HENRIQUE MIYAMOTO

HUMBERTO NOMURA NISHIZAKI

ESTUDO E DESENVOLVIMENTO DE UM SIMULADOR CIRÚRGICO

Projeto de Formatura apresentado à Escola Politécnica da Universidade de São Paulo para a obtenção de título de engenheiro mecânico com ênfase em automação e sistemas

SÃO PAULO

2001

HENRIQUE MIYAMOTO
HUMBERTO NOMURA NISHIZAKI

ESTUDO E DESENVOLVIMENTO DE UM SIMULADOR CIRÚRGICO

Projeto de Formatura apresentado à Escola Politécnica da Universidade de São Paulo para a obtenção de título de engenheiro mecânico com ênfase em automação e sistemas

Orientador:
Prof. Dr. Lucas A. Moscato

SÃO PAULO
2001

Às nossas famílias, por todo apoio dado ao longo de nossas vidas.

AGRADECIMENTOS

Aos amigos Rogério da Silva Santana, Reneu Luiz Andrioli Júnior e Marcos Alexandre Scholtz, pela ajuda no desenvolvimento do projeto.

Aos alunos de iniciação científica Érick Dario Leon Bueno de Camargo, Tito Coutinho Melco e ao José Augusto Calvo Lonardoni, pelo trabalho prévio.

A todos aqueles que, direta ou indiretamente, ajudaram-nos no sucesso deste projeto.

RESUMO

Neste relatório será apresentada uma revisão bibliográfica do estado da arte na área de manipuladores e sistemas de realidade virtual aplicados na área médica, e o desenvolvimento de um sistema de simulação gráfica de cirurgias minimamente invasivas. O trabalho foi dividido em duas partes, uma é o hardware, que consiste de um apontador espacial com retorno de forças e todo seu sistema de controle, e a outra é o software, que consiste do desenvolvimento dos módulos de controle, de simulação e gráfico.

ABSTRACT

In this report, will be presented the state-of-art of handlers and virtual reality systems for medical applications., and the development of a graphical simulation facility for minimally invasive surgeries. The project was divided in two parts, one is the hardware, which consists of a handler with feedback system and its control system, and the other is the software, which consists of the development of a control module, a simulation module and a graphical module.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

1 INTRODUÇÃO

As cirurgias podem ser classificadas em cirurgias invasivas e minimamente invasivas. As cirurgias invasivas são as tradicionais, nas quais os cirurgiões abrem passagem através do tecido externo até a região de interesse, enquanto que as minimamente invasivas são resultantes de novas técnicas, nas quais o cirurgião atinge a região interna de interesse através de pequenas hastes, que contém um dispositivo mecânico de pinças. Essas hastes são introduzidas no paciente por pequenas incisões no tecido externo. A visualização nesse caso é feita através de uma câmera de vídeo inserida pelas hastes.

As cirurgias minimamente invasivas possuem as seguintes vantagens: mínimos traumas no organismo, baixos índices de infecções hospitalares e baixos custos do pós-operatório. Entretanto elas possuem algumas desvantagens, como as limitações mecânicas dos instrumentos cirúrgicos disponíveis e a dificuldade de manuseio dos mesmos. Por isso tais cirurgias necessitam de um maior treinamento e planejamento dos procedimentos a serem realizados.

A melhor maneira para planejar os procedimentos seria a realização de ensaios. Esses ensaios podem ser realizados tanto em modelos plásticos como em corpos mortos, entretanto ambos possuem limitações na disponibilidade e na incapacidade de prever todas as situações. Uma terceira solução é a utilização de simuladores controlados por computador.

O simulador deve fornecer todas as informações que o cirurgião necessita para realizar o procedimento cirúrgico: a visão e o tato. A visão é suprida pelas imagens da câmera de vídeo, que podem ser geradas pelo computador através da modelagem gráfica tridimensional do organismo do paciente; já a sensação de tato exige dispositivos com retorno de força na mão do usuário. Algoritmos implementados no simulador devem captar os movimentos do cirurgião e processá-las de forma a gerar imagens e sinais para se gerar um retorno de força no dispositivo de controle.

Para captar os movimentos e dar o retorno de força utiliza-se dispositivos conhecidos como "haptic devices", que no presente projeto serão denominados "mouses espaciais", que são dispositivos dotados de motores e sensores que captam o movimento da mão do usuário e ao mesmo tempo podem gerar um retorno de forças no dispositivo.

2 ESTADO DA ARTE

Devido à importância citada anteriormente, a área de cirurgias minimamente invasivas e treinamento em ambientes virtuais vem sendo intensamente pesquisada. É possível encontrar informações sobre o desenvolvimento de simuladores que possibilitam a interação simultânea de dois ou mais usuários, simuladores especializados cirurgias específicas, mouses comerciais, software de interface.

Segundo as referências pesquisadas, os simuladores devem ter uma taxa de atualização gráfica que varia entre 10 e 25 Hz, e uma taxa de atualização de força de no mínimo 40 ou 200 Hz, a taxa de amostragem mínima para uma boa resposta de força deve ser de no mínimo 200Hz. Os dispositivos Phantom utilizam uma taxa de amostragem entre 500 e 1000 Hz.

2.1 HARDWARE

2.1.1 PHANTOM DESKTOP– DISPOSITIVO DE CAPTURA DE COORDENADAS EM SEIS GRAUS DE LIBERDADE, COM RETORNO DE FORÇA EM TRÊS GRAUS DE LIBERDADE

Desenvolvidos no MIT e comercializados pela SensAble Technologies Inc., os manipuladores espaciais PHANTOM™ fornecem dispositivos de captura de coordenadas espaciais com ou sem retorno de força. A seguir apresentam-se as características principais deste modelo:

Figura 1 – O Phantom Desktop

Características básicas:

Resolução nominal	1100 dpi (0,02 mm)
Área de trabalho	16 x 13 x 13 cm
Backdrive friction	0,06 N
Força máxima aplicável	6,4 N
Força continuamente aplicável (24 h)	1,7 N
Resistência à flexão	3,16 N/mm
Inércia (Massa aparente nos dedos)	< 75 g
Dimensões da base	18 x 16 cm
Retorno de força	3 Graus de liberdade (x, y, z)
Captura de posição	6 Graus de liberdade (x, y, z, yaw, pitch, roll)+
Tensão de entrada	90 – 260 V AC
Freqüência da rede	47 – 63 Hz
Corrente de entrada	2 A à 115 V AC 1 A à 230 V AC
Requisitos básicos de sistema	PCs com CPU Intel: Windows 2000 ou Windows NT 4.0 Pentium 300 MHz 64 Mb RAM 30 Mb livres em disco Placa aceleradora gráfica
Requisitos básicos de sistema (continuação)	Silicon Graphics: Octane com Irix 6.5 64 Mb RAM 30 Mb livres em disco Placa aceleradora gráfica
Interface	Porta Paralela EPP via cabo DB-25. Não exige placa especial

Tabela I– Características do Phantom

2.1.2 HAPTICMASTER

O HapticMaster é um manipulador de seis graus de liberdade que emprega um mecanismo paralelo para aplicar as forças de reação nos dedos do operador. A manopla do manipulador é sustentada por três conjuntos de pantógrafos.

A seguir está uma figura do equipamento:

Figura 2 – O HapticMaster

O sistema do HapticMaster utiliza três conjuntos de pantógrafos ao invés de mecanismos lineares. Cada pantógrafo é acionado por três motores DC, que são energizados por amplificadores PWM. A ponta superior do pantógrafo é conectada a um vértice do topo da plataforma por uma junção esférica. Esse mecanismo tem as mesmas vantagens de um mecanismo octaédrico, mas com esta configuração ela aumenta o volume de trabalho e a *backdrivability* do manipulador paralelo.

A área de trabalho do manipulador é um volume esférico de aproximadamente 40 cm. O ângulo de cada junção é medida por potenciômetros. A carga máxima aceitável é de 2,5 kg.

2.1.3 MOUSE ESPACIAL DESENVOLVIDO PELO DEPARTAMENTO DE ENGENHARIA MECATRÔNICA E SISTEMAS MECÂNICOS DA EPUSP

Alunos de iniciação científica do Departamento de Engenharia Mecatrônica da EPUSP desenvolveram um dispositivo de interface entre o computador e a mão do operador, chamada de "mouse espacial". Trata-se de um dispositivo contendo uma base fixa a uma mesa, três braços e três articulações, resultando num dispositivo com três graus de liberdade na sua extremidade. Um quarto grau de liberdade pode ser futuramente instalado na forma de uma pinça, que possibilitará a instalação de um acionador on-off ou abre-fecha, com o objetivo de simular com realismo a manipulação da ferramenta pelo usuário do simulador.

A seguir apresenta-se na figura 3 a configuração mecânica do mouse espacial desenvolvido:

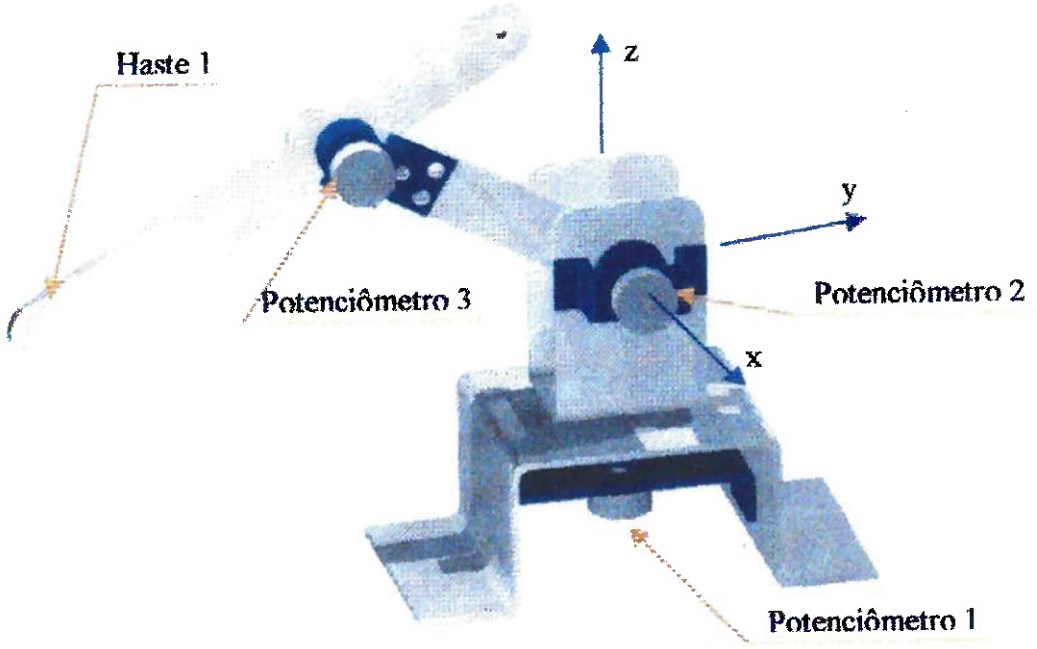

Figura 3 – Mouse espacial

Para que seja possível saber a posição da ponta do mouse espacial a cada instante, três potenciômetros foram instalados nas três articulações do mouse espacial. Estes potenciômetros geram um nível de tensão proporcional ao ângulo imposto neles, de forma aproximadamente linear, fazendo com que seja possível determinar a posição da ponta do mouse espacial a partir de cálculos geométricos simples.

Os potenciômetros foram escolhidos como sensores de posição por seu baixo custo e facilidade de instalação por seu tamanho compacto. Por outro lado, eles são periféricos analógicos, requerendo um hardware adicional, geralmente um conversor analógico/digital para se conectar ao computador, além de serem dispositivos mecânicos, portanto propensos a um desgaste mecânico que reduz sua vida útil .

Para validar a linearidade dos potenciômetros foi feito um ensaio de suas curvas de voltagem de saída versus deslocamento angular, no qual obteve-se um grau de linearidade satisfatório para o intervalo de deslocamento a ser utilizado.

A saída analógica dos potenciômetros é ligada ao microcomputador através da placa de aquisição de dados CAD12/36 da Lynx, onde é convertida para um sinal digital que pode ser utilizado para o processamento de sua posição no espaço e nos programas de saída gráfica.

Segundo informações apontadas no capítulo 2 – Estado da Arte, a taxa de amostragem mínima para se conseguir uma boa resposta de força é de 200 Hz, taxa que é possível de ser atingida na placa CAD12/36, a ser descrita com maiores detalhes no capítulo 2.1.4.

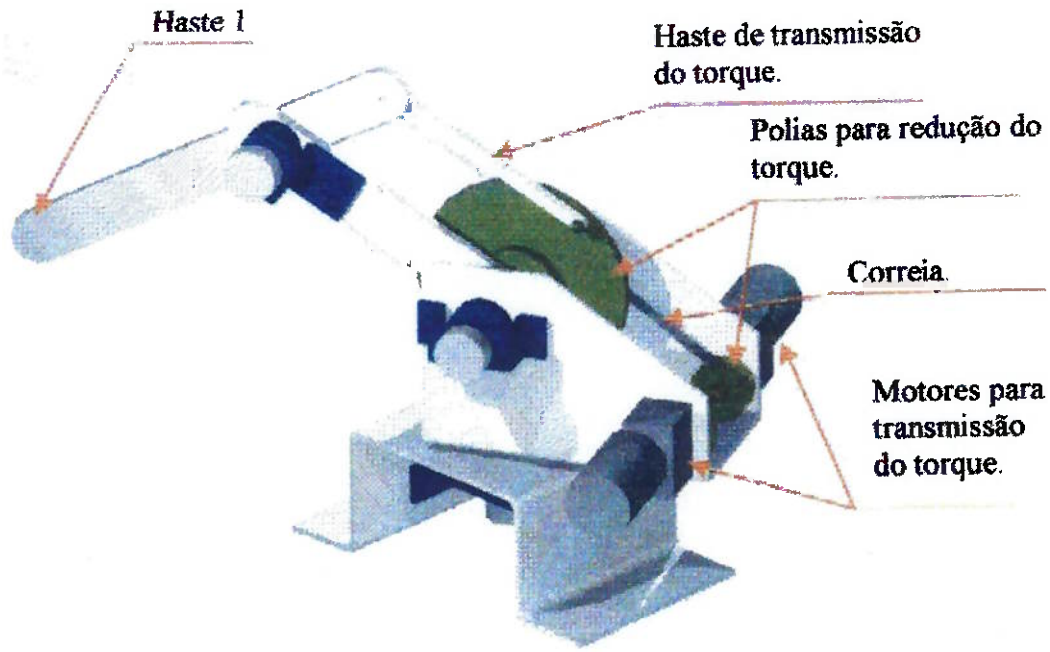

Figura 4 – Mouse espacial com retorno de forças

O torque é transmitido pelo seguinte método:

A força que deve ser aplicada no mouse é transmitida, por um programa de controle do sistema de "feedback", aos motores;

Os motores transmitem um torque às polias motoras e estas, através das correias, transmitem o torque (ampliado pela diferença de raios das polias) às polias movidas;

As polias movidas transmitem o torque às hastes que, através de suas articulações, transmitem a força até a ponta da haste do operador.

2.1.4 PLACA DE AQUISIÇÃO DE DADOS CAD12/36

A CAD12/36 é uma placa de expansão que permite integrar o uso de microcomputadores compatíveis com a família IBM PC ao meio ambiente externo. A CAD12/36 faz o elo entre os micros e os diversos sinais que existem externamente, sejam eles sinais analógicos ou digitais. A CAD12/36 possui, basicamente, os seguintes recursos:

- Conversor analógico/digital (A/D), para leitura de sinais analógicos;
- Entradas e saídas digitais que permitem a leitura e acionamento de variáveis digitais;
- Base de tempo interna e contadores, que permitem temporizar as operações do sistema;
- Expansão para conexão de subsistemas, geralmente conversores Digitais/analógicos (D/A), que podem ser usados para produzir sinais analógicos de estímulo, controle, set-point ou geração de formas de onda.

Características gerais da placa CAD12/36:

- Conversor A/D de 12 bits de resolução;
- 16 entradas analógicas simples ou 8 diferenciais multiplexadas;
- Possibilidade de, com o auxílio de circuitos externos, realizar amostragem simultânea de até 16 canais;
- Suporte para interrupções;
- Base de tempo interna (2,00 MHz), com 3 contadores ; temporizadores de 16 bits;
- Até 4 saídas analógicas ou outras expansões;
- 16 entradas digitais;

- 16 saídas digitais;
- Seqüência de leitura e ganhos programáveis através de memória de canais;
- Aquisição em "burst" propiciada por buffer (FIFO) de 16 posições;
- Suporte para DMA (Direct Memory Access ou Acesso Direto à Memória), permitindo a velocidade máxima de coleta de sinais independentemente da velocidade da UCP (Unidade Central de Processamento) do microcomputador.

2.2 SOFTWARE

Existem diversos softwares que permitem o desenvolvimento de simuladores ou aplicações para dispositivos hápticos. Boa parte deles sugerem a utilização do padrão OpenGL para as aplicações gráficas.

2.2.1 GHOST® SDK

A própria SensAble Technologies, inc. fornece soluções em software para o desenvolvimento de ferramentas que facilitam o desenvolvimento de aplicações táteis. O GHOST® SDK (General Haptic Open Software Toolkit) é um software desenvolvido em C++ que facilita a tarefa de criar estas aplicações. Ele cuida de todo o aspecto computacional para simular a física do tato, assim o usuário só precisa se preocupar com as propriedades desejadas para os objetos, como localização, massa, coeficiente de atrito e resistência. Equipes de desenvolvimento podem criar e distribuir bibliotecas que adaptam outras aplicações, incluindo soluções médicas, de animação, projeto e CAD.

2.2.2 FREEFORM™ MODELING SYSTEM

Outro software da SensAble, este programa aproveita toda a capacidade de seus apontadores espaciais. Esta é a primeira ferramenta que permite aos escultores e

designers usarem seu senso tátil para modelar formas tridimensionais no computador. Afirma ser tão intuitivo quanto modelar argila ou espuma, aliando todas as ferramentas computacionais gráficas disponíveis em modeladores 3-D.

2.2.3 LHX

O LHX (Library for Haptics) é um software dividido em módulos desenvolvido na Universidade de Tsukuba, que visa desenvolver aplicações variadas para dispositivos como o Phantom e o Argone. Ele é composto por sete módulos: *device driver* – gerencia os sensores de entrada e os atuadores de saída da interface haptíca; *haptic renderer* – é responsável pelo cálculo das forças, dureza e viscosidade a serem simulados; *model manager* – implementa os objetos virtuais, controlando suas formas e características; *primitive manager* – responsável pelas formas primitivas dos objetos, como cilindros, esferas, cubos, etc...; *Autonomy engine* – determina o comportamento dos objetos virtuais, a gravidade, viscosidade e elasticidade são implementadas nesse módulo; *Communication interface* – permite que o sistema seja utilizado por múltiplos usuários; *Visual display manager* – gera a imagem gráfica do ambiente virtual.

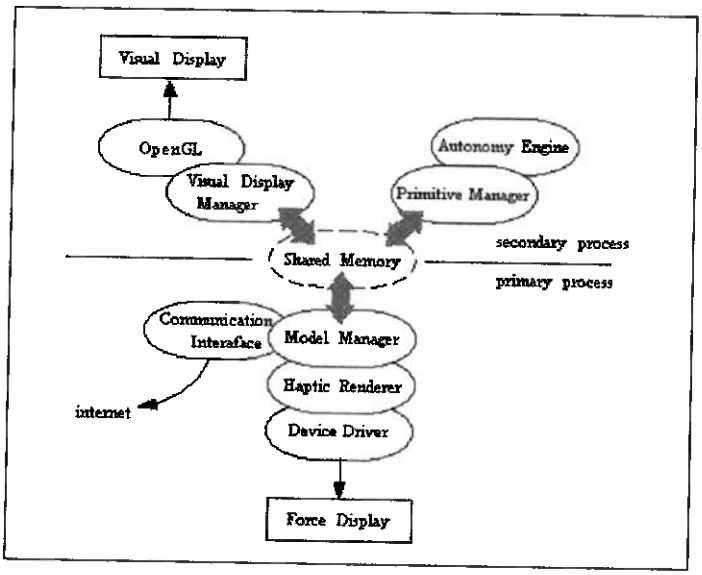

Figura 5 – Estrutura básica do LHX

2.2.4 VRML

O VRML (Virtual Reality Modelling Language) é um formato aberto para gráficos tridimensionais na Internet. Tecnicamente falando, o VRML não é nem realidade virtual nem uma linguagem de modelagem. Realidade virtual tipicamente implica em experiências 3D imersivas (como visores e capacetes de realidade virtual) e dispositivos de entrada 3D (Como luvas digitais e apontadores espaciais). Porém este não é o objetivo da linguagem VRML. Ele é simplesmente um formato de troca de arquivos e dados 3D, usando e definindo uma linguagem muito semelhante às aplicações 3D encontradas no mercado, como transformações hierárquicas, fontes de luz, pontos de vista, geometria, animação, neblina (fog), propriedades de materiais e criação e aplicação de texturas.

Outra característica é que a linguagem VRML é análoga à popular linguagem HTML. Isto significa que o VRML serve como uma linguagem simples e multiplataforma para publicar páginas 3D na Internet, e isto é motivado pelo fato de muitas aplicações serem melhor experimentadas tridimensionalmente, como jogos, visualizações científicas e em engenharia, experiências educacionais e arquitetura. Estas aplicações requerem uma grande interação, animação e participação do usuário, muito além do que é capaz uma página sem profundidade como a proporcionada pelo HTML.

2.2.5 OPENGL

O OpenGL é um software de interface para hardware gráfico. Trata-se de uma biblioteca gráfica com aproximadamente 150 comandos distintos, que são utilizados para especificar objetos e operações a fim de se obter aplicações 3-D interativas, através da criação e renderização de modelos 3-D a partir de formas primitivas. Ela também se caracteriza por ser portável entre quaisquer plataformas com a biblioteca OpenGL instalada (software), ou implementada (hardware), desde PCs comuns até estações de trabalho baseadas no sistema operacional UNIX. Desta forma, o OpenGL acabou por tornar-se um padrão em interfaces gráficas 3D no mercado. Dentre as plataformas que suportam o OpenGL estão o MS-Windows, Linux, SGI, Macintosh e Sun.

