ALGORITMO DE AUXÍLIO DE PERCEPÇÃO DE OBJETOS PARA ROBÔS DE SERVIÇO: USO DE HIPÓTESES DE CONTATO ENTRE O OBJETO E MÃO ROBÓTICA

Márcio Henrique Diniz Marques

Dissertação apresentada à Escola de Engenharia de São Carlos da Universidade de São Paulo, como parte dos requisitos para a obtenção do Título de Mestre em Engenharia Mecânica.

Orientador: Prof. Dr. Glauco Augusto de Paula Caurin

São Carlos
2008

Ficha catalográfica preparada pela Seção de Tratamento
da Informação do Serviço de Biblioteca – EESC/USP

M357a

Marques, Márcio Henrique Diniz
Algoritmo de auxílio de percepção de objetos para robôs de serviço : uso de hipóteses de contato entre o objeto e mão robótica / Márcio Henrique Diniz Marques ; orientador Glauco Augusto de Paula Caurin. -- São Carlos, 2008.

Dissertação (Mestrado-Programa de Pós-Graduação em Engenharia Mecânica e Área de Concentração em Dinâmica das Máquinas e Sistemas) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2008.

1. Robótica - Robôs de serviço. 2. Manuseio. 3. Tato. 4. Identificação. 5. Mobilidade. 6. Ambientes perigosos. 7. Precisão. I. Título.

FOLHA DE JULGAMENTO

Candidato: Engenheiro **MARCIO HENRIQUE DINIZ MARQUES**

Dissertação defendida e julgada em 25/03/2008 perante a Comissão Julgadora:

Aprovado

Prof. Associado **GLAUCO AUGUSTO DE PAULA CAURIN (Orientador)**
(Escola de Engenharia de São Carlos/USP)

Aprovado

Prof. Dr. **ADRIANO ALMEIDA GONÇALVES SIQUEIRA**
(Escola de Engenharia de São Carlos/USP)

APROVADO

Prof. Dr. **MARCELO BECKER**
(Pontifícia Universidade Católica de Minas Gerais/PUC-MG)

Prof. Associado **JONAS DE CARVALHO**
Coordenador do Programa de Pós-Graduação em
Engenharia Mecânica

Prof. Associado **GERALDO ROBERTO MARTINS DA COSTA**
Presidente da Comissão da Pós-Graduação da EESC

AGRADECIMENTOS

Ao Professor Glauco, que sempre me incentivou e apoiou, pela confiança depositada em mim no desenvolvimento deste trabalho.

Aos meus pais e minhas irmãs, que também sempre me incentivaram muito e apoiaram meu ingresso no programa de mestrado.

Ao meu Tio Luiz Antônio pela grande ajuda na conclusão do trabalho.

Aos meus primos Adriano (pelas dúvidas tiradas em programação... e foram muitas!!) e Paulo Estevão e sua esposa Cíntia, pelo grande apoio nas minhas idas à São Carlos e pelo incentivo dado a mim.

RESUMO

MARQUES, M. H. D. (2008). *ALGORITMO DE AUXÍLIO DE PERCEPÇÃO DE OBJETOS PARA ROBÔS DE SERVIÇO: USO DE HIPÓTESES DE CONTATO ENTRE O OBJETO E MÃO ROBÓTICA.* Dissertação (Mestrado) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2008.

Os robôs de serviço originados primeiramente nos livros e filmes de ficção são, atualmente mais do que fruto da imaginação dos autores de ficção-científica, e sim uma promessa para num futuro próximo auxiliar o homem em questões como: a substituição do homem em ambientes perigosos, insalubres e atividades repetitivas (e tediosas); como auxiliar doméstico atuando como uma extensão inteligente do corpo humano; substituto do homem em caminhadas espaciais, mergulhos em grandes profundidades, ..., etc. No presente trabalho será discutido meios de como refinar atuadores robóticos antropomórficos (mãos) através de comparações entre as distâncias relativas ao sistema de coordenadas da mão e ao sistema de coordenadas do objeto manipulado, de modo que sejam agrupadas em conjuntos de contato que serão tratados para gerar hipóteses de contatos. Com esse recurso os robôs no futuro poderão ter uma maior interação com o meio ambiente, possibilitando identificar objetos empunhados (durante o manuseio) através do tato, dando-lhes assim condições de atuarem no ambiente de trabalho humano com mínimas alterações no ambiente. Assim os robôs poderão ganhar maior mobilidade e funcionalidade para substituir o homem em tarefas perigosas de uma forma mais interativa e precisa nesse tipo de ambiente.

Palavras-chaves: robôs serviçais, manuseio, tato, identificação, mobilidade, ambientes perigosos, precisão.

ABSTRACT

MARQUES, M. H. D. (2008). *ALGORITM ALGORITHM OF OBJECT PERCEPTION FOR SERVICE ROBOTS: CONTACT HYPOTHESES BETWEEN A GRASPED OBJECT AND THE ROBOTIC HAND.* M.Sc. Dissertation – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2008.

Service robots as we know from fiction books and movies, are today, more than a product from the imagination of Sci-Fi authors, instead of this initial concept, they are today a promise for the a future auxiliary for the men in matters like: workers at hazardous environments and repetitive tasks; as domestic auxiliary working together with a human beam as a extension of the men's body; a quick substitute in space walking; deep diving applications, .., etc. In the present work will be discussed how to improve robotics actuators (hands) by studying comparative measurements between distances related to the hand coordinating system and grasped object coordinating system, in order allow the creation of contact sets that will analyzed to generate contact hypothesis. With this feature robots in the future will be able to interact better at the environment where they will work, as well at human environment with minimal changes to receive robots. Therefore robots could perform works with more mobility and functionality in order to replace humans in hazardous tasks counting with accuracy and more interactivity at these environments.

Key-words: service robots, hands, measurements, contact hypothesis, hazardous tasks, interactivity, accuracy.

Lista de Figuras

SUMÁRIO

1 - Introdução

Há aproximadamente 150 anos os autores de ficção científica criaram uma sociedade na qual as máquinas ajudavam os homens enquanto serviçais no trabalho e em casa, substituindo e mimetizando serviços, tais quais serviçais humanos deveriam executar. Ao mesmo tempo, essa idéia impulsionou os engenheiros, que de forma ambiciosa, trabalharam para fazer desses robôs da ficção objetos reais do dia-a-dia. Considerado que nos últimos vinte anos, no mundo técnico da indústria o seguinte panorama se apresenta para a robótica e, claro, para os robôs nos dias atuais: máquinas especializadas podem ser encontradas executando trabalhos repetitivos, tediosos (porém precisos) e muitas vezes insalubres, tornando a qualidade do trabalho a ser executado mais fácil de ser obtida e num tempo menor de execução. No entanto, a visão difundida pelo cinema, pelos resultados de estudos em pesquisas, em mídias eletrônicas e pelos escritores de ficção científica, de robôs auxiliando em tarefas cotidianas, continua impondo maiores desafios tecnológicos aos pesquisadores dessa área, muitos ainda a serem vencidos.

Em contraste com as máquinas industriais, um robô projetado para interagir em um ambiente dominado por humanos não pode ser preparado especialmente para execução de uma simples tarefa e muito menos é desejável que se ajuste o ambiente de trabalho, ou à forma de convivência, às necessidades especiais do robô. Ao invés disso é esperado que o robô seja flexível o suficiente para ajustar-se automaticamente aos requisitos solicitados pela tarefa que estará executando naquele momento. É claro que o homem pode trabalhar nesse ambiente muito bem, pois muitos objetos foram projetados para seu próprio uso, ou a própria evolução humana deu lhe a capacidade de adaptar-se a diversos ambientes e a desenvolver habilidades especializadas para cada nova tarefa. Portanto, não é de se surpreender que, na tarefa de desenvolvimento de um robô para uso geral, são encontradas habilidades humanas simuladas por meio de projetos antropomórficos (figura 1.1).

Figura 1. 1 – Braço + Mão (DLR Hand I)

Podemos citar como exemplo de desenvolvimento a fim de expandir as capacidades de um robô, para locomoção em ambientes não-estruturados, alguns robôs foram equipados com pernas (figura 1.2a). Para execução de ações, mãos antropomórficas foram construídas para tal

(figura 1.2b) e para uma interação mais agradável entre robôs e homens, robôs com faces artificiais foram desenvolvidos (figura 1.3a). Alguns laboratórios ou empresas tem se empenhado no desenvolvimento de humanóides completos (figura 1.3b).

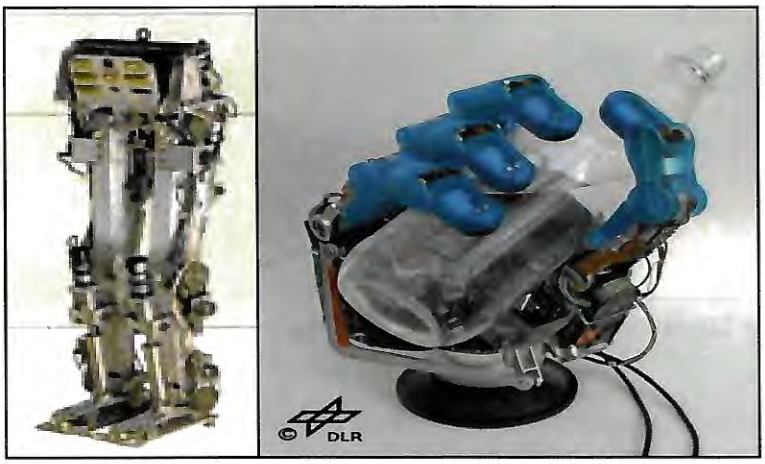

Figura 1.2 a– Projetos de pernas **Figura 1.2 b**– Projetos de mãos robóticas

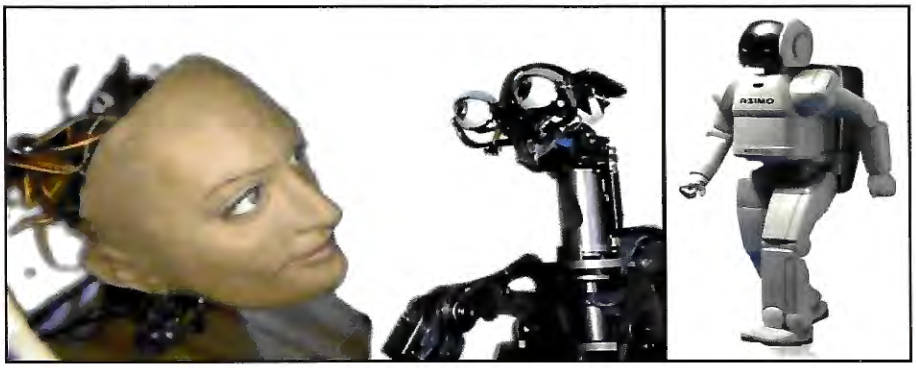

Figura 1.3 a– Simulação de faces **Figura 1.3 b**– Projetos humanóides

Com intenção de levá-los ao caminho visionário dos robôs de serviço, novos conhecimentos foram adquiridos no desenvolvimento de robôs humanóides tornando possível cada vez mais diversas aplicações para as mais variadas tarefas.

Os robôs de serviço como objetos comandados para execução de tarefas do mundo real podem ser divididas em dois grupos:

i. Robôs que podem ser intuitivamente *tele-operados*, ou seja, um operador humano usa o robô para estender suas capacidades ou para atingir lugares de ambientes nocivos, perigosos ou insalubres, já conhecidos previamente como não adequados para seres humanos. Para as finalidades descritas, atualmente (robôs antropomórficos ou não), já se encontram em uso (figura 1.4a e 1.4b).

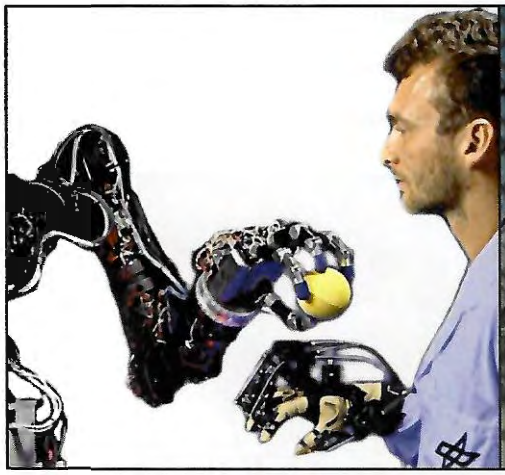

Figura 1.4 a – Tele-operação (DLR Hand I)

Figura 1.4 b – Robô tele-operado de limpeza (Louvre)

ii. Robôs antropomórficos ou não que podem ser operados de forma *autônoma*, ou seja, o robô recebe ordens verbais e automaticamente converte (decompõe) essas ordens em tarefas individuais e depois cumpre o trabalho desejado sem nenhuma intervenção humana adicional. Esse robô é, até agora, uma promessa e tópico de intensa pesquisa. Somente alguns usos comerciais de aplicações simples trouxeram ao mercado, como exemplo dessas utilizações podemos citar máquinas de limpeza de estações de trens, aspiradores de pó ou cortadores de grama.

O trabalho de pesquisa não privilegiou estudar o largo campo dos robôs antropomórficos como máquinas completas, mas concentrou-se no interesse em um específico subsistema desse equipamento, a mão. Em geral, a função dos robôs humanóides é de explorar ou manipular objetos dentro do ambiente no qual ele foi inserido. Os seres humanos usam suas mãos para esse propósito, o que na maioria dos casos se mostra uma ferramenta apropriada, pois a mão humana é, depois da face, a região do corpo humano mais sensível [12]. Ela pode ser, devido a isso, perfeitamente capaz de ser empregada para examinar objetos de forma minuciosa e precisa. Ressalta-se que, com seus 27 ossos e 20 músculos, ela é capaz de realizar movimentos complexos e é, portanto, bem adaptada para manipular objetos suavemente. Logo, não é de se surpreender que no desejo de construir um robô antropomórfico, os pesquisadores direcionaram seus interesses mais especificamente para o desenvolvimento da assim chamada "mão antropomórfica habilidosa". Para os robôs essas mãos são ferramentas mais flexíveis utilizadas para se interagir em um ambiente antes unicamente humano.

1.1 - Motivação

No campo dos robôs de serviço, atividades simples do dia-a-dia humano podem se tornar extremamente trabalhosas quando no desenvolvimento de um sistema robótico de um robô de serviço. Dentre os requisitos desse tipo de robô, destacam-se alguns de fundamental importância, como, o de movimentar-se em um ambiente humano, o de localizar objetos de interesse e manipulá-los. Porém, durante as fases finais de aproximação da mão robótica com o objeto de interesse, este último é subtraído da visão do robô, obstruído pela sua própria mão. Assim sendo são necessários recursos adicionais para fornecer maiores informações para que o sistema robótico como um todo entenda melhor o ambiente a sua volta e possa completar as tarefas solicitadas de maneira satisfatória.

1.2 - Objetivos

O objetivo deste trabalho é o de apresentar uma abordagem, focada em um manipulador robótico antropomórfico, onde o robô deverá identificar os objetos manipulados, em simulações computacionais, tal qual, metaforicamente, uma pessoa cega reconhece um objeto manuseando-o ou explorando-o através do tato. Assim, o robô que trabalhar com esse algoritmo terá condições de fazer um ajuste fino do posicionamento e da orientação do objeto que está manipulando (o objeto de interesse). Esta capacidade é particularmente importante quando o manipulador encontra-se nas fases finais de aproximação para empunhar o objeto de interesse, quando na maioria dos casos o objeto é obstruído do campo de visão (das câmeras do robô) pelo próprio manipulador robótico, como mostrado na seqüência de figuras 1.5, 1.6, 1.7 e 1.8. O mesmo acontece aos humanos, que tem o objeto obstruído pela mão.

Figura 1. 5 – Aproximação final

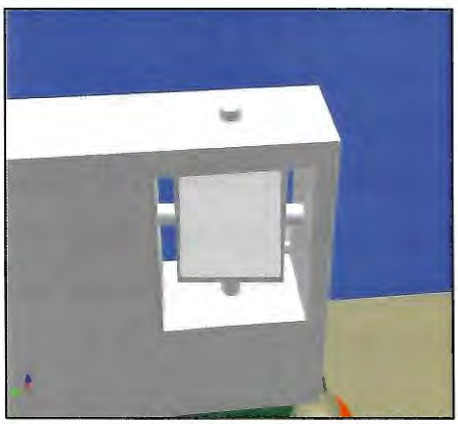

Figura 1. 6 – Vista traseira

Figura 1. 7 – Vista câmera direita obstrução parcial

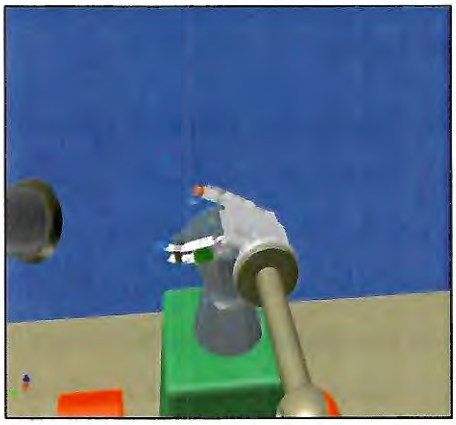

Figura 1. 8 – Obstrução quase total do objeto

Pode-se observar igualmente que, quando da manipulação de objetos, os dedos tocam as faces ocultas (figuras 1.9 e 1.10) do objeto que está dentro do campo de visão e mesmo assim, devido à experiência prévia com a fixação e manipulação de vários objetos de diferentes formas e tamanhos, os seres humanos conseguem identificar e realizar tarefas com um objeto mesmo que este seja desconhecido.

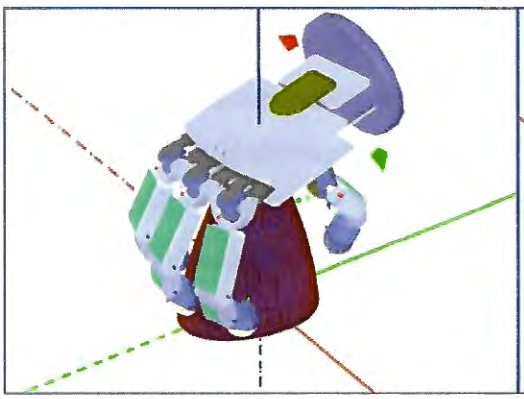

Figura 1. 9 – A questão das faces ocultas (frente)

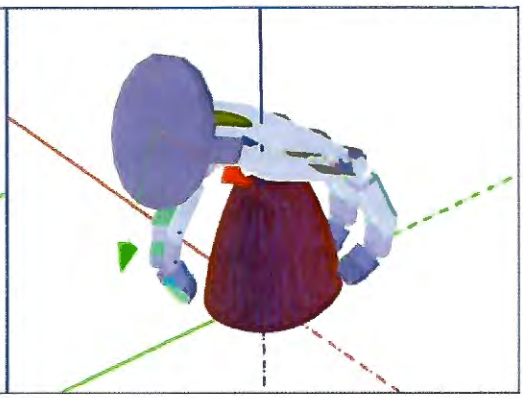

Figura 1. 10 – A questão das faces ocultas (traseira)

Há casos em que os humanos podem, por meio de uma exploração tátil, reconhecer um objeto pelo toque e até descobrir e melhorar a forma como este é manipulado. Porém, neste trabalho, somente será abordada a operação de objetos já conhecidos via contato com o manipulador, supondo que esses foram previamente reconhecidos (em forma e tipo de objeto, por exemplo, copo, talher ou cafeteira) mediante outros sistemas de reconhecimentos, como o sistema de visão.

1.3 - Resumo dos Capítulos

Neste tópico será descrito a divisão deste trabalho para maior facilidade e entendimento.

Capítulo 1 – É o capitulo introdutório, onde é abordada a questão sobre as duas vertentes dos robôs atuais: Os robôs pré-programados para tarefas repetitivas e os robôs programados para uma interação mais rica com o ambiente a sua volta. A motivação e os objetivos serão apresentados no capítulo 1.

Capítulo 2 – Este capítulo conduzirá o leitor às definições do que seriam robôs de serviço (citando exemplos), a conceitos de mãos mecânicas (sejam próteses e de robóticas), discorrerá sobre os tipos de sensores mais freqüentes usados por robôs de serviço e, por fim, serão apresentadas algumas definições sobre os tipos de empunhaduras (*grasp*) mais comuns.

Capítulo 3 – Serão apresentados os *softwares* utilizados neste trabalho e o fluxo de trabalho utilizando os aplicativos.

Capítulo 4 – Serão relatadas as metodologias usadas, definições, ferramentas matemáticas e a forma como foram implementados funções e procedimentos em linguagem C/C^{++}. Será apresentado o procedimento para gerar as hipóteses de contatos através da análise de conjuntos de contatos.

Capítulo 5 – Mostra testes e resultados alcançados usando os aplicativos gerados neste trabalho.

Capítulo 6 – Conclusões.

2 - Retrospecto Relacionado à Pesquisa

2.1- Robôs de Serviço

Define-se como Robô de Serviço um robô apto a executar as tarefas realizadas por seres humanos, de forma que possa auxiliá-los como um complemento de sua extensão corporal e de sua vontade. Estes robôs também seriam empregados para trabalhos em ambientes de alto risco à vida, trabalhos repetitivos sem supervisão humana, funções de busca e exploração, acompanhamento de pessoas idosas e pacientes em repouso, sendo que nos últimos dois exemplos citados o robô somente deverá ser empregado em casos extremos de escassez de pessoas para a execução dessas tarefas, ou quando for mais econômico, mais confiável e mais eficiente na execução dessas tarefas.

Em sua essência, o projeto de um robô de serviço é basicamente composto dos seguintes elementos (figura 2.1):

- Uma base móvel, que dará condições do robô se movimentar no ambiente de trabalho, podendo ser composta de um sistema de rodas para os ambientes mais simples ou de pernas artificiais, caso o ambiente de trabalho apresente um relevo irregular;
- Sistema de localização de direcionamento (GPS, câmeras de vídeo, IMU, laser, etc);
- Sistema de busca e localização de objetos de interesse para realização de tarefas, com câmeras de vídeo, por ultra-som, ..., etc;
- Manipulador (braço ou garra) para execução de tarefas.

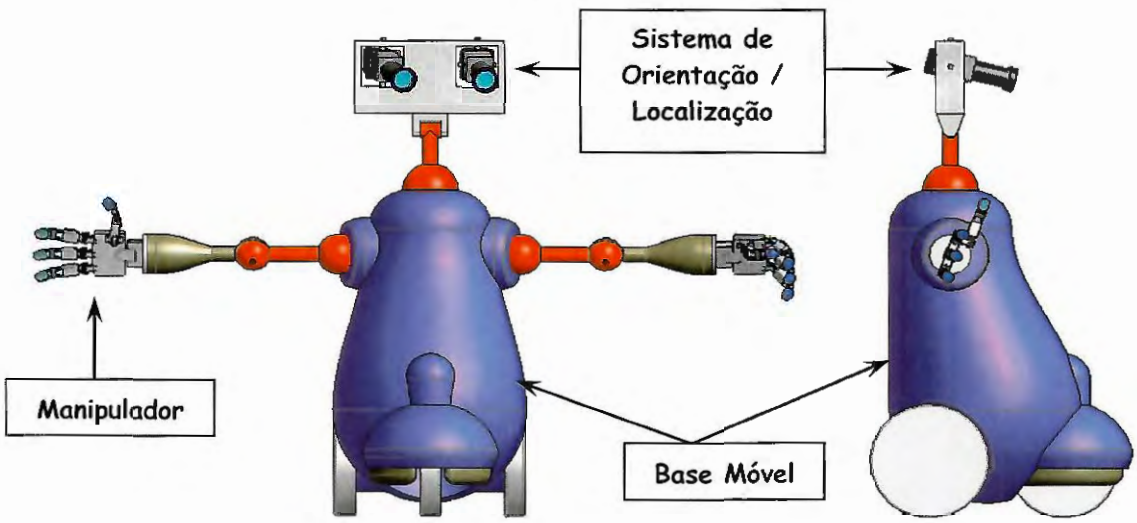

Figura 2. 1 – Descrição básica de um Robô de Serviço.

A partir, dessa idéia conceitual básica para o projeto de um Robô de Serviço é que se irão compor suas funções, baseadas em necessidades e então de posse desses dados, começar-se-á projetá-lo (interface, comportamento, interação com o ambiente e pessoas, hardware e, por fim software) para atender amplamente a função necessária.

A seguir serão apresentados alguns exemplos desse tipo de robô.

Twendy-one – É um de robô simbiótico (figura 2.2a) projetado pela SUGANO LABORATORY [28] para auxiliar um ser humano em suas tarefas diárias (figuras 2.3). Ele conta com sistema de visão estéreo, sistemas de reconhecimento de voz e mãos robóticas (figuras 2.2b) capaz de manipular objetos de vários formatos, de adaptar-se a objetos de rigidez diferentes (regulando a força empregada conforme o objeto reage à força aplicada nele, evitando assim danos aos objetos), como também ajustar-se a movimentos humanos (aperto de mãos ou mãos dadas) e abraçar pessoas de modo a ajuda-los a se levantar ou a deitar-se na cama.

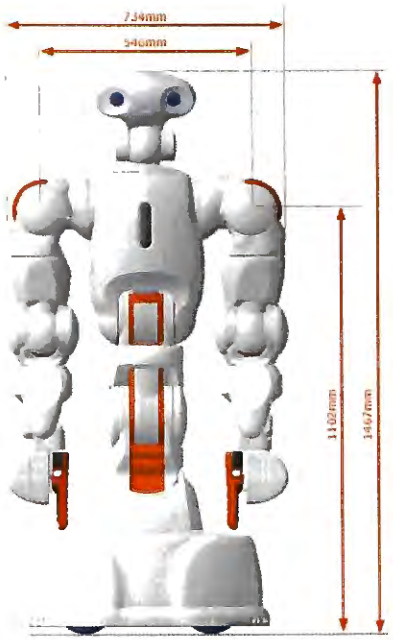

Figura 2.2 a – Twendy-one, dimensões principais

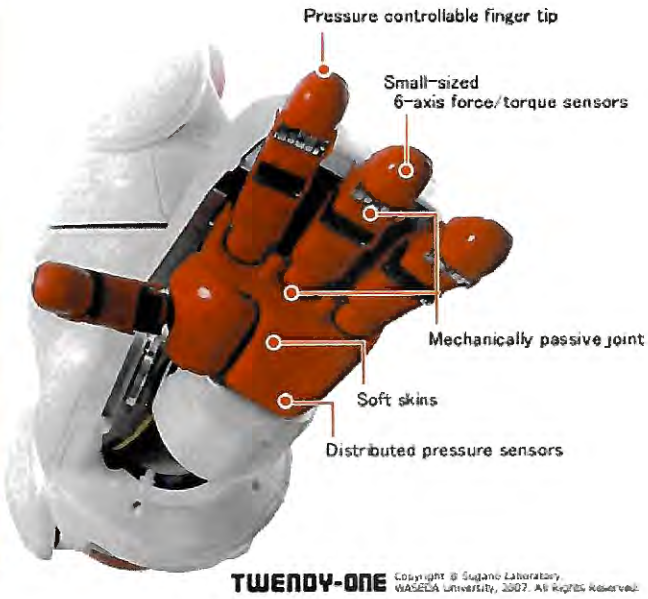

Figura 2.2 b – Twendy-one, detalhe da mão

Figura 2. 3 – Twendy-one em tarefa doméstica

REEM-B – É um robô de serviço (figura 2.4) capaz de caminhar de forma dinâmica (1.5 km/h), reconhecer e agarrar objetos, levantar objetos relativamente pesados (15 quilogramas, o que corresponde a 25% do seu peso), buscar objetos sem intervenção humana dentro do ambiente de trabalho, evitando obstáculos. Construído pela PAL Technology Robotics [20], o REEM-B também obedece a comandos de voz, é capaz de reconhecer faces e de lembrar de compromissos, avisando os usuários com antecedência. Seu lançamento está marcado para abril de 2008.

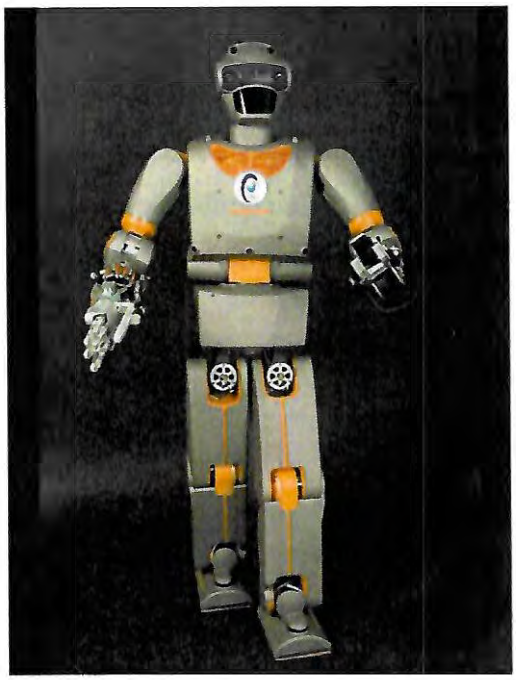

Figura 2. 4 – O robô de serviço REEM-B

2.2- Pesquisa de Garras (atuadores) e Mãos Robóticas

Desde a primeira tentativa de se imitar um ser humano através da criação de um robô humanóide, percebeu-se que a mão humana é, de longe, uma das partes do ser humano mais difícil de ser reproduzida, devido ao alto grau de versatilidade e precisão de movimentos que são executados. Tal versatilidade e flexibilidade de movimentos deve-se ao fato de que as mãos possuem como característica um desenho realmente muito complexo. Cada mão humana, figura 2.5, é composta de 27 ossos e 20 músculos capazes de executar diversos movimentos e, possui a habilidade de adaptar-se facilmente a qualquer ambiente.

Figura 2. 5 – Raio X de uma mão humana.

No caso dos robôs, a pesquisa e o desenvolvimento de mãos robóticas confiáveis (inspirados nas mãos humanas) levaram os pesquisadores a vários conceitos e definições importantes. Como as mãos robóticas são os hardwares bases para o desenvolvimento deste trabalho, serão apresentadas brevemente neste capítulo as definições que envolvem esse tema.

