DESENVOLVIMENTO DE UM SISTEMA MECATRÔNICO INTERATIVO PARA AUXILIAR NO TRATAMENTO DE REABILITAÇÃO DE CRIANÇAS COM DEFICIÊNCIA MOTORA NAS PERNAS

Adriano José Marques Ribeiro

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

Orientador: Prof. Dr. Glauco Augusto de Paula Caurin

São Carlos
2004

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

R482d

Ribeiro, Adriano José Marques
Desenvolvimento de um sistema mecatrônico interativo para auxiliar no tratamento de reabilitação de crianças com deficiência motora nas pernas / Adriano José Marques Ribeiro. -- São Carlos, 2004.

Dissertação (Mestrado) -- Escola de Engenharia de São Carlos-Universidade de São Paulo, 2004.
Área: Engenharia Mecânica.
Orientador: Prof. Dr. Glauco Augusto de Paula Caurin.

1. Plasticidade cerebral. 2. Deficiência motora. 3. Ambiente virtual. 4. Jogo. 5. Interação. 6. Tempo real. 7. Force-feedback. I. Título.

FOLHA DE JULGAMENTO

Candidato: Engenheiro **ADRIANO JOSÉ MARQUES RIBEIRO**

Dissertação defendida e julgada em 16-11-2004 perante a Comissão Julgadora:

Aprovado

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

Aprovado

Prof. Dr. **LUIZ CARLOS FELÍCIO**
(Escola de Engenharia de São Carlos/USP)

Aprovado

Profa. Dra. **AGMA JUCI MACHADO TRAINA**
(Instituto de Ciências Matemáticas e de Computação/USP)

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

Profa. Titular **MARIA DO CARMO CALIJURI**
Presidente da Comissão de Pós-Graduação

AGRADECIMENTOS

Ao Professor Glauco, pela confiança no desenvolvimento deste trabalho.

Aos meus pais, pelo apoio e carinho. A paciência de vocês não tem fim.

Em especial ao meu irmão Paulo Estevão, que contribuiu significativamente para o desenvolvimento dos sistemas eletrônicos do projeto.

RESUMO

RIBEIRO, A. J. M. (2004). *Desenvolvimento de um sistema mecatrônico interativo para auxiliar no tratamento de reabilitação de crianças com deficiência motora nas pernas.* Dissertação (Mestrado) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2004.

O tratamento de reabilitação motora compreende o emprego de um conjunto de técnicas terapêuticas que visam incitar a ocorrência da plasticidade cerebral, atividade fisiológica na qual o sistema nervoso central busca estabelecer novas ligações sinápticas para promover a recuperação da função neuromuscular. Observando-se as técnicas atualmente empregadas e fundamentando-se no fato da criança apresentar alta capacidade de sofrer plasticidade neural, decidiu-se desenvolver um sistema mecatrônico interativo como ferramenta de auxílio ao tratamento de reabilitação de crianças com deficiência motora nas pernas. O presente trabalho descreve a implementação deste sistema, composto por uma bicicleta instrumentada, integrada a um ambiente virtual de aparência lúdica, desenvolvidos com o objetivo de proporcionar a execução dos exercícios físicos de forma divertida e cativante, estimulando a ocorrência de uma plasticidade neural mais rápida e eficiente. Ao final do trabalho são relatadas algumas demonstrações práticas realizadas, onde foram avaliados os níveis de motivação das crianças, além de alguns depoimentos médicos a respeito das expectativas do projeto.

Palavras-chaves: plasticidade cerebral, deficiência motora, ambiente virtual, jogo, interação, tempo real, *force-feedback*.

ABSTRACT

RIBEIRO, A. J. M. (2004). *Development of interactive mechatronic system to assist the rehabilitation treatment of children with motor disturbances in their legs.* M.Sc. Dissertation – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2004.

The rehabilitation treatment consists in a set of therapeutic techniques that try to stimulate cerebral plasticity. Cerebral plasticity is a physiologic phenomenon where the brain searches for new neural connections to supply the deficient function. It is known that the young brain has a high cerebral plasticity capacity. Based on this, an interactive mechatronic system was developed to assist the rehabilitation treatment of children with motor disturbances in their legs. This system is composed of an instrumented bicycle and an entertaining virtual world generated by computer, developed to stimulate the child to accomplish the physical activities in a captivating and fun way and, consequently, increasing the cerebral plasticity performance. In the end of this document some practical experiments were reported, showing the motivation of the children. Some medical opinions regarding the expectations of the project were also reported.

Keywords: *cerebral plasticity, motor disturbances, virtual world, game, interaction, real time, force-feedback.*

SUMÁRIO

LISTA DE FIGURAS

1 INTRODUÇÃO

O sucesso da aplicação de sistemas tecnológicos em programas educativos e de treinamento contribui para que outras possibilidades sejam exploradas e abre novas perspectivas para o uso em diferentes campos do conhecimento. Nos últimos anos, a área de saúde vem sendo impulsionada pelas novas tecnologias integradas aos procedimentos médicos, com destaque para a utilização de sistemas de realidade virtual e jogos computacionais no tratamento de reabilitação de crianças portadores de algum tipo de deficiência.

Os programas de reabilitação buscam, por meio de brincadeiras e atividades lúdicas, minimizar as limitações destas crianças especiais e ampliar suas potencialidades. Neste contexto, os jogos computacionais podem exercer um papel fundamental. Toda criança revela um grande fascínio por videogames e jogos computacionais. Afinal, brincar é o meio de expressão mais importante no desenvolvimento da criança. Jogos eletrônicos oferecem um mundo de fantasias, desejos e desafios que, por si só, já incitam a ocorrência de novas ligações sinápticas, proporcionando o aprimoramento das habilidades motoras e estimulando a concentração, a memória e o raciocínio lógico.

O presente projeto trata do desenvolvimento de um sistema mecatrônico interativo, no qual uma bicicleta especialmente instrumentada é integrada a um ambiente virtual tridimensional de aparência lúdica. Para jogar ou mover-se dentro do cenário virtual, a criança precisa pedalar e coordenar a direção do seu movimento. Um motor controlado computacionalmente reproduz a resistência ao movimento de acordo com o relevo do ambiente virtual. Os objetivos e a qualidade visual do jogo, além de estimularem suas funções cognitivas, motivam a criança a realizar as atividades físicas de forma bastante divertida. É importante mencionar, no entanto, que esta tecnologia não substitui o tratamento convencional, mas o complementa (como uma ferramenta de auxílio) na busca por uma recuperação ou aprendizagem mais eficiente.

A dissertação deste trabalho está divida em capítulos. No terceiro capítulo é feita uma breve fundamentação fisiológica a respeito da deficiência motora em crianças, apresentando suas principais causas e técnicas terapêuticas normalmente empregadas. O quarto capítulo faz uma abordagem tecnológica dos sistemas de realidade virtual e busca

colocar o leitor a par de algumas das principais pesquisas e produtos relacionados à aplicação de ambientes virtuais em terapias. No quinto capítulo são apresentadas as ferramentas computacionais usadas no desenvolvimento do projeto. O sexto capítulo oferece um embasamento teórico e descreve a metodologia necessária à implementação do ambiente virtual. No sétimo capítulo é descrita a modelagem do ambiente virtual. No oitavo capítulo são apresentadas as interfaces de comunicação e os processos de captura e envio de dados. No nono capítulo são apresentados e discutidos os resultados. No décimo capítulo são mostradas as conclusões do trabalho. Por fim, um apêndice reúne observações complementares ao projeto.

2 OBJETIVO

O objetivo deste projeto é desenvolver um sistema mecatrônico interativo como ferramenta de auxilio no tratamento de reabilitação de crianças com deficiência motora nas pernas. O sistema, composto por uma bicicleta instrumentada e integrada a um ambiente virtual de aparência lúdica, visa proporcionar à criança a prática divertida de atividades físicas, estimulando suas habilidades cognitivas e motoras de forma relacionada, em prol da ocorrência de uma plasticidade cerebral mais eficiente e, conseqüentemente, de uma reabilitação mais otimizada.

3 ASPECTOS FISIOLÓGICOS

3.1 Deficiência motora

A deficiência motora refere-se ao comprometimento do aparelho locomotor, formado pelos sistemas esquelético, muscular e nervoso. As doenças ou lesões que afetam estes sistemas, isoladamente ou em conjunto, podem produzir quadros de limitações físicas com graus variáveis, dependendo dos segmentos corporais afetados e do tipo de lesão ocorrida (NAPOLI, 2003).

3.1.1 Dados estatísticos

A Organização Mundial da Saúde estima que, em tempos de paz, 10% da população dos países desenvolvidos é composta por pessoas com algum tipo de deficiência. Nos países subdesenvolvidos, estima-se que este percentual seja de 12 a 15%, dentre os quais 20% são portadores de deficiência motora. No Brasil, especificamente, cerca de 24,5 milhões de pessoas (14,5% da população) possuem algum tipo de deficiência, dentre as quais 22,9% apresentam deficiência motora (IBGE, 2000).

3.1.2 Tipos de deficiência motora

Quanto à região do corpo comprometida, é possível classificar a deficiência motora nas seguintes categorias (IBC, 2004):

- Monoparesia: perda parcial das funções motoras de apenas um membro do corpo.
- Monoplegia: perda total das funções motoras de apenas um membro do corpo.
- Paraparesia: perda parcial das funções motoras dos membros inferiores.
- Paraplegia: perda total das funções motoras dos membros inferiores.
- Triparesia: perda parcial das funções motoras de três membros do corpo.
- Triplegia: perda total das funções motoras de três membros do corpo.
- Hemiparesia: perda parcial das funções motoras de um dos lados do corpo.
- Hemiplegia: perda total das funções motoras de um dos lados do corpo.
- Tetraparesia: perda parcial das funções motoras dos membros superiores e inferiores.
- Tetraplegia: perda total das funções motoras dos membros superiores e inferiores.

- Amputação: perda física total ou parcial de um ou mais membros do corpo.

3.1.3 Causas principais da deficiência motora

De um modo geral, o comprometimento das funções motoras resulta de uma ou mais lesões no sistema nervoso central, podendo ser de origem espinhal (lesões na medula) ou de origem encefálica (paralisia cerebral, esclerose múltipla ou acidente vascular cerebral) (SACI, 2002).

3.1.3.1 Lesão medular

A lesão medular ocorre quando um evento traumático (acidente automobilístico, mergulho em águas rasas, agressão com arma de fogo ou queda) sofrido pela pessoa resulta em lesão das estruturas medulares, interrompendo a passagem de estímulos nervosos através da medula.

A lesão medular pode ser completa ou incompleta. É completa quando não houver movimentos voluntários abaixo da região lesionada e incompleta em caso contrário. A medula pode também ser lesionada por causas não traumáticas, como doenças hemorrágicas, tumores e infecções virais.

Infelizmente, ainda não existem métodos fisioterapêuticos ou cirúrgicos capazes de restaurar a função medular. No entanto, estudos envolvendo implantes de células tronco e células progenitoras oferecem uma expectativa otimista para futuros tratamentos (SARAH, 2004).

3.1.3.2 Esclerose Múltipla

A esclerose múltipla é uma doença neurológica inflamatória que acomete mais freqüentemente pessoas com idade entre 20 e 40 anos, sendo mais comum em mulheres e em indivíduos de cor branca. Acredita-se que a esclerose múltipla seja uma doença autoimune, onde o próprio sistema imunológico ataca a mielina cerebral (desmielinização) (SACI, 2002).

3.1.3.3 Acidente vascular cerebral

A interrupção repentina do fluxo sangüíneo em algumas partes do cérebro caracteriza uma isquemia cerebral. Caso este fluxo permaneça interrompido por um longo período de tempo, a conseqüente falta de oxigenação pode provocar uma lesão cerebral, resultando no chamado acidente vascular cerebral (AVC) (SARAH, 2004).

Algumas vezes, os sintomas do AVC podem melhorar espontaneamente em poucos dias, ou até mesmo em poucos minutos, caracterizando o chamado déficit neurológico focal reversível. Em outros casos, no entanto, as conseqüências do AVC podem ser graves, com comprometimento significativo das funções motoras, sensitivas e/ou fisiológicas. A ocorrência do AVC é mais freqüente em adultos com mais de 45 anos, normalmente os que possuem diabetes, pressão alta, doenças cardíacas ou altas taxas de colesterol (SACI, 2002).

3.1.3.4 Paralisia cerebral

Segundo a Associação de Assistência à Criança Deficiente (2004), 72% das crianças atendidas diariamente são portadoras de paralisia cerebral.

O termo paralisia cerebral é usado para definir qualquer desordem psicomotoras decorrente de uma lesão cerebral não progressiva. Dependendo da região lesionada e de sua extensão, a paralisia cerebral pode ou não acarretar uma deficiência mental. Uma criança com paralisia cerebral pode apresentar alterações que variam desde uma leve falta de coordenação dos movimentos até a incapacidade para segurar um objeto, para falar ou mesmo para deglutir um alimento.

Geralmente os portadores de paralisia cerebral possuem movimentos involuntários, como espasmos musculares repentinos (rigidez muscular) ou a hipotonia (flacidez muscular). A falta de equilíbrio dificulta o deslocamento erétil e a capacidade de segurar objetos.

Desde que o médico inglês William Little, em 1860, descreveu pela primeira vez as alterações clínicas encontradas em uma criança com paralisia cerebral e as relacionou à hipoxia (baixa oxigenação cerebral), valorizou-se muito essa disfunção fisiológica como fator determinante na causa de lesões cerebrais irreversíveis. Mesmo depois de Sigmund Freud, em 1897, ter afirmado que as prováveis causas da paralisia cerebral poderiam também estar relacionadas a agressões ocorridas nas fases mais precoces da vida intra-

uterina (baseado no fato de que muitas destas crianças apresentavam, além das alterações motoras, retardamento mental, convulsões e distúrbios visuais), ainda se manteve a opinião de que a paralisia cerebral decorria exclusivamente da hipoxia perinatal.

Freud acreditava que, em certos casos, os problemas relacionados ao nascimento seriam conseqüências de um desenvolvimento anormal do cérebro. Durante anos, suas observações não foram muito valorizadas. Porém, no final da década de 70, pesquisas importantes realizadas nos Estados Unidos e na Austrália demonstraram que a hipoxia não é a principal causa de paralisia cerebral e que, na maioria das crianças com paralisia cerebral, a causa ainda era desconhecida. Desordens genéticas, fatores teratogênicos ou outras influências nas fases iniciais da gravidez necessitavam de investigações mais profundas.

Os avanços tecnológicos na genética e demais áreas relacionadas têm contribuído para uma melhor compreensão das causas da paralisia cerebral. Um número significativo de crianças que, em tempos passados, seriam diagnosticadas como portadores de paralisia cerebral por hipoxia perinatal (por simplesmente demorarem a chorar ou apresentarem cianose), são hoje, após uma sessão de ressonância magnética, diagnosticadas como portadores de paralisia cerebral por malformação cerebral, decorrente de uma desordem genética ou um fator agressivo ocorrido nas primeiras semanas de gestação.

Dentre as causas pré-natais, as mais importantes, além das desordens genéticas, são as infecções congênitas (citomegalia, toxoplasmose, rubéola) e a hipoxia fetal, decorrente de complicações maternas, como hemorragias. A exposição da mãe a substâncias tóxicas ou a agentes teratogênicos (radiação, álcool, cocaína e certos medicamentos), principalmente nos primeiros meses de gestação, produz um aumento significativo nas chances de se gerar uma criança com problemas. As causas perinatais estão relacionadas às complicações ocorridas durante o parto. Após o nascimento, as principais causas da paralisia cerebral devem-se a infecções no sistema nervoso central, como meningites e encefalites, ao traumatismo crânio-encefálico provocado por acidentes e à hipoxia cerebral grave, normalmente ocorrida em casos de afogamento, convulsões prolongadas e paradas cardíacas (SARAH, 2004).

3.2 Plasticidade cerebral e fisioterapia

O sistema nervosos central (SNC) possui uma rede neural complexa, com células altamente especializadas, que fazem milhares de conexões a todo momento e determinam a sensibilidade e as ações motoras. Na presença de lesões, há um desarranjo nesta rede neural e o SNC inicia seus processos de reorganização e regeneração. A plasticidade neural refere-se à capacidade que o SNC possui em alterar algumas das suas propriedades morfológicas e funcionais em resposta às alterações do ambiente. A análise dos aspectos plásticos do SNC permite-nos relacioná-los a vários fatores, como a influência do meio ambiente, o estado emocional, o nível cognitivo e outros agentes que interferem direta ou indiretamente na plasticidade cerebral e, conseqüentemente, na reabilitação do paciente neurológico (OLIVEIRA *et al.*, 2000).

Durante muito tempo acreditou-se que o SNC, após seu desenvolvimento, tornava-se uma estrutura rígida, que não poderia ser modificada, e que lesões nele seriam permanentes, pois suas células não poderiam ser reconstituídas ou reorganizadas. Hoje, sabe-se que o SNC tem grande adaptabilidade e que, mesmo no cérebro adulto, há evidências de plasticidade na tentativa de regeneração.

Um paciente que experimenta os fenômenos da recuperação após lesão cerebral possui um SNC anormal ou atípico, não só em termos das disfunções alteradas ou perdidas, mas também em termos das conexões sinápticas destruídas ou modificadas, devido ao processo de reorganização do SNC. Este processo é também responsável pelas modificações que são observadas clinicamente no sistema neuromuscular dos pacientes. Por esses meios, diz-se que o indivíduo pode reaprender a executar as atividades previamente desenvolvidas por ele de forma espontânea e harmoniosa. Porém, o processo é lento e gradual, devendo ser valorizados os pequenos progressos de cada dia.

A reabilitação do cérebro lesionado pode promover a reconexão dos circuitos neuronais rompidos. Quando a perda de conectividade neural é pequena, a recuperação tende a ser autônoma. Já uma grande lesão pode implicar na perda permanente da função. Há, no entanto, lesões potencialmente recuperáveis, mas que, para tanto, necessitam de objetivos precisos durante o tratamento, mantendo níveis adequados de estímulos facilitadores e inibidores.

O conhecimento dos mecanismos celulares e funcionais dos fenômenos da plasticidade do SNC contribui para o esclarecimento das causas dos desequilíbrios cinesiopatológicos no diagnóstico das perdas da independência funcional dos pacientes. Além disso, tem a função de nortear o programa de intervenção terapêutica, contribuindo para o estabelecimento de limites, duração da intervenção ou seleção de métodos e técnicas que sejam mais apropriadas à recuperação funcional do sistema nervoso.

O SNC reage aos estímulos recebidos pelos sensores biológicos do paciente. Portanto, é responsabilidade do terapeuta selecionar os métodos mais eficientes à necessidade de cada paciente. Uma variedade de combinações de procedimentos terapêuticos pode ser adotada para ajudar o indivíduo a aprender ou a reaprender o padrão de resposta normal. Esta abordagem dá ao profissional a opção de escolher entre vários procedimentos e promove um ambiente de aprendizado flexível, dinâmico e interessante.

A recuperação da força muscular e o aumento na habilidade funcional ocorrem por meio de vários processos fisiológicos. Os neurônios recuperados desenvolvem brotamentos axonais para reinervar as fibras musculares órfãs. Os ganhos funcionais iniciais são atribuídos à redução do edema cerebral, à absorção de tecido lesado e a melhora do fluxo vascular local. Como o sistema nervoso em desenvolvimento é mais plástico do que o sistema nervoso adulto, uma lesão em uma criança é geralmente caracterizada por uma boa recuperação funcional. Contudo, uma lesão em um idoso pode ser mais danosa e de difícil recuperação. Quanto menor for a lesão, maiores são as chances de que ocorra uma recuperação significativa.

Atualmente, pesquisas estão comprovando que a atuação da fisioterapia, por meio de estímulos aos padrões normais de movimento e inibição dos padrões anormais, provoca uma aceleração no processo de recuperação funcional cerebral. Um experimento realizado em pacientes com AVC crônico evidenciou um aumento significativo das atividades cerebrais nas áreas corticais relacionadas à ação motora, graças a um efetivo programa de reabilitação que induzia o paciente a se movimentar (SANTOS *et al.*, 2002).

A atuação correta e eficaz da equipe de reabilitação na estimulação da plasticidade é de fundamental importância para a máxima recuperação da função motora da criança. O tratamento adequado à criança com paralisia cerebral precisa ser realizado por uma equipe interdisciplinar, onde cada profissional deve ultrapassar as barreiras de sua atuação,

permitindo a troca de conhecimentos com profissionais de outras áreas, a fim de que a criança seja estimulada de uma maneira global e tenha um melhor desenvolvimento neuropsicomotor.

O terapeuta, de uma maneira geral, deve propiciar situações que motivem a criança a explorar o meio ambiente, a fim de que ela se torne um agente ativo no aprendizado de padrões motores e cognitivos. Neste sentido, o terapeuta deve buscar uma ligação afetiva com a criança por meio de brincadeiras que estimulem o desenvolvimento de suas funções. Existem várias técnicas de tratamento, contudo a sua eficácia dependerá, antes de tudo, do terapeuta gostar e acreditar no que faz, gerando condições para que a criança com paralisia cerebral seja capaz, dentro de suas limitações, de se desenvolver, de aprender e de ser feliz (SOARES, 2002).

3.3 Técnicas fisioterapêuticas empregadas na reabilitação motora de crianças

O tratamento para a reabilitação de crianças com deficiência motora tem como função promover o mais alto nível de funcionamento do aparelho neuromusculoesquelético, a aprendizagem ou reaprendizagem das habilidades motoras requeridas pelas atividades diárias ou, ainda, a adaptação do paciente a uma nova realidade. A seguir são apresentadas as técnicas fisioterapêuticas normalmente empregadas no tratamento convencional para a reabilitação de crianças com deficiência motora (KISNER, 1999).

- Cinesioterapia: Consiste em técnicas de alongamento e fortalecimento muscular, por meio de exercitação passiva e ativa dos grupos musculares.

- Mecanoterapia: Consiste no uso de aparelhos ou dispositivos mecânicos para fins de fortalecimento muscular e mobilização das articulações. Exemplos: Halteres, anilhas, *thera-band*, escada de *Ling*, *medicine Ball*, *master cooper*, mesa de *Kanavel*, roda de ombro e escada de dedos.

- Termoterapia: Consiste no uso do calor, promovendo o relaxamento muscular e aumentando o fluxo sangüíneo devido a dilatação dos vasos. Formas de aplicação:

Compressas, bolsas térmicas, radiação infravermelha, forno de *Bier* e banhos em parafina quente.

- Eletroterapia: Consiste na aplicação de estímulos elétricos por meio de eletrodos, com a função de excitar um número determinado de fibras musculares. Exercitam a musculatura e dilatam os vasos, melhorando a irrigação sangüínea e intensificando os processos metabólicos.

- Hidroterapia: Consiste na elaboração de atividades físicas dentro da água, promovendo o fortalecimento muscular e aprimorando a noção de equilíbrio.

- Massoterapia: Consiste na aplicação de massagens terapêuticas, visando o relaxamento, a desinflamação muscular e a drenagem linfática. Exemplos: *Shantalla*, polpagem, RPG, miofascioterapia, *Bobath* e *Kabat*.

- Equoterapia: Uso da equitação (montagem sobre cavalos) para o aprimoramento do equilíbrio, da coordenação motora e para o fortalecimento muscular.

4 ASPECTOS TECNOLÓGICOS

4.1 Realidade Virtual

Segundo Kalawsky (1993), a primeira experiência para a obtenção de um realismo artificial surgiu em 1956, quando o cineasta Morton Heilig desenvolveu um sistema baseado em vídeos chamado sensorama, que simulava um passeio motociclístico pela cidade de Nova York. Em 1961, a empresa Philco desenvolveu o primeiro sistema de televisão, cuja visualização das imagens era feita por meio de visores acoplados a um capacete. O capacete possuía um rastreador de posição, permitindo ao usuário controlar remotamente uma câmera de televisão a partir dos movimentos de sua cabeça (KALAWSKY, 1993). Quatro anos mais tarde, Ivan Sutherland, considerado por muitos como o precursor da realidade virtual, desenvolveu o primeiro vídeo-capacete que possibilitava a interação do usuário com gráficos gerados por computador (HAND, 1994).

O termo realidade virtual, no entanto, foi definido por Jaron Lanier (fundador da VPL Research Inc.), no início da década de 80, para diferenciar as simulações tradicionais feitas por computador das simulações que envolviam múltiplos usuários em um ambiente compartilhado (ARAÚJO, 1996).

Hoje em dia, não há uma definição única para a realidade virtual, pois os desenvolvedores de *softwares* e pesquisadores tendem a defini-la segundo suas próprias experiências. Hancock (1995) a define como a forma mais avançada de interfaceamento entre o usuário e o computador. Latta e Oberg (1994) a consideram uma interface entre o mundo real e o virtual, por meio da qual os usuários visualizam e interagem com os elementos virtuais. Hand (1994), por sua vez, já trata a realidade virtual como um paradigma, onde técnicas computacionais são empregadas para possibilitar a interação do usuário com algo irreal, mas que pode ser considerado real durante o processo interativo.

De forma geral, a realidade virtual é caracterizada pela existência de um ambiente tridimensional gerado por computador, no qual o usuário pode interagir, em tempo real, com seus elementos virtuais. Neste contexto, a realidade virtual busca impressionar os sentidos do usuário de forma que o irreal se pareça real.

4.1.1 Conceitos de realidade virtual

4.1.1.1 Ambientes virtuais

Um ambiente de realidade virtual ou simplesmente ambiente virtual é um cenário tridimensional criado por computador e exibido, em tempo real, por meio de técnicas de computação gráfica.

4.1.1.2 Sistemas imersivos e não imersivos

Em um sistema de realidade virtual imersivo, as cenas virtuais são apresentadas de forma que o usuário se sinta dentro no cenário tridimensional. A sensação de imersão é obtida com o uso de dispositivos especiais, como capacetes de visualização ou cavernas virtuais.

Em um sistema de realidade virtual não imersivo, as cenas virtuais são exibidas na tela de um monitor ou dispositivo semelhante. É importante esclarecer que o fato do usuário não se sentir dentro do ambiente virtual, não significa que ele não esteja interagindo com o mesmo. O baixo custo e a fácil utilização fazem dos monitores uma opção relevante. Há, contudo, uma forte tendência em se utilizar a realidade virtual imersiva na grande maioria das aplicações futuras (NETTO *et al.*, 2002).

Embora a classificação acima esteja relacionada ao fator visual, a rigor, o termo imersão assume uma abordagem bem mais ampla, já que os demais sentidos corporais também podem ser explorados (ROBERTSON *et al.*, 1993). Assim, a não imersão caracterizada pelo uso de monitores pode ser ampliada com o uso, por exemplo, de caixas de sons estrategicamente posicionadas para fornecer uma sonorização tridimensional.

4.1.1.3 Interatividade e envolvimento

A interação está relacionada à capacidade do computador detectar as entradas fornecidas pelo usuário e, em função destas, modificar, em tempo real, os modelos do ambiente virtual.

A idéia de envolvimento, por sua vez, está associada ao grau de motivação para o engajamento de uma pessoa em determinada atividade. Neste contexto, um bom ambiente

virtual deve apresentar alta qualidade gráfica, animações, simulações, sons, objetivos ou outros recursos que favoreçam o entretenimento do usuário (ARAÚJO, 1996).

4.1.1.4 Estereoscopia

Quando alguém olha para uma determinada cena, devido ao distanciamento natural existente entre os olhos, cada olho a enxerga sob um ponto de vista diferente. Esta pequena diferença visual entre as imagens captadas pelos olhos fornece ao cérebro dados visuais que o possibilitam determinar a profundidade dos elementos da cena. A este processo dá-se o nome de estereoscopia ou visão binocular.

A estereoscopia também pode ser obtida artificialmente, por meio de procedimentos ou sistemas que fazem com que o observador veja um par de imagens, uma para cada olho, obtidas de uma mesma cena sob pontos de vistas distintos (DURAND, 2004).

4.1.2 Principais dispositivos de realidade virtual

4.1.2.1 Monitores

Embora não proporcionem a imersão visual do usuário no ambiente tridimensional, os monitores são ainda os dispositivos mais utilizados para a visualização gráfica. Existem três tipos básicos de monitores disponíveis no mercado: os *CRTs*, os *LCDs* e os monitores com telas de plasma.

Os *CRTs* (*Cathode Ray Tube*) são os monitores comuns, usados pela grande maioria dos usuários de computador. São basicamente constituídos de um canhão de elétrons, um conjunto de placas defletoras e uma tela recoberta com uma camada de fósforo. Antes de colidirem com a tela fosforescente, os elétrons emitidos pelo canhão são desviados para a posição desejada devido ao campo elétrico produzido pela diferença de potencial entre placas defletoras (CARRARA, 2002).

Os monitores *LCDs* (*Liquid Crystal Display*) possuem telas de cristal líquido que não emitem luz e necessitam de uma iluminação externa para que sejam visualizadas adequadamente. Seu princípio de funcionamento baseia-se no fato de certos cristais alterarem o ângulo de polarização da luz quando submetidos a uma diferença de potencial elétrico. Os *LCDs* possuem um ângulo de observação bastante limitado, porém consomem

pouca energia e podem ser fabricados em dimensões bastante reduzidas, o que os tornam ideais para a implantação em computadores portáteis (*notebooks*) (CARRARA, 2002).

Os monitores com telas de plasma dispõe de pequenas células contendo gases, que emitem luzes quando submetidos a uma determinada tensão elétrica. Assim como os *LCDs*, as telas de plasma também podem ser fabricadas em dimensões muito reduzidas, porém a um custo significativamente maior (CARRARA, 2002). A figura 4.1 mostra um exemplo de cada um destes monitores.

Figura 4.1 - Exemplos de monitores *CRT*, *LDC* e com tela de plasma

4.1.2.2 *Shutter Glasses*

Segundo Johansson (2000), cada imagem formada na retina do olho humano persiste por cerca de um décimo de segundo. Os *shutter glasses* são óculos especiais que exploram esta característica ocular para proporcionar ao usuário a visão estereoscópica. Seu princípio de funcionamento consiste em bloquear rapidamente a visão de cada olho, de forma alternada e sincronizada com a seqüência de imagens fornecidas pelo computador. Assim, enquanto a imagem relativa ao olho esquerdo é apresentada, o campo de visão do olho direito permanece tapado e o do esquerdo não. Se a imagem apresentada for relativa ao olho direito, então ocorre o processo inverso. Como a freqüência de apresentação das sucessivas imagens é alta, a seqüência é vista de forma contínua (NETTO *et al.*, 2002).

Pode-se também obter a visão estereoscópica com o uso de óculos com filtros coloridos. Neste processo, a distinção visual entre as imagens relativas aos olhos esquerdo e direito é obtida por meio da exibição e filtragem das imagens em cores complementares, como, por exemplo, o vermelho com o azul ou o vermelho com verde. Assim, se as imagens relativas aos olhos direito e esquerdo são apresentadas respectivamente nas cores vermelho e azul, o filtro do olho direito deverá ser azul enquanto que o do olho esquerdo deverá ser vermelho, de forma que a imagem esquerda seja notada somente pelo olho

esquerdo e a imagem direita somente pelo olho direito. Apesar da exibição colorida, a imagem estereoscópica é vista em preto e branco. Embora provoquem um certo cansaço visual, os óculos com filtros coloridos são muito baratos e possibilitam que várias pessoas participem simultaneamente da experiência de realidade virtual (VINCE, 1995). A figura 4.2 ilustra um exemplo de cada dispositivo estereoscópico.

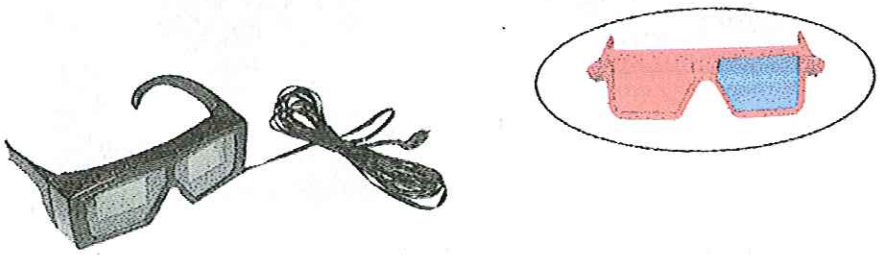


Figura 4.2 - *Shutter glasses* (à esquerda) e óculos com filtros coloridos (à direita)
Fonte: *Virtual & Really* (2002)

4.1.2.3 *HMD* e *BOOM*

Por possibilitar o total isolamento visual do usuário em relação ao mundo real, o *HMD* (*Head Mounted Display*) é o dispositivo de interfaceamento mais empregado nos sistemas de realidade virtual. A visualização do ambiente tridimensional é feita por meio de duas pequeninas telas de cristal líquido, uma para cada olho, integradas a um par de lentes especiais, que possibilitam a focalização das imagens (já que estas são exibidas muito próximas ao olhos).

O *HMD* não é somente um dispositivo de saída, mas também um dispositivo de entrada de dados. Ele contém sensores eletromagnéticos que detectam a orientação da cabeça do usuário em relação ao cenário tridimensional e a transmitem, na forma de dados, para o computador.

O *BOOM* (*Binocular Omni-Oriented Monitor*), também conhecido como *Head-Coupled Display*, é um dispositivo de visualização semelhante ao *HMD*, porém acoplado a um braço mecânico articulado e sensorizado, que possibilita a captura do movimento da cabeça em até seis graus de liberdade (BOLAS, 1994).

Embora limitado, o *BOOM* é mais barato do que o *HMD* e proporciona ao pesquisador uma fácil transição entre a visualização imersiva (*BOOM*) e a não imersiva

(monitor), o que o torna bastante atrativo no meio científico (MACHADO *et al.*, 2002). A figura 4.3 mostra um exemplo de um *HMD* e de um *BOOM*.