Para que o OpenGL seja independente da plataforma utilizada, nenhum comando que envolva operações gráficas em janelas e rotinas para entrada de dados foi incluído no OpenGL. No lugar disso, deve-se trabalhar através do sistema de janelas que está rodando na plataforma utilizada ou outras bibliotecas que ofereçam funções de manipulação de eventos e janelas, como o GLUT (OpenGL Utilities Toolkit).

O OpenGL fornece um poderoso mas primitivo conjunto de comandos de renderização de modelos 3-D. Isso dá muita liberdade ao programador, mas ao mesmo tempo apresenta uma complexidade muito grande na execução de comandos de alto nível. Apesar disso, o usuário pode desejar ter sua própria biblioteca de rotinas e comandos baseados no OpenGL de forma a simplificar o seu trabalho. Para isso, já foram desenvolvidas diversas rotinas e bibliotecas adicionais que apresentam características especializadas, como o OpenGL Utility Library (GLU), a OpenGL Extension to the X Window System (GLX), que contém formas de se criar modelos em OpenGL e associando-os a janelas gráficas em sistemas X Window; e o Open Inventor, que se caracteriza por ser uma biblioteca orientada a objetos baseada em OpenGL que fornece objetos e métodos para criar aplicativos gráficos 3-D. Criado pela Silicon Graphics e escrito em C++, Open Inventor fornece objetos e modelos pré-construídos em OpenGL para criar e editar cenários tridimensionais de forma muito mais fácil e rápida.

Apesar de ser utilizado na maioria dos casos em hardware específico para gráficos 3-D, implementações do OpenGL apenas por software são também possíveis, o que fazem com que o alcance dele seja maior ainda, principalmente para usuários de PCs menos sofisticados. Além disso, a tendência é a de que placas gráficas aceleradoras 3-D sejam itens comuns para o usuário comum em pouco tempo, possibilitando o alcance cada vez maior de aplicações otimizadas.

Para se renderizar uma imagem na tela, geralmente segue-se o seguinte roteiro de comandos:

1) Construir formas a partir de primitivas geométricas, criando por meio destas descrições matemáticas de objetos (O OpenGL considera pontos, linhas, polígonos, imagens e bitmaps como primitivas);

2) Organizar os objetos em um cenário tridimensional e selecionar o ponto de visão desejado da cena montada;

3) Calcular a cor de todos os objetos. A cor pode ser explicitamente atribuída pelo programa, determinada por uma condição de iluminação, obtida após uma "colagem" de uma textura sobre o objeto, ou uma combinação destas três ações.

4) Converter toda a descrição matemática dos objetos em pixels na tela do computador. Este processo é chamado rasterização (rasterization).

Durante estes estágios, OpenGL pode executar outras operações, como eliminar partes dos objetos que estão cobertos por outros objetos no cenário. Adicionalmente, depois que a cena é rasterizada é possível fazer algumas operações nos pixels antes que eles sejam desenhados na tela.

2.2.5.1 GLUT – The OpenGL Utility Toolkit

O OpenGL Utility Toolkit é uma interface de programação que fornece ferramentas para simplificar a criação de programas baseados no software de interface gráfica OpenGL.

Uma das maiores realizações do OpenGL foi ser um software de interface desenvolvido para ser independente do sistema utilizado. Porém, uma operação como a

criação na tela de uma janela requer chamadas diretamente no sistema de janelas utilizado. Estas chamadas são descritas através de uma interface de programação que forneça a ligação entre o OpenGL e o sistema de janelas adotado, como por exemplo o API do Windows. Toda esta independência do OpenGL resulta num sistema muito poderoso, mas extremamente difícil de ser utilizado, uma vez que o programador precisa estar bastante familiarizado com a API em que ele está programando, e aprender APIs pode ser bastante cansativo. Além disso, ao se escrever um programa que utilize o OpenGL somado às chamadas ao sistema de janelas utilizado, o programa acaba se tornando dependente do sistema de janela, perdendo sua portabilidade. O GLUT foi desenvolvido a partir disto para preencher esta necessidade de uma interface entre o OpenGL e o sistema de janelas adotado pelo programador. A interface foi desenvolvida para ser simples e ao mesmo tempo servir para se criar programas eficientes em OpenGL.

Através de diversas funções é possível executar funções como inicializar e criar múltiplas janelas para renderização pelo OpenGL, manejar eventos relacionados com estas janelas, controlar diversos dispositivos de entrada, carregar mapas de cores, inicializar e desenhar primitivas gráficas tridimensionais (cubos, cones, esferas, toróides tanto sólidos como em *wireframe*), gerenciar processos em segundo plano, oferecer suporte à criação de menus simples, suporte para criação de fontes, além de gerenciar o loop de execução do programa após ter concluído todo o processo de configuração do mesmo.

2.2.6 OPEN INVENTOR

O Open Inventor é um software gráfico 3-D orientado a objetos. O Open Inventor é uma biblioteca de objetos e métodos, que podem ser utilizados para criar programas 3-D interativos. Independente do sistema de janelas utilizado, escrito e utilizado em C++, inclui no seu pacote algumas funções em C.

O Open Inventor é um conjunto de "blocos" (objetos) que permitem ao usuário usar todo o potencial de equipamentos de hardware gráficos com o mínimo de esforço em programação. Graças às características de encapsulamento e herança características da programação orientada a objetos, os programas são escritos de forma mais intuitiva e

rápida pelo programador, que passa a se preocupar com o que fazer, ao invés de como fazer.

Baseado em OpenGL, o programa inclui bibliotecas que podem ser utilizadas, ampliadas e distribuídas para outros usuários.

O Open Inventor utiliza o OpenGL para renderização. No OpenGL, no entanto, o processo de renderização é explícito, enquanto que no Open Inventor a renderização, assim como todos os outros comandos, são encapsulados dentro dos objetos. Isto isola o desenvolvedor de detalhes que não são interessantes para o completo entendimento da aplicação, o que simplifica este processo.

No manual do OpenGL, o autor faz um paralelo entre a programação gráfica em OpenGL e no Open Inventor à construção de uma casa. No OpenGL, a construção é acompanhada pelo seu criador passo a passo, que precisa comprar cada peça – tijolos, parafusos, cimento, madeira, telhas separadamente e acompanhar a montagem da casa desde o assentamento dos tijolos à passagem de fiação elétrica pelas paredes. No Open Inventor, no entanto, o criador encontra conjuntos de construção – paredes com toda a fiação e tubulação prontas, telhado, portas e janelas já prontos, bastando encaixá-los como desejar. Assim observa-se que no processo de criação da casa pelo OpenGL o criador tem total liberdade para construí-la como desejar, mas no entanto precisa conhecer cada processo (assentamento de tijolos, colocação de piso) detalhadamente e de forma a fazer o melhor aproveitamento de recursos. Já no Open Inventor a construção é bem mais rápida e simples, a construção já é otimizada, porém o criador já não possui tanta flexibilidade na criação da casa.

3 PROJETO DO SIMULADOR CIRÚRGICO

Neste projeto foi desenvolvido um simulador para o mouse espacial desenvolvido pelo departamento. Como o projeto é uma continuação de um trabalho de iniciação científica, ele é baseado nos trabalhos realizados anteriormente, assim a placa de aquisição é a CAD12/36 da Lynx, e o mouse espacial é o mesmo construído pelos alunos de iniciação científica.

O ambiente gráfico do simulador está desenvolvido na linguagem C, utilizando-se o programa MS-Visual C++, além das bibliotecas gráficas do OpenGL e do GLUT.

O simulador está dividido em módulos, de maneira semelhante ao LHX. Foi necessário desenvolver um módulo gráfico, um módulo de renderização de forças, um *device driver* e um *autonomy engine*. Nesse caso não foi necessário um módulo de comunicação, pois o simulador não aceita múltiplos usuários e o módulo gráfico engloba o gerenciador de primitivas, o gerenciador de modelos e gerenciador de display.

Neste projeto existem três pontos críticos, o primeiro deles é a modelagem e o método de detecção de colisões, o segundo é a modelagem da interferência entres os corpos e o método de análise das forças envolvidas, e o terceiro ponto envolve a escolha do hardware e do software para essa aplicação. A análise desses aspectos será mostrada a seguir.

4 ESTUDO DE VIABILIDADE

O projeto é uma continuação do trabalho desenvolvido pelos alunos de iniciação científica: estudo da placa de aquisição, o programa de aquisição de dados o mouse sem retorno de força, a modelagem dos motores e o mouse com retorno de força.

4.1 ESCOLHA DO HARDWARE E DO SOFTWARE

Um dos aspectos mais importantes dos simuladores cirúrgicos é a necessidade de ser em tempo real, ou seja, as informações devem ser processadas o mais rápido possível de maneira que os cirurgiões tenham a sensação de operar um paciente real. Estudos na área concluíram que para se obter um resultado satisfatório para o cirurgião, as imagens devem ser atualizadas nas telas a uma taxa de no mínimo 10 Hz , o retorno de força deve ocorrer no mínimo a uma taxa de 40 HZ, a taxa de amostragem no dispositivo de comando deve ser de no mínimo 200 Hz. No Phantom essa amostragem é feita numa taxa de 1000 Hz.

O hardware a ser utilizado foi o mouse espacial desenvolvido pelo Departamento e em fase de montagem, que é o dispositivo com retorno de força. Foi necessário uma análise de seu funcionamento para determinar se ele será capaz de atingir as exigências propostas. A taxa de amostragem é determinada pela capacidade da placa de aquisição, sendo ela capaz de amostrar com uma freqüência máxima de 2 MHz, sendo possível dividir essa freqüência com a programação dos timers embutidos na placa.

A parte gráfica foi desenvolvida na linguagem C++, utilizando a biblioteca gráfica do OpenGL e do GLUT. Essa linguagem foi a escolhida devido ao fato do Open GL ser uma linguagem suficientemente poderosa para a aplicação proposta, por ela ser distribuída gratuitamente na Internet como software de domínio público, e devido à familiaridade com a linguagem de programação e com o uso do OpenGL, além do OpenGL ser utilizado como plataforma gráfica de vários simuladores cirúrgicos. Mais vantagens podem ser vistas no capítulo 2 –Estado da Arte.

Os demais módulos também foram desenvolvidos na linguagem C.

Neste projeto, os computadores utilizados foram PC's comuns, sem nenhuma aceleração gráfica por placas especiais; assim, a capacidade de processamento é inferior à utilizada nos simuladores pesquisados em outros projetos (onde foi utilizado o processamento paralelo, com até um processador dedicado para cada módulo). Devido a esta restrição, o simulador não utiliza nenhum efeito fotorrealístico, como sombras, texturas e superfícies complexas. Implementou-se somente a visualização do leiaute da ferramenta e a pele na tela do computador.

4.2 DETECÇÃO DE COLISÃO

Detecção de colisão é o processo pelo qual os sistemas de simulação gráfica têm condições de processar, manipular e reportar informações referentes ao contato físico entre objetos na tela, de forma a determinar as conseqüências lógicas deste contato. Usados em sistemas de simulação de ambientes gráficos reais, eles são extensamente requeridos em aplicações na computação gráfica no desenvolvimento de jogos, na robótica, e em sistemas hápticos como o do presente projeto.

Este sem dúvida é o maior gargalo computacional em cenários virtuais muito complexos, já que todos os polígonos de todos os sólidos que compõe toda a cena devem ser analisados e comparados com outros de forma a caracterizar a colisão, além de executar a conseqüência deste contato, desde impedir a intersecção entre os objetos que se colidem até mesmo provocar a deformação dos mesmos.

Existem inúmeros algoritmos para o processo de detecção de colisão, cada um com uma aplicabilidade e grau de abrangência:

a) Detecção de colisão por força bruta

O algoritmo mais simples, porém o mais exigente em termos computacionais, é o método de força bruta, que consiste basicamente em testar todos os polígonos contra todos os restantes da cena a cada período de animação. Este método funciona bem para modelos simples, com poucos polígonos ou objetos sólidos mais simples, mas devido ao aumento exponencial de comparações a serem feitas na medida em que o cenário vai ficando mais complexo, este algoritmo fica rapidamente inviável de ser utilizado.

O algoritmo pode ser melhorado através do uso de um conjunto de objetos que são candidatos a se colidirem, por exemplo. Neste caso, apenas os polígonos ou sólidos que tenham um indicador marcado como "verdadeiro" são testados. Este método pode desconsiderar os sólidos ou polígonos que nunca irão se colidir entre si no instante considerado, o que pode simplificar a exigência computacional.

Outra técnica que pode ser utilizada é o uso de volumes de envolvimento (bounding volumes). Esta técnica cerca sólidos muito complexos (compostos por muitos polígonos) através do uso de sólidos mais simples, como cubos ou esferas, que se tornam então o sólido de teste. Estes sólidos são mais simples de se comparar matematicamente pelo método da força bruta, tornando o método mais rápido ou mesmo utilizável. No entanto, se dois objetos estiverem muito próximos entre si, seus volumes de limite tornam-se inúteis. Além disso, objetos muito complexos (objetos côncavos, por exemplo) podem requerer volumes de limite muito grandes e que limitam falsos espaços se forem utilizados sólidos como cubos ou esferas. Pode-se então utilizar sólidos mais complexos, mas há um ponto em que aumentar a complexidade destes sólidos irá fazer com que o método perca a sua eficiência.

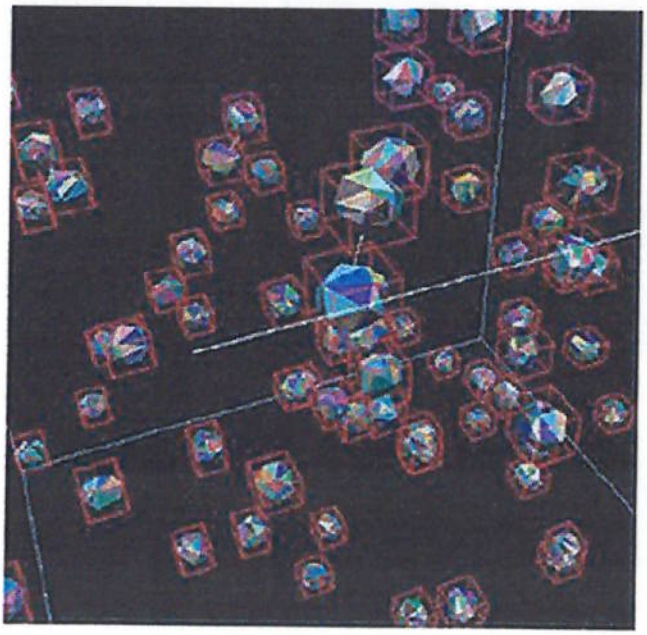

Figura 6 – Volumes de envolvimento nos sólidos

b) Detecção de colisão por geometria analítica

Um dos métodos mais rápidos existentes para a detecção de colisão é o que usa métodos de Ray Tracing. Utilizando técnicas de geometria analítica, este método é utilizado na computação gráfica principalmente em algoritmos de iluminação e de geração de sombras em cenários virtuais.

Define-se primeiramente um raio através de representação vetorial. Essencialmente um raio parte de um ponto inicial e se desloca na direção de um vetor de direção. Então a equação do raio é:

$$PointOnRay = Raystart + t \cdot Raydirection \quad (1)$$

Onde t é um número real entre $[0,\infty]$

Substituindo t por zero na equação, obtém-se o ponto inicial, e substituindo-se valores diferentes obtém-se outros pontos ao longo do vetor.

Para se detectar a colisão entre a ferramenta (simplificada por um cilindro) e um plano representando o órgão do paciente, primeiro deve-se determinar a equação vetorial do plano:

$$X_n \bullet X = d \quad (2)$$

onde:

X_n e X são vetores e d é um número real;

X_n é um vetor normal ao plano;

Dot ($\bullet$) representa um produto escalar;

X é um ponto na superfície do plano (Vetor de comprimento zero)

d é um valor real que representa a distância do plano ao longo da normal; por exemplo, se o ponto pertencer ao plano, d fica igual a zero.

Essencialmente um plano divide o espaço em duas partes, portanto tudo o que é necessário para defini-lo é somente um ponto e uma normal que saia deste ponto e que

seja considerado normal ao plano que se deseja definir, por exemplo, se for utilizado como ponto o ponto de origem (0,0,0) e para a normal o vetor (0,1,0) o plano definido é o formado pelos eixos x e z.

Usando a equação do plano a normal é substituída pelo valor de X_n citado anteriormente e o ponto da qual a normal se origina é substituído por X. O valor de d que está faltando pode ser facilmente calculado utilizando-se um produto escalar.

Se ocorre a intersecção de um raio com um plano então deve haver algum ponto no raio que satisfaça a equação:

$$X_n \bullet PointOnRay = d \qquad (3)$$

pois qualquer ponto que pertence ao plano especificado satisfaz esta equação.

Substituindo-se (1) em (3):

$$(X_n \bullet Raystart) + t \cdot (X_n \bullet Raydirection) = d \qquad (4)$$

E isolando-se t:

$$t = \frac{(d - Xn \bullet Raystart)}{(Xn \bullet Raydirection)} \qquad (5)$$

Esta é a base da detecção de colisão. Através deste cálculo é possível saber qual a distância do Raystart (ponto de partida do raio) até a intersecção com a tela. Todas as variáveis à direita da equação acima estão disponíveis:

- O Raystart, dado pela posição dada pela extremidade do mouse espacial;
- Um pivô, onde se dá a articulação da ferramenta, sendo essencialmente o ponto de passagem da ferramenta ao corpo do paciente;
- A Raydirection, facilmente calculada pelo Raystart e o pivô;
- X_n, previamente definido para posicionar a superfície do órgão virtual;

Como o comprimento da ferramenta d é outro valor conhecido, então escrevendo-se a equação na forma:

$$t = d - \left[\frac{X_n \bullet (PointOnRay - Raystart)}{(X_n \bullet Raydirection)} \right] \quad (6)$$

Pode-se detectar a colisão da ponta da ferramenta, além da direção na qual esta ocorre, a partir do pseudo-código: Se t <= 0 ⇒ Está ocorrendo a colisão.

4.3 CÁLCULO DA INTERFERÊNCIA

Detectado o contato, inicia-se o cálculo da força devido a penetração do apontador no corpo. O simulador é do tipo contínuo, ou seja , ele considera as forças devido a penetração do apontador no corpo, ignorando os efeitos devido ao atrito.

Para modelar as forças de reação dos objetos, foi utilizado inicialmente um modelo massa-mola, com uma malha discretizada e independente, isto é, a penetração em um ponto não afeta os pontos ao redor. Assim a superfície do corpo ficaria modelada como na figura abaixo.

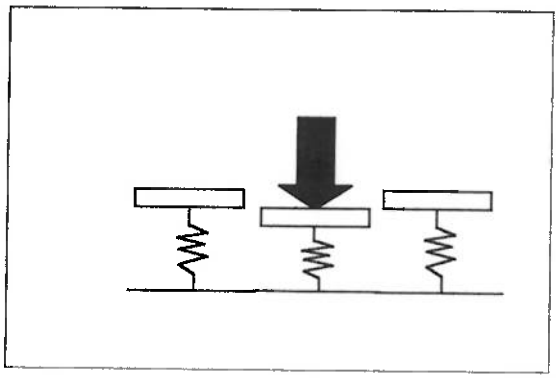

Figura 7 – Modelo Massa-Mola independente

O segundo passo será modelar a resistência da superfície do corpo de maneira que a penetração em um ponto produza deformações nos pontos ao redor. Para isso existem diversos métodos como MEF (elementos finitos), o MDF (diferenças finitas) ou a utilização de modelo analítico por composição de molas em série e paralelo.

4.3.1 MEF (MÉTODO DOS ELEMENTOS FINITOS)

Este é o método mais preciso para o cálculo das forças resultantes devido a penetração no corpo, entretanto exige uma quantidade de cálculos muito elevada, o que inviabiliza o seu uso em um processamento em tempo real. Uma saída possível seria a utilização de um software de elementos finitos (por exemplo o Ansys) para modelar as forças em cada nó superfície do corpo de forma *batch*, e guardar essas informações em arquivo, que seria utilizado para calcular a força resultante. Essa solução apresentaria resultados satisfatórios em relação aos valores das forças, com uma velocidade razoável, entretanto a saída gráfica seria prejudicada, pois esse método permite calcular apenas a força em cada nó, ignorando os efeitos de deslocamento nos nós adjacentes, o que impossibilitaria a saída para a tela. A determinação da deformação nos pontos adjacentes poderia ser feita por uma interpolação matemática, talvez por uma curva spline, o que geraria o efeito de deformação na tela, o que pode não ser a deformação verdadeira.

4.3.2 MDF (MÉTODO DAS DIFERENÇAS FINITAS)

Nesta solução seria utilizado o método das diferenças finitas para calcular as deformações e as forças nos nós da superfície do corpo. Este método não exige um esforço computacional tão elevado quanto o MEF, sendo possível a sua implementação no programa. Entretanto é necessário um estudo profundo da discretização da malha e da tolerância adotada, de forma a garantir a convergência do método. Também será necessário averiguar se o método é capaz de calcular as respostas no tempo necessário.

4.3.3 MODELAGEM POR UM SISTEMA DE MOLAS EM SÉRIE E EM PARALELO

Essa solução consiste em modelar a superfície do corpo em uma malha de nós ligada por molas em várias direções, como no exemplo ilustrado abaixo:

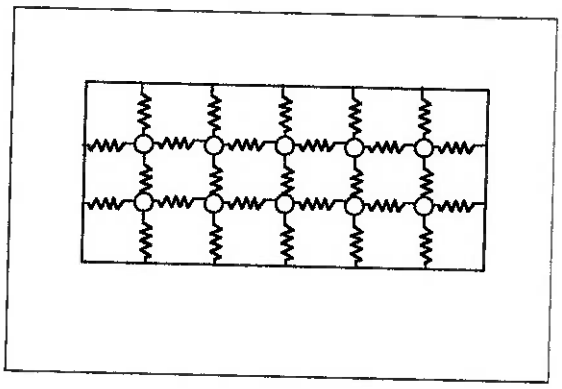

Figura 8 – Malha com um sistema de molas

A solução seria calcular analiticamente os efeitos da penetração do apontador na malha. Para uma malha muito grande esse método se torna inviável, pois a modelagem dessa malha exigiria uma grande quantidade de cálculos.