Citando Haidacher [12], como ponto de partida, a maioria das mãos desenvolvidas artificialmente podem ser hoje associadas pelo menos à uma das duas grandes de áreas de aplicações, conforme BUTTERFAß [16], citadas a seguir: próteses de mãos (aplicações médicas) e mãos robóticas.

Não será objeto deste trabalho pesquisa sobre próteses de mãos, e para maiores informações sugere-se consultar s referências [12], [17] e [21].

Para as mãos robóticas, no entanto, podem ser observados dois conceitos principais, como sendo então uma subdivisão a classificação de BUTTERFAß [16], sendo:

- O conceito de projeto *modular* de mãos robóticas, que integra na própria mão os atuadores e a maioria dos circuitos de controles da mão.

- O conceito de projeto *não-modular* de mãos robóticas, no qual, ao contrário do conceito anterior os atuadores e a maioria dos circuitos de controles foram "migrados" para o antebraço, ou outras partes do robô.

No caso dos projetos *modulares*, estes são geralmente maiores (em proporção) que uma mão humana devido à integração de componentes dentro da mão robótica, porém, são facilmente portáveis de um projeto para o outro, ou de uma aplicação para outra, devido ao fato de que seus principais componentes já estarem embarcados na própria mão.

Como exemplo de projetos *modulares*, podemos citar as seguintes mãos robóticas:

- DLR Hand I (ver [15], [10] e [12]) e II (ver [16] e [13]), ambos os projetos têm quatro dedos, cada um com quatro graus de liberdade e sensores de torque em cada junta. Somente a DLR Hand I (figura 2.6) possui sensores táteis nas pontas dos dedos. Apesar das geometrias serem antropomórficas, essas mãos têm um tamanho 1.5 vezes maior que uma mão humana. A DLR Hand II (figura 2.7) pesa 1,8 kg.

Figura 2. 6 – DLR Hand I

Figura 2. 7 – DLR Hand II

- HIT / DLR Hand [11], figuras 2.8 e 2.9, possui um desenho similar ao das mãos DLR Hand I e II, porém esse projeto tem como objetivo reduzir a complexidade dos projetos DLR Hand I e II reduzindo o custo e o tamanho e, aumentar a robustez da mão, comparada com suas antecessoras.

Figura 2. 8 – DLR HIT-HAND (sem cobertura)

Figura 2. 9 – DLR HIT-HAND (com cobertura)

Já para os projetos *não-modulares,* as dimensões das mãos robóticas podem ser reduzidas às dimensões das mãos humanas, uma vez que o número de componentes que compõem a própria mão podem ser reduzidos em número.

Dentre exemplos de projetos nessa área pode-se citar:

- Kanguera [10], projeto que tem como meta reproduzir fielmente a mão humana em uma estrutura similar aos ossos e tendões que a formam (figuras 2.10 e 2.11). Apesar de *não-modular* esse projeto está previsto a implementação, em próximas etapas, de sensores de força e tato embarcados na mão.

Figura 2. 10 – Kanguera, vista geral

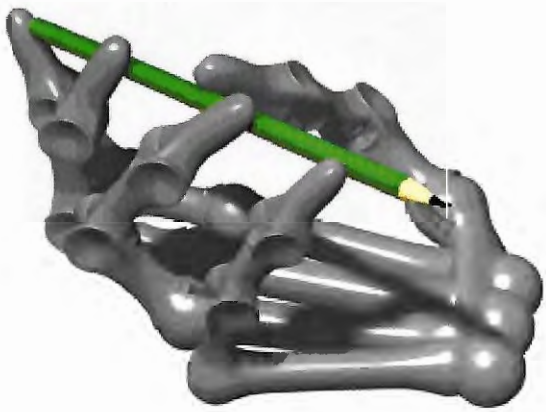

Figura 2. 11 – Kanguera, segurando um lápis

- Utah / MIT Hand [24], mão com quatro dedos e 16 graus de liberdade pneumaticamente acionada. Os atuadores estão localizados no braço robótico que leva a mão (figura 2.12), sendo sua força transmitida por tendões. Para o planejamento de trajetórias (da mão e dedos), devem ser levados em conta mudanças nas tensões dos tendões devido a mudanças na configuração da mão.

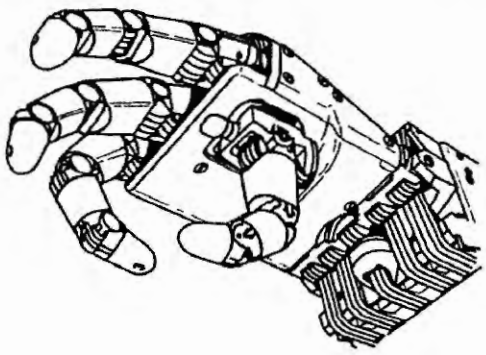

Figura 2. 12 – Utah / MIT Hand

- NASA Hand [3], desenvolvida para o chamado Robonauta (figura 2.13), esta mão foi projetada com o intuito de auxiliar o acesso remoto dos astronautas em missões espaciais. A NASA Hand (figura 2.14) tem cinco dedos e 14 graus de liberdade no total, e seu acionamento está localizado no antebraço. Tal projeto caracteriza-se por uma similaridade de forma bastante pronunciada com a mão humana, fator extremamente útil em situações tele-operadas.

Figura 2. 13 – O Robonauta (NASA)

Figura 2. 14 – A NASA Hand

2.3- Sensores Comumente usados

Qualquer robô de serviço que se preze, deve no mínimo, ter condições de orientar seus membros e corpo de forma adequada ao meio no qual está inserido. Essa tarefa sempre é feita, tanto nos robôs quanto nos humanos, respectivamente através de sensores e sentidos.

Assim como os seres humanos contam com cinco sentidos básicos para a percepção do mundo, a visão, a audição, o tato, o paladar e o olfato os robôs contam igualmente com a visão (embora ainda esta área necessite de muito desenvolvimento) por meio de câmeras de vídeos, com a audição (microfones), com sensores de posicionamentos de juntas (os encoders), sensores de torque, sensores de temperatura e sensores de pressão para certas localidades que poderiam ser chamados de "sensores de tato".

Nesta seção será abordar-se-á a questão da forma como os sensores são empregados para informar o centro de decisões do robô, quais as posições de seus membros e, por fim, a posição de suas ferramentas, mãos, dedos e corpos.

2.2.1- Câmeras de Vídeo

Câmeras de vídeo podem ser usadas pelos robôs de serviço na exploração do ambiente de trabalho ajudando o robô a se orientar (figura 2.15) através de caminhos e rotas (armazenadas em um mapa). As câmeras de vídeos também são empregadas para reconhecer os objetos de interesse (figura 2.16) com os quais o robô deverá interagir para cumprir uma determinada tarefa (figura 2.17). Para as fases finais de aproximação entre o robô e o objeto de interesse, um sistema de visão estéreo (explicado na seção 2.2.1.1) fornece ao robô uma precisão maior para guiar a mão robótica até que esta agarre o objeto.

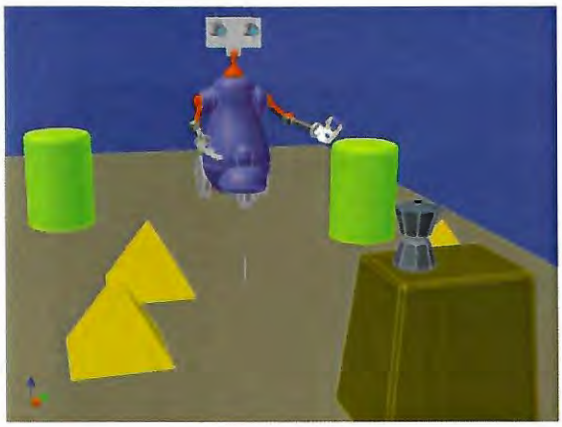

Figura 2. 15– Orientação no ambiente (navegação).

Figura 2. 16– Identificação de tarefa.

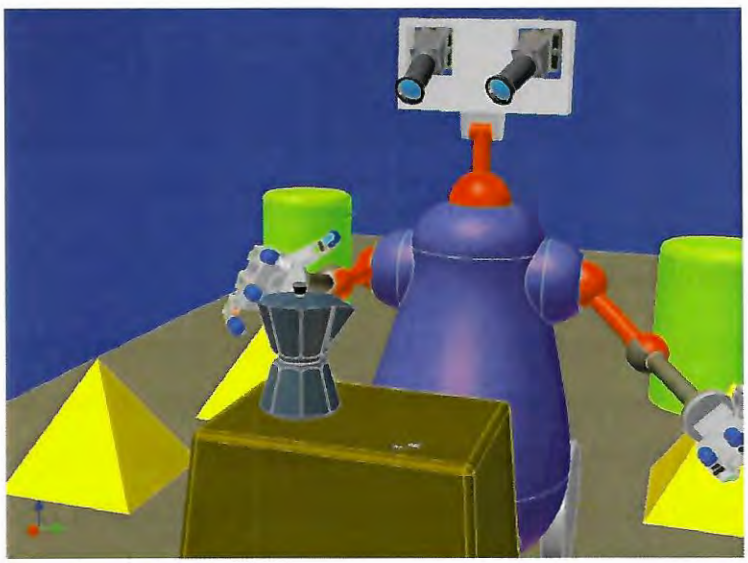

Figura 2. 17 – Apanhar o Objeto de Interesse.

2.2.1.1 - Visão Estéreo

A visão estéreo [30] dos robôs é uma parte do campo da visão computacional muitas vezes usada por robôs móveis para detectar obstáculos. Na visão estéreo duas imagens do ambiente são captadas por duas câmeras separadas (tal qual nosso sistema visual) numa

distância conhecida. Um computador compara as imagens sobrepondo-as para encontrar elementos que se combinam entre as duas imagens. A porção de elementos os quais o computador não conseguiu estabelecer uma relação é chamada de disparidade e é usada para se calcular a distância em que o objeto está do observador.

2.2.2- Microfones

Os microfones são utilizados tanto como interfaces de comando para fornecer tarefas para o robô (reconhecimento de voz) quanto para orientação do robô e reconhecimento de objetos por ecolocalização, à semelhança dos morcegos (figura 2.18).

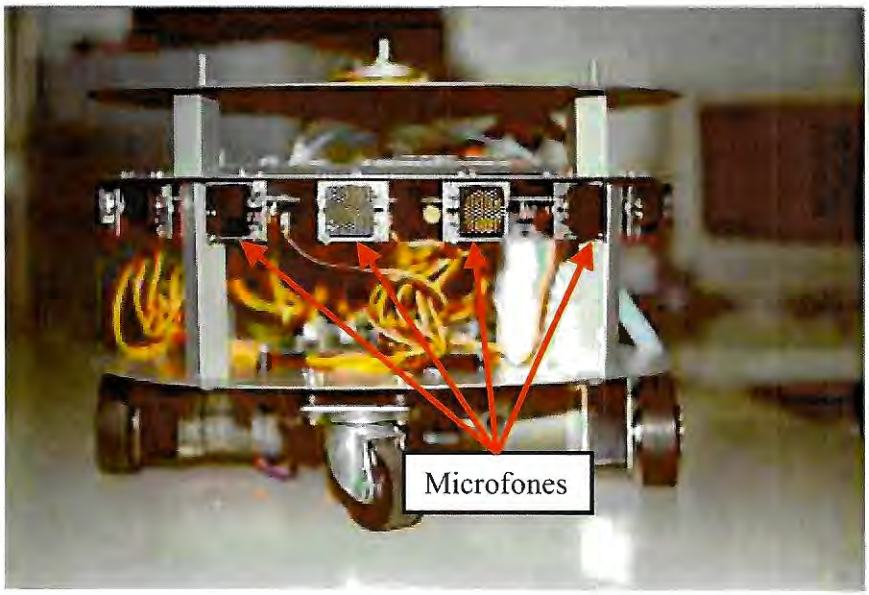

Figura 2. 18 – Robô guiado através do som.

2.2.2.1 - Ecolocalização

Ecolocalização ou Biosonar [31] é um sentido, uma sofisticada capacidade biológica de detectar a forma, posição e distância de objetos (obstáculos no ambiente) ou animais através da emissão de ondas ultra-sônicas (no ar ou na água) e cronometragem do tempo gasto para essas ondas após serem emitidas, refletirem no alvo e voltarem à fonte sobre a forma de eco (ondas refletidas – figura 2.19). É feita também uma análise das ondas refletidas pelo alvo de forma a extrair informações sobre forma e tamanho do objeto. Para diversos mamíferos, morcegos, golfinhos e baleias, essa capacidade é de importância crucial em condições onde a visão é insuficiente, à noite no caso dos morcegos ou em águas escuras ou turvas para os golfinhos, seja para locomoção ou para captura de presas. Alguns pássaros também utilizam a ecolocalização para voarem em cavernas. Baseado nessa capacidade natural os seres humanos desenvolveram a "ecolocalização artificial" criando assim o radar e o sonar, equipamentos

muitas vezes presentes um alguns robôs, auxiliando-os a refinar as informações do ambiente à sua volta e na busca de objetos dentro desse ambiente.

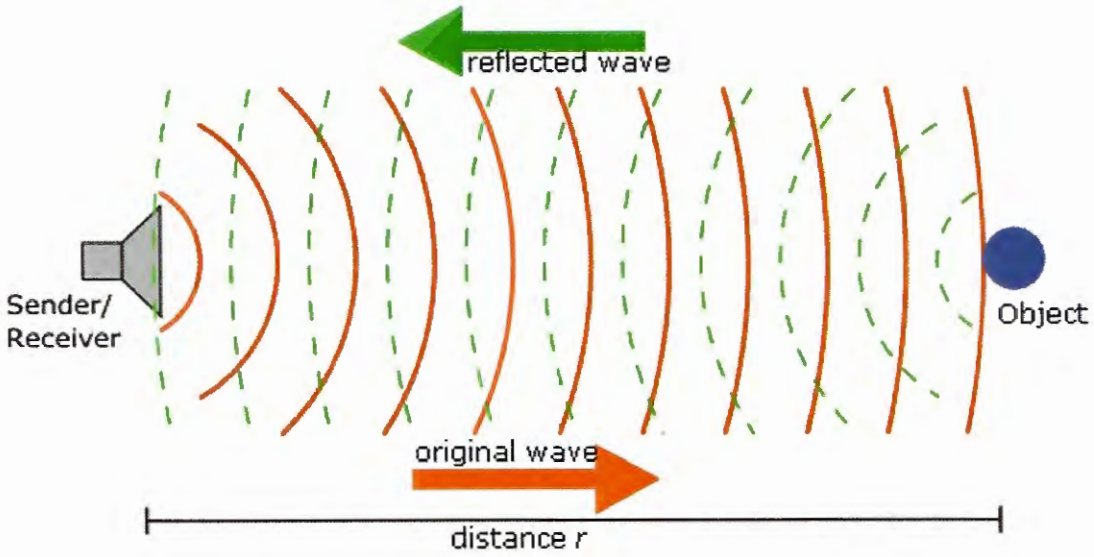

Figura 2. 19 – Princípios do funcionamento da ecolocalização.

2.2.3- Sensores de Torque

Com relação a esses sensores, pode-se também determinar o ponto de contato através de estimativas dos torques aferidos em cada articulação da mão robótica, desde que se tenha um pleno conhecimento dos momentos gerados por cada elemento dessa mão robótica. Obviamente, inúmeros testes deverão ser feitos para calibração, utilizando-se vários objetos conhecidos para estender ao máximo o conhecimento sobre a mão em questão (ver [12], capitulo 3). Neste ponto, uma implementação de uma rede neural pode abreviar o tempo de aprendizagem e maximizar as chances de bons resultados em relação aos pontos de contatos obtidos através desse meio. Nas figuras 2.20a, 2.20b e 2.20c é apresentado um sensor de tamanho reduzido onde foi viabilizada sua implantação (figura 2.21) mediante várias juntas de articulações da mão DLR HAND II.

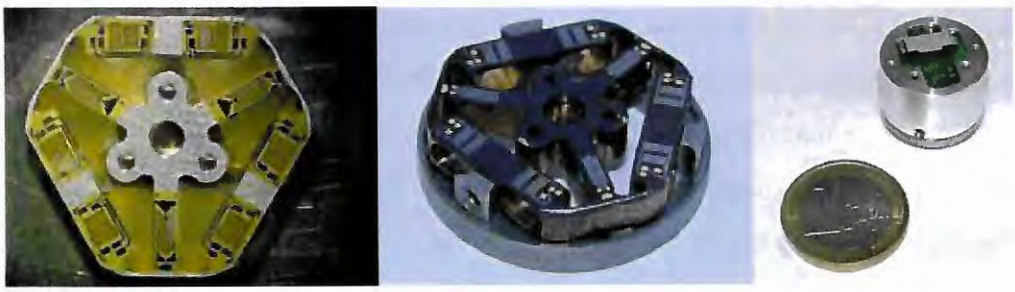

Figura 2.20 a – Sensor de torque (visto de cima),
Figura 2.20 b– Perspectiva,
Figura 2.20 c – Comparação de tamanho com uma moeda.

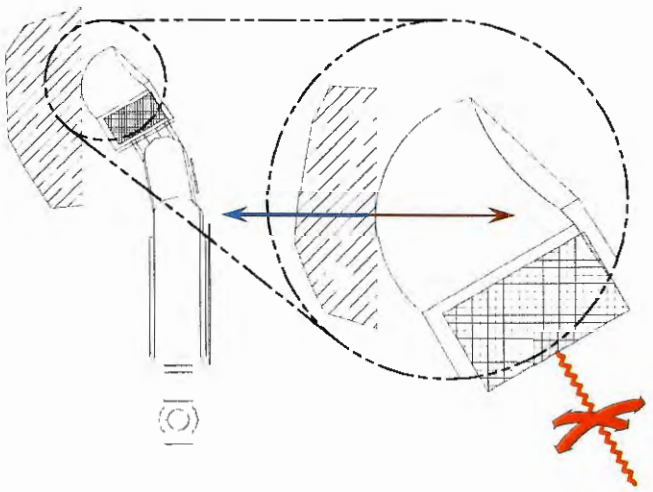

Figura 2. 21 – Esquema mostrando sensor de torque montado na ponta do dedo do robô.

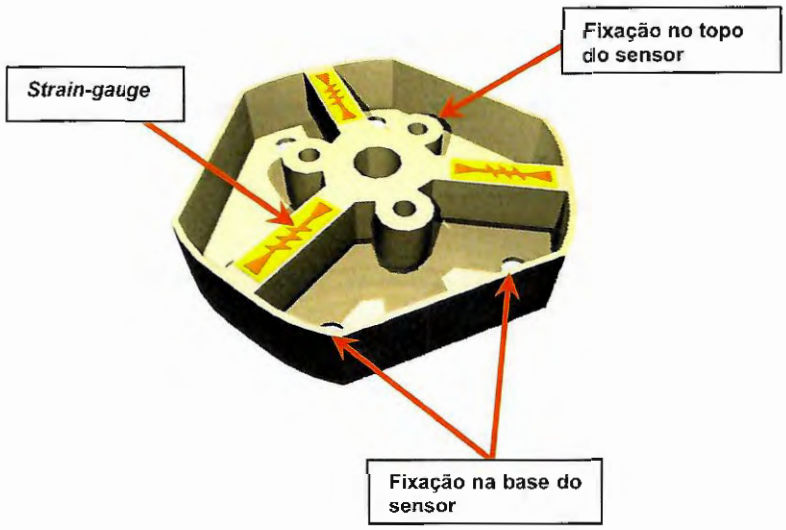

Figura 2. 22 – Principais componentes de um sensor de torque em miniatura similar ao usado por [15].

O funcionamento de um sensor de torque como mostrado nas figuras 2.20a, 2.20b e 2.20c é bastante simples. O sensor é composto de uma base – figura 2.22 – (que é fixada a um elo anterior da cadeia cinemática), três pequenas ligações onde são acondicionadas *strain gauges* (que fornecerão uma resposta elétrica proporcional à deformação dessas ligações) e de uma união de topo flangeada para ligar o sensor ao elo posterior da cadeia cinemática. Desse modo quando um momento é transmitido nas ligações do topo do sensor através das três ligações que unem a base do sensor, as *strain gauges* irão alterar suas resistências elétricas a passagem de corrente que poderão ser medidas e, através análises destes sinais elétricos devido a variação da resistência das *strain gauges* juntamente com o conhecimento prévio das

propriedades mecânicas do corpo do próprio sensor, pode-se então verificar qual a deformação sofrida pelas ligações do sensor e, por fim o torque ao qual este foi submetido.

2.2.4- Sensores de Tato (*Tactile Sensors*)

Pode-se usar sensores de tato para determinar pontos de contatos em mãos robóticas, pois são capazes de indicar forças locais e suas posições. Muito embora esses sensores existiam de forma rica somente em ambientes de laboratórios, atualmente começam a se difundir em escala comercial [29].

Desde que vencidas as dificuldades técnicas (redução de tamanho deste sensor e resolução), esse seria o sensor mais indicado para se conseguir localizar com precisão os pontos de contatos de um objeto empunhado por uma mão robótica.

Como exemplo deste tipo de sensor, o de pressão de malha resistiva, seria provavelmente o melhor candidato para uma localização precisa de um ponto de contato.

Nesse sensor tem-se um substrato onde estão situadas várias regiões condutoras na forma de pontos de anéis (figura 2.23), uma camada de uma espuma condutora (cuja condutividade dependerá da pressão aplicada à espuma) e uma superfície protetora e isolante para se evitar danos à "pele artificial".

Em cada ponto circunscrito em um anel, tem-se uma ligação elétrica conectada a uma matriz de endereçamento (figura 2.24). A espuma condutora, no entanto, somente apresentará boa condutividade quando sofrer uma pressão de compressão, permitindo então que se tenha passagem elétrica entre um ponto e um anel (note que o ponto e o anel não são ligados entre si sob o substrato), enviando um sinal à matriz de endereçamento que, por sua vez, enviará o dado a um programa que informará ao centro de controle de decisões do robô, qual sensor foi ativado. Desse modo o centro de decisões poderá reagir ao sinal táctil.

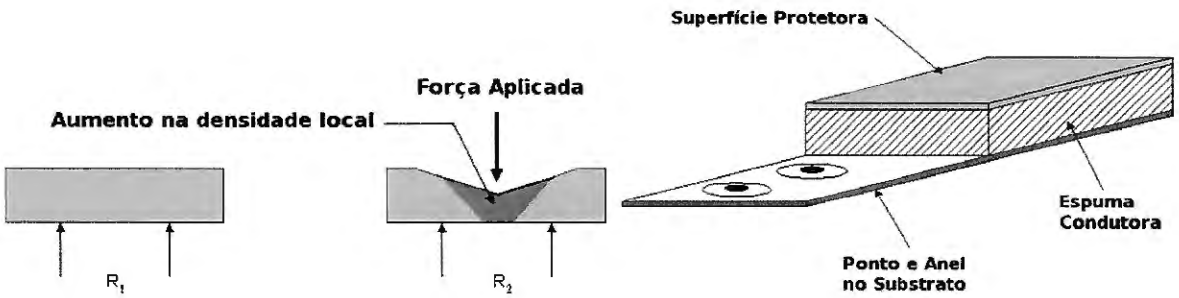

Figura 2. 23– Princípios básicos de um sensor de pressão do tipo "malha resistiva".

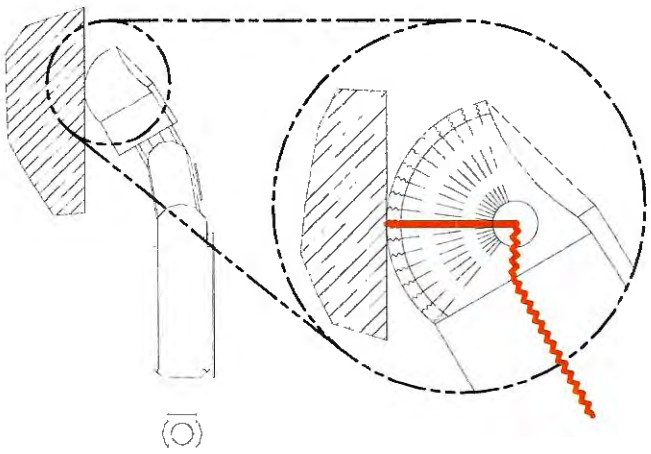

Figura 2. 24 – Esquema de aplicação de um sensor de pressão do tipo "malha resistiva" em um dedo robótico para mapeamento do ponto de contato.

2.4- Modelos Físicos de contatos

Define-se empunhadura como sendo o modo mais correto de se segurar um objeto com as mãos. Empunhar um objeto consiste no ato de se segurar um objeto com a mão. Diz-se que o objeto está sendo empunhado.

Enquanto a maioria dos autores concordam com apenas dois tipos de empunhaduras, citando como exemplos a empunhadura de precisão e a empunhadura de potência, na verdade pode-se ter ainda subdivisões dentro destas classificações tal como a empunhadura do tipo pinça.

Para maiores informações sobre tais classificações, é sugerida a consulta de [18], onde é apresentado um estudo sobre os mecanismos de empunhadura e suas classificações.

No entanto é citado abaixo, para um maior comodidade e rápida classificação, três tipos de empunhaduras:

- Empunhadura de precisão (figura 2.25); Onde a força aplicada no objeto empunhado é, na maioria das vezes pequena, porém o controle sobre o movimento do objeto é bem preciso, apesar de não ter amplitude (sem grandes deslocamentos), este tipo de empunhadura permite o uso de um algoritmo de reconhecimento de objetos através do contato da mão com o objeto quando este a examina somente com a ponta dos dedos,

- Empunhadura tipo "pinça" (figura 2.26); Garante certa precisão, maior movimento sobre o objeto em relação a empunhadura tipo precisão, porém a precisão atingida através dessa empunhadura não permite o uso de um algoritmo de reconhecimento de objetos através do contato da mão com o objeto quando este a examina somente com a ponta dos dedos,

- Empunhadura tipo "potência" (figura 2.27); Corresponde a última fase de uma empunhadura onde a instrução final do robô seria o uso de uma determinada ferramenta, na qual o uso de uma força relativamente grande fosse requerido. Somente após o reconhecimento do objeto (assim com identificação de sua posição e orientação) esse tipo de empunhadura poderia ser usada conforme surgisse a necessidade de maiores forças de atuação em um determinado objeto, após este ter sido devidamente reconhecido.

Figura 2. 25 – Precisão **Figura 2. 26** – Pinça **Figura 2. 27** – Potência

3 - Ferramentas computacionais

3.1 - Sistemas Operacionais

Sistema operacional (como é conhecido no Brasil) é um programa (*software*) ou um conjunto de programas cuja função é servir de interface entre um computador e o usuário. É comum utilizar-se a abreviatura SO (em português) ou OS (do inglês *Operating System*). Em outras palavras, um sistema operacional é formado por um conjunto de programas e rotinas computacionais que têm como objetivo criar uma camada de abstração entre o usuário e o hardware propriamente dito. Entende-se por usuário todo e qualquer objeto que precise de acesso aos recursos de um computador (seja ele um usuário "real" ou aplicativo). No presente trabalho utiliza-se recursos de dois sistemas operacionais distintos: o Microsoft Windows e o GNU/Linux, aproveitando, assim, os programas desenvolvidos para ambos os sistemas.

3.1.1 - A plataforma Microsoft Windows

O *Microsoft Windows* é o sistema operacional vendido sob licença concedida pela Microsoft ® e presente na grande maioria dos computadores pessoais (*PCs*). Daí o fato de não ser uma surpresa a existência de muitas ferramentas computacionais que foram desenvolvidas para essa plataforma, tendo em vista que o público alvo dos desenvolvedores de aplicativos, sejam fãs de jogos, usuários de suítes para escritórios, de IDEs para desenvolvimento, de programas de CAD,..., etc. Como a versão *Microsoft Windows XP*, atualmente é a mais difundida optou-se por desenvolver esse trabalho nessa plataforma.

3.1.1.1 - O compilador Dev-Cpp

O compilador Dev-Cpp é um compilador gratuito distribuído sob a licença GNU/GPL desenvolvido para a plataforma Windows que pode ser adquirido na página http://www.bloodshed.net/dev/devcpp.html da internet. Possui uma interface gráfica bastante agradável e é de fácil atualização via web, dando, desse modo, a oportunidade ao usuário, de configurar sua base de desenvolvimento conforme uma ampla gama de possibilidades disponíveis na internet.

3.1.2 - A plataforma GNU/Linux

As distribuições de sistemas operacionais baseadas na licença GNU/Linux [9], que no início do seu desenvolvimento (início dos anos 90) eram sistemas operacionais com poucos recursos gráficos e pouca compatibilidade de *hardware*. Atualmente vem ganhando terreno frente ao Windows da Microsoft, pois nos últimos anos, desde o lançamento do Kernel 2.6 em 2005, os problemas de compatibilidades de *hardware* vêm sendo resolvidos juntamente com os fabricantes de equipamentos que passaram a enxergar nesses sistemas terreno fértil para novas vendas, pois reduz o custo de aquisição de um computador pessoal e investimentos. As interfaces gráficas (*Fluxbox, Window Maker,...*) e *Desktops* (KDE e Gnome) foram aperfeiçoadas tornando-se mais atraentes para o público com pouco conhecimento de informática. e que juntamente com os surgimento de aplicativos voltados para diversos públicos alvos e maior gama de opções desses, contribuirão, ainda mais, para a disseminação desse sistema operacional. Tendo em vista essas vantagens e atrativos, a implementação deste trabalho também foi realizada na distribuição Debian Etch 4.0, que é um sistema GNU/Linux.

Figura 3. 1 – Logotipo do GNU Project

3.1.2.1 - O compiladores gcc e g++

O GNU *Compiler Collection* (chamado usualmente por GCC) é um conjunto de compiladores de linguagens de programação produzido pelo projeto GNU. É *software* livre distribuído pela *Free Software Foundation* (FSF) sob os termos da GNU GPL, e é um componente-chave do conjunto de ferramentas GNU. É o compilador padrão para sistemas operacionais UNIX e Linux e certos sistemas operacionais derivados tais como o Mac OS X.