Figura 4.3 - Exemplos de um *HMD* (à esquerda) e de um *BOOM* (à direita)
Fontes: *Inition* (2004) e *University of Michigan* (2003)

4.1.2.4 *CAVE*

Uma *CAVE (Cave Automatic Virtual Environment)* consiste em uma pequena sala na qual as paredes, o teto e o chão são telas de projeção semitransparentes. Os projetores são posicionados externamente à *CAVE*, um para cada tela, e as imagens projetadas se combinam para gerar um cenário virtual totalmente imersivo. Além de proporcionar a imersão sem o uso de capacetes de visualização, uma *CAVE* permite que várias pessoas participem simultaneamente do processo imersivo. É possível ainda projetar imagens estereoscópicas, sendo necessário, neste caso, o uso de óculos especiais. Como complemento, estes sistemas podem apresentar dispositivos acústicos e de rastreamento.

Nota-se, no entanto, que uma *CAVE* requer uma estrutura computacional bastante avançada, já que são processados, em tempo real, todos pares estereoscópicos de imagens (um para cada tela), os sons e demais procedimentos envolvidos na interação do usuário com o ambiente. (NETTO *et al.*, 2002).

O Laboratório de Sistemas Integráveis da USP possui uma *CAVE* (a única existente na América Latina), freqüentemente usada no desenvolvimento de pesquisas acadêmicas e industriais (LSI, 2002).

Figura 4.4 - Exemplos de *CAVEs* para visualização de projetos de engenharia
Fontes: *University of Michigan* (2004) e SGI (2003)

4.1.2.5 Dispositivos de *force-feedback* e *tactile-feedback*

Os dispositivos de *force-feedback* e *tactile-feedback*, também conhecidos como *haptics interfaces*, caracterizam-se por uma espécie de vestimenta eletromecânica, capaz de sensibilizar o usuário por meio do contato físico.

Nos sistemas de realidade virtual, os dispositivos de *force-feedback* buscam simular a resistência à deformação (elasticidade) de um objeto virtual, o seu peso, as suas características inerciais ou quaisquer outras ações que exigem contração muscular por parte do usuário. Já os dispositivos de *tactile-feedback* procuram sensibilizar a pele do usuário, simulando efeitos como a rugosidade de uma superfície virtual, forma geométrica e temperatura (BURDEA, 2000).

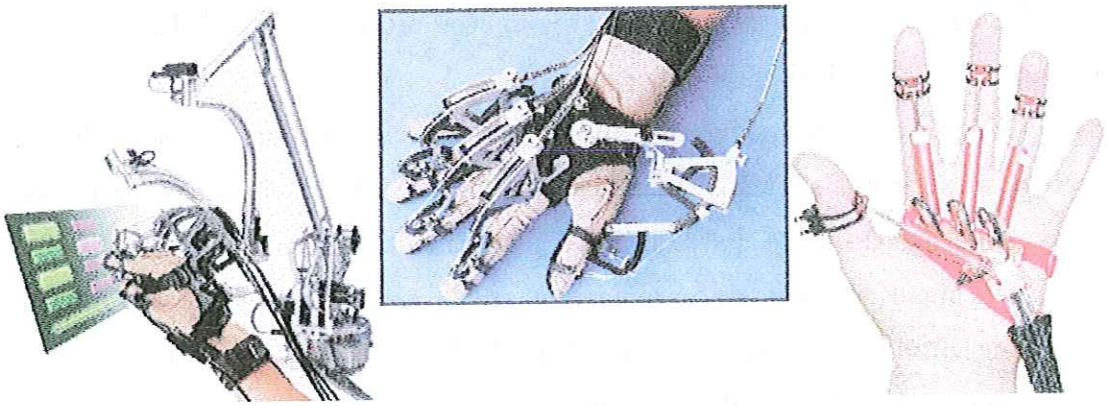


Figura 4.5 - Dispositivos de *force-feedback* acoplados em mãos
Fontes: *Immersion* (2003) e ASME (2003)

4.1.2.6 *Dataglove*

Este dispositivo consiste em uma luva sensorizada, por meio da qual é possível capturar os movimentos da mão do usuário (STURMAN e ZELTZER, 1994). Normalmente, a aquisição dos dados do movimento é obtida com o uso de potenciômetros ou sensores de fibra ótica. Estes últimos vêm sendo mais empregados, pois reduzem a dimensão do dispositivo. Seu princípio de funcionamento baseia-se na redução das intensidades luminosas devido aos dobramentos das fibras, que ocorrem na região das juntas dos dedos. A intensidade de luz que atravessa cada fibra é captada por uma foto-célula, que por sua vez emite um sinal de tensão proporcional ao dobramento da mesma (MACHADO *et al.*, 2002). A figura 4.6 mostra alguns tipos de *Dataglove*.

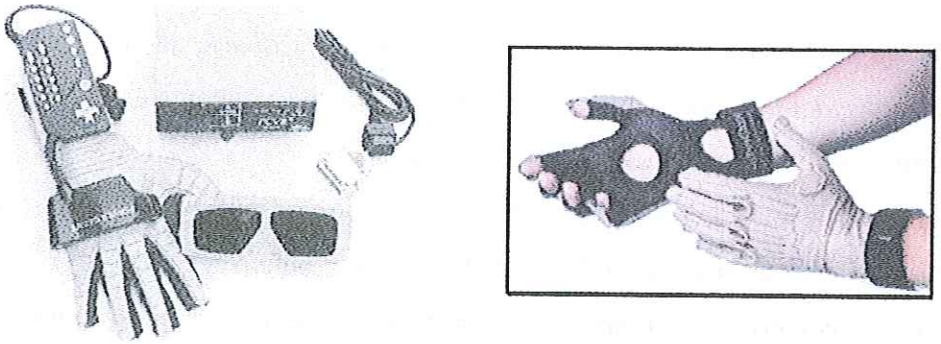


Figura 4.6 - Exemplos de *Datagloves* usadas em sistemas de realidade virtual
Fontes: UNB (2003) e *Immersion* (2003)

4.1.2.7 Sensores que detectam sinais biológicos

Sensores que detectam entradas biológicas processam atividades indiretas, como comandos de voz e sinais elétricos musculares.

Em sistemas de realidade virtual, o reconhecimento de comandos de voz pode facilitar a execução de tarefas no ambiente tridimensional, principalmente quando as mãos estiverem ocupadas ou inacessíveis a um determinado dispositivo de entrada.

Os impulsos elétricos provenientes da atividade muscular podem ser detectados com o uso de eletrodos colocados sobre ou sob a pele do usuário. O processamento destes sinais possibilita a determinação do posicionamento do usuário no cenário virtual. Pode-se, inclusive, capturar o movimento dos olhos com o uso desta técnica (PIMENTEL e TEIXEIRA, 1995).

4.1.3 Telepresença e realidade aumentada

Embora não estejam diretamente relacionadas ao desenvolvimento deste projeto, a telepresença e a realidade aumentada (*Augmented Reality*) vêm sendo muito pesquisadas nos últimos anos.

Um sistema de telepresença utiliza dispositivos robóticos, câmeras de vídeo e microfones para projetar as capacidades motoras e sensoriais do usuário a um ambiente remoto, permitindo que o mesmo interaja com um ambiente real fisicamente separado do local de controle (onde o usuário está). Operando os dispositivos de controle, o usuário envia ações aos dispositivos robóticos e recebe destes um *feedback* sensorial, contribuindo para a sensação de presença no ambiente remoto (ARAÚJO, 1996). Controle de robôs, exploração planetária e aplicações em medicina são exemplos de pesquisas em desenvolvimento.

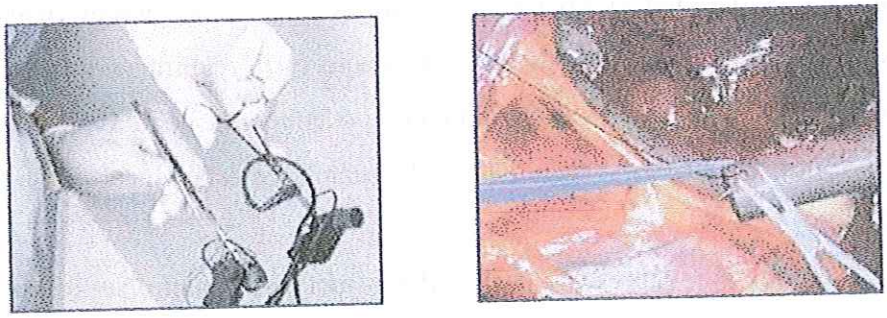


Figura 4.7 – Cirurgia sendo executada com técnicas de telepresença
Fonte: Telemedicina (2003)

A realidade aumentada, por sua vez, busca suplementar ou enriquecer o cenário real com informações visuais geradas por computador. Neste sentido, o usuário utiliza um capacete especial, dotado de um visor de cristal-líquido semitransparente (no qual são exibidas as imagens virtuais), que o permite visualizar as imagens reais e virtuais simultaneamente. É importante que a composição das imagens seja feita com precisão. A figura 4.8 ilustra uma aplicação da realidade aumentada na área médica.

Figura 4.8 – Simulação de cirurgia usando técnicas de realidade aumentada
Fontes: *Microvision* (2002)

4.1.4 Aplicações da realidade virtual em terapias

A realidade virtual tem grande aplicação no tratamento de pacientes que apresentam algum tipo de deficiência. Os pacientes são imersos em ambientes virtuais, onde executam tarefas específicas relacionadas ao treinamento ou à reabilitação das funções fisiológicas associadas à deficiência. Nestes ambientes, o terapeuta pode avaliar o desempenho ou a precisão dos movimentos executados pelo paciente e estabelecer diagnósticos ao longo do tratamento.

Pessoas que apresentam distúrbios motores podem se beneficiar dos recursos tecnológicos oferecidos pela realidade virtual. O pesquisador J. P. Wann, da *Edinburgh University*, desenvolveu um sistema de *feedback* computadorizado (que opera em tempo real) para tratar crianças que sofrem de paralisia cerebral. O sistema captura os sinais musculares para possibilitar o controle de diferentes atividades dentro de um cenário virtual lúdico. Os resultados obtidos mostraram que as crianças tratadas com este sistema apresentaram um desempenho significativamente maior do que as que foram submetidas ao tratamento convencional (WANN, 1993).

Pacientes que sofrem de ataxia não conseguem combinar as informações recebidas pela visão com as recebidas pela propriocepção (percepção espacial de si mesmo), resultando em distúrbios na postura, no equilíbrio e dificuldades na coordenação dos movimentos. Para compensar suas falhas, tais pacientes freqüentemente buscam negligenciar a sua propriocepção. O *San Raffaele Hospital*, em Milão, há vários anos emprega técnicas de realidade virtual para estimular e educar a propriocepção destes

pacientes, imergindo-os em ambientes virtuais nos quais as imagens são projetadas de forma distorcida (MOLENDI e PATRIARCA, 1992).

Em mundos virtuais, pessoas com paralisia cerebral são capazes de realizar tarefas complexas usando dispositivos de entrada adequados às suas restrições motoras. Existem atualmente dispositivos de realidade virtual dotados de sensores capazes de detectar com grande precisão o mínimo movimento de algum membro ou órgão do corpo humano. Os pesquisadores Lusted e Knapp do *BioControl Systems Inc.*, criaram um dispositivo especial chamado *BioMuse*, capaz de detectar sinais biológicos humanos e transmiti-los a um computador. O Dr. *Dave Warner*, neurologista do *Advanced Technology Center, Loma Linda University*, Califórnia, emprega com sucesso o *BioMuse* em tratamentos de reabilitação motora. Pacientes com lesões medulares aprenderam a usar a realidade virtual para manipular objetos virtuais pertinentes ao tratamento fisioterápico. Em uma dessas aplicações, um paciente tetraplégico conseguiu tocar instrumentos musicais eletrônicos apenas com pequenos movimentos musculares da face e do pescoço. Outros conseguiram deslocar objetos virtuais usando os movimentos dos olhos. Estima-se que futuramente, pessoas deficientes e normais atuem e se comuniquem em ambientes virtuais, sem que, contudo, a deficiência seja notada (KUHLEN e DOHLE, 2000).

Os Centros *Hines Rehabilitation* e *R&D* desenvolveram um sistema de realidade virtual que possibilita ao deficiente motor testar um protótipo de cadeira de rodas antes mesmo da sua fabricação. Usando um *HMD*, o paciente pode movimentar-se com a cadeira virtual em salas virtuais e verificar o seu potencial de obstrução, como a passagem por portas estreitas, aclives, declives etc (KUHLEN e DOHLE, 2000).

Segundo Kuhlen e Dohle (2000), pessoas com pouca ou nenhuma capacidade de comunicação verbal podem usar os dispositivos de realidade virtual para obterem uma nova forma de expressão. Usando-se um *DataGlove*, por exemplo, pode-se gravar e converter gestos em falas ou textos escritos.

Disfunções na visão, especialmente o estrabismo, são estudadas por pesquisadores da *San Jose University*. Eles utilizam o *BioMuse* para determinar a distorção angular entre os eixos oculares e, com base nesta informação, projetam imagens em um ambiente virtual de forma que seja necessário forçar (exercitar) o alinhamento dos olhos para que os objetos virtuais consigam ser focalizados (KUHLEN e DOHLE, 2000).

Um grupo de pesquisadores da Universidade da Carolina do Norte, desenvolveu um mundo virtual para estimular a aprendizagem em crianças autistas. Os resultados foram espetaculares. Após poucas semanas imersas nesse ambiente virtual, as crianças já haviam adquirido um vocabulário com mais de 200 palavras. A explicação se deve ao fato do mundo virtual não possuir seres humanos, apenas objetos lúdicos e pouco ameaçadores, porém capazes de ensinar algo à criança (SABBATINE, 1993).

A empresa canadense IREX (2003) desenvolveu um sistema de realidade virtual que possibilita a prática de atividades físicas, em tratamentos de reabilitação, de uma forma bastante lúdica (figura 4.9). O paciente é posicionado em frente a um grande painel verde e filmado por uma câmera digital conectada a um computador. O fundo verde é computacionalmente removido e substituído por um cenário virtual relacionado à atividade física. A imagem do paciente é processada de forma que o programa permita a sua interação, em tempo real, com os elementos do ambiente virtual. Após ser processada, cada a imagem é exibida ao paciente através de um grande monitor.

A característica marcante deste sistema é o fator lúdico, que proporciona entusiasmo e motivação na execução das atividades envolvidas no tratamento.

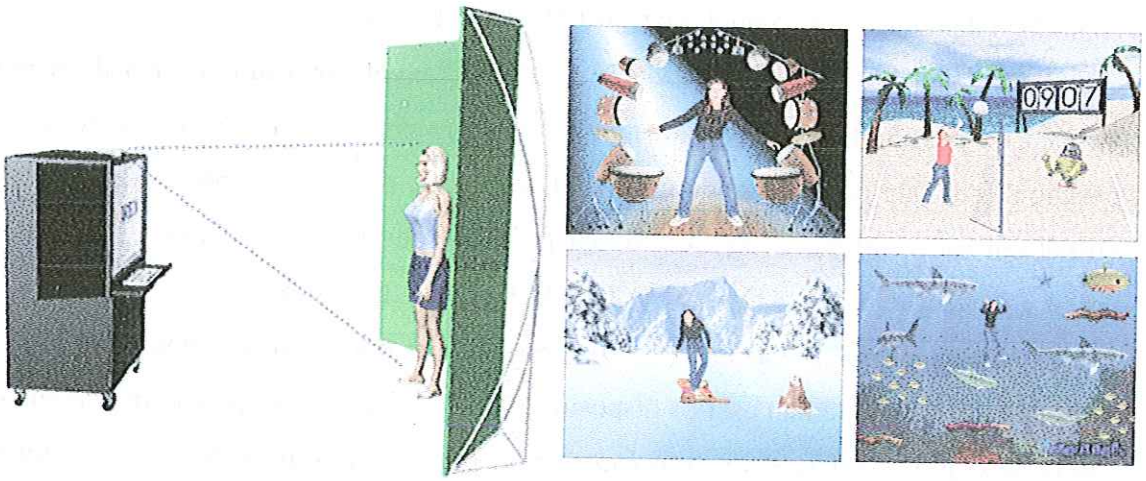


Figura 4.9 - Sistema de realidade virtual desenvolvido pela empresa IREX
Fonte: IREX (2003)

A importância da realidade virtual para os terapeutas está relacionada à possibilidade de avaliação e manipulação das informações gráficas obtidas, que os auxiliam na elaboração de diagnósticos mais precisos. À medida que os pacientes executam as

atividades motoras, o sistema de realidade virtual permite ao terapeuta visualizar, em tempo real, as trajetórias dos movimentos do paciente no espaço virtual, a partir de pontos de vista distintos. O Laboratório de Marcha Helena Pereira de Moraes, na AACD (2004), dispõe de equipamentos computacionais sofisticados que efetuam análises tridimensionais das posições dos segmentos corpóreos durante a marcha (o andar) do paciente. Trata-se de um sistema de visão computacional para a captura dos movimentos em tempo real, capaz de fornecer dados visuais que possibilitam ao terapeuta estabelecer um diagnóstico seguro e preciso sobre qual ou quais grupos musculares do paciente estão comprometidos (figura 4.10).

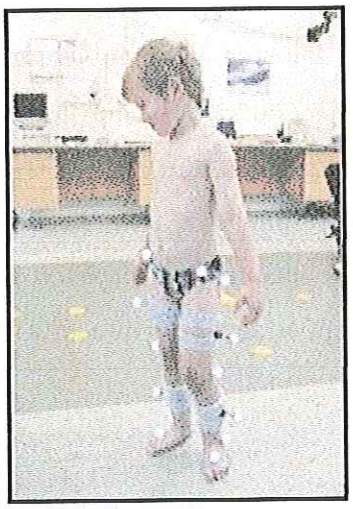


Figura 4.10 - Análise de marcha por meio de um sistema de visão computacional
Fonte: ***Vicon Motion Systems*** **(2003)**

Os pesquisadores do *Human–Machine Interface Laboratory*, na *Rutgers University*, EUA, desenvolvem luvas especiais (dotadas de atuadores que possibilitam o *force-feedback*) para auxiliar na reabilitação de pessoas que apresentam déficit motor nas mãos (decorrente de acidentes ou de cirurgias). A luva é conectada a um computador, que solicita, por meio de uma interface gráfica, a execução de diversas tarefas virtuais envolvendo habilidade, coordenação, velocidade e força. Pode-se, por exemplo, apertar uma bola virtual e sentir, por meio dos atuadores, a sua resistência à deformação. Segundo G. Burdea (2002), a principal vantagem está na facilidade para se coletar os dados enquanto o paciente se exercita.

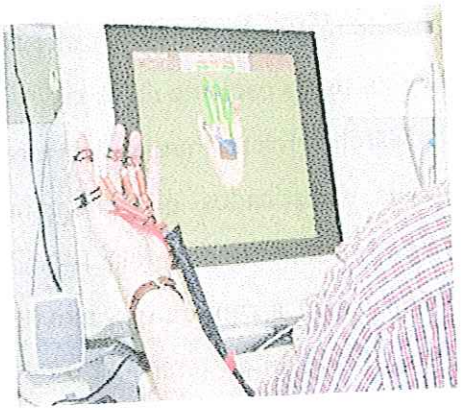


Figura 4.11 – Paciente exercitando-se com a luva de *force-feedback* desenvolvida na *Rutgers University*
Fonte: BURDEA (2002)

Burdea e sua equipe também desenvolveram um dispositivo de *force-feedback* conhecido como *Rutgers Ankle*, que auxilia no processo de reabilitação de pessoas que possuem um comprometimento na movimentação dos tornozelos (decorrente de lesões ou cirurgias). O dispositivo corresponde a uma pequena plataforma com seis graus de liberdade, dotada de atuadores pneumáticos, capazes de fornecer resistência (controlada) a ações externas. O paciente apóia seu pé sobre esta plataforma e executa os exercícios solicitados pelo computador.

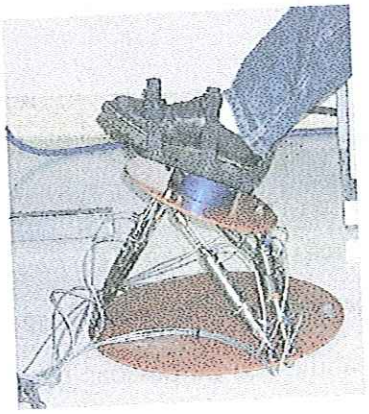


Figura 4.12 - Plataforma *Rutgers Ankle*
Fonte: CAIP (2003)

Segundo Burdea (2003), os dispositivos de *force-feedback* devem proporcionar a execução dos exercícios de reabilitação de forma divertida, já que muitos deles são difíceis de serem feitos. Para garantir este divertimento, são implementados jogos computacionais com os quais o paciente interage. Um dos jogos desenvolvidos consiste no controle de um avião por meio do dispositivo *Rutgers Ankle*. O computador simula um espaço aéreo

contendo vários aros flutuando em diferentes posições, dentro dos quais o avião deve passar. Há a contagem de pontos e de tempo, estimulando o paciente a superar suas marcas anteriores. Quando o paciente apresentar-se mais capacitado e habilidoso, pode-se colocá-lo em um nível de controle mais difícil, onde o mesmo voará sob turbulências e nevoeiros. Burdea (2003) afirma que o uso de jogos contribui para que o paciente realmente aprenda a coordenar o movimento do(s) pé(s), e não somente exerça força durante o tratamento.

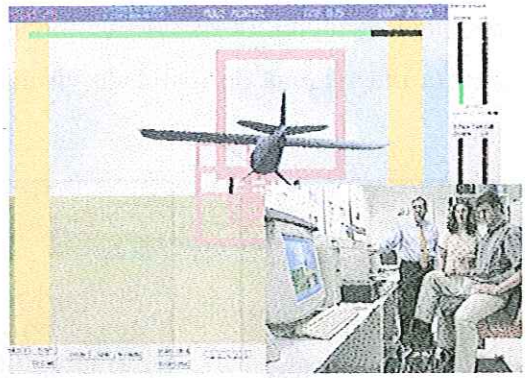


Figura 4.13 - Controle de uma avião virtual por meio da plataforma *Rutgers Ankle*
Fonte: CAIP (2003)

Os dispositivos de *force-feedback GaitMaster1* e *GaitMaster2*, desenvolvidos pelo *VR Lab, Tsukuba University*, Japão (2002), possibilitam simular caminhadas sobre terrenos virtuais irregulares, como areia, lama, pedregulhos e escadas. A figura 4.14 mostra estes dispositivos.

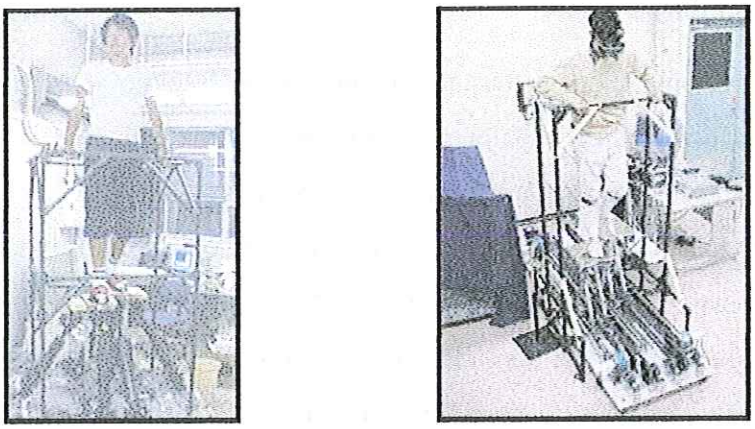


Figura 4.14 – Dispositivos *GaitMaster1* (à esquerda) e *GaitMaster2* (à direita)
Fonte: *VR Lab.* (2002)

Os pesquisadores da *Delft University of Technology* (2003), Holanda, desenvolvem sistemas de realidade virtual que auxiliam no tratamento de pessoas com algum tipo de fobia (claustrofobia, acrofobia, antropofobia etc). O tratamento consiste em imergir o paciente em um cenário virtual, onde são simuladas as situações causadoras da fobia. Por exemplo, pessoas que têm medo de altura (acrofobia) podem ser inseridas em um elevador virtual desprovido de paredes. A cada seção da terapia, o elevador coloca o paciente a uma maior altitude, enquanto o médico monitora os seus batimentos cardíacos e a sua pressão arterial. A figura 4.15 mostra um sistema de realidade virtual para tratamento de pessoas com acrofobia.

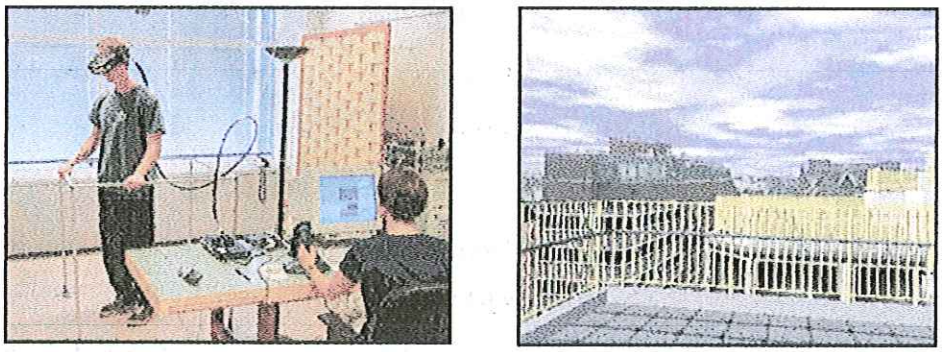


Figura 4.15 - Sistema de realidade virtual para tratamento de acrofobia
Fonte: *Delft University of Technology* (2003)

4.2 Jogos computacionais

Jogos computacionais despertam um grande fascínio nas crianças, uma vez que possibilitam interação e entretenimento com um mundo até então desconhecido, desafiador e repleto de fantasias. Estas características rapidamente chamaram a atenção de pedagogos e pesquisadores educacionais, que viram nos jogos de computador uma ferramenta com um enorme potencial para o ensino. Os jogos educacionais baseiam-se em uma abordagem autodirigida, onde a criança aprende por si só, por meio das descobertas provenientes de sua interação com o cenário virtual. O professor, neste caso, assume o papel de moderador ou mediador do processo, apenas fornecendo orientações e selecionando os *softwares* adequados e condizentes com sua prática pedagógica. Ao invés de uma simples coletora de informações, quando joga, a criança intrinsecamente pesquisa, seleciona, confronta informações e elabora metodologias em busca dos resultados desejados.

O sucesso do uso de jogos computacionais como ferramenta de estímulo ao processo educacional contribui para que outras possibilidades sejam exploradas e abre campo para a aplicação no tratamento de crianças portadoras de necessidades especiais.

Santarosa *et al.* (2002) destaca que atividades lúdicas e jogos computacionais possibilitam diagnosticar e trabalhar a percepção visual e auditiva das crianças com dificuldades na aprendizagem, favorecendo um melhor desempenho escolar, a construção da leitura e uma mudança positiva na sua auto-estima. Antunha (2002), por sua vez, afirma que os jogos computacionais contribuem para o desenvolvimento integral do sistema nervoso em seus aspectos psicomotores e cognitivos.

A terapeuta ocupacional Ana I. A. Oliveira (2004), do Laboratório de Tecnologia Assistiva da Universidade Estadual do Pará, usa jogos computacionais como ferramenta de auxílio ao tratamento de crianças portadoras de paralisia cerebral. Segundo ela, a diversidade de quadros clínicos é acentuada pela falta de recursos adaptativos ou de estimulação adequados ao tratamento de cada criança. Conseqüentemente, muitas são tidas como deficientes mentais por não apresentarem uma forma clara de expressão, fruto da falta de oportunidades experimentais necessárias ao seu desenvolvimento cognitivo. Oliveira (2004) afirma, no entanto, que a ação pode ser desencadeada com o uso de jogos muito simples, envolvendo aspectos lúdicos relacionados à aprendizagem.

Pesquisadores do Departamento de Engenharia Elétrica da EESC-USP desenvolveram um sistema baseado em jogos computacionais para auxiliar no tratamento de crianças com déficit motor nas mãos. Segundo Amate *et al.* (2004), a reabilitação dos movimentos das mãos é de fundamental importância no desenvolvimento da criança, pois um comprometimento na coordenação motora dos membros superiores pode muitas vezes impedir a escrita ou, em casos mais graves, a simples manipulação de objetos. O sistema desenvolvido por Amate *et al.* (2004) consiste em um conjunto de jogos musicais computadorizados que são ativados segundo um protocolo de movimentos pré-definido. A acessibilidade ao *software* é obtida com o uso de um dispositivo composto por uma série de manípulos, cujas posições podem ser modificadas em função da deficiência ou da evolução do paciente no decorrer do tratamento. Uma interface amigável permite ao fisioterapeuta escolher a seqüência de movimentos desejada, o número de repetições e a sua velocidade de reprodução. Testes efetuados em crianças com ataxia mostraram uma melhoria

significativa na coordenação motora das mãos, devido à motivação e ao caráter lúdico proporcionados pelos jogos (AMATE *et al.*, 2004).

Os pesquisadores do *National Rehabilitation Hospital* (2004), em Washington, também buscam desenvolver jogos e sistemas lúdicos que auxiliem no tratamento de reabilitação de crianças com deficiência motora. Um dos seus jogos, o *Catch the Flies*, busca estimular a execução rápida e coordenada dos movimentos da criança. Ao apertar um botão, a criança permite que um sapo capture as moscas que passam voando à sua frente. O terapeuta posiciona o botão em uma local estratégico, de forma que a criança tenha que executar um exercício físico (levantar e esticar o braço, por exemplo) para alcançá-lo. A interface do jogo permite que parâmetros como velocidade e precisão possam ser controlados. A figura 4.16 ilustra a interface gráfica do jogo *Catch the Flies*.

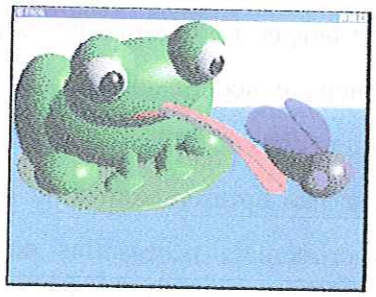


Figura 4.16 - Interface gráfica do jogo *Catch the Flies*
Fonte: *National Rehabilitation Hospital* (2004)

Outro jogo interessante desenvolvido pelo *National Rehabilitation Hospital* (2004) é o *See-Saw*, que utiliza como dispositivo de entrada um assento sustentado por cabos elásticos providos de sensores. Um personagem virtual carismático responde, por meio de animações e efeitos sonoros, aos movimentos da criança (quando a mesma se balança, pula ou estica os cabos elásticos que sustentam o assento). A figura 4.17 apresenta o dispositivo de entrada e a interface gráfica do jogo *See-Saw*.

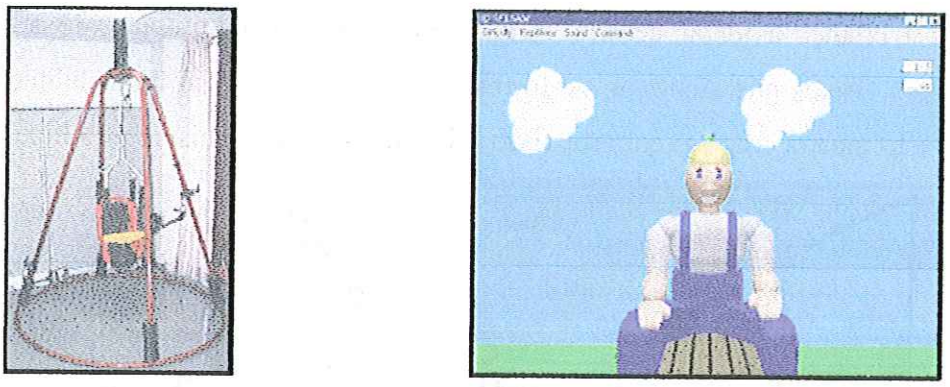


Figura 4.17 - À esquerda, o assento sustentado por cabos elásticos sensorizados. À direita, a interface gráfica do jogo *See-Saw*.

Fonte: *National Rehabilitation Hospital* (2004)

O sistema *IVES* (*Interactive Video Exercise System*), também desenvolvido pelos pesquisadores do *National Rehabilitation Hospital* (2004), permite que a criança brinque com um jogo de videogame por meio de um dispositivo de entrada especial, que requer um certo nível de força para ser acionado. Tal dispositivo é dotado de células de carga, que possibilitam registrar e avaliar, em tempo real, a força muscular exercida pela criança. A figura 4.18 mostra uma criança interagindo com o sistema *IVES*.

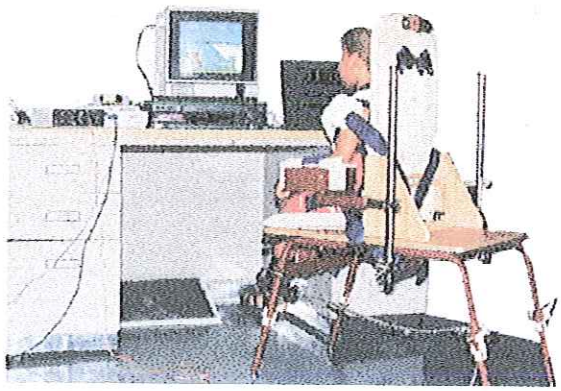


Figura 4.18 - Criança jogando um game por meio do sistema *IVES*

Fonte: *National Rehabilitation Hospital* (2004)

A pesquisadora e oftalmologista Eva Lindstedt (2003) usa jogos computacionais como ferramentas de estimulação nos tratamentos de reabilitação de crianças com déficit visual. As técnicas empregadas possibilitam às crianças acompanharem ou interagirem com sons e desenhos que se movimentam na tela. Lindstedt (2003) afirma que a qualidade visual

motiva a criança a tentar detectar o que está acontecendo, propiciando a execução de exercícios visuais que auxiliam, de forma agradável, o processo de reabilitação visual. A figura 4.19 mostra crianças com subvisão brincando com os jogos de reabilitação.