Uma alternativa é criar uma malha menor (célula), e considerar que os efeitos sobre ela são semelhantes em qualquer parte da malha maior. Essa aproximação não foge muito da realidade, pois ao puxar um ponto do tecido da pele, apenas uma pequena região sofre deformação por influência da força aplicada. Assim bastaria calcular o efeito da penetração do apontador no meio da célula e deslocar esse efeito para o ponto em que ocorreu o contato.

Essa solução ainda precisa ser comparada com os demais, para verificar se os seus resultados são compatíveis.

Esse método pode apresentar um problema de continuidade nas bordas das células e, dependendo do resultado, podem ocorrer saltos na deformação da superfície.

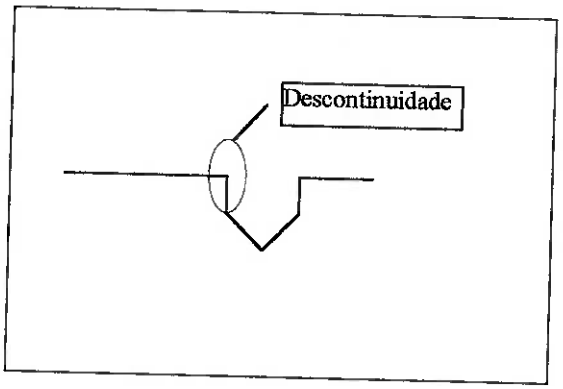

Figura 9 – Problema de descontinuidade na malha

Uma solução para esse problema é utilizar uma interpolação na borda, de maneira a suavizar essa descontinuidade, o que não é real, mas fornece resultados gráficos mais satisfatórios.

Para determinar qual dos métodos é melhor, será realizado um estudo que comparará os resultados. Nesse estudo serão levados em conta, além da precisão dos resultados, o custo computacional e a complexidade do algoritmo, pois em dos requisitos do projeto é a velocidade dos cálculos de força e deformação, pois a taxa de resposta deverá ser de 10 Hz para deformação e de 40 Hz para força, incluindo o tempo de processamento de imagem e controle dos motores.

4.4 MÓDULOS DO SIMULADOR

Baseado nos softwares disponíveis no mercado, o simulador está dividido em módulos, facilitando a sua implementação e a correção de possíveis erros. Ele é dividido em:

4.4.1 DEVICE DRIVER

O *device driver* é baseado no programa de aquisição desenvolvido pelo aluno Erick Darío León Bueno de Camargo, pois seu programa é capaz de calibrar os sensores e configurar a placa de aquisição utilizada. O programa que controla os motores foi

desenvolvido baseados nesses estudos. Ele envia os sinais baseado nos resultados do módulo de renderização de forças.

4.4.2 AUTONOMY ENGINE

O *autonomy engine* foi desenvolvido em paralelo com o módulo de renderização. Como foi dito anteriormente, os objetos possuem como características apenas a constante de elasticidade, pois o modelo adotado foi o massa-mola. A inércia dos objetos foi ignorada, pois o simulador dá apenas a resposta devido à penetração nos objetos, e não é capaz de simular a locomoção dos mesmos. A estrutura desse módulo depende diretamente do método utilizado no módulo de renderização de forças.

4.4.3 MÓDULO DE CÁLCULO DE FORÇAS

Este módulo é responsável pelo cálculo das forças de retorno nos dispositivos de controle. Como foi discutido anteriormente, sua estrutura depende do método adotado para o cálculo das forças e deformações.

4.4.4 MÓDULO GRÁFICO

O módulo gráfico utiliza as bibliotecas do OpenGL e do GLUT, suas entradas dependem dos resultados do módulo de renderização de forças. Aqui é importante salientar que os gráficos devem mostrar apenas a informação estritamente necessária, pois o uso de efeitos visuais estéticos consome uma capacidade computacional indisponível para o projeto.

4.5 CONTROLE DOS MOTORES

Uma das atividades desenvolvidas foi o estudo e a análise do controle dos motores para a atuação no retorno de forças.

O sistema implementado é caracterizado como sendo de malha aberta, pois a intenção é apenas o de gerar uma força aproximadamente proporcional à posição do mouse espacial, não sendo necessário controlar sua posição ou força aplicada; desta forma não há necessidade de uma realimentação da força aplicada pelos motores.

Para atingir esse objetivo, foi necessário modelar os motores e a estrutura, pois a força aplicada na mão do usuário é a resultante de um torque aplicado pelos motores na estrutura. Por sua vez, a saída do motor (torque) é gerada por um sinal elétrico enviado pelo computador, e determinada pelo programa de controle dos motores a partir dos dados relativos à posição do mouse espacial no ambiente virtual.

O sistema esquematizado a seguir mostra como os sub-sistemas estão interligados:

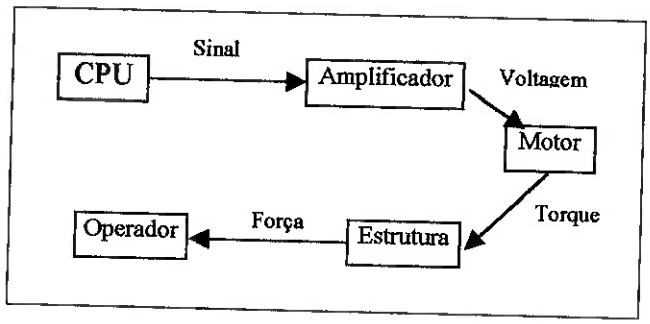

Figura 10 - Integração dos sub-sistemas

5 IMPLEMENTAÇÃO

5.1 MODELAGEM EM ELEMENTOS FINITOS

Para que o algoritmo de retorno de forças funcionasse de forma rápida e próxima à real, optou-se pela simplificação do modelo de pele para um que apresentasse reações apenas na vertical, ou seja, que ignorasse por exemplo as forças de atrito com a pele. Como as operações também implicam em pequenas deformações, considerou-se que o modelo deveria obedecer à lei de Hooke F=kx.

Uma das etapas do trabalho foi o de fazer a análise de curvas de força versus deformação verticais para uma superfície plana presa em suas extremidades, que constituiria a pele virtual e definiria a área de trabalho do simulador desenvolvido. Com as deformações, pode-se inferir um valor de elasticidade para cada ponto discretizado da malha, que se torna um parâmetro de entrada no cálculo do retorno de forças.

Com o auxílio do Sr. Rogério Silva Santana, foi possível fazer toda esta análise sem ocupar uma grande quantidade de tempo, já que o interesse não seria o de explorar o software Ansys, e sim somente obter valores qualitativos de elasticidade da pele nestas condições em cada ponto da malha.

Foi utilizado um modelo de placas, com a seguinte geometria:

Espessura	2,5 mm
Dimensões laterais	100mm X 100mm
Número de elementos	400 (20 x 20 nós)
Número de pontos de amostragem	200 (10 x 10 nós)
Força de deformação imposta	1 N
Módulo de elasticidade E	$81N/mm^2$
Coeficiente de Poisson ν	0,4

Tabela II - Parâmetros para o modelo em Ansys

Como o mouse espacial possui muitas folgas e alta inércia devido aos seus contatos e acionamentos, optou-se por simular um material semelhante à borracha, pois outros elementos mais elásticos poderiam tornar taxas de retorno de forças tão baixos que acabariam sendo filtradas pelo sistema.

Primeiramente foi criado um modelo de pele com 10000 elementos, mas o tempo para efetuar os cálculos de deformação para um ponto, de aproximadamente 15 minutos, motivou a procura de malhas com menos elementos mas que ainda gerassem resultados com qualidade. A malha com 400 nós mostrou-se ideal em termos de precisão de resultados versus velocidade de cálculo (de aproximadamente 4 segundos em cada ciclo).

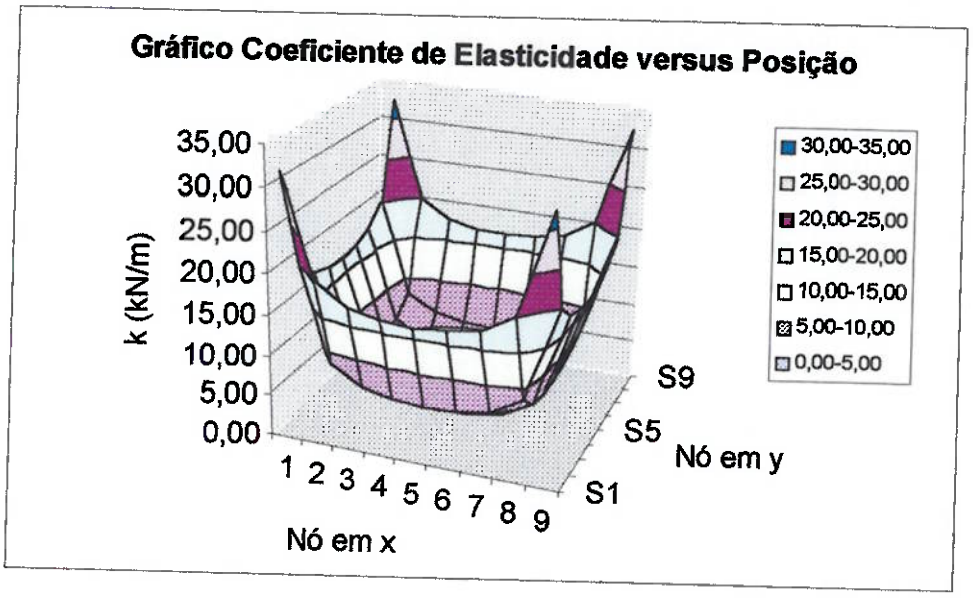

Figura 11 – Coeficientes de Elasticidade da Pele

Uma imagem da malha pode ser vista na figura a seguir:

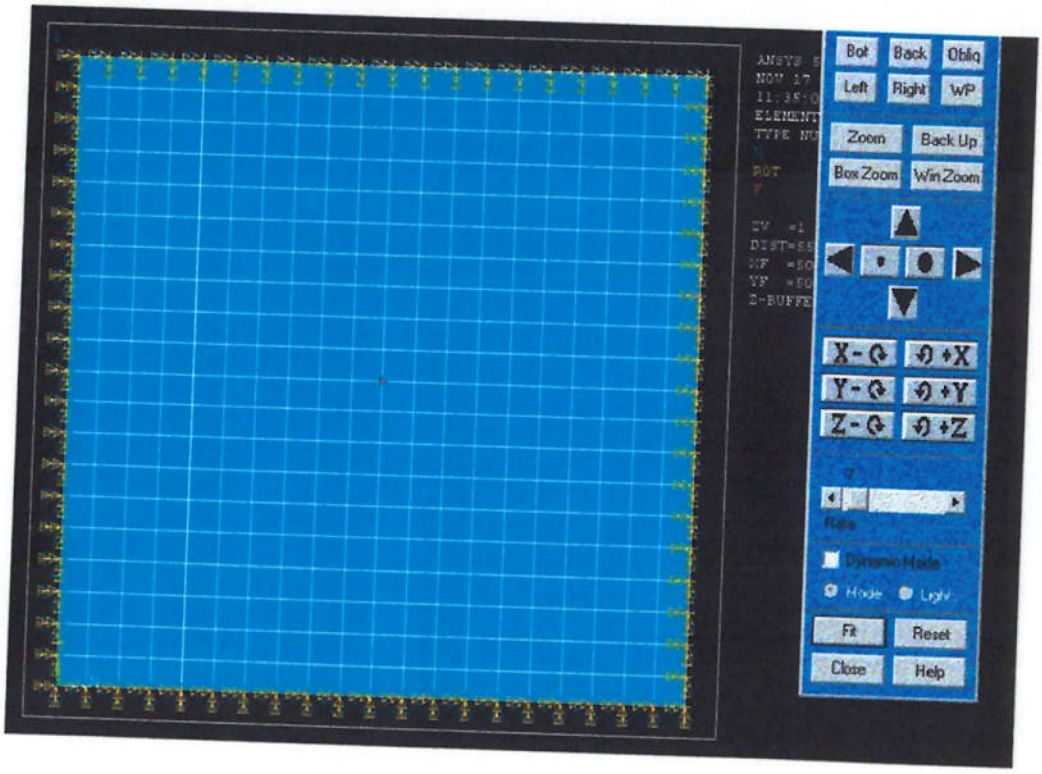

Figura 12 - Modelo da pele em Ansys

Na figura pode-se observar o engastamento feito nas bordas da pele. Pela simetria do modelo, foi necessário calcular deformações para apenas 1/8 dos nós da malha, o que agilizou ainda mais o trabalho.

Aproveitou-se a oportunidade para conhecer algumas potencialidades e funções especiais do Ansys que possibilitam a visualização gráfica de deformações e tensões na malha:

a) Força aplicada em nó próximo às bordas

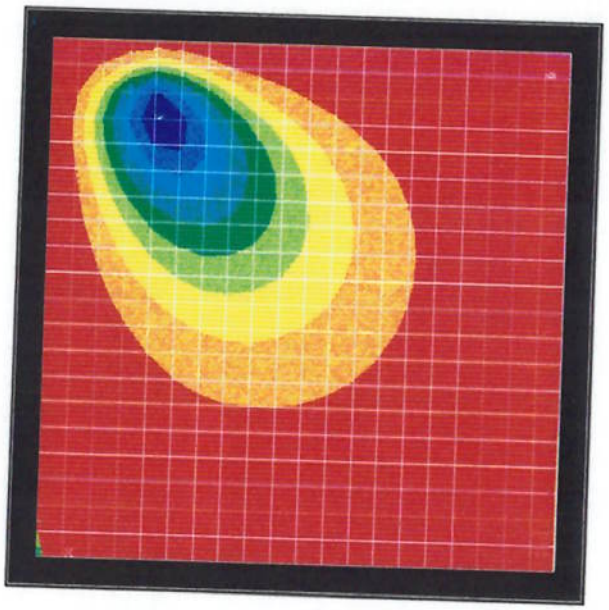

Figura 13 - Deformação para força aplicada próximo à borda

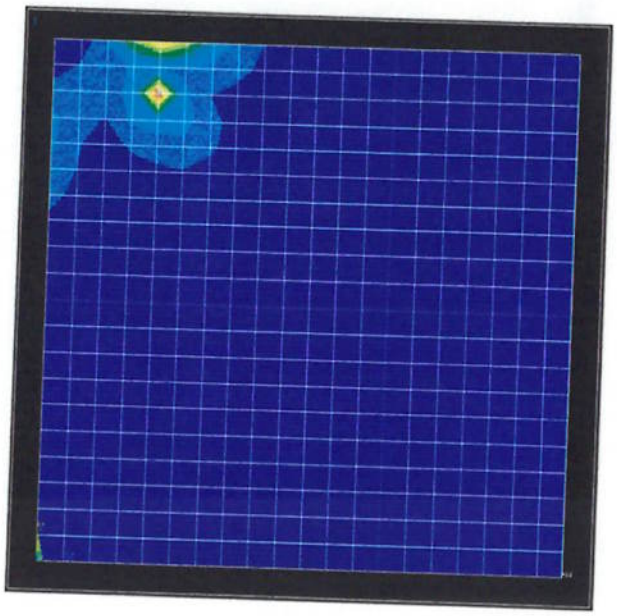

Figura 14 – Tensões de Von Mises na malha devido à força aplicada próximo à borda

Nestas simulações pôde-se observar um fenômeno interessante, mas que não será considerado no simulador: as deformações máximas não se localizam no ponto de aplicação da força quando este é feito próximo às bordas. Isto decorre do fato das bordas terem sido engastadas, o que faz com que o material se comporte como uma viga engastada em uma das extremidades e sendo fletida por uma força no centro da barra. Neste caso a deformação máxima também não se situa no ponto de aplicação da força. A tensão no entanto é máxima no ponto de contato da ferramenta, como pode ser observado na Figura 14 – Tensões de Von Mises na malha devido à força aplicada próximo à borda.

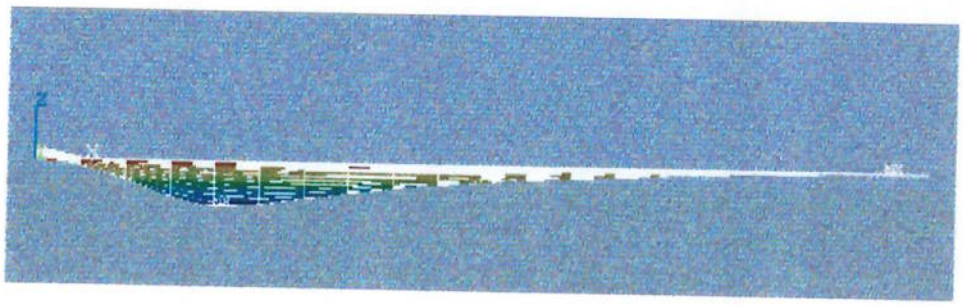

Figura 15 - Deformação vista de perfil

Os resultados obtidos podem ser melhor analisados na seguinte tabela:

32216,49	20096,46	17689,72	16903,31	16700,35	16903,31	17689,72	20096,46	32216,49
20096,46	8340,28	6088,28	5366,82	5183,50	5366,82	6088,28	8340,28	20096,46
17689,72	6088,28	3911,90	3226,22	3053,71	3226,22	3911,90	6088,28	17689,72
16903,31	5366,82	3226,22	2558,46	2391,26	2558,46	3226,22	5366,82	16903,31
16700,35	5183,50	3053,71	2391,26	2225,68	2391,26	3053,71	5183,50	16700,35
16903,31	5366,82	3226,22	2558,46	2391,26	2558,46	3226,22	5366,82	16903,31
17689,72	6088,28	3911,90	3226,22	3053,71	3226,22	3911,90	6088,28	17689,72
20096,46	8340,28	6088,28	5366,82	5183,50	5366,82	6088,28	8340,28	20096,46
32216,49	20096,46	17689,72	16903,31	16700,35	16903,31	17689,72	20096,46	32216,49

Tabela III – Coeficientes de Elasticidade da Pele

A deformação no ponto central, ao contrário, tem a seguinte configuração:

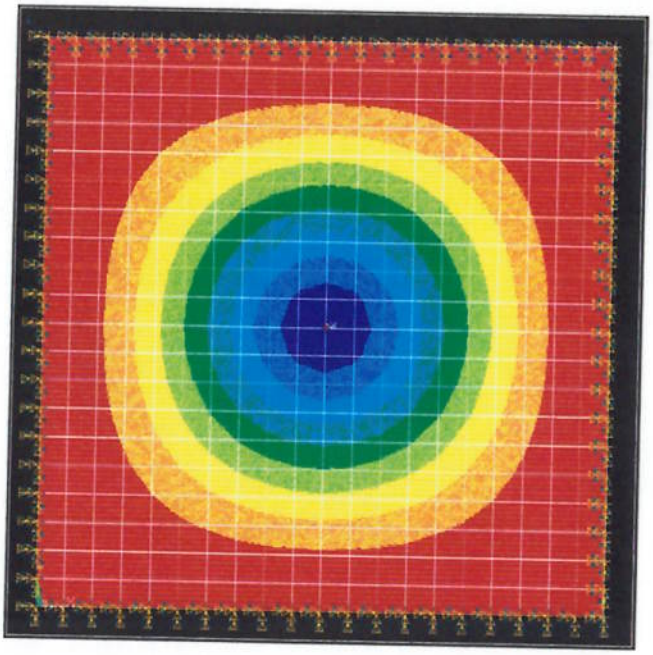

Figura 16 - Deformação devido à aplicação de força no centro da malha

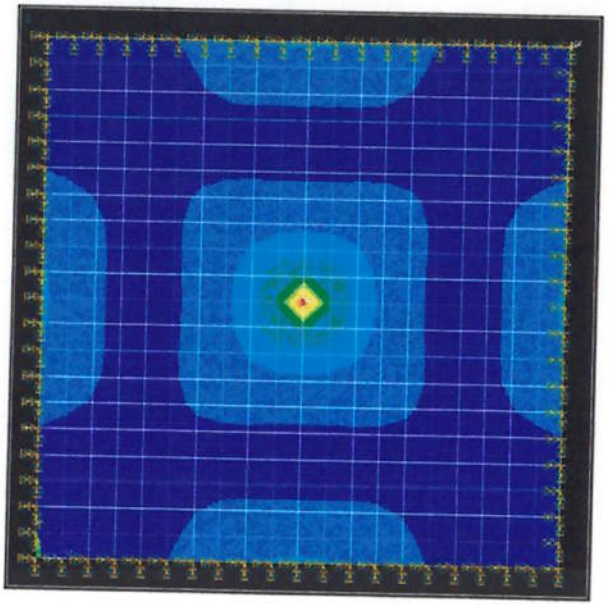

Figura 17 -Tensões de Von Mises devido à força aplicada no centro da malha

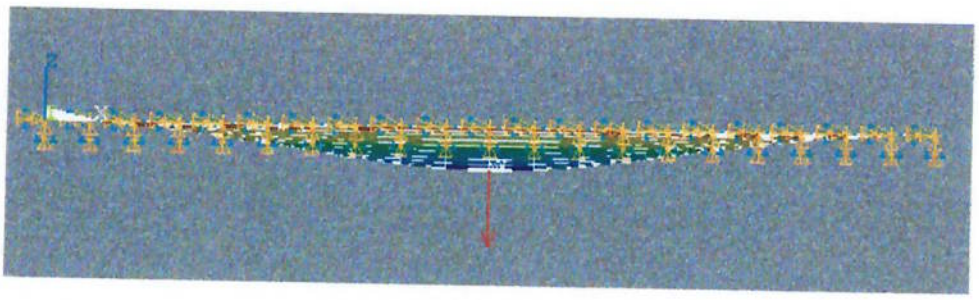

Figura 18 - Deformação vista de perfil

A partir da figura acima, pode-se observar que a força aplicada num ponto influi em praticamente toda a área da malha. O raio de influência é bastante elevado, tendo cerca de 1000 vezes a ordem de grandeza da penetração.