Originalmente designado por GNU C Compiler (compilador C GNU), por suportar somente a linguagem de programação C, foi mais tarde estendido para suportar a compilação de C++, Fortran, Ada, Java e Objective-C, entre outros.

3.2 - A API gráfica OpenGL

3.2.1 - O que é OpenGL?

O termo OpenGL [19] refere-se à uma API (*Application Programming Interface*) destinada a criar uma interface para o *hardware* gráfico do computador. Ela é considerada o melhor ambiente para desenvolvimento portável de aplicativos gráficos 2D e 3D e, desde a sua introdução em 1992, a OpenGL tornou-se a mais amplamente usada e suportada interface programável de aplicações 2D e 3D em programas destinados aos mais variados ambientes como por exemplo, o científico e o de jogos. Devido à sua padronização proporcionada pelo seu grande sucesso, essa API forneceu as condições tecnológicas para que as empresas pudessem desenvolver milhares de aplicativos voltados para uma ampla variedade de plataformas (Sistemas Operacionais).

A OpenGL auxilia a inovação e acelera o desenvolvimento de aplicações incorporando um amplo conjunto de funções de *rendering*, mapeamento de texturas, efeitos especiais, e outras poderosas funções de visualizações. Desse modo, os desenvolvedores podem usufruir do poder da OpenGL através de todos sistemas operacionais, garantindo amplo suporte nas aplicações desenvolvidas através dessa API.

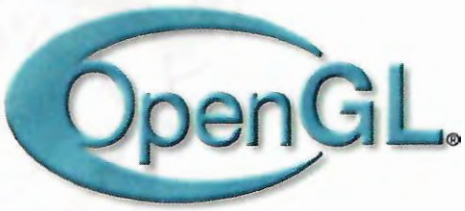

Figura 3. 2 – Logotipo da OpenGL

3.2.2 - O que é GLUT?

GLUT é a *OpenGL Utility Toolkit*, kit de ferramentas (*toolkit*) para criação e gerenciamento de janelas, formas primitivas, funções de teclado e mouse, independente de plataforma (sistema operacional) para se desenvolver programas OpenGL. Ela implementa uma interface de programação para aplicativos de janelas para OpenGL. A GLUT faz da tarefa de criação de janelas, algo de aprendizagem consideravelmente fácil e totalmente portável para qualquer plataforma, sendo então, o ponto de partida para quem deseja começar a utilizar a OpenGL em pequenos e médios programas. Considerando-se que a GLUT não é uma *toolkit* completa com todos os elementos necessários para a construção de aplicativos complexos e sofisticados, é aconselhável desenvolver tais aplicativos utilizando-se as *toolkits* nativas dos próprios sistemas para os quais se está implementando. A GLUT é simples, fácil e pequena.

A GLUT tem bibliotecas nas linguagens de programação C/C++, FORTRAN e Ada, estendendo ainda mais o seu acesso a vários perfis de desenvolvedores. A versão atual da GLUT é a 3.7, datada de agosto de 1998.

A GLUT não é *Open Source* e seu *copyright* é mantido por Mark Kilgard. Assim, sua licença tal como está hoje, não permite que nenhum outro desenvolvedor modifique essa biblioteca atualizando-a. Vale lembrar que a última versão da GLUT é de agosto de 1998 e, desse modo, outras versões baseadas na GLUT mais atuais, tais como a FreeGLUT (X-*Consortium license*) e a SDL [26] (GNU LGPL *license*) foram lançadas de modo a auxiliar os desenvolvedores a criarem janelas facilmente para o desenvolvimento de aplicativos de pequeno e médio porte.

Figura 3. 3 – Logotipo da SDL

3.2.3 - FreeGLUT

FreeGLUT é a alternativa *Open Source* da GLUT, assim sendo sua atualização pode ocorrer com mais freqüência e por um número maior de pessoas. A FreeGLUT pode ser conseguida através da página web http://freeglut.sourceforge.net/, onde se pode adquirir maiores informações sobre o andamento do projeto, solicitação de recursos adicionais para a *toolkit*, contato via e-mail com os desenvolvedores (somente para cadastrados), ..., etc. Nesse trabalho utilizou-se a FreeGLUT para a implementação OpenGL.

3.3 - A linguagem C / C++ e a OpenGL

A OpenGL, como uma API, segue a convenção de chamada da linguagem C. Isso significa que os programas escritos em C podem facilmente chamar as funções da OpenGL, isso porque as funções OpenGL foram escritas em C, como também pela razão que se segue: é fornecido um conjunto de funções C intermediárias que chamam funções escritas em *Assembly* ou outra linguagem de programação. Foi em decorrência desses fatores, que a linguagem C / C++ foi a escolhida para se desenvolver o presente trabalho.

3.4 - Programas de CAD

Uma das mais poderosas ferramentas do engenheiro contemporâneo, são os programas de CAD (***C**omputer **A**ided **D**esign* – desenho auxiliado por computador). Atualmente devido à concorrência entre as empresas, a velocidade de desenvolvimento de um produto tem crescido de forma acentuada, fator que torna praticamente impossível uma empresa manter-se competitiva sem contar com esses programas. Por isso, além de auxiliar na modelagem em si do produto, os *softwares* mais evoluídos (como o INVENTOR) ajudam o gerenciamento do fluxo de informações no interior dos projetos por eles desenvolvidos.

Na modelagem virtual do presente trabalho, recorreu-se à três programas de CAD que serão discutidos mais detalhadamente a seguir.

3.4.1 - O AutoCAD

O AutoCAD, da fabricante Autodesk, é no momento, dentre os programas de CAD, com certeza, o mais vendido no Brasil. Com uma interface muito prática e didática este programa conquistou adeptos pelo mundo todo e principalmente no mercado nacional. Se não é um dos melhores programas de CAD, com certeza deve ser o mais conhecido pelos brasileiros. O grande ponto a favor deste *software* é sua versatilidade em desenhos em duas dimensões, embora possua na versão 2005 (usada neste trabalho) ferramentas limitadas para pequenos projetos em três dimensões (3D). Devido a esses fatores e, ao grande número de recursos que o aplicativo disponibiliza, seria impensável não utilizá-lo neste trabalho.

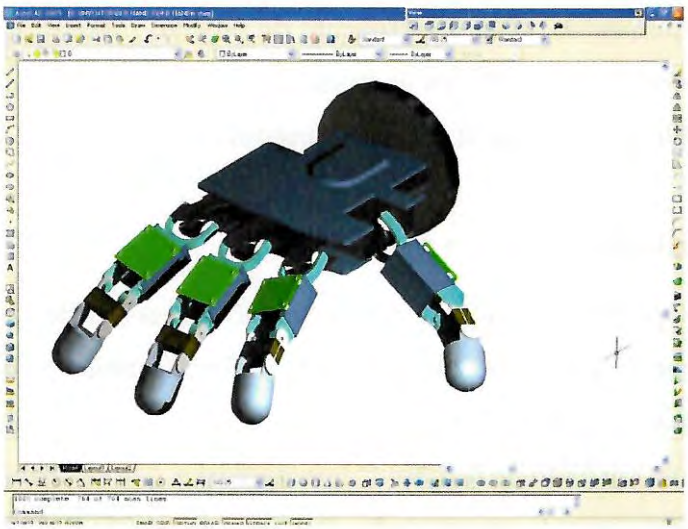

Figura 3. 4 – Interface do AutoCAD

3.4.2 O Autodesk INVENTOR (R10)

Se o AutoCAD é o produto da Autodesk destinado a projetos em 2D, com o INVENTOR, a fabricante Autodesk dedicou o desenvolvimento deste *software,* totalmente voltado para projetos em 3D, porém, com ferramentas poderosas para facilitar a emissão de desenhos detalhados em 2D, visando uma produção rápida de produtos/projetos. Diferente do AutoCAD, o INVENTOR (na versão 10) trabalha mais de uma extensão de arquivos, sendo destinadas para peças, montagens, bibliotecas de peças padronizadas, apresentações e folhas de desenhos. Essas extensões são interligadas pelo *software* de forma transparente para o usuário com o intuito de manter o projeto e os componentes dentro deste sempre atualizados. As ferramentas de modelagem em 3D do INVENTOR são mais desenvolvidas que no AutoCAD, dando ao seu usuário maior versatilidade e rapidez no projeto, além de proporcionar maior ousadia ao projetista no desenvolvimento de seus desenhos.

Neste trabalho foi utilizado o INVENTOR Release 10 como ferramenta de modelagem 3D, de detalhamento 2D e de exportação para a extensão *.STL.

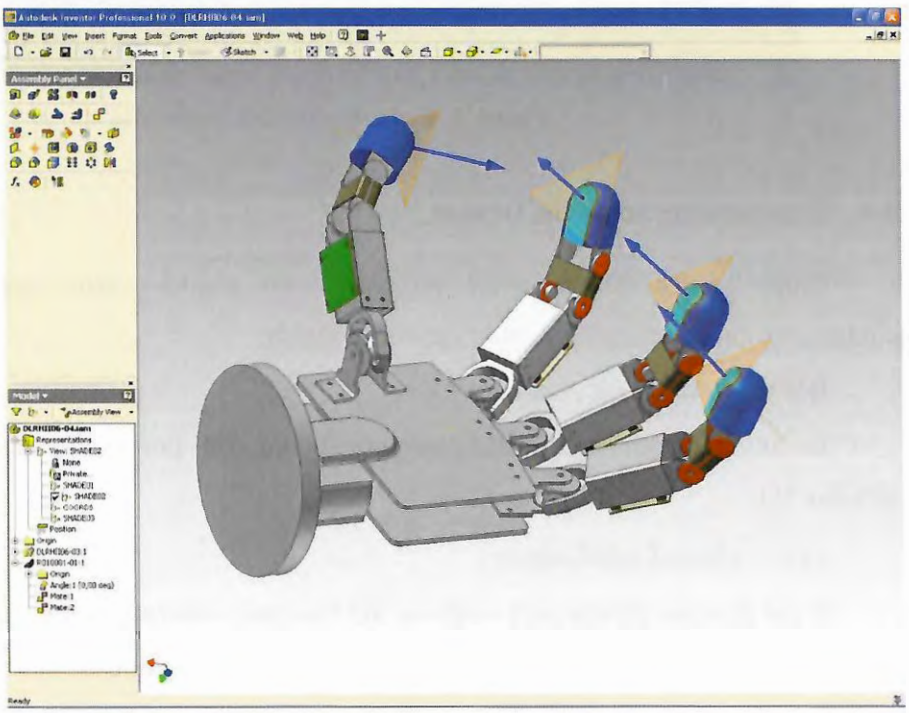

Figura 3. 5 – Interface do INVENTOR R10

3.4.3 O 3D Studio Max 8.0

Para importação do formato .STL, adequação dos modelos aos quadrantes corretos e, por fim, para a exportação em formato .3DS (formato usado na programação OpenGL) foi utilizado o programa 3D Studio Max 8.0 ,que é um programa de modelagem em 3D, porém voltado para um público alvo mais artístico, que visa o modelamento de jogos, cenários,

maquetes com texturas mais realistas e realização de animações. O programa 3D Studio Max 8.0 foi desenvolvido pela Autodesk.

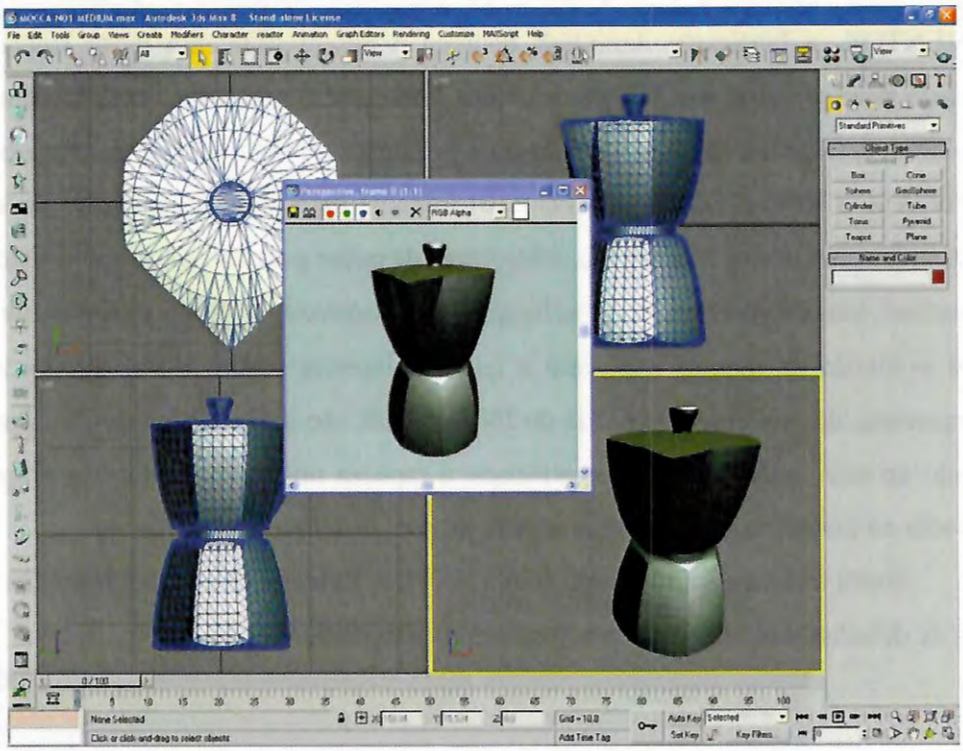

Figura 3. 6 – Interface do 3D Studio Max

3.4.4 Extensões de arquivos Usados

Como já foi citado, neste projeto foram usados vários programas de CAD e visualização, cujas extensões serão descritas a seguir:

DWG – (Autodesk Autocad)

Foi utilizada principalmente para projetos em 2D, porém, igualmente, para armazenar entidades 3D;

STL – (**S**tero**L**itho**G**raphy)

É um formato criado pela empresa 3D Systems, utilizado neste trabalho como "matéria semi-acabada", uma vez que foi exportado do software Autodesk INVENTOR e importado pelo software 3D Studio Max para trabalho de arte-final.

X-File – Este formato de arquivo, criado pela Microsoft, neste trabalho, foi utilizado como unidade básica. É a base de armazenamento de dados do objeto simplificado que será estocado na memória para, posteriormente, ser comparado com o modelo refinado. Nestes arquivos estão armazenados os seguintes dados: Vértices, Normais dos Vértices, índices dos vértices que correspondem as faces, materiais e coordenadas de texturas. Ele pode ser gerado no 3D Studio Max através da instalação de um plugin gratuito desenvolvido pela Pandasoft (www.andyather.co.uk).

3.4.5 Metodologia de trabalho com softwares de desenho

Seguem abaixo para melhor compreensão dois fluxogramas (figura 3.7 e 3.8) mostrando como os softwares de CAD foram usados no presente trabalho.

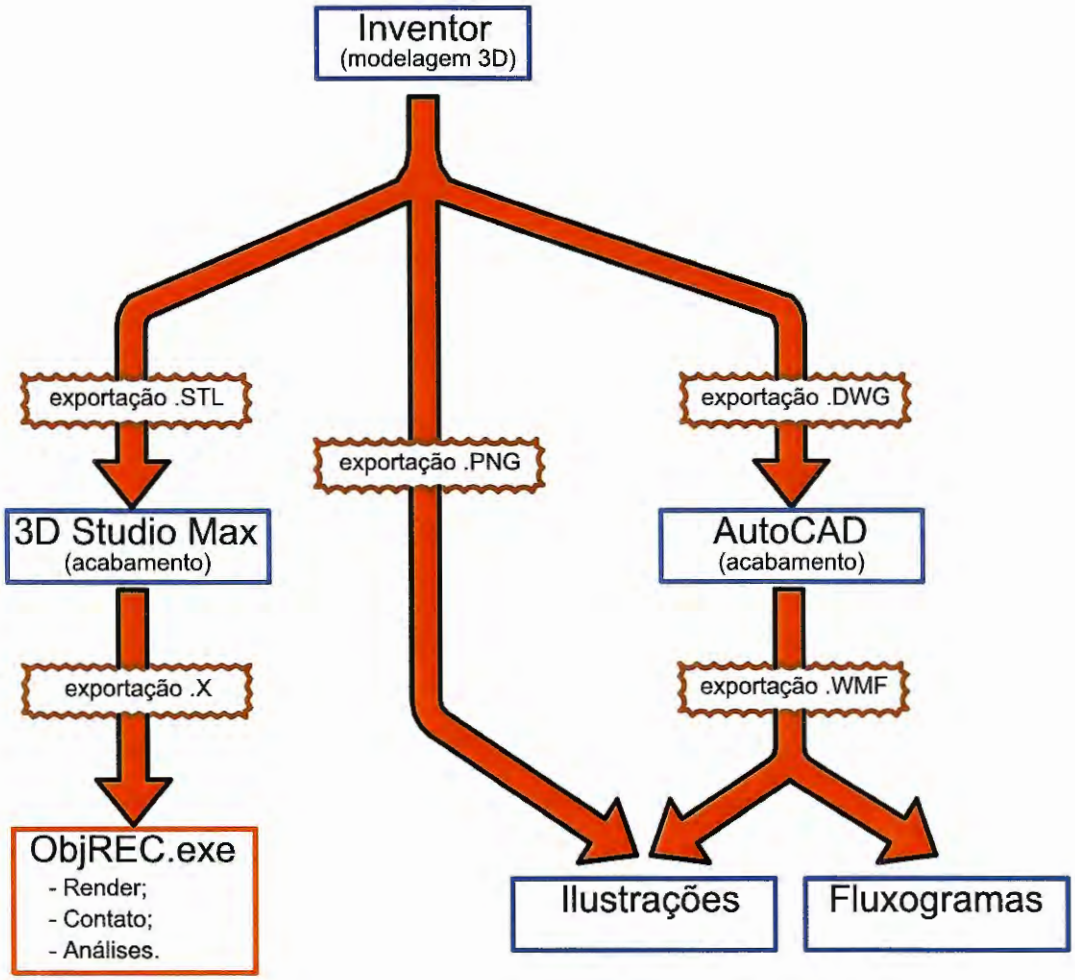

Figura 3. 7 – Fluxograma dos objetos modelados em 3D.

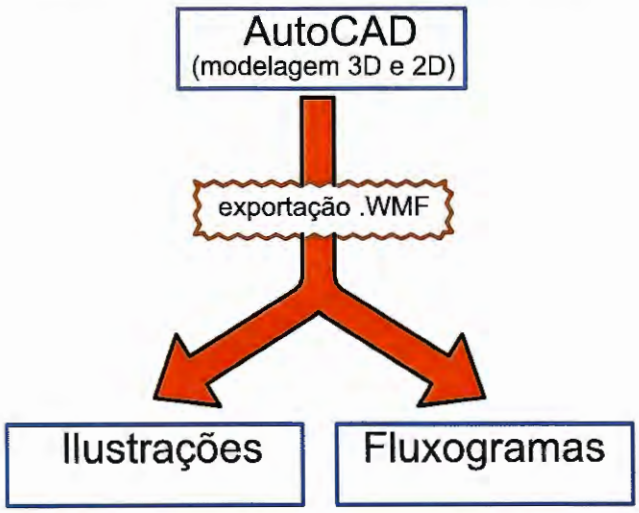

Figura 3. 8 – Fluxograma dos objetos modelados em 2D.

4 - Fundamentação Teórica e Metodológica

4.1 Ambiente Virtual e Linguagem de programação

Para o teste dos algoritmos empregados foi elaborado um ambiente virtual em OpenGL (ver [5]) programado em C / C++ ([25], [2] e [6]), com o intuito de simular um robô empunhando um objeto real[1]. Sobre esse ambiente virtual falaremos a seguir.

4.1.1 Sistema de coordenadas OpenGL

O ambiente tridimensional (3D) foi modelado através de um sistema de coordenadas cartesianas (x, y, z) no qual foram posicionados a mão robótica e os objetos a serem empunhados.

Existem dois tipos de sistemas de coordenadas cartesianas (ou referenciais), normalmente adotados para o uso em aplicações gráficas tridimensionais: os que são baseados na regra da mão esquerda, conhecidos como *left-handed cartesian coordinates system*, e os que são baseados na regra da mão direita, conhecidos como *right-handed cartesian coordinates system*. O que diferencia um do outro é o sentido adotado para a orientação do eixo *z*, o qual é obtido com base nas regras abaixo mencionadas.

Com a mão posicionada na origem do sistema, mantendo-se a direção do polegar perpendicular ao plano que contém os eixos *x* e *y*, aplica-se uma rotação nos demais dedos, partindo-se dos valores positivos do eixo *x* para os valores positivos do eixo *y*. O sentido do eixo *z* é então definido pelo apontamento do dedo polegar. A figura 4.1 ilustra os dois tipos de sistemas de coordenadas.

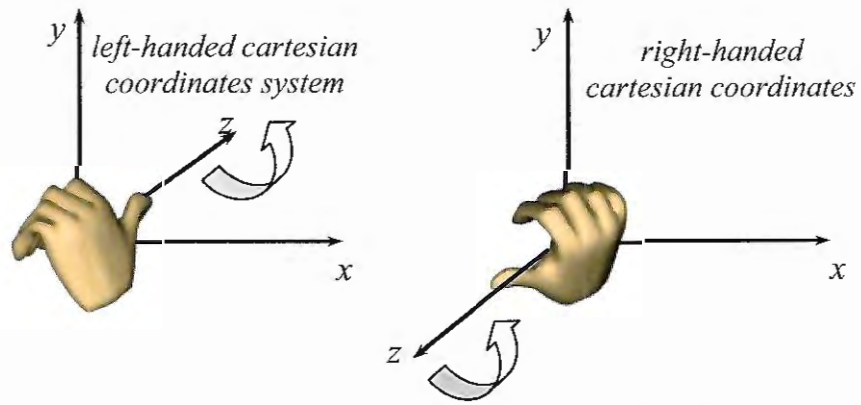

Figura 4. 1 - Sistemas de coordenadas baseados nas regras da mão direita e esquerda.

[1] Parte das imagens e definições usadas neste capítulo foram publicadas com autorização de RIBEIRO, Adriano J. Marques, maiores detalhes ver [23].

A rigor, um sistema de coordenadas tridimensional é um conjunto composto por três vetores linearmente independentes, capazes de gerar, por combinação linear, quaisquer outros vetores deste espaço. Caso os vetores geradores sejam unitários e ortogonais entre si, diz-se que eles constituem uma base ortonormal.

Uma vez adotada uma convenção de eixos para a definição de um sistema de coordenadas, qualquer produto vetorial efetuado neste espaço seguirá automaticamente essa mesma convenção.

O sistema de coordenadas definido para o cenário tridimensional, onde os objetos são gerados e manipulados, é chamado de sistema de coordenadas do mundo ou sistema de coordenadas universal. Para fins didáticos, será adotada a letra U para representar esse referencial, de forma que qualquer vetor expresso nele terá a seguinte notação:

$$^{U}\vec{V} = \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix}^{U} \tag{1}$$

ou

$$^{U}\vec{V} = {}^{U}(v_x, v_y, v_z) \tag{2}$$

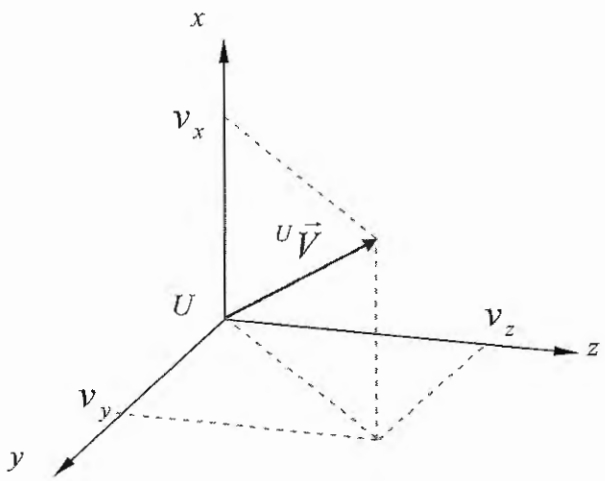

Figura 4. 2 - Definição de um vetor no sistema de coordenadas universal.

O *OpenGL* adota o sistema de coordenadas baseado na regra da mão direita. Assim, se for desejado que uma aplicação 3D, desenvolvida em um sistema baseado na regra da mão direita (como é o caso do *3d studio max*), seja executada com o uso do *OpenGL*, não haverá nenhuma dificuldade de implementação dada a compatibilidade dos sistemas.

Os sistemas de coordenadas apresentados neste trabalho são baseados na regra da mão direita, uma vez que o ambiente virtual do projeto foi desenvolvido com o uso do *OpenGL*.

4.1.2 Vértices e Faces

Qualquer objeto tridimensional gerado por computador é formado, em sua essência, por um conjunto de vértices. Cada vértice é representado por um vetor posição de coordenadas reais, geralmente expresso no sistema de coordenadas universal. Embora um Ponto ou Vetor no espaço seja descrito em termos das coordenadas (x , y , z), para uma implementação em C / C++ e aplicação destas coordenadas nas Matrizes de Transformações Homogêneas (que serão apresentadas mais adiante), o conceito de vetor foi estendido para um vetor de tamanho 4, (x , y , z , s), onde sempre será atribuído a 's' o valor 1, necessário para o cálculo matricial.

$$\text{VETOR4} = [x \quad y \quad z \quad s]^T \tag{3}$$

Para isso foi definido um novo tipo de variável, a partir de uma estrutura de dados (ver [14], capitulo 7) conforme abaixo:

```
//-------------------------------------------------------------------------------
//Cria estrutura de um Vetor de tamanho 4 ------------------------------------
struct VETOR4
{
    float x;
    float y;
    float z;
    float s;
};
//-------------------------------------------------------------------------------
```

Apesar do nome VETOR4, inferir sobre seu uso em geometria espacial, esse novo tipo de variável também pode ser utilizada e entendida como uma matriz *float* de tamanho 4, que poderá, igualmente ser usada como armazenadora de pontos e outros dados.

Vale lembrar que no presente estudo ela é definida como a variável matrix4x4, que é uma matriz 4x4 constituída de uma estrutura de dados de quatro variáveis do tipo VETOR4. Assim, pode-se montar uma matriz 4x4 com 4 linhas de 4 vetores como abaixo:

$$\text{matrix4x4} = \begin{bmatrix} \text{VETOR4 Lin1} \\ \text{VETOR4 Lin2} \\ \text{VETOR4 Lin3} \\ \text{VETOR4 Lin4} \end{bmatrix} = \begin{bmatrix} \text{Lin1.x} & \text{Lin1.y} & \text{Lin1.z} & \text{Lin1.s} \\ \text{Lin2.y} & \text{Lin2.y} & \text{Lin2.z} & \text{Lin2.s} \\ \text{Lin3.x} & \text{Lin3.y} & \text{Lin3.z} & \text{Lin3.s} \\ \text{Lin4.x} & \text{Lin4.y} & \text{Lin4.z} & \text{Lin4.s} \end{bmatrix} \tag{4}$$

A implementação desta estrutura de dados em C / C++ foi feita no trecho de código a seguir.

```
//-------------------------------------------------------------------------------
//Cria estrutura de uma Matriz [4] x [4] ----------------------------------------
struct matrix4x4
{
    VETOR4 Lin1;
    VETOR4 Lin2;
    VETOR4 Lin3;
    VETOR4 Lin4;
};
//-------------------------------------------------------------------------------
```

Retornando ao conceito de face tem-se, por exemplo, na figura 4.3 os vértices P_1, P_2 e P_3, representados respectivamente pelos vetores posição ${}^{U}\vec{P}_1$, ${}^{U}\vec{P}_2$ e ${}^{U}\vec{P}_3$.

As faces de uma malha são os polígonos que a compõe, normalmente triângulos. Cada face apresenta um vetor normal, gerado pelo produto vetorial de dois vetores definidos pelos vértices da face. Desta forma, o sentido do vetor normal é dependente da escolha dos vetores usados no produto vetorial. Os vetores normais são calculados para nos fornecer parâmetros de entrada no algoritmo de hipótese de contatos e também possibilitar uma visualização mais adequada do objeto simulado na tela quando na aplicação do *shading* na etapa de *rendering*. A figura 4.3 ilustra uma face e seu vetor normal.

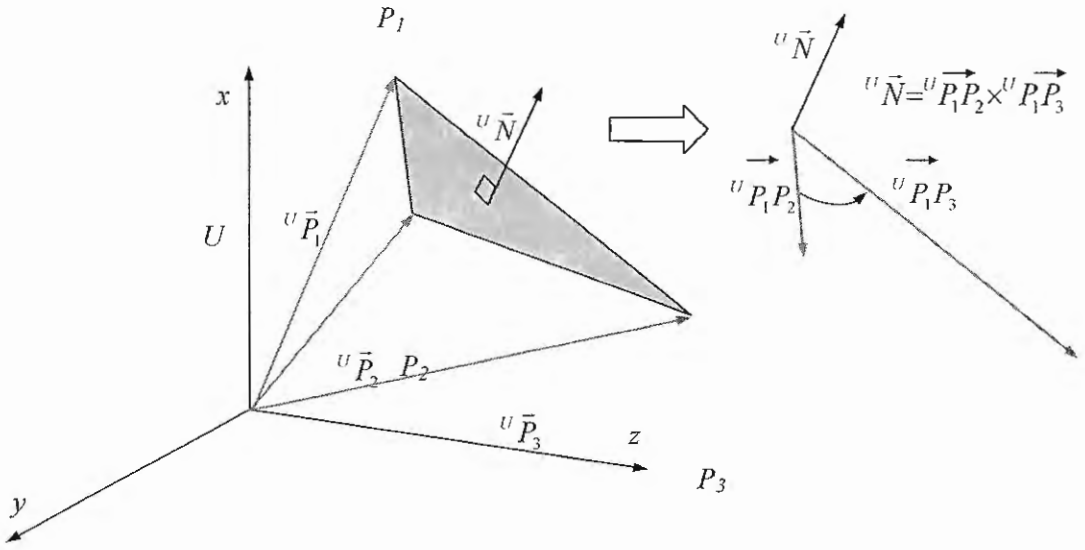

Figura 4. 3 - Vértices de uma face expressos no sistema de coordenadas universal.