Figura 4.19 - Crianças com déficit visual brincando com jogos de reabilitação

Fonte: LINDSTEDT, E. (2003)

5 FERRAMENTAS COMPUTACIONAIS

5.1 O *Microsoft Windows*

O *Microsoft Windows* é o sistema operacional presente na grande maioria dos computadores pessoais (*PCs*) atualmente existentes. Decorrente deste fato, não é uma surpresa que existam muitas ferramentas computacionais desenvolvidas para auxiliar a implementação de jogos sobre esta plataforma, já que o público alvo dos desenvolvedores de jogos são os usuários deste sistema operacional. Como o ambiente virtual deste projeto apresenta as mesmas características de um jogo para *PC*, optou-se por desenvolvê-lo sobre a plataforma *Microsoft Windows XP*, a fim de que fosse possível trabalhar com tais ferramentas.

5.2 O *Microsoft DirectX*

Uma *Application Programming Interface* (*API*) é um conjunto de rotinas implementadas em baixo nível para promover um interfaceamento eficiente entre o *software* e o *hardware*. O *Microsoft DirectX*, por sua vez, consiste em um pacote de *APIs* criado especificamente para facilitar e otimizar o desenvolvimento de jogos e aplicações multimídia de alta performance. Suas rotinas possibilitam a implementação de gráficos bidimensionais e tridimensionais, músicas, efeitos sonoros, acesso a dispositivos de entrada e aplicações em rede.

O *Microsoft DirectX*, versão 9.0, é constituído pelos seguintes componentes:

- *DirectX Graphics*: *API* usada para a geração e manipulação de modelos gráficos. O *DirectX Graphics* é uma combinação das *APIs DirectDraw* e *Direct3D* das versões anteriores do *DirectX*.
- *DirectInput*: *API* usada para processar os dados provenientes de dispositivos de entrada (teclados, mouses e *joysticks*). Este componente também oferece suporte à tecnologia de *force-feedback* presente em *joysticks* especiais.
- *DirectPlay*: *API* que oferece recursos para a programação da "jogabilidade" via rede (*multiplayer games*).
- *DirectSound*: *API* usada no desenvolvimento de aplicações de áudio de alta performance.

- *DirectMusic: API* usada para a implementação de trilhas sonoras nos formatos *wav* e *MIDI*.
- *DirectShow: API* usada para a captura e reprodução de diferentes formatos de vídeos e sons.
- *DirectSetup: API* usada para possibilitar a instalação automática dos vários componentes do *DirectX*.
- *Direct Media Objects: API* usada para programar e decodificar os compactadores (*encoders* e *decoders*) de vídeo e áudio.

Além da *DirectX Graphics*, existem outras *APIs* e bibliotecas gráficas que oferecem recursos para a elaboração e o processamento eficiente de gráficos tridimensionais. A mais renomada é a *OpenGL*, que disponibiliza um conjunto de funções para a criação de primitivas básicas, como vértices, linhas e polígonos. De fato, qualquer malha mais complexa é formada por uma combinação destas primitivas básicas. Porém, efetuar tal modelagem por meio da programação pode se tornar uma tarefa muito árdua, ou até mesmo impraticável, caso a malha deva apresentar uma aparência orgânica, como um corpo humano ou uma planta. Nestes casos, é imprescindível a utilização de um *software* de modelagem para a geração da malha. Uma vez gerada, as posições dos seus vértices podem ser exportadas pelo programa de modelagem na forma de um arquivo específico, que será interpretado por uma rotina (implementada pelo programador) capaz de converter a estruturação deste arquivo em um código compilável (contendo as funções da *API* gráfica). A implementação desta rotina, no entanto, pode ser bastante complexa e demorada se, além das posições dos vértices, também for desejado exportar os materiais, as texturas e as animações inerentes à malha.

Por outro lado, o *DirectX* possui um formato próprio de arquivo, chamado *X file*. Por meio de *plugins* específicos, alguns programas de modelagem, como o *3d studio max*, conseguem exportar suas malhas neste formato. A grande vantagem consiste no fato da *API DirectX Graphics* já possuir complexas funções que interpretam todas as estruturas de um *X file*. Aprender a trabalhar com estas funções não é uma tarefa simples, mas ainda assim é um processo mais rápido do que implementar uma rotina interpretadora.

5.3 O *Microsoft Visual C++*

A implementação de um ambiente gráfico tridimensional, para aplicações em tempo real, requer o uso de uma linguagem de programação que ofereça recursos adequados ao processamento dos cálculos envolvidos na geração e manipulação dos modelos tridimensionais. Por serem extremamente rápidas, C e C++ são as linguagens de programação mais utilizadas no desenvolvimento destes ambientes.

O *Microsoft Visual C++ 6.0* é um compilador de grande versatilidade, cuja enorme quantidade de recursos o capacitam para a implementação de qualquer aplicação computacional. Por ser desenvolvido pela mesma fabricante do *DirectX*, este compilador apresenta um acesso otimizado às suas *APIs*, o que o transforma na ferramenta de programação mais utilizada pelos desenvolvedores de jogos e aplicações multimídia.

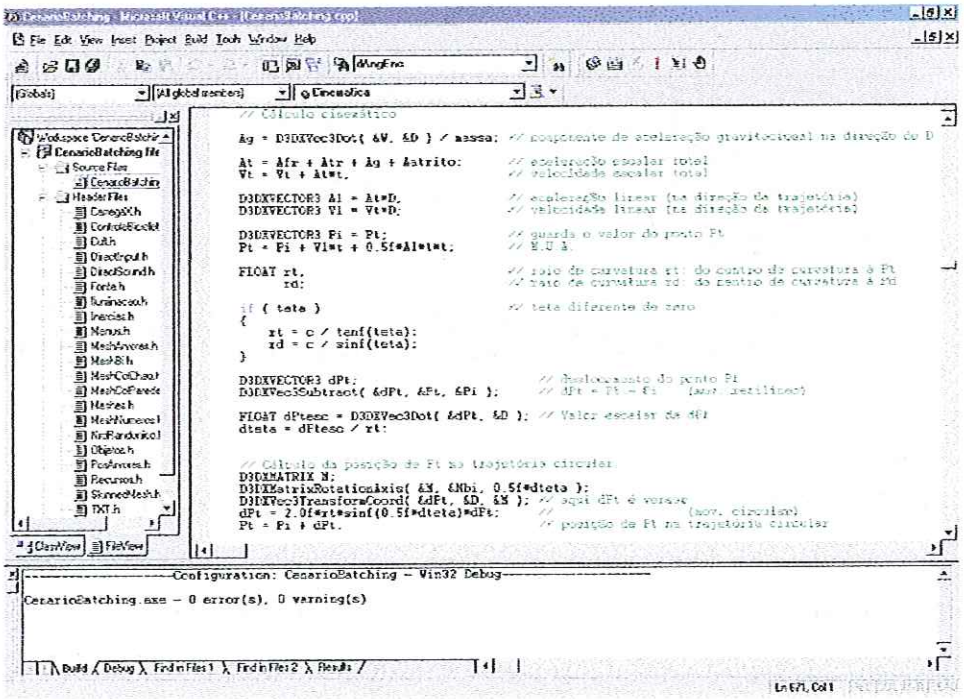


Figura 5.1 - Interface gráfica do *Microsoft Visual* C++ 6.0

5.4 O *3d studio max*

O *3d studio max*, desenvolvido pela *Discreet*, é uma poderosa ferramenta de modelagem, animação e *rendering*, amplamente utilizada no desenvolvimento de jogos, propagandas e filmes. Sua interface gráfica (figura 5.2) disponibiliza uma diversidade de recursos que possibilitam a modelagem de praticamente qualquer objeto conhecido.

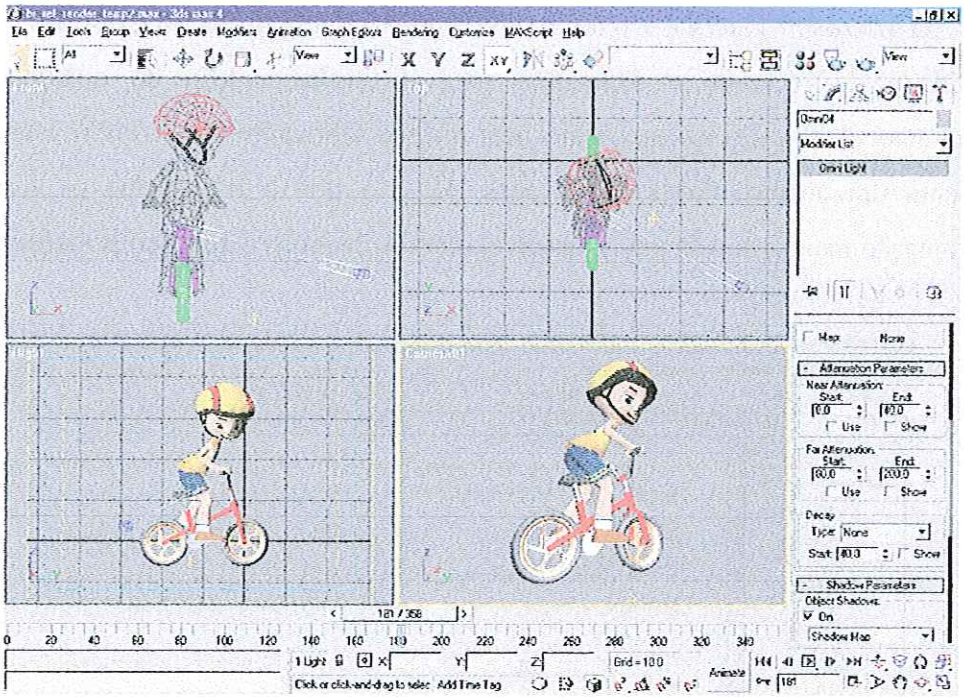


Figura 5.2 - Interface gráfica do *3d studio max 4.0*

O *3d studio max* pode exportar arquivos em diferentes formatos, dentre os quais destaca-se o *X file* (*.x*), suportado pelo *DirectX*. No entanto, a exportação no formato *.x* não está disponibilizada diretamente no *software* de modelagem, sendo necessário o carregamento de *plugins* específicos para a execução desta tarefa. Uma boa opção é o uso do *plugin Panda*, desenvolvido pela *PandaSoft*.

5.5 O *Paint Shop Pro*

O *Paint Shop Pro* é uma ferramenta de edição de imagens desenvolvida pela *Jasc Software* para atender ao mercado gráfico. Ele dispõe de uma grande variedade de recursos para a criação e tratamento de imagens. Todas as texturas do ambiente virtual deste projeto foram elaboradas com este *software*. A figura 5.3 ilustra a sua interface de trabalho.

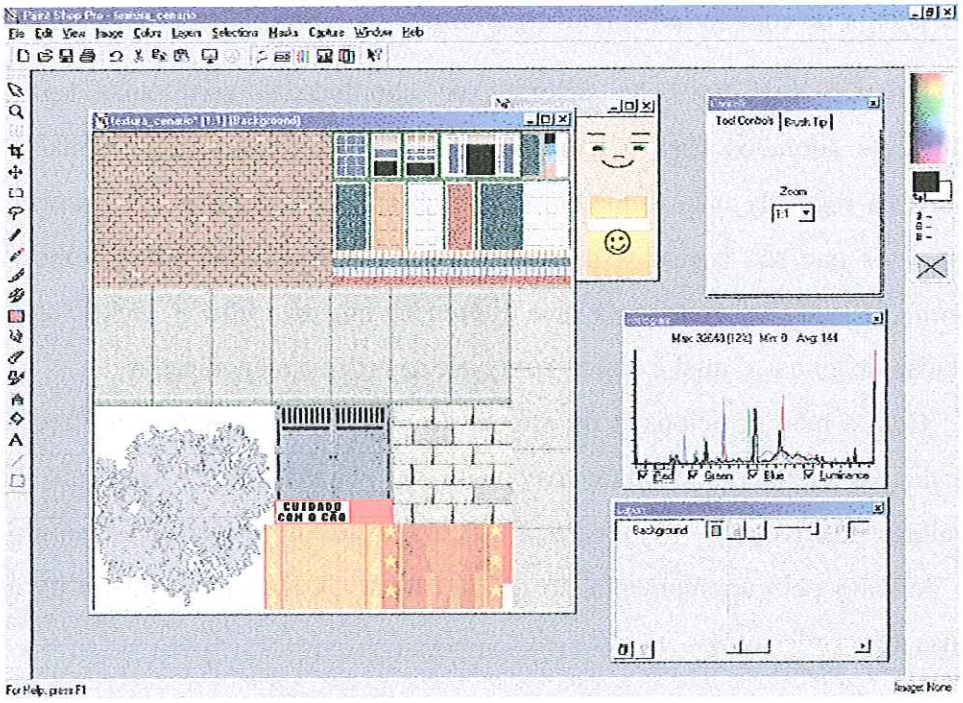


Figura 5.3 - Interface gráfica do *Paint Shop Pro 5.0*

No próximo capítulo serão apresentadas maiores informações sobre texturas.

6 FUNDAMENTAÇÃO TEÓRICA E METODOLÓGICA

Desenvolver programas com o uso de *APIs* gráficas requer uma familiarização com os princípios envolvidos na elaboração, transformação e apresentação de geometrias espaciais. A seguir são apresentados os principiais conceitos necessários à construção e manipulação das cenas tridimensionais.

6.1 Sistema de coordenadas tridimensional

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

Com a mão posicionada na origem do sistema, mantendo-se a direção do polegar perpendicular ao plano que contém os eixos *x* e *y*, aplica-se uma rotação nos demais dedos, dos valores positivos do eixo *x* para os valores positivos do eixo *y*. O sentido do eixo *z* é então definido pelo apontamento do dedo polegar. A figura 6.1 ilustra os dois tipos de sistemas de coordenadas.

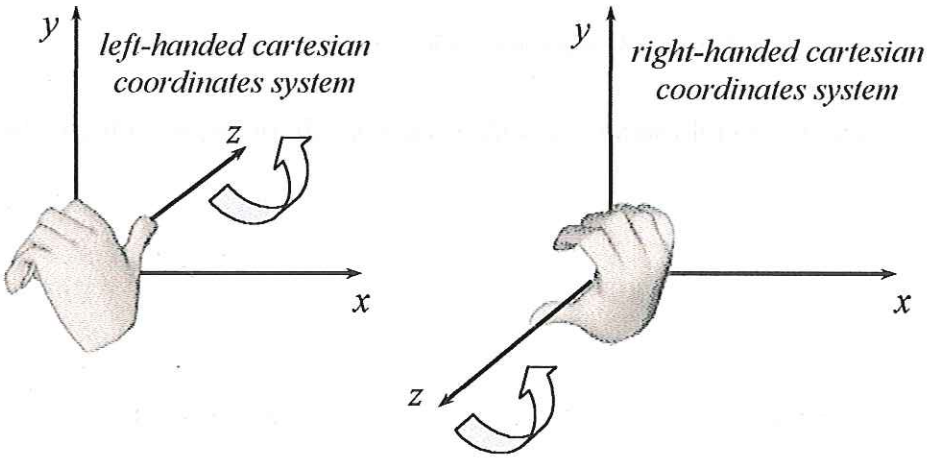


Figura 6.1 - Sistemas de coordenadas baseados nas regras da mão direita e esquerda

A rigor, um sistema de coordenadas tridimensional é um conjunto composto por três vetores linearmente independentes capazes de gerar, por combinação linear, quaisquer

outros vetores deste espaço. Caso os vetores geradores sejam unitários e ortogonais entre si, dizemos que eles constituem uma base ortonormal.

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

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

$$ {}^{U}\vec{V} = \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix}^{U} \quad \text{ou} \quad {}^{U}\vec{V} = {}^{U}(v_x, v_y, v_z) $$

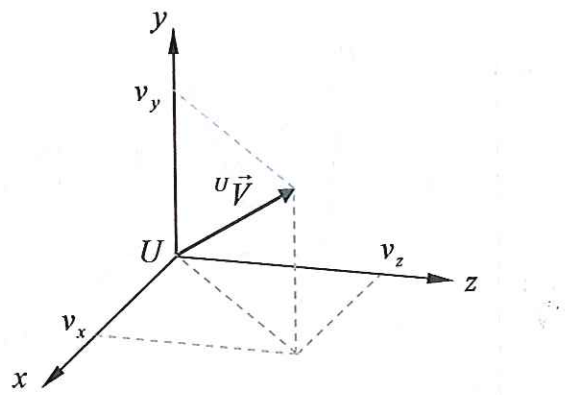


Figura 6.2 - Definição de um vetor no sistema de coordenadas universal

O *Microsoft Direct3D* adota o sistema de coordenadas baseado na regra da mão esquerda. Assim, se for desejado que uma aplicação 3D, desenvolvida em um sistema baseado na regra da mão direita (como é o caso do *3d studio max*), seja executada com o uso do *Microsoft Direct3D*, deve-se efetuar duas transformações para a conversão correta dos dados: Alterar a ordem dos vértices que definem a geração de cada face triangular e inverter o sentido do eixo *z*.

Os sistemas de coordenadas apresentados neste trabalho são baseados na regra da mão esquerda, uma vez que o ambiente virtual do projeto foi desenvolvido com o uso do *Microsoft Direct3D*.

6.2 Vértices e Faces

Qualquer objeto tridimensional gerado por computador é formado, em sua essência, por um conjunto de vértices. Cada vértice é representado por um vetor posição de coordenadas reais, geralmente expresso no sistema de coordenadas universal. Por exemplo, na figura 6.3 os vértices P_1, P_2 e P_3 são representados respectivamente pelos vetores posição ${}^{U}\vec{P}_1$, ${}^{U}\vec{P}_2$ e ${}^{U}\vec{P}_3$.

As faces de uma malha são os polígonos que a compõe, normalmente triângulos. Cada face apresenta um vetor normal, gerado pelo produto vetorial de dois vetores definidos pelos vértices da face. Desta forma, o sentido do vetor normal é dependente da escolha dos vetores usados no produto vetorial. Os vetores normais às faces são definidos principalmente para possibilitar a aplicação do *shading* (descrito no item 6.5.5). A figura 6.3 ilustra uma face e seu vetor normal.

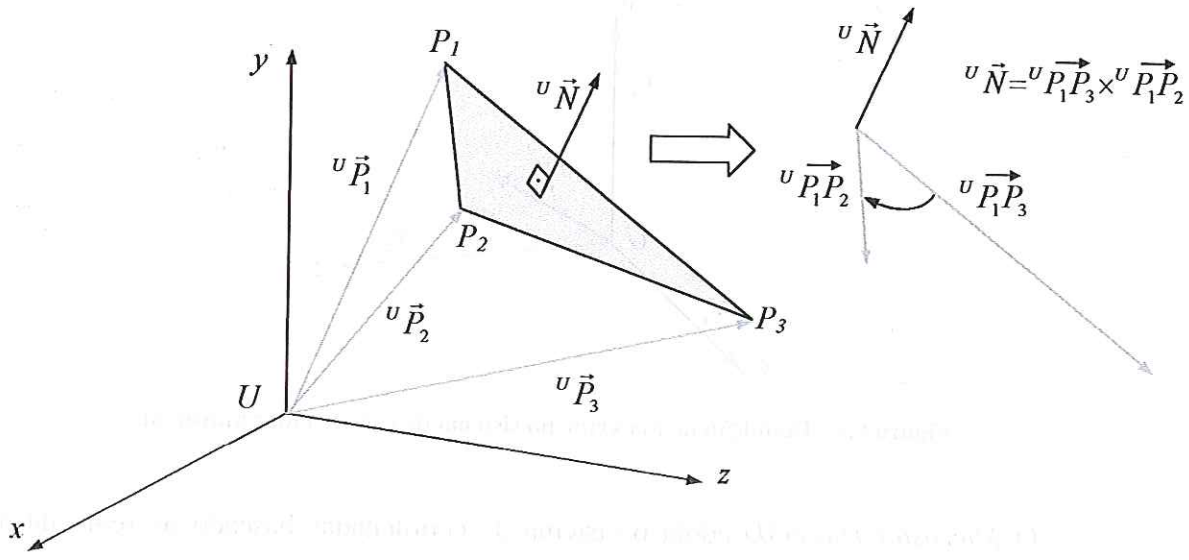


Figura 6.3 - Vértices de uma face expressos no sistema de coordenadas universal

Observe que, para definir o sentido do vetor normal, o produto vetorial segue a mesma convenção do sistema de coordenadas universal. Se, no entanto, o referencial fosse baseado na regra da mão direita, o vetor normal gerado pelo produto vetorial mostrado na figura anterior apontaria para baixo.

6.3 Transformações gráficas tridimensionais

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

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

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

6.3.1 Transformações geométricas de translação

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

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

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

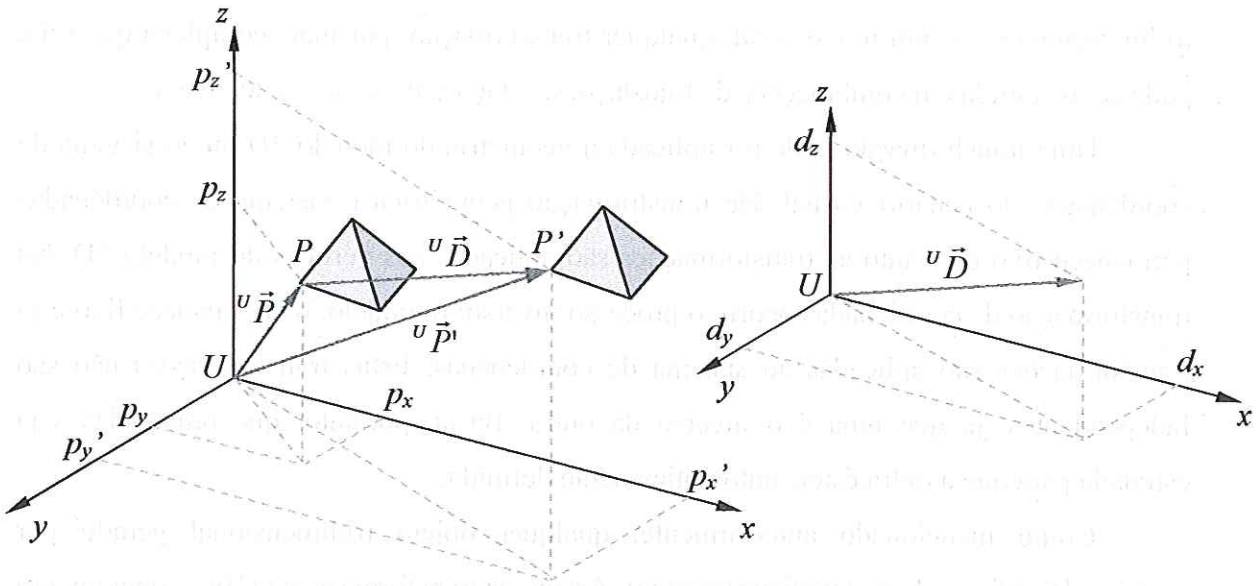


Figura 6.4 - Translação de um modelo tridimensional

6.3.2 Transformações geométricas de variação da escala

A variação da escala é aplicada com o intuito de alterar as dimensões dos objetos. Uma vez que essa transformação é feita com relação à origem do sistema de coordenadas, as distâncias dos vértices dos objetos até esta também são alteradas. Após uma transformação de variação de escala, um vértice P de um objeto assume uma nova posição P', segundo a operação:

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

onde s_x, s_y e s_z são os fatores (ou coeficientes) de escala nas direções dos eixos x, y e z respectivamente. Se estes fatores forem maiores do que a unidade, o objeto é aumentado; se forem menores, o objeto é reduzido. Coeficientes iguais proporcionam uma variação uniforme no tamanho do objeto, enquanto que coeficientes distintos proporcionam deformações no objeto. Na figura 6.5, o cubo C foi submetido a uma transformação de variação uniforme de escala, produzindo o cubo C', de tamanho maior. Já a aplicação, em C, de uma transformação de variação não uniforme de escala produziu o paralelepípedo C''. Note que as transformações alteraram as posições dos vértices do objeto. De fato, o único ponto cuja posição permanece inalterada numa transformação de variação de escala é a origem do sistema de coordenadas.

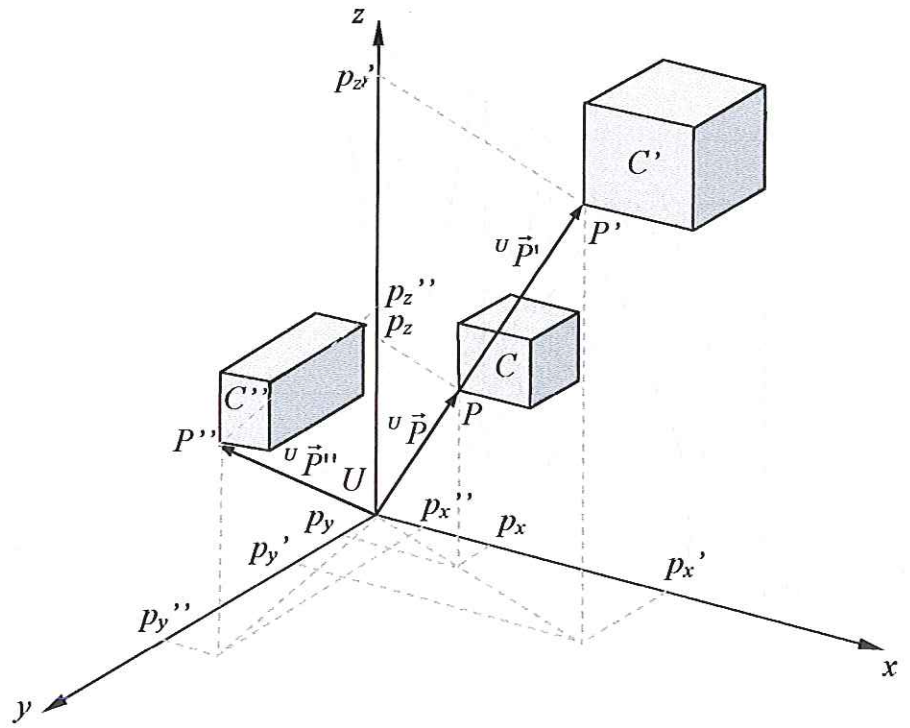


Figura 6.5 - Transformações de variação de escala

6.3.3 Transformações geométricas rotação

Na rotação, o objeto gira em torno de um determinado eixo e a distância dos vértices a este eixo permanece inalterada durante a transformação. Basicamente, qualquer eixo pode ser definido por um vetor direcional. Assim, existem infinitos eixos em torno dos quais pode-se aplicar uma rotação. É interessante, no entanto, enfatizar as transformações

de rotação em torno dos eixos cartesianos (ou canônicos), já que qualquer rotação pode ser obtida pela composição de três outras efetuadas consecutivamente sobre estes eixos.

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

Na figura 6.6, um objeto é rotacionado de um ângulo θ ao redor do eixo x, de forma que um vértice P passe a ocupar uma nova posição P'. A projeção ortogonal do vetor ${}^{U}\vec{P}$ sobre o plano yz tem módulo r e forma um ângulo φ com o eixo y.

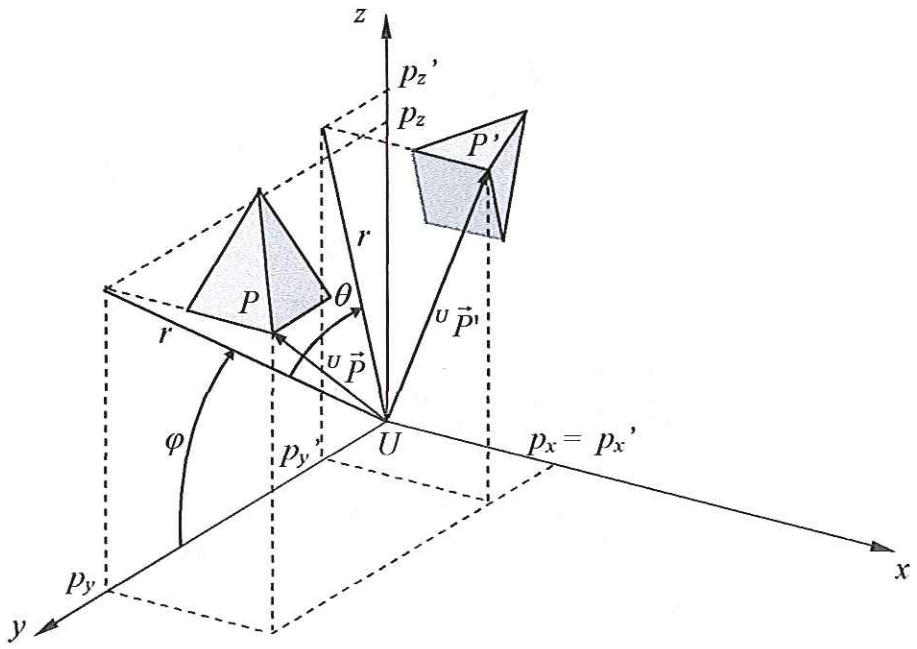


Figura 6.6 - Rotação de um objeto em torno do eixo x

Com base na figura 6.6, pode-se deduzir que:

$$\begin{cases} p_x' = p_x \\ p_y' = r\cos(\theta + \varphi) \\ p_z' = r\,\mathrm{sen}(\theta + \varphi) \end{cases}$$

Aplicando-se algumas propriedades trigonométricas obtém-se:

$$\begin{cases} p_x' = p_x \\ p_y' = r\cos\theta\cos\varphi - r\,\text{sen}\,\theta\,\text{sen}\,\varphi \\ p_z' = r\,\text{sen}\,\theta\cos\varphi + r\cos\theta\,\text{sen}\,\varphi \end{cases} \quad \text{(I)}$$

Da figura, também se deduz que:

$$\begin{cases} p_y = r\cos\varphi \\ p_z = r\,\text{sen}\,\varphi \end{cases} \quad \text{(II)}$$

Substituindo-se (II) em (I), obtém-se as coordenadas de P':

$$\begin{cases} p_x' = p_x \\ p_y' = p_y\cos\theta - p_z\,\text{sen}\,\theta \\ p_z' = p_y\,\text{sen}\,\theta + p_z\cos\theta \end{cases}$$

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

Para uma rotação de θ em torno de eixo y:

$$\begin{cases} p_x' = p_x\cos\theta + p_z\,\text{sen}\,\theta \\ p_y' = p_y \\ p_z' = -p_x\,\text{sen}\,\theta + p_z\cos\theta \end{cases}$$

Para uma rotação de θ em torno de eixo z:

$$\begin{cases} p_x' = p_x\cos\theta - p_y\,\text{sen}\,\theta \\ p_y' = p_x\,\text{sen}\,\theta + p_y\cos\theta \\ p_z' = p_z \end{cases}$$

Perceba que a coordenada referente ao eixo de rotação não é alterada pela transformação.