A simetria da malha é bem representada pela Figura 16 - Deformação devido à aplicação de força no centro da malha e pela Figura 17 -Tensões de Von Mises devido à força aplicada no centro da malha.

5.2 MODELAGEM DINÂMICA

Existem duas possibilidades para modelar a dinâmica do mouse com retorno de forças. A primeira é a utilização de um modelo se mecanismo de quatro barras, com acionamentos nas barras de base. A segunda opção é desacoplar as barras, e tratá-las como mecanismos independentes.

5.2.1 MÉTODO DO DESACOPLAMENTO DAS BARRAS

O método é baseado na decomposição da força resultante em uma componente vertical (Fz) e uma horizontal (Fxy).

A força horizontal será aplicada pelo motor da base do mouse, e modelagem será simplesmente a aplicação de um torque na base do mouse de maneira a anular o torque provocado pela força (Fxy).

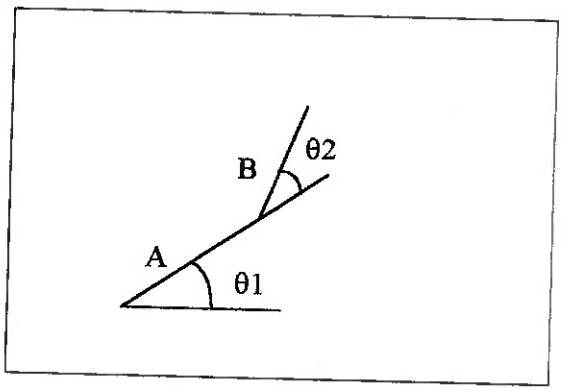

Figura 19 – Vista lateral das hastes do manipulador

$$M = \left(A * \cos(\theta 1) + B * \cos(\theta 1 + \theta 2)\right) * Fxy \quad (7)$$

A força vertical será anulada pelo torque aplicado nas outras duas articulações.

Nesse modelo foi admitida a seguinte hipótese: cada motor irá anular uma parcela igual da força Fz, assim cada barra estará sob efeito de uma força Fz/2.

Diagrama de forças da primeira barra do mouse espacial.

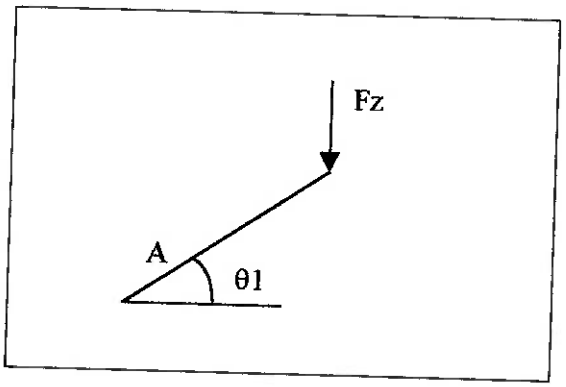

Figura 20 – Representação da força aplicada na haste de suporte

A partir da força aplicada na extremidade da barra, calcula-se o torque que o motor deve aplicar à barra:

$$Mt = Fz * A * \cos(\theta 1) \quad (8)$$

A barra que sustenta a última barra é considerada como se estivesse imóvel, assim deve-se equilibrar os momentos na barra AB, ao redor da articulação que une as duas barra. A força F é provocada pelo torque do motor aplicado a alavanca C, que empurra a barra paralela a primeira.

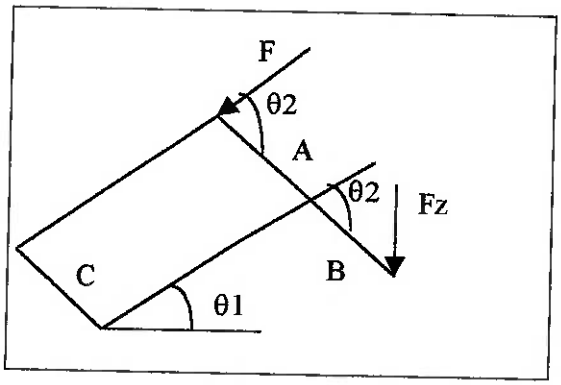

Figura 21 – Representação do equilíbrio de forças na haste da ponta

Da equação de equilíbrio dos momentos, obtêm-se:

$$Mt = C * \frac{A}{B} * \frac{sin(90 + \theta 1 - \theta 2)}{sin(\theta 2)} \qquad (9)$$

Entre as duas modelagens, o método do mecanismo de quatro barras apresenta o resultado mais coerente, sendo este o escolhido para a implementação.

Note que o torque dos motores será 3,2 vezes menor do que o calculado, devido a relação de transmissão entre as engrenagens.

5.3 HARDWARE

Aqui serão apresentados os aspectos considerados em cada componente de hardware utilizado. Um aspecto importante do hardware é a aquisição de dados:

5.3.1 AQUISIÇÃO DOS DADOS

A partir do estudo desenvolvido pelo aluno Erick Dario Leon Bueno de Camargo, foi desenvolvida a interface de aquisição de dados do mouse com retorno de forças.

Inicialmente, foi tentada a utilização de interrupções para realizar a aquisição de dados a uma freqüência constante. Entretanto a solução não se mostrou viável, devido à complexidade em se implementar as interrupções por hardware no ambiente Windows e a falta de necessidade. A aquisição a uma freqüência constante é necessária quando se deseja medir a velocidade. Como essa variável não é utilizada, foi decidido não implementar as interrupções.

Assim foi implementada uma função semelhante à desenvolvida anteriormente pelo aluno de iniciação científica, onde a aquisição é ativada via software a cada loop do programa.

Durante as experiências, foi observado um ruído na aquisição dos sinais. Foi realizado um estudo dos sinais dos potenciômetros (através do uso de um osciloscópio), que não apresentaram ruídos. Assim para contornar esse problema, foi implementado um filtro lógico no programa de aquisição que minimiza essas pequenas oscilações.

Esse filtro lógico consiste em realizar diversas medidas (valor livremente escolhido), e calcular a sua média. Outro artifício utilizado é o truncamento dos valores medidos na segunda casa decimal.Ao realizar os testes, foi percebido que as oscilações se concentravam na terceira casa decimal, assim ao filtrar essa casa, as oscilações praticamente desapareceram.

5.3.2 POTENCIÔMETROS

Para o mouse com retorno de forças, foram testados diversos potenciômetros. A seguir as curvas dos potenciômetros testados:

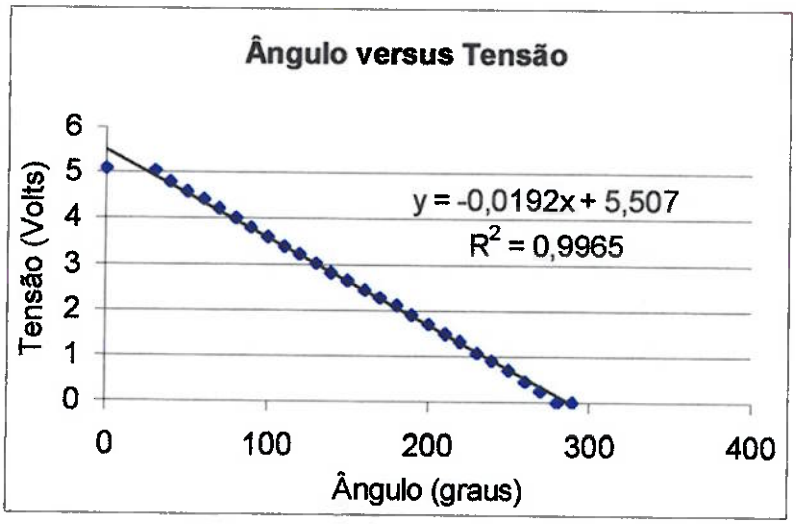

Figura 22 - Potenciômetro 1 Tensão versus Ângulo

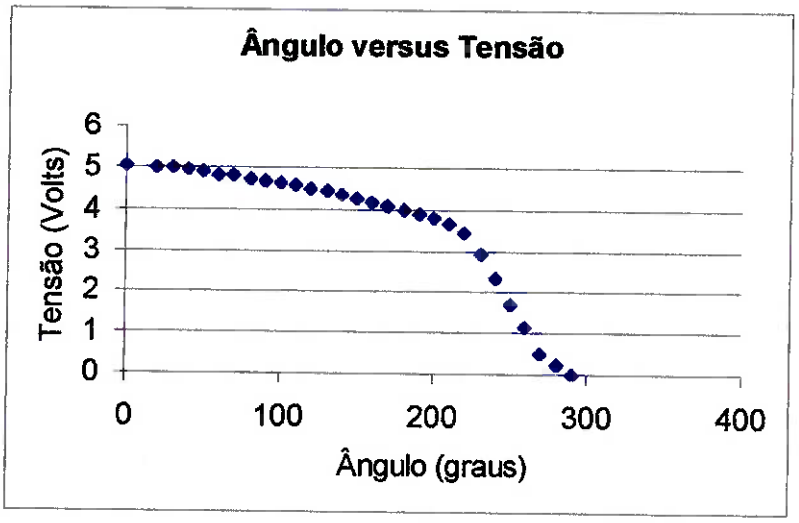

Figura 23 – Potenciômetro 2 Tensão versus Ângulo

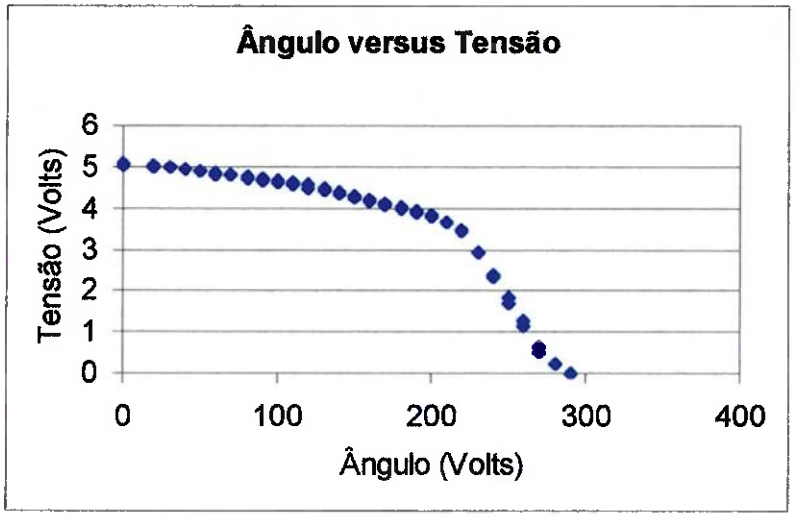

Figura 24 – Potenciômetro 3 Tensão versus Ângulo

Os potenciômetros 2 e 3 não apresentam um comportamento linear totalmente linear, mas no trecho utilizado (de 50 a 150 graus) seu comportamento é linear.

5.3.3 MOTORES ELÉTRICOS

Os motores utilizados no mouse são moto redutores DC do tipo CHP da Bosch.

Nesse projeto é necessário controlar o torque gerado pelos motores, assim foram levantadas as curvas de Torque X Voltagem. Esse estudo foi realizado pelo aluno de iniciação científica Tito Coutinho Melco.

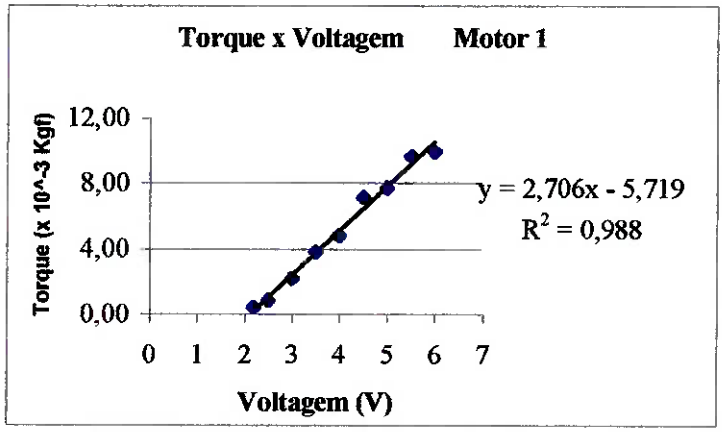

Figura 25 – Curva do motor 1 Torque versus Tensão

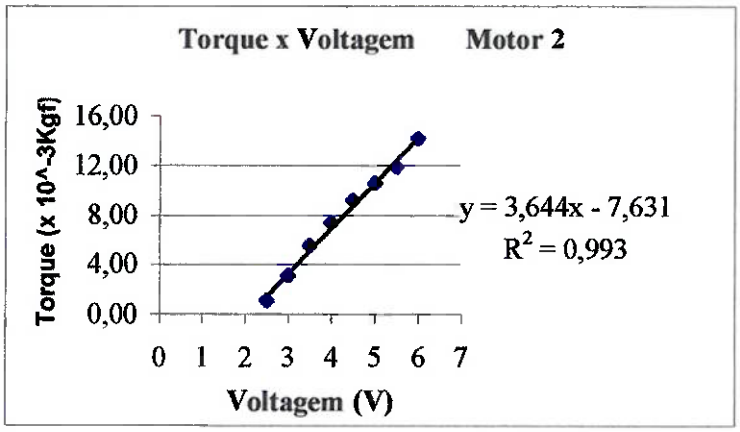

Figura 26 - Torque versus Tensão

Com os resultados foram obtidas as seguintes equações:

$$T_1 = 2{,}706 \times V - 5{,}719$$
$$T_2 = 3{,}644 \times V - 7{,}631 \qquad (10)\ (11)$$

A partir dessas equações e do modelo de forças utilizados, é possível determinar a tensão necessária a ser aplicada.

5.3.4 CIRCUITO DE POTÊNCIA

Foi utilizado um amplificador com realimentação negativa.

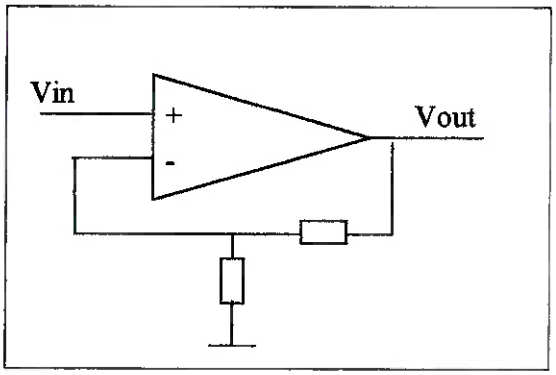

Figura 27 Circuito de Potência

O circuito de potência foi construído pelos alunos de graduação Karine Moriya e Luiz Felipe Almeida Souza, em seu projeto de formatura.

Foi utilizado um amplificador operacional LM12CL da National, utilizando resistores de 20kΩ.

O ganho de tensão é dado pela equação abaixo:

$$Vout = \frac{R1 + R2}{R2} \times Vin \qquad (12)$$

$$Vout = \frac{20 + 20}{20} \times Vin \qquad (13)$$

$$Vout = 2 \times Vin \qquad (14)$$

5.3.5 MOUSE ESPACIAL

A construção do mouse foi concluída pelo aluno Tito Coutinho Melco, que seguiu o projeto original, proposto à FAPESP.

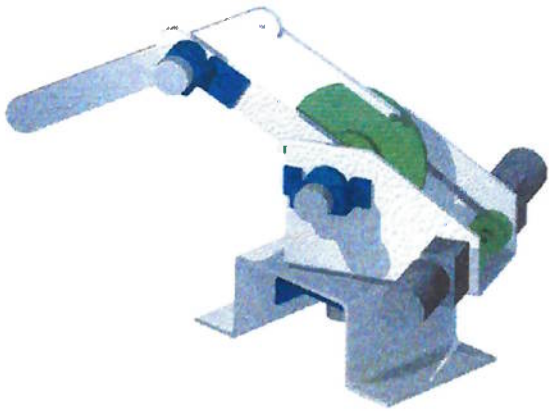

Figura 28 – Mouse espacial

O modelo proposto apresenta apenas duas articulações com motores, o controle do terceiro grau de liberdade não constava do projeto.

Foi observado que o mouse apresenta uma inércia bastante elevada, que pode prejudicar a sensibilidade do retorno de forças ou mesmo o acionamento dos motores. Prevê-se, por exemplo, a presença do fenômeno de histerese no acionamento dos mesmos, o que só poderia ser minimizada através da substituição dos motores por outros de melhor qualidade e numa construção mecânica mais refinada.

Para o mouse foi construída uma base de madeira pesada, que facilita o seu uso. Nessa base foram colocadas duas marcas que ajudam a calibrar o mouse.

Na calibração do mouse, as marcas indicam os seguintes ângulos:

Posição 1: ângulo 1 = 65, ângulo 2 = 0 e ângulo 3 = 50.

Posição 2: ângulo 1 = 90, ângulo 2 = 40 e ângulo 3 = 130.

Essas marcações determinam o volume de trabalho máximo do mouse.

5.4 SOFTWARE

A seguir serão apresentados alguns dos algoritmos utilizados no programa assim como as funções desenvolvidas.

O programa foi dividido em diversos módulos, segundo suas aplicações.

5.4.1 ALGORITMO DE DETECÇÃO DE CHOQUES

Foi desenvolvido um algoritmo de detecção de choque baseado em geometria analítica. Este algoritmo foi encontrado em site especializado em computação gráfica, e utiliza as equações apresentadas anteriormente em 4.2 (b).

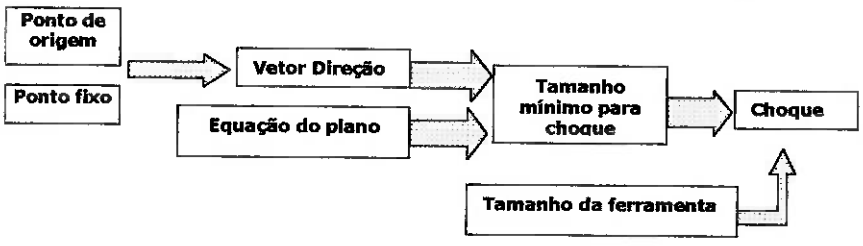

Figura 29 – Algoritmo da detecção de choque

$$PointOnRay = Raystart + t \cdot Raydirection$$

$$X_n \bullet X = d$$

$$X_n \bullet PointOnRay = d$$

$$(X_n \bullet Raystart) + t \cdot (X_n \bullet Raydirection) = d$$

$$t = \frac{(d - Xn \bullet Raystart)}{(Xn \bullet Raydirection)}$$

$$t = d - \left[\frac{X_n \bullet (PointOnRay - Raystart)}{(X_n \bullet Raydirection)} \right] \quad (16)$$

Onde:

X_n e X são vetores e d é um número real;

X_n é um vetor normal ao plano;

Dot ($\bullet$) representa um produto escalar;

X é um ponto na superfície do plano (Vetor de comprimento zero)

d é um valor real que representa a distância do plano ao longo da normal; por exemplo, se o ponto pertencer ao plano, d fica igual a zero.

Este algoritmo foi implementado em conjunto com uma rotina gráfica, que possibilita a visualização de um tronco de cone (representando a ferramenta), um plano (representando o órgão), uma esfera (representando o ponto fixo de entrada no corpo do paciente) e uma esfera que aparece quando ocorre o contato.

No módulo gráfico do simulador, foram obtidos avanços na forma do cálculo do choque entre a ferramenta e o órgão a ser operado. As rotinas foram generalizadas, de forma que a detecção de choque passasse a ser feita independente da localização (quadrante) do ponto fixo, e independente da direção e o sentido da mesma.

Nas primeiras versões do programa, o simulador não calculava adequadamente as transformações de rotação da ferramenta, o que fazia com que ela não se movimentasse de acordo com a vontade do operador. As rotinas também não tratavam adequadamente as posições singulares (como por exemplo as posições de coordenadas igual a zero em x, y ou z).

Isto foi resolvido com uma rotina de cálculo de rotações que contemple todas as condições e singularidades propostas. Através de uma seqüência de comparações o simulador consegue definir qual o ângulo de rotação da ferramenta nas direções y e z para obter a nova posição da mesma no espaço. As figuras a seguir ilustram o funcionamento da rotina de detecção de colisão para um quadrante específico.

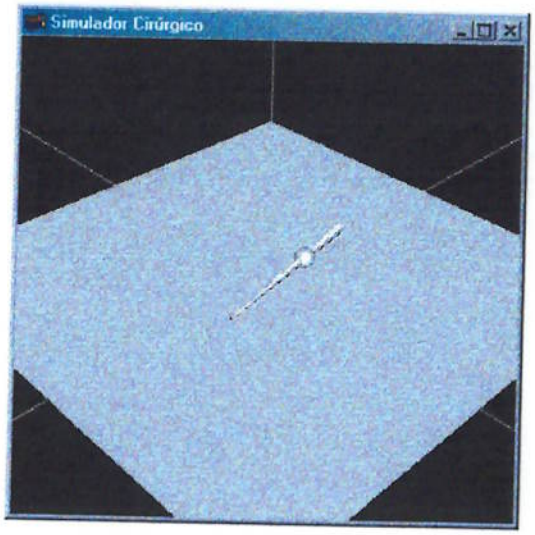

Figura 30 – Ferramenta e órgão - antes do choque

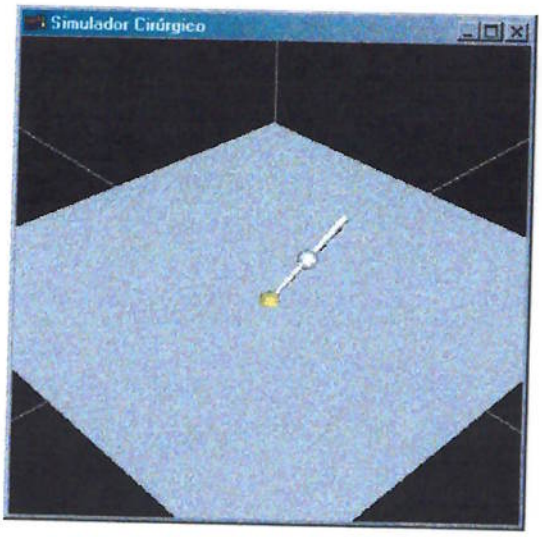

Figura 31 – Ferramenta e órgão – após o choque

Após a implementação desta rotina, torna-se bastante simples sua interface com o módulo Device Driver (Módulo de captura de posição no mouse espacial) pois a entrada do módulo gráfico é a posição da extremidade de manipulação da ferramenta.