É importante ressaltar que para definir o sentido do vetor normal, o produto vetorial segue a mesma convenção do sistema de coordenadas universal. Desse modo, o produto vetorial para o cálculo dos vetores normais deve levar em conta que o vetor resultante deverá sair para fora do objeto para que este seja visualizado no *rendering*.

Nesse sentido, deve-se ficar atento para esse cálculo dos vetores normais, quanto à "regra da mão direita".

4.1.3 Transformações geométricas

As aplicações gráficas tridimensionais normalmente apresentam recursos para a movimentação e manipulação dos modelos 3D dentro do cenário virtual. Estas ações são realizadas por meio da aplicação de transformações gráficas sobre os objetos tridimensionais. De um modo geral, qualquer transformação, por mais complexa que seja, pode ser resumida em combinações de translações, rotações e variações de escala.

Uma transformação pode ser aplicada à geometria do modelo 3D ou ao sistema de coordenadas do cenário virtual. Na transformação geométrica o sistema de coordenadas permanece fixo enquanto as transformações são aplicadas aos vértices do modelo 3D (figura 4.4). Na transformação de coordenadas ocorre o processo inverso: o modelo 3D permanece fixo e as transformações são aplicadas ao sistema de coordenadas. Estas transformações não são independentes, já que uma é o inverso da outra. Basta, portanto, que uma delas seja estudada para que a outra esteja automaticamente definida.

Como mencionado anteriormente, qualquer objeto tridimensional gerado por computador é formado por vértices (pontos). Assim, as transformações gráficas operam, em última análise, sobre os vértices dos objetos. Logo, quando se diz que um determinado objeto sofreu uma transformação, na realidade esta transformação foi aplicada a todos os seus vértices.

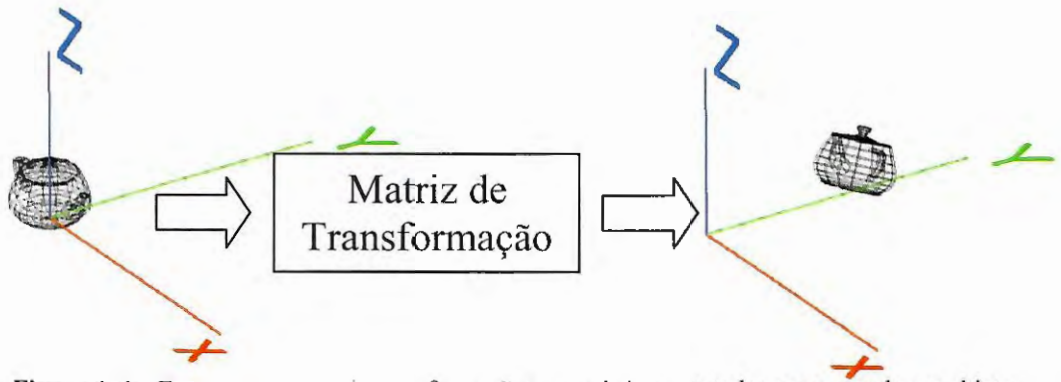

Figura 4. 4 – Esquema mostrando transformações geométricas em todos os pontos de um objeto.

Embora a OpenGL tenha funções bem otimizadas para transformações de translação (glTranslatef), rotação de torno de um eixo (glRotatef) e de escala (glScalef), deve-se ressaltar que tais funções são do tipo ***void*** e, portanto, não retornam valores. Devido a isso foram implementadas funções personalizadas para executar as transformações (que são basicamente

operações matriciais) retornando valores para serem aplicados e analisados conforme a necessidade do programador.

4.1.4 Transformações geométricas de translação (TRANS)

Transladar um objeto, significa deslocar o mesmo de forma que a sua orientação original seja mantida, ou seja, durante a transformação todos os seus vértices percorrem trajetórias paralelas e de mesmo comprimento.

Um vértice P, de coordenadas ${}^{U}(p_x, p_y, p_z)$, pertencente a um determinado objeto que é deslocado na direção definida pelo vetor ${}^{U}\vec{D}={}^{U}(d_x, d_y, d_z)$, terá sua nova posição P' a partir da soma vetorial de ${}^{U}\vec{P}$ com ${}^{U}\vec{D}$, como mostrado a seguir:

$$ {}^{U}\vec{P}' = {}^{U}\begin{pmatrix} p_x' \\ p_y' \\ p_z' \end{pmatrix} = {}^{U}\vec{P} + {}^{U}\vec{D} = {}^{U}\begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix} + {}^{U}\begin{pmatrix} d_x \\ d_y \\ d_z \end{pmatrix} = {}^{U}\begin{pmatrix} p_x + d_x \\ p_y + d_y \\ p_z + d_z \end{pmatrix} \qquad (5) $$

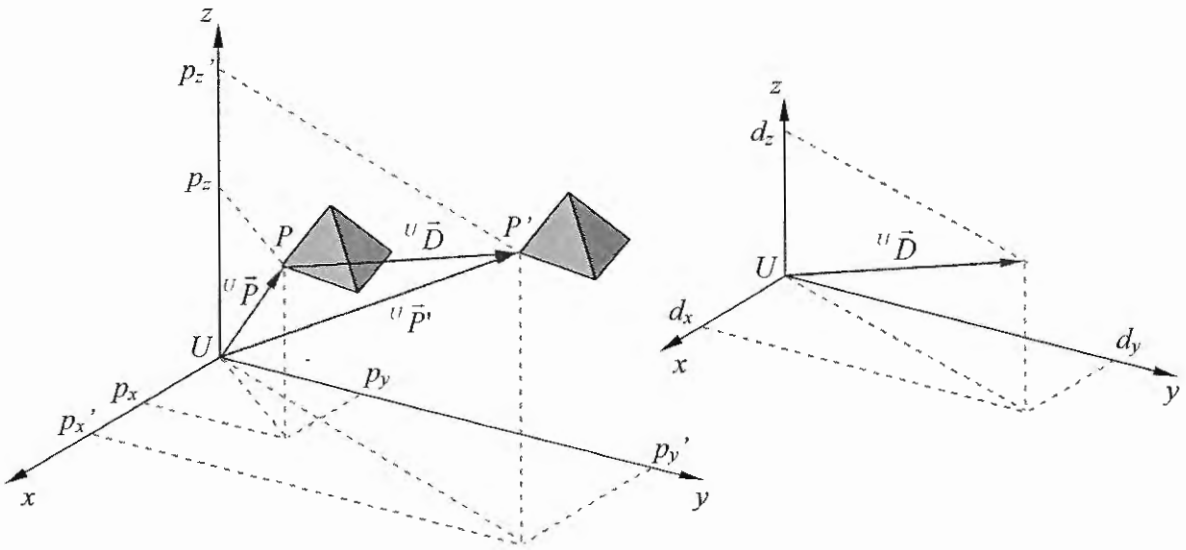

Figura 4. 5 - Translação de um modelo tridimensional.

4.1.5 Transformações geométricas de variação da escala (SCALE)

A variação da escala é aplicada com o intuito de alterar as dimensões dos objetos. Uma vez que essa transformação é feita com relação à origem do sistema de coordenadas, as distâncias dos vértices dos objetos até ela também são alteradas. Após uma transformação de

variação de escala, um vértice P de um objeto assume uma nova posição P', segundo a operação:

$$ {}^{U}\vec{P}' = \begin{pmatrix} p_x' \\ p_y' \\ p_z' \end{pmatrix} = \begin{pmatrix} s_x p_x \\ s_y p_y \\ s_z p_z \end{pmatrix} \qquad (6) $$

onde s_x, s_y e s_z são os fatores (ou coeficientes) de escala nas direções dos eixos x, y e z respectivamente. Se esses fatores forem maiores do que a unidade, o objeto é aumentado; se forem menores, o objeto é reduzido. Coeficientes iguais proporcionam uma variação uniforme no tamanho do objeto, enquanto que coeficientes distintos proporcionam deformações no objeto. Na figura 4.6, o cubo C foi submetido a uma transformação de variação uniforme de escala, produzindo o cubo C', de tamanho maior. Já a aplicação, em C, de uma transformação de variação não uniforme de escala produziu o paralelepípedo C''. Note que as transformações alteraram as posições dos vértices do objeto. De fato, a origem do sistema de coordenadas é o único ponto cuja posição permanece inalterada numa transformação de variação de escala.

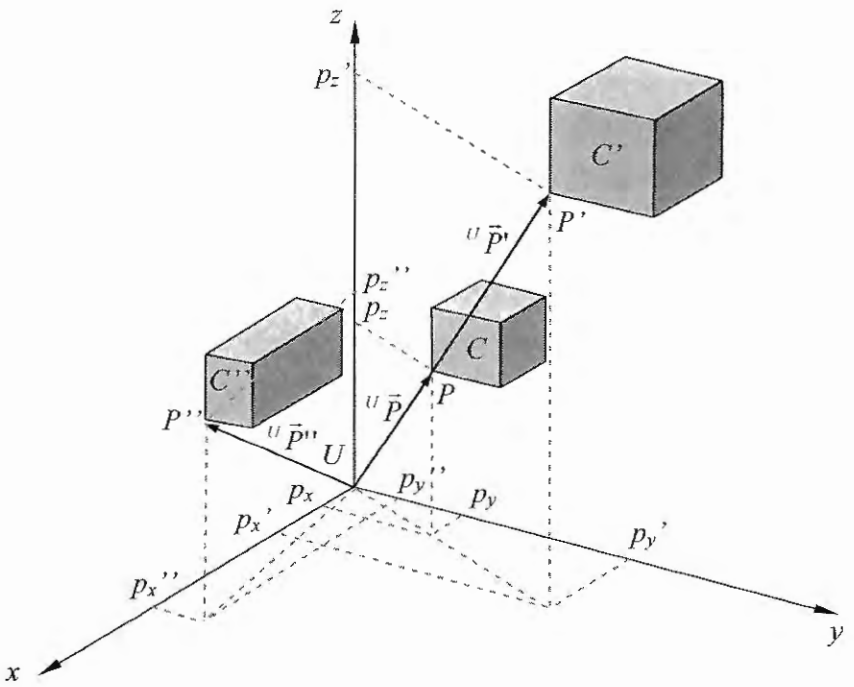

Figura 4. 6 - Transformações de variação de escala.

4.1.6 Transformações geométricas rotação (ROT)

Na rotação, o objeto gira em torno de um determinado eixo e a distância dos vértices em relação a esse eixo permanece inalterada durante a transformação. Basicamente, qualquer eixo pode ser definido por um vetor direcional. Assim, existem infinitos eixos em torno dos quais pode-se aplicar uma rotação. É interessante, no entanto, enfatizar as transformações de rotação em torno dos eixos cartesianos (ou canônicos), já que qualquer rotação pode ser obtida pela composição de três outras efetuadas consecutivamente sobre esses eixos.

Obviamente, o ângulo de rotação deve ser medido em um plano perpendicular ao eixo de rotação. Nos sistemas de coordenadas baseados na regra da mão direita o sentido positivo de rotação é definido pela aplicação dessa mesma regra sobre o eixo especificado.

Na figura 4.7, um objeto é rotacionado a partir de um ângulo θ ao redor do eixo y, de forma que um vértice P passe a ocupar uma nova posição P'. A projeção ortogonal do vetor ${}^{U}\vec{P}$ sobre o plano xz tem módulo r e forma um ângulo φ com o eixo x.

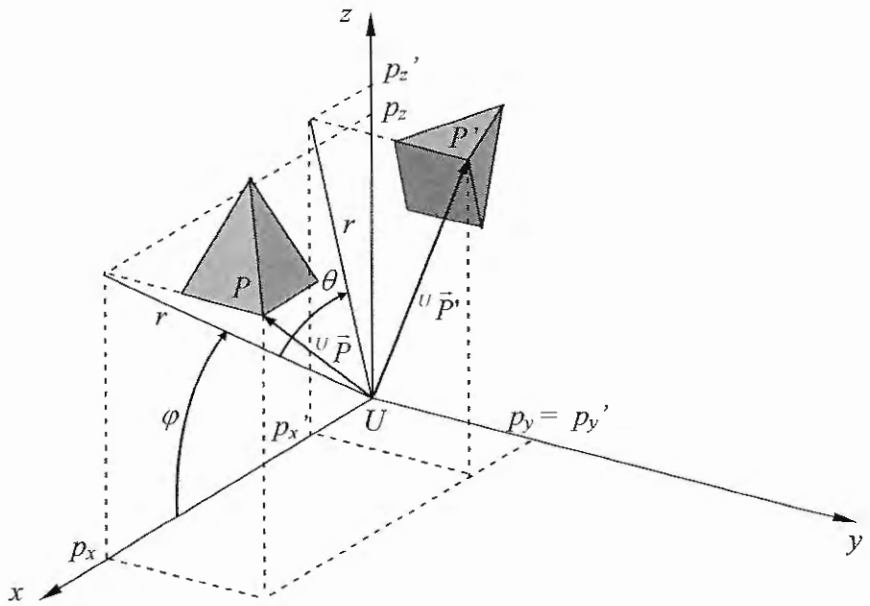

Figura 4. 7 - Rotação de um objeto em torno do eixo x.

Com base na figura 4.7, pode-se deduzir que:

$$p_x' = r\cos(\theta + \varphi) \tag{7}$$

$$p_y' = p_y \tag{8}$$

$$p_z' = rsen(\theta + \varphi) \tag{9}$$

Aplicando-se algumas propriedades trigonométricas obtém-se:

$$p_x' = r\cos\theta\cos\varphi - rsen\theta sen\varphi \quad (10)$$

$$p_y' = p_y \quad (11)$$

$$p_z' = rsen\theta\cos\varphi + r\cos\theta sen\varphi \quad (12)$$

Da figura, também se deduz que:

$$p_x = r\cos\varphi \quad (13)$$

$$p_z = rsen\varphi \quad (14)$$

Substituindo-se (13) e (14) em (7) e (8) respectivamente, obtém-se as coordenadas de P':

$$p_x' = p_x\cos\theta - p_z sen\theta \quad (15)$$

$$p_y' = p_y \quad (16)$$

$$p_z' = p_x sen\theta + p_z\cos\theta \quad (17)$$

Pode-se desenvolver o mesmo raciocínio para se obter as rotações em torno dos demais eixos cartesianos. Assim:

Para uma rotação de θ em torno de eixo x:

$$p_x' = p_x \quad (18)$$

$$p_y' = p_y\cos\theta + p_z sen\theta \quad (19)$$

$$p_z' = -p_y sen\theta + p_z\cos\theta \quad (20)$$

Para uma rotação de θ em torno de eixo z:

$$p_x' = p_x\cos\theta - p_y sen\theta \quad (21)$$

$$p_y' = p_x sen\theta + p_y\cos\theta \quad (22)$$

$$p_z' = p_z \quad (23)$$

Percebe-se que a coordenada referente ao eixo de rotação não é alterada pela transformação.

4.1.7 Transformações homogêneas

Por serem lineares, as transformações de rotação e variação de escala podem ser representadas por meio de um produto entre uma matriz 3x3 e um vetor de três componentes. No entanto, a transformação de translação resulta de uma soma de vetores, não admitindo um

produto matricial. As transformações homogêneas surgiram com o intuito de padronizar todas as transformações na forma de matrizes, uma vez que a organização matricial facilita a implementação computacional e acelera a execução dos cálculos. As transformações homogêneas são descritas em matrizes de dimensão 4x4, com a quarta linha sendo igual a $(0 \quad 0 \quad 0 \quad 1)$. Neste sentido, as transformações deduzidas anteriormente podem ser reescritas sob a forma de matrizes homogêneas.

A transformação de **translação** pode ser reescrita como:

$$^{U}\vec{P}' = T_{\vec{D}}\,{}^{U}\vec{P} \tag{24}$$

ou

$$^{U}\begin{pmatrix} p_x' \\ p_y' \\ p_z' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot {}^{U}\begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix} \tag{25}$$

Para a implementação de uma função de Translação em C / C++ a função criada é a seguinte (para maiores detalhes de funções em C, consultar [25], capítulo 6):

```
//-------------------------------------------------------------------------------
// FN Carrega Matriz de TRANSLAÇÃO com X,Y,Z -------------------------------
matrix4x4 Trans (float tx, float ty, float tz)
{
matrix4x4 MTrans = {    { 1 , 0 , 0 , tx },
                        { 0 , 1 , 0 , ty },
                        { 0 , 0 , 1 , tz },
                        { 0 , 0 , 0 , 1  }    };
return MTrans;
}
//-------------------------------------------------------------------------------
```

As transformações de **rotação** em torno dos eixos cartesianos podem ser reescritas como:

- **<u>rotação em torno do eixo *x*</u>**:

$$^{U}\vec{P}' = R_{\theta,x}\,{}^{U}\vec{P} \tag{26}$$

ou

$$^{U}\begin{pmatrix} p_x' \\ p_y' \\ p_z' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\operatorname{sen}\theta & 0 \\ 0 & \operatorname{sen}\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot {}^{U}\begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix} \tag{27}$$

Para a implementação de uma função de Rotação em torno de X em C / C++ a função criada é a seguinte:

```
//-------------------------------------------------------------------------------
// FN Carrega Matriz de ROTAÇÃO com X ------------------------------------------
matrix4x4 Rotax(float rx)
{
matrix4x4 MRotx = {     { 1 , 0   , 0  , 0 },
                        { 0 , cx  , -sx , 0 },
                        { 0 , sx  , cx  , 0 },
                        { 0 , 0   , 0  , 1 }      };
return MRotx;
}
//-------------------------------------------------------------------------------
```

- **rotação em torno do eixo *y***:

$$^{U}\vec{P}' = R_{\theta,y}\,{}^{U}\vec{P} \tag{28}$$

ou

$$\begin{pmatrix} p_x' \\ p_y' \\ p_z' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta & 0 & sen\theta & 0 \\ 0 & 1 & 0 & 0 \\ -sen\theta & & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix} \tag{29}$$

Para a implementação de uma função de Rotação em torno de Y em C / C++ a função criada é a seguinte:

Sendo que c(n) é cosf(n) e s(n) é senf(n), temos;

```
//-------------------------------------------------------------------------------

// FN Carrega Matriz de ROTAÇÃO com Y ------------------------------------------
matrix4x4 Rotay(float ry)
{
matrix4x4 MRoty = {     { cy  , 0 , sy  , 0  },
                        { 0   , 1 , 0   , 0  },
                        {-sy  , 0 , cy  , 0  },
                        { 0   , 0 , 0   , 1  }      };
return MRoty;
}
//-------------------------------------------------------------------------------
```

- **Rotação em torno do eixo *z***:

$$^{U}\vec{P}' = R_{\theta,z}\,{}^{U}\vec{P} \tag{30}$$

ou

$$\begin{pmatrix} p_x' \\ p_y' \\ p_z' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta & -sen\theta & 0 & 0 \\ sen\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix} \tag{31}$$

Para a implementação de uma função de Rotação em torno de Y em C / C++ a função criada é a seguinte:

```
// ------------------------------------------------------------------------
// FN Carrega Matriz de ROTAÇÃO com Z ------------------------------
matrix4x4 Rotaz(float rz)
{
matrix4x4 MRotz = {     { cz  , -sz , 0  , 0  },
                        { sz  , cz  , 0  , 0  },
                        { 0   , 0   , 1  , 0  },
                        { 0   , 0   , 0  , 1  }     };
 return MRotz;
}
// ------------------------------------------------------------------------
```

Nota-se que todas as funções acima retornam variáveis **matrix4x4** (matrizes 4x4) como respostas. Estas funções servem para atribuir valores a variáveis do tipo matrix4x4 usadas no programa.

Para aplicação destas variáveis, foram definidas uma função de multiplicação entre matrizes e outra função de multiplicação entre uma matriz e um vetor, apresentadas em seqüência abaixo.

```
// ------------------------------------------------------------------------------------
// FN Multiplicação: --- Matriz M[4][4] x Matriz N[4][4] = Matriz Mf[4][4] -----
matrix4x4 MatrixMatrix4x4(const matrix4x4 M, const matrix4x4 N)
{
matrix4x4 Mf;

Mf.Lin1.x = M.Lin1.x*N.Lin1.x + M.Lin1.y*N.Lin2.x + M.Lin1.z*N.Lin3.x + M.Lin1.s*N.Lin4.x;
Mf.Lin1.y = M.Lin1.x*N.Lin1.y + M.Lin1.y*N.Lin2.y + M.Lin1.z*N.Lin3.y + M.Lin1.s*N.Lin4.y;
Mf.Lin1.z = M.Lin1.x*N.Lin1.z + M.Lin1.y*N.Lin2.z + M.Lin1.z*N.Lin3.z + M.Lin1.s*N.Lin4.z;
Mf.Lin1.s = M.Lin1.x*N.Lin1.s + M.Lin1.y*N.Lin2.s + M.Lin1.z*N.Lin3.s + M.Lin1.s*N.Lin4.s;

Mf.Lin2.x = M.Lin2.x*N.Lin1.x + M.Lin2.y*N.Lin2.x + M.Lin2.z*N.Lin3.x + M.Lin2.s*N.Lin4.x;
Mf.Lin2.y = M.Lin2.x*N.Lin1.y + M.Lin2.y*N.Lin2.y + M.Lin2.z*N.Lin3.y + M.Lin2.s*N.Lin4.y;
Mf.Lin2.z = M.Lin2.x*N.Lin1.z + M.Lin2.y*N.Lin2.z + M.Lin2.z*N.Lin3.z + M.Lin2.s*N.Lin4.z;
Mf.Lin2.s = M.Lin2.x*N.Lin1.s + M.Lin2.y*N.Lin2.s + M.Lin2.z*N.Lin3.s + M.Lin2.s*N.Lin4.s;

Mf.Lin3.x = M.Lin3.x*N.Lin1.x + M.Lin3.y*N.Lin2.x + M.Lin3.z*N.Lin3.x + M.Lin3.s*N.Lin4.x;
Mf.Lin3.y = M.Lin3.x*N.Lin1.y + M.Lin3.y*N.Lin2.y + M.Lin3.z*N.Lin3.y + M.Lin3.s*N.Lin4.y;
Mf.Lin3.z = M.Lin3.x*N.Lin1.z + M.Lin3.y*N.Lin2.z + M.Lin3.z*N.Lin3.z + M.Lin3.s*N.Lin4.z;
Mf.Lin3.s = M.Lin3.x*N.Lin1.s + M.Lin3.y*N.Lin2.s + M.Lin3.z*N.Lin3.s + M.Lin3.s*N.Lin4.s;

Mf.Lin4.x = M.Lin4.x*N.Lin1.x + M.Lin4.y*N.Lin2.x + M.Lin4.z*N.Lin3.x + M.Lin4.s*N.Lin4.x;
Mf.Lin4.y = M.Lin4.x*N.Lin1.y + M.Lin4.y*N.Lin2.y + M.Lin4.z*N.Lin3.y + M.Lin4.s*N.Lin4.y;
Mf.Lin4.z = M.Lin4.x*N.Lin1.z + M.Lin4.y*N.Lin2.z + M.Lin4.z*N.Lin3.z + M.Lin4.s*N.Lin4.z;
Mf.Lin4.s = M.Lin4.x*N.Lin1.s + M.Lin4.y*N.Lin2.s + M.Lin4.z*N.Lin3.s + M.Lin4.s*N.Lin4.s;

return Mf;
}
// ------------------------------------------------------------------------------------


//-------------------------------------------------------------------------------------
// FN Multiplicação: --- Matriz [4][4] x Vetor [4][1] = Vetor [4][1] ------------
VETOR4 Matrix4x4Vetor (const matrix4x4 M, const VETOR4 V)
{
   VETOR4 Vf4;

    Vf4.x = M.Lin1.x * V.x + M.Lin1.y * V.y + M.Lin1.z * V.z + M.Lin1.s * V.s;
    Vf4.y = M.Lin2.x * V.x + M.Lin2.y * V.y + M.Lin2.z * V.z + M.Lin2.s * V.s;
    Vf4.z = M.Lin3.x * V.x + M.Lin3.y * V.y + M.Lin3.z * V.z + M.Lin3.s * V.s;
```

```
    Vf4.s = M.Lin4.x * V.x + M.Lin4.y * V.y + M.Lin4.z * V.z + M.Lin4.s * V.s;
return Vf4;
}
//---------------------------------------------------------------------------
```

4.1.8 Composição das transformações

Geralmente, os modelos tridimensionais são submetidos a várias transformações consecutivas. Neste caso é possível determinar, por meio de produtos matriciais, uma única matriz homogênea que descreve a composição de todas as transformações aplicadas ao objeto. A figura 4.8 abaixo representa um objeto no qual foram aplicadas seqüencialmente uma translação, uma variação de escala e uma rotação em torno do eixo *x*.

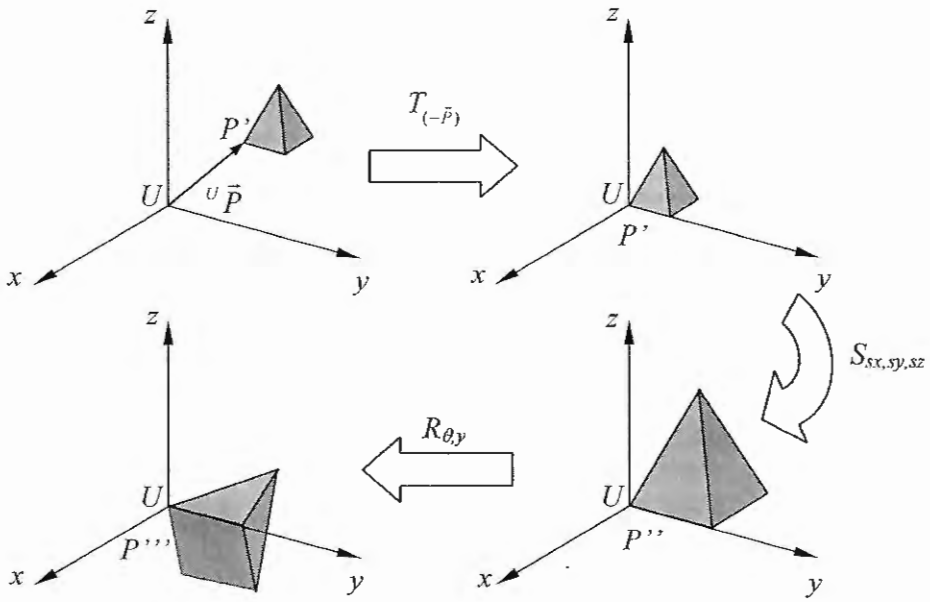

Figura 4. 8 - Seqüência de transformações geométricas aplicadas em um objeto.

Equacionando estas transformações tem-se:

Translação para a origem: $${}^{U}\vec{P}' = T_{(-\vec{P})}\,{}^{U}\vec{P} \quad (32)$$

Variação de escala: $${}^{U}\vec{P}'' = S_{sx,sy,sz}\,{}^{U}\vec{P}' \quad (33)$$

Rotação em torno de y: $${}^{U}\vec{P}''' = R_{\theta,y}\,{}^{U}\vec{P}'' \quad (34)$$

Combinando-se essas equações, obtêm-se a seguinte equação matricial:

$${}^{U}\vec{P}''' = R_{\theta,y}S_{sx,sy,sz}T_{(-\vec{P})}\,{}^{U}\vec{P} \quad (35)$$

Generalizando-se este resultado, pode-se afirmar que:

$${}^{U}\vec{P}^{n} = G\,{}^{U}\vec{P} \quad (36)$$

onde $$G = G_n G_{n-1} G_{n-2} \cdots G_3 G_2 G_1 \qquad (37)$$

Onde a seqüência de transformações descrita no cálculo da matriz composta deve ser efetuada da direita para a esquerda, ou seja, G_1 é a primeira transformação e G_n a última. A inversão desta ordem poderá gerar valores incorretos, já que o produto matricial não é comutativo.

No caso da implementação em C / C++ deve-se definir as variáveis TransTemp, RotTemp e Frm, onde respectivamente tem-se duas matrizes temporárias e uma matriz para armazenar o resultado final de várias transformações.

Por exemplo:

Para efetuar uma TRANSLAÇÃO em (x , y , z), uma ROTAÇÃO em Z (-90°), uma TRANSLAÇÃO em (x , 0.0 , 0.0) e por fim uma ROTAÇÃO em Y (90°), o procedimento seria o seguinte:

Procedimento 4.1.8-(1) – Posicionamento e orientação de um sistema de coordenadas **Frm**, a partir da **Origem** com **TRANS(x,y,z) → RotZ(-90°) → TRANS(x) → RotY(90).**

```
//---------------------------------------------------------------------------

Frm = LoadID4x4();
// Carrega Frm com a Matriz Identidade

TransTemp = Trans ( x , y , z );
// Carrega TransTemp com a Matriz de Translação em (x, y, z)

Rot = -90 * (M_PI / 180);
RotTemp = Rotaz(Rot);
// Carrega RotTemp com a Matriz de Rotação em Z com -90 graus
Frm = MatrixMatrix4x4( RotTemp , TransTemp);
// Atualiza Frm com o resultado da multiplicação matricial RotTemp x TransTemp

TransTemp = Trans ( x , 0 , 0 );
// Carrega TransTemp com a Matriz de Translação em (x, 0, 0)

Frm = MatrixMatrix4x4( Frm , TransTemp);
// Atualiza Frm com o resultado da multiplicação matricial Frm x TransTemp

Rot = 90 * (M_PI / 180);
RotTemp = Rotay(Rot);
// Carrega RotTemp com a Matriz de Rotação em Y com 90 graus
Frm = MatrixMatrix4x4( Frm , RotTemp);
// Atualiza Frm com o resultado da multiplicação matricial Frm x RotTemp
//---------------------------------------------------------------------------
```

4.2 *Rendering*

Rendering é o processo pelo qual uma imagem é sintetizada no computador a partir dos dados que descrevem a cena virtual, ou seja, no caso do presente estudo essa etapa serve para

gerar uma imagem qualitativa e quantitativa do algoritmo (gráfica e com informações em texto) dos resultados obtidos pelo algoritmo. Possibilita, igualmente, criar um ambiente virtual mais amigável para interpretar os resultados obtidos. No estudo apresentado não será detalhado o processo de *render* do ambiente virtual; mas deve-se informar que o processo de *render* foi implementado para mostrar os objetos de 6 modos:

i. ***Shaded-Smooth*** (figura 4.9a), consiste em uma representação tridimensional sombreada, mostrando os objetos de forma realista;

ii. ***Shaded-Smooth EdgesON*** (figura 4.9b), tal qual a representação *shaded-Smooth* porém mostra os limites das faces triangulares em cores de destaque;

iii. ***WireFrame*** (figura 4.10a), representa o objeto tridimensional sem efeitos de luz e sombreamento, somente pelas suas faces triangulares representadas por linhas;

iv. ***Shade-Smooth*** + **Baricentros** + **Normais** (figura 4.10b), mostra o objeto sombreado juntamente com os baricentros de cada uma de suas faces, na cor vermelha, e com linhas azuis representando os vetores normais das faces que compões o objeto;

v. ***Shade -Smooth EdgesON*** + **Baricentros** + **Normais** (figura 4.11a), mostra a representação conforme o item acima, porém com os limites das faces triangulares em cores de destaque;

vi. ***Bounding Box*** (figura 4.11b), mostra o objeto representado como um paralelepípedo com as dimensões principais do objeto (largura, comprimento e profundidade).

