cálculo da profundidade dos pontos (*Zki*) estivessem com problemas, não seria possível fazer o teste, pois os gráficos dependem diretamente dessas duas funções.

Como mais uma vez os resultados não foram os esperados, decidiu-se iniciar o projeto novamente, apenas re-utilizando as fórmulas de cálculo que, pelos testes iniciais, estão corretas.

CAPÍTULO 4

CONCLUSÕES

Na presente fase final de desenvolvimento, considera-se que os objetivos no que diz respeito à familiarização com as formulações e métodos matemáticos e compreensão do trabalho em si foram alcançados. Infelizmente, não foi possível concluir em tempo hábil os últimos testes e uma melhor reformulação do algoritmo, que ainda apresenta erros nos valores alcançados.

Este trabalho ainda deixa lugar a uma possível continuação, a fim de aprimorar o algoritmo e desenvolver uma interface gráfica, com fornecimento de imagens e entrada de dados, ou os pontos estimados, passados pelo usuário.

REFERÊNCIAS BIBLIOGRÁFICAS

Debevec, P. E.; Modeling and Rendering Architecture from Photographs. Dissertation. UC at Berkeley. Disponível em www.debevec.org/Thesis/debevec-phdthesis-1996.pdf, 1996

Debevec, P. E.; Taylor, C. J.; Malik, J.; Modeling and Rendering Architecture from Photographs: A hybrid geometry- and image-based approach. *Proceedings of the 23rd SIGGRAPH Conference*, p. 11-20, 1996.

Raposo, A. B.; Szenberg, F.; Gattass, M.; Celes, W.; Visão Estereoscópica, Realidade Virtual, Realidade Aumentada e Colaboração, *Anais do XXIV Congresso da Sociedade Brasileira de Computação*, v. 2, , cap. 7, XXIII JAI - Livro Texto, SBC, Brasil, 2004. Disponível em www.tecgraf.puc-rio.br/publications/artigo_2004_visao_estereoscopica_realidade_virtual.pdf.

Boldrini, J.L., Costa, S.I.R., Figueiredo, V.L., Wetzler, H.G.; Álgebra Linear, 3ª edição. Ed. Harbra, 1980.

Compilador LCC-Win32. Disponível em *www.cs.virginia.edu/~lcc-win32/index.html* ou em *www.q-software-solutions.com/lccwin32/*

Viewpoint Corporation. www.metacreations.com/products/canoma/, 2000.

Simple 3D. www.simple3d.com/, 1999.

Compilador Microsoft Visual C++ 6.0. http://msdn2.microsoft.com/pt-br/visualc/default.aspx

Compilador Microsoft Visual C# Express 2005. Disponível em http://msdn.microsoft.com/vstudio/express/visualcsharp/)

Golden Software Inc. www.goldensoftware.com/products/grapher/grapher.shtmlR, 1992.

APÊNDICE A

ALGORITMOS DOS CÁLCULOS

A.1 – Cálculo da função de minimização

Dado pela seguinte fórmula:

$$f = \min \sum_{k=1}^{l} \left[\sum_{i=1}^{n_k} (z_{ki}\ A_k\ Q_{ki} - A_k\ T_k - S_i)^2 \right]$$

O seguinte algoritmo:

```
for (k = 0; k < nimag; k++){
   for (i = 0; i < npon; i++){
      if (ki_ex[k][i] > 0) {
         deltap = Ak[k]*(zki[k][i]*Qki[k][i] - Tk[k]) - Si[i];
         deltap._1, deltap._2, deltap._3, Si[i]._1, Si[i]._2, Si[i]._3);
         f += deltap*deltap;
      }
   }
}
```

A.2 – Cálculo da função de pontos estimados (*Si*)