5.4.2 CÁLCULO DAS DEFORMAÇÕES

Foi definido que o cálculo das deformações seria feito de forma "batch" através de um pré-processamento em Ansys. Esta solução foi escolhida pois apresenta um resultado mais próximo ao real, além de economizar capacidade de processamento, pois não há a necessidade de se calcular os efeitos da malha inteira a cada loop do processo.

Foi desenvolvido um algoritmo que permite dividir uma superfície de quatro lados em uma malha, e localizar em que divisão ocorre o choque.

Esse algoritmo utiliza geometria analítica para determinar se o ponto pertence a uma determinada parte da malha.

Inicialmente, calcula-se o perímetro de uma divisão da malha.

Em seguida, determinam-se os quatro vetores, que unem os vértices da divisão da malha com o ponto de contato.

Calculam-se as projeções desses vetores nas laterais da divisão da malha, se a soma dessas projeções for maior do que o perímetro da divisão da malha, o ponto não pertence a esta divisão, caso contrário pertence.

$$Perimetro = 2 \times \sqrt{\vec{V} \bullet \vec{V}} + 2 \times \sqrt{\vec{H} \bullet \vec{H}}$$

$$\overrightarrow{Vetor1} = Ponto\_de\_Contato - Vertice1$$

$$\overrightarrow{Vetor2} = Ponto\_de\_Contato - Vertice2$$

$$\overrightarrow{Vetor3} = Ponto\_de\_Contato - Vertice3$$

$$\overrightarrow{Vetor4} = Ponto\_de\_Contato - Vertice4$$

$$\text{Prod}\_escalar(Vertical) = \vec{V} \bullet \overrightarrow{Vetor1}$$

$$\text{Prod}\_escalar(Horizontal) = \vec{H} \bullet \overrightarrow{Vetor1}$$

$$\text{Se } \sum \text{Prod}\_escalar \leq Perimetro$$

$$Pomto\_de\_Contato \in (Vertice1, Vertice2, Vertice3, Vertice4)$$

Caso o ponto não pertença a esta divisão da malha, o processo se repete para a próxima divisão.

5.4.3 MÓDULO DE AQUISIÇÃO

Este módulo realiza interface entre o mouse espacial e o computador, sendo responsável pelo controle da placa de aquisição e o tratamento dos dados obtidos. Seu desenvolvimento foi baseado no trabalho de iniciação científica do aluno Erick

O módulo foi dividido em 5 funções, cada uma responsável por uma atividade da aquisição de dados.

As funções são as seguintes:

Leitor: a função básica do módulo, responsável pela ativação via software da conversão A/D de um canal determinado, e pelo armazenamento do valor convertido em uma variável.

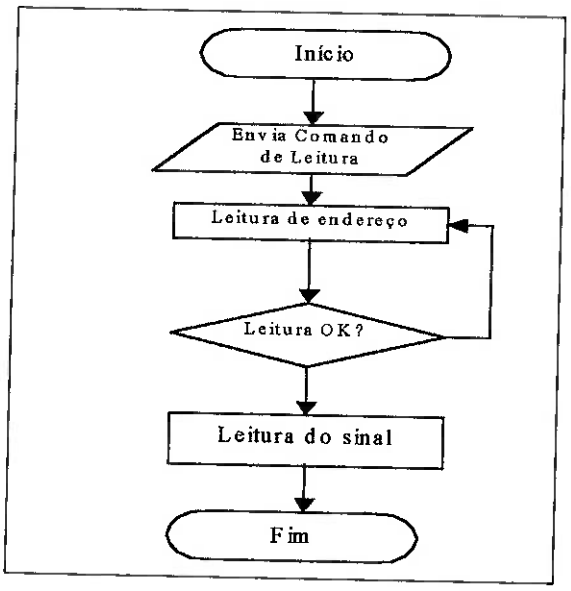

Figura 32 – Fluxograma da função Leitor

Leia: função responsável pela coleta dos dados dos três canais utilizados. Utiliza a função leitor como uma função interna.

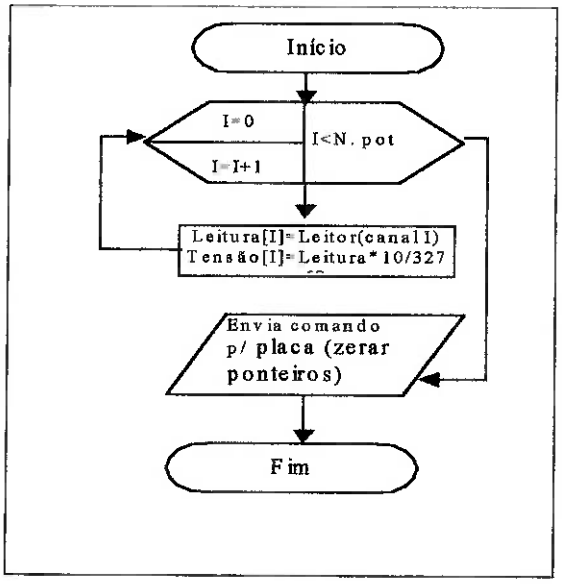

Figura 33 – Fluxograma da função Leia

Inicializa: função responsável pela inicialização da placa, nela estão contidas as palavras de comando que determinam o modo de operação da placa.

- A função realiza os seguintes passos:
- Programar a placa em modo zero
- Programar a memória dos canais
- Auto-calibrar o conversor A/D
- Esvaziar a FIFO
- Programar a placa no modo desejado (nesse caso ativação da conversão via software, ou "polling")

Calibra: função responsável pela calibragem do mouse espacial. Colocando o mouse em duas posições pré determinadas, a função armazena a voltagem medida pelos potenciômetros em cada posição, determinando a relação entre voltagem e ângulo medido.

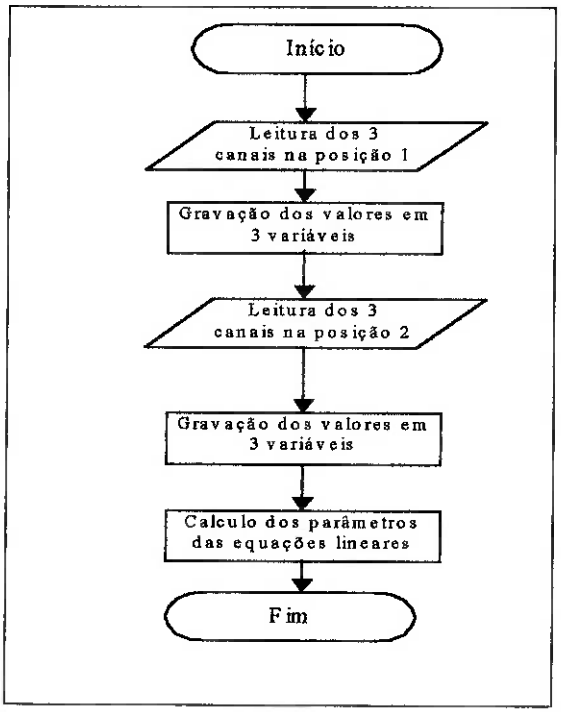

Figura 34 - Fluxograma da função Calibra

SpinDisplay: função responsável pela interpretação dos sinais obtidos, ela calcula a posição da ponta do manipulador em coordenadas cartesianas, a partir dos sinais de ângulos medidos. Ela também é responsável pela filtragem dos sinais, ao estabelecer uma média dos sinais aquisitados.

5.4.4 MÓDULO MATEMÁTICO

Módulo composto por algumas funções matemáticas, que são usadas como suporte pelos outros módulos.

Suas funções são as seguintes:

- Arredondamento: função que trunca o valor dos sinais medidos, é utilizada pela SpinDisplay para ajudar na filtragem dos sinais.
- Produto_escalar: função que calcula o produto escalar entre dois vetores.
- Rotação: função que calcula os ângulos de rotação para que o sistema de coordenadas se alinhe com a direção da ferramenta. Esta função é utilizada pelo módulo gráfico.

Os fluxogramas não serão mostrados, por se tratar de algoritmos matemáticos conhecidos.

5.4.5 MÓDULO DE SIMULAÇÃO

Módulo responsável pela simulação computacional, ele calcula as interações físicas entre os objetos simulados, calculando os parâmetros utilizados pelo módulo gráfico. Este módulo também é responsável pelo controle dos motores do mouse espacial.

Ele é composto pelas seguintes funções:

Choque2: é a função principal deste módulo, ele é responsável pela identificação da ocorrência do choque entre a ferramenta e o plano escolhido e a penetração da mesma. Todas as outras funções deste módulo são suas sub-rotinas.

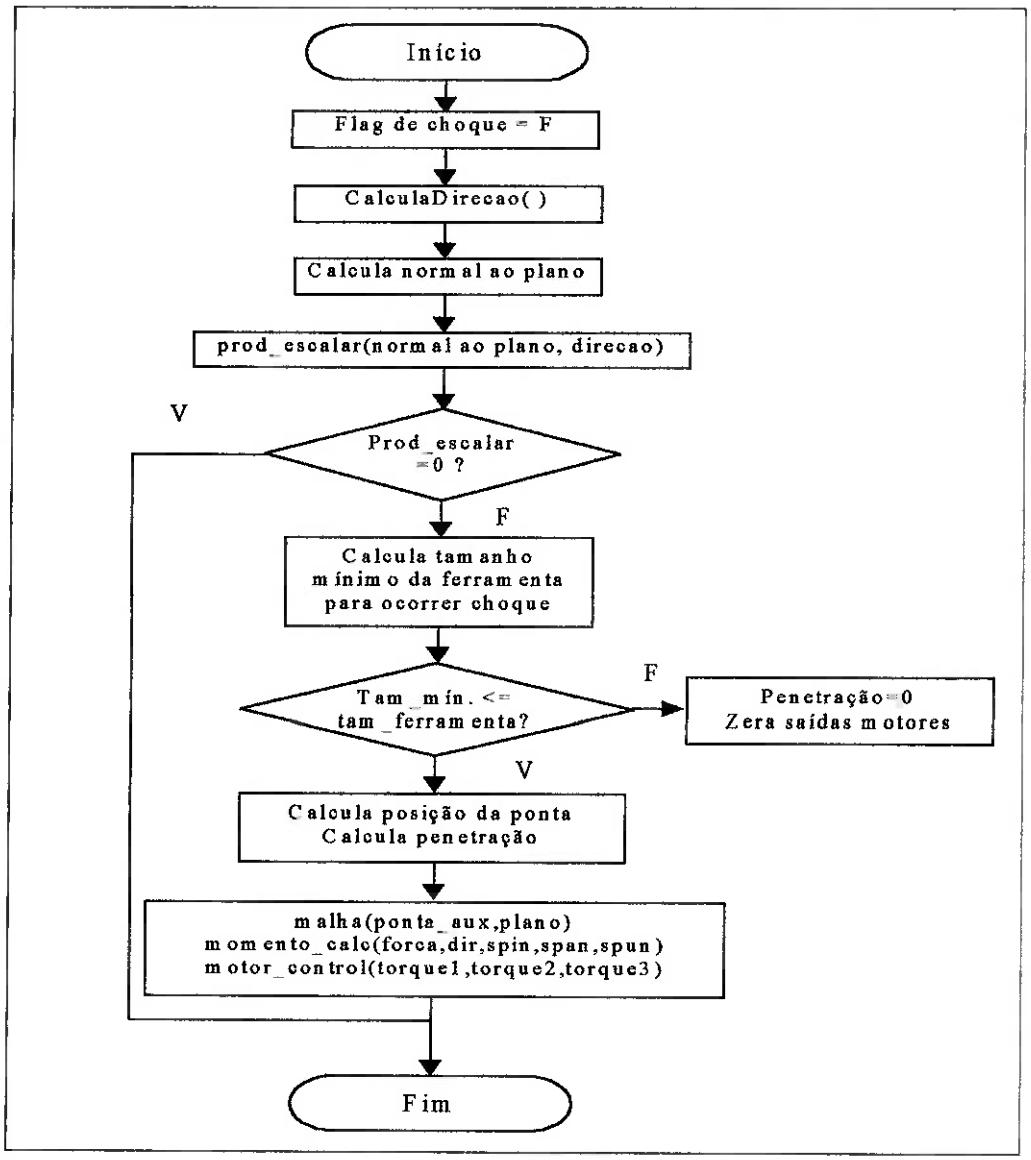

Figura 35 – Fluxograma da função Choque2

CalculaDireção: função responsável pela representação da ferramenta, ela calcula a direção que a ferramenta virtual está apontando.

Malha: sub-rotina da função choque2, é responsável pela verificação da ocorrência do choque entre a ferramenta e uma determinada área do plano. Ela identifica o elemento no qual ocorre o choque e sua respectiva constante de elasticidade. A partir desses dados e da penetração calculada, essa função calcula a força aplicada sobre a superfície.

Momento_calc: outra sub-rotina da função choque2, ela calcula o torque aplicado pelos motores, a partir do modelo dinâmico.

Motor_control: função calcula a voltagem necessária para gerar o torque nos motores, além de enviar o sinal de controle

5.4.6 MÓDULO GRÁFICO

Deformação de superfícies via "Splines" ou "Curvas de Bèziér"

Esta solução propõe a renderização de uma superfície de Bézier a partir de "pontos de controle", que são pontos que não necessariamente pertencem à superfície, mas definem a curvatura da mesma. O OpenGL oferece amplo suporte para a modelagem de curvas e superfícies de Bèzier, privando o usuário até da matemática envolvida na definição dos mesmos. Para uma curva simples, por exemplo, definem-se três pontos para uma curva de Bèziér de segundo grau e quatro pontos para uma curva de Bèziér de terceiro grau, estes dois sendo os mais comumente utilizados em computação gráfica. Esta é uma vantagem do ponto de vista computacional, já que não é necessário o processamento e armazenamento da posição de todos os pontos da superfície em cada instante de tempo, que demandaria um poder computacional bastante elevado, porém esta característica mostra-se como uma desvantagem, por ser necessário definir posições de pontos em lugares não pertencentes à curva ou superfície para fazer com que a superfície passe por um ponto específico no espaço.

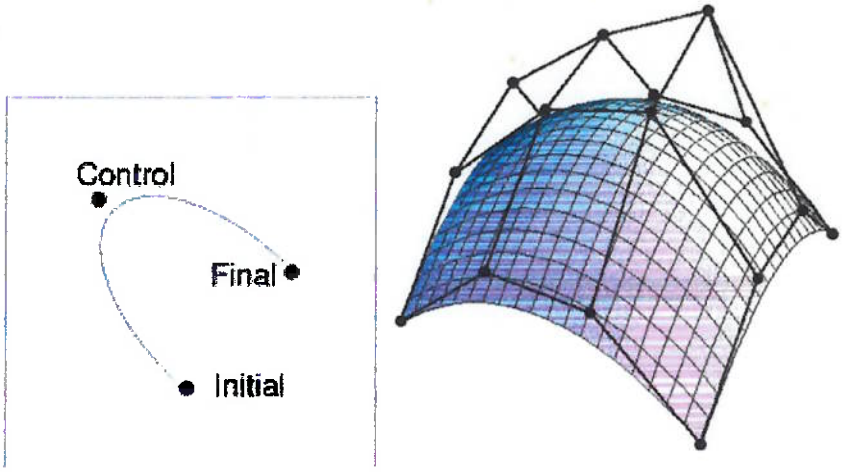

Figura 36 – Curva e superfície de Bèziér e seus pontos de controle

Manipulação direta de malhas poligonais

Por mais variadas que sejam as técnicas de modelamento geométrico, seja por funções quadráticas, paramétricas (splines) ou outras, elas usam todas saídas na forma de pontos formando polígonos e malhas de polígonos.

Pode-se, por exemplo, renderizar rapidamente malhas criando estruturas de dados para estes pontos, utilizando algoritmos que podem ser implementados facilmente a partir destes dados. É o caso da digitalização de um objeto físico através de equipamentos de captura de coordenadas. A malha gerada representa exatamente o dado, limitada pela precisão do medidor que foi utilizado.

A manipulação direta destes pontos é a solução para problemas de modificação da forma de superfícies em pontos determinados. Pode-se tratar cada vértice da estrutura independentemente, e incorporar funções de modificação para estes conjuntos de pontos, de forma que se consiga manipular estes dados de maneira controlável e flexível.

A seguir demonstra-se através de um programa simples em OpenGL como isto ocorre.

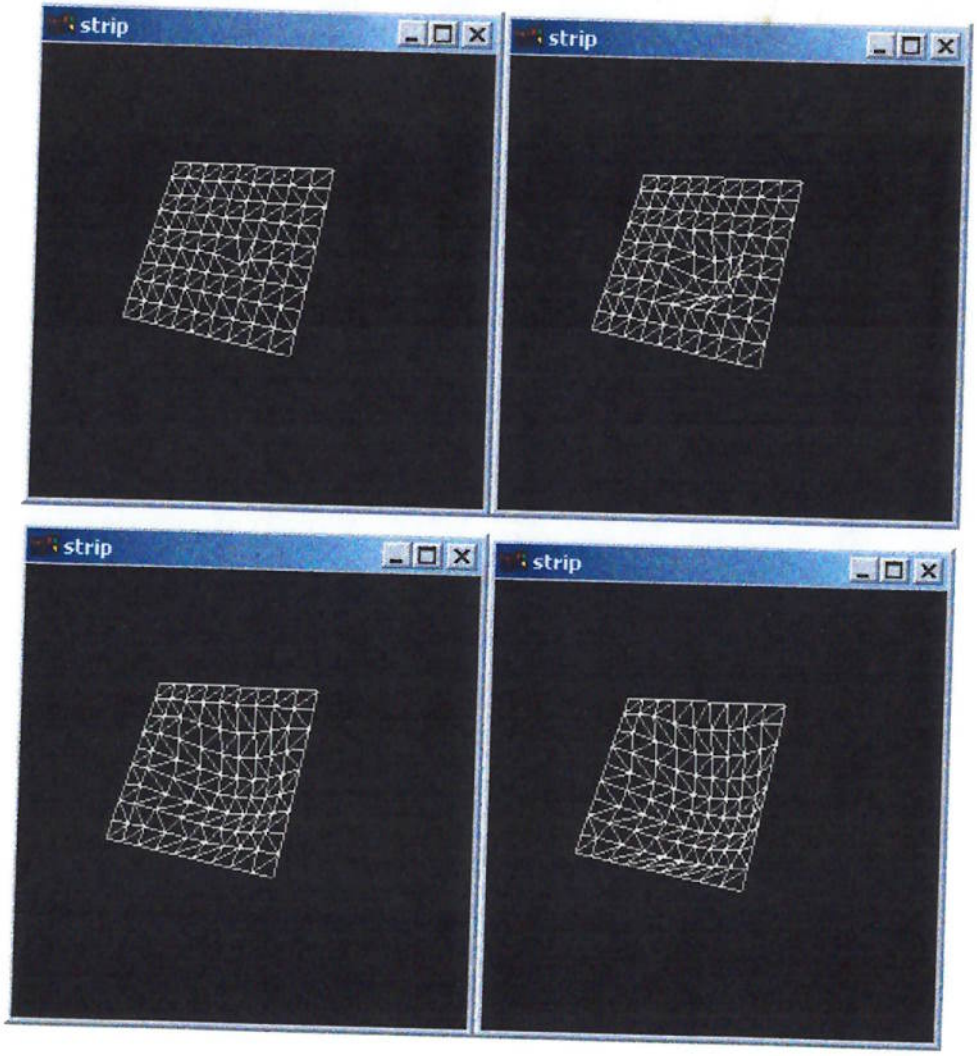

Figura 37 – Deformação da malha através de deformação direta

6 RESULTADOS

A seguir apresentam-se algumas imagens do simulador desenvolvido. Na tela são exibidas as coordenadas da origem da ferramenta e as coordenadas do ponto fixo.