6.3.4 Transformações homogêneas

Por serem lineares, as transformações de rotação e variação de escala podem ser representadas por meio de um produto entre uma matriz 3x3 e um vetor de três componentes. No entanto, a transformação de translação resulta de uma soma de vetores, não admitindo um produto matricial. As transformações homogêneas surgiram com o intuito de padronizar todas as transformações na forma de matrizes, uma vez que a

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

A transformação de translação pode ser reescrita como:

$$ {}^{U}\vec{P}' = T_{\vec{D}}\ {}^{U}\vec{P} \qquad \text{ou} \qquad {}^{U}\begin{pmatrix} p_x' \\ p_y' \\ p_z' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot {}^{U}\begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix} $$

A transformação de variação de escala pode ser reescrita como:

$$ {}^{U}\vec{P}' = S_{sx,sy,sz}\ {}^{U}\vec{P} \qquad \text{ou} \qquad {}^{U}\begin{pmatrix} p_x' \\ p_y' \\ p_z' \\ 1 \end{pmatrix} = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot {}^{U}\begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix} $$

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

- rotação em torno do eixo x:

$$ {}^{U}\vec{P}' = R_{\theta,x}\ {}^{U}\vec{P} \qquad \text{ou} \qquad {}^{U}\begin{pmatrix} p_x' \\ p_y' \\ p_z' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\operatorname{sen}\theta & 0 \\ 0 & \operatorname{sen}\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot {}^{U}\begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix} $$

- rotação em torno do eixo y:

$$ {}^{U}\vec{P}' = R_{\theta,y}\ {}^{U}\vec{P} \qquad \text{ou} \qquad {}^{U}\begin{pmatrix} p_x' \\ p_y' \\ p_z' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta & 0 & \operatorname{sen}\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\operatorname{sen}\theta & & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot {}^{U}\begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix} $$

- rotação em torno do eixo z:

$$ {}^{U}\vec{P}' = R_{\theta,z}\,{}^{U}\vec{P} \qquad \text{ou} \qquad {}^{U}\begin{pmatrix} p_x' \\ p_y' \\ p_z' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta & -\operatorname{sen}\theta & 0 & 0 \\ \operatorname{sen}\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot {}^{U}\begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix} $$

Pode-se também, com base no Teorema de *Euler* (CARRARA, 2002), definir uma matriz homogênea para representar a transformação de rotação em torno de um vetor conhecido. Assim, a rotação de um ângulo θ em torno do vetor ${}^{U}\vec{E} = {}^{U}(e_x, e_y, e_z)$ pode ser representada pela transformação matricial abaixo:

$$ {}^{U}\vec{P}' = R_{\theta,\vec{E}}\,{}^{U}\vec{P} $$

ou

$$ {}^{U}\begin{pmatrix} p_x' \\ p_y' \\ p_z' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta + e_x^2(1-\cos\theta) & e_x e_y(1-\cos\theta) - e_z\operatorname{sen}\theta & e_x e_z(1-\cos\theta) + e_y\operatorname{sen}\theta & 0 \\ e_x e_y(1-\cos\theta) + e_z\operatorname{sen}\theta & \cos\theta + e_y^2(1-\cos\theta) & e_y e_z(1-\cos\theta) - e_x\operatorname{sen}\theta & 0 \\ e_x e_z(1-\cos\theta) - e_y\operatorname{sen}\theta & e_y e_z(1-\cos\theta) + e_x\operatorname{sen}\theta & \cos\theta + e_z^2(1-\cos\theta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot {}^{U}\begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix} $$

6.3.5 Composição das transformações

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

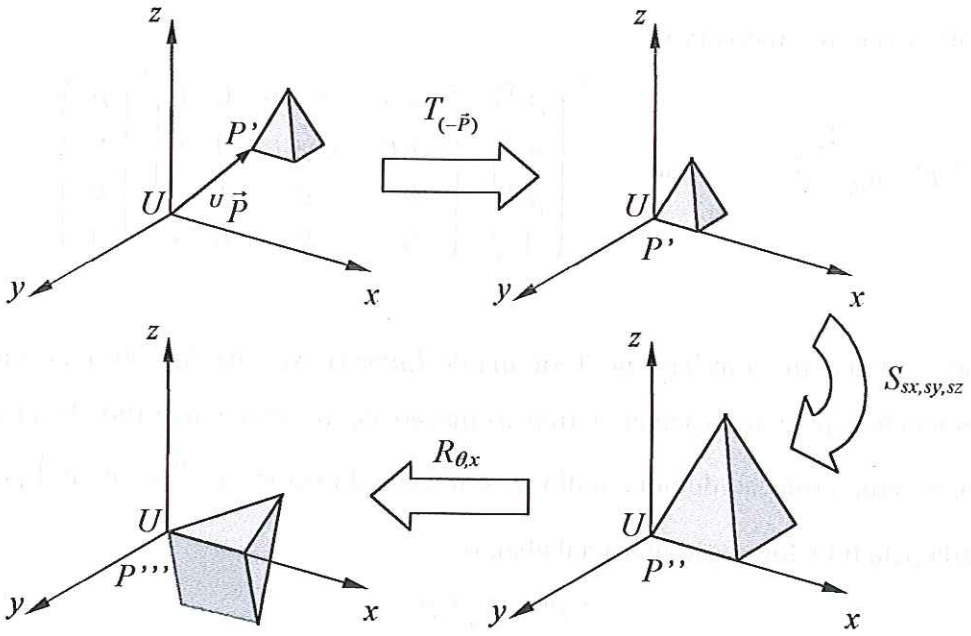


Figura 6.7 - Seqüência de transformações geométricas aplicadas em um objeto

Equacionando estas transformações tem-se:

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

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

Rotação em torno de x: $^U\vec{P}''' = R_{\theta,x}\,{}^U\vec{P}''$

Combinando-se estas equações, obtém-se a seguinte equação matricial:

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

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

$$^U\vec{P}^n = G\,{}^U\vec{P} \quad \text{onde} \quad G = G_n G_{n-1} G_{n-2} \cdots G_3 G_2 G_1$$

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

6.4 Transformações de visualização

Na grande maioria dos casos, a apresentação de um ambiente tridimensional gerado por computador é feita em dispositivos cuja saída é bidimensional. Assim, para que o cenário virtual seja visualizado, seus modelos tridimensionais devem ser projetados em uma determinada superfície (geralmente plana), a fim de que seja gerada uma imagem bidimensional da cena. A esta superfície dá-se o nome de superfície de projeção.

As projeções mais empregadas nas aplicações gráficas são a projeção paralela e a projeção em perspectiva. Na projeção paralela os vértices de todos os objetos tridimensionais são projetados na superfície de projeção segundo retas paralelas. Na projeção em perspectiva, as retas projetantes convergem para um ponto específico, chamado de ponto de projeção.

No caso da superfície ser plana, pode-se defini-la segundo a equação geral do plano:

$$n_x x + n_y y + n_z z - n_x r_x - n_y r_y - n_z r_z = 0$$

onde n_x, n_y e n_z são as componentes de um vetor ${}^{U}\vec{N}$ normal ao plano e r_x, r_y e r_z são as coordenadas de um ponto R pertencente ao plano - todas elas definidas em relação ao sistema de coordenadas universal.

Antes que qualquer projeção seja efetuada, é necessário que a superfície de projeção possua um sistema de coordenadas próprio, a fim de que as coordenadas universais dos vértices de cada objeto sejam convertidas em coordenadas de visualização. Há muitas maneiras para se definir este sistema de coordenadas, no entanto uma definição bastante comum é a apresentada na figura 6.8:

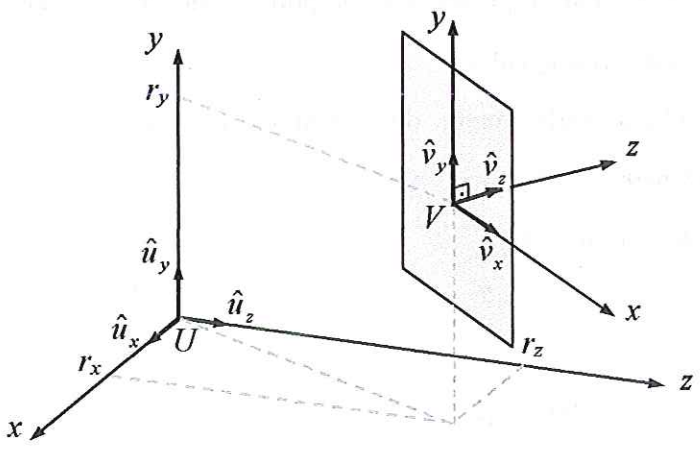


Figura 6.8 - Definição do referencial V pertinente à superfície de projeção

onde $\{\hat{u}_x, \hat{u}_y, \hat{u}_z\}$ e $\{\hat{v}_x, \hat{v}_y, \hat{v}_z\}$ são as bases ortonormais geradoras do sistema de coordenadas universal (*U*) e do sistema de coordenadas de visualização (*V*), respectivamente.

Para se determinar o sistema de coordenadas de visualização pode-se proceder da seguinte forma:

- Toma-se o ponto (r_x, r_y, r_z) como sendo a origem do sistema.
- Normaliza-se o vetor ${}^{U}\vec{N}$, definindo o versor $\hat{v}_z$.
- A partir de um outro ponto do plano de projeção, define-se o versor $\hat{v}_y$, geralmente na direção vertical.
- $\hat{v}_x$ é obtido pelo produto vetorial dos outros versores direcionais.

Uma vez definida a base $\{\hat{v}_x, \hat{v}_y, \hat{v}_z\}$, pode-se encontrar uma matriz mudança de base que transforma as coordenadas universais de qualquer vetor em coordenadas de visualização. A seguir é feita uma breve descrição do procedimento adotado para a determinação dessa matriz.

Considere um vetor qualquer $\vec{T}$, não nulo, expresso em ambos os sistemas de coordenadas por meio de uma combinação linear dos respectivos versores, conforme mostrado abaixo:

$$ {}^{U}\vec{T} = {}^{U}t_x\hat{u}_x + {}^{U}t_y\hat{u}_y + {}^{U}t_z\hat{u}_z \qquad \text{(I)} $$

$$ {}^{V}\vec{T} = {}^{V}t_x\hat{v}_x + {}^{V}t_y\hat{v}_y + {}^{V}t_z\hat{v}_z \qquad \text{(II)} $$

onde a notação ${}^{j}t_i$ adotada representa a componente do vetor $\vec{T}$ na direção do eixo *i*, em relação ao sistema de coordenadas *j*.

Expressando-se cada versor direcional de *V* como uma combinação linear dos versores direcionais de *U*, obtém-se:

$$ \begin{cases} \hat{v}_x = v_{xx}\hat{u}_x + v_{xy}\hat{u}_y + v_{xz}\hat{u}_z \\ \hat{v}_y = v_{yx}\hat{u}_x + v_{yy}\hat{u}_y + v_{yz}\hat{u}_z \\ \hat{v}_z = v_{zx}\hat{u}_x + v_{zy}\hat{u}_y + v_{zz}\hat{u}_z \end{cases} \qquad \text{(III)} $$

Substituindo-se (III) em (II), chega-se à expressão:

$$ {}^{U}t_x\hat{u}_x + {}^{U}t_y\hat{u}_y + {}^{U}t_z\hat{u}_z = {}^{V}t_x(v_{xx}\hat{u}_x + v_{xy}\hat{u}_y + v_{xz}\hat{u}_z) + {}^{V}t_y(v_{yx}\hat{u}_x + v_{yy}\hat{u}_y + v_{yz}\hat{u}_z) + {}^{V}t_z(v_{zx}\hat{u}_x + v_{zy}\hat{u}_y + v_{zz}\hat{u}_z) $$

ou na forma matricial:

$$ {}^{U}\begin{pmatrix} t_x \\ t_y \\ t_z \\ 1 \end{pmatrix} = \begin{pmatrix} v_{xx} & v_{yx} & v_{zx} & 0 \\ v_{xy} & v_{yy} & v_{zy} & 0 \\ v_{xz} & v_{yz} & v_{zz} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot {}^{V}\begin{pmatrix} t_x \\ t_y \\ t_z \\ 1 \end{pmatrix} $$

Como a matriz acima é ortogonal, sua inversa é igual à sua transposta. Logo, pode-se escrever:

$$ {}^{V}\begin{pmatrix} t_x \\ t_y \\ t_z \\ 1 \end{pmatrix} = \begin{pmatrix} v_{xx} & v_{xy} & v_{xz} & 0 \\ v_{yx} & v_{yy} & v_{yz} & 0 \\ v_{zx} & v_{zy} & v_{zz} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot {}^{U}\begin{pmatrix} t_x \\ t_y \\ t_z \\ 1 \end{pmatrix} $$

Os referenciais U e V, no entanto, não apresentam origens coincidentes. Neste caso, uma translação inicial faz-se necessária antes que sejam efetuadas as transformações de rotação definidas pela matriz mudança de base. Desta forma, a seqüência de transformações que converte as coordenadas universais de qualquer ponto em coordenadas de visualização pode ser representada pela composição matricial descrita abaixo.

$$ {}^{V}\begin{pmatrix} t_x \\ t_y \\ t_z \\ 1 \end{pmatrix} = \begin{pmatrix} v_{xx} & v_{xy} & v_{xz} & 0 \\ v_{yx} & v_{yy} & v_{yz} & 0 \\ v_{zx} & v_{zy} & v_{zz} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & -r_x \\ 0 & 1 & 0 & -r_y \\ 0 & 0 & 1 & -r_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot {}^{U}\begin{pmatrix} t_x \\ t_y \\ t_z \\ 1 \end{pmatrix} $$

6.5 Rendering

Rendering é o processo pelo qual uma imagem é sintetizada no computador a partir dos dados que descrevem a cena virtual. Não há, entretanto, um método único para se processar as etapas seqüenciais envolvidas no *rendering*, já que existem muitas maneiras diferentes para se modelar, transformar, iluminar e texturizar os objetos virtuais. Alguns autores consideram as etapas de geração e transformação dos objetos como parte do processo de *rendering*, outros preferem considerar apenas as etapas posteriores a estas.

6.5.1 Eliminação das faces ocultas

Antes dos objetos serem projetados na superfície de projeção, deve-se verificar se existem faces que não podem ser vistas pelo observador. Esta análise pode ser feita por meio de um produto escalar entre o versor normal de cada face e o versor que define a direção de projeção do polígono, como mostra a figura 6.9:

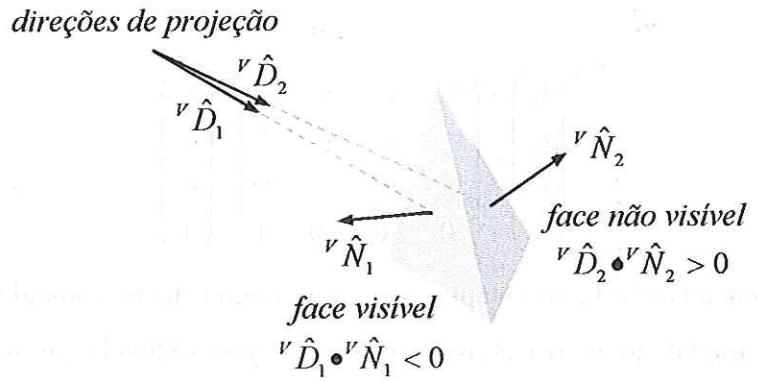


Figura 6.9 - Determinação das faces que não são vistas pelo observador

A direção de projeção da face é definida pela reta que passa pelo ponto de projeção e pelo centro do polígono. A face é considerada como visível se o produto escalar for positivo, ou seja, se o ângulo entre a direção normal e a direção de projeção da face for menor do que 90°.

6.5.2 Eliminação dos polígonos que estão fora do volume de visualização

O volume de visualização é um espaço fechado, definido em coordenadas universais, que confina a totalidade da cena que será projetada. Ele é definido por seis planos e sua forma depende do tipo de projeção usada. Para a projeção em perspectiva, o volume de visualização assume o formato de um tronco de pirâmide, onde o plano de recorte anterior normalmente coincide com o plano de projeção. A figura 6.10 ilustra o volume de visualização para uma projeção em perspectiva.

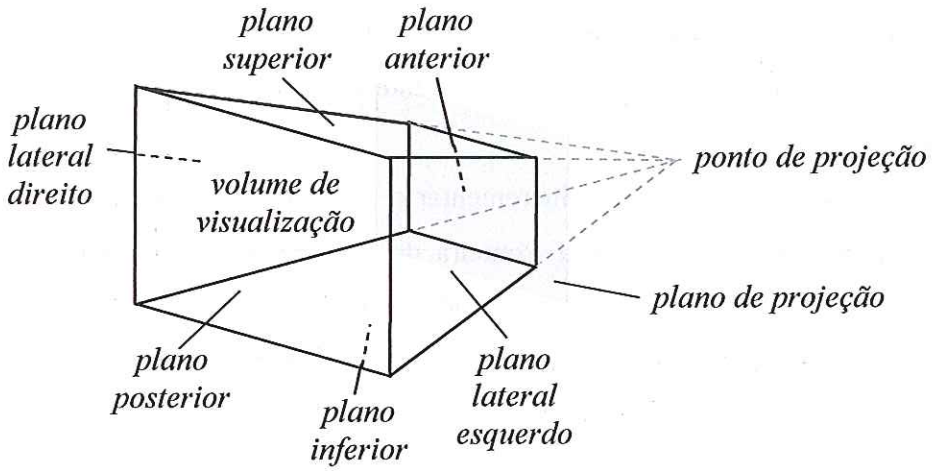


Figura 6.10 - Volume de visualização para uma projeção em perspectiva

As janelas de visualização fornecidas por alguns sistemas operacionais ou implementadas pelo ambiente de programação já delimitam a visualização da cena, executando uma espécie de recorte nos polígonos localizados na sua fronteira. No entanto isto não impede que o cálculo do *shading* (descrito no item 6.5.5) seja feito para todas as faces presentes no cenário virtual, o que torna o processo de *rendering* bastante lento. Assim, é recomendável a implementação de um algoritmo que descarte os polígonos que estejam completamente fora do volume de visualização. Um polígono estará fora deste volume se todos os seus vértices também o estiverem.

Cada plano do volume de visualização é definido por três pontos de coordenadas universais. A partir destes três pontos consegue-se obter dois vetores coplanares, cujo produto vetorial gera um vetor normal ao plano. Baseado na equação geral do plano, pode-se definir a seguinte função que verifica a posição de um determinado ponto em relação ao plano:

$$f(x,y,z) = n_x x + n_y y + n_z z - n_x x_0 - n_y y_0 - n_z z_0$$

onde n_x, n_y e n_z são as coordenadas universais do vetor normal ao plano e x_0, y_0 e z_0 são as coordenadas universais de um dos pontos geradores do plano.

Desta forma, o método se baseia em uma verificação do sinal da função $f(x,y,z)$ quando as coordenadas dos vértices são substituídas na mesma. Cada vértice deve ser avaliado em relação aos seis planos. A associação do sinal da função com "o lado do plano onde o ponto está localizado" depende do sentido do vetor normal ao plano. Logo, é

importante que se estabeleça uma convenção de sentidos para os vetores normais dos planos do volume de visualização. Geralmente adotam-se os vetores normais apontando para fora do volume.

Adicionalmente, pode-se incrementar o algoritmo para que seja efetuado o recorte dos polígonos que interceptam a fronteira do volume de visualização. No entanto, este assunto requer uma abordagem demasiadamente extensa para que seja entendido com clareza e, portanto, não será tratado aqui. Para maiores informações, veja Alan H. Watt (1992).

6.5.3 Projeção em perspectiva

Uma vez definido o sistema de coordenadas de visualização, podemos realizar a projeção em perspectiva dos objetos tridimensionais. Como mencionado anteriormente, a projeção em perspectiva necessita de um ponto de projeção para a representação dos objetos na superfície de projeção, como mostra a figura 6.11:

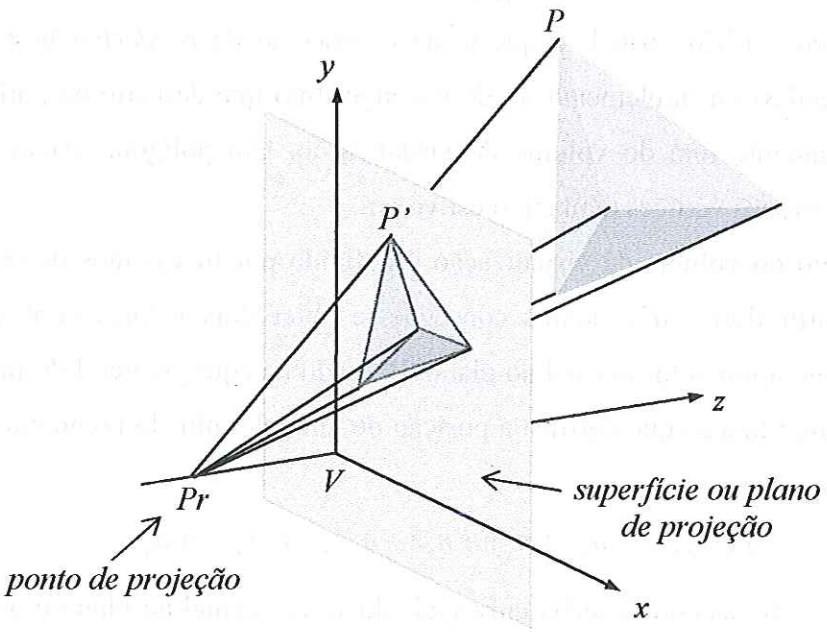


Figura 6.11 - Projeção em perspectiva de um modelo 3D sobre a superfície de projeção

Segundo a figura 6.11, o ponto de projeção Pr possui coordenadas (0 , 0 , pr_z) em relação ao sistema de coordenadas de visualização. A reta definida por este ponto e pelo vértice $P(p_x, p_y, p_z)$ pode ser representada pela seguinte equação paramétrica:

$$\begin{cases} x = p_x t \\ y = p_y t \\ z = pr_z + (p_z - pr_z)t \end{cases}$$

O ponto $P'(p_x', p_y', p_z')$ resulta da interseção desta reta com o plano de projeção. Como qualquer ponto pertencente ao plano de projeção apresenta cota nula em relação ao sistema de coordenadas de visualização, a posição de P' pode ser determinada fazendo-se $z = 0$ na equação da reta de projeção, o que resulta em:

$$p_x' = \frac{pr_z p_x}{pr_z - p_z} \quad , \quad p_y' = \frac{pr_z p_y}{pr_z - p_z} \quad \text{e} \quad p_z' = 0$$

Em virtude do caráter não linear destas expressões, não existe uma forma matricial para representá-las integralmente. Pode-se, entretanto, definir uma matriz de projeção em perspectiva que engloba parte das operações para minimizar a necessidade do uso de cálculos não matriciais:

$$V\begin{pmatrix} p_x \\ p_y \\ p_z \\ h \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1/pr_z & 1 \end{pmatrix} \cdot V\begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix}$$

Desta operação obtém-se a relação de projeção em perspectiva:

$$h = 1 - \frac{p_z}{pr_z}$$

As coordenadas de projeção podem agora ser expressas como:

$$p_x' = \frac{p_x}{h} \quad \text{e} \quad p_y' = \frac{p_y}{h}$$

6.5.4 Iluminação

Na Natureza, a radiação eletromagnética emitida por uma determinada fonte de luz pode ser refletida em muitos objetos antes de atingir nossas retinas. A cada reflexão, uma parte da radiação é absorvida pela superfície do objeto, outra parte é refratada e o restante é refletido de volta ao meio. Esta parcela refletida incide então sobre outros objetos, iluminando-os e sendo parcialmente refletida por eles. O processo continua até que as

últimas parcelas de luz sejam totalmente absorvidas por outras superfícies. Assim, o grau de iluminação de um corpo não depende apenas dos raios da fonte de luz que incidem sobre ele, mas, sobretudo, da luz indireta que chega até ele proveniente das diversas reflexões nos outros objetos.

A radiação refletida pode seguir um caminho especular, no qual o ângulo de incidência é igual ao ângulo de reflexão, ou então ser refletida difusamente, sem que haja uma direção preferencial. A radiação absorvida produz um aumento na temperatura do objeto, fazendo com que este emita mais radiação, preferencialmente no espectro infravermelho. A radiação refratada atravessa o objeto, normalmente sofrendo um desvio de trajetória.

A rigor, a interação da luz com os objetos é um fenômeno extremamente complexo e os modelos matemáticos desenvolvidos para simular o seu comportamento são aproximações válidas em casos específicos.

Obviamente, o cálculo requerido para se tentar simular com precisão uma luz natural é demasiadamente caro, em termos de processamento computacional, para ser aplicado na elaboração de gráficos 3D em tempo real. Nestes casos, as *APIs* gráficas são implementadas com modelos de iluminação otimizados para o processamento em tempo real. Nestes modelos, são computadas as reflexões difusa e especular cujas radiações incidentes são provenientes apenas da fonte de luz, ou seja, os raios luminosos provenientes da reflexão em outros objetos são descartados no cálculo da iluminação. Para se tentar suprir essa parcela de iluminação desconsiderada, aplica-se uma iluminação isotrópica aos objetos do cenário tridimensional. Essa iluminação, conhecida como iluminação ambiente, é processada de modo que cada objeto da cena seja iluminado por uma certa quantidade de radiação provinda de todas as direções. Desta forma, as superfícies que forem submetidas a este efeito passarão a refletir esta luz, mesmo estando na sombra. Embora não privilegie a realidade, essa técnica favorece o rápido processamento dos cálculos com uma qualidade gráfica aceitável.

6.5.4.1 Fontes de luz

Em geral, *APIs* gráficas como o *Microsoft Direct3D* possuem três tipos de fontes de luz implementadas: A fonte de luz direcional, a fonte de luz pontual (*omni*) e a fonte de luz focada (*spot*).

A fonte de luz direcional é definida pela direção e sentido de um único vetor e pela cor. Não apresenta parâmetros de atenuação ou alcance dos raios luminosos.

A fonte de luz pontual emite luz igualmente em todas as direções. É definida por um vetor posição, por uma cor e pela configuração dos parâmetros de atenuação e alcance dos raios luminosos.

A fonte de luz focada é definida por um vetor posição, por um vetor direcional (em torno do qual é gerado o cone de luz), por uma cor e pela configuração dos parâmetros de atenuação, alcance e *falloff*. Este último parâmetro configura a intensidade da penumbra de luz na borda da base do cone.

6.5.4.2 Reflexão da luz ambiente

A intensidade da luz ambiente refletida por uma superfície é definida pela relação abaixo:

$$I_{ra} = K_a I_{ia}$$

onde I_{ia} é a intensidade de luz ambiente incidente sobre a superfície e K_a é o coeficiente de reflexão da luz ambiente.

6.5.4.3 Reflexão difusa

A modelagem da reflexão difusa é também bastante simples. Por ser isotrópica, ela não depende da direção de observação. No entanto, a intensidade de radiação refletida difusamente por uma superfície varia de acordo com o ângulo definido entre a direção da luz incidente e a normal da superfície. Quanto menor for este ângulo, maior será a intensidade de luz difusa refletida. A relação que define a intensidade de luz refletida difusamente é descrita como:

$$I_{rd} = K_d I_{if} \cos\theta$$

onde I_{if} é a intensidade da fonte de luz, K_d é o coeficiente de reflexão difusa e θ é o ângulo que a normal da superfície faz com a direção de incidência da luz.

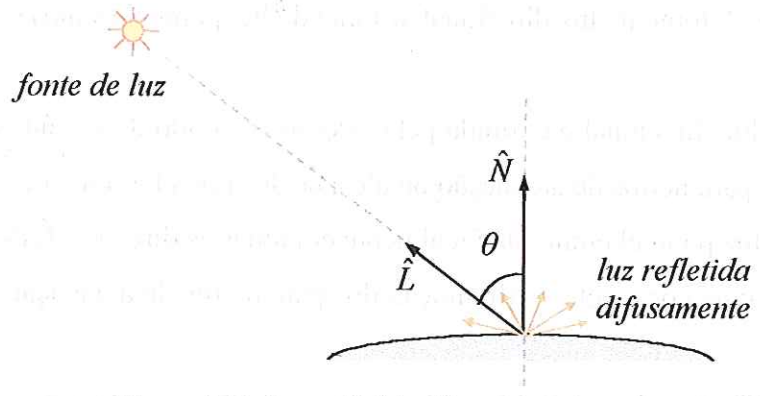


Figura 6.12 - Luz refletida difusamente por uma superfície

A relação acima também pode ser descrita em termos vetoriais:

$$I_{rd} = K_d I_{if} \hat{L} \bullet \hat{N}$$

onde $\hat{N}$ é o versor normal à superfície e $\hat{L}$ é o versor direcional da luz incidente, cujo sentido é definido da superfície do objeto para a fonte de luz.

6.5.4.4 Reflexão especular

Ao contrário da reflexão difusa, a reflexão especular pode ser representada por vários modelos computacionais. No entanto, poucos modelam perfeitamente o fenômeno de reflexão, efetuando apenas cálculos que geram um efeito brilhante na superfície dos objetos. A rigor, a reflexão especular também considera a presença de outros objetos na cena, cujas imagens deveriam ser projetadas na superfície refletora. Porém, o processamento desses modelos é tão oneroso, em termos computacionais, que são poucos os pacotes de *rendering* capazes efetivamente de realizar a reflexão de objetos.

Na reflexão especular, o ângulo de incidência da luz na superfície é igual ao ângulo de reflexão. Desta forma, o efeito de reflexão (*high-light*) é notado apenas quando a direção de observação for igual à direção de reflexão.

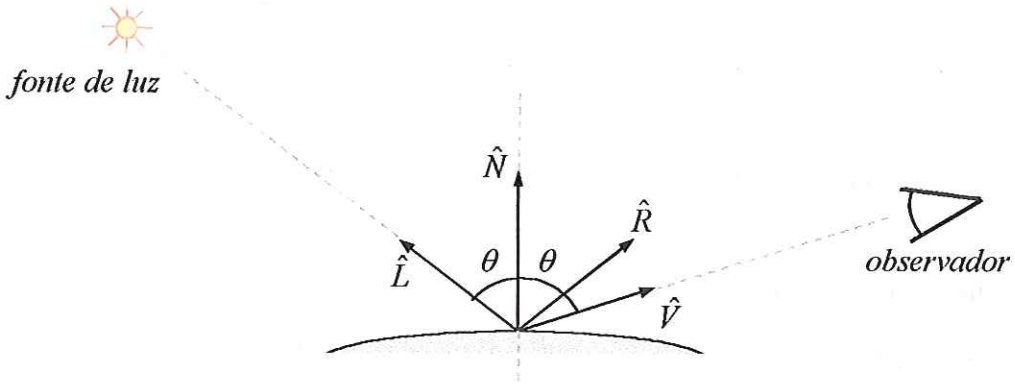


Figura 6.13 - Luz refletida especularmente por uma superfície

O versor $\hat{R}$, que define a direção da reflexão especular, pode ser determinado a partir dos versores $\hat{L}$ e $\hat{N}$, segundo a relação abaixo:

$$\hat{R} = 2(\hat{L} \bullet \hat{N})\hat{N} - \hat{L}$$

Se a direção de observação da superfície é definida pelo versor $\hat{V}$, então o observador somente poderá ver o efeito de reflexão se $\hat{V} = \hat{R}$, o que é muito pouco provável de ocorrer. Para contornar este problema, pode-se considerar que o efeito de reflexão pode ser visto desde que as direções de reflexão e observação formem um ângulo suficientemente pequeno. Bui T. Phong (WATT, 1992) desenvolveu o seguinte modelo para representar a reflexão especular:

$$I_{rs} = K_s I_{if} \left(\hat{R} \bullet \hat{V}\right)^n$$

onde K_s é o coeficiente de reflexão especular e n é o fator que define a intensidade do efeito de reflexão.

6.5.4.5 Reflexões combinadas

Pode-se definir uma função que combine os modelos anteriormente descritos:

$$I = I_{ra} K_a + I_{if} \left[K_d \hat{L} \bullet \hat{N} + K_s \left(\hat{R} \bullet \hat{V}\right)^n \right]$$

A função acima, normalmente conhecida como função de iluminação, considera que a luz incidente é proveniente de apenas uma fonte de luz. Porém, ela pode facilmente ser generalizada para o caso de se existirem m fontes iluminando a superfície:

$$I = I_{ra}K_a + \sum_{j=1}^{m} {}^{j}I_{if}\left[K_d\hat{L}\bullet\hat{N} + K_s\left(\hat{R}\bullet\hat{V}\right)^n\right]$$

onde ${}^{j}I_{if}$ representa a intensidade da fonte luz *j*.

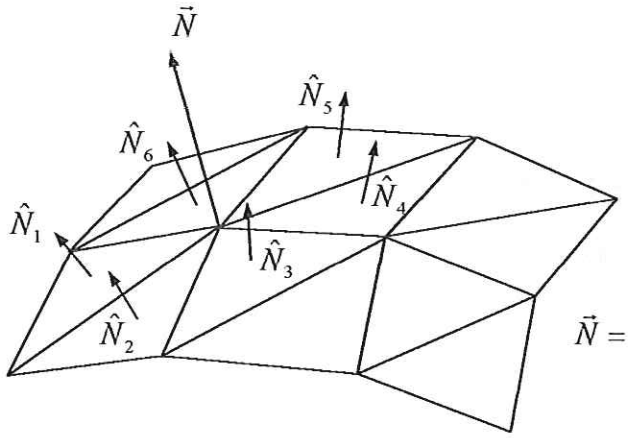
Na geração de uma imagem colorida, a função de iluminação é aplicada individualmente em cada um dos canais RGB que compõe a cor da luz incidente. Como a cor de um objeto é definida basicamente pela reflexão difusa, valores distintos de K_d são utilizados, um para cada canal, para se configurar a cor desejada.

6.5.5 Shading

Shading ou tonalização poligonal é o processo de obtenção das cores dos *pixels* que formam os polígonos, a partir das propriedades definidas para a superfície do objeto e dos dados de iluminação.

Embora existam várias técnicas para se efetuar o *shading*, abordaremos aqui a técnica desenvolvida por *Gouraud* (WATT, 1992), que foi a usada no *rendering* do ambiente virtual deste projeto.

O modelo de *Gouraud* consiste em uma técnica de interpolação bi-linear que busca gerar uma aparência suavizada (*smoothing*) para a superfície facetada dos objetos tridimensionais. Para que a tonalização seja contínua e as arestas no interior dos objetos não sejam percebidas, o método define, para cada vértice do objeto, um vetor resultante da soma dos versores normais de todas as faces que compartilham este mesmo vértice. A figura 6.14 mostra este procedimento.



$\vec{N} = \sum_{i=1}^{m} \hat{N}_i$ onde m é o número de faces que compartilham o vértice.

Figura 6.14 - Determinação dos vetores usados no cálculo do *shading*

Após sua determinação, o vetor soma $\vec{N}$ deve ser normalizado ($\hat{N}$) para que o cálculo da iluminação seja processado adequadamente.

O cálculo das intensidades luminosas, em cada vértice, é feito a partir dos versores $\hat{L}$ e $\hat{N}$, com base na equação de iluminação descrita anteriormente. O versor $\hat{L}$ é definido pela normalização do vetor que vai do vértice à fonte de luz. Uma vez projetados os vértices do polígono, com base nas suas coordenadas de visualização, inicia-se o processo de varredura (*rasterization*), *pixel* a *pixel*, onde são interpoladas as intensidades luminosas.