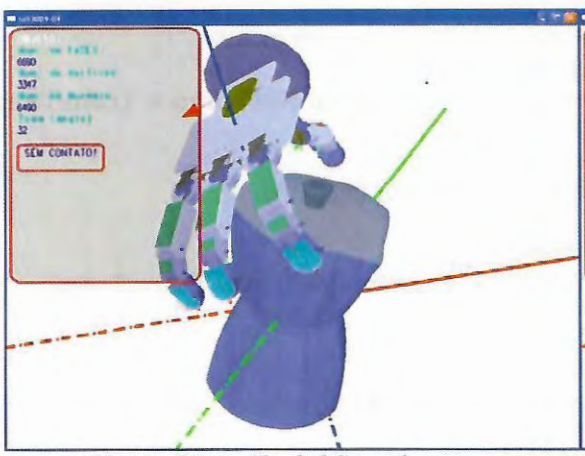

Figura 4.9 a – *Shaded-Smooth.*

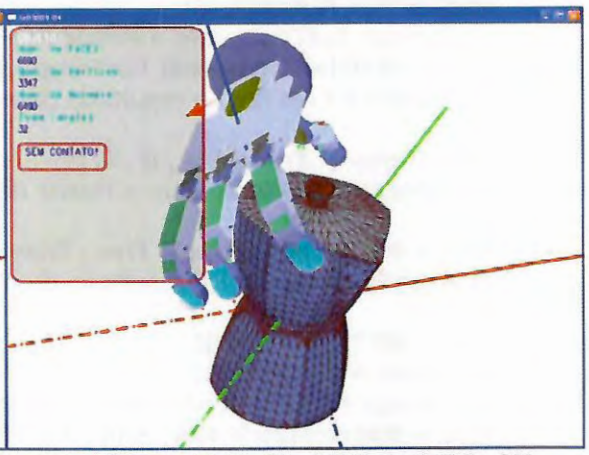

Figura 4.9 b – ***Shaded-Smooth EdgeON.***

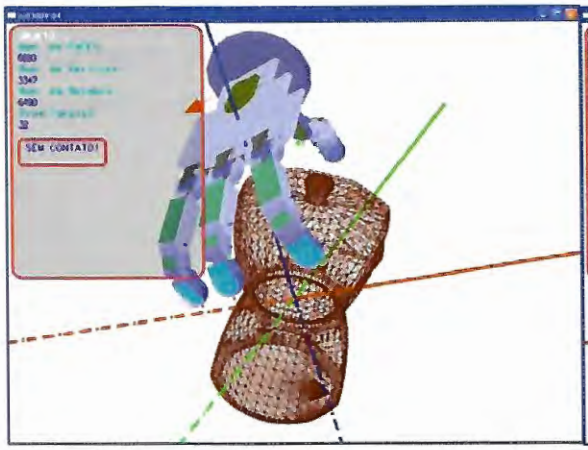

Figura 4.10 a – *Wireframe.*

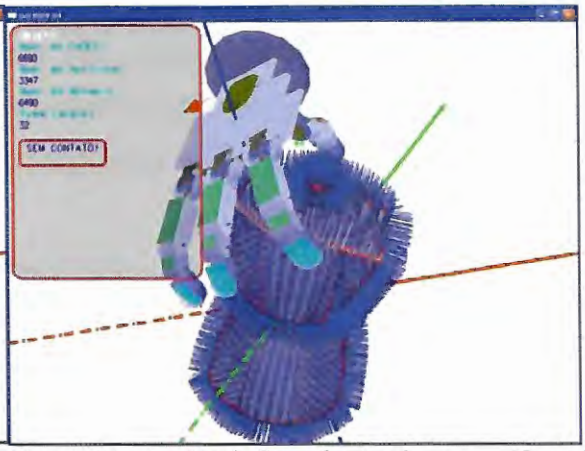

Figura 4.10 b – *Shade-Smooth* + Baricentros + Normais.

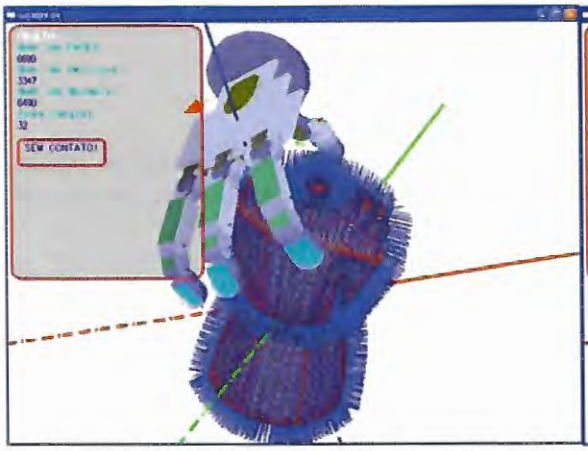

Figura 4.11 a – *Shaded-Smooth EdgeON* + Baricentros + Normais.

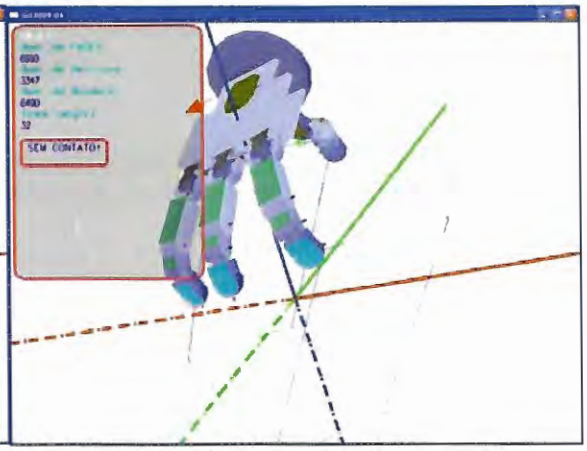

Figura 4.11 b– *Bouding Box.*

4.3 Cálculo de Contato

4.3.1 Análise da Empunhadura como uma interação de Múltiplos Contatos

Quando uma pessoa vendada toca um objeto somente com um único toque de um dedo, pouca informação pode ser extraída dessa interação. Porém, quando a mesma pessoa toca o objeto com todos os dedos, mesmo em um único toque, na maioria das vezes pode-se reconhecer um objeto conhecido sem precisar deslizar os dedos pelo objeto, explorando seus limites. Isso acontece pelo fato de que as superfícies do objeto apresentam aos dedos limites físicos bem definidos que reagem com cada dedo no sentido contrário à força de manipulação aplicada ao objeto. Tais fatores fazem com que sinta-se uma pressão na ponta dos dedos que dará informações sobre o objeto manipulado e, quando essa informação é, de forma associativa, relacionada aos dados armazenados na memória, aos vários objetos conhecidos e

repertoriados no cérebro, busca-se, desse modo, identificar qual objeto enquadra-se melhor com as estimativas de contato (figura 4.12). Pode-se intuir então, que está se procedendo à uma ação que, ao mesmo tempo que analisa os limites físicos do objeto (tamanho e peso), compara e analisa, igualmente, as restrições que seus limites físicos impõem aos dedos quando esses percorrem o objeto com o objetivo de explorá-lo.

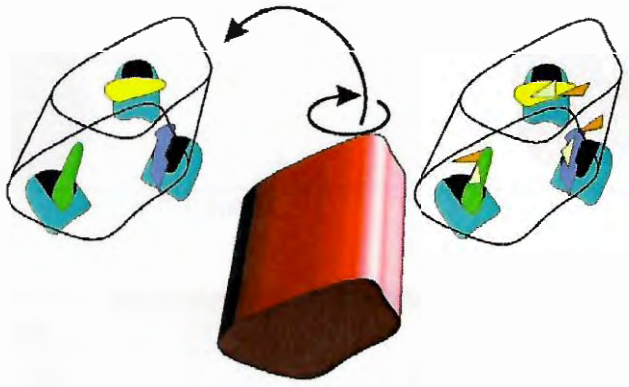

Figura 4. 12 – Cruzamento de informações com as medidas feitas através dos dedos (em azul) em contato com o objeto e suas restrições (esquerda) e as hipóteses de contato geradas, representadas por triângulos amarelos e âmbar mostrados à direita.

Desse ponto de vista, é possível concluir que somente um dedo não fornece muita informação sobre o objeto manipulado, pois isso se deve a pouca restrição de movimento que um dedo aplica ao objeto. No entanto a cada dedo adicional (figura 4.13) em contato com o objeto, novas restrições de movimentos são colocadas de forma que com somente três dedos pode-se identificar a maioria dos objetos cotidianos conhecidos. Com quatro dedos pode-se então aumentar a precisão e o índice de acertos de forma considerável.

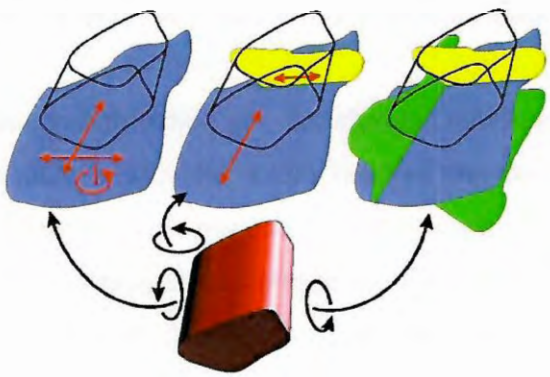

Figura 4. 13 – Incremento gradual de restrições durante empunhadura de um objeto. Cada plano CIANO, AMARELO e VERDE, representam um novo dedo. As setas indicam as restrições (forças e momentos).

Desse modo, conforme exposto acima, para a simulação de contato e reconhecimento desse objeto por uma mão robótica, o procedimento utilizado foi dividido em duas etapas:

1) – Etapa de "aprendizado", feita em uma etapa *off-line*, em que são coletadas as características do objeto, número de faces, vetor normal e distâncias entre as faces, entre outros, de forma a facilitar uma busca rápida dessas características na memória do computador. O objeto armazenado não necessita, por razões de armazenamento, ser muito detalhado, porém deve conter informações suficientes para o reconhecimento do objeto na fase *on-line*.

2) – Etapa de interação com o objeto. Esta etapa acontecerá em tempo real (*online*), onde serão comparadas as medidas de contato tomadas pela mão em contato com o objeto e os dados armazenados na memória quando na fase *off-line*. O objeto usado nessa fase deverá ser rico em detalhes tal qual um objeto observado no dia-a-dia. Tal objeto também será o mesmo exibido na tela durante a fase de desenho do OpenGL (*rendering*).

4.3.2 Detecção do contato

Após definido o objeto a ser simulado, este será posicionado no cenário conforme a necessidade do teste a ser executado, no qual a mão virtual será inserida nesse cenário para interagir com o objeto virtual em uma simulação que representa a mão fechando os dedos, visando agarrar o objeto. (figura 4.14).

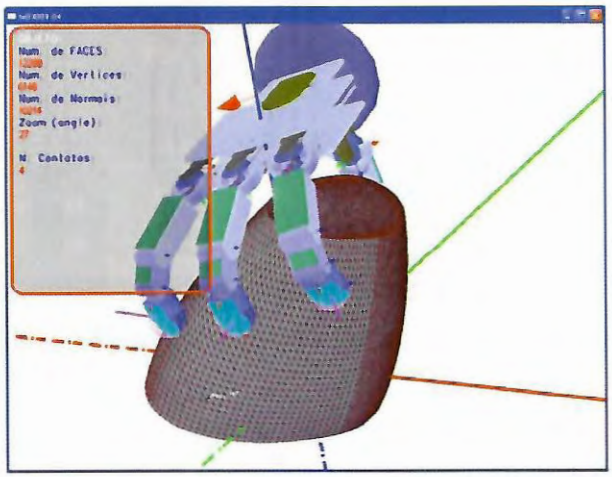

Figura 4. 14 – Objeto e Mão Virtual interagindo no cenário.

Desse modo, a mão parte de uma posição inicial (parcial ou totalmente aberta) para a posição fechada como mostra a seqüência de figuras 4.15a, 4.15b, 4.15c e 4.15d, (sem necessariamente atingir esta posição final) mediante pequenos incrementos nos ângulos das articulações dos dedos (figura 4.16), tendo em vista que para cada incremento é feita uma

varredura entre as pontas dos dedos da mão e todos os pontos possíveis do objeto virtual (matemático).

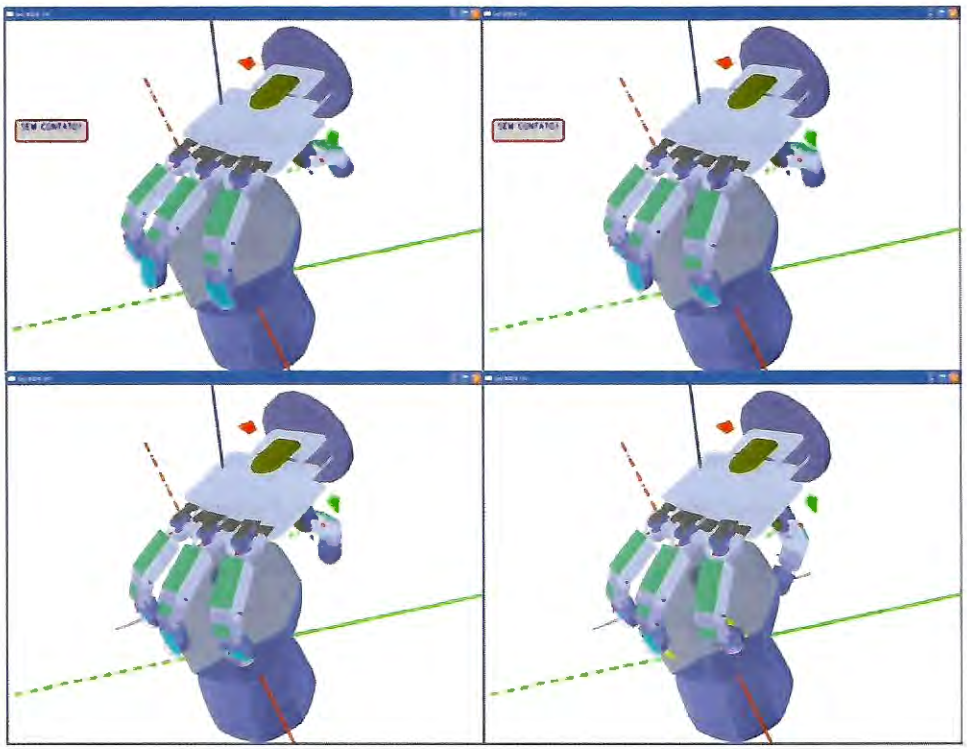

Figura 4.15 a, Figura 4.15 b, Figura 4.15 c e Figura 4.15 d – Seqüência mostrando o ato de apanhar o objeto.

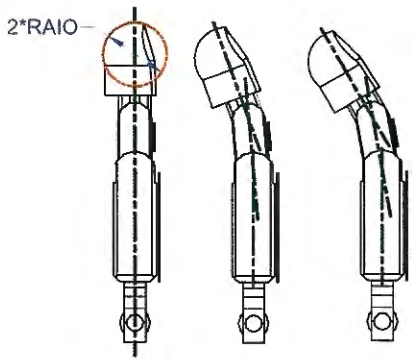

Figura 4. 16 – Incrementos dos ângulos dos dedos.

É mostrado a seguir o algoritmo de como é feito a checagem de contato para os dedos da mão.

```
FOR i = 0 TO NumFaces
{
        SE contato (PontaDedo01)
                ENTÃO       retorna DEDOREPLY(Dedo01);
        SE contato (PontaDedo02)
                ENTÃO       retorna DEDOREPLY(Dedo02);
        SE contato (PontaDedo03)
                ENTÃO       retorna DEDOREPLY(Dedo03);
        SE contato (PontaDedo04)
                ENTÃO       retorna DEDOREPLY(Dedo04);
}
```

Para o cálculo do contato da função **contato (PontaDedo*n*)** acima procede-se conforme abaixo.

1 – É calculada a distância *d* entre centro C, figura 4.17, da esfera imaginaria, geometricamente semelhante a ponta de cada dedo, e sua projeção na face triangular segundo o procedimento abaixo:

2 – Dos vértices V1, V2 e V3, serão definidos os vetores $\overrightarrow{V1V2}$, $\overrightarrow{V1V3}$ e $\overrightarrow{V2V3}$.

3 – A normal da face $\vec{n}$ é calculada através do seguinte modo:

$$\vec{n} = \overrightarrow{V1V2} \times \overrightarrow{V1V3} \tag{38}$$

4 – É definido o vetor $\overrightarrow{V1C}$ do tomado do vértice v1 e centro C da esfera.

5 – É calculada a distância *d* através do produto escalar conforme abaixo:

$$d = \vec{n} \bullet \overrightarrow{V1C} \tag{39}$$

6 – É verificada se *d* é menor que o raio *r* da esfera imaginária do dedo e também um valor positivo, em caso verdadeiro é feita a verificação se a projeção do centro C, o ponto P se encontra dentro da face triangular, conforme os próximos passos.

7 – Como descrito em no item 2 os vetores $\overrightarrow{nV1V2}$, $\overrightarrow{nV1V3}$ e $\overrightarrow{nV2V3}$ são calculados pelo produto vetorial entre $\vec{n}$ e $\overrightarrow{V1V2}$, $\overrightarrow{V1V3}$ e $\overrightarrow{V2V3}$, respectivamente.

8 – Através de produtos escalares é verificado se o ponto P se encontra situado dentro do triangulo formado pelo três vértices da seguinte forma;

a) São calculados os produtos escalares *d*12, *d*13 e *d*23, como abaixo:

$$d12 = \overrightarrow{V1C} \bullet \overrightarrow{nV1V2} \tag{40}$$

$$d13 = \overrightarrow{V1C} \bullet \overrightarrow{nV1V3} \tag{41}$$

$$d23 = \overrightarrow{V1C} \bullet \overrightarrow{nV2V3} \tag{42}$$

É feita a seguinte verificação lógica ao final:

SE ($d12 \leq 0$ **AND** $d13 \leq 0$ **AND** $d23 \leq 0$)
ENTÃO TEMOS CONTATO.

A seguir é mostrada figura 4.17 mostrando o esquema de modelagem geométrica do cálculo de contato entre o dedo a face triangular.

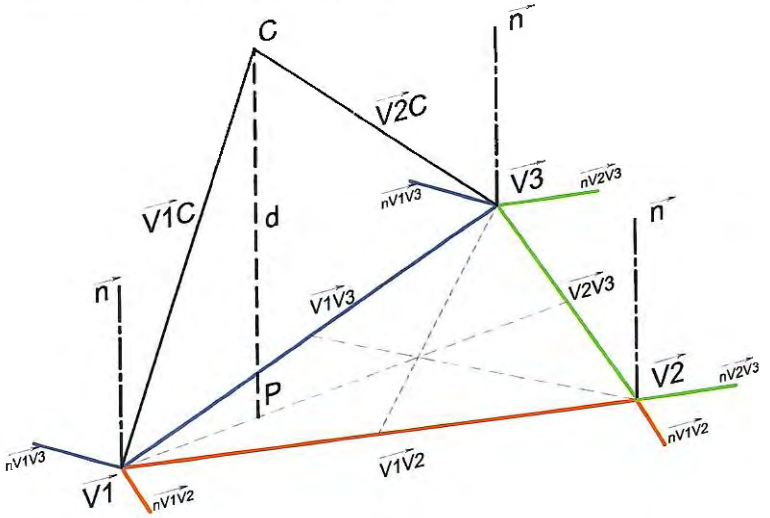

Figura 4. 17 – Modelagem vetorial para cálculo de contato entre dedo e face triangular.

A seqüência de figuras 4.18a, 4.18b, 4.18c e 4.18d, mostra algumas possibilidades de posições que podemos encontrar a esfera de centro C em relação a face triangular.

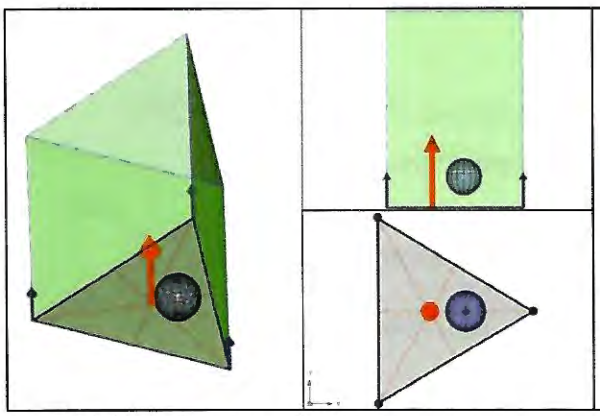

Figura 4.18 a – Iminência de Contato.

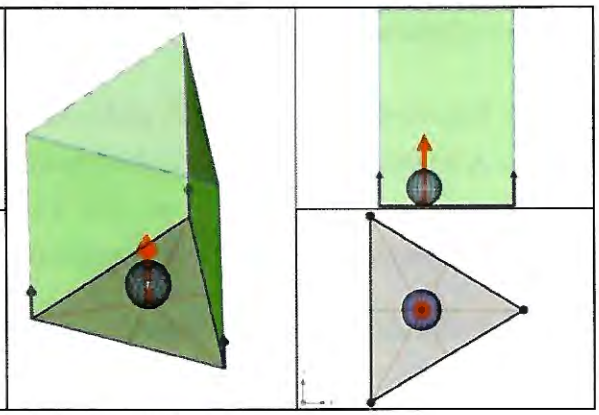

Figura 4.18 b – Contato .

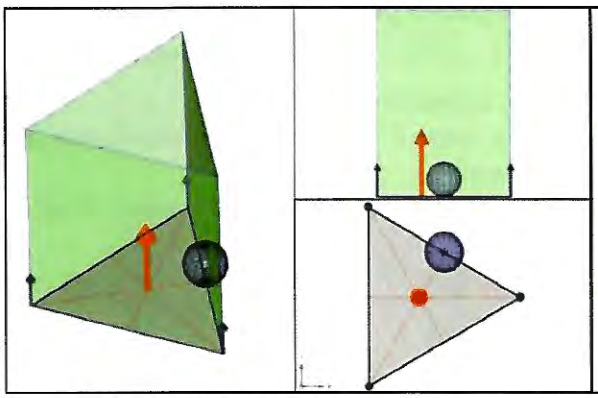

Figura 4.18 c – Limite do Contato.

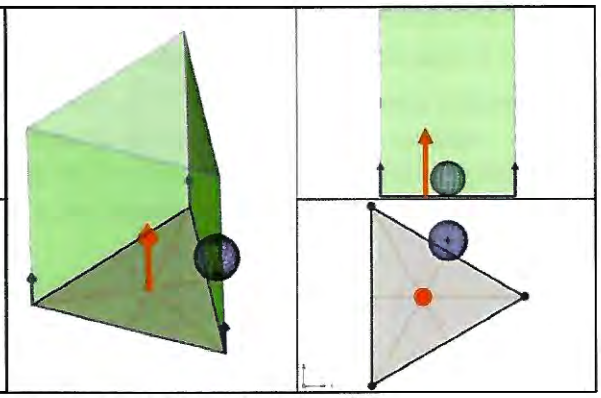

Figura 4.18 d – Sem Contato.

Assim, verificado o toque desse dedo com o objeto, serão armazenados: os ângulos das articulações do dedo em contato, o ponto de contato com o objeto e o vetor normal da face que foi tocada no objeto. Para os demais dedos, os ângulos das articulações continuarão a ser incrementados até o toque do dedo com o objeto, ou até que o dedo encontre-se completamente fechado, nesse caso esse dedo não fará parte do cálculo de contato.

4.3.3 Modelagem da Mão

Para essa simulação será usada uma mão antropomórfica direita de três dedos e um polegar com 16 graus de liberdade, conforme a figura 4.19, em que são apresentadas também as dimensões principais da mão simulada.

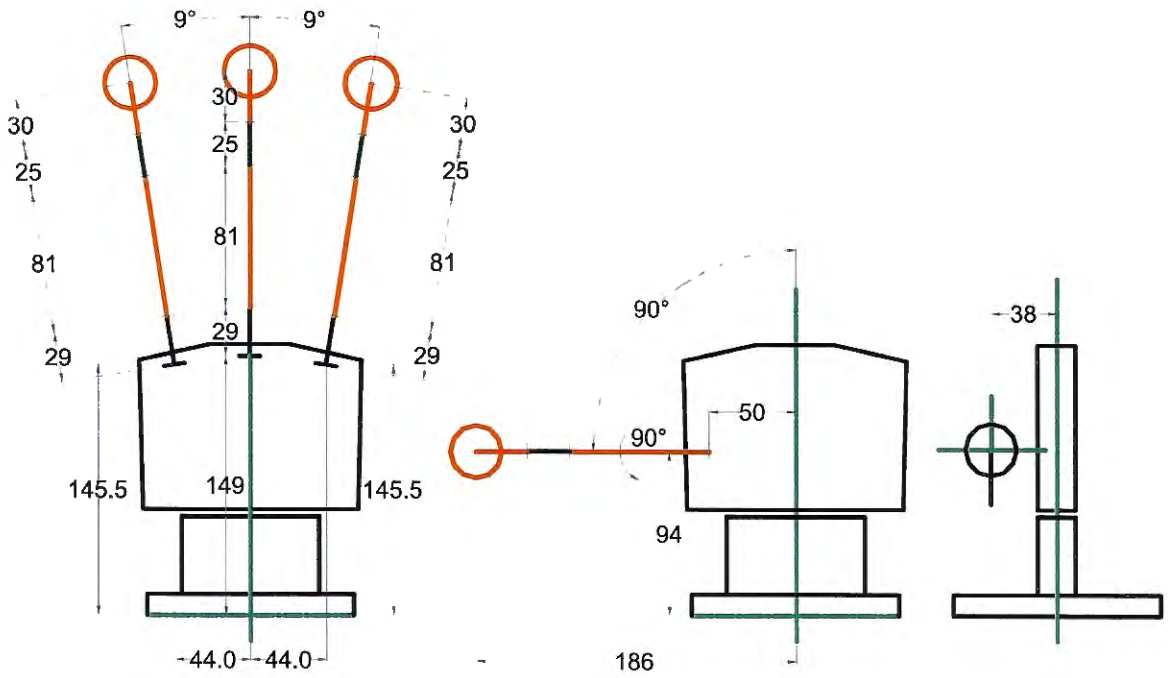

Figura 4. 19 – Esquema da cadeia cinemática da mão.

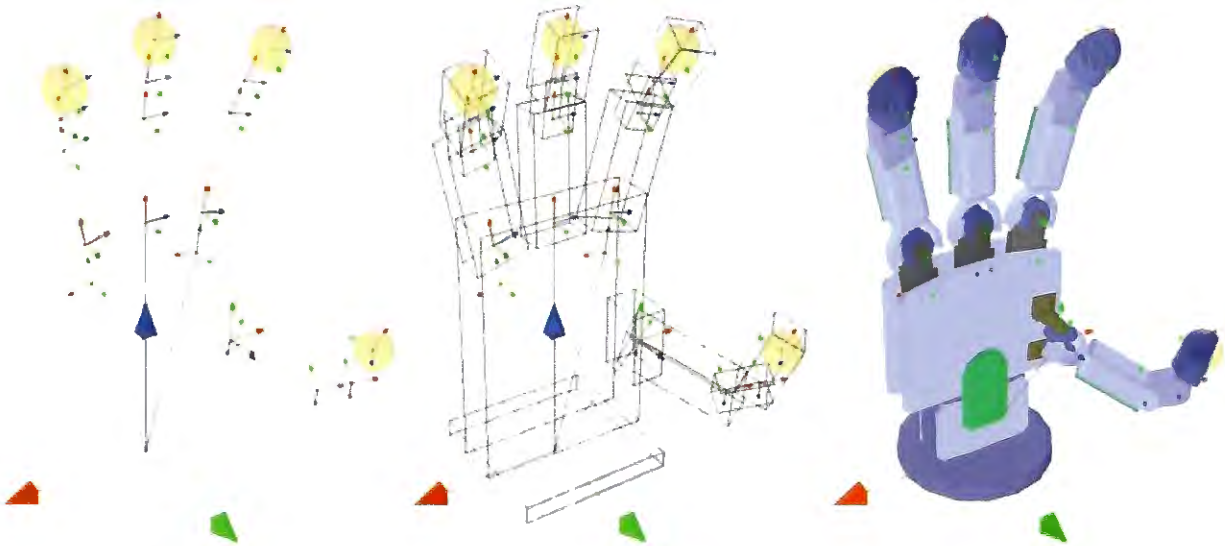

Figura 4. 20 – Seqüência mostrando a implementação da cadeia cinemática em OpenGL.