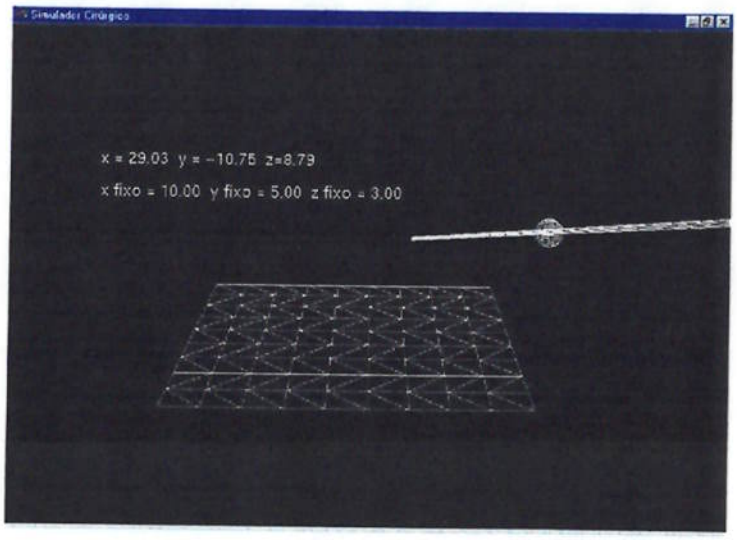

Figura 38 – Simulador Cirúrgico – Vista Inicial

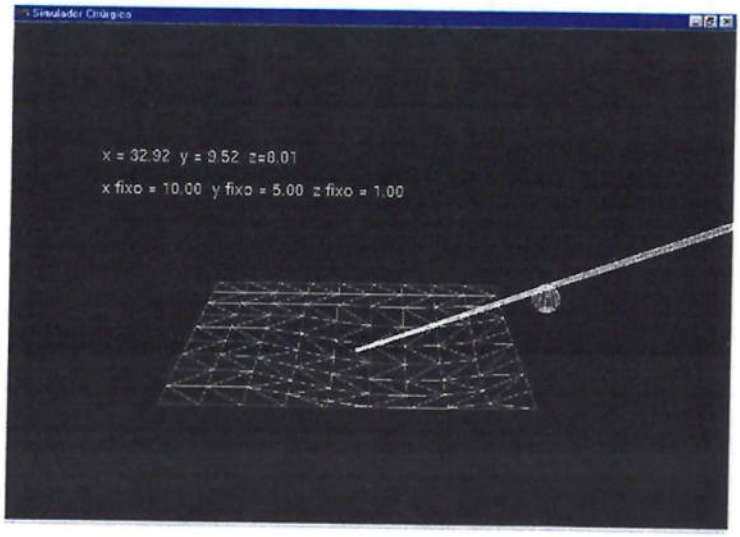

Figura 39 – Deformação do órgão pelo contato com a ferramenta

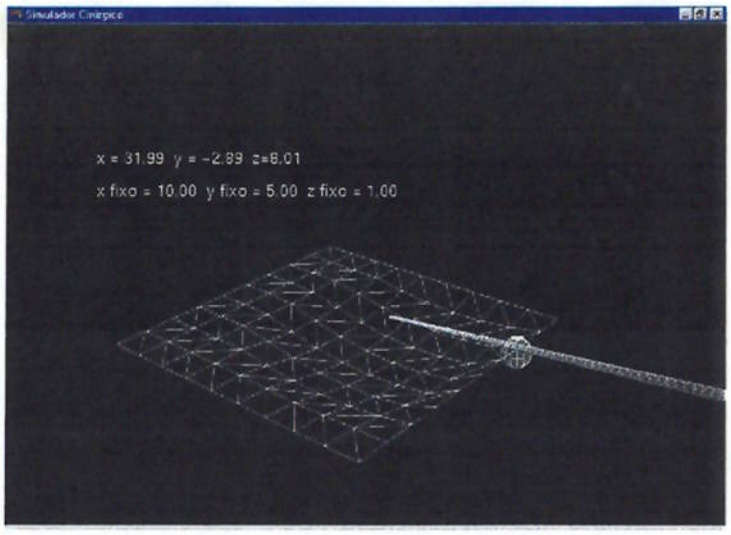

Figura 40 – Rotação de câmera

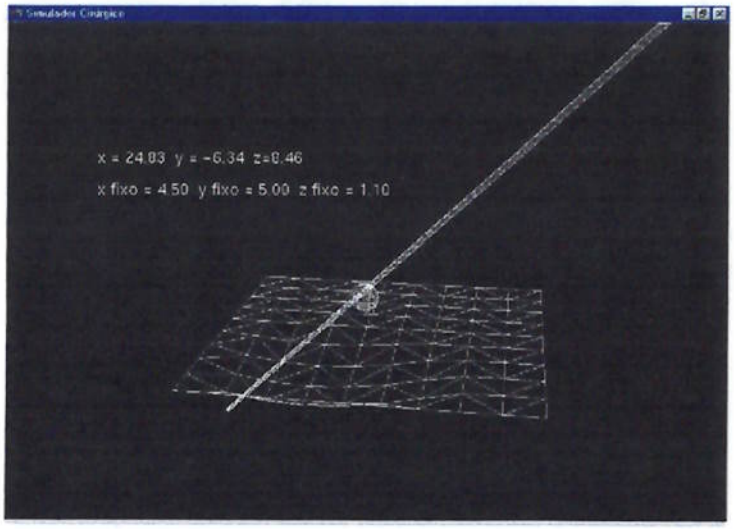

Figura 41 – Rotação de câmera

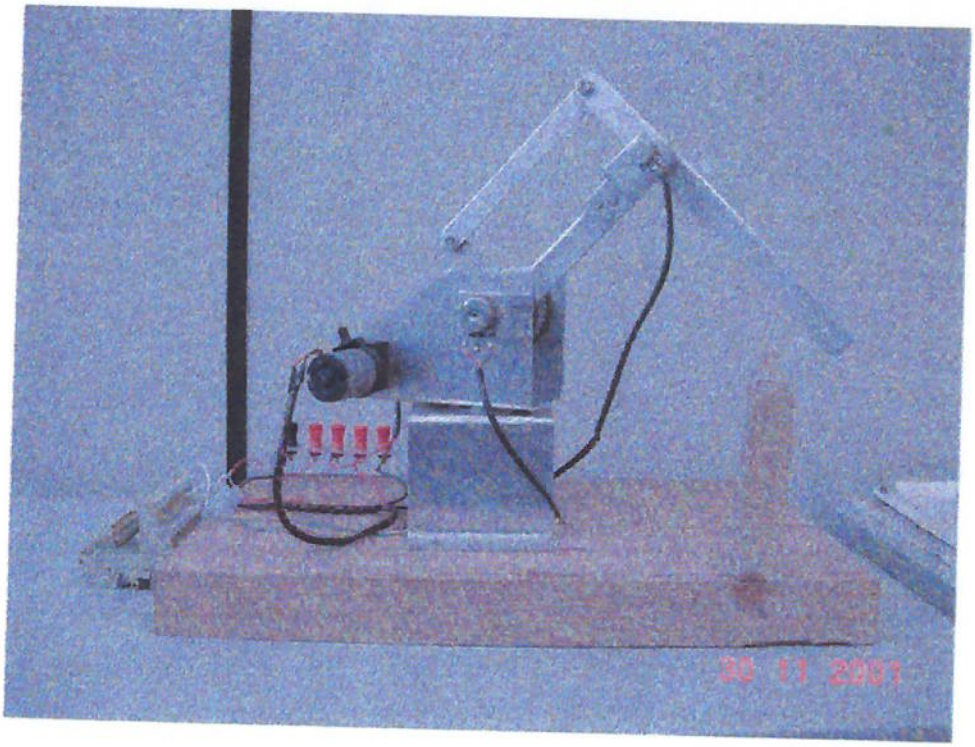

Figura 42 - Mouse com retorno de forças

7 CONCLUSÕES

Com a integração de todos os componentes descritos nos capítulos anteriores, obteve-se o simulador gráfico de cirurgias minimamente invasivas com uma interface de retorno de forças sobre o mouse do departamento.

É importante salientar que as funções geradas são modularizadas, o que permite a sua adaptação a outros tipos de dispositivos de entrada e saída, bastando apenas alterar alguns parâmetros dos seus módulos. Os módulos matemáticos podem ser totalmente mantidos.

O módulo de detecção de choques apenas contempla colisões de cilindros com planos. Para uma aplicação voltada à simulação de colisão com órgãos virtuais, é necessário desenvolver rotinas de colisão com outros tipos de sólidos, por exemplo outros cilindros, esferas, além de outras primitivas gráficas.

Como era esperado, o sistema não mostrou sensibilidade suficiente para transmitir os torques de forma gradual e suave. Isso decorreu por diversos fatores:

- Os acionamentos consistiram de motores DC e redutores acoplados de baixa qualidade. Sua inércia é muito grande, o que limita a sensibilidade de saída do mesmo em baixas voltagens.
- O mouse espacial possui um desalinhamento de montagem do eixo das engrenagens dos braços, o que fez com que a correia do motor 2 ficasse com folga excessiva e a correia do motor 1 ficasse excessivamente tensionada. Como resultado, o motor 1 só é acionado quando se aplicam altas voltagens na sua entrada e o motor 2 gira em falso até conseguir tensionar suficientemente a correia e transmitir o movimento.
- As correias de transmissão foram mal dimensionadas, o que requer força adicional para dobrá-las, o que gera ainda mais inércia no sistema.
- As articulações apresentam folga excessiva, o que prejudica a medição dos ângulos pelos potenciômetros.

Outro problema detectado no dispositivo de retorno de forças foi no acionamento do motor DC. Mesmo quando o sinal de controle era zero, a carga do

motor apresenta uma voltagem de 0,5 V, o que acaba aumentando ainda mais a resistência do sistema ao movimento.

Os potenciômetros foram uma solução prática e barata de capturar a posição do mouse espacial. Uma conseqüência do uso deles foi a falta de precisão nas medidas e alguns saltos de tensão de saída nos transitórios que provavelmente decorrem do atrito com a bobina interna. Para estabilizar o sinal foi preciso implementar soluções que incluíram o cálculo da média de diversas capturas de posição e o truncamento de valores medidos na segunda casa decimal. O uso de encoders ópticos eliminaria este problema e tornaria o simulador muito mais preciso, uma vez que desta maneira pode-se eliminar o cálculo da média, que faz com que a freqüência de amostragem caia nesta razão. O problema desta implementação seria o custo, bastante elevado.

O encoder a ser utilizado deve ser do tipo absoluto, já que o sistema não faz o acompanhamento do sinal de entrada dos sensores de forma contínua. O risco de perder passos nestas condições é elevada, o que impossibilita o uso de encoders incrementais.

Outro problema relacionado aos ruídos detectados está diretamente relacionado com a fonte de tensão. Por ser utilizada apenas uma fonte para alimentar os potenciômetros, o driver de potência e os motores, este compartilhamento da fonte para os circuitos de controle e de potência acaba gerando oscilações de tensão nos sinais de leitura. Isto foi detectado ligando-se e desligando-se o circuito de potência durante a execução do simulador. A solução ideal para o sistema seria utilizar circuitos de potência separados, como por exemplo através de um controle PWM dos acionamentos, e chaveamento do driver através de relês ópticos, o que separa claramente o sinal de controle dos valores de potência, o que elimina estes ruídos.

O computador utilizado para desenvolver o trabalho não foi o ideal para fazer um bom uso dos recursos gráficos oferecidos pelo OpenGL e não ajudou a minimizar os saltos de medição. O uso de um computador com maior poder de processamento, aliado ao uso de uma placa aceleradora de vídeo possibilitaria a implementação de recursos adicionais, como texturas reais de pele, um detalhamento melhor da ferramenta, a renderização do ambiente virtual em tela total, o que melhoraria consideravelmente a qualidade da apresentação do trabalho, uma vez que esta saída é efetivamente o que o usuário observa quando da execução do programa. Velocidades maiores de processamento também possibilitariam uma maior freqüência de amostragem pela placa

de aquisição, o que melhorariam a precisão de leitura e conseqüentemente a saída de tensão para os motores poderia ser melhor controlada.

No primeiro semestre de 2001 foram requisitadas as peças para a montagem do acionamento da base. Como elas foram só foram entregues pelo Sr. Walter de Britto no dia 22/11/2001 (nove dias antes da apresentação), sua utilização ficou comprometida devido ao prazo para a entrega dos resultados.

Em relação ao software, o controle dos acionamentos está pronto, faltando apenas a implementação física do terceiro motor.

8 MANUAL DO USUÁRIO

A seguir apresenta-se o manual do usuário do simulador.

8.1 INSTALAÇÃO DO HARDWARE

O hardware do simulador compõe-se de:

- Mouse espacial;
- 2 (duas) fontes de alimentação:
  - A: Terra e +5V DC
  - B: -15V e +15V
- Driver de potência para os motores;
- Placa de aquisição CAD 12/36 da Lynx Tecnologia.

Deve-se alimentar os potenciômetros com a fonte A e o driver de potência com a fonte B.

Conectar os motores nas saídas do circuito de potência (Out1 para o motor 1 e Out 2 para o motor 2). Ligar o terra do circuito de potência ao terra da fonte.

Conectar a placa de aquisição com o equipamento através do cabo com conector DB-37disponível da seguinte maneira:

Pino 8: Sinal de entrada do potenciômetro 1 (Verde)

Pino 9: Sinal de entrada do potenciômetro 2 (Amarelo)

Pino 11: Sinal de entrada do potenciômetro 3 (Roxo)

Pino 24: Terra (Referência da placa de aquisição)

Pino 23: Sinal de controle do motor 1 (conectar na entrada In 1 do circuito de potência)

Pino 5: Sinal de controle do motor 2 (conectar na entrada In 2 do circuito de potência)

8.2 INSTALAÇÃO DO SOFTWARE

No micro com a placa de aquisição instalada, a biblioteca GLUT32.DLL deve estar instalada no diretório \Windows\System

8.3 EXECUÇÃO DO PROGRAMA

Ao iniciar o programa, ele irá exibir uma tela pedindo para posicionar o mouse na posição 1.

- Posicione o mouse com a ponta encostada na marca direita sobre a base, de maneira que a haste da base esteja paralela ao plano da base, a seguir a haste da ponta deve ser levantada ao máximo, mantendo a haste da base paralela ao plano da base.
- Tecle <enter>

O programa irá pedir para colocar o mouse na posição 2:

- Posicione o mouse com a ponta sobre a ponta da haste de madeira a esquerda do mouse.
- Tecle <entre>

O programa será inicializado, com a tela de simulação.

8.4 COMANDOS DO PROGRAMA

O programa possui diversos comandos que mudam a posição da câmera e a posição do ponto fixo, permitindo uma melhor visualização da imagem e uma maior interação com o simulador:

W: Move o ponto fixo no eixo y no sentido positivo

S: Move o ponto fixo no eixo y no sentido negativo

A: Move o ponto fixo no eixo x no sentido negativo

D: Move o ponto fixo no eixo x no sentido positivo

Home: Rotaciona a câmera no eixo x no sentido negativo

End: Rotaciona a câmera no eixo x no sentido positivo

Delete: Rotaciona a câmera no eixo y no sentido negativo

Page Down: Rotaciona a câmera no eixo y no sentido positivo

Insert: Rotaciona a câmera no eixo z no sentido negativo

Page Up: Rotaciona a câmera no eixo z no sentido positivo

F1: Retorna a câmera à posição inicial

Esc: Sai do programa

9 BIBLIOGRAFIA

Vidal Filho, W. B.; Moscato, L.A.; Lima, R. G. - **Development of a "spatial" interface for surgery simulator** – Induscon'98; São Paulo

Buttolo, P.; Oboe, R.; Hannaford, B. **Architectures for shared haptic virtual environments** - Comput. & Graphics, Vol. 21; No. 4; pp. 421-429; Grã-Bretanha;1997

Yagel, R.. et al. **Building a virtual environment for endoscopic sinus surgery simulation** - Comput. & Graphics, Vol. 20; No. 6; pp. 813-823; Grã-Bretanha;1996

Iwata, H.; Yano, H.; Hashimoto, W. **LHX: an integrated software tool for haptic interface** - Comput. & Graphics, Vol. 21; No. 4; pp. 413-420; Grã-Bretanha;1997

SensAble Technologies, Estados Unidos, **Apresentação da descrição de Phantom e seus similares**. Disponível em <http//:www.sensable.com>. Acesso em: 05 de Maio. 2001

Iwata Lab – HapticMaster, Japão, **Apresentação da descrição do Haptic Master**. Disponível em < http://intron.kz.tsukuba.ac.jp/HM/txt.html>. Acesso em 10 de Maio, 2001.

Melco, T. C. **Desenvolvimento de um apontador espacial, 1°. relatório**; São Paulo; 2000

Camargo, E. D. **Desenvolvimento de programa para aquisição de dados e controle - 1° Relatório**; São Paulo; 2000

Lynx Tecnologia Eletrônica Ltda. **Manual do usuário e de Referência do conversor A/D e D/A para microcomputadores CAD12/36**; São Paulo, 1997

Hashimoto, W.; Iwata, H. **A Versatile Software Platform for Visual/Haptic Environment**

Woo, M. et al. **The OpenGL programming guide: the official guide to learning OpenGL, version 1.1** - Addison-Wesley Publishing Company; Estados Unidos; 1999

Wernecke, J. **The Inventor Mentor : Programming Object-Oriented 3D Graphics With Open Inventor, Release 2**; Open Inventor Architecture Group

Camargo, E. D; **Desenvolvimento de programa para aquisição de dados e controle – Relatório Final**; São Paulo; 2001

Moriya, K.; Souza, L. F. A. **Sistema de controle de um mecanismo de posicionamento no plano horizontal.** São Paulo, 1998. 77p. Trabalho de Formatura. Escola Politécnica, Universidade de São Paulo.

National Semiconductor Corporation. **LM12CL 80W Operational Amplifier**, ,Japão,1999.

NeHe Productions (OpenGL). Estados Unidos. **Apresenta informações e discussão sobre computação gráfica.** Disponível em: <nehe.gamedev.net>. Acesso em: 17 de jun. 2001

Allan, J.B.; Wyvill, B.; Witten, I. H. **A METHODOLOGY FOR DIRECT MANIPULATION OF POLYGON MESHES**; Computer Science Technical Reports, Department of Computer Science, The University of Calgary, Alberta, Canada

Dionisio, J. et al. **The virtual touch: Haptic interfaces in virtual environments.** Comput. & Graphics, Vol. 21; No. 4; pp. 459-468; Grã-Bretanha;1997.

Web3D Consortium. Estados Unidos. **Apresentação sobre o VRML.** Disponível em: <www.vrml.org>. Acesso em: 5 de Maio, 2001

Boulos, P.; Camargo, I. **Geometria Analítica – Um Tratamento Vetorial** ; Makron Books, São Paulo, 1987.

Malvino, A. P.; **Eletrônica** – Quarta Edição Vol.II; Makron Books, São Paulo, 1997.