Considere o exemplo apresentado na figura 6.15. Inicialmente as coordenadas dos vértices do polígono são projetadas na superfície de projeção segundo as relações estabelecidas pela projeção em perspectiva, obtendo-se as coordenadas de visualização (x_1, y_1), (x_2, y_2) e (x_3, y_3). Em seguida são calculadas as intensidades luminosas I_1, I_2 e I_3 associadas a estes vértices, com base nas propriedades da superfície e da fonte de luz. Inicia-se então um processo de varredura, linha por linha, de cima para baixo, decrementando-se a coordenada y_s. As coordenadas x_a, y_a, x_b e y_b são geradas pela interpolação das arestas, como mostrado na figura 6.16. A seguir são interpoladas as intensidades I_a e I_b nas arestas, a partir das relações:

$$I_a = \frac{1}{y_1 - y_2}\left[I_1(y_s - y_2) + I_2(y_1 - y_s)\right]$$

$$I_b = \frac{1}{y_1 - y_3}\left[I_1(y_s - y_3) + I_3(y_1 - y_s)\right]$$

A coordenada x_s é então variada entre x_a e x_b, *pixel* a *pixel*, possibilitando a interpolação da intensidade I_s em cada ponto, segundo a equação:

$$I_s = \frac{1}{x_b - x_a}\left[I_a(x_b - x_s) + I_b(x_s - x_a)\right]$$

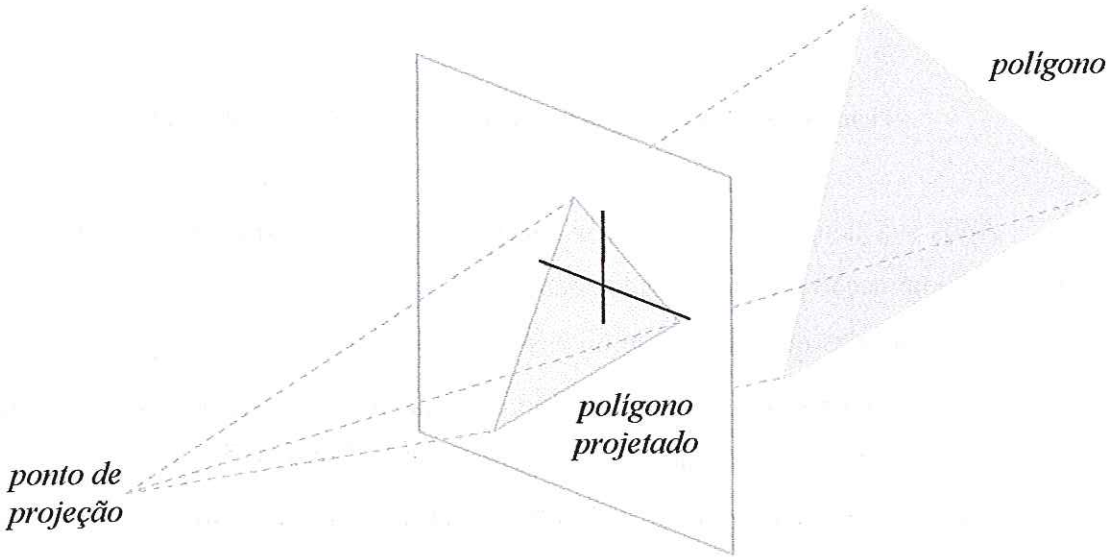


Figura 6.15 - Projeção do polígono sobre a superfície de projeção

Para melhorar a eficiência dos métodos as equações acima são colocadas na forma incremental:

$$I_{a,n} = I_{a,n-1} + \Delta I_a$$

$$I_{b,n} = I_{b,n-1} + \Delta I_b$$

$$I_{s,m} = I_{s,m-1} + \Delta I_s$$

onde os incrementos podem ser calculados previamente por:

$$\Delta I_a = \frac{\Delta y}{y_2 - y_1}(I_2 - I_1)$$

$$\Delta I_b = \frac{\Delta y}{y_3 - y_1}(I_3 - I_1)$$

$$\Delta I_s = \frac{\Delta x}{x_b - x_a}(I_b - I_a)$$

sendo que os incrementos Δy e Δx são unitários. As intensidades $I_{a,n}$ e $I_{b,n}$ são inicializadas com o valor de I_1, enquanto que $I_{s,m}$ é inicializada com I_a.

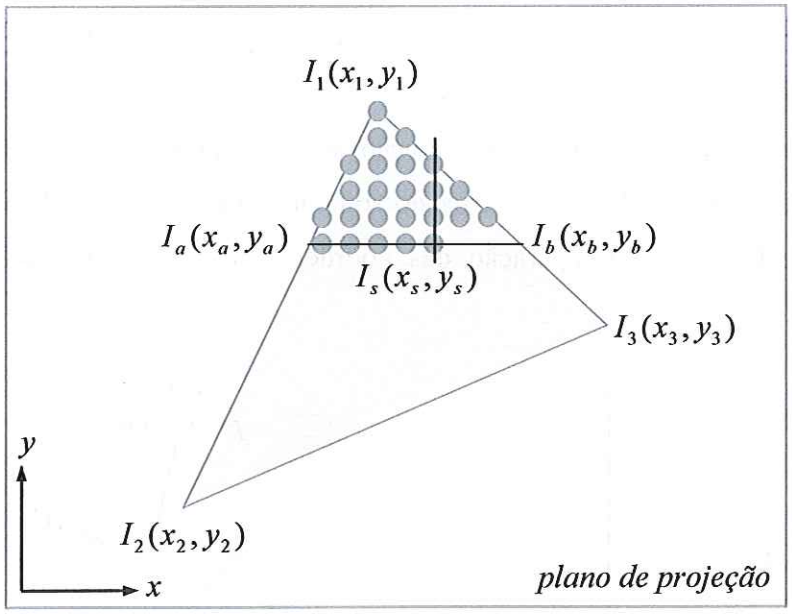


Figura 6.16 - Tonalização poligonal interpolada por varredura

A figura 6.17 ilustra um mesmo objeto submetido a duas técnicas de *shading* distintas: a *Flat* e a *Gouraud*. No método *Flat* todos os *pixels* de um mesmo polígono possuem a mesma cor, já que não há interpolação das intensidades luminosas nas faces do objeto. Seu cálculo é bastante simples e se baseia no ângulo definido entre a direção normal de cada face e a reta que passa pelo centro do polígono e pela fonte de luz. Embora muito eficiente, essa técnica gera uma aparência facetada para a superfície do objeto, o que muitas vezes não é desejado.

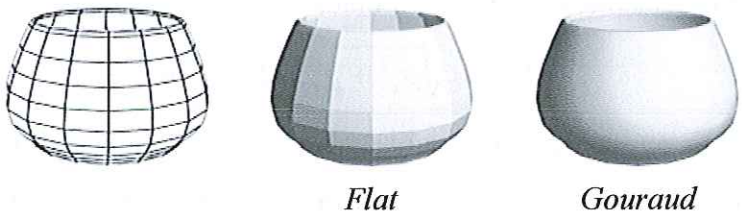


Figura 6.17 - Tonalização *Flat* e *Gouraud*

6.5.6 Depth buffer

O *depth buffer*, também conhecido como *z-buffer*, é um método implementado para armazenar informações sobre a profundidade dos polígonos que serão processados. Em um processo de *rendering*, quando existem polígonos sobrepostos na cena, o *depth buffer* é usado para identificar, durante o processo de varredura (*rasterization*), qual dos pontos sobrepostos (um de cada polígono) está mais próximo do ponto de projeção.

Na projeção em perspectiva, a comparação das distâncias dos pontos sobrepostos ao ponto de projeção deve ser feita na direção da reta projetante. Assim, seria mais conveniente que o termo *z-buffer* fosse usado somente na projeção paralela, uma vez que este método se baseia na comparação das coordenadas *z* de visualização dos pontos analisados.

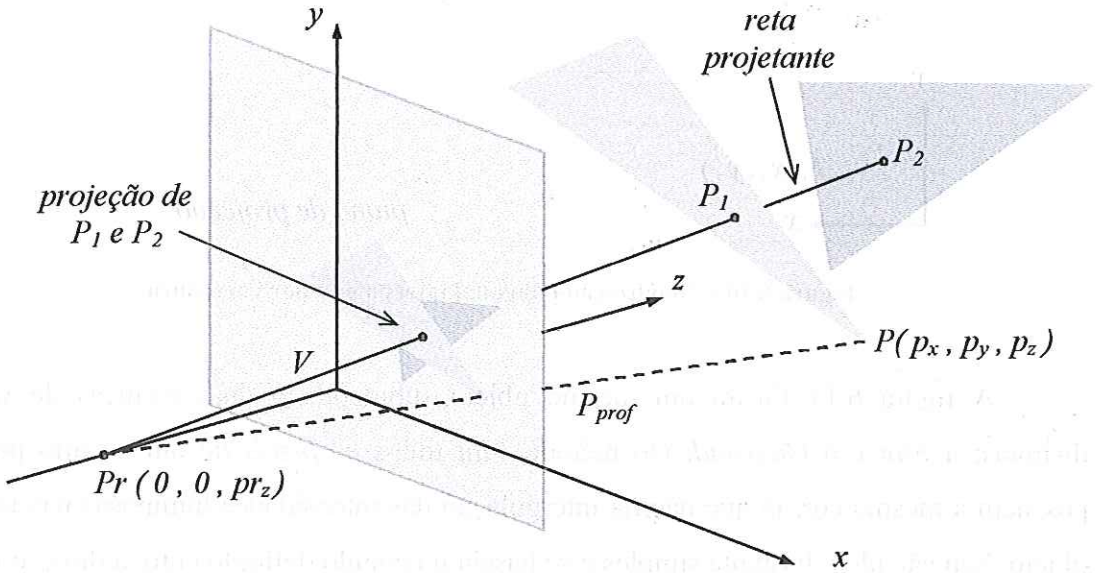


Figura 6.18 - Ilustração da projeção de polígonos sobrepostos

A profundidade de cada vértice em relação ao ponto de projeção pode ser definida pelo módulo do segmento que une estes pontos:

$$P_{prof} = \sqrt{p_x^{\,2} + p_y^{\,2} + (p_z - pr_z)^2}$$

Entretanto, apesar de oferecer uma solução, esta formulação ainda é cara, em termos computacionais. Um método mais eficiente pode ser obtido convertendo-se a projeção em perspectiva em uma projeção paralela, onde a coordenada *z* de cada vértice pode ser

transformada pela relação *h*, deduzida no item 2.5.3. Assim, pode-se expressar a profundidade do vértice *P* como:

$$P_{prof} = \frac{p_z}{1 - p_z / pr_z}$$

Na figura 6.19 é mostrado o efeito da aplicação desta transformação nos vértices de um cubo. Note que, na conversão para a projeção paralela, o objeto é deformado no espaço tridimensional, de modo que as relações geométricas entre suas arestas e as retas projetantes sejam mantidas. Assim, a imagem projetada permanece inalterada.

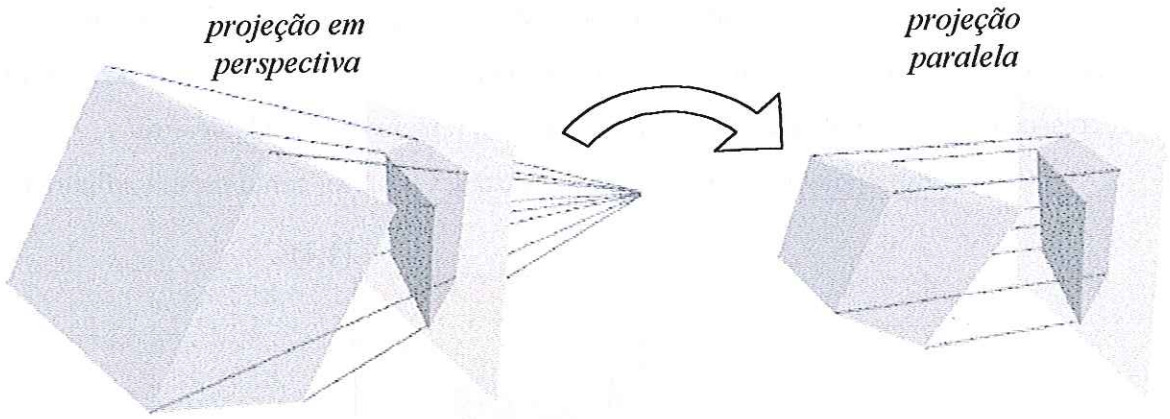


Figura 6.19 - Transform. das coordenadas dos vértices de um objeto segundo a relação *h*

A aplicação desta técnica é feita durante o *shading* do objeto, interpolando-se as cotas transformadas dos vértices dos polígonos juntamente com as intensidades luminosas. Desta forma, o algoritmo de *depth buffer* cria, para cada *pixel* da imagem, um espaço de memória para armazenar o valor da profundidade interpolada. Basicamente, o método declara uma matriz de dimensão igual à resolução da imagem, onde cada elemento armazena o valor de profundidade associado a um *pixel* da imagem. Os elementos desta matriz são inicializados com valores muito grandes para garantir que, em uma primeira comparação, qualquer polígono dentro do volume de visualização seja apresentado. A cada novo polígono projetado, para cada *pixel* formador deste, é verificado se o valor da profundidade obtida é maior ou menor do que o previamente armazenado no elemento matricial associado ao *pixel* analisado.

O método de *depth buffer* é bastante rápido e de fácil implementação. Sua compatibilidade com programas e *APIs* gráficas fez com que fossem desenvolvidas placas aceleradoras de vídeo baseadas integralmente nele. A crescente redução nos preços das memórias tem ajudado na sua consolidação.

6.5.7 Texturização dos polígonos

A palavra textura, em geral, refere-se ao grau de rugosidade ou aspereza de uma superfície. Em termos computacionais, textura é o nome que se dá à imagem digital (*bitmap*) aplicada (mapeada) sobre a malha para lhe fornecer uma aparência mais realística.

A imagem de uma textura é armazenada na memória por meio de uma matriz, onde o número de bits reservado para cada elemento depende do número de cores possíveis para representar a imagem. A unidade elementar de uma textura é chamada de *texel*.

Cada textura possui um sistema de coordenadas próprio, como mostra a figura 6.20:

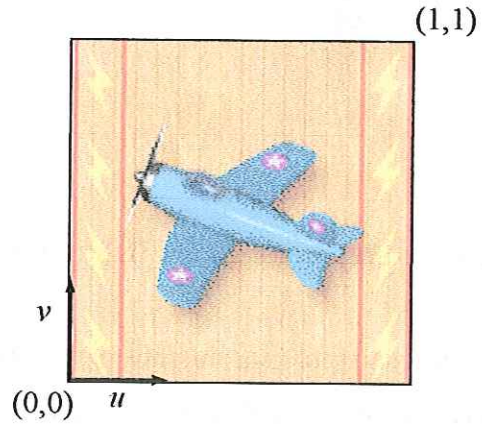


Figura 6.20 – Representação do sistema de coordenadas da textura

O mapeamento de um polígono consiste no modo como a textura é aplicada sobre o mesmo. Para tanto, são definidas as posições dos pontos T_1, T_2 e T_3 no espaço da textura, onde cada ponto T_i está associado a um vértice V_i do polígono. Assim, qualquer ponto T no interior do polígono definido sobre a textura pode ser expresso pela seguinte combinação linear:

$$T = \alpha_1 (T_2 - T_1) + \alpha_2 (T_3 - T_1)$$

Com os valores de α_1 e α_2 pode-se determinar, no polígono do espaço tridimensional, um ponto V correspondente a T, segundo a relação abaixo:

$$V = \alpha_1 (V_2 - V_1) + \alpha_2 (V_3 - V_1)$$

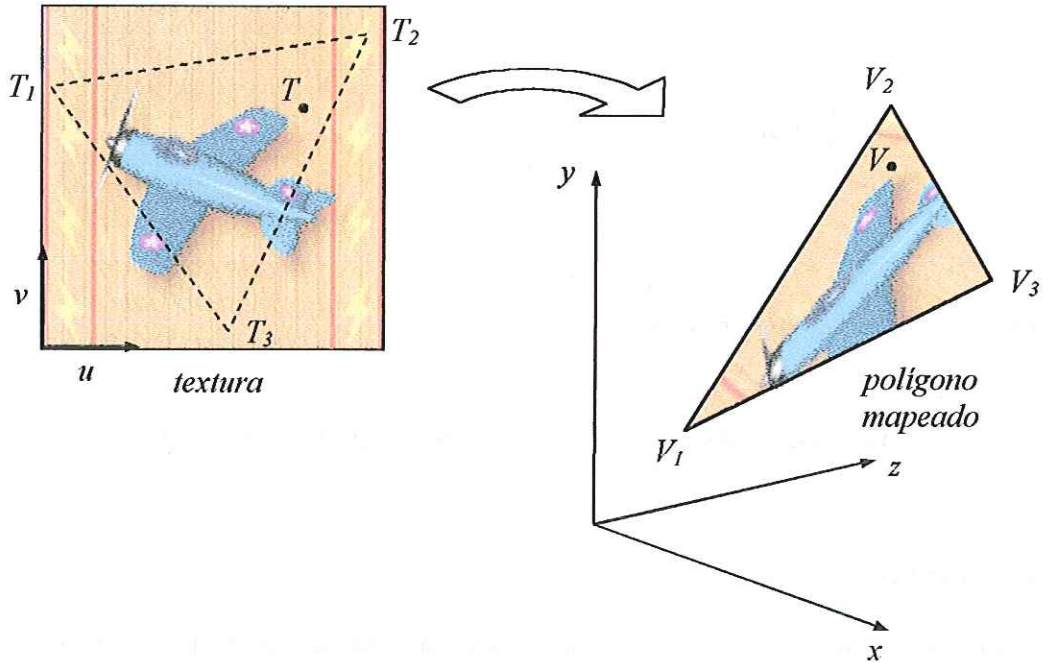


Figura 6.21 - Mapeamento da textura sobre um polígono

A apresentação do polígono mapeado é normalmente feita em uma ordem inversa, ou seja, para cada *pixel* da tela, são calculadas as posições, no espaço da textura, dos *texels* correspondentes.

Neste processo, admite-se que o *pixel* tenha um formato quadrangular. Projetam-se então os quatro vértices do *pixel* sobre o polígono, definindo neste uma região quadrilátera. Com base nas relações algébricas de mapeamento, determina-se a área da textura correspondente àquela região. Por fim, efetuando-se uma média entre as cores dos *texels* desta área, define-se a cor do *pixel* correspondente. Esse procedimento é aplicado em cada *pixel* formador do polígono.

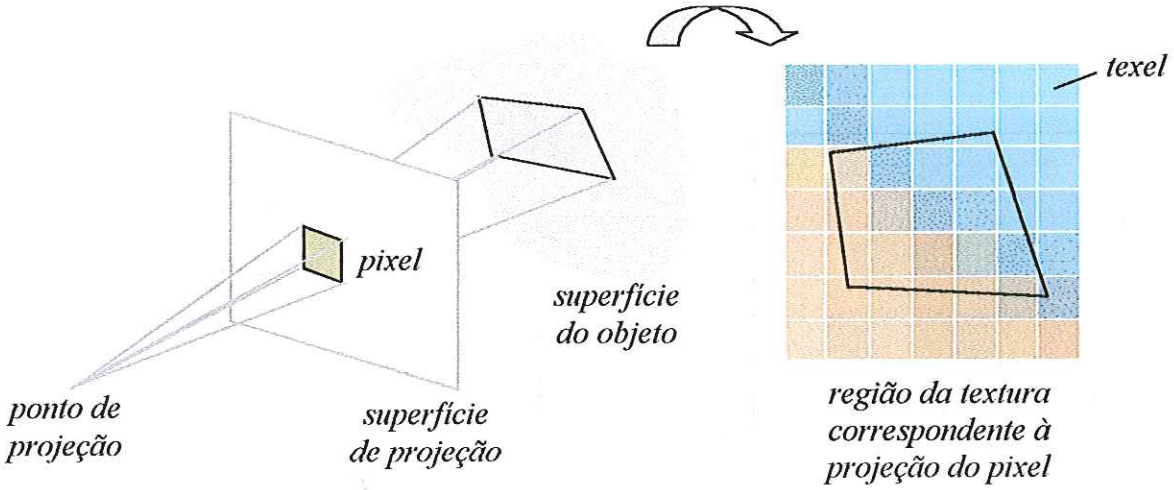


Figura 6.22 - *Rendering* do polígono mapeado (processo inverso)

6.6 Colisões geométricas

Quando as superfícies de dois objetos tridimensionais se interceptam, diz-se que eles sofreram uma colisão geométrica. Em sua essência, uma colisão geométrica é caracterizada por uma interseção entre polígonos. Como todo polígono pode ser descrito por meio de um plano limitado, a técnica utilizada se resume basicamente na verificação de interseções entre planos dentro de regiões específicas.

Em aplicações gráficas onde não se privilegia a interação em tempo real, a verificação da colisão pode ser realizada com extrema precisão. Nestes casos, o algoritmo verifica, para cada face de um objeto, se ocorre a interseção desta com alguma das faces dos outros objetos. No entanto, esse processo de checagem pode ser muito oneroso quando os objetos tridimensionais apresentarem um grande número de polígonos. Por esse motivo, nos ambientes gráficos que operam em tempo real, a checagem da colisão é efetuada com o uso de técnicas alternativas mais eficientes (em relação ao tempo de processamento), porém bem menos precisas.

Uma solução bastante utilizada em jogos consiste na criação de formas geométricas simples circunscritas aos objetos. Neste contexto, se não há uma preocupação com a precisão da colisão, podem-se definir matematicamente esferas imaginárias circunscritas aos objetos e verificar se a distância entre os centros do par de esferas em análise de colisão

é menor do que a soma dos seus raios. Muitos programadores, entretanto, adotam a definição de paralelepípedos imaginários circunscritos aos objetos, o que geralmente proporciona uma análise de colisão bem mais precisa do que a oferecida pela técnica anterior, porém requerendo um tempo maior de processamento. Mesmo assim, é muito mais eficiente verificar a colisão entre paralelepípedos do que entre objetos formados por muitos polígonos. Pode-se, eventualmente, combinar estas duas técnicas, aplicando-as de acordo com a forma geométrica de cada objeto. A figura 6.23 ilustra dois objetos no qual foram aplicadas estas técnicas.

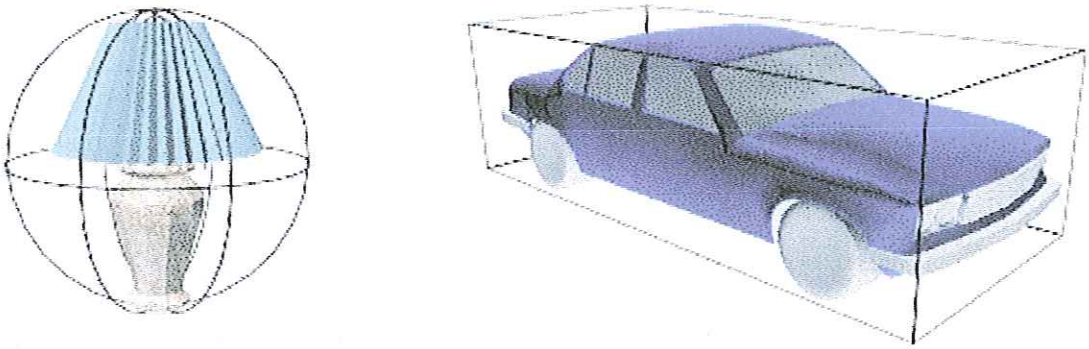


Figura 6.23 - Esfera e paralelepípedo circunscritos em objetos

Em certas aplicações gráficas, as partes do objeto passíveis de colisão são bem definidas. Nestes casos, pode-se empregar outra eficiente técnica, cuja idéia consiste na escolha de um ponto estratégico do qual será lançado um "raio interceptador". Em termos técnicos, o processo é feito a partir de um vetor direcional e das coordenadas universais do ponto estratégico, definindo-se a semi-reta na qual se deseja verificar a ocorrência de colisão. A análise de colisão é feita calculando-se as distâncias (na direção do vetor) deste ponto estratégico às faces interceptadas pela semi-reta. O procedimento matemático implementado no método é apresentado a seguir:

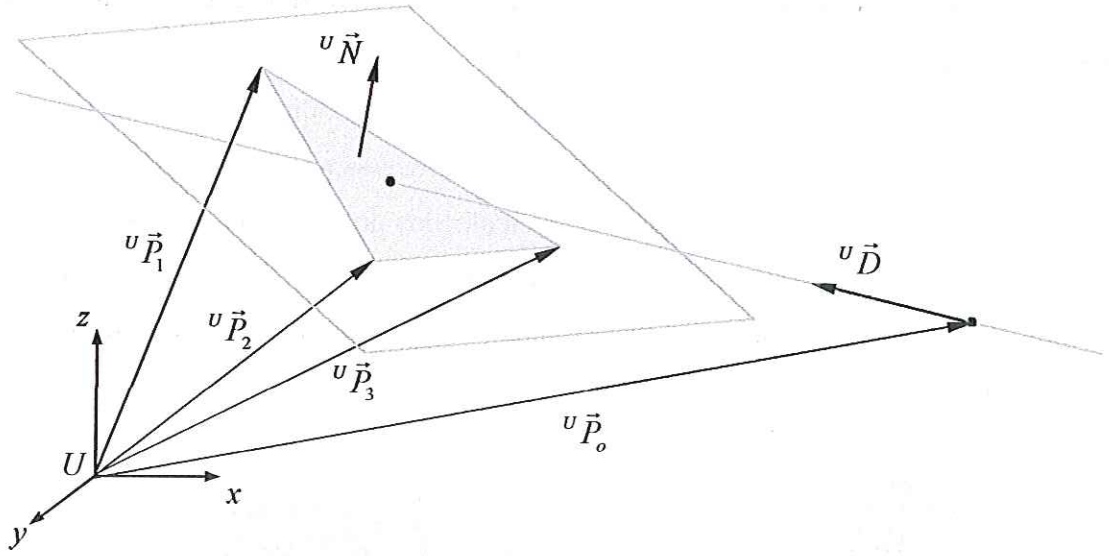


Figura 6.24 – Ilustração da técnica de colisão por meio de "raio interceptador"

Inicialmente, com as coordenadas universais do vetor direcional $^U\vec{D}=^U(d_x,d_y,d_z)$ e do ponto estratégico $^U\vec{P}_o=^U(p_{ox},p_{oy},p_{oz})$, define-se a equação paramétrica do raio:

$$\begin{cases} x = p_{ox} + d_x t \\ y = p_{oy} + d_y t \\ z = p_{oz} + d_z t \end{cases} \quad , \qquad t \geq 0$$

Em seguida, formulam-se as equações dos planos que contêm as faces sujeitas à interceptação da reta. Para cada face, define-se a equação:

$$n_x x + n_y y + n_z z - (n_x p_{1x} + n_y p_{1y} + n_z p_{1z}) = 0$$

onde n_x, n_y e n_z são as coordenadas universais do versor normal à face e p_{1x}, p_{1y} e p_{1z} são as coordenadas universais de um dos seus vértices.

Do sistema formado pelas equações anteriores, pode-se determinar o parâmetro t:

$$t = \frac{(n_x p_{1x} + n_y p_{1y} + n_z p_{1z}) - (n_x p_{ox} + n_y p_{oy} + n_z p_{oz})}{(n_x d_x + n_y d_y + n_z d_z)}$$

Se $t \geq 0$, então o raio intercepta o plano. Note que o denominador da equação acima representa o produto escalar do vetor normal ao plano com o vetor direcional do raio. Assim, se o raio for paralelo ao plano, o produto escalar destes vetores será nulo e, de fato, t não terá uma solução real.

Substituindo-se t na equação paramétrica do raio, encontram-se as coordenadas do ponto de interseção:

$$^{U}\begin{pmatrix} p_{\text{int}\,x} \\ p_{\text{int}\,y} \\ p_{\text{int}\,z} \\ 1 \end{pmatrix} = {}^{U}\begin{pmatrix} p_{ox} \\ p_{oy} \\ p_{oz} \\ 1 \end{pmatrix} + {}^{U}\begin{pmatrix} d_x \\ d_y \\ d_z \\ 1 \end{pmatrix} t$$

Dependendo do número de *bits* reservado para as variáveis, condições próximas ao paralelismo podem gerar o *overflow* das coordenadas do ponto de interseção, fazendo com que seja necessário impor algumas condicionais a fim de que se previnam tais problemas.

Antes do cálculo da distância, é necessário verificar se o ponto de interseção obtido está contido na face geradora do plano. Qualquer ponto no interior de um polígono pode ser descrito por uma combinação linear dos vértices formados deste polígono, como mostra a expressão abaixo:

$$^{U}\vec{P}_1 u + {}^{U}\vec{P}_2 v + {}^{U}\vec{P}_3 w = {}^{U}\vec{P}_{\text{int}}$$

O ponto de interseção estará dentro da face desde que as seguintes condições sejam satisfeitas:

$$0 \le u \le 1$$
$$0 \le v \le 1$$
$$0 \le w \le 1$$

Uma vez determinado um ponto de interseção dentro da face analisada, pode-se calcular a sua distância em relação ao ponto estratégico pela expressão:

$$Dist = \sqrt{(p_{\text{int}\,x} - p_{ox})^2 + (p_{\text{int}\,y} - p_{oy})^2 + (p_{\text{int}\,z} - p_{oz})^2}$$

6.7 Ordenação de *arrays*

A ordenação de *arrays* ou matrizes é muito usada na implementação de aplicações gráficas em tempo real, principalmente quando se deseja exibir os objetos virtuais segundo uma ordem seqüencial de *rendering*.

Existem diversos métodos de ordenação, uns mais eficientes do que outros. No entanto, considerando-se a capacidade de processamento computacional existente atualmente, para um número pequeno de elementos, qualquer método de ordenação é executado com extrema rapidez. No presente projeto, foi aplicado o método *Bublesort* (método da Bolha) para ordenar os elementos representantes das árvores do cenário virtual. Seu algoritmo é apresentado a seguir.

```
Para ( j = 0 ) até ( j < NroElementos – 1 ) faça
    Para ( i = j+1 ) até ( i < NroElementos ) faça
        Se ( Elemento[j] > Elemento[i] ) então
            temp ← Elemento[i];
            Elemento[i] ← Elemento[j];
            Elemento[j] ← temp;
        Fim Se;
        i ← i+1;
    Fim Para;
    j ← j+1;
Fim Para;
```

onde *NroElementos* é o número de elementos do *array Elementos* e *temp* é uma variável temporária. A ordenação do *Bublesort* é feita por meio de dois laços. O laço externo determina qual elemento do *array* será usado como referência. O laço interno compara cada elemento com a referência e, quando encontrar um menor do que esta, efetua a troca.

6.8 Conceituação geral sobre o funcionamento de um jogo

Em um desenho animado, uma seqüência de imagens é apresentada sob uma determinada taxa de amostragem para que se obtenha o efeito de animação. O funcionamento de um jogo ou ambiente virtual em tempo real é muito semelhante ao de um desenho animado, diferenciando-se apenas no fato do usuário poder direcionar ou até mesmo alterar, a cada momento, a seqüência de imagens apresentada pelo computador. Desta forma, pode-se dizer que um jogo é caracterizado por propiciar, em tempo real, a interação do usuário com o cenário virtual.

Em um jogo 3D, as imagens são geradas a cada iteração de acordo com o que é processado no seu *loop* principal. No entanto, até que cada ciclo esteja totalmente concluído, a imagem em construção permanece armazenada em um espaço de memória conhecido como *back buffer*. Quando todos os cálculos do ciclo forem processados e a imagem estiver completamente construída, o conteúdo do *back buffer* é copiado para um outro *buffer* de memória, chamado de *front buffer*. Este *buffer* está localizado na memória do dispositivo (ou placa) de vídeo e seu conteúdo é apresentado diretamente na tela do monitor. Concluído o ciclo, o *back buffer* é esvaziado e o processo é repetido.

Esta técnica de transferência do conteúdo do *back buffer* para o *front buffer* é conhecida como *page flipping*. Sem este recurso o usuário veria os objetos sendo construídos durante a execução de cada ciclo, o que não é desejado.

É importante citar que a limpeza do conteúdo do *back buffer* não implica na limpeza do conteúdo do *front buffer*, ou seja, a imagem exibida na tela do monitor permanece inalterada até que uma nova imagem seja armazenada (sobreposta) no *front buffer*.

De forma simplificada, o algoritmo de um jogo 3D apresenta a seguinte estrutura:

- *Declarações das variáveis globais*
- *Criação de uma janela de aplicação*
- *Inicializações das variáveis que fazem o interfaceamento com o hardware*
- *Carregamento dos arquivos (malhas, texturas e outros)*

- *Loop principal do jogo*
  - *Leitura dos dados provenientes dos dispositivos de entrada*
  - *Cálculos físicos (se existirem)*
  - *Aplicação das transformações geométricas e de visualização*
  - *Eliminação dos polígonos não visíveis*
  - *Exclusão dos polígonos que estão fora do campo de visão*
  - *Projeção dos polígonos sobre a superfície de projeção*
  - *Processamento do depth-buffer, texturização e shading*
  - *Apresentação dos gráficos na janela de aplicação*
  - *Processamento dos efeitos sonoros*
- *Fim do Loop*
- *Fechamento da janela de aplicação*
- *Liberação da memória ocupada pelas variáveis*

6.9 A exportação dos modelos no formato *.x (X file format)*

Arquivos com este formato podem guardar dados referentes aos modelos tridimensionais e as relações de dependência (hierarquia de dados) entre dois ou mais modelos para fins de animação.

Os *X files* são dirigidos por *templates*, que podem ser definidas e configuradas pelo usuário. Uma *template* é uma definição da maneira como os dados de um objeto devem ser alojados no arquivo. Obviamente, pelo fato do *X file* ser um formato de arquivo criado para trabalhar com *DirectX*, há muitas *templates* já pré-definidas e reconhecidas pelo componente *Direct3D*. As mais utilizadas são:

Header, Vector, Coords2d, Quaternion, Matrix4x4, ColorRGBA, ColorRGB, Indexed Color, Boolean, Material, TextureFilename, MeshFace, MeshFaceWraps, MeshTextureCoords, MeshNormals, MeshVertexColors, MeshMaterialList, Mesh, FrameTransformationMatrix, Frame, FloatKeys, TimedFloatKeys, AnimationKey, AnimationOptions, Animation e *AnimationSet*.