Dado pela seguinte fórmula:

$$S_i = \frac{1}{l}\left[I_3 - \frac{1}{l}\sum_{k=1}^{l}\left(\frac{A_k\ Q_{ki}\ Q_{ki}^T\ A_k^T}{Q_{ki}^T\ Q_{ki}} \right) \right]^{-1} \sum_{k=1}^{l} A_k \left(\frac{Q_{ki}^T\ T_k}{Q_{ki}^T\ Q_{ki}} Q_{ki} - T_k \right)$$

O seguinte algoritmo:

```
for (i = 0; i < npon; i++) { // loop dos pontos Si
   deltaq._1 = 0.;
   deltaq._2 = 0.;
   deltaq._3 = 0.;
   aqqa = identity(0.);
   j = 0;
   for (k = 0; k < nimag; k++) { // loop dos pontos Qki
      if (ki_ex[k][i] > 0) {
         Q_ki = Qki[k][i];
```

```
        QtQ = Q_ki*Q_ki;
        deltaq += Ak[k]*((Q_ki*Tk[k]/(QtQ))*Q_ki - Tk[k]);
        auxi = Ak[k]*Q_ki;
        aqqa += prod_transpose(auxi, auxi)/(QtQ);
        j++;
      }
    }
    auxi = (1./j) * inverse(I3 - aqqa/j) * deltaq;
    Pi[i] = auxi;
    soma += auxi;
}

soma = soma/npon;
for (i = 0; i < npon; i++) {
   Pi[i] -= soma;
}
```

A.3 – Cálculo da função de profundidade dos pontos (*Zki*)

Dado pela seguinte fórmula:

$$z_{ki} = \frac{Q_{ki}^T \, [A_k^T \; S_i + T_k]}{Q_{ki}^T \; Q_{ki}}$$

O seguinte algoritmo:

```
for (i = 0; i < npon; i++) {  // loop dos pontos Si
   for (k = 0; k < nimag; k++) {    // loop dos pontos Qki
      if (ki_ex[k][i] > 0) {
         Q_ki = Qki[k][i];
         QtQ = Q_ki*Q_ki;
         z_ki[k][i] = Q_ki*(transpose(Ak[k])*Si[i] + Tk[k])/QtQ;
       }
    }
}
```

A.4 – Cálculo da função de translação (*Tk*)

Dado pela seguinte fórmula:

$$T_k = \frac{1}{n_k} \sum_{i=1}^{n_k} A_k^T \; (z_{ki} \; A_k \; Q_{ki} - S_i) = \frac{1}{n_k} \sum_{i=1}^{n_k} (z_{ki} \; Q_{ki} - A_k^T \; S_i)$$

O seguinte algoritmo:

```
for (k = 1; k < nimag; k++) { // loop dos pontos Qki
   for (i = 0; i < npon; i++) { // loop dos pontos Si
      if (ki_ex[k][i] > 0) {
```

```
        auxi += zki[k][i]*Qki[k][i] -  transpose(Ak[k])*Si[i];
        nk++;
      }
   }
   T_k[k] = auxi/nk;
}
```

A.5 – Cálculo da derivada da função de minimização com relação ao ângulo alfa

Dado pela seguinte fórmula:

$$\tan\alpha_k = \frac{\operatorname{sen}\alpha_k}{\cos\alpha_k} = \frac{\sum_{i=1}^{n_k}(r_{ix}w_{iz} - r_{iz}w_{ix})}{\sum_{i=1}^{n_k}(r_{ix}w_{ix} + r_{iz}w_{iz})}$$

O seguinte algoritmo:

```
for (k = 1; k < nimag; k++) { // loop das imagens
   sian = 0;
   coan = 0;
   for (i = 0; i < npon; i++) { // loop dos pontos
      if (ki_ex[k][i] > 0) {
         Ri = Rbeta[k]*Rteta[k]*(zki[k][i]*Qki[k][i] - Tk[k]);
         sian += Ri._1*Si[i]._3 - Ri._3*Si[i]._1;
         coan += Ri._1*Si[i]._1 + Ri._3*Si[i]._3;
      }
   }
   alfa_k[k] = atan2(sian, coan);
}
```