4.3.4 Cadeia Cinemática da Mão

Na modelagem da cadeia cinemática da mão, a seqüência de implementação (figura 4.20) foi dividida em duas partes;

i – Base da mão, que corresponde ao pulso, a palma e as bases (conexões) para cada dedo (figura 4.21);

ii – Dedos, que serão conectados às suas respectivas bases formando a mão propriamente dita (figura 4.22). Cada dedo contém 4 graus de liberdade.

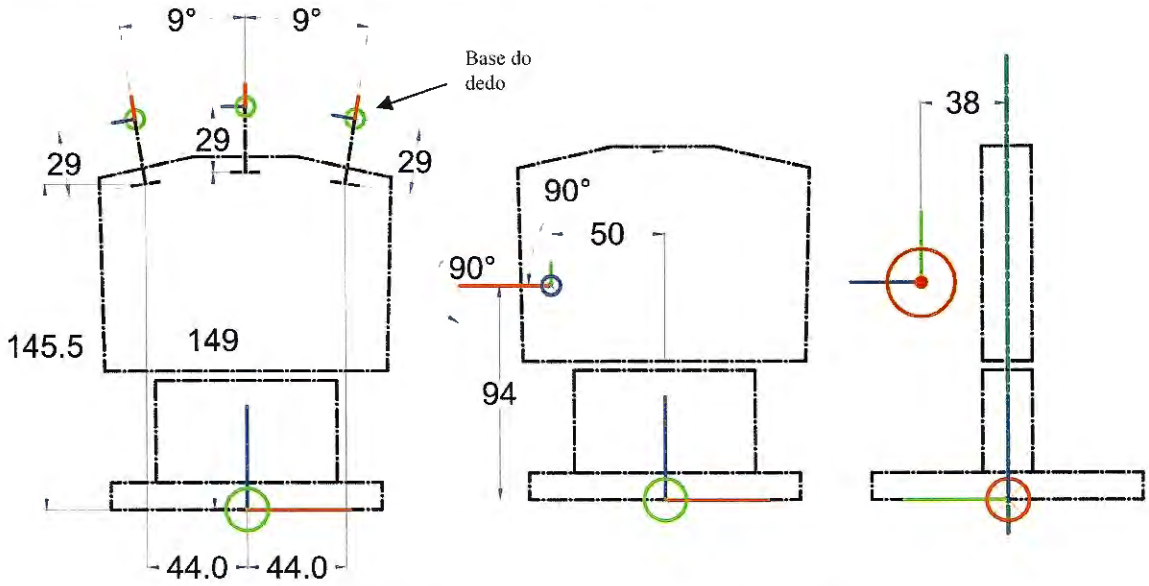

Figura 4. 21 – Posicionamentos das Bases dos dedos.

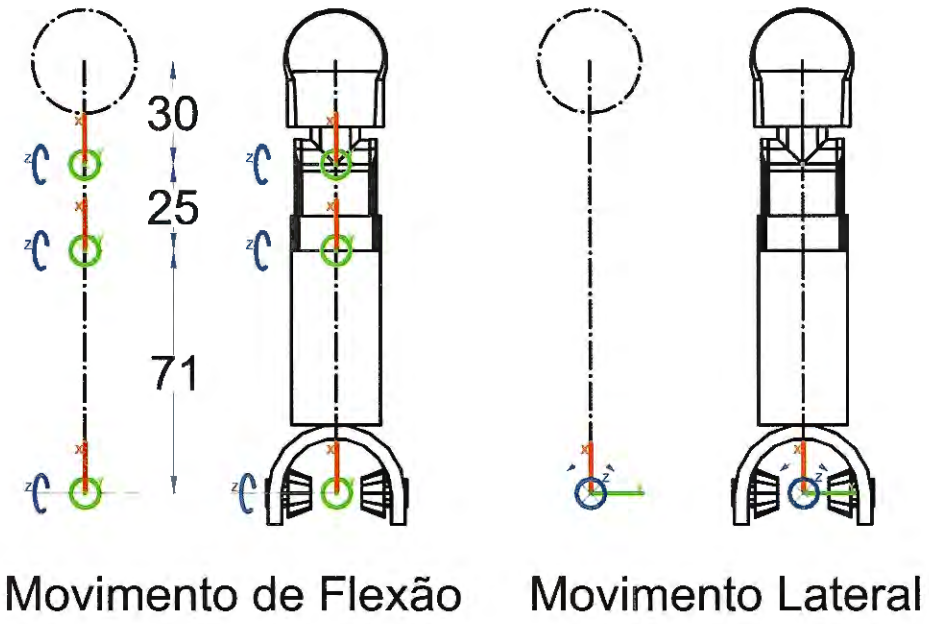

Figura 4. 22 – Modelagem de cada dedo articulado.

Cada uma das bases da mão é posicionada por meio de matrizes de Transformações Homogêneas de Translação e Rotação. Descreve-se abaixo como foi posicionada cada base para cada dedo:

- Primeiro é definido um tipo de variável chamado VETOR4, sendo que:

$$\text{VETOR4} = [x \quad y \quad z \quad s]^T \tag{43}$$

Note que será atribuído para s o valor 1.

- Depois é definido um tipo de variável chamada matrix4x4, sendo que;

$$\text{matrix4x4}=\begin{bmatrix}\text{VETOR4} & \text{Lin1}\\ \text{VETOR4} & \text{Lin2}\\ \text{VETOR4} & \text{Lin3}\\ \text{VETOR4} & \text{Lin4}\end{bmatrix}=\begin{bmatrix}\text{Lin1.x} & \text{Lin1.y} & \text{Lin1.z} & \text{Lin1.s}\\ \text{Lin2.y} & \text{Lin2.y} & \text{Lin2.z} & \text{Lin2.s}\\ \text{Lin3.x} & \text{Lin3.y} & \text{Lin3.z} & \text{Lin3.s}\\ \text{Lin4.x} & \text{Lin4.y} & \text{Lin4.z} & \text{Lin4.s}\end{bmatrix} \quad (44)$$

- São definidas matrix4x4 temporárias e são executadas etapas de atribuições e multiplicações de matrizes conforme o procedimento 4.1.8-(1), visto anteriormente, porém para os pontos Bd1, Bd2, Bd3 e Bd4, a seguir.

BaseDedo01 (polegar):

Bd1 = Trans (-50.0 , 38.0 , 94.0) → RotX(90°) → RotZ(180°)

BaseDedo02:

Bd2 = Trans (-44.0 , 0.0 , 145.5) → RotY(-9°) → Trans (0.0 , 0.0 , 29.0) → RotY(-90)

BaseDedo03:

Bd3 = Trans (0.0 , 0.0 , 178.0) → RotY(-90°)

BaseDedo04:

Bd4 = Trans (-44.0 , 0.0 , 145.5) → RotY(9°) → Trans (0.0 , 0.0 , 29.0) → RotY(-90)

Vale lembrar que anteriormente a cada primeira translação descrita acima é feita uma pré-multiplicação da matriz que contém essa primeira translação com a matriz que contém a posição e orientação da mão no espaço (Frm);

$$Frm=\begin{pmatrix} v_{xx} & v_{xy} & v_{xz} & x_{hnd}\\ v_{yx} & v_{yy} & v_{yz} & y_{hnd}\\ v_{zx} & v_{zy} & v_{zz} & z_{hnd}\\ 0 & 0 & 0 & 1\end{pmatrix};\quad Bf4 = Frm \times Trans \quad (45)$$

4.3.4.1 Função HandDir (MãoDireita)

Depois de criada matematicamente, orientada e desenhada, a mão é colocada a interagir com o objeto, onde é feito um calculo de colisões de faces. Foi criada então, uma função chamada HandDir (nomeada em uma mescla de Inglês com Português) que corresponde a uma mão antropomórfica direita de um robô, cujo protótipo da função é apresentado abaixo:

HNDReply HandDir (**matrix4x4** HNDframe, **STRUCT_OBJSTAT** Objeto);

Como pode-se observar, as entradas dessa função são uma variável do tipo **matrix4x4**, que corresponde à matriz de transformação homogênea do sistema de coordenadas (orientação e posição) da mão no espaço pronta para empunhar o objeto e, a uma variável (estrutura de dados) do tipo **STRUCT_OBJSTAT**, que armazena o número total de vértices, faces, normais, uma matriz (*array*) com a lista de normais de vértices do objeto (para *rendering* apenas), uma matriz (*array*) com a lista dos vértices e uma matriz (*array*) com o lista das faces.

```
//-------------------------------------------------------------------------------
struct STRUCT_OBJSTAT
{
        int TotalVertices;
        int TotalFaces;
        int TotalNormals;
        VETOR4 *NORMALlist;
        STRUCT_VERTICE *VERTEXlist;
        STRUCT_FACE *FACElist;
};
//-------------------------------------------------------------------------------
```

Tal função é uma função que retorna uma estrutura de dados do tipo **HNDReply**, a qual corresponde ao "retorno" ou *feedback* da mão que é mostrada a seguir:

```
//-------------------------------------------------------------------------------
//Cria estrutura de Retorno da MÃO ----------------------------------------------
struct HNDReply
{
        VETOR4 Xcf1, Xcf2, Xcf3, Xcf4; // Ptos de contatos dedos 1, 2, 3 e 4
        VETOR4 Acf1, Acf2, Acf3, Acf4; // Vetor Normal à face em Contato
        VETOR4 ang1, ang2, ang3, ang4; // Âng. de Juntas dedos 1, 2, 3 e 4.
        FcInd4 Fc;
};
//-------------------------------------------------------------------------------
```

Abaixo, uma explicação mais detalhada sobre os retornos da função HandDir:

Xcf1, Xcf2, Xcf3 e Xcf4 – São os pontos de contato das pontas dos dedos que atingiram o objeto em uma determinada interação. Como cada ponto corresponde a um **VETOR4**, tem-se então os seguintes dados Xcf1(x1,y1,z1,s1), Xcf2(x2,y2,z2,s2), Xcf3(x3,y3,z3,s3) e Xcf4(x4,y4,z4,s4). Neste caso s1 = s2 = s3 = s4 = 0 (zero), pois são tratados como vetores espaciais.

Acf1, Acf2, Acf3 e Acf4 – São vetores normais correspondentes às faces que entraram em contato com o objeto e são compostos de estruturas **VETOR4.** Tem-se então: Acf1(x1,y1,z1,s1), Acf2(x2,y2,z2,s2), Acf3(x3,y3,z3,s3) e Acf4(x4,y4,z4,s4). Neste caso s1 = s2 = s3 = s4 = 0 (zero), pois são tratados como vetores espaciais.

ang1, ang2, ang3 e ang4 – São os ângulos das articulações dos dedos e também são compostos de estruturas **VETOR4**, tem-se então: ang1(x1,y1,z1,s1), ang2(x2,y2,z2,s2), ang3(x3,y3,z3,s3) e ang4(x4,y4,z4,s4). Para o caso dos ângulos das articulações, todos os valores do VETOR4 são utilizados para as coordenadas '*s*', para os quais podem ser atribuídos quaisquer valores.

Fc – É o índice de identificação do dedo em contato, pois a cada incremento Δang_n nos ângulos das articulações (ang1, ang2, ang3 e ang4) é feita uma checagem verificando se algum dedo entrou em contato com o objeto e essa estrutura composta de quatro variáveis tipo **int** (dID1, dID2, dID3 e dID4) retorna um valor **true** no caso de o dedo ter feito contato com o objeto e um valor **false**, caso contrário.

Após todos os dedos terem entrado em contato com o objeto, ou chegado à posição "fechado" é executada rotina de hipótese de contato.

4.4 O Objeto

Para representar o objeto empunhado, foram criados dois tipos de objetos:

- Objeto Refinado, que deverá conter amplas informações e detalhes em números de faces, para descrever o objeto tal qual o objeto real a ser manipulado. Dessa forma, a aproximação será bem próxima da realidade. Este Objeto será *renderizado* em uma fase *on-line* do teste de contato e também será usado para coletar dados de contato simulando um objeto do dia-a-dia.

- Objeto Simplificado, que deverá conter apenas informações suficientes para o seu reconhecimento, sendo o número de faces apenas o necessário para o reconhecimento do objeto detalhado. Esse objeto deverá ser armazenado na memória em uma fase *off-line*.

A seguir, os objetos serão explicados com mais detalhes.

4.4.1 O Objeto Refinado

Os objetos que serão tratados como Objetos Refinados (figura 4.23a e 4.23b) deverão ser desenhados pelo programa de CAD, de acordo com os detalhes que esses apresentam no dia-a-dia, ou seja, com poucas simplificações de formas e devem contar com detalhes importantes das formas que compõem estes objetos.

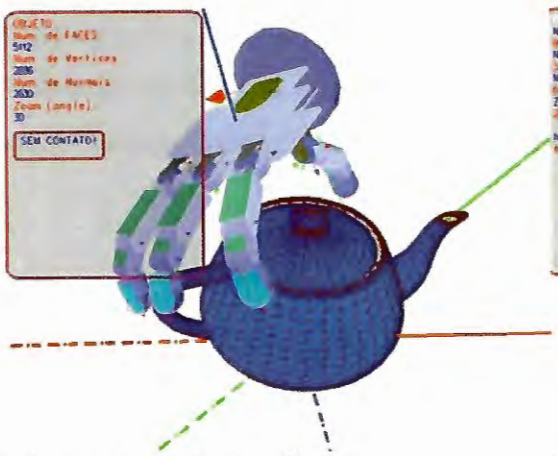

Figura 4.23 a – Objeto Refinado – Chaleira.

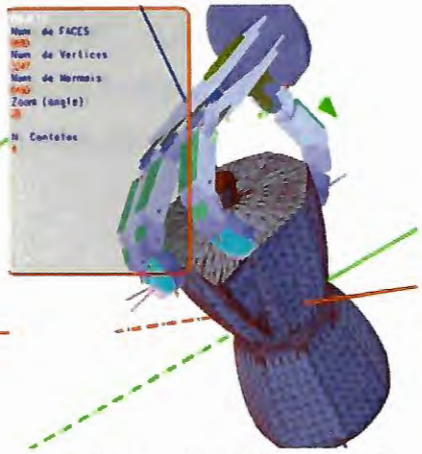

Figura 4.23 b– Objeto Refinado – Cafeteira.

4.4.2 O Objeto Simplificado (aproximação entre Objeto Real *versus* Digital Simplificado)

O Objeto Simplificado (figura 4.24) é gerado a partir de uma aproximação de um objeto refinado, com a intenção de ser uma representação simplificada, ele é portanto, gerado com menos informação e sua função é representar o objeto refinado de forma compacta e simplificada, porém de forma consistente e robusta para o cálculo das hipóteses de contato. Ressalta-se que o objeto da figura 4.24 representa um dos objetos das figuras 4.23a e 4.23b, porém de forma simplificada.

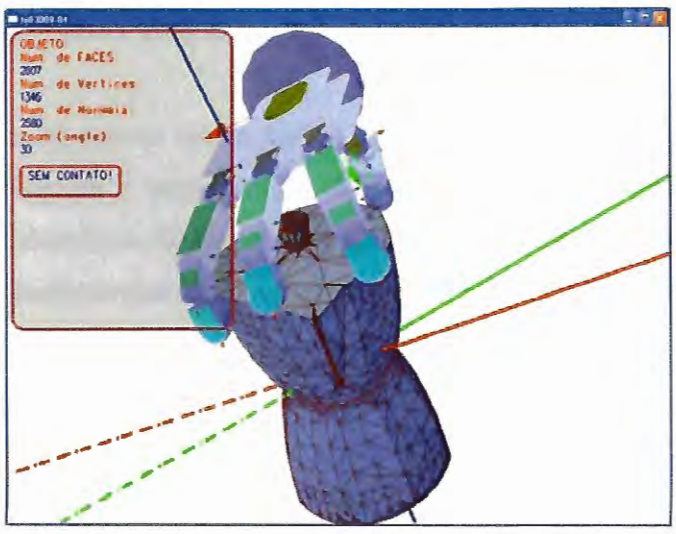

Figura 4. 24 – Objeto Simplificado.

4.5 Análise e Armazenamento do Objeto (fase *OFF-LINE*)

O objetivo de se criar o objeto simplificado é o de reunir o máximo de informações consistentes para um bom reconhecimento, utilizando-se, porém, pouca memória e garantindo que terá pouco cálculo durante a fase *on-line*. Neste caso, não só é feita uma análise, como também o objeto é devidamente armazenado na memória através de artifícios heurísticos de maneira a simplificar a busca de uma boa solução para uma hipótese de contato sem gerar uma busca em profundidade, que é muito dispendiosa em termos computacionais (ver [25], capítulo 23).

Tal objeto é primeiramente desenhado no programa de CAD Autodesk INVENTOR R10, sendo então exportado para o formato .STL, para em seguida, ser importado para dentro de um outro programa de modelagem, chamado 3D Studio Max 8. No interior do 3D Studio podem ser feitos de acordo com a necessidade: ajustes de exposição (normais de vértices), inserção de propriedades de cor, brilho, especularidade de formas (alterando o formato em escala ou pelos vértices) e muitos outros ajustes que até ultrapassam o escopo do trabalho. Após ser devidamente trabalhado, o Objeto é então exportado no formato .X (X-File) que é um formato de texto que pode ser facilmente lido através de uma implementação em linguagem C. Esse último fator foi decisivo na escolha desse formato que apesar de ser voltado mais para aplicações de entretenimento, pode ser facilmente convertido em uma ferramenta científica, tendo em vista que um objeto simplificado pode ser computado facilmente em dados para posterior acesso.

O formato .STL poderia ter sido empregado, somente para o objeto (desenho e extração de dados), pois para o desenho do manipulador virtual (mão antropomórfica) este formato não apresenta bons aspectos de *rendering*. Ele exigiria a implementação de uma rotina para atribuir as normais de vértices para o *shading* e para estabelecer critérios de *smooth* para uma visualização suavizada, o que fugiria do foco do estudo.

4.5.1 Número de faces *versus* performance

Para se fazer uma busca das possíveis faces admissíveis que compõem o objeto e que correspondem ao que mais se aproxima das faces tomadas pelo tato da mão antropomórfica, deve-se realizar uma abordagem de forma a tentar reduzir o custo computacional durante a fase *on-line* da detecção e cálculo. Se isso não for feito, uma busca completa (sem o mínimo de critério) para um objeto de m faces, seria nada mais que uma busca de comparação de faces da ordem de m^m, tornando o cálculo inviável, mesmo para objetos simples do dia a dia, os quais muitas vezes necessitam de um número de faces da ordem de 10.000 faces para serem

descritos, uma busca completa nesse objeto levaria 317 anos, se cada interação demorasse 1μs (citado em [12], capitulo 4).

Desse modo, deve-se realizar uma análise heurística do problema para a escolha de uma abordagem viável, que ao mesmo tempo proporcione uma "boa solução" para o problema sem que seja necessário realizar buscas exaustivas, como as buscas de profundidade primeiro e buscas de extensão primeiro (ver [25], capitulo 23).

Com essa finalidade, o objeto foi estudado e armazenado com base nas seguintes características:

- O número de faces do objeto;
- As distâncias MÁXIMAS entre os pares de faces;
- As distâncias MÍNIMAS entre os pares de faces;
- Os ângulos entre as normais de duas faces.

A partir disso foi estabelecida uma forma de armazenamento e acesso do objeto de forma a acelerar o cálculo eliminando os resultados inviáveis no início do processo de seleção de faces que entrarão para o cálculo de hipótese de contato.

A seguir esses passos, serão melhor descritos.

4.5.2 Numeração de faces (1ª fase do armazenamento)

O primeiro passo é realizar a numeração das faces para identificação e acesso através de um índice. Isto é feito a partir da criação de uma estrutura de dados chamada STRUCT_FACE, onde são armazenados os seguintes parâmetros do Objeto:

- O conjunto de 3 vértices (Vert1, Vert2 e Vert3) que formam uma face;
- Os Valores máximo e mínimo (VMAX e VMIN) em que o objeto se situa nos eixos XYZ;
- O Baricentro (Xo) do Objeto;
- A extrusão do Baricentro (X1);
- A distância entre a Face e a Origem do Objeto;
- O vetor Normal da Face (NormaFace);
- Os índices dos vértices (indVert1, indVert2 e indVert3) necessários no *rendering* (no desenho propriamente dito);
- Os índices das normais (indNormal1, indNormal2 e indNormal3) também necessários no *rendering* (iluminação e shade);
- Identificação do Material.

A implementação de tal estrutura em C / C++ pode ser consultada abaixo para maiores detalhes:

```
//-------------------------------------------------------------------------------
struct STRUCT_FACE
{
   VETOR4 Vert1, Vert2, Vert3;                    // Vértices da FACE
   VETOR4 VMAX, VMIN;
   VETOR4 VBOX[8];
   VETOR4 Xo;                                      // Baricentro da FACE
   VETOR4 X1;                                      // Pto de vn (VNormal*Mg)
   float Distance;                                 // Dist de Xo para Origem (o)
   VETOR4 NormalFace;                              // Normal da FACE
   DWORD indVert1,  indVert2,  indVert3;           // Índice das VÉRTICE
   DWORD indNormal1, indNormal2, indNormal3;       // Índice das NORMAIS
   int   material;                                 // ID do MATERIAL
};
//-------------------------------------------------------------------------------
```

Após a criação desses artifícios pode-se então extrair dados do objeto e agrupá-los de forma a criar uma estrutura que defina as estatísticas do objeto, tais como os números de Faces, de Vértices, de Normais e de Materiais (para o *rendering*), a lista de Normais, de Vértices e de Faces e, por fim as distâncias relativas a cada par de face que compõe o objeto, conforme a estrutura de dados a seguir:

```
//-------------------------------------------------------------------------------
struct STRUCT_OBJSTAT
{
    int                TotalVertices;
    int                TotalFaces;
    int                TotalNormals;
    VETOR4             *NORMALlist;
    STRUCT_VERTICE     *VERTEXlist;
    STRUCT_FACE        *FACElist;
    float              *Distancia;
};
//-------------------------------------------------------------------------------
```

Através da estrutura de dados STRUCT_OBJSTAT que mostra as estatísticas do objeto, pode-se calcular as distâncias dos pares de faces e armazená-las dentro de uma outra estrutura do tipo DATASET, que é uma abstração matemática de um conjunto. Em C / C++ sua implementação é conforme descrito:

```
//-------------------------------------------------------------------------
struct DATASET
{
   vector <float> fvalor;
   vector <int>   KId;
   vector <int>   LId;
};
//-------------------------------------------------------------------------
```

Sendo que **fvalor** representa uma variável vector (ver [2], capitulo 29) do tipo ponto flutuante, onde são armazenados **dmin** (distância mínima), **dmax** (distância máxima) e **alfa**

(ângulo entre as superfícies normais). As variáveis KId e LId são os índices de faces das faces k e l, analogamente, que serão guardados para acesso e reconhecimento do objeto.

Sendo o par de faces k e l pertencente ao objeto, forma-se então o conjunto (DATASET) Fo_2 das distâncias e ângulos entre os pares de faces (k , l), onde:

$$Fo_2 = \{ (1 , 1) ; (1 , 1) ; (1 , n) ; ... ; (n , 1) ; (n , 2) ; (n , 3) ; ... ; (n , n) \} \quad (46)$$

$$Fo_2 = \{ (i , j) \mid i = 1, ..., n ; j = 1, ..., n \} \text{ de tamanho } n^2. \quad (47)$$

O conjunto Fo_2 guardará para cada par de face a máxima e a mínima distância entre os pares de faces e o ângulo entre as superfícies normais que compõe o objeto.

Então, para cada par de face (i , j), tem-se [dmax,oc,(ij), dmin,oc,(ij) e alfa(ij)]. Assim quando entra-se com um par de faces na função de acesso para esse conjunto serão retornadas as três grandezas descritas acima; (i , j) → [dmax,oc,(ij) ;dmin,oc,(ij) ; alfa(ij)].

O cálculo das distâncias máximas, será feito comparando-se as distâncias entre os vértices de duas faces, sendo então comparadas 9 possibilidades a serem verificadas antes de se chegar à maior distância das faces.

Para cálculo das distâncias mínimas, é realizado um procedimento por meio de uma comparação entre os produtos escalares e os 6 vértices que estão envolvidos entre as duas faces. O menor dos produtos escalares entre o vetor normal da face e o respectivo vetor orientado entre dois vértices, fornece a mínima distância entre as duas faces.

O cálculo do ângulo entre as normais dos pares de faces, também será extraído por meio do arco cosseno obtido do produto escalar dos dois vetores normais unitários.

Desse modo, o conjunto Fo_2, será composto por três arquivos: Fo_dmin.HNDObj, Fo_dmax.HNDObj e Fo_alfa.HNDObj. Uma listagem exemplo do arquivo Fo_alfa.HNDObj é mostrada a seguir:

```
50721;
0.000000    0    0
0.000000    0    1
3.141593    0    2
3.141593    0    3
1.373401    0    4
1.373401    0    5
1.373401    0    6
1.373401    0    7
1.373401    0    8
1.373401    0    9
1.373401    0    10
1.373401    0    11
...
```

Na primeira linha é extraído, durante a leitura do arquivo, o número de pares de faces armazenado dentro do arquivo e, após essa informação ser obtida, nas linhas seguintes são lidos os valores das variáveis **fvalor**, **KId** e **LId**, respectivamente e separados por tabulações. No exemplo anterior pode-se notar que os pares de faces (0 , 2) e (0 , 3) possuem ângulos entre suas normais iguais sendo esse valor é de **3.141593**.

4.5.3 Criação do Container (2° fase do armazenamento)

Após a numeração das faces e armazenamento das máximas e mínimas distâncias entre cada par de faces e o ângulo dos vetores normais entre essas faces, o conjunto de pares de faces P_2 é então rearranjado de modo a se ter um acesso mais direto à informação contida neste conjunto. Assim sendo, é criado um container H_3 contendo os pares de faces de forma a serem acessados via as distâncias máximas entre faces [$d_{max,oc,ij}$], mínimas distâncias entre faces [$d_{min,oc,ij}$]e seus ângulos normais [α_{ij}].

Apesar do nome container, esse é implementado na forma de três conjuntos derivados dos conjuntos Fo_dmin.HNDObj, Fo_dmax.HNDObj e Fo_alfa.HNDObj, os quais após serem ordenados pelos seus valores (sejam estes dmin, dmax e alfa) e não mais pelos seus índices de faces (apesar de estes serem armazenados no arquivos também), recebem os nomes de H3_dmin.HNDObj, H3_dmax.HNDObj e H3_alfa.HNDObj. Uma listagem exemplo do arquivo H3_dmax.HNDObj é mostrada a seguir:

```
115440;
23.9718     172   173
23.9718     172   172
23.9718     173   173
23.9937     75    75
23.9937     74    74
23.9937     74    75
26.6187     332   333
26.6187     332   332
26.6187     333   333
...
156.0774    90    215
156.0774    90    214
156.0896    17    157
156.0896    16    157
...
```

Verifica-se que sua leitura é semelhante a do arquivo Fo_dmax.HNDObj, porém o arquivo é agora ordenado por ordem crescente de seus valores e não mais pelos índices. Na figura 4.25 pode-se visualizar como este container é constituído.

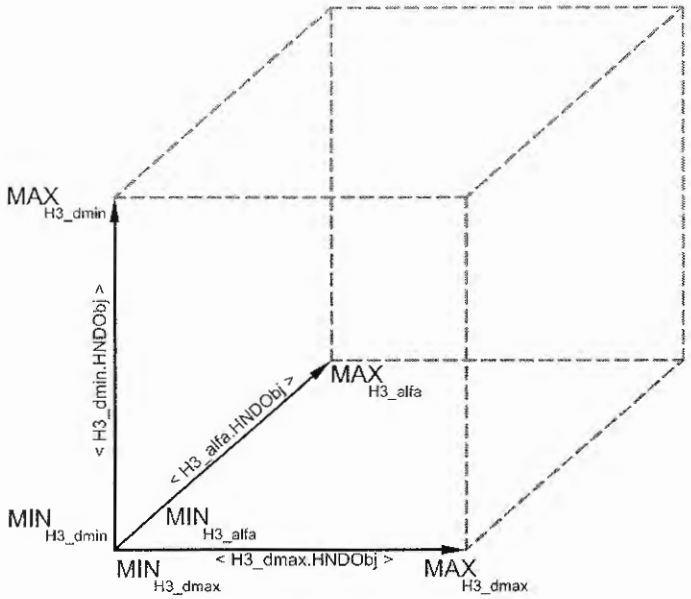

Figura 4. 25 – Container H3.

Assim sendo, o container montado é constituído de inúmeras células de armazenamento que serão acessadas através de 3 coordenadas: fvalor(H3_dmin), fvalor(H3_dmax) e fvalor(H3_alfa), acessadas conforme mostrado na figura 4.26.

No próximo capítulo será apresentado o procedimento de acesso a essas faces.

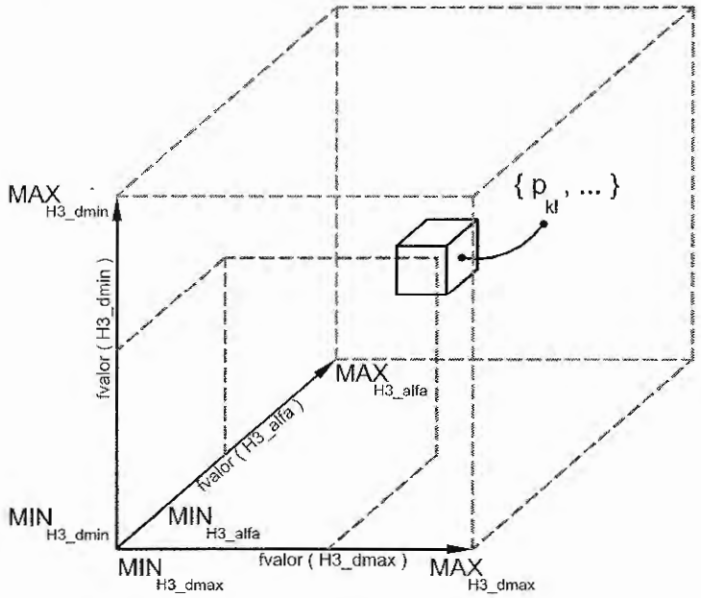

Figura 4. 26 – Acesso de pares de faces admissíveis.