ANEXO 1

Arquivo simulador_final.cpp

```
/*********************************************************************
PMC-592 - Simulador cirúrgico - versão final

 Henrique Miyamoto
 Humberto Nomura Nishizaki

Observações gerais
* As medidas físicas usadas são: cm, N e radianos
* [Matriz_k] = [N]/[m]


*********************************************************************/

/*  //******** As medidas físicas usadas são: cm, N e radianos
  Matriz_k: N/m
*/

//*********** Bibliotecas basicas ************************************

#ifdef _WIN32
#include <windows.h>
#endif
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glut.h>

#include <stdio.h>
#include <math.h>

#include <stdlib.h>
#include <conio.h>
#include "material.h"

//********** Definicoes de Constantes do Programa ********************

#ifndef M_PI                                    // Alguns math.h nao definem o pi=3.14159...
#define M_PI 3.14159265358979323846
#endif
#define RADDEG 57.2957795130823208768463    // Conversao Rad -> Graus
#define DEGRAD 0.01745329251994             // Conversao Graus -> Rad


//********** Definições da Placa *************************************
#define EndBase  0x0380          // Endereco base da placa
// #define t 15                  //
// #define p 15.8                // Cuidado! mesmo nome da variavel do text()
#define pot 3                    // Numero de potenciometros
#define G1    0x70
#define Bip   0x80
#define size_k 10                // Tamanho da matriz de coeficientes de elasticidade ( 10/10/2001)
```

```
#pragma intrinsic (outp(unsigned short port, int databyte));
#pragma intrinsic (inp(unsigned short port));

//************** Variáveis para a aquisição pela placa **************
int val[3],
        i,
        media=20        ,
        marca=0,
        precisao;
float leitura[pot],
         tensao[5],
         constantes[6],
         zera[pot],
         valor[12],
         medant[1],
         tenant[1],
         vant;
//       apont_l2 = 1.5,
//       apont_l3 = 1.75;

//************** Fim variaveis da placa ******************************


//************** Variaveis choque ************************************

GLfloat ponto_fixo[3],                  // Ponto de entrada no corpo do paciente
                origem_raio[3],         // Ponto de origem da ferramenta (mao do medico)
                plano[16],              // Parametros da equacao parametrica do plano + coordenadas
dos vértices (4) ( 10/10)
                dir[3],                 // Vetor direcao da ferramenta
                ponta_aux[3],           // Extremidade da ferramenta
                penetracao,             // Profundidade de penetracao ( 09/10/2001)
                forca,                  // Variavel com o modulo da forca ( 10/10/2001)
                matriz_k[size_k-1][size_k-1],           // Matriz c/ os coeficientes de elasticidade
(Ansys) ( 10/10/2001)
                torque1,torque2,torque3, // torque para cada um dos motores
                ponto_prox[11],         // Variavel que contem as coordenadas do ponto (x,y,z,
delta1,delta2,linha e coluna)
                malha_aux[size_k][size_k*3],            // Malha discretizada
                byte_a1,byte_b1,byte_a2,byte_b2,byte_a3,byte_b3,        // Bytes dos motores (
06/11/2001)
                k_matriz;               // Variável com o k da malha

int choque_f = 0;

float t_min_global;

//int     size_k;                       // Tamanho da matriz k ( 10/10/2001)

//************** Fim variaveis choque ********************************


//************** Variaveis cinematica inversa ************************

// ângulos finais em graus medidos pelos potenciômetros
GLfloat spin,
                span,
                spun;
```

```
//************* Fim variaveis cinematica inversa ********************

//************** Variáveis gráficas *********************************

GLuint  listaFerramenta,  // Valor inteiro que identifica unicamente a lista ferramenta
                listaEsfera,            // Valor inteiro que identifica unicamente a lista Esfera
                listaEsferaFixa,
                listaPele,              // Valor inteiro que identifica unicamente a lista Pele
                listaEixos;             // Valor inteiro que identifica unicamente a lista Eixos

GLfloat tamanho=30,             // Tamanho da ferramenta
                rotz=0,                 // Rotacao em torno de z
                roty=0,                 // Rotacao em torno de y
                coord_int[4];           // Vetor de coordenadas intermediarias

GLUquadricObj *qobj;            // Cria um novo objeto GLUquadric e retorna um apontador para ele
(cap 11)

char s[300];                            // Comprimento do vetor de caracteres da rotina text

// Algumas variaveis da malha do orgao
#define MAX_MESH 10             // Tamanho da malha do orgao
GLfloat mesh[MAX_MESH][MAX_MESH];

float dist_global=0;

float camx=0.0, camy=0.0, camz=0.0;     // Angulos da camera

//**************** Fim variaveis graficas ****************************

float volt1,volt2,volt3;

//************** Definicao das funcoes ******************************

float Leitor(char canal);
void leia();
void Inicializa(void);
void Calibra(void);

static void init(void);
void text(int x, int y, char* s);
void display(void);
void reshape(int w, int h);
void keyboard(unsigned char key, int x, int y);

double prod_escalar(float ref1[1], float ref2[], int tam);
void calculaDirecao(void);
void choque2(float ponto_fixo[], float origem_raio[], float plano[],float tamanho);
void rotacao(float dir[]);
void malha(float pto_contato[],float plano[]);
void anime(void);
void momento_calc(float forca,float dir[],float spin, float span, float spun);
void motor_control(float torque1, float torque2, float torque3);                // 06/11/2001
float arredondamento(float num, int casas);             // 14/11/2001

//*************** Fim definicao das funcoes *************************
```

```
//*************************************************************************
//*************** Funções de leitura da placa ************************
//*************************************************************************

// Funcao Leitura ***********************************************************
float Leitor(char canal)
{
        int a;
        outp (EndBase+4, 2);
        outp (EndBase+5, canal);
        do
        {
                a = inp(EndBase+3);
        } while ((a & 0x0001) == 0);
        a = inp(EndBase+4);
        a = a + (inp(EndBase+5)<<8);

        return (a);
}


// Funcao Leia **************************************************************
void leia()
{
 int I;
 for (I = 0; I<pot ; I++)
  {
   leitura[I] = Leitor(I);
   tensao[I] = 10*leitura[I]/32768;
  }
  outp(EndBase+4,1);   // zerar o ponteiro
  outp(EndBase+5,0);   // dos canais.
}

// Funcao de inicializacao da placa de aquisicao ********************
void Inicializa(void)
{
        int aux, canal;

        // rotina ModoZero
        outp(EndBase+4, 3);        // poe no byte do registrador de
        outp(EndBase+5, 0);        // modo (3) tudo 0.

        // rotina ProgramaMemoria
        outp(EndBase+4, 0);        // registrador de Limite recebe
        outp(EndBase+5, 15);       // maximo de 16 canais. 0 - 15

        for (canal = 0; canal<pot ; canal++)
  {
                outp (EndBase+4, 1);                    // armazena o canal a ser lido (0)
                outp (EndBase+5, canal);          // no ponteiro para memoria(1).
                outp (EndBase+4, 4);                    // armazena o tipo de dado no
                outp (EndBase+5, Bip|G1|canal); // endereco de escrita em memoria.
  }

        // rotina AutoCalibrar
```

```
        outp (EndBase+4, 6);    // realiza a calibracao colocando qualquer
        outp (EndBase+5, 0);    // valor (0) no comando para auto-calibracao.
        Sleep(1000);

        // rotina EsvaziarFifo
        outp (EndBase+4, 5);   // esvazia a Fifo(5) escrevendo
        outp (EndBase+5, 0);   // qualquer valor(0) nela.

        outp(EndBase+4,1);      // zerar o ponteiro
        outp(EndBase+5,0);      // dos canais.

        // realizacao da primeira leitura, esta nao interessa.
        for (canal = 0; canal<pot ; canal++)
        {
                outp (EndBase+4, 2);
                outp (EndBase+5, canal);
                do
                {
                        aux = inp(EndBase+3);
                } while ((aux & 0x0001) == 0);
                Sleep(20);
                aux = inp(EndBase+4);
                aux = aux + (inp(EndBase+5)<<8);
        }
        // fim da funcao
        outp(EndBase+4,1);      // zerar o ponteiro
        outp(EndBase+5,0);      // dos canais.
}

// Funcao Calibracao ************************************************
void Calibra(void)
{
        float teta[pot][2], aux_tensao[pot];

        outp (EndBase+8, 0);
        outp (EndBase+9, 0);
        outp (EndBase+10, 0);
        outp (EndBase+11,0);
        outp (EndBase+12, 0);
        outp (EndBase+13, 0);
        teta[0][0]=30*DEGRAD;
        teta[0][1]=90*DEGRAD;
        teta[1][0]=0*DEGRAD;
        teta[1][1]=40*DEGRAD;
        teta[2][0]=50*DEGRAD;
        teta[2][1]=130*DEGRAD;
        printf("Coloque o apontador na posicao 1!\n\n");
        getch();
        leia();
        aux_tensao[0] = tensao[0];
        aux_tensao[1] = tensao[1];
        aux_tensao[2] = tensao[2];
        printf("Coloque o apontador na posicao 2!");
        getch();
        leia();

        constantes[0] = (teta[0][1] - teta[0][0])/(tensao[0] - aux_tensao[0]);
        constantes[1] = teta[0][0] - aux_tensao[0]*constantes[0];
```

```
        constantes[2] = (teta[1][1] - teta[1][0])/(tensao[1] - aux_tensao[1]);
        constantes[3] = teta[1][0] - aux_tensao[1]*constantes[2];
        constantes[4] = (teta[2][1] - teta[2][0])/(tensao[2} - aux_tensao[2]);
        constantes[5] = teta[2][0] - aux_tensao[2]*constantes[4];

}

//*****************************************************************
//********** Funcoes graficas ***************************************
//*****************************************************************

// Funcao de inicializacao grafica **********************
static void init(void)
{
        glClearColor(0.0, 0.0, 0.0, 0.0);               // cor de fundo negro
        glShadeModel(GL_FLAT);                          // Seleciona entre sombra simples (GL_FLAT) ou
suavizada (GL_SMOOTH)
        glClearDepth(1.0);                  // Especifica o valor do Depth Buffer quando este e limpo
        glEnable(GL_DEPTH_TEST);            // Autoriza o teste de depth, utilizando o depth buffer
        glDepthFunc(GL_LESS);               // Criterio de comparacao do depth test para desenhar o pixel
num dado z

        glLightfv(GL_LIGHT0, GL_AMBIENT, LightAmbient);     // Especifica as config. de luz
ambiente (p/ LIGHT1)
        glLightfv(GL_LIGHT0, GL_DIFFUSE, LightDiffuse);     // Especifica a cor da luz
        glLightfv(GL_LIGHT0, GL_SPECULAR, LightSpecular);   // Especifica a cor de refracao da
luz
        glLightfv(GL_LIGHT0, GL_POSITION,LightPosition0);   // Especifica a posicao da luz

        glLightfv(GL_LIGHT1, GL_AMBIENT, LightAmbient);     // Especifica as config. de luz
ambiente (p/ LIGHT1)
        glLightfv(GL_LIGHT1, GL_DIFFUSE, LightDiffuse);     // Especifica a cor da luz
        glLightfv(GL_LIGHT1, GL_SPECULAR, LightSpecular);   // Especifica a cor de refracao da
luz
        glLightfv(GL_LIGHT1, GL_POSITION,LightPosition1);   // Especifica a posicao da luz
        glLightf(GL_LIGHT1, GL_SPOT_CUTOFF, 45.0);
        glLightfv(GL_LIGHT1, GL_SPOT_DIRECTION, spot_direction);

  glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);      // Configuracao do brilho do
material sob a luz

        qobj = gluNewQuadric();

        listaFerramenta = glGenLists (1);  // GERACAO DE LISTA PARA FERRAMENTA
                glNewList(listaFerramenta, GL_COMPILE);
                glMaterialfv(GL_FRONT, GL_AMBIENT, mat_amb_prata);
                glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_dif_prata);
                glMaterialfv(GL_FRONT, GL_SPECULAR, mat_spc_prata);
                glMaterialfv(GL_FRONT, GL_EMISSION, mat_emission_escuro);
                glPushMatrix();
                        gluCylinder(qobj, 0.2, 0.05, tamanho, 15, 1);
                glPopMatrix();
        glEndList();

        listaEsfera = glGenLists (1);               // GERACAO DE LISTA PARA ESFERA
                glNewList(listaEsfera, GL_COMPILE);
                glMaterialfv(GL_FRONT, GL_AMBIENT, mat_amb_amarelo);
                glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_dif_amarelo);
```

```
            glMaterialfv(GL_FRONT, GL_SPECULAR, mat_spc_amarelo);
            glMaterialfv(GL_FRONT, GL_EMISSION, mat_emission_escuro);
            glPushMatrix();
                gluSphere(qobj, 0.4, 15, 15);
            glPopMatrix();
        glEndList();

        listaEsferaFixa = glGenLists (1);          // GERACAO DE LISTA PARA ESFERA
            glNewList(listaEsferaFixa,GL_COMPILE);
            glMaterialfv(GL_FRONT, GL_AMBIENT, mat_amb_prata);
            glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_dif_prata);
            glMaterialfv(GL_FRONT, GL_SPECULAR, mat_spc_prata);
            glMaterialfv(GL_FRONT, GL_EMISSION, mat_emission_escuro);
            glPushMatrix();
                gluSphere(qobj, 0.4, 15, 15);
            glPopMatrix();
        glEndList();

        listaPele = glGenLists(1);
            glNewList(listaPele, GL_COMPILE);
            glMaterialfv(GL_FRONT, GL_AMBIENT, mat_amb_vermelho);
            glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_dif_vermelho);
            glMaterialfv(GL_FRONT, GL_SPECULAR, mat_spc_vermelho);
            glMaterialfv(GL_FRONT, GL_EMISSION, mat_emission_claro);
            glPushMatrix();
                glBegin(GL_POLYGON);
                    glVertex3f (plano[4],plano[5],plano[6]);
                    glVertex3f (plano[7],plano[8],plano[9]);
                    glVertex3f (plano[10],plano[11],plano[12]);
                    glVertex3f (plano[13],plano[14],plano[15]);
                glEnd();
            glPopMatrix();
        glEndList();

        listaEixos = glGenLists(1);
            glNewList(listaEixos, GL_COMPILE);
            glMaterialfv(GL_FRONT, GL_EMISSION, mat_emission_claro);
            glPushMatrix();
            glBegin (GL_LINES);
                glVertex3f (-20.0, 0.0, 0.0);
                glVertex3f (20.0, 0.0, 0.0);
                glVertex3f (0.0, 20.0, 0.0);
                glVertex3f (0.0, -20.0, 0.0);
                glVertex3f (0.0, 0.0, -20.0);
                glVertex3f (0.0, 0.0, 20.0);
          glEnd ();
            glPopMatrix();
            glMaterialfv(GL_FRONT, GL_EMISSION, mat_emission_escuro);
        glEndList();

        glEnable(GL_LIGHTING);
        glEnable(GL_LIGHT0);
        glEnable(GL_LIGHT1);
}

// Funcao de escrita de texto ***************************************
void text(int x, int y, char* s)
{
```

```
        int lines;
        char* p;

        glDisable(GL_DEPTH_TEST);
        glMatrixMode(GL_PROJECTION);
  glPushMatrix();
                glLoadIdentity();
                glMaterialfv(GL_FRONT, GL_EMISSION, mat_emission_claro);
                glOrtho(0,                glutGet(GLUT_WINDOW_WIDTH),                0,
glutGet(GLUT_WINDOW_HEIGHT), -1, 1);
                glMatrixMode(GL_MODELVIEW);
                glPushMatrix();
                        glLoadIdentity();
                        glRasterPos2i(x, y);
                        for(p = s, lines = 0; *p; p++)
                        {
                                if (*p == '\n')
                                {
                                lines++;
                                glRasterPos2i(x, y-(lines*18));
                                }
                                glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, *p);
                        }
                        glMatrixMode(GL_PROJECTION);
                glPopMatrix();
                glMatrixMode(GL_MODELVIEW);
        glPopMatrix();
        glEnable(GL_DEPTH_TEST);
}

// Funcao desenha_orgao ***************************************
void desenhaOrgao(void)
{
  int i, k, swap = 0;
        float dist_ponto_colisao, raio;

        glPolygonMode(GL_FRONT, GL_LINE);
        glPolygonMode(GL_BACK, GL_LINE);

        raio=fabs(2*penetracao);
        glMaterialfv(GL_FRONT, GL_EMISSION, mat_emission_claro);

  for (k = 1; k < MAX_MESH-1; k++) {
        for (i = 1; i < MAX_MESH-1; i++) {
                if(choque_f==1)
                {
                        dist_ponto_colisao=sqrt((ponto_prox[9]-k)*(ponto_prox[9]-
k)+(ponto_prox[10]-i)*(ponto_prox[10]-i));
                        dist_global = dist_ponto_colisao;
                }
                else
                {
                        dist_ponto_colisao = 0;
                        raio=1;
                }
                if( dist_ponto_colisao <= raio)
                {
                        mesh[k][i]=penetracao*sin((1-dist_ponto_colisao/raio)*M_PI/2)/3;
```

```
                }
                else
                        mesh[k][i] = 0.0;
        }
    }
    glPushMatrix();
                glBegin(GL_TRIANGLE_STRIP);
                for (k = 0; k < MAX_MESH-1; k++)
                {
                        if (swap)
                        {
                          for (i = MAX_MESH-1; i >= 0; i--)
                                {
                                        glVertex3f(i, mesh[k][i], k);
                                        glVertex3f(i, mesh[k+1][i], k+1);
                                        if (i == 0)
                                                glVertex3f(i, mesh[k+1][i], k+1);
                                }
                        }
                        else
                        {
                                for (i = 0; i < MAX_MESH; i++)
                                {
                                        glVertex3f(i, mesh[k][i], k);
                                        glVertex3f(i, mesh[k+1][i], k+1);
                                        if (i == MAX_MESH-1)
                                                glVertex3f(i, mesh[k+1][i], k+1);
                        }
                        }
                swap ^= 1;
        }
    glEnd();
    glPopMatrix();
}

// Funcao display - atualizado a cada loop **************************
void display(void)
{
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glPushMatrix();
                glTranslatef(5.0,5.0,0.0);
                glRotatef(camx,1.0,0.0,0.0);
                glRotatef(camy,0.0,1.0,0.0);
                glRotatef(camz,0.0,0.0,1.0);
                glTranslatef(-5.0,-5.0,0.0);
                glPushMatrix();
                        glRotatef(-90.0,1.0,0.0,0.0);
                        desenhaOrgao();
                glPopMatrix();
                glPushMatrix();
                        glTranslatef(ponto_fixo[0], ponto_fixo[1], ponto_fixo[2]);
                        glCallList(listaEsferaFixa);
                glPopMatrix();
                glPushMatrix();
                        glTranslatef(origem_raio[0],origem_raio[1],origem_raio[2]);
                        glRotatef(rotz,0.0,0.0,1.0);
                        glRotatef(roty,0.0,1.0,0.0);
                        glCallList(listaFerramenta);
```

```
                        sprintf(s,"x = %.2f  y = %.2f  z=%.2f\n\nx fixo = %.2f  y fixo = %.2f  z fixo =
%.2f",origem_raio[0],origem_raio[1],origem_raio[2],ponto_fixo[0],ponto_fixo[1],ponto_fixo[2]);
                        text(100,400,s);
                glPopMatrix();
        glPopMatrix();

        glFlush();
        glutSwapBuffers();
}

// Funcao reshape - executado no redimensionamento da janela ********
void reshape(int w, int h)
{
  glViewport(0, 0, (GLsizei) w, (GLsizei) h);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluPerspective(30, (GLfloat) w/(GLfloat) h, 1.0, 100.0);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
        gluLookAt(5, -20, 10, 5, 10, 0, 0, 1, 0);
}


// Funcao keyboard - Captura teclado ********************************
void
keyboard(unsigned char key, int x, int y)
{
   switch (key) {
                case 'a':
                        ponto_fixo[0]=ponto_fixo[0]-0.1;
                        break;
                case 'd':
                        ponto_fixo[0]=ponto_fixo[0]+0.1;
                        break;
                case 'w':
                        ponto_fixo[1]=ponto_fixo[1]+0.1;
                        break;
                case 's':
                        ponto_fixo[1]=ponto_fixo[1]-0.1;
                        break;
                case '+':
                        ponto_fixo[2]=ponto_fixo[2]+0.1;
                        break;
                case '-':
                        ponto_fixo[2]=ponto_fixo[2]-0.1;
                        break;
                case 27:
                        byte_a1=0;
                        byte_b1=0;
                        byte_a2=0;
                        byte_b2=0;
                        byte_a3=0;
                        byte_b3=0;
                        outp (EndBase+8, byte_a1);
                        outp (EndBase+9, byte_b1);
                        outp (EndBase+10, byte_a2);
                        outp (EndBase+11, byte_b2);
                        outp (EndBase+12, byte_a3);
```

```
                        outp (EndBase+13, byte_b3);
                        exit(0);
                        break;
                case 127:
                        camy = camy-1;
                        break;
        }
        anime();
}

void special_keys(int a_keys, int x, int y)
{
        switch (a_keys) {
                case GLUT_KEY_HOME:
                        camx = camx-1;
                        break;
                case GLUT_KEY_END:
                        camx = camx+1;
                        break;
                case GLUT_KEY_PAGE_DOWN:
                        camy = camy+1;
                        break;
                case GLUT_KEY_PAGE_UP:
                        camz = camz+1;
                        break;
                case GLUT_KEY_INSERT:
                        camz = camz-1;
                        break;
                case GLUT_KEY_F1:
                        camx=0.0;
                        camy=0.0;
                        camz=0.0;
                        break;
                case 127:
                        camy = camy-1;
                        break;
                default:
                break;
        }
        glutPostRedisplay();
}


// Funcao spinDisplay ***********************************************
void spinDisplay(void)
{
        int kk=0;
        float aux_x,aux_y,aux_z;            // 14/11/2001

        valor[1] = 0;
        valor[5] = 0;
        valor[9] = 0;
        for(i=0;i<media;i++)
        {
                leia();
                valor[0] = tensao[0];
                valor[4] = tensao[1];
                valor[8] = tensao[2];
```

```
                valor[1] = valor[1] + valor[0];
                valor[5] = valor[5] + valor[4];
                valor[9] = valor[9] + valor[8];
                kk+=1;
        }

        valor[1] = arredondamento(valor[1]/kk,1);
        valor[5] = arredondamento(valor[5]/kk,1);
        valor[9] = arredondamento(valor[9]/kk,1);

        aux_x =((valor[1]*constantes[0] + constantes[1]));
        aux_y = ((valor[5]*constantes[2] + constantes[3]));
        aux_z = ((valor[9]*constantes[4] + constantes[5]));

        spin=aux_x;
        span=aux_y;
        spun=aux_z;


        origem_raio[0]=10+arredondamento((3+15*cos(span)+18*cos(span-spin))*sin(spun),2);
        origem_raio[1]=5+arredondamento((3+15*cos(span)+18*cos(span-spin))*cos(spun),2);
        origem_raio[2]=arredondamento(8+7+15*sin(span)+18*sin(span-spin),2);
}

// ********************************************************************
// ********** Funcoes geometricas *********************************
// ********************************************************************


// Funcao prod_escalar ***********************************************
double prod_escalar(float ref1[], float ref2[], int tam)
{
        double resp;
        int i;

        resp=0;

        for (i=0;i<tam;i++)
                resp=ref1[i]*ref2[i]+resp;
        return(resp);
}

// Funcao calculaDirecao *********************************************
void calculaDirecao(void)
{
        int i;
        double aux_mod;
        float mod_dir;

        // Cálculo da direção

        for (i=0;i<3;i++)
        {
                dir[i]=ponto_fixo[i]-origem_raio[i];
        }

        aux_mod=dir[0]*dir[0]+dir[1]*dir[1]+dir[2]*dir[2];
        mod_dir=sqrt(aux_mod);
```

```
        for (i=0;i<3;i++)
        {
                dir[i]=dir[i]/mod_dir;
        }
}

// Funcao choque2 **********************************************
void choque2(float ponto_fixo[], float origem_raio[], float plano[],float tamanho)
{
        float xn[3], ponta[3], t_min, aux1;
        int i;

        choque_f = 0;
        marca=marca+1;

//      calculaDirecao();

        // Cálculo do t mín para ocorrer o choque

        xn[0]=plano[0];
        xn[1]=plano[1];
        xn[2]=plano[2];
        aux1=prod_escalar(xn,dir,3);

        // Fim do cálculo do tamanho mínimo

        if (aux1==0)
        {
        //      (" Paralelo\n")
                penetracao=0;
                byte_a1=0;
                byte_b1=0;
                byte_a2=0;
                byte_b2=0;
                byte_a3=0;
                byte_b3=0;
                outp (EndBase+8, byte_a1);
                outp (EndBase+9, byte_b1);
                outp (EndBase+10, byte_a2);
                outp (EndBase+11, byte_b2);
                outp (EndBase+12, byte_a3);
                outp (EndBase+13, byte_b3);
                volt1=volt2=volt3=0;
        }
        else
        {
                t_min=fabs((-plano[3]-prod_escalar(xn,origem_raio,3))/(prod_escalar(xn,dir,3)));
                t_min_global = t_min;
                if(t_min<=tamanho)
                {
//                      " Ocorreu choque por tamanho mínimo

                        for (i=0;i<3;i++)
                        {
                                ponta[i]=origem_raio[i]+tamanho*dir[i];    // Ponta real da ferramenta
                                ponta_aux[i]=origem_raio[i]+t_min*dir[i]; // Ponto de contato no
plano
```

```
                        }
                        // Cálculo da penetração ( 09/10/2001) (10/10/2001)
                        penetracao=arredondamento(tamanho-t_min,3);
                        malha(ponta_aux,plano);

                        if(choque_f==0)
                        {
                                penetracao=0;
                                byte_a1=0;
                                byte_b1=0;
                                byte_a2=0;
                                byte_b2=0;
                                byte_a3=0;
                                byte_b3=0;
                                outp (EndBase+8, byte_a1);
                                outp (EndBase+9, byte_b1);
                                outp (EndBase+10, byte_a2);
                                outp (EndBase+11, byte_b2);
                                outp (EndBase+12, byte_a3);
                                outp (EndBase+13, byte_b3);
                                volt1=volt2=volt3=0;
                        }
                        else
                        {
                                momento_calc(forca,dir,spin,span,spun);          //( 26/10/2001)
                                motor_control(torque1,torque2,torque3);          // 06/11/2001
                        }
                }
                else
                {
                        penetracao=0;
                        byte_a1=0;
                        byte_b1=0;
                        byte_a2=0;
                        byte_b2=0;
                        byte_a3=0;
                        byte_b3=0;
                        outp (EndBase+8, byte_a1);
                        outp (EndBase+9, byte_b1);
                        outp (EndBase+10, byte_a2);
                        outp (EndBase+11, byte_b2);
                        outp (EndBase+12, byte_a3);
                        outp (EndBase+13, byte_b3);
                        volt1=volt2=volt3=0;
                }
        }
}

// Funcao rotacao ***********************************************
void rotacao(float dir[])
{
        float Tz[16];

        if(dir[0]==0 && dir[1]==0 && dir[2]==0)
                rotz=roty=0;
        else if (dir[0]!=0 && dir[1]!=0 && dir[2]==0)
        {
```

```
                if (dir[0]>0 && dir[1]>0)
                        rotz=abs(atan(dir[1]/dir[0])*RADDEG);
                if (dir[0]<0 && dir[1]>0)
                        rotz=180-abs(atan(dir[1]/dir[0])*RADDEG);
                if (dir[0]<0 && dir[1]<0)
                        rotz=180+abs(atan(dir[1]/dir[0])*RADDEG);
                if (dir[0]>0 && dir[1]<0)
                        rotz=-abs(atan(dir[1]/dir[0])*RADDEG);
        }
        else if (dir[0]!=0 && dir[1]==0 && dir[2]!=0)
                rotz=0;
        else if (dir[0]==0 && dir[1]!=0 && dir[2]!=0)
        {
// novo!
                if(dir[1]>0)
                        rotz=-90;
                else if (dir[1]<0)
                        rotz=90;
        }
        else if (dir[0]==0 && dir[1]==0 && dir[2]!=0)
                rotz=0;
        else if (dir[0]!=0 && dir[1]==0 && dir[2]==0)
                rotz=0;
        else if (dir[0]==0 && dir[1]!=0 && dir[2]==0)
        {
                if (dir[1]>0)
                        rotz=90;
                else if (dir[1] < 0)
                        rotz=-90;
        }
        else if (dir[0]!=0 && dir[1]!=0 && dir[2]!=0)
        {
                if (dir[0]>0 && dir[1]>0)
                        rotz=abs(atan(dir[1]/dir[0])*RADDEG);
                if (dir[0]<0 && dir[1]>0)
                        rotz=180-abs(atan(dir[1]/dir[0])*RADDEG);
                if (dir[0]<0 && dir[1]<0)
                        rotz=180+abs(atan(dir[1]/dir[0])*RADDEG);
                if (dir[0]>0 && dir[1]<0)
                        rotz=-abs(atan(dir[1]/dir[0])*RADDEG);
        }


// Rotacao do sistema de coordenadas

        // Preenchimento da matriz

        Tz[0]=cos(-rotz*DEGRAD);
        Tz[1]=-sin(-rotz*DEGRAD);
        Tz[2]=0;
        Tz[3]=0;
        Tz[4]=sin(-rotz*DEGRAD);
        Tz[5]=cos(-rotz*DEGRAD);
        Tz[6]=0;
        Tz[7]=0;
        Tz[8]=0;
        Tz[9]=0;
        Tz[10]=1;
```

```
        Tz[11]=0;
        Tz[12]=0;
        Tz[13]=0;
        Tz[14]=0;
        Tz[15]=1;
        coord_int[0]=Tz[0]*dir[0]+Tz[1]*dir[1]+Tz[2]*dir[2]+Tz[3]*1;
        coord_int[1]=Tz[4]*dir[1]+Tz[5]*dir[1]+Tz[6]*dir[2]+Tz[7]*1;
        coord_int[2]=Tz[8]*dir[0]+Tz[9]*dir[1]+Tz[10]*dir[2]+Tz[11]*1;
        coord_int[3]=Tz[12]*dir[0]+Tz[13]*dir[1]+Tz[14]*dir[2]+Tz[15]*1;

// Calculo do roty

        if(coord_int[0]==0 && coord_int[2]==0)
                roty=0;
        else if (coord_int[0]!=0 && coord_int[2]==0)
        {
                if (coord_int[0]>0)
                        roty=90;
                else if (coord_int[0]<0)
                        roty=-90;
        }
        else if (coord_int[0]==0 && coord_int[2]!=0)
        {
                if (coord_int[2]>0)
                        roty=0;
                else if (coord_int[2]<0)
                        roty=180;
        }
        else if (coord_int[0]!=0 && coord_int[2]!=0)
        {
                if(coord_int[0]>0 && coord_int[2]>0)
                        roty=90-abs(atan(coord_int[2]/coord_int[0])*RADDEG);
                else if(coord_int[0]<0 && coord_int[2]>0)
                        roty=-90+abs(atan(coord_int[2]/coord_int[0])*RADDEG);
                else if(coord_int[0]<0 && coord_int[2]<0)
                        roty=-90-abs(atan(coord_int[2]/coord_int[0])*RADDEG);
                else if(coord_int[0]>0 && coord_int[2]<0)
                        roty=90+abs(atan(coord_int[2]/coord_int[0])*RADDEG);
        }
}


// Funcao malha ********************************************** ( 21/11/2001)
void malha(float pto_contato[],float plano[])
{

        int linha,coluna,aux,coluna_aux,i;
        float delta1[3],delta2[3],perimetro,vetor1[3],vetor2[3],vetor3[3],vetor4[3];
        float prod1,prod2,prod3,prod4,per_calc;
        float prod5,prod6,prod7,prod8,mod1,mod2;
        float x1,y1,z1,dif,tam_1,tam_2,tam_3,tam_4;

        // Variacao ao longo da linha
        delta1[0]=(plano[7]-plano[4])/(size_k-1);
        delta1[1]=(plano[8]-plano[5])/(size_k-1);
        delta1[2]=(plano[9]-plano[6])/(size_k-1);
```

```
        // variacao ao longo das colunas
        delta2[0]=(plano[13]-plano[4])/(size_k-1);
        delta2[1]=(plano[14]-plano[5])/(size_k-1);
        delta2[2]=(plano[15]-plano[6])/(size_k-1);

        // Perimetro

        perimetro=2*sqrt(delta1[0]*delta1[0]+delta1[1]*delta1[1]+delta1[2]*delta1[2])+2*sqrt(delta2[0]
*delta2[0]+delta2[1]*delta2[1]+delta2[2]*delta2[2]);

        // Construcao da malha da superficie

        for(linha=0;linha<size_k;linha++)
        {
                x1=plano[4]+linha*delta2[0];
                y1=plano[5]+linha*delta2[1];
                z1=plano[6]+linha*delta2[2];

                coluna=0;
                coluna_aux=0;

                while(coluna<size_k*3)
                {
                        if(coluna!=0)
                                coluna_aux=coluna/3;



                        malha_aux[linha][coluna]=x1+coluna_aux*delta1[0];
                        malha_aux[linha][coluna+1]=y1+coluna_aux*delta1[1];
                        malha_aux[linha][coluna+2]=z1+coluna_aux*delta1[2];
                        coluna=coluna+3;
                }
        }

        // Procura do ponto de contato na malha

        aux=0;
        linha=0;
        coluna=0;
        mod1=sqrt(delta1[0]*delta1[0]+delta1[1]*delta1[1]+delta1[2]*delta1[2]);
        mod2=sqrt(delta2[0]*delta2[0]+delta2[1]*delta2[1]+delta2[2]*delta2[2]);

        for(i=0;i<3;i++)
        {
                delta1[i]=delta1[i]/mod1;
                delta2[i]=delta2[i]/mod2;
        }

        while((aux==0) && (linha<size_k))
        {
                coluna=0;
                while ((aux==0) && (coluna<size_k*3))
                {
                        vetor1[0]=pto_contato[0]-malha_aux[linha][coluna];
                        vetor1[1]=pto_contato[1]-malha_aux[linha][coluna+1];
                        vetor1[2]=pto_contato[2]-malha_aux[linha][coluna+2];
                        prod1=fabs(prod_escalar(delta1,vetor1,3));
```

```
                    prod2=fabs(prod_escalar(delta2,vetor1,3));

                    vetor2[0]=pto_contato[0]-malha_aux[linha][coluna+3];
                    vetor2[1]=pto_contato[1]-malha_aux[linha][coluna+4];
                    vetor2[2]=pto_contato[2]-malha_aux[linha][coluna+5];
                    prod3=fabs(prod_escalar(delta1,vetor2,3));
                    prod4=fabs(prod_escalar(delta2,vetor2,3));

                    vetor3[0]=pto_contato[0]-malha_aux[linha+1][coluna];
                    vetor3[1]=pto_contato[1]-malha_aux[linha+1][coluna+1];
                    vetor3[2]=pto_contato[2]-malha_aux[linha+1][coluna+2];
                    prod5=fabs(prod_escalar(delta1,vetor3,3));
                    prod6=fabs(prod_escalar(delta2,vetor3,3));

                    vetor4[0]=pto_contato[0]-malha_aux[linha+1][coluna+3];
                    vetor4[1]=pto_contato[1]-malha_aux[linha+1][coluna+4];
                    vetor4[2]=pto_contato[2]-malha_aux[linha+1][coluna+5];
                    prod7=fabs(prod_escalar(delta1,vetor4,3));
                    prod8=fabs(prod_escalar(delta2,vetor4,3));

                    per_calc=prod1+prod2+prod3+prod4+prod5+prod6+prod7+prod8;

                    dif=fabs(perimetro-per_calc);

                    if(dif<=0.0001)
                            aux=1;

                    if(aux==0)
                            coluna=coluna+3;
            }

            if(aux==0)
            {
                    linha=linha+1;
            }
    }

    if (aux==1)
    {
            coluna_aux=coluna/3;
            k_matriz=matriz_k[linha][coluna_aux];
            forca=k_matriz*penetracao*0.01;          // Obervar as unidades utilizadas cm
            choque_f=1;

            ponto_prox[3]=delta1[0];
            ponto_prox[4]=delta1[1];
            ponto_prox[5]=delta1[2];
            ponto_prox[6]=delta2[0];
            ponto_prox[7]=delta2[1];
            ponto_prox[8]=delta2[2];
            ponto_prox[9]=linha;
            ponto_prox[10]=coluna_aux;
    }
    else
            choque_f=0;
}
```

```
// Funcao momento_calc **************************************************
void momento_calc(float forca,float dir[],float spin, float span, float spun) //( 26/10/2001)
{
        float n1,n2,n3; // Relacao de transmissao entre os motores e as engrenagens
        float mod_xy,dir_z,Fz,Fxy;
        float barra_a,barra_b,barra_c,barra_d,barra_e;
        float torque_aux;

        barra_a=15;             // geometria do mecanismo medidas em cm (observar potência do
motor) Newton ou KGF
        barra_b=7.5;
        barra_c=18;
        barra_d=7.5;
        barra_e=15;

        n1=n2=3.2;              // Número de dentes na coroa=32 N dentes no pinhao=10
        n3=1;


        mod_xy=sqrt(dir[0]*dir[0]+dir[1]*dir[1]);
        dir_z=fabs(dir[2]);
        if((mod_xy==0) && (dir_z==0))
        {
                Fz=0;
                Fxy=0;
        }
        else if((mod_xy!=0) && (dir_z!=0))
        {
                Fz=forca*sin(dir_z/mod_xy);
                Fxy=forca*cos(dir_z/mod_xy);
        }
        else if ((mod_xy==0) && (dir_z!=0))
        {
                Fz=forca;
                Fxy=0;
        }
        else if ((mod_xy!=0) && (dir_z==0))
        {
                Fz=0;
                Fxy=forca;
        }
        // Cálculo dos torques na barra da base (A Fz  e Fxy são em N.m, mas a barras são em cm
        torque_aux=Fz*barra_a*cos(span)*0.01;
        torque1=torque_aux/n1;

        // Cálculo do torque dom motor da haste da ponta (observar como é calculado o spin span e
spun)
        if (spin!=0)
        {
                torque_aux=(Fz*(barra_d*barra_c/barra_b)*sin(90*DEGRAD+span-
spin)/sin(spin))*0.01;
                torque2=torque_aux/n2;
        }
        else if (spin==0)
        {
                torque_aux=barra_b*Fz*cos(span)*0.01;
                torque2=torque_aux/n2;
        }
```

```
        // Cálculo do torque do motor da base (plano xy)
        torque3=((barra_a*cos(span)+barra_b*cos(spin)*cos(span))*Fxy/n3)*0.01;
}

void motor_control(float torque1, float torque2, float torque3)                    // 06/11/2001
{
//      float volt1,volt2,volt3;
        float k_motor, aux_volt,step1,step2,step3;
        int resto1,resto2,resto3;

        // os torques são calculados em N.m, e devem ser convertidos para Kgf.m
        // 1 Kgf = 10 N => deve-se dividir por 10 os torques calculados
        k_motor=1;
        volt1=-((torque1*100+5.719)/2.706)/2;          // O ganho do Amp OP é 2, portanto a
voltagem de saída deve ser a metade
        volt2=((torque2*100+7.631)/3.644)/2;
        volt3=torque3/k_motor/2;


        // Verificar a tabela de bytes

        if(volt1>1.8)
        {
                byte_a1=15<<4;
                byte_b1=22;
        }
        else
        if(volt1<-1.8)
        {
                byte_a1=2<<4;
                byte_b1=234;
        }
        else
        {
                step1=fabs(volt1/0.0049);

                if(volt1<0)
                        byte_b1=256-(int)(step1/16);
                else
                        byte_b1=(int)(step1/16);

                resto1=(int)(step1-byte_b1*16);

                if(volt1<0)
                {

                        byte_a1=(17-resto1)<<4;
                }
                else
                {
                        byte_a1=resto1<<4;
                }
        }

        if(volt2>1.8)
        {
                byte_a2=15<<4;
```

```
            byte_b2=22;
    }
    else if(volt2<-1.8)
    {
            byte_a2=2<<4;
            byte_b2=234;
    }
    else
    {
            step2=fabs(volt2/0.0049);
            if(volt2<0)
                    byte_b2=256-(int)(step2/16);
            else
                    byte_b2=(int)(step2/16);
            resto2=(int)(step2-byte_b2*16);
            if(volt2<0)
            {
                    byte_a2=(17-resto2)<<4;
            }
            else
            {
                    byte_a2=resto2<<4;
            }
    }

    if(volt3>1.8)
    {
            byte_a3=15<<4;
            byte_b3=22;
    }
    else if(volt3<-1.8)
    {
            byte_a3=2<<4;
            byte_b3=234;
    }
    else
    {
            step3=fabs(volt3/0.0049);
            if(volt3<0)
                    byte_b3=256-(int)(step3/16);
            else
                    byte_b3=(int)(step3/16);
            resto3=(int)(step3-byte_b3*16);
            if(volt3<0)
            {
                    byte_a3=(17-resto3)<<4;
            }
            else
            {
                    byte_a3=resto3<<4;
            }
}

outp (EndBase+8, byte_a1);
outp (EndBase+9, byte_b1);
outp (EndBase+10, byte_a2);
outp (EndBase+11, byte_b2);
outp (EndBase+12, byte_a3);
```

```
        outp (EndBase+13, byte_b3);
}

float arredondamento(float num, int casas)
{
        float num1,num_trunc,resto;
        int i,divisor;

        num1=num;
        divisor=1;
        for(i=1;i<=casas;i++)
        {
                num1=num1*10;
                divisor=divisor*10;
        }
        num_trunc=(int)(num1);
        resto=num1-num_trunc;
        if(resto>=0.5)
                return (num_trunc+1)/divisor;
        else
                return (num_trunc)/divisor;
}


// ********************************************************************
// ********** Funcao de uniao ***************************************
// ********************************************************************
// Funcao anime - Uniao de funcoes da placa, geometricas e graficas *
void anime(void)
{
        spinDisplay();
        calculaDirecao();
        choque2(ponto_fixo,origem_raio,plano,tamanho);
        rotacao(dir);
        glutPostRedisplay();
}

// ********************************************************************
// ********** Programa principal ************************************
// ********************************************************************
int main(int argc, char **argv)
{
        float aux_plano,mod_plano;
        int i;
        int j;

        Inicializa();
        Calibra();

        plano[0]=0;
        plano[1]=0;
        plano[2]=1;
        plano[3]=0;
        plano[4]=0;
        plano[5]=0;
        plano[6]=0;
        plano[7]=MAX_MESH;
        plano[8]=0;
```

```
plano[9]=0;
plano[10]=MAX_MESH;
plano[11]=MAX_MESH;
plano[12]=0;
plano[13]=0;
plano[14]=MAX_MESH;
plano[15]=0;

matriz_k[0][0]=32200;
matriz_k[0][1]=20100;
matriz_k[0][2]=17700;
matriz_k[0][3]=16900;
matriz_k[0][4]=16700;
matriz_k[0][5]=16900;
matriz_k[0][6]=17700;
matriz_k[0][7]=20100;
matriz_k[0][8]=32200;

matriz_k[1][0]=20100;
matriz_k[1][1]=8340;
matriz_k[1][2]=6090;
matriz_k[1][3]=5370;
matriz_k[1][4]=5180;
matriz_k[1][5]=5370;
matriz_k[1][6]=6090;
matriz_k[1][7]=8340;
matriz_k[1][8]=20100;

matriz_k[2][0]=17700;
matriz_k[2][1]=6090;
matriz_k[2][2]=3910;
matriz_k[2][3]=3230;
matriz_k[2][4]=3050;
matriz_k[2][5]=3230;
matriz_k[2][6]=3910;
matriz_k[2][7]=6090;
matriz_k[2][8]=17700;

matriz_k[3][0]=16900;
matriz_k[3][1]=5370;
matriz_k[3][2]=3230;
matriz_k[3][3]=2560;
matriz_k[3][4]=2390;
matriz_k[3][5]=2560;
matriz_k[3][6]=3230;
matriz_k[3][7]=5370;
matriz_k[3][8]=16900;

matriz_k[4][0]=16700;
matriz_k[4][1]=5180;
matriz_k[4][2]=3050;
matriz_k[4][3]=2390;
matriz_k[4][4]=2230;
matriz_k[4][5]=2390;
matriz_k[4][6]=3050;
matriz_k[4][7]=5180;
matriz_k[4][8]=16700;
```

```
 matriz_k[5][0]=16900;
 matriz_k[5][1]=5370;
 matriz_k[5][2]=3230;
 matriz_k[5][3]=2560;
 matriz_k[5][4]=2390;
 matriz_k[5][5]=2560;
 matriz_k[5][6]=3230;
 matriz_k[5][7]=5370;
 matriz_k[5][8]=16900;

 matriz_k[6][0]=17700;
 matriz_k[6][1]=6090;
 matriz_k[6][2]=3910;
 matriz_k[6][3]=3230;
 matriz_k[6][4]=3050;
 matriz_k[6][5]=3230;
 matriz_k[6][6]=3910;
 matriz_k[6][7]=6090;
 matriz_k[6][8]=17700;

 matriz_k[7][0]=20100;
 matriz_k[7][1]=8340;
 matriz_k[7][2]=6090;
 matriz_k[7][3]=5370;
 matriz_k[7][4]=5180;
 matriz_k[7][5]=5370;
 matriz_k[7][6]=6090;
 matriz_k[7][7]=8340;
 matriz_k[7][8]=20100;

 matriz_k[8][0]=32200;
 matriz_k[8][1]=20100;
 matriz_k[8][2]=17700;
 matriz_k[8][3]=16900;
 matriz_k[8][4]=16700;
 matriz_k[8][5]=16900;
 matriz_k[8][6]=17700;
 matriz_k[8][7]=20100;
 matriz_k[8][8]=32200;

aux_plano=plano[0]*plano[0]+plano[1]*plano[1]+plano[2]*plano[2];
mod_plano=sqrt(aux_plano);

for (i=0;i<3;i++)
{
        plano[i]=plano[i]/mod_plano;
}

ponto_fixo[0]=10;
ponto_fixo[1]=5;
ponto_fixo[2]=3;

glutInitWindowSize(500, 500);
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
glutCreateWindow("Simulador Cirúrgico");
init();
glutReshapeFunc(reshape);
```

```
        glutKeyboardFunc(keyboard);
        glutSpecialFunc(special_keys);
        glutDisplayFunc(display);
        glutIdleFunc(anime);
        glutMainLoop();
        return 0;
}
```