Para uma melhor compreensão, considere, como exemplo, a exportação de um cubo modelado e mapeado no *3d studio max*. O cubo foi gerado nas dimensões 100x100x100,

com centro na origem do sistema de coordenadas. Todas as suas faces foram mapeadas com a mesma textura, conforme mostra a figura 6.27:

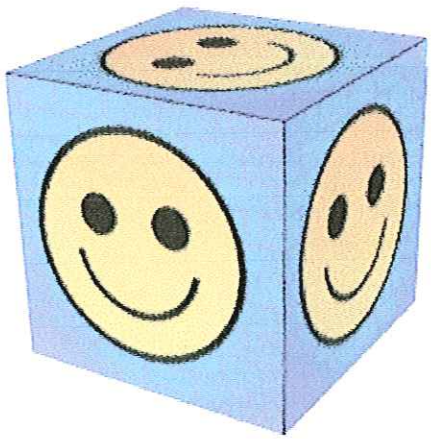


Figura 6.25 - Objeto criado no *3d studio max* e exportado no formato *X file*

Neste caso, o *X file* gerado tem a seguinte estrutura:

```
xof 0303txt 0032

Material PDX1_-_Default {
 1.000000;1.000000;1.000000;1.000000;;
 3.200000;
 0.000000;0.000000;0.000000;;
 0.000000;0.000000;0.000000;;

 TextureFilename  {
  "imagem.bmp";
 }
}

Mesh Cubo {
 36;
 -50.000000;-50.000000;50.000000;,
 50.000000;50.000000;50.000000;,
 -50.000000;50.000000;50.000000;,
 50.000000;50.000000;50.000000;,
 -50.000000;-50.000000;50.000000;,
 50.000000;-50.000000;50.000000;,
 -50.000000;-50.000000;-50.000000;,
 50.000000;50.000000;-50.000000;,
 50.000000;-50.000000;-50.000000;,
 50.000000;50.000000;-50.000000;,
 -50.000000;-50.000000;-50.000000;,
 -50.000000;50.000000;-50.000000;,
 -50.000000;-50.000000;50.000000;,
 50.000000;-50.000000;-50.000000;,
```

```
 50.000000;-50.000000;50.000000;,
 50.000000;-50.000000;-50.000000;,
 -50.000000;-50.000000;50.000000;,
 -50.000000;-50.000000;-50.000000;,
 50.000000;-50.000000;50.000000;,
 50.000000;50.000000;-50.000000;,
 50.000000;50.000000;50.000000;,
 50.000000;50.000000;-50.000000;,
 50.000000;-50.000000;50.000000;,
 50.000000;-50.000000;-50.000000;,
 50.000000;50.000000;50.000000;,
 -50.000000;50.000000;-50.000000;,
 -50.000000;50.000000;50.000000;,
 -50.000000;50.000000;-50.000000;,
 50.000000;50.000000;50.000000;,
 50.000000;50.000000;-50.000000;,
 -50.000000;50.000000;50.000000;,
 -50.000000;-50.000000;-50.000000;,
 -50.000000;-50.000000;50.000000;,
 -50.000000;-50.000000;-50.000000;,
 -50.000000;50.000000;50.000000;,
 -50.000000;50.000000;-50.000000;;
 12;
 3;0,1,2;,
 3;3,4,5;,
 3;6,7,8;,
 3;9,10,11;,
 3;12,13,14;,
 3;15,16,17;,
 3;18,19,20;,
 3;21,22,23;,
 3;24,25,26;,
 3;27,28,29;,
 3;30,31,32;,
 3;33,34,35;;

 MeshMaterialList  {
  1;
  12;
  0,
  0,
  0,
  0,
  0,
  0,
  0,
  0,
  0,
  0,
  0,
  0;
  { PDX1_-_Default }
 }

 MeshTextureCoords  {
  36;
  1.000000;1.000000;,
```

```
 0.000000;0.000000;,
 1.000000;0.000000;,
 0.000000;0.000000;,
 1.000000;1.000000;,
 0.000000;1.000000;,
 0.000000;1.000000;,
 1.000000;0.000000;,
 1.000000;1.000000;,
 1.000000;0.000000;,
 0.000000;1.000000;,
 0.000000;0.000000;,
 0.000000;1.000000;,
 1.000000;0.000000;,
 1.000000;1.000000;,
 1.000000;0.000000;,
 0.000000;1.000000;,
 0.000000;0.000000;,
 0.000000;1.000000;,
 1.000000;0.000000;,
 1.000000;1.000000;,
 1.000000;0.000000;,
 0.000000;1.000000;,
 0.000000;0.000000;,
 0.000000;1.000000;,
 1.000000;0.000000;,
 1.000000;1.000000;,
 1.000000;0.000000;,
 0.000000;1.000000;,
 0.000000;0.000000;,
 0.000000;1.000000;,
 1.000000;0.000000;,
 1.000000;1.000000;,
 1.000000;0.000000;,
 0.000000;1.000000;,
 0.000000;0.000000;;
}
}
```

Segue-se uma breve explanação sobre a estruturação deste arquivo:

A primeira linha do arquivo descreve que se trata de um *X file* (xof), exportado no formato texto (txt) e com 32bits (0032) reservados para as variáveis declaradas como pontos flutuantes. O arquivo também poderia ser exportado nos formatos binário (bin), binário comprimido (bzip) e texto comprimido (tzip). Além da possibilidade de exportação em 32bits também seria possível fazê-lo com 64 bits (0064), aumentando a precisão dos pontos flutuantes.

A *template Material* armazena os dados do material aplicado às faces da malha. As quatro primeiras linhas do corpo desta *template* referem-se as propriedades do material refletir, absorver ou emitir luz. A primeira e a terceira linha indicam, respectivamente, as

cores das luzes difusa e especular refletidas pela superfície da malha. A Segunda linha indica o valor da intensidade do efeito *highlight*. A quarta linha indica a cor da luz emitida pelo material. Por fim, a presença do componente *TextureFilename* indica que, além das propriedades de superfície do material, também existe uma textura aplicada sobre as faces. Este componente é opcional; caso ele não esteja presente, a malha não apresentará uma textura mapeada.

A *template Mesh* armazena os dados da malha "Cubo". A primeira linha do corpo desta *template* indica o número de vértices (36) necessários para a construção da malha. De imediato, poderia-se pensar que este valor não esteja correto, já que um cubo possui oito vértices. No entanto, a malha é gerada face por face, onde cada face é composta por 3 vértices. Como o processo de geração é seqüencial, vértices com identificações diferentes podem ser definidos na mesma posição. Após esta primeira linha são apresentadas as coordenadas cartesianas de cada vértice do cubo, definidos em relação ao sistema de coordenadas universal. Note que o caracter "," separa um vértice do outro. A linha após ";;" indica o número de faces (12) da malha. Logo abaixo são apresentadas as descrições das 12 faces da malha, cada uma definida em uma linha, onde o número 3 indica que a face é triangular e os demais números representam os índices dos vértices formadores da face.

A *template MeshMaterialList* armazena os dados referentes às faces nas quais foram aplicados os materiais e as texturas. A primeira linha do seu corpo indica o número de materiais aplicados à malha (apenas um, no caso). A segunda linha indica o número de faces que contém este material. No caso deste cubo, todas as 12 faces contêm o mesmo material. Em seguida são apresentados os índices dos materiais aplicados à cada face. Ao material em questão, por ser o primeiro e único, foi atribuído o índice 0. Por fim, após o caracter ";", é feita uma referência ao material descrito na *template Material*.

A *template MeshTextureCoords* armazena os dados referentes ao mapeamento da textura nas faces da malha. A primeira linha do corpo desta *template* indica o número de vértices usados neste mapeamento. As demais linhas representam as coordenadas de cada vértice, definidas no espaço da textura. No caso deste cubo, a malha inteira está mapeada, pois todos os vértices foram usados.

7 MODELAGEM DO AMBIENTE VIRTUAL

O processo de modelagem de um objeto virtual está diretamente relacionado ao tipo de aplicação para a qual ele foi criado. Na área de entretenimento, por exemplo, a conceituação adotada na modelagem de malhas para o desenvolvimento de filmes difere bastante da adotada no desenvolvimento de jogos.

Geralmente, em um modelo desenvolvido para filmes, não há uma preocupação com o número de polígonos ou com o tamanho das texturas que serão mapeadas sobre sua malha. De fato, é desejado que tais malhas sejam modeladas com uma grande quantidade de faces, o que contribui para a geração de cenas mais realísticas. O *rendering* é feito com altíssima qualidade, geralmente com técnicas de *raytracing*, onde os cálculos são processados por meio de *software*. Este procedimento é muito lento, podendo levar horas para que um único *frame* (quadro) seja finalizado.

Já em um modelo desenvolvido para jogos, a redução do número de polígonos é uma preocupação constante, pois, diferentemente do caso anterior, é necessário que muitos *frames* sejam "renderizados" a cada segundo, a fim de que a interação, em tempo real, entre o usuário e o ambiente virtual seja satisfatória. Desta forma, há um comprometimento entre a qualidade gráfica e número de polígonos da malha em construção. Não é uma surpresa que artistas gráficos capazes de modelar objetos com poucos polígonos e alta qualidade são muito cobiçados por empresas desenvolvedoras de jogos.

Em ambientes gráficos que operam em tempo real, as texturas desempenham um papel fundamental. Além de fornecerem maior realismo ao cenário, as texturas contribuem para a redução do número de polígonos dos modelos, uma vez que muitos de seus detalhes podem ser desenhados diretamente sobre as mesmas ao invés de serem modelados por meio de vértices e faces. Uma textura bem mapeada pode fornecer ao usuário a impressão de estar interagindo com um cenário tridimensional bastante detalhado e ainda assim tornar mais leve o processamento dos gráficos.

O desenvolvimento de um jogo para computador requer a elaboração de um *storyboard* ou, pelo menos, de alguns esboços que permitam visualizar, em um primeiro momento, o que será criado. São efetuados, posteriormente, os processos de modelagem e mapeamento das texturas que, no caso deste projeto, foram feitos com o uso dos *softwares 3d studio max* e *Paint Shop Pro*.

7.1 Modelos estáticos indeformáveis

Modelos estáticos indeformáveis são aqueles que não sofrem transformações geométricas e que conservam, ao longo do tempo, as distâncias relativas entre os seus vértices. De forma mais objetiva, esses modelos podem ser tratados como corpos rígidos que não se movimentam pelo espaço tridimensional.

Salvo algumas exceções, praticamente todo o cenário do ambiente virtual é constituído de modelos estáticos indeformáveis. Por não sofrerem translações ou rotações, estes modelos são gerados e posicionados de acordo com a arquitetura definida para cenário virtual, ou seja, antes de serem exportados, eles já devem estar adequadamente posicionados dentro do ambiente do *software* de modelagem.

Apesar de ser possível exportar todo o cenário como uma única malha, não se deve, para fins de performance, efetuar este procedimento. Exportá-lo como um conjunto de pequenas malhas é altamente recomendado, já que, durante o processo de *rendering*, podem ser excluídas as malhas que estiverem fora do volume de visualização.

A figura 7.1 ilustra o processo de concepção, modelagem e mapeamento de alguns modelos do cenário virtual.

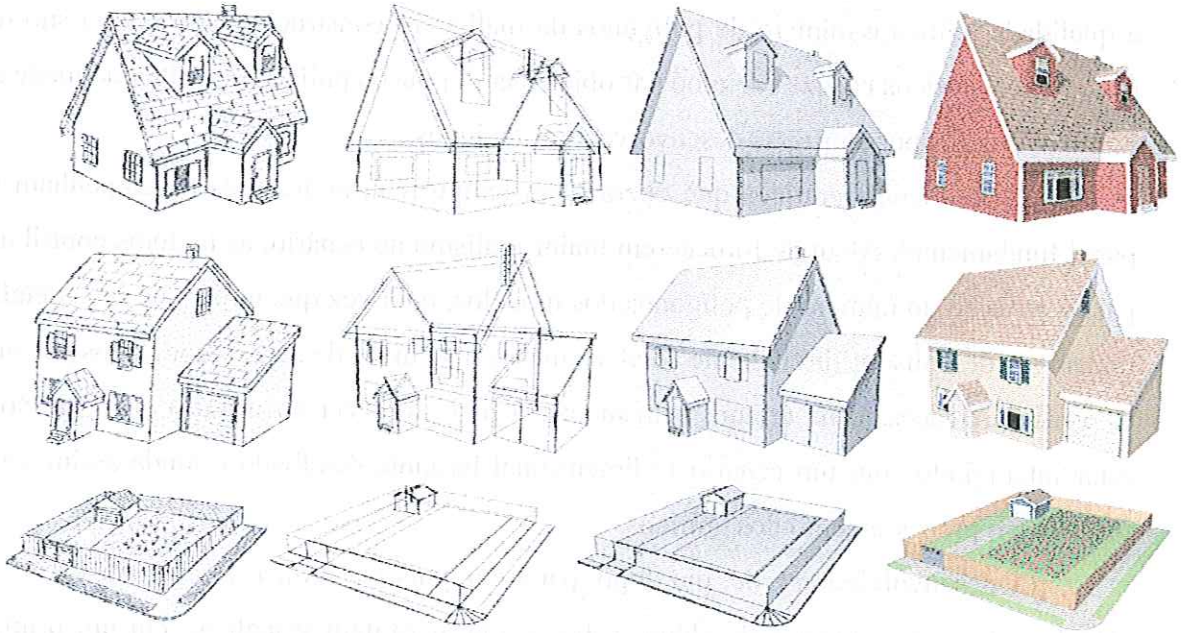


Figura 7.1 - Concepção, modelagem e mapeamento de modelos do cenário virtual

A figura 7.2 ilustra uma parte do cenário composta por alguns dos modelos apresentados na figura anterior.

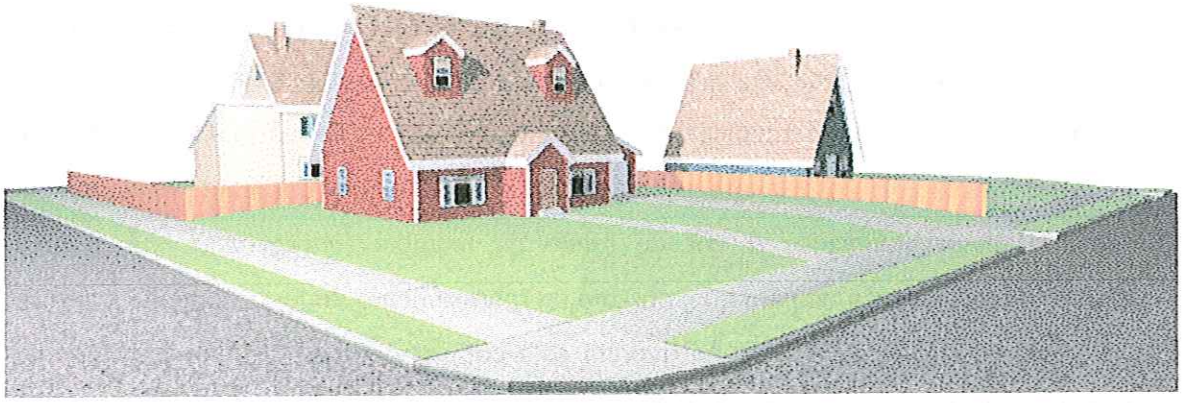


Figura 7.2 - *Rendering* de parte do cenário virtual

7.2 Modelos dinâmicos indeformáveis

Embora conservem, ao longo do tempo, as distâncias relativas entre os seus vértices, os modelos dinâmicos indeformáveis podem ser transladados ou rotacionados dentro do ambiente virtual. São, assim, tratados como corpos rígidos que se movimentam pelo espaço tridimensional.

Neste projeto, os modelos que se enquadram nesta categoria são as árvores e os componentes da bicicleta (rodas, quadro e o conjunto garfo + guidom).

7.2.1 Árvores

De imediato, pode parecer estranho que as árvores deste projeto sejam classificadas como modelos dinâmicos indeformáveis. Desconsiderando-se os efeitos dos ventos, poderia-se pensar que as árvores deveriam ser tratadas como modelos estáticos, uma vez que as posições dos seus vértices permanecem inalteradas ao longo do tempo. No entanto, para que um modelo estático de árvore tenha uma aparência realística, sua malha deve apresentar um número muito grande de faces, já que cada folha seria modelada com pelo menos um polígono. Neste caso, se for desejada a presença de várias árvores no cenário, o processamento de todos esses polígonos seria demasiadamente lento, comprometendo a interação do usuário com o ambiente virtual.

Uma solução razoável para este problema foi obtida com a implementação de uma técnica que fornece ao usuário a ilusão de estar interagindo com um cenário onde existam árvores altamente detalhadas, embora, na verdade, elas sejam compostas por poucos polígonos. A técnica consiste, basicamente, em modelar árvores planas e rotacioná-las de forma que estas sempre permaneçam de frente para a visão do usuário. O detalhamento dos galhos e folhas é obtido pela aplicação de uma textura (contendo a imagem de uma árvore) sobre o plano modelado. A cor de fundo da imagem (*background color*) deve ser estrategicamente definida para que, durante o processamento do jogo, somente a árvore seja apresentada, isto é, os *texels* que tiverem esta cor serão tratados como transparentes. A figura 7.3 ilustra a textura de uma árvore.

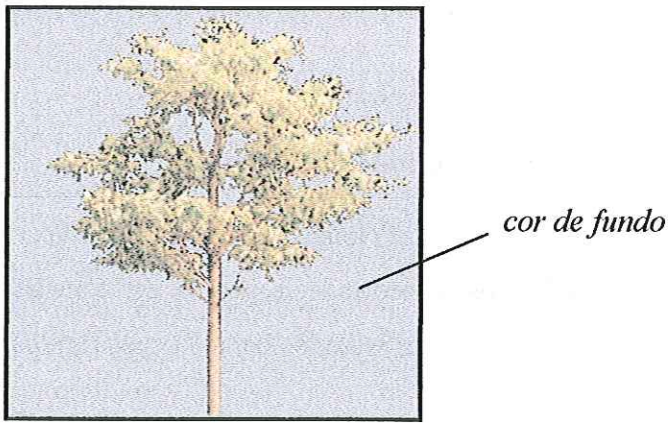


Figura 7.3 - Textura de uma árvore

Conhecidas as posições da câmera (ponto de projeção), das árvores e suas orientações iniciais no cenário tridimensional, pode-se definir um procedimento matemático para que as transformações geométricas necessárias sejam efetuadas. Considere a seguinte figura:

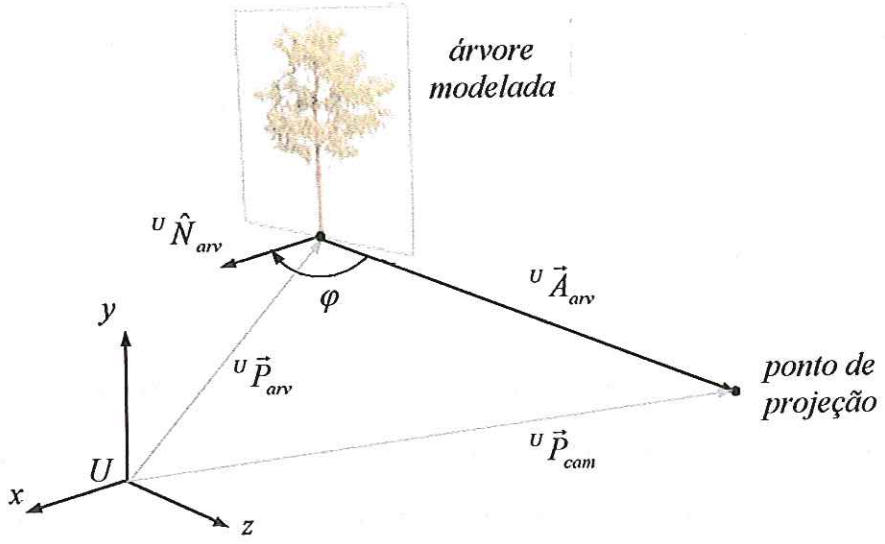


Figura 7.4 – Esquematização vetorial do posicionamento da árvore em relação ao observador

Segundo a figura 7.4, os vetores ${}^{U}\vec{P}_{cam}$, ${}^{U}\vec{P}_{arv}$ e ${}^{U}\hat{N}_{arv}$ representam respectivamente a posição do ponto de projeção, a posição da árvore e seu versor normal inicial. A partir destas grandezas vetoriais, pode-se determinar o vetor direcional do ponto de projeção em relação à árvore:

$$ {}^{U}\vec{A}_{arv} = {}^{U}\vec{P}_{cam} - {}^{U}\vec{P}_{arv} $$

Normalizando-se ${}^{U}\vec{A}_{arv}$, obtém-se o versor ${}^{U}\hat{A}_{arv}$:

$$ {}^{U}\hat{A}_{arv} = \frac{{}^{U}\vec{A}_{arv}}{\left|{}^{U}\vec{A}_{arv}\right|} $$

Pode-se então determinar o ângulo entre a direção de observação da árvore e sua direção normal por meio da seguinte expressão:

$$ \varphi = \cos^{-1}\left({}^{U}\hat{A}_{arv} \bullet {}^{U}\hat{N}_{arv}\right) \qquad \text{(I)} $$

Há, no entanto, um problema em se usar diretamente o valor deste ângulo no cálculo da transformação de rotação da árvore. Devido às inclinações do cenário, a direção do vetor ${}^{U}\vec{A}_{arv}$ pode não paralela ao plano *xz* do sistema de coordenadas universais. Nestes casos, o ângulo obtido pela formulação anterior corresponderia a uma rotação do plano da árvore em torno de um eixo não vertical, o que não seria adequado.

Para que cada árvore seja rotacionada somente ao redor do seu eixo vertical (paralelo ao eixo y do sistema de coordenadas universais), é necessário anular o valor da componente vertical (y) do vetor ${}^{U}\vec{A}_{arv}$, antes da sua normalização:

$$ {}^{U}\vec{A}_{arv} = {}^{U}\begin{pmatrix} a_{arvx} \\ 0 \\ a_{arvz} \\ 1 \end{pmatrix} $$

Devido as limitações da função $\cos^{-1}$, a equação (I) sempre fornecerá como resultado o menor ângulo entre os versores ${}^{U}\vec{A}_{arv}$ e ${}^{U}\hat{N}_{arv}$, o que não evidencia a escolha do sentido de rotação para as árvores. Uma solução para o problema consiste em orientar inicialmente todas as árvores da mesma forma, de modo que seus versores normais iniciais sejam idênticos. No caso deste projeto, tais versores foram inicialmente definidos na direção e sentido do eixo x do sistema de coordenadas universais. Com isso, o sentido de rotação pode ser determinado com base no sinal da componente z do versor ${}^{U}\hat{A}_{arv}$. Para um maior esclarecimento deste processo, abaixo são apresentados alguns exemplos mostrando quando o sentido de rotação deve ser tratado como positivo e quando deve ser tratado como negativo.

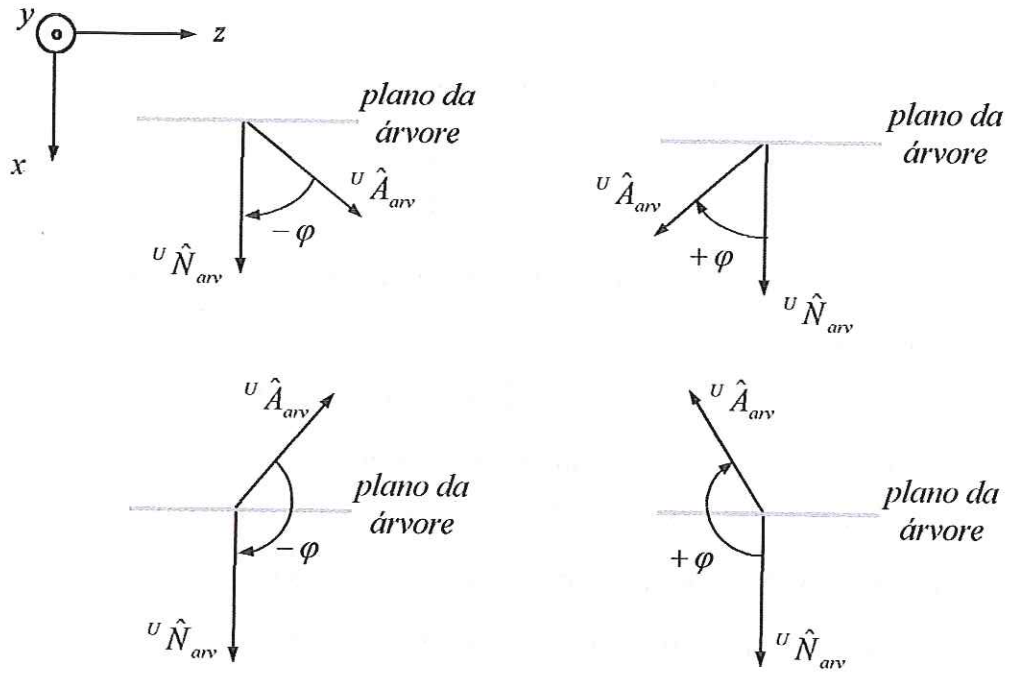


Figura 7.5 – Aplicação do sentido de rotação em diferentes casos

Note que o ângulo é negativo somente quando a componente z de ${}^{U}\hat{A}_{arv}$ for positiva. Para solucionar esse inconveniente, basta impor a seguinte condicional algorítmica antes que as transformações sejam efetuadas:

$$Se\ a_{arvz} > 0\ então\ \varphi = -\varphi$$

Após o cálculo do ângulo φ e a verificação do sentido de rotação, são efetuadas as transformações geométricas que posicionam a árvore de frente para a visão do usuário. O procedimento é simples:

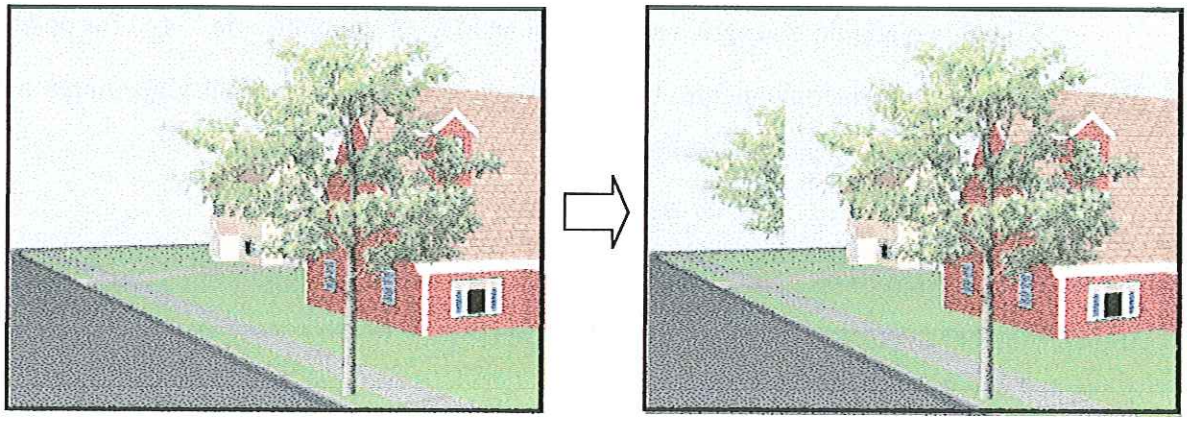
- Translada-se a árvore para a origem do sistema de coordenadas universais.
- Rotaciona-se a árvore do ângulo φ.
- Translada-se novamente a árvore para a sua posição original.

A transformação completa pode ser representada pela seguinte composição matricial, aplicada em cada vértice da malha da árvore:

$${}^{U}\vec{V}_f = T_{parv}.R_{\varphi,y}.T_{-parv}.{}^{U}\vec{V}_i$$

Para que a técnica ofereça bons resultados, o eixo de rotação deve coincidir com o eixo longitudinal que passa pelo centro do caule. Percebe-se que quanto mais simétrica for a imagem da árvore, melhores serão os resultados.

O processo de *rendering* das árvores é seqüencial, ou seja, elas são processadas e apresentadas na tela uma por vez. A efetuação da transparência dos *texels* da textura de um objeto é valida somente durante o seu *rendering*. Assim, quando outro modelo é posteriormente processado, não mais é checada a transparência da textura do modelo anterior. Podem, então, surgir problemas quando existem árvores sobrepostas dentro do volume de visualização. Veja o que acontece ao se executar, sucessivamente, o *rendering* de duas árvores sobrepostas, de forma que a primeira a ser processada encontra-se à frente da segunda:



rendering da primeira árvore *rendering da segunda árvore*

Figura 7.6 - *Rendering* sucessivo de duas árvores sobrepostas

O que ocorreu? Durante o *rendering* da segunda árvore, a transparência dos *texels* da primeira não mais foi checada e o algoritmo de *depth-buffer* determinou que os pontos desta estão mais próximos do ponto de projeção do que os pontos da segunda árvore, mantendo, assim, a coloração dos *pixels* correspondentes aos *texels* da primeira.

A solução para o problema é ordenar, a cada *frame*, a seqüência de *rendering* das árvores presentes no volume de visualização, de acordo com a profundidade de cada uma, de forma que a árvore que estiver mais longe do ponto de projeção seja a primeira a ser processada, e a que estiver mais perto, a última.

Como o número de árvores presentes dentro do volume de visualização é normalmente pequeno, qualquer método de ordenação fornece os resultados com extrema rapidez. Neste projeto, usou-se o método *buble sort*, cujo algoritmo foi apresentado no capítulo anterior.

A ordenação adequada da seqüência de *rendering* das duas árvores anteriores produz a seguinte imagem:

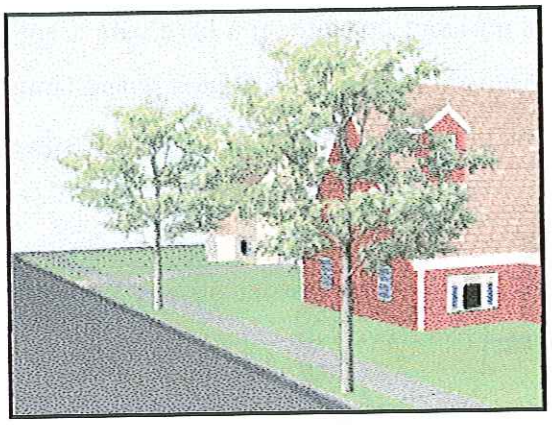


Figura 7.7 - Imagem gerada corretamente (após a ordenação da seqüência de *rendering*)

7.2.2 A bicicleta virtual

A bicicleta é o elemento virtual por meio do qual o usuário interage com o ambiente tridimensional. Qualquer ação sobre os dispositivos de entrada é diretamente refletida nela. Ela é quem sofre as colisões e possibilita, de acordo com as suas limitações mecânicas, a movimentação do usuário pelo ambiente tridimensional. A figura 7.8 ilustra o processo de concepção, modelagem e mapeamento da bicicleta.

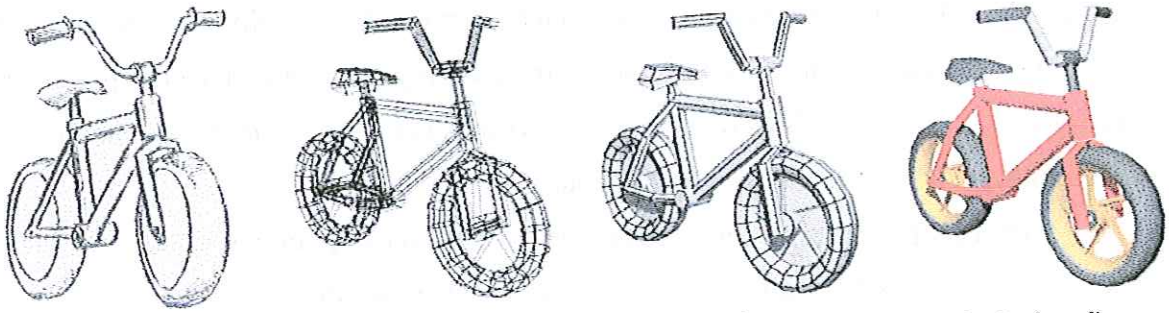
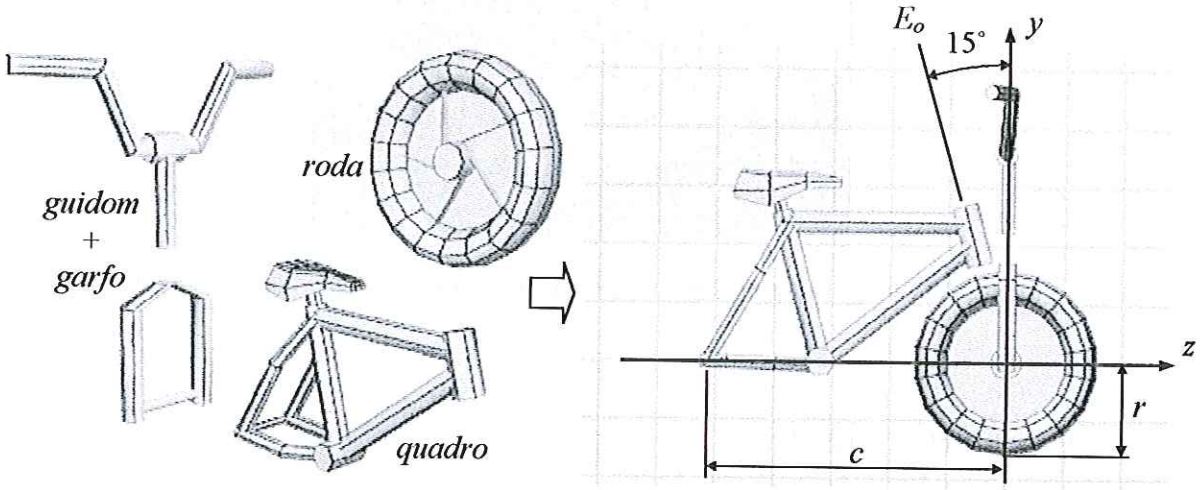


Figura 7.8 - Concepção, modelagem e mapeamento da bicicleta virtual

A bicicleta é composta por quatro corpos rígidos que podem sofrer transformações geométricas de translação e rotação ao longo do tempo. São eles: as rodas dianteira e traseira, o quadro e o conjunto guidom + garfo. Antes de ser exportada, cada peça modelada deve estar posicionada estrategicamente dentro do cenário tridimensional, uma

vez que os valores destas posições constituem a base para a aplicação das transformações geométricas. A figura 7.9 mostra estes corpos rígidos separadamente e suas posições dentro do ambiente do *software* de modelagem, imediatamente antes de serem exportados.



a) corpos rígidos que compõe a bicicleta

b) posicionamento inicial dos corpos rígidos no cenário tridimensional

Figura 7.9 - Componentes da bicicleta e suas posições iniciais antes da exportação

Apesar de serem corpos distintos, o guidom e o garfo são tratados como um único corpo rígido, pois sofrem as mesmas transformações geométricas. Antes de serem exportados, eles são posicionados verticalmente, como apresentado na figura 7.9.b. Este procedimento visa facilitar a aplicação das transformações, já que, devido à geometria da bicicleta, somente é desejado rotacionar estes corpos em torno de um eixo oblíquo (E_o) em relação à reta que contém os centros das rodas.