4.6 Cálculo de Contato

Neste capítulo será apresentado como é feito o cálculo das hipóteses de contato, para se estimar a posição e orientação do objeto que está sendo empunhado pela mão.

4.6.1 Heurísticas

Como já citado anteriormente, é muito importante uma escolha correta do método de busca de soluções para o problema de identificação de objetos via toque, pois o uso de um método mal selecionado pode levar a um caminho computacionalmente inviável mesmo para os computadores atuais. Hoje, mesmo com os computadores avançados (comparados aos de décadas atrás), deve-se ter o compromisso de sempre buscar uma boa solução para os problemas complexos, evitando-se assim grandes desperdícios, como no caso de programar os computadores para que sempre encontrem soluções otimizadas, mediante algoritmos de buscas extensivas e demoradas, em outras palavras usando-se a força bruta antes de se pensar em qualquer solução elegante. No caso do trabalho em questão o problema é ainda mais complexo se for considerada a comparação feita entre o objeto simplificado e armazenado na memória (fase *off-line*) e o objeto refinado, apresentado à mão durante a fase de empunhadura e reconhecimento (fase *on-line*).

Quando um objeto é empunhado por uma mão robótica (como a mão simulada), esta tem como retorno (através de sensores táteis simulados) os pontos de contatos de cada ponta dos dedos (Xcf1, Xcf2, Xcf3 e Xcf4) e os respectivos vetores normais que contêm estes pontos de contatos nas superfícies das pontas dos dedos (Acf1, Acf2, Acf3 e Acf4).

Assim, com base nestas informações, pode-se calcular as distâncias relativas entre os pontos de contatos das pontas dos dedos, tendo em vista que há 6 possibilidades de distâncias para cada par de dedo (figura 4.27), sendo : d^M_{c12}, d^M_{c13}, d^M_{c14}, d^M_{c23}, d^M_{c24} e d^M_{c34}, o índice M representa a **M**ão e o índice c representa **C**ontato. Já o par de algarismos indica a quais dedos pertence a distância calculada. Lembrando que estes cálculos serão feitos durante a fase *on-line*, ou seja, durante o tempo de empunhadura do objeto. Os ângulos normais podem ser extraídos facilmente da própria matriz de transformação que orienta a ponta de cada dedo da mão, seja esta simulada ou não, uma vez que em uma mão real os sensores táteis e *encoders* forneceriam as posições finais de cada dedo em tempo real.

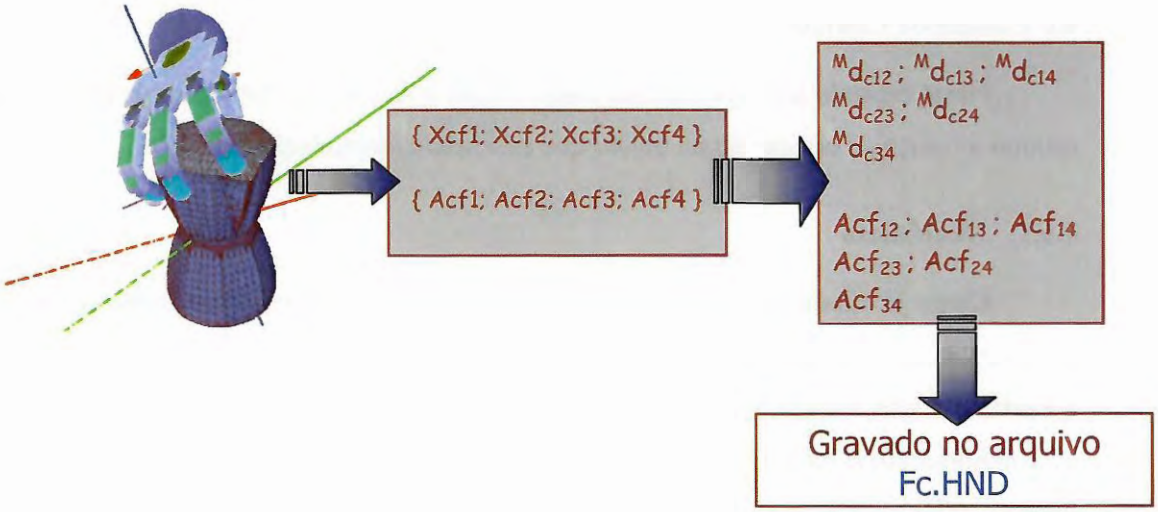

Figura 4. 27 – Retorno de informações da Mão na fase on-line.

Uma vez calculadas as distâncias entre os pares de dedos, estas podem ser comparadas com as distâncias entre os pares de faces que compõe o objeto, pois quaisquer que sejam as distâncias, (entre os pontos de contato ou entre os pares de faces do objeto) são estas distâncias relativas e escalares medidas que independem do sistema de coordenadas aos quais estão sendo referenciados (sistema de coordenadas do Universo, ou sistema de coordenadas da Mão, ou sistema de coordenadas do Objeto).

Então, após o cálculo das distâncias de contato e seus respectivos ângulos para cada par de dedo da mão, esses dados são inseridos (juntamente com a incerteza ε) na função de busca, que usará estes valores como entradas em uma função, e retirará do container H3 as faces admissíveis de contato.

Sejam então H3_dmin[i], o i-ésimo valor contido no arquivo H3_dmin.HNDOBj, e d_{cij}^{M} a distância a qual está sendo verificada dentro do conjunto H3_dmin.HNDOBj(figura 4.28), serão obtidos deste arquivo todos os pares de faces que satisfaçam a seguinte regra:

$$\text{H3\_dmin}[i] < [\, d_{cij}^{M} + \varepsilon \,] \qquad (48)$$

onde i variará de 0 até determinado valor que satisfaça a equação (48).

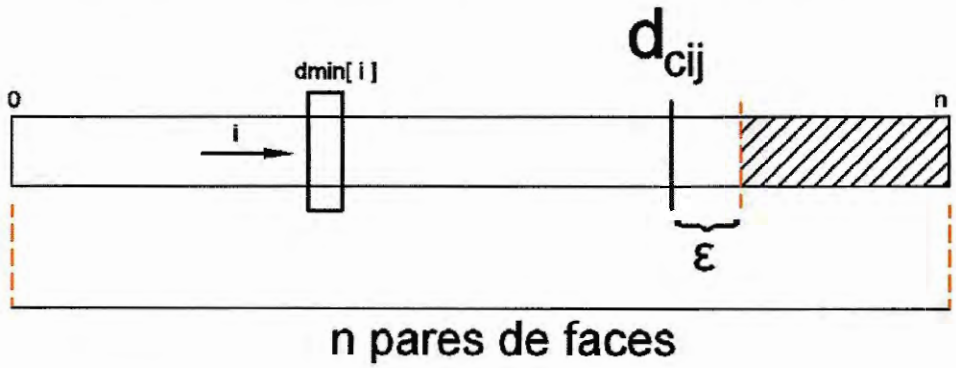

Figura 4. 28 – Sistema de busca de pares de faces em H3_dmin.HNDOBj.

Para H3_dmax[i], o i-ésimo valor contido no arquivo H3_dmax.HNDOBj, e (novamente) d_{cij}^{M} a distância a qual esta sendo verificada dentro do conjunto H3_dmax.HNDOBj (figura 4.29), serão obtidos deste arquivo todos os pares de faces que satisfaçam a seguinte regra:

$$\text{H3\_dmax}[i] > [\, d_{cij}^{M} + \varepsilon \,] \qquad (49)$$

onde i variará de n (decrescendo) até determinado valor que satisfaça a equação (49).

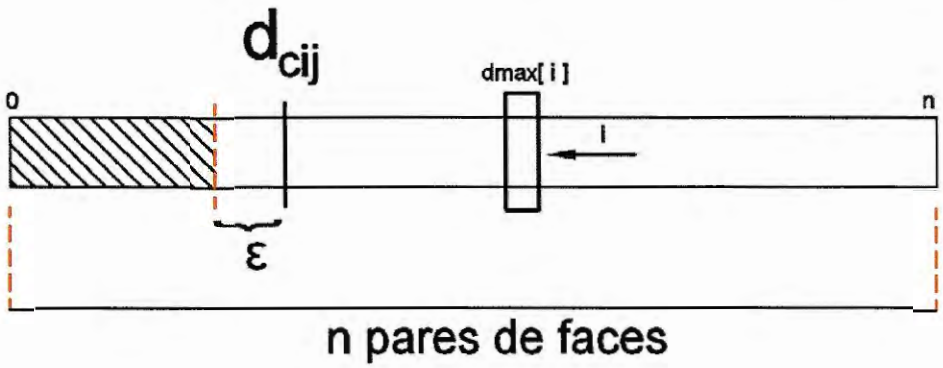

Figura 4. 29 – Sistema de busca de pares de faces em H3_dmax.HNDOBj.

E finalmente para H3_alfa[i], o i-ésimo valor contido no arquivo H3_alfa.HNDOBj, e (novamente) Acf_{ij} o ângulo entre as normais dos dedos i e j que esta sendo verificado dentro do conjunto H3_alfa.HNDOBj (figura 4.30), serão obtidos deste arquivo os pares de faces que satisfaçam a seguinte regra:

$$[Acf_{ij} - \varepsilon\,] < \text{H3\_alfa}[i] < [Acf_{ij} + \varepsilon\,] \qquad (50)$$

onde i variará de 0 (crescendo) até determinado valor que satisfaça a equação (50).

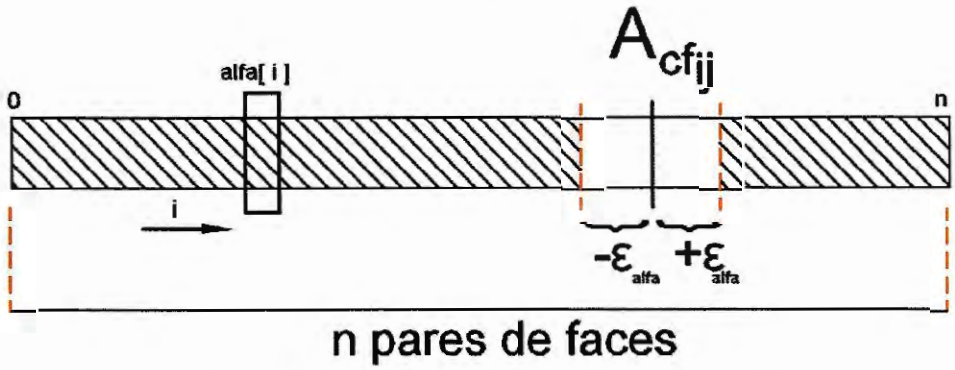

Figura 4. 30 – Sistema de busca de pares de faces em H3_alfa.HNDOBj.

Para cada procedimento de busca descrito acima, os pares de faces que satisfizeram as equações 48, 49 e 50, deverão ser armazenados respectivamente nos seguintes conjuntos:

Dmin_temp → [K_dmin_temp , L_dmin_temp]

Dmax_temp → [K_dmax_temp , L_dmax_temp]

Alfa_temp → [K_alfa_temp , L_alfa_temp]

Os pares de faces serão armazenados temporariamente nestes vetores até a etapa seguinte, que será descrita na próxima seção.

4.6.2 Determinação de Hipóteses de Contatos via da Análise de Conjuntos de Contatos

Após a etapa de busca de faces admissíveis e estocagem destas faces nos conjuntos como descrito acima, novamente são ordenados estes conjuntos, porém agora, usando como referência o índice k dos pares de faces, visando facilitar os algoritmos de buscas que serão aplicados em seguida.

Os três conjuntos acima citados, depois de ordenados (conforme mostra a figura 4.31), serão chamados de A (para Dmax_temp), B (para Dmin_temp) e C (para Alfa_temp).

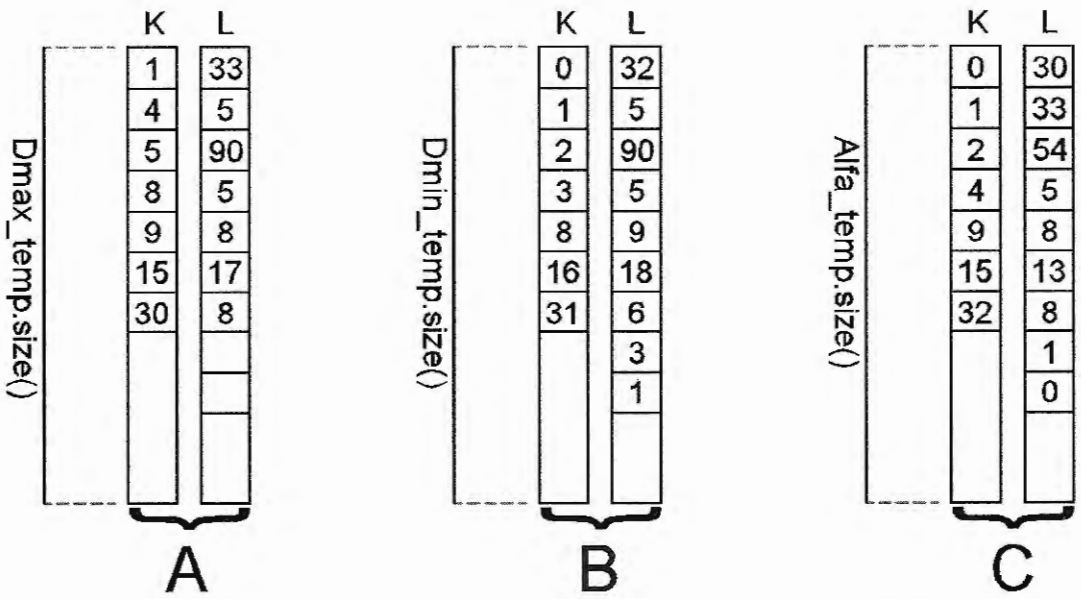

Figura 4. 31 – Sistema de busca de pares de faces em H3_alfa.HNDOBj.

A partir de então é rodado um algoritmo de busca de faces entre os conjuntos A e B, buscando relacionar se existem pares de faces comuns entre A e B, ou seja, tenta buscar quais elementos (pares de faces) são comuns e, então, realiza uma operação A $\bigcap$ B, o resultado é

armazenado no conjunto Rc(ij), figura 4.32, que será também ordenado fixando o índice *k* como referência.

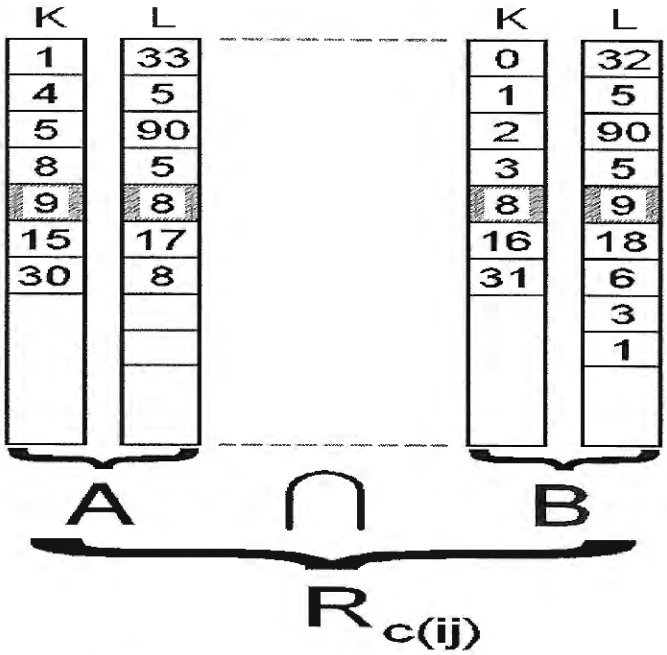

Figura 4. 32 – Operação de Intersecção entre os conjuntos A e B.

Com o resultado Rc(ij) testa-se a intersecção deste com o conjunto C e o resultado, se houver, será guardado no conjunto $R_2c(ij)$, figura 4.33, conforme a operação booleana abaixo:

$$R_2c(ij) = Rc(ij) \cap C \qquad (51)$$

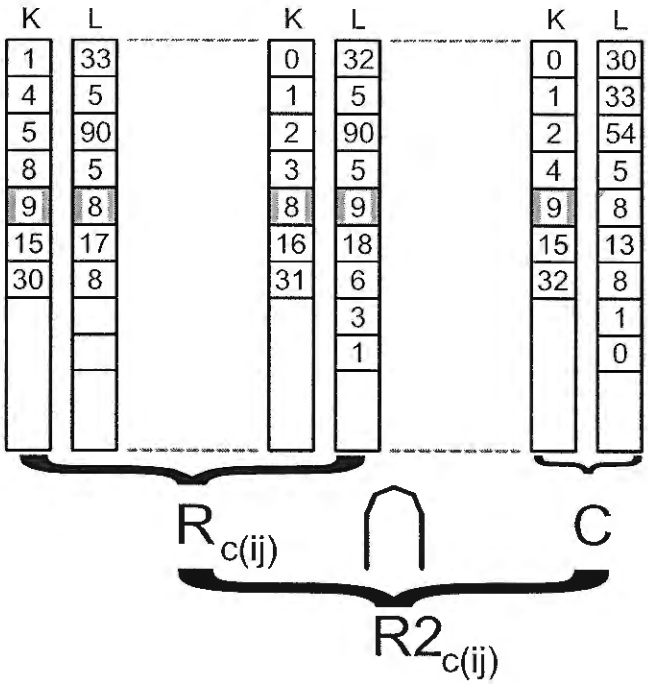

Figura 4. 33 – Operação de Intersecção entre os conjuntos Rc(ij) e C, com possível resultado.

Lembrando que R2c(ij) são os resultados admissíveis para o par de dedos *i* e *j*, para uma mão de 4 dedos tem-se os seguintes conjuntos de resultados:

$R2c_{(12)} = \{(K_1,l_1), (K_1,l_1), (K_1,l_1), ..., (K_n,l_n)\}$

$R2c_{(13)} = \{(K_1,l_1), (K_1,l_1), (K_1,l_1), ..., (K_n,l_n)\}$

$R2c_{(14)} = \{(K_1,l_1), (K_1,l_1), (K_1,l_1), ..., (K_n,l_n)\}$

$R2c_{(23)} = \{(K_1,l_1), (K_1,l_1), (K_1,l_1), ..., (K_n,l_n)\}$

$R2c_{(24)} = \{(K_1,l_1), (K_1,l_1), (K_1,l_1), ..., (K_n,l_n)\}$

$R2c_{(34)} = \{(K_1,l_1), (K_1,l_1), (K_1,l_1), ..., (K_n,l_n)\}$

Tais conjuntos de resultados admissíveis para cada par de dedos serão agrupados em um conjunto de soluções S, sendo $S = \{R2c_{(12)}, R2c_{(13)}, R2c_{(14)}, R2c_{(23)}, R2c_{(24)}, R2c_{(34)}\}$, para as hipóteses de contatos encontradas podendo fornecer os seguintes resultados a serem analisados:

- $S = \{\}$ ou Ø, ou seja não houve reconhecimento do objeto – Isto pode ocorrer caso o objeto não tenha sido simplificado e armazenado de forma adequada, eliminando assim contornos importantes para o reconhecimento do objeto;

- $S = \{R2c_{(12)}, R2c_{(13)}, R2c_{(14)}, R2c_{(23)}, R2c_{(24)}, R2c_{(34)}\}$, sendo que o número de elementos dos conjuntos $R2c_{(12)}$, $R2c_{(13)}$, $R2c_{(14)}$, $R2c_{(23)}$, $R2c_{(24)}$ e $R2c_{(34)}$ seja apenas um, de forma que teremos o reconhecimento do objeto com apenas um par de face para cada dedo – isso acontece em casos em que foram feitas boas simplificações e armazenagem de objetos;

- $S = \{R2c_{(12)}, R2c_{(13)}, R2c_{(14)}, R2c_{(23)}, R2c_{(24)}, R2c_{(34)}\}$, sendo que o número de elementos dos conjuntos $R2c_{(12)}$, $R2c_{(13)}$, $R2c_{(14)}$, $R2c_{(23)}$, $R2c_{(24)}$ e $R2c_{(34)}$ maior que um elemento, ou seja, reconhecimento do objeto com vários pares de faces para cada dedo – a melhor solução neste caso será descrita na próxima seção.

4.6.3 Melhorando os resultados encontrados

Caso, na realização das operações de intersecções um ou vários resultados sejam obtidos, este(s) deve(m) ser testado(s) para verificar se o(s) resultado(s) encontrado(s) obedece(m) a condição real em que a mão empunha o objeto. No caso de um resultado apenas para cada dedo, deve-se verificar se os pares de faces (para cada par dedo) encontrados realmente indicam que o objeto simplificado da memória realmente foi reconhecido, assim,

esse é orientado e colocado em contato com a mão segundo a orientação determinada pelos pares de faces encontrados e, juntamente com a mão, posicionado segundo os ângulos de articulação que foram detectados durante a fase de contato. Caso a mão não consiga agarrar o objeto usando os ângulos de articulações (extraídos durante o *grasp*) para os pares de faces dados, o resultado é então descartado, caso contrário, é aceito e exibido na tela.

No caso de vários pares de faces para cada dedo, é realizado o mesmo procedimento descrito acima, porém, é esperado que apenas um par de faces para cada dedo seja aceito no final desta fase de refino de resultados. Tal procedimento só é concretizado após várias comparações entre os parâmetros de entradas (ângulos das articulações) e os resultados encontrados (faces admissíveis), neste caso, o resultado com os menores erros será aceito.

5 - Resultados

Neste capítulo serão apresentados alguns resultados extraídos neste trabalho.

Objeto 1 – **Cafeteira**.

Objeto Refinado: (figura 5.1a)

6690 Faces

3347 Vértices

6690 Vetores normais

Objeto Simplificado: (figura 5.1b)

2607 Faces

1346 Vértices

2607 Vetores normais

- Tempo de ordenação de cada conjunto Fo para gerar H_3: **17,33 segundos em média**.
- Memória gasta durante armazenagem de Fo + H_3: **2 x 187.432 = 374.864 Kbytes**.
- Memória gasta no final da armazenagem de H_3: **187.432 Kbytes**.

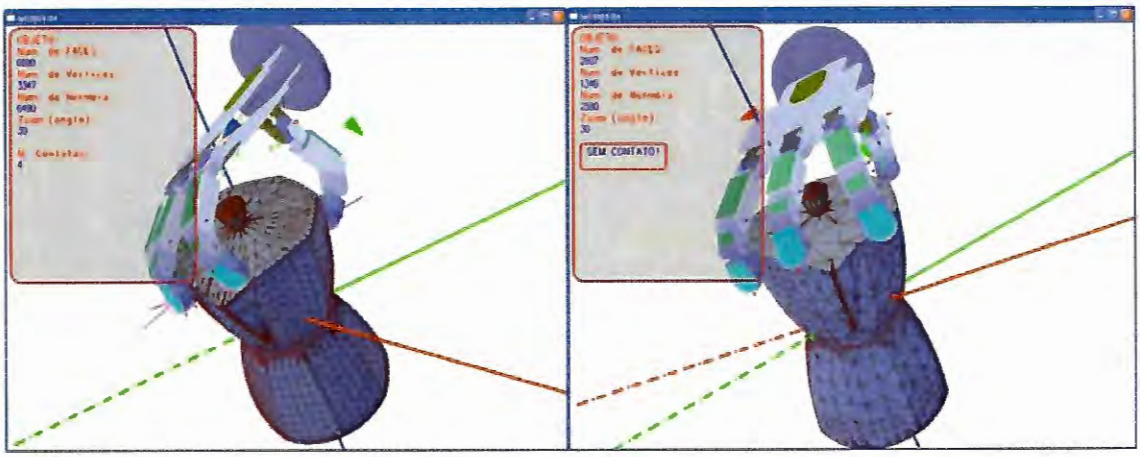

Figura 5.1 a – ObjREC.exe - Cafeteira Refinada.

Figura 5.1 b – ObjREC.exe - Cafeteira Simplificada.

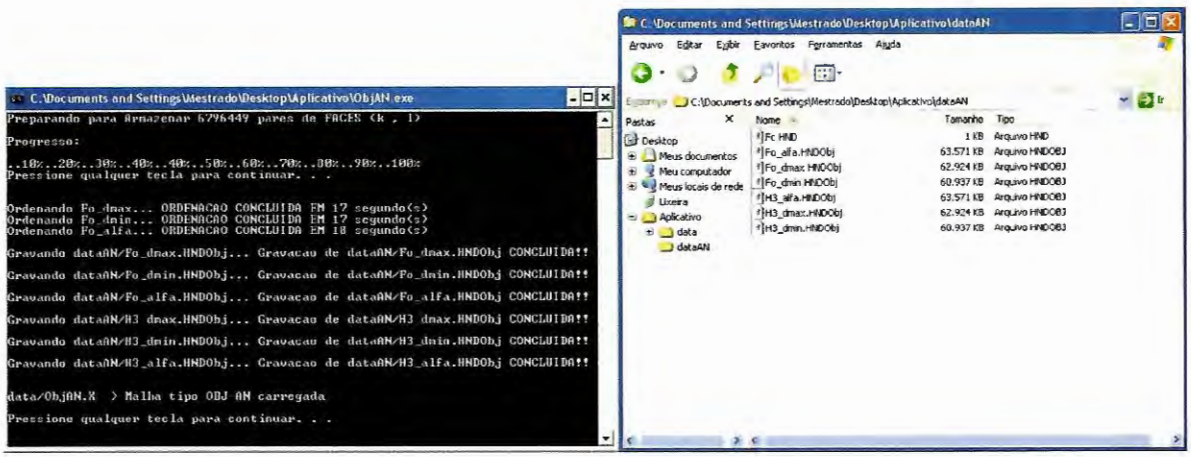

Figura 5.2 a –ObjAN.exe – Cafeteira Simplificada.

Figura 5.2 b – Arquivos gerados - Cafeteira Simplificada.

Objeto 2 – **Mouse**.

Objeto Refinado: (figura 5.3a)

4118 Faces

2061 Vértices

4118 Vetores normais(*)

Objeto Simplificado: (figura 5.3b)

1724 Faces

864 Vértices

1724 Vetores normais(*)

(*) A exibição dos vetores normais na tela diferente do número de faces refere-se às normais do arquivo X-File, que não entram no cálculo de contato.

- Tempo de ordenação de cada conjunto Fo para gerar H_3: **6 segundos em média**.
- Memória gasta durante armazenagem de Fo + H_3: **2 x 80.035 = 160.070 Kbytes**.
- Memória gasta no final da armazenagem de H_3: **80.035 Kbytes**.

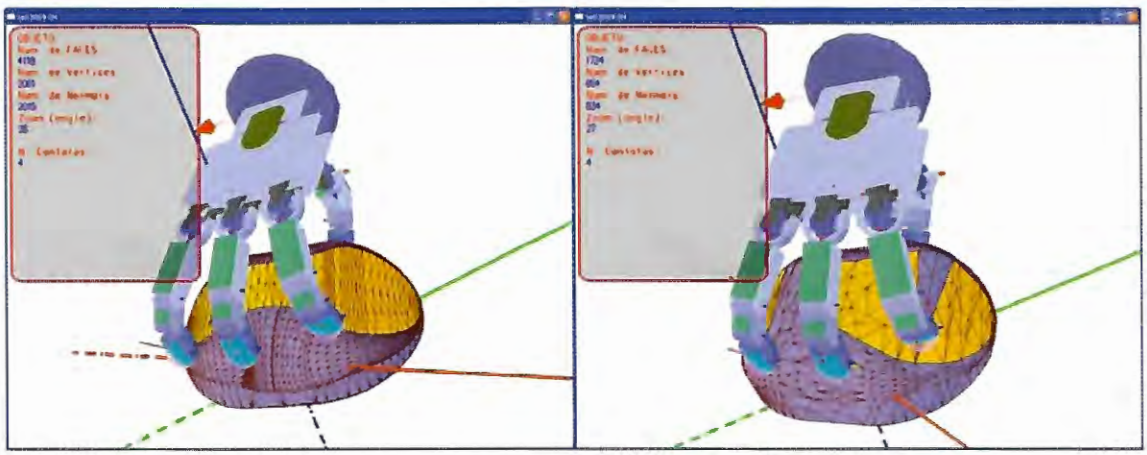

Figura 5.3 a – ObjREC.exe - Mouse Refinado.

Figura 5.3 b – ObjREC.exe - Mouse Simplificado.

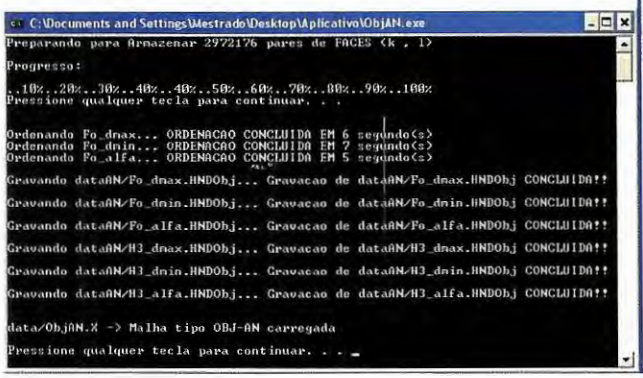

Figura 5.4 a –ObjAN.exe – Mouse Simplificado.

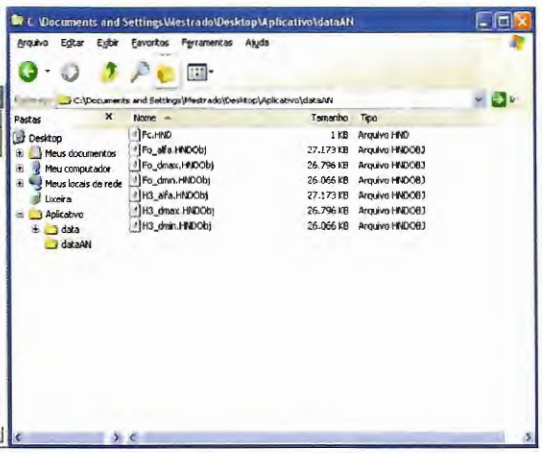

Figura 5.4 b – Arquivos gerados - Mouse Simplificado.