A.6 – Cálculo da derivada da função de minimização com relação ao ângulo beta

Dado pela seguinte fórmula:

$$\tan\beta_k = \frac{\operatorname{sen}\beta_k}{\cos\beta_k} = \frac{\sum_{i=1}^{n_k}(r_{iz}w_{iy} - r_{iy}w_{iz})}{\sum_{i=1}^{n_k}(r_{iy}w_{iy} + r_{iz}w_{iz})}$$

O seguinte algoritmo:

```
for (k = 1; k < nimag; k++) { // loop das imagens
   sian = 0;
   coan = 0;
   for (i = 0; i < npon; i++) { // loop dos pontos
      if (ki_ex[k][i] > 0) {
```

```
        Ri = Rteta[k]*(zki[k][i]*Qki[k][i] - Tk[k]);
        Wi = transpose(Ralfa[k])*Si[i];
        sian += Ri._3*Wi._2 - Ri._2*Wi._3;
        coan += Ri._2*Wi._2 + Ri._3*Wi._3;
      }
   }
   beta_k[k] = atan2(sian, coan);
}
```

A.7 – Cálculo da derivada da função de minimização com relação ao ângulo teta

Dado pela seguinte fórmula:

$$\tan\theta_k = \frac{\operatorname{sen}\theta_k}{\cos\theta_k} = \frac{\sum_{i=1}^{n_k}(r_{iy}w_{ix} - r_{ix}w_{iy})}{\sum_{i=1}^{n_k}(r_{ix}w_{ix} + r_{iy}w_{iy})}$$

O seguinte algoritmo:

```
for (k = 1; k < nimag; k++) { // loop das imagens
   sian = 0;
   coan = 0;
   for (i = 0; i < npon; i++) { // loop dos pontos
      if (ki_ex[k][i] > 0) {
         Ri = zki[k][i]*Qki[k][i] - Tk[k];
         Wi = transpose(Ralfa[k]*Rbeta[k])*Si[i];
         sian += Ri._2*Wi._1 - Ri._1*Wi._2;
         coan += Ri._1*Wi._1 + Ri._2*Wi._2;
      }
   }
   teta_k[k] = atan2(sian, coan);
}
```

A.8 – Cálculo da matriz de rotação

Dado pela seguinte fórmula:

$$A_k = R_y(\alpha_k)\, R_x(\beta_k)\, R_z(\theta_k)$$

O seguinte algoritmo:

```
for (k = 1; k < nimag; k++) { // loop das imagens
   Ak[k] = Ralfa[k]*Rbeta[k]*Rteta[k];
}
```

Sendo que *Ralfa, Rbeta* e *Rteta* são calculados, respectivamente, por:

$$R_y(\alpha_k) = \begin{pmatrix} \cos\alpha_k & 0 & -\text{sen}\,\alpha_k \\ 0 & 1 & 0 \\ \text{sen}\,\alpha_k & 0 & \cos\alpha_k \end{pmatrix},$$

$$R_x(\beta_k) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\beta_k & \text{sen}\,\beta_k \\ 0 & -\text{sen}\,\beta_k & \cos\beta_k \end{pmatrix}$$

e

$$R_z(\theta_k) = \begin{pmatrix} \cos\theta_k & \text{sen}\,\theta_k & 0 \\ -\text{sen}\,\theta_k & \cos\theta_k & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Usando os algoritmos:

- Ralfa:

```
for (k = 1; k < nimag; k++) { // loop das imagens
   Ralfa[k] = rotmay(alfa_ang[k]);
}
```

- Rbeta:

```
for (k = 1; k < nimag; k++) { // loop das imagens
   Rbeta[k] = rotmax(beta_ang[k]);
}
```

- Rteta:

```
for (k = 1; k < nimag; k++) { // loop das imagens
   Rteta[k] = rotmaz(teta_ang[k]);
}
```