Para proporcionar maior eficiência no processamento, em tempo real, dos modelos virtuais, é recomendado privá-los de um alto nível de detalhamento. Neste contexto, as rodas dianteira e traseira são tratadas como idênticas. Assim, não há a necessidade das duas serem modeladas, já que uma pode ser obtida a partir da translação da outra. No caso deste projeto, apenas a roda dianteira foi modelada, com seu centro na origem do sistema de coordenadas e seu eixo transversal perpendicular ao plano *yz*. O quadro da bicicleta foi posicionado de modo que os centros das rodas traseira e dianteira estejam sobre o eixo *z* do sistema de coordenadas, como mostrado na figura 7.9.b.

O posicionamento inicial de cada componente da bicicleta no cenário virtual do jogo corresponde exatamente ao seu posicionamento no ambiente do *software* de modelagem, imediatamente antes da sua exportação.

7.2.2.1 Os eixos da bicicleta

Para que os cálculos sejam efetuados, é necessário definir alguns eixos para a bicicleta. A representação destes eixos é feita a partir de versores, como mostrado na figura 7.10.

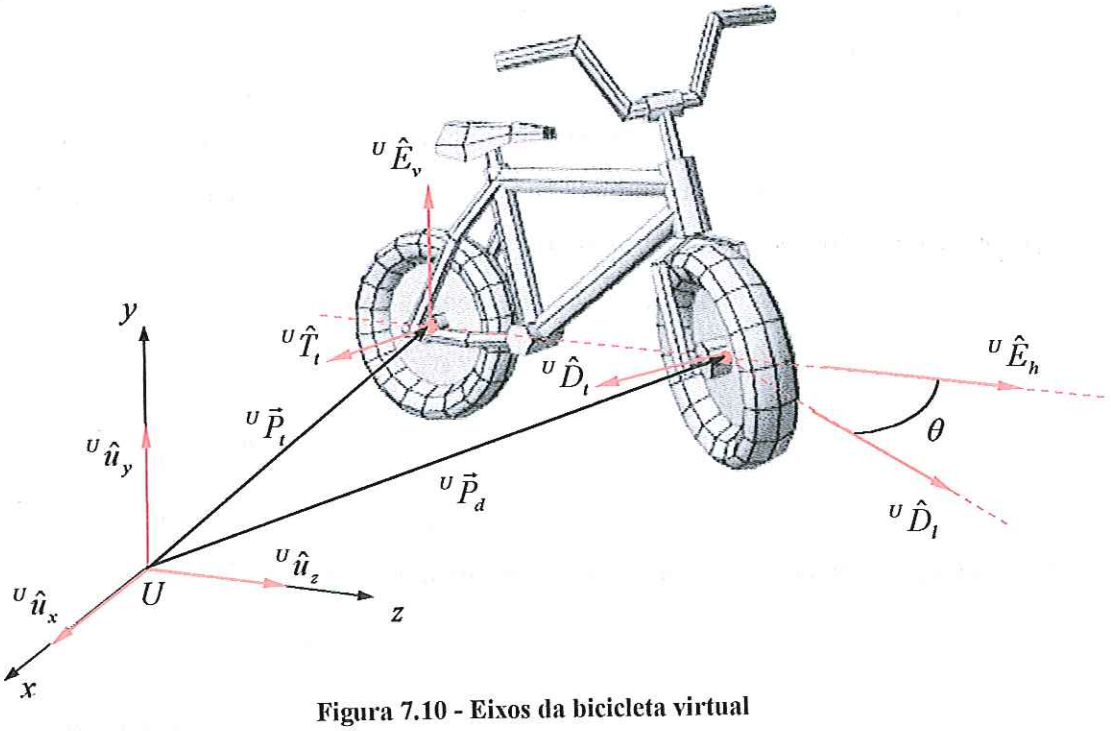


Figura 7.10 - Eixos da bicicleta virtual

Segue-se uma descrição destes versores:

- ${}^{U}\hat{E}_h$ → representa o eixo longitudinal horizontal da bicicleta e é determinado pela normalização do vetor diferença das posições dos centros das rodas dianteira e traseira.
- ${}^{U}\hat{E}_v$ → representa o eixo longitudinal vertical da bicicleta e inicialmente é definido como igual ao versor ${}^{U}\hat{u}_y$.

- ${}^{U}\hat{T}_t$ → representa o eixo transversal da roda traseira e é determinado pelo produto vetorial de ${}^{U}\hat{E}_v$ com ${}^{U}\hat{E}_h$.
- ${}^{U}\hat{D}_l$ → representa o eixo longitudinal da roda dianteira e é determinado pela rotação de ${}^{U}\hat{E}_h$ em torno de ${}^{U}\hat{E}_v$.
- ${}^{U}\hat{D}_t$ → representa o eixo transversal da roda dianteira e é determinado pelo produto vetorial de ${}^{U}\hat{E}_v$ com ${}^{U}\hat{D}_l$.

7.2.2.2 As condições iniciais

Uma vez conhecido o posicionamento inicial da bicicleta no cenário tridimensional, deve-se implementar as condições iniciais necessárias à execução dos cálculos. Assim, com base na figura 7.9.b, são definidas as seguintes inicializações:

Versores geradores do sistema de coordenadas universal:

$${}^{U}\hat{u}_x = \begin{pmatrix}1\\0\\0\\1\end{pmatrix}^{U} \qquad {}^{U}\hat{u}_y = \begin{pmatrix}0\\1\\0\\1\end{pmatrix}^{U} \qquad {}^{U}\hat{u}_z = \begin{pmatrix}0\\0\\1\\1\end{pmatrix}^{U}$$

Posições iniciais dos centros das rodas dianteira e traseira:

$${}^{U}\vec{P}_d = \begin{pmatrix}0\\0\\0\\1\end{pmatrix}^{U} \quad \text{e} \quad {}^{U}\vec{P}_t = \begin{pmatrix}0\\0\\-c\\1\end{pmatrix}^{U},$$

onde c é a distância entre os centros das rodas

Velocidade inicial do centro da roda traseira:

$${}^{U}\vec{V}_t = \begin{pmatrix}0\\0\\0\\1\end{pmatrix}^{U}$$

Versores dos eixos longitudinais vertical e horizontal da bicicleta:

$$ {}^{U}\hat{E}_v = {}^{U}\begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \end{pmatrix} \quad \text{e} \quad {}^{U}\hat{E}_h = \frac{{}^{U}\vec{P}_d - {}^{U}\vec{P}_t}{\left| {}^{U}\vec{P}_d - {}^{U}\vec{P}_t \right|} $$

7.2.2.3 Dinâmica da bicicleta

Considerando-se os objetivos deste projeto e seus usuários finais (crianças portadoras de deficiência motora), vê-se que não há a necessidade de se implementar um cálculo físico minucioso, que considere as forças e torques envolvidos no equilíbrio e nas derrapagens da bicicleta em terrenos diversos. Além do mais, tal procedimento poderia proporcionar à criança deficiente uma grande dificuldade para controlar a bicicleta dentro do cenário virtual. Neste contexto, duas hipóteses foram assumidas:

a) A conservação do momento angular (responsável pelo equilíbrio da bicicleta quando em movimento) não foi considerada, assumindo-se que a criança sempre esteja equilibrada sobre a bicicleta.

b) O atrito presente entre os pneus e o terreno é sempre suficiente para impedir que bicicleta derrape, em quaisquer circunstâncias.

Vale ressaltar, no entanto, que a bicicleta não é tratada visualmente como um triciclo, já que, nos movimentos curvos, ela se inclina para garantir o equilíbrio dinâmico e proporcionar maior realismo à cena (essa abordagem será apresentada mais à frente).

Para elucidar melhor o raciocínio desenvolvido neste capítulo, considere, por hora, que o controle da bicicleta seja feito pelo teclado. No próximo capítulo será apresentado um mecanismo de controle cujos dados de entrada são provenientes de outros dispositivos, como potenciômetros e *encoders*.

As linguagens de programação apresentam funções que identificam quando uma determinada tecla foi pressionada. Desta forma, pode-se designar teclas específicas para virar, acelerar ou retardar o movimento da bicicleta. Considere que as quatro setas (teclas) do teclado tenham sido usadas para estas finalidades, de forma que o pressionamento de cada uma esteja associado ao valor *true* da variável booleana correspondente, de acordo com a tabela seguinte:

Tabela 7.1 – Teclas (setas) associcadas às váriáveis booleanas que controlam a bicicleta

Tecla	*Variável booleana*
left	esquerda
right	direita
up	frente
down	tras

A leitura dos dados de entrada é processada dentro do *loop* do jogo, a cada iteração. Assim, a rotação do guidom é realizada por meio de um processo incremental, com base nos valores das variáveis "esquerda" e "direita".

Considere que θ seja o ângulo correspondente à rotação do guidom, como mostra a figura 7.11:

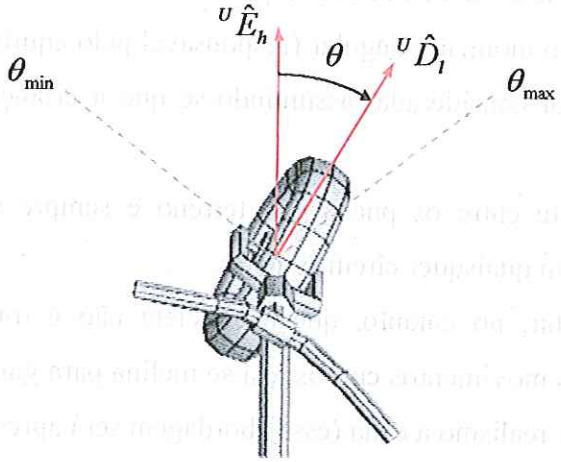


Figura 7.11 - Rotação do guidom

Pode-se facilmente efetuar a rotação do guidom com base no seguinte algoritmo:

```
Se (direita = true) então
    Se (θ ≤ θ_max ) então
        θ ← θ + θ_incr ;
    Fim Se
Fim Se
```

```
Se (esquerda = true) então
    Se (θ ≥ θ_min) então
        θ ← θ - θ_incr ;
    Fim Se
Fim Se
```

Neste algoritmo, o valor atribuído ao incremento θ_{incr} é dependente da velocidade de processamento do jogo, ou seja, se um grande número de *frames* é processado por segundo, θ_{incr} deve ser suficientemente pequeno para que a bicicleta não vire muito rapidamente. Normalmente este procedimento gera bons resultados, porém há casos em que o *fps* (*frames per second*) do jogo varia muito, produzindo uma inconstância no movimento de rotação do guidom. Pode-se solucionar este problema atualizando-se o ângulo θ a cada intervalo de tempo constante ao invés de a cada iteração.

As acelerações de um corpo resultam de suas propriedades inerciais e das forças que são aplicadas sobre ele. A aceleração é então uma conseqüência. Para simplificar a implementação dos cálculos, trabalharemos diretamente com esta conseqüência, ou seja, aplicaremos acelerações no lugar das forças.

Como os pneus da bicicleta não derrapam, apenas são consideradas as acelerações atuantes na direção da sua trajetória, ou seja, na direção do versor ${}^{U}\hat{E}_h$. A figura 7.12 ilustra as acelerações escalares resultantes das forças de tração e resistência aplicadas sobre a bicicleta.

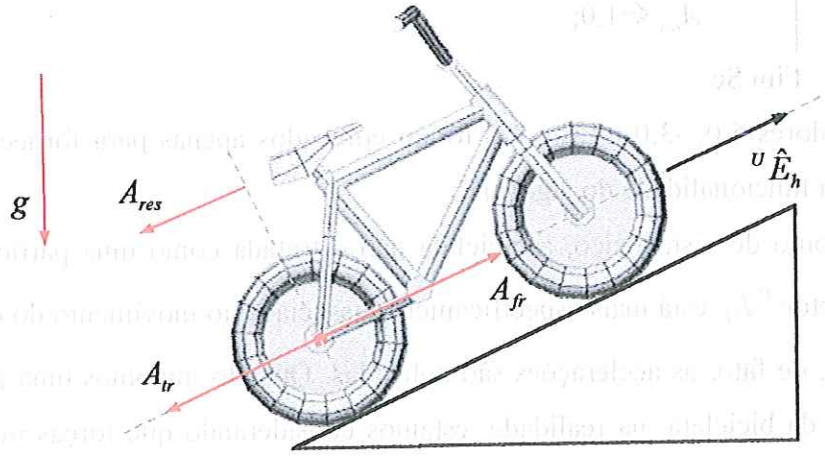


Figura 7.12 - Acelerações escalares resultantes das forças de tração e resistência aplicadas na bicicleta

Onde:

- g é a aceleração escalar da gravidade local.
- A_{fr} é a aceleração escalar resultante da força de tração para frente
- A_{tr} é a aceleração escalar resultante da força de tração para trás
- A_{res} é a aceleração escalar resultante das forças de resistência ao movimento.

O algoritmo que possibilita a aplicação destas acelerações é mostrado abaixo:

```
Se (frente = true) então
|       A_fr ← 5.0;
Senão
|       A_fr ←0.0;
Fim Se
Se (tras = true) então
|       A_tr ←-3.0;
Senão
|       A_tr ←0.0;
Fim Se
Se ( V_t > 0 ) então
|       A_res ←-1.0;
Senão
|       A_res ←1.0;
Fim Se
```

Os valores 5.0, -3.0, -1.0 e 1.0 foram colocados apenas para fornecer uma noção qualitativa da funcionalidade do algoritmo.

Do ponto de vista físico, a bicicleta não é tratada como uma partícula. Assim, a direção do vetor ${}^{U}\hat{E}_h$ está mais especificamente associada ao movimento do centro da roda traseira onde, de fato, as acelerações são aplicadas. Quando impomos uma aceleração em algum ponto da bicicleta, na realidade, estamos considerando que forças implícitas estão atuando sobre os seus componentes inerciais, de modo que desta atuação resulta a aceleração daquele ponto específico.

Neste contexto, todo o cálculo cinemático é efetuado sobre o centro da roda traseira, a partir do qual são determinados os movimentos das demais partes da bicicleta.

Consideremos, por hora, apenas o movimento do centro da roda traseira, restrito à trajetória definida pelo versor direcional ${}^{U}\hat{E}_h$, como mostrado na ilustração abaixo:

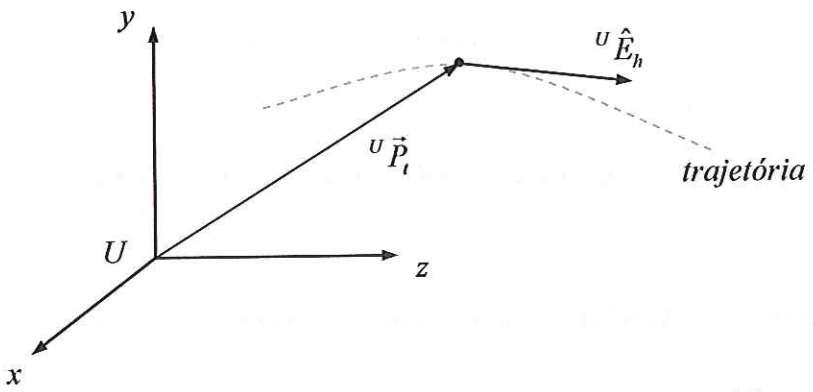


Figura 7.13 - Movimento do centro da roda traseira no espaço tridimensional

Uma vez que o movimento se restringe à direção de ${}^{U}\hat{E}_h$, pode-se considerar que o valor escalar da aceleração tangencial deste ponto resulta da soma escalar de todas as acelerações impostas nesta mesma direção. As acelerações de tração e resistência já são definidas na direção da trajetória, no entanto, é necessária a execução de um pequeno cálculo para considerar a influência da aceleração da gravidade no seu movimento.

Se g é o módulo da aceleração da gravidade, pode-se definir o seu vetor como:

$$ {}^{U}\vec{G} = -g\,{}^{U}\hat{u}_y $$

Basta, então, efetuar o produto escalar entre ${}^{U}\vec{G}$ e ${}^{U}\hat{E}_h$ para que se determine a componente escalar da aceleração gravitacional na direção da trajetória do centro da roda traseira.

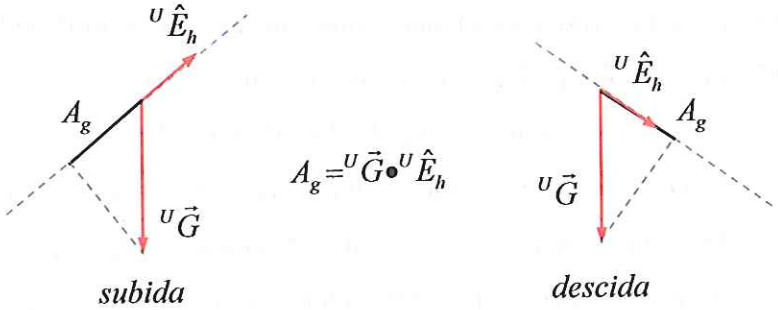


Figura 7.14 - Obtenção da componente escalar da aceleração gravitacional na direção da trajetória do centro da roda traseira

É importante não confundir valores escalares com valores modulares. Enquanto que estes últimos são obrigatoriamente positivos, os valores escalares podem ser negativos.

Definidas as acelerações na direção da trajetória, pode-se estabelecer um algoritmo incremental para calcular a posição do centro da roda traseira a cada iteração, como mostrado a seguir:

/ Cálculo da aceleração escalar tangencial do centro da roda traseira */*

$A_t \leftarrow A_{fr} + A_{tr} + A_{res} + A_g$;

/ Definição do vetor aceleração tangencial de acordo com a direção de* ${}^{U}\hat{E}_h$ **/*

${}^{U}\vec{A}_t \leftarrow A_t * {}^{U}\hat{E}_h$;

/ Definição do vetor velocidade de acordo com a direção de* ${}^{U}\hat{E}_h$ **/*

${}^{U}\vec{V}_t \leftarrow V_t * {}^{U}\hat{E}_h$;

/ Cálculo da posição vetorial do centro da roda traseira no cenário 3D */*

${}^{U}\vec{P}_t \leftarrow {}^{U}\vec{P}_t + {}^{U}\vec{V}_t * dt + 0.5 * {}^{U}\vec{A}_t * dt * dt$;

/ Cálculo da velocidade escalar do centro da roda */*

$V_t \leftarrow V_t + A_t * dt$;

onde *dt* representa o intervalo de tempo de cada iteração. Note que a movimentação do centro da roda traseira ao longo do tempo é composta de pequenos deslocamentos, nos quais a aceleração tangencial foi tomada como constante. É evidente que se trata de um procedimento físico bastante simplificado. Contudo, a alta taxa de amostragem faz com que esta abordagem produza bons resultados visuais, totalmente compatíveis com as finalidades do projeto.

Obviamente, as variáveis vetoriais apresentadas acima devem ser declaradas por meio de uma estrutura (ou classe) que armazene as suas coordenadas cartesianas. Em linguagem C, por exemplo, poder-se-ia definir a seguinte estrutura:

struct tipoVetor{ float x; float y; float z; };

onde qualquer variável declarada como *tipoVetor* teria três componentes reais. Há também a necessidade de se implementar funções que executem as operações vetoriais. Algumas *APIs* gráficas, como o *Direct3D*, já apresentam funções otimizadas para se trabalhar com vetores. Para maiores informações, consulte o apêndice A.

É importante esclarecer que as linguagens de programação não apresentam os caracteres utilizados na representação das variáveis vetoriais do algoritmo anterior. Eles foram usados apenas para ilustrar o desenvolvimento do raciocínio de forma mais clara. Na implementação do algoritmo, basta que se substituam tais caracteres por outros aceitos pela linguagem de programação.

Analisemos o movimento da bicicleta, agora tratada como um conjunto de corpos rígidos articulados. De acordo com a figura 7.15, mantendo-se o guidom rotacionado de θ, verifica-se que os centros das rodas traseira e dianteira descrevem trajetórias circulares concêntricas, de raios R_t e R_d respectivamente. Se $\theta \neq 0$, então $|R_t| < |R_d|$. Se $\theta = 0$, estes raios são infinitos e a trajetória da bicicleta é retilínea.

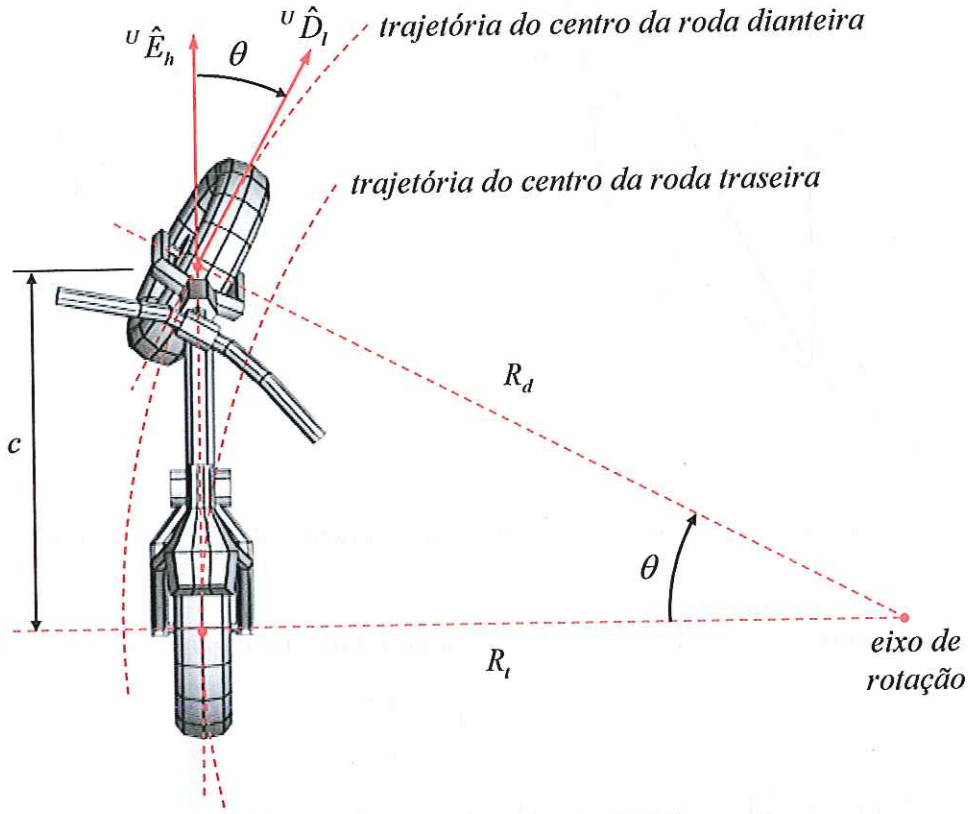


Figura 7.15 - Movimento da bicicleta em trajetória curvilínea

Da figura, R_t e R_d podem ser expressos, em função do ângulo θ, como abaixo:

$$R_t = \frac{c}{tg\theta} \qquad \text{e} \qquad R_d = \frac{c}{\text{sen}\,\theta}$$

O procedimento adotado para a determinação das posições dos centros das rodas nas respectivas trajetórias circulares é descrito a seguir:

A cada iteração, antes que os cálculos sejam efetuados, a posição do centro da roda traseira é temporariamente armazenada na variável ${}^{U}\vec{P}_{ti}$. O vetor posição ${}^{U}\vec{P}_{t}$ é então calculado na direção de ${}^{U}\hat{E}_{h}$, com base no método cinemático apresentado anteriormente. O comprimento dP_t do vetor deslocamento ${}^{U}d\vec{P}_{t}$, definido pela diferença entre ${}^{U}\vec{P}_{t}$ e ${}^{U}\vec{P}_{ti}$, é rebatido sobre a trajetória, como mostra a figura 7.16.

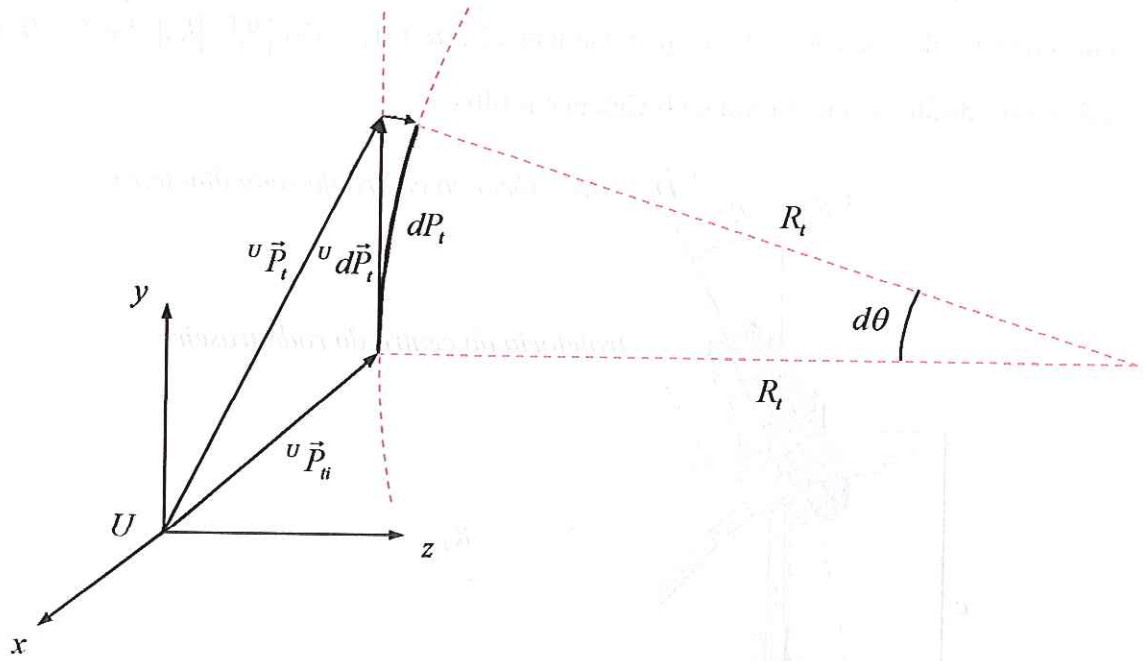


Figura 7.16 – Rebatimento de dP_t sobre a trajetória do centro da roda traseira

Com o valor de dP_t, pode-se determinar o incremento angular $d\theta$, como abaixo:

$$d\theta = \frac{dP_t}{R_t}$$

Da figura 7.16, pode-se estabelecer as seguintes relações geométricas:

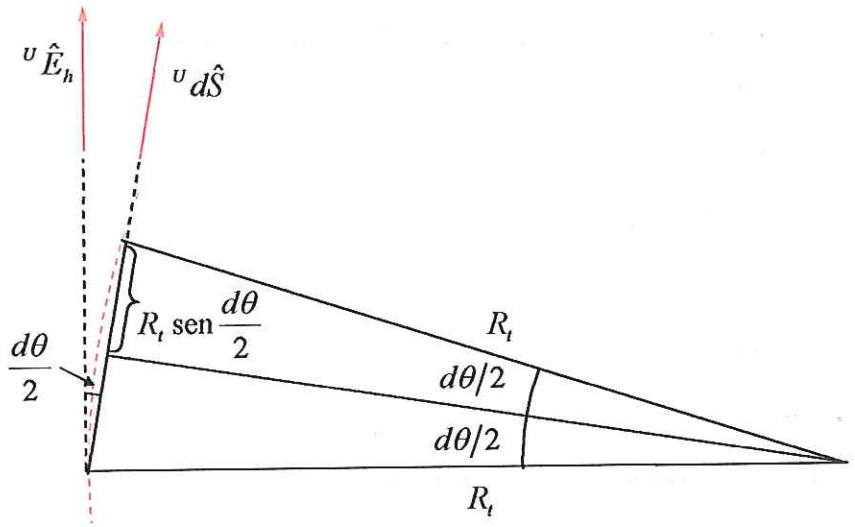


Figura 7.17 - Relações geométricas

O versor ${}^{U}d\hat{S}$ pode ser determinado pela rotação do versor ${}^{U}\hat{E}_h$ de $\frac{d\theta}{2}$ em torno de ${}^{U}\hat{E}_v$:

$$ {}^{U}d\hat{S} = R_{\frac{d\theta}{2},\hat{E}v} \cdot {}^{U}\vec{E}_h $$

A posição do centro da roda traseira na sua trajetória circular é então calculada pela expressão:

$$ {}^{U}\vec{P}_t = {}^{U}\vec{P}_{ti} + 2R_t \operatorname{sen}\left(\frac{d\theta}{2}\right) {}^{U}d\hat{S} $$

O versor ${}^{U}\hat{E}_h$ é atualizado rotacionando-se o mesmo de $d\theta$ em torno de ${}^{U}\hat{E}_v$ (veja figura 7.10):

$$ {}^{U}\vec{E}_h = R_{d\theta,\hat{E}v} \cdot {}^{U}\vec{E}_h $$

O versor ${}^{U}\hat{T}_t$ é atualizado por meio do produto vetorial de ${}^{U}\hat{E}_v$ com ${}^{U}\hat{E}_h$:

$$ {}^{U}\hat{T}_t = {}^{U}\vec{E}_v \times {}^{U}\vec{E}_h $$

Por fim, a posição do centro da roda dianteira pode ser calculada pela expressão:

$$ {}^{U}\vec{P}_d = {}^{U}\vec{P}_t + c\,{}^{U}\hat{E}_h $$

Reescrevendo o processo na forma algorítmica:

/ Cálculo dos raios das trajetórias dos centros das rodas */*

Se ($\theta \neq 0$) então

$R_t \leftarrow c / tg\,\theta$;

$R_d \leftarrow c / \text{sen}\,\theta$;

Fim Se

/ Cálculo do vetor deslocamento* $^{U}d\vec{P}_t$ **/*

$^{U}d\vec{P}_t \leftarrow {}^{U}\vec{P}_t - {}^{U}\vec{P}_{ti}$;

/ Cálculo do valor escalar do vetor deslocamento */*

$dP_t \leftarrow {}^{U}d\vec{P}_t \bullet {}^{U}\hat{E}_h$;

/ Cálculo do incremento angular correspondente ao deslocamento* dP_t **/*

$d\theta \leftarrow dP_t / R_t$;

/ Cálculo do versor* $^{U}d\hat{S}$ **/*

$^{U}d\hat{S} \leftarrow R_{\frac{d\theta}{2},\hat{E}v} * {}^{U}\vec{E}_h$;

/ Cálculo da posição do centro da roda traseira sobre sua trajetória circular */*

$${}^{U}\vec{P}_t \leftarrow {}^{U}\vec{P}_{ti} + 2 * R_t * \text{sen}\left(\frac{d\theta}{2}\right)^{U} * d\hat{S} ;$$

/ Atualização de* $^{U}\hat{E}_h$ **/*

$^{U}\hat{E}_h \leftarrow R_{d\theta,\hat{E}v} * {}^{U}\vec{E}_h$;

/ Atualização de* $^{U}\hat{T}_h$ **/*

$^{U}\hat{T}_h \leftarrow {}^{U}\vec{E}_v \times {}^{U}\vec{E}_h$;

/ Cálculo da posição do centro da roda dianteira */*

$^{U}\vec{P}_d \leftarrow {}^{U}\vec{P}_t + c\,{}^{U}\hat{E}_h$;

Os deslocamentos angulares dos centros da rodas traseira e dianteira são iguais. Assim, pode-se escrever a seguinte relação escalar:

$$\frac{V_t}{R_t} = \frac{V_d}{R_d}$$

Como $\frac{R_t}{R_d} = \cos\theta$, a velocidade escalar do centro da roda dianteira pode ser expressa por:

$$V_d = \frac{V_t}{\cos\theta}$$

Como não há escorregamento entre os pneus e o chão, pode-se estabelecer as seguintes relações:

$$\omega_t = \frac{V_t}{r} \quad \text{e} \quad \omega_d = \frac{V_d}{r}, \quad \text{onde:}$$

- ω_t é a velocidade angular escalar da roda traseira.
- ω_d é a velocidade angular escalar da roda dianteira.
- r é o raio da roda (as rodas são idênticas).

Pode-se agora estabelecer um algoritmo incremental para determinar os deslocamentos angulares de cada roda (em relação aos seus respectivos eixos):

$$dAng_t \leftarrow dAng_t + \omega_t * dt;$$

$$dAng_d \leftarrow dAng_d + \omega_d * dt;$$

Tais deslocamentos serão necessários nas transformações geométricas das rodas.

7.2.2.4 Colisões da bicicleta virtual

As colisões da bicicleta são efetuadas aplicando-se a técnica de emissão e interceptação de raios virtuais, a partir de pontos estratégicos relacionados à bicicleta.