Objeto 3 – **Chaleira.**

Objeto Refinado: (figura 5.5a)

5112 Faces

2886 Vértices

5112 Vetores normais(*)

Objeto Simplificado: (figura 5.5b)

1560 Faces

944 Vértices

1560 Vetores normais(*)

(*) A exibição dos vetores normais na tela diferente do número de faces refere-se às normais do arquivo X-File, que não entram no cálculo de contato.

- Tempo de ordenação de cada conjunto Fo para gerar H_3: **4,66 segundos em média.**
- Memória gasta durante armazenagem de Fo + H_3: **2 x 63.129 = 126.258 Kbytes.**
- Memória gasta no final da armazenagem de H_3: **63.129 Kbytes.**

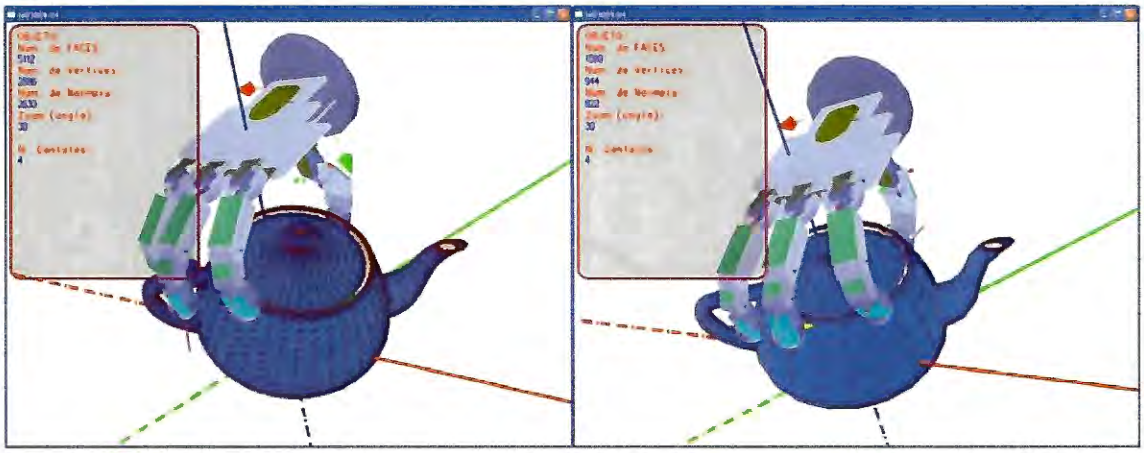

Figura 5.5 a – ObjREC.exe - Chaleira Refinada.

Figura 5.5 b – ObjREC.exe - Chaleira Simplificada.

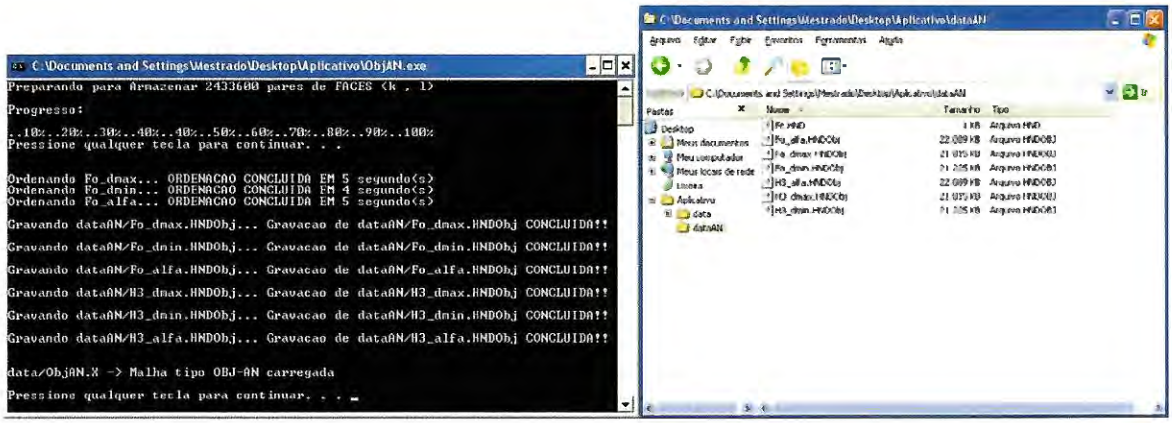

Figura 5.6 a –ObjAN.exe – Chaleira Simplificada.

Figura 5.6 b – Arquivos gerados - Chaleira Simplificada.

Objeto 4 – **Chave**.

Objeto Refinado: (figura 5.7a)

4572 Faces

2443 Vértices

4572 Vetores normais(*)

Objeto Simplificado: (figura 5.7b)

1668 Faces

991 Vértices

1668 Vetores normais(*)

(*) A exibição dos vetores normais na tela diferente do número de faces refere-se às normais do arquivo X-File, que não entram no cálculo de contato.

- Tempo de ordenação de cada conjunto Fo para gerar H_3: **4,33 segundos em média.**
- Memória gasta durante armazenagem de Fo + H_3: **2 x 74.676 = 149.352 Kbytes.**
- Memória gasta no final da armazenagem de H_3: **74.676 Kbytes.**

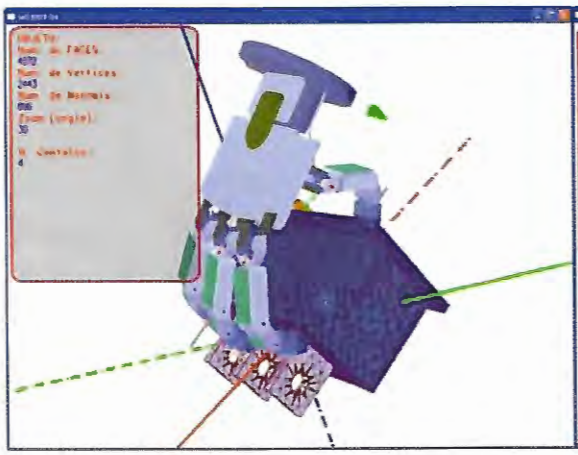

Figura 5.7 a – ObjREC.exe - Chave Refinada.

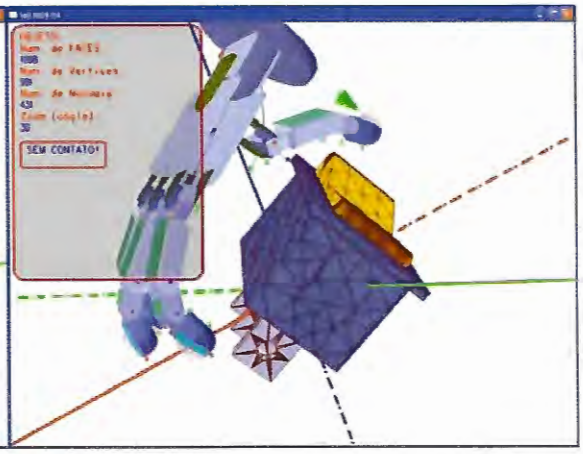

Figura 5.7 b – ObjREC.exe - Chave Simplificada.

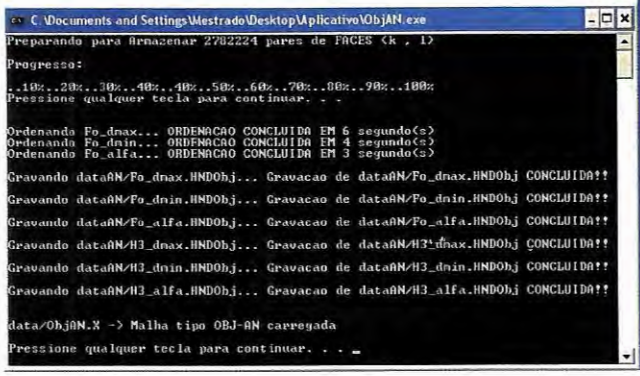

Figura 5.8 a – ObjAN.exe – Chave Simplificada.

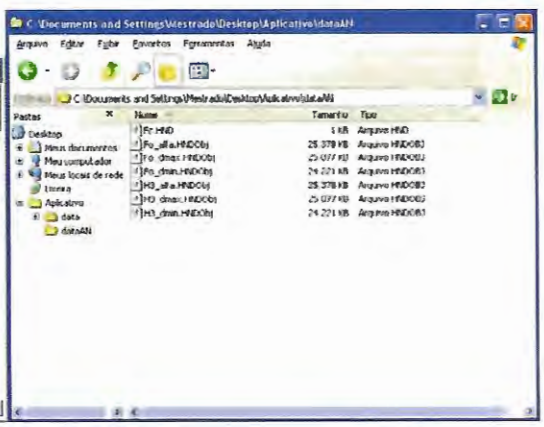

Figura 5.8 b – Arquivos gerados - Chave Simplificada.

Objeto 4 – **Tronco**.

Objeto Refinado: (figura 5.9a)

486 Faces

245 Vértices

486 Vetores normais(*)

Objeto Simplificado: (figura 5.9b)

318 Faces

161 Vértices

318 Vetores normais(*)

(*) A exibição dos vetores normais na tela diferente do número de faces refere-se às normais do arquivo X-File, que não entram no cálculo de contato.

- Tempo de ordenação de cada conjunto Fo para gerar H_3: **0,33 segundos em média**.
- Memória gasta durante armazenagem de Fo + H_3: **2 x 5.958 = 11.916 Kbytes**.
- Memória gasta no final da armazenagem de H_3: **5.958 Kbytes**.

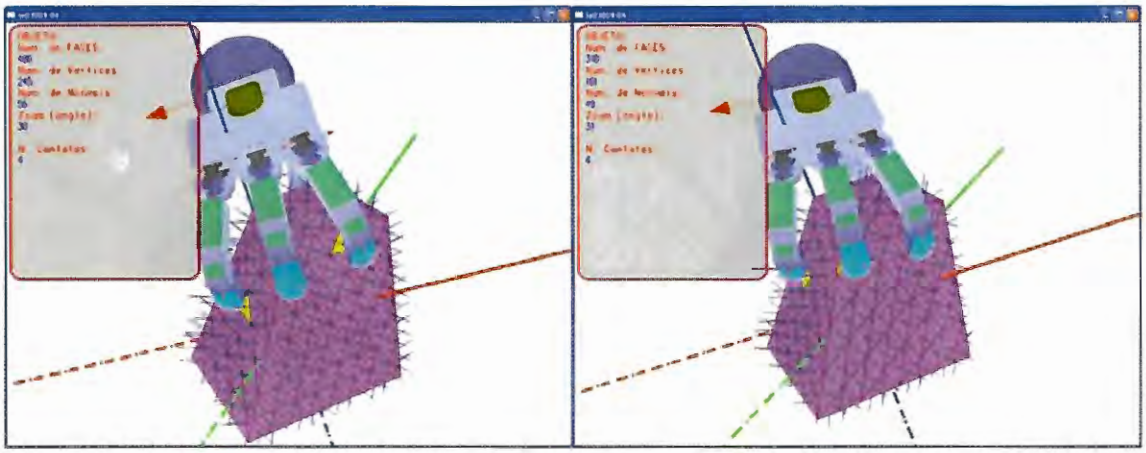

Figura 5.9 a – ObjREC.exe - Tronco Refinado.

Figura 5.9 b – ObjREC.exe - Tronco Simplificada.

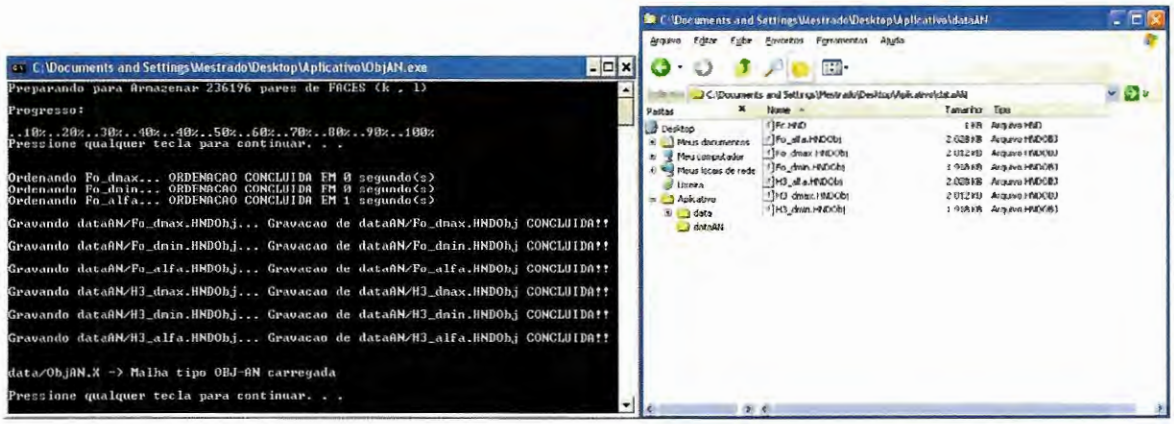

Figura 5.10 a – ObjAN.exe – Tronco Simplificado.

Figura 5.10 b – Arquivos gerados - Tronco Simplificado.

6 - Conclusões

De forma resumida, este trabalho propõe a elaboração de um ambiente de simulação computacional para a análise experimental (não teórica) dos fenômenos mecânicos que ocorrem quando uma mão robótica antropomórfica agarra (ou tenta agarrar) e manipula um objeto. Há aspectos relevantes a considerar neste trabalho como:

(1) Detecção e análise de contato;

(2) O reconhecimento da posição e da orientação de um objeto através do tato;

(3) Modelagem de Cadeia Cinemática da Mão Robótica;

(4) Computação gráfica e geométrica,

A implementação das simulações numéricas em computador foi feita através de programação C/C++ com recursos OpenGL. Esse enfoque cria uma base de códigos e funções que servem como uma plataforma segura e ponto de partida para a investigações de novas abordagens, implementação de melhorias ou mesmo de novas metodologias. Uma necessidade que já se vislumbra consiste em embarcar parte do código desenvolvido no controle de mãos robóticas antropomórficas. O conhecimento adquirido sobre o meio em que o sistema se insere permite comportamentos mais sofisticados, que se mostrariam 'mais inteligentes' para o usuário.

As tarefas de elaboração do ambiente de simulação foram divididas em duas fases: Uma fase *off-line*, ou de aprendizagem[2], que onde as informações do Objeto são agrupadas e armazenadas de forma conveniente em espaço de memória; e a fase *on-line* (em tempo real), onde o usuário interage com a mão no ambiente virtual e simula a mão agarrando, empunhando ou manipulando o objeto, exibindo depois o objeto simplificado (guardado na memória) no monitor, mostrando se o robô foi capaz de reconhecer sua posição no espaço ou não.

Para ambas as fases, os códigos dos programas foram implementados de forma aberta e modular, o que dará condições de se usar suas funções em projetos futuros feitos em C/C++ e para aplicações em OpenGL.

A implementação da fase *off-line*, foi concluída e os testes apresentados no capítulo 5 demonstram que o desempenho alcançado atende com sucesso as necessidades atuais de pesquisa. Para o futuro, serão estudadas formas adicionais de compactação de dados para que se reduza o tamanho dos arquivos gerados.

---

[2] não se refere aqui necessariamente ao uso de Redes Neurais Artificiais, no entanto também não se exclui a possibilidade futura de uso de ferramentas relacionadas com inteligência computacional.

Já para fase *on-line*, parcialmente alcançada, tem-se as seguintes realizações e metas de desenvolvimento futuro:

I – Realizações

(1) A criação de um ambiente 3D, base para estudos e visualizações de resultados.

(2) Programação modular de funções matemáticas e de interação do usuário em C/C++ utilizando recursos em OpenGL.

(3) Programação em C/C++ de funções que modelam a cadeia cinemática da mão robótica com fundamentos aprendidos no curso de Introdução à Robótica.

(4) Programação da função de detecção de colisão dos dedos da mão robótica com o objeto, cujo retorno de dados fornece ao programa principal o ponto de contato, o vetor normal ao contato, o ângulo das articulações dos dedos e o índice de face tocada no objeto (para representar o toque deste com o dedo).

(5) Criação do procedimento para busca de hipóteses de contatos (capítulo 4).

II – Metas para o desenvolvimento futuro do projeto

(6) Programação das funções de hipóteses de contatos para simulação no ambiente virtual criado.

(7) Programação para versões futuras de um aplicativo para a plataforma GNU/Linux (citada no capítulo 3), baseada em GTK++ / OpenGL, que dará ao programa uma interface mais acessível e amigável, além do fato de ser também multi-plataforma.

(8) Melhorias no próprio código também deverão ser feitas para que este tenha uma maior performance e, após versões mais estáveis e robustas, poderá ser criada uma versão embarcada deste software para sistemas micro-processados ou micro-controlados, para novos projetos de mãos robóticas que futuramente poderão ser criados pelo Laboratório futuramente.

Pode-se então concluir, com base no exposto acima, que o projeto foi bem sucedido e dará condições de num futuro próximo, ser melhorado, gerando novas abordagens para a questão de manipulação de objetos por parte dos robôs de serviços, fazendo com que estes robôs tenham condições de executar trabalhos de forma mais hábil, e assim auxiliar o homem em ambientes mais complexos.

Referências Bibliográficas

[1] ACCLAIMIMAGES.
Disponível em <http://www.acclaimimages.com/search_terms/cleaner.html>, acesso em outubro de 2005.

[2] BUENO, ANDRE DUARTE: Programação Orientada a Objeto Com C++, 2003. Editora Novatec; São Paulo.

[3] C. S. Lovchik and M. A. Diftler. The Robonaut Hand: A dextrous robot hand for space. In *Proc. IEEE Conf. on Robotics and Automation*, pages 907 – 912, Detroit, Michigan, USA, May 1999.

[4] CNN.COM.
Disponivel em <http://edition.cnn.com/2004/TECH/02/02/social.robots.ap/>, acesso em maio de 2007.

[5] COHEN, MARCELO; MANSSOUR, Isabel Harb. OpenGL - Uma Abordagem Prática e Objetiva. Editora Novatec, 2006.

[6] CPLUSPLUS REFERENCE.
Disponível em < http://www.cplusplus.com/reference/>, Acesso em agosto de 2005.

[7] DLR - Institute of Robotics and Mechatronics, Disponível em <http://www.dlr.de/rm-neu/en/desktopdefault.aspx/tabid-3975/>, acesso em maio de 2005.

[8] GIORIA, GUSTAVO DO SANTOS. Protótipo Experimental de mão artificial de 5 dedos. Iniciação Científica. *Escola de Engenharia de São Paulo,* 2003.

[9] GNU – General Public License (2005).
Diponínel em <http://www.gnu.org/>, Acesso em maio de 2005.

[10] H. LIU, P. Meusel, J. Butterfaß, and G. Hirzinger. DLR's Multisensory Articulated Hand. part II: The parallel torque/position control system. In *Proc. IEEE Conf. on Robotics and Automation*, pages 2087 – 2093, Leuven, 1998.

[11] H. LIU. The HIT/DLR Dexterous Hand: Work in progress. In *Proc. of the IEEE Intl. Conf. on Robotics and Automation, Taipei,Taiwan*, September 2003.

[12] HAIDACHER, STEFFEN (2003). Contact Point and Object Position from Force/Torque and Position Sensors for Grasps with a Dextrous Robotic Hand. *Institut für Robotik und Mechatronik.*

[13] HAIDACHER, STEFFEN, J. Butterfaß, Max Fischer, Markus Grebenstein, Klaus J¨ohl, Klaus Kunze, Matthias Nickl, Nikolaus Seitz, and Gerd Hirzinger. DLR Hand II: Hard- and software architecture for information processing. In *Proc. of the IEEE Intl. Conf. on Robotics and Automation, Taipei, Taiwan*, September 2003.

[14] HONDA ASIMO.
Disponível em < http://asimo.honda.com/Gallery.aspx>, acesso em maio 2007.

[15] J. BUTTERFAß, G. Hirzinger, S. Knoch, and H. Liu. DLR's Multisensory Articulated Hand. part I: Hard- and software architecture. In *Proc. IEEE Conf. on Robotics and Automation*, pages 2081 – 2086, Leuven, 1998.

[16] J. BUTTERFAß, M. Grebenstein, H. Liu and G. Hirzinger. DLR-Hand II: Next Generation of Dextrous Robot Hand. In *Proc IEEE Conf. On Robotics and Automation*, pages 109 – 114, Seoul, Korea, May 2001.

[17] M. C. CARROZZA, B. Massa, S. Micera, M. Zecca, and P. Dario. A "Wearable" Artificial Hand for Prosthetics and Humanoid Robotics Applications. In Proc. IEEE-RAS International Conference on Humanoid Robots HUMANOIDS 2001, Waseda University International Conference Center, Tokyo, Japan, November 2001.

[18] MARK R. CUTKSOKY. On Grasp Choice, Grasp Models, and the Design of Hands for Manufacturing Tasks. IEEE Transactions on Robotics and Automation, 5(3):269 – 279, June 1989.

[19] OPENGL - The Industry Standard for High Performance Graphics.
Diponínel em <http://www.opengl.org>, Acesso em maio de 2005.

[20] PAL TECHNOLOGY ROBOTICS.
Diponínel em <http://www.pal-robotics.com/newweb/>, Acesso em março de 2008.

[21] PETER J. KYBERD, Colin Light, Paul H. Chappell, Jim M. Nightingale, Dave Whatley, and Mervyn Evans. The design of anthropomorphic prosthetic hands: A study of the Southampton Hand. Robotica, 19:593 – 600, 2001.

[22] R M CROWDER (January 1998). Tactile sensing.
Disponível em: <http://www.soton.ac.uk/~rmc1/robotics/artactile.htm>, Acesso em setembro de 2007.

[23] RIBEIRO, ADRAINO J. MARQUES. Desenvolvimento de um Sistema Mecatrônico Interativo para Auxiliar no Tratamento de Reabilitação de Crianças com Deficiência Motora nas Pernas. Dissertação (Mestrado em Engenharia Mecânica). *Escola de Engenharia de São Paulo,* Outubro 2004.

[24] S.C. JACOBSEN, J. E. Wood, D. F. Knutti, and K. B. Biggers. The UTAH/M.I.T. Dextrous Hand: Work in Progress. *Int. Journal of Robotics Research*, 3(4):21 – 50, Winter 1984.

[25] SCHILDT, HERBERT. C Completo e Total. – 3° edição. Editora:. Makron Books. Publicação 1998.

[26] SDL - Simple DirectMedia Layer (2007).
Diponínel em <http://www.libsdl.org/>, Acesso em maio de 2007.

[27] SIEGWART, R., Nourkakhsh I. Introduction to Autonomous Mobile Robots, MIT Press 2004.

[28] SUGANO LABORATORY, WASEDA University.
Disponível em <http://twendyone.com/index_e.html>, Acesso em março de 2008.

[29] WEISS-ROBOTICS.
Disponível em <http://www.weiss-robotics.de/>, Acesso em novembro de 2005.

[30] WIKIPEDIA.
Disponiveil em <http://en.wikipedia.org/wiki/Stereopsis>, Acessado em abril de 2008.

[31] WIKIPEDIA.
Disponiveil em <http://pt.wikipedia.org/wiki/Ecolocaliza%C3%A7%C3%A3o>, Acessado em abril de 2008

ANEXO I

Explicação sobre os aplicativos gerados neste trabalho.

Devido a questão ter sido dividida em duas partes, Fase *on-line* e Fase *off-line*, foram gerados dois programas responsáveis por cada etapa, sendo eles chamados ObjREC.exe e ObjAN.exe, respectivamente.

ObjAN.exe – Este programa é responsável pela leitura e armazenagem do arquivo ObjAN.X guardado dentro da pasta /data, contida no diretório deste programa. Este arquivo (ObjAN.X) guarda as informações do objeto discretizado, e este será lido pelo programa com a finalidade de ter seus pares de faces contabilizados, calculadas suas distâncias dmax, dmin e seus ângulos alfa, e guardados juntamente com seus índices (Fo_dmax.HNDObj , Fo_dmin.HNDObj e Fo_alfa.HNDObj).

Em seguida o programa ordena os conjuntos acima e os grava nos arquivos H3_dmax.HNDObj , H3_dmin.HNDObj e H3_alfa.HNDObj, deixando-os prontos para serem analisados pelo programa ObjREC.exe.

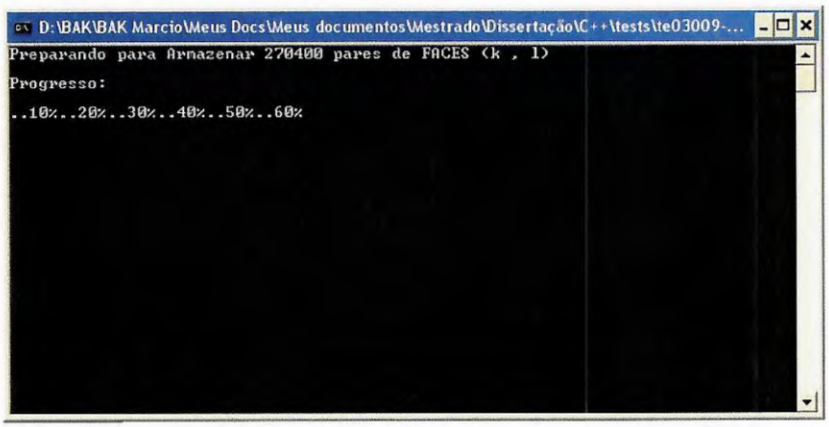

Figura A-I. 1 – Interface do programa ObjAN.exe.

ObjREC.exe – Este programa é responsável pela exibição, interação, simulação, extração de dados de contatos e exibição de resultados de contatos. Ele é implementado em OpenGL e necessita das DLLs opengl32.dll, glut32.dll e glaux.dll para ser executado. O local onde estas DLLs devem ser encontradas é a pasta C:\windows\System32, onde C: é o Hard Drive onde o Windows estiver instalado.

- Exibição – O programa gerará sua própria janela onde será exibido o cenário com a mão e o objeto a ser simulado.

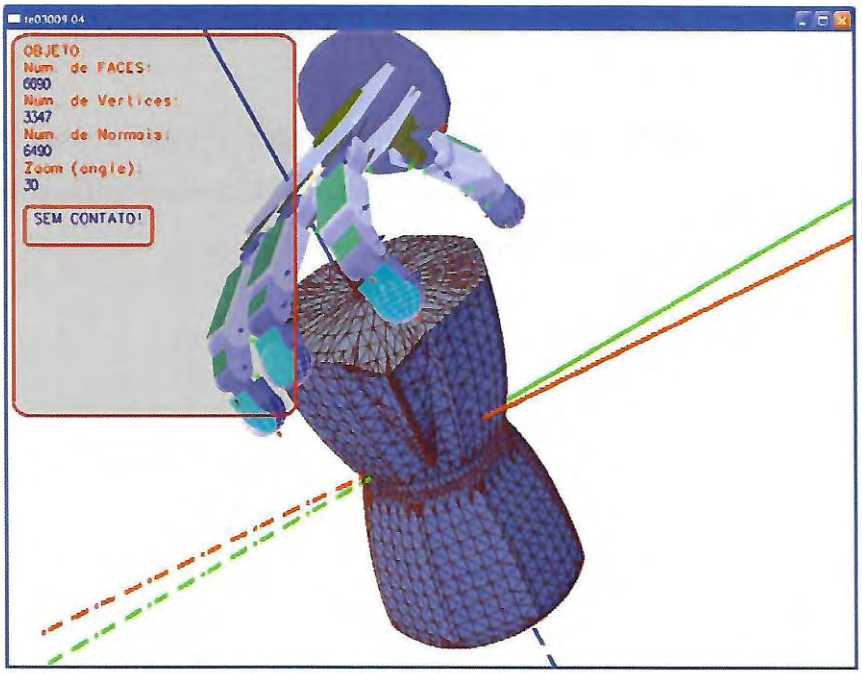

Figura A-I. 2 – Interface do programa ObjREC.exe.

- Interação – O usuário poderá interagir com o programa usando o mouse para movimentar a mão e o obejto (3D Orbit) e as teclas de funções descritas abaixo;

F1 – Alterna Rotate dos Eixos XY para YZ;

F2 – Mostra / Oculta Eixo de Coordenadas (Coords ON/OFF);

F3 – Alterna os tipo de exibição do objeto,

Shade → Wireframe →Shade + Baricentros → Shade + Wire + Baricentros → BoudingBox

F4 – Habilita / Desabilita Faces Ocultas (CULL FACE);

F5 – Habilita / Desabilita movimento da Mão;

F6 – Salva arquivo Fc.HND, com os dados de contatos da mão;

F7 – Mostra / Oculta Quadro de TEXTO;

F8 – Alterna os tipo de exibição da Mão,

Shade → BoudingBox → Wireframe (Cadeia Cinemática da Mão).

F9 – Fecha a Mão sobre o objeto (agarra o objeto);

F10 – RESET → Restaura as variáveis conforme o inicio da execução do programa;

F11 – Verificação das hipóteses de contatos.

- Extração de Dados – O usuário poderá gravar um teste de contato apertando-se a tecla F6. Assim será gravado um arquivo Fc.HND que será analisado, caso o usuário queira, pelo programa para testar as hipóteses de contatos.

- Exibição dos resultados encontrados – Com a tecla F11 serão rodados os procedimentos descritos nos capítulos 4, com a finalidade de gerar as hipóteses de contatos e exibir estes resultados na tela.

ANEXO II

Composição dos Programas.

ObjAN.exe – É composto pelos seguintes códigos fontes:

Fontes C / C++:

CustomMath3.cpp
ObjAN03009-05.cpp
X_ObjANLoader2_5.cpp

Headers:

CustomMath3.h
X_ObjANLoader2_5.h

ObjREC.exe É composto pelos seguintes códigos fontes:

Fontes C / C++:

CustomMath3.cpp
Eixos.cpp
Hand.cpp
Iluminacao.cpp
Init_stds.cpp
Misc.cpp
Mouse.cpp
Te03009-05.cpp
Teclado.cpp
X_Loader2_5_1.cpp

Headers:

CustomMath3.h
X_Loader2_5_1.h
Misc.h
Te03009-05.h
refxyz.h