Arquivo Material.h

```
GLfloat LightAmbient[] = { 0.0f, 0.0f, 0.0f, 0.0f };
GLfloat LightDiffuse[] = { 1.0f, 1.0f, 1.0f, 1.0f };
GLfloat LightSpecular[] = { 1.0f, 1.0f, 1.0f, 1.0f };
GLfloat LightPosition[] = { -2.0f, 2.0f, 2.0f, 1.0f };

GLfloat mat_amb_amarelo[] = { 0.13, 0.13, 0.00, 1.00 };
GLfloat mat_dif_amarelo[] = { 0.95, 0.95, 0.00, 1.00 };
GLfloat mat_spc_amarelo[] = { 0.85, 0.85, 0.00, 1.00 };

GLfloat mat_amb_prata[] = { 0.13, 0.13, 0.00, 1.00 };
GLfloat mat_dif_prata[] = { 1.00, 1.00, 1.00, 1.00 };
GLfloat mat_spc_prata[] = { 1.00, 1.00, 1.00, 1.00 };

GLfloat mat_amb_vermelho[] = { 0.00, 0.00, 0.00, 1.00 };
GLfloat mat_dif_vermelho[] = { 1.00, 0.00, 0.00, 1.00 };
GLfloat mat_spc_vermelho[] = { 0.00, 0.00, 0.00, 1.00 };

GLfloat mat_amb_azulesc[] = { 0.10, 0.10, 1.00, 1.00 };
GLfloat mat_dif_azulesc[] = { 0.50, 0.50, 1.00, 1.00 };
GLfloat mat_spc_azulesc[] = { 0.30, 0.30, 0.00, 1.00 };

GLfloat mat_amb_azul[] = { 0.10, 0.10, 1.00, 1.00 };
GLfloat mat_dif_azul[] = { 0.50, 0.50, 1.00, 1.00 };
GLfloat mat_spc_azul[] = { 0.30, 0.30, 0.00, 1.00 };

GLfloat mat_emission_escuro[] = { 0.00, 0.00, 0.00, 0.00 };
GLfloat mat_emission_claro[] = { 0.75, 0.75, 0.75, 1.00 };

GLfloat mat_shininess[] = { 100.0 };
```