7.2.2.4.1 Colisão com as paredes

Os pontos estratégicos (emissores) definidos para a checagem de colisão da bicicleta com as paredes são os centros das rodas dianteira e traseira, dos quais são emitidos os "raios interceptadores" (dois dianteiros e dois traseiros), como mostra a figura 7.18.

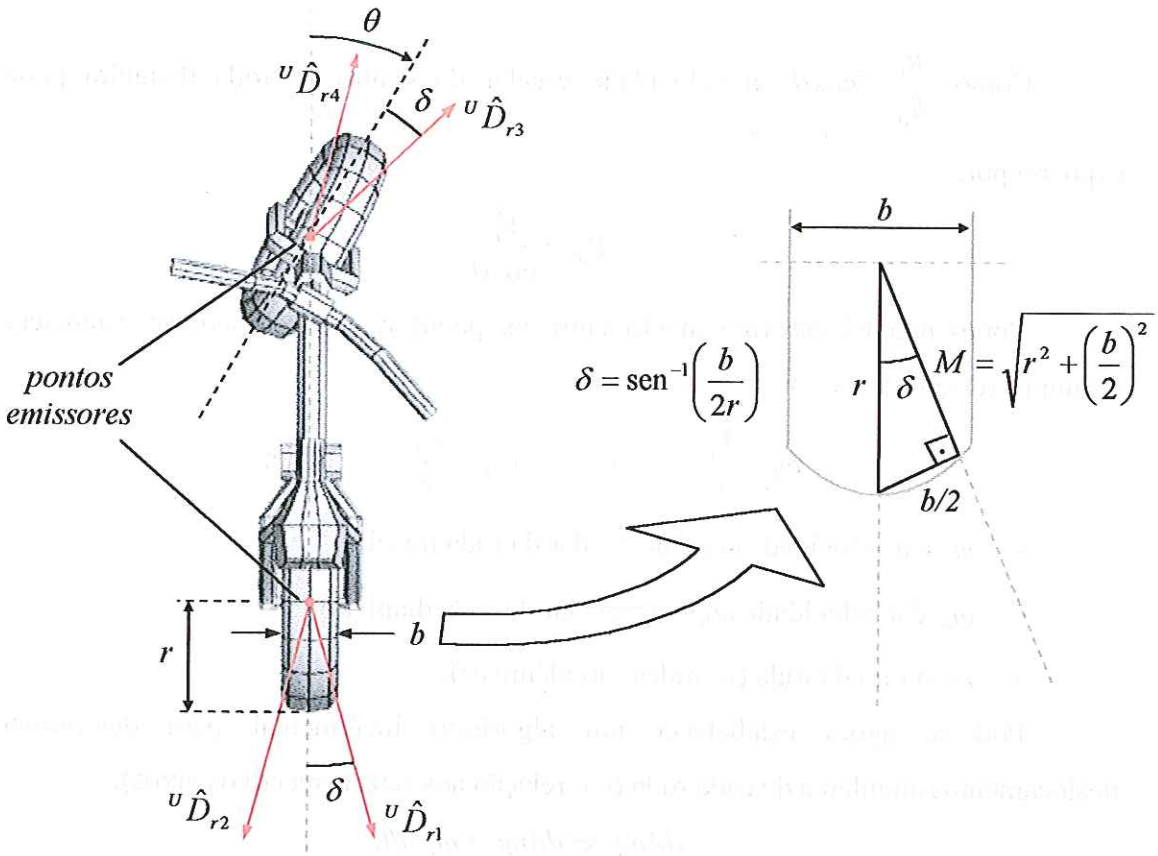


Figura 7.18 – Ilustração dos raios interceptadores emitidos a partir dos centros das rodas

O ângulo δ é uma função das dimensões geométricas da roda, permanecendo invariante no tempo. Para fins de performance, todas as variáveis cujos conteúdos permaneçam constantes durante o processamento do jogo devem ser calculadas antes do *loop* principal.

Os vetores direcionais dos raios apresentados na figura 7.18 são obtidos por meio da rotação de ${}^{U}\hat{E}_h$ em torno de ${}^{U}\hat{E}_v$, como mostrado a seguir:

Raio traseiro direito:

- Ponto emissor: Centro da roda traseira
- Vetor direcional: ${}^{U}\hat{D}_{r1} = -R_{-\delta,\hat{E}v}.{}^{U}\hat{E}_h$ (rotação de $-\delta$)

Raio traseiro esquerdo:

- Ponto emissor: Centro da roda traseira
- Vetor direcional: ${}^{U}\hat{D}_{r2} = -R_{\delta,\hat{E}v}.{}^{U}\hat{E}_h$ (rotação de δ)

Raio dianteiro direito

- Ponto emissor: Centro da roda dianteira
- Vetor direcional: ${}^{U}\hat{D}_{r3} = R_{\theta+\delta,\hat{E}v}.{}^{U}\hat{E}_{h}$ (rotação de $\theta+\delta$)

Raio dianteiro esquerdo

- Ponto emissor: Centro da roda dianteira
- Vetor direcional: ${}^{U}\hat{D}_{r4} = R_{\theta-\delta,\hat{E}v}.{}^{U}\hat{E}_{h}$ (rotação de $\theta-\delta$)

A função *D3DXIntersect* do *Microsoft Direct3D* foi implementada com base no método de colisão descrito no capítulo anterior. Fornecidos um raio (especificado por um ponto emissor e um vetor direcional) e uma malha (especificada pelas coordenadas dos seus vértices e índices), esta função verifica quais faces da malha são interceptadas pelo raio e fornece a distância do ponto emissor ao ponto de interseção mais próximo.

Considerando-se a malha das paredes e os raios anteriores, o processo de checagem da colisão consiste em comparar a distância (*dist*) obtida pela aplicação da função *D3DXIntersect* com o valor *M*, descrito na figura 7.18. Se *dist* for menor do que *M*, então o pneu penetrará na parede ocorrendo, portanto, a colisão.

Uma vez detectada a colisão de um dos pneus, deve-se deslocar a bicicleta de modo este não penetre na parede.

Considere a figura 7.19:

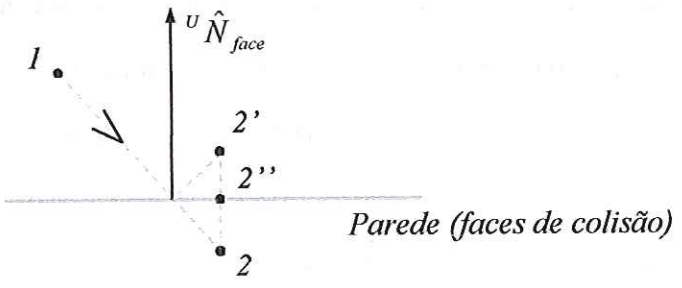


Figura 7.19 – Ilustração da colisão, imediatamente antes e após o choque

Na figura, *1* e *2* representam a posição da extremidade de contato do pneu em iterações sucessivas, imediatamente antes e imediatamente após a sua penetração na parede.

Se tratarmos o pneu como um corpo rígido e considerarmos que choque é perfeitamente elástico, sua extremidade de contato deverá ocupar a posição *2'* ao invés da

2. No entanto, se considerarmos o choque como sendo inelástico e desprezarmos o atrito entre a parede e o pneu, sua extremidade de contato deverá ocupar a posição *2''*.

Considere, agora, a colisão do pneu dianteiro com a parede, baseado nesta última hipótese, como mostra figura 7.20:

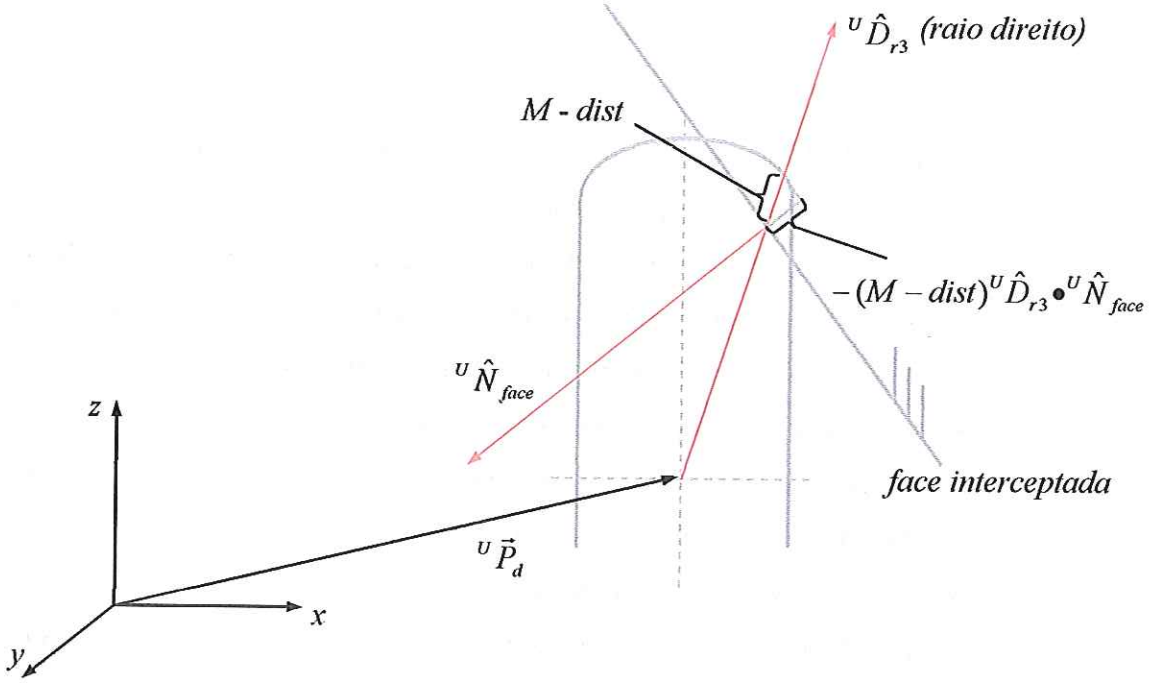


Figura 7.20 - Colisão do pneu dianteiro com a parede

Para evitar a penetração do pneu, o deslocamento do ponto de contato na direção normal à face da parede pode ser obtido pela seguinte atualização de ${}^{U}\vec{P}_{d}$:

$$ {}^{U}\vec{P}_{d} \leftarrow {}^{U}\vec{P}_{d} + [(dist - M)^{U}\hat{D}_{r3} \bullet {}^{U}\hat{N}_{face}]^{U}\hat{N}_{face} $$

Para manter as dimensões geométricas da bicicleta, a posição do centro da roda traseira deve então ser atualizada para:

$$ {}^{U}\vec{P}_{t} \leftarrow {}^{U}\vec{P}_{d} - c^{U}\hat{E}_{h} $$

De forma análoga, este procedimento pode ser aplicado aos outros três raios interceptadores, como abaixo:

Raio dianteiro esquerdo:

- atualização de ${}^{U}\vec{P}_{d}$: $ {}^{U}\vec{P}_{d} \leftarrow {}^{U}\vec{P}_{d} + \left[(dist - M)^{U}\hat{D}_{r4} \bullet {}^{U}\hat{N}_{face}\right]^{U}\hat{N}_{face} $

- atualização de ${}^{U}\vec{P}_{t}$: $ {}^{U}\vec{P}_{t} \leftarrow {}^{U}\vec{P}_{d} - c^{U}\hat{E}_{h} $

Raio traseiro direito:

- atualização de ${}^{U}\vec{P}_t$: ${}^{U}\vec{P}_t \leftarrow {}^{U}\vec{P}_t + \left[(dist - M)\,{}^{U}\hat{D}_{r1} \bullet {}^{U}\hat{N}_{face}\right]{}^{U}\hat{N}_{face}$

- atualização de ${}^{U}\vec{P}_d$: ${}^{U}\vec{P}_d \leftarrow {}^{U}\vec{P}_t + c\,{}^{U}\hat{E}_h$

Raio traseiro esquerdo:

- atualização de ${}^{U}\vec{P}_t$: ${}^{U}\vec{P}_t \leftarrow {}^{U}\vec{P}_t + \left[(dist - M)\,{}^{U}\hat{D}_{r2} \bullet {}^{U}\hat{N}_{face}\right]{}^{U}\hat{N}_{face}$

- atualização de ${}^{U}\vec{P}_d$: ${}^{U}\vec{P}_d \leftarrow {}^{U}\vec{P}_t + c\,{}^{U}\hat{E}_h$

7.2.2.4.2 Colisão com o terreno

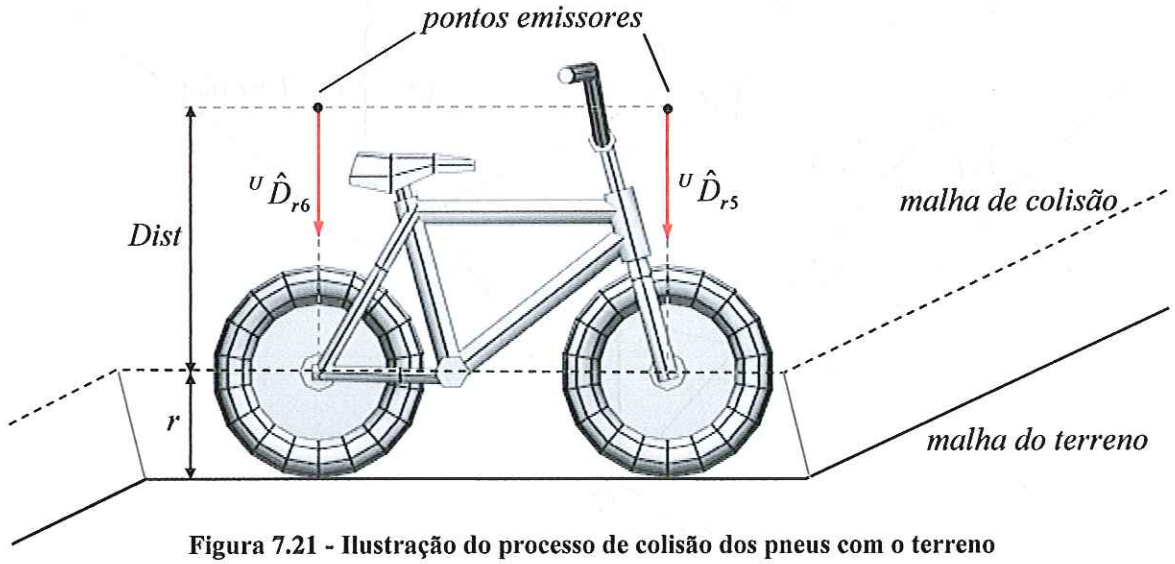


Figura 7.21 - Ilustração do processo de colisão dos pneus com o terreno

A checagem de colisão dos pneus da bicicleta com as faces do terreno é feita por meio de dois raios interceptadores (${}^{U}\hat{D}_{r5}$ e ${}^{U}\hat{D}_{r6}$), lançados verticalmente, para baixo, a partir de pontos emissores posiconados a uma distância conhecida (*Dist*) acima dos centros das rodas, como mostrado na figura 7.21.

A princípio, poderia-se pensar em efetuar a checagem da colisão em relação à própria malha do terreno, determinado-se a distância dos pontos emissores à mesma (usando a função *D3DXIntersect*) e comparando-se o valor obtido com *Dist* + *r*. No entanto, seria problemático simular com precisão o contato entre os pneus e o terreno

durante suas mudanças de inclinação. Uma boa solução consiste em criar uma nova malha, paralela ao terreno, porém deslocada r deste na direção e sentido dos versores normais às suas faces. Esta nova malha, mostrada na figura 7.21 com o nome de malha de colisão, não sofre o processo de *rendering* e, portanto, não é apresentada na tela. Sua finalidade apenas se restringe à checagem da colisão, cuja comparação de distância, neste caso, é feita em relação a *Dist*.

Considere a figura 7.22:

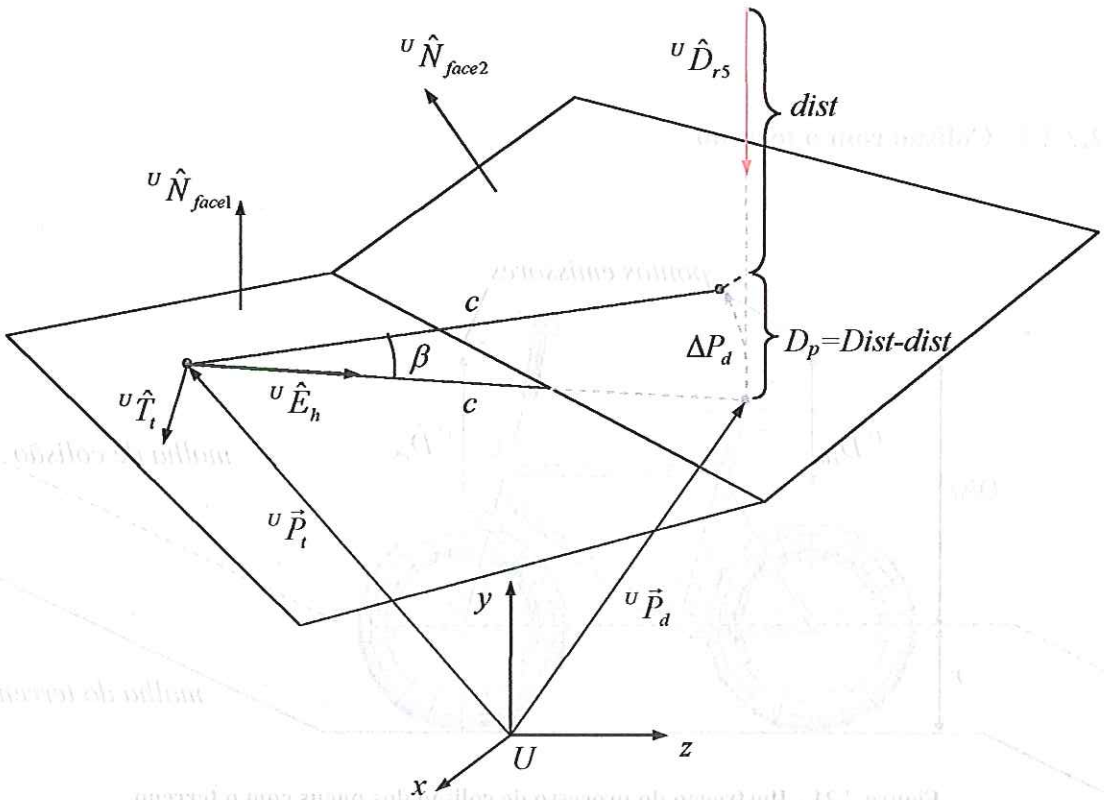


Figura 7.22 - Penetração do centro da roda dianteira na malha de colisão

Na figura 7.22 é ilustrada a iteração exata na qual o centro da roda dianteira penetra na malha de colisão, no momento em que a bicicleta avança sobre uma inclinação diferente do terreno.

O raciocínio consiste em rotacionar o seguimento c (distância entre os centros das rodas da bicicleta) em torno do versor transversal $^U\hat{T}_t$ de modo que o centro da roda dianteira seja posicionado sobre a face inclinada. A nova posição de $^U\vec{P}_d$ pode ser obtida resolvendo-se o sistema abaixo, composto pelas equações do plano que contém a face

penetrada, da esfera de raio c centrada no centro da roda traseira e do plano que passa por este ponto e é normal a ${}^{U}\hat{T}_t$:

$$\begin{cases} n_{fx}x + n_{fy}y + n_{fz}z - (n_{fx}v_x + n_{fy}v_y + n_{fz}v_z) = 0 \\ (x - p_{tx})^2 + (y - p_{ty})^2 + (z - p_{tz})^2 = c^2 \\ t_{tx}x + t_{ty}y + t_{tz}z - (t_{tx}p_{tx} + t_{ty}p_{ty} + t_{tz}p_{tz}) = 0 \end{cases}$$

onde: n_{fx}, n_{fy} e n_{fz} são as coordenadas do versor ${}^{U}\hat{N}_{face2}$

v_{fx}, v_{fy} e v_{fz} são as coordenadas de um vértice da face penetrada pelo raio

t_{tx}, t_{ty} e t_{tz} são as coordenadas do versor ${}^{U}\hat{T}_t$

p_{tx}, p_{ty} e p_{tz} são as coordenadas do centro da roda traseira

Embora este sistema apresente soluções muito precisas, seus cálculos são demasiadamente extensos para que sejam feitos a cada iteração, comprometendo a performance do jogo. Outro inconveniente é o fornecimento de duas soluções, fazendo com que seja necessário impor condicionais para se isolar a solução desejada.

Uma solução alternativa que oferece bons resultados consiste em determinar a nova posição do centro da roda dianteira de forma aproximada. Como as iterações são processadas muito rapidamente, a penetração D_p calculada em cada iteração é muito pequena quando comparada a dimensão c. Desta forma, pode-se considerar que o deslocamento escalar ΔP_d mostrado na figura 7.22 seja aproximadamente igual ao valor de penetração D_p. Assim, pode-se estabelecer a seguinte relação geométrica:

$$\beta \cong \frac{D_p}{c} = \frac{Dist - dist}{c}$$

onde *dist* é a distância fornecida pela função *D3DXIntersect*.

Em seguida, de acordo com a regra da mão esquerda, o versor ${}^{U}\hat{E}_h$ é rotacionado de $-\beta$ em torno do versor ${}^{U}\hat{T}_t$:

$${}^{U}\hat{E}_h = R_{-\beta,\hat{T}_t}\,{}^{U}\hat{E}_h$$

A nova posição do centro da roda dianteira pode agora ser expressa por:

$${}^{U}\vec{P}_d = {}^{U}\vec{P}_t + c\,{}^{U}\hat{E}_h$$

O mesmo procedimento deve ser aplicado na checagem de penetração do centro da roda traseira nas faces da malha de colisão. Neste caso, porém, devido à aplicação da regra da mão esquerda, o sentido da rotação de ${}^{U}\hat{E}_h$ em torno de ${}^{U}\hat{T}_t$ é positivo. Assim:

$${}^{U}\hat{E}_h = R_{\beta,\hat{T}_t}\,{}^{U}\hat{E}_h$$

A nova posição do centro da roda traseira pode agora ser expressa por:

$${}^{U}\vec{P}_t = {}^{U}\vec{P}_d - c\,{}^{U}\hat{E}_h$$

Após a execução dos cálculos de colisão, o versor ${}^{U}\hat{E}_v$ deve ser atualizado pelo produto vetorial abaixo:

$${}^{U}\hat{E}_v = {}^{U}\hat{E}_h \times {}^{U}\hat{T}_t$$

7.2.2.5 Inclinação aproximada da bicicleta

O cálculo cinemático descrito anteriormente não considera a inclinação da bicicleta em trajetórias curvilíneas. Desta forma, a bicicleta é manipulada de modo que seu quadro sempre permaneça em um plano vertical. Essa abordagem facilita a implementação computacional dos cálculos, porém proporciona uma certa artificialidade ao movimento. Assim, apenas para gerar maior naturalidade ao comportamento dinâmico da bicicleta, foram implementados cálculos que promovem, nas curvas, a sua inclinação de forma aproximada. Neste sentido, os versores ${}^{U}\hat{E}_v$, ${}^{U}\hat{E}_h$ e ${}^{U}\hat{T}_t$ não sofrem transformações e a inclinação da bicicleta é efetuada apenas para fins de visualização.

Nos cálculos envolvidos no processo de inclinação da bicicleta, foram assumidas as seguintes hipóteses:

- A reta que contém os pontos de contato entre os pneus e o terreno é sempre paralela à reta que contém os centros das rodas.
- O centro de massa (*c.m.*) da bicicleta está localizado no meio do seguimento que une os centros das rodas, como mostra a figura a seguir:

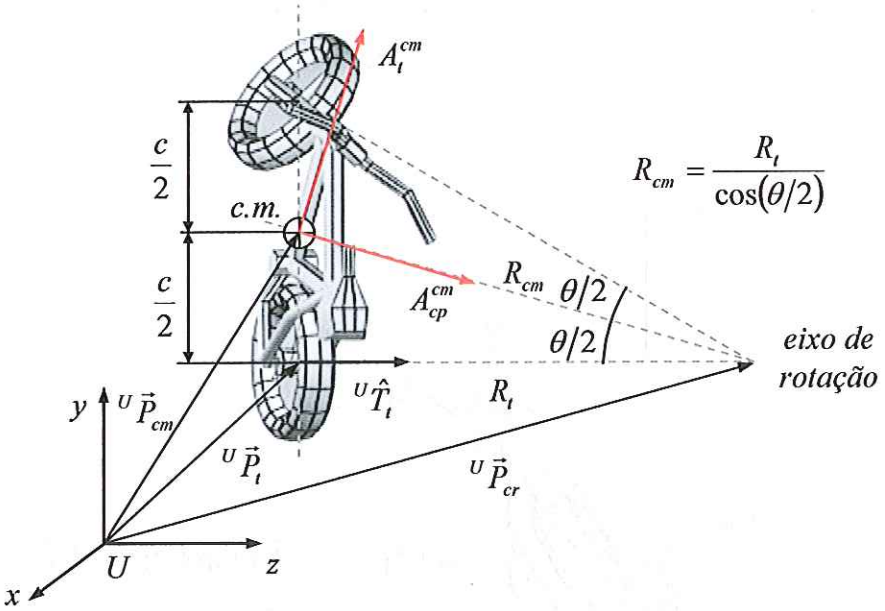


Figura 7.23 - Inclinação da bicicleta em trajetórias curvilíneas

Com base nos dados da figura, obtém-se as seguintes expressões:

- Posição do centro de massa:

$$^U\hat{P}_{cm} = {}^U\hat{P}_t + \frac{c}{2}{}^U\hat{E}_h$$

- Velocidade escalar do centro de massa:

$$V_{cm} = \frac{R_{cm}}{R_t}V_t$$

- Versor direcional da aceleração resultante centrípeta da bicicleta:

$$^U\hat{a}_{cp} = R_{\left(\frac{\theta}{2}+\frac{\pi}{2}\right),\hat{E}_v} \cdot {}^U\hat{E}_h$$

- Valor escalar da aceleração resultante centrípeta da bicicleta:

$$A_{cp}^{cm} = \frac{V_{cm}^2}{R_{cm}}$$

- Versor direcional da aceleração resultante tangencial da bicicleta:

$$^U\hat{a}_t = R_{\frac{\theta}{2},\hat{E}_v} \cdot {}^U\hat{E}_h$$

- Valor escalar da aceleração resultante tangencial da bicicleta:

$$A_t^{cm} = \frac{A_{fr} + A_{tr} + A_{res} + A_g}{\cos(\theta/2)}$$

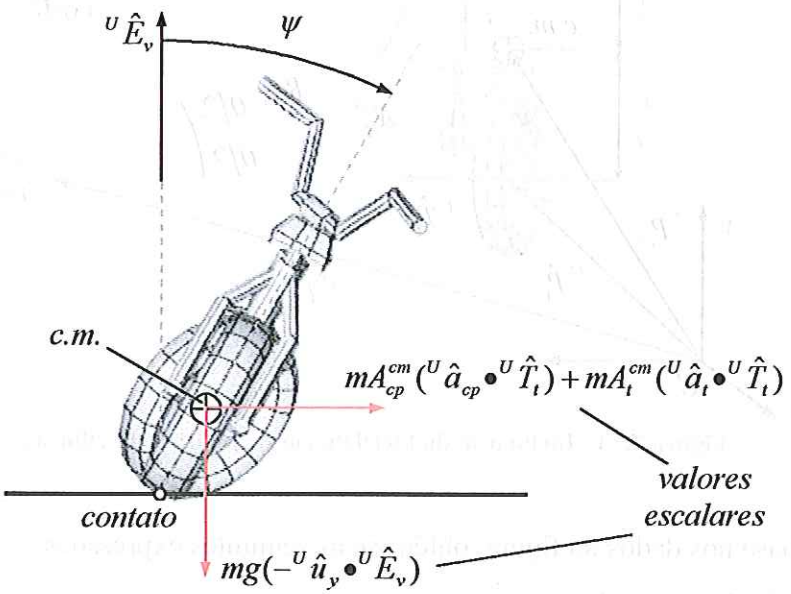


Figura 7.24 - Ilustração das intensidades das forças atuantes no *c.m.*

O ângulo de inclinação da bicicleta, ψ, pode ser determinado efetuando-se um equilíbrio de momentos em relação ao eixo que contém os pontos de contato entre os pneus e o terreno. Com exceção da força peso e das resultantes centrípeta e tangencial da bicicleta, todas as outras forças não geram momento em relação ao eixo especificado. Note que, dependendo da inclinação do terreno, as direções das forças mencionadas podem não coincidir com as dos versores ${}^{U}\hat{T}_t$ e ${}^{U}\hat{E}_v$. Desta forma, é preciso considerar apenas as projeções destas nas direções destes versores. A rigor, em movimentos curvos, a bicicleta é inclinada com o intuito de se produzir um braço de alavanca que, em conjunto com a força gravitacional, gera um torque capaz de equilibrar o momento produzido pela ação centrífuga.

$$\sum M_{contato} = 0$$

Assim:

$$mg(-{}^{U}\hat{u}_y) \bullet {}^{U}\hat{E}_v (r \operatorname{sen}\psi) - (mA_{cp}^{cm\,U}\hat{a}_{cp} \bullet {}^{U}\hat{T}_t + mA_t^{cm\,U}\hat{a}_t \bullet {}^{U}\hat{T}_t)(r\cos\psi) = 0$$

Logo, o ângulo de inclinação da bicicleta pode ser expresso como:

$$\psi = tg^{-1}\left(\frac{A_{cp}^{cm\,U}\hat{a}_{cp} \bullet {}^{U}\hat{T}_t + A_t^{cm\,U}\hat{a}_t \bullet {}^{U}\hat{T}_t}{g(-{}^{U}\hat{u}_y)\bullet {}^{U}\hat{E}_v}\right)$$

Obviamente, a posição do centro de massa da bicicleta não está localizada sobre o eixo que une os centros das rodas, mas acima deste. No entanto, a equação do momento mostra que o ângulo de inclinação da bicicleta independe da distância do seu *c.m.* ao eixo que contém os pontos de contato entre os pneus e o terreno. Desta forma, o procedimento adotado simplifica o desenvolvimento dos cálculos e fornece bons resultados.

7.2.2.6 Transformações geométricas e *rendering*

Após o tratamento cinemático da bicicleta, é feita a sua apresentação na tela. Nesta etapa, são aplicadas as transformações geométricas para o posicionamento adequado dos seus componentes.

Uma vez que o versor ${}^{U}\hat{E}_v$ pode não estar na direção vertical, precisou-se definir alguns novos ângulos para que as transformações fossem aplicadas corretamente.

Considere a figura 7.25:

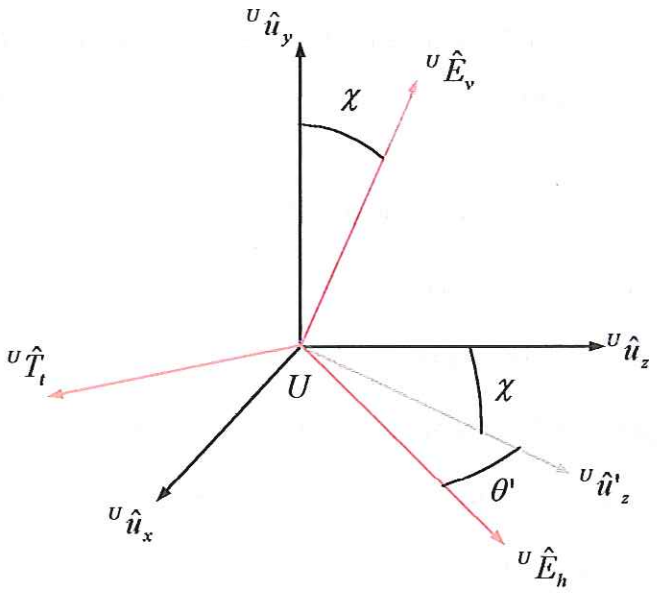


Figura 7.25 – Definição de novos ângulos para facilitar a aplicação das transformações

O ângulo χ, definido pelos versores ${}^{U}\hat{E}_{v}$ e ${}^{U}\hat{u}_{y}$ pode, a cada iteração, ser determinado pela expressão:

$$\chi = \cos^{-1}\left({}^{U}\hat{E}_{v} \bullet {}^{U}\hat{u}_{y}\right)$$

O versor ${}^{U}\hat{u}'_{z}$ é obtido pela rotação de ${}^{U}\hat{u}_{z}$ em torno de ${}^{U}\hat{T}_{t}$, segundo a transformação abaixo:

$${}^{U}\hat{u}'_{z} = R_{\chi,\hat{T}_{t}} \cdot {}^{U}\hat{u}_{z}$$

O ângulo θ' pode então ser definido a partir do produto escalar dos versores ${}^{U}\hat{u}'_{z}$ e ${}^{U}\hat{E}_{h}$, como mostrado abaixo:

$$\theta' = \cos^{-1}\left({}^{U}\hat{E}_{h} \bullet {}^{U}\hat{u}'_{z}\right)$$

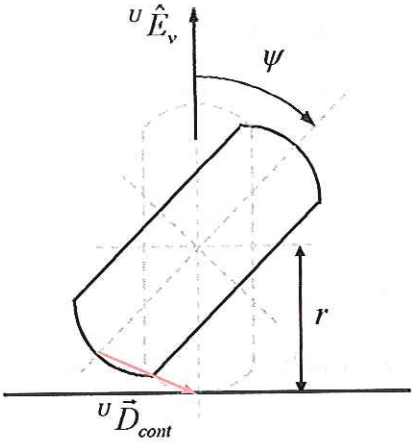
Note que θ' representa o deslocamento angular da bicicleta em torno do eixo de rotação, medido no plano perpendicular a ${}^{U}\hat{E}_{v}$.

As transformações geométricas dos componentes da bicicleta são efetuadas, a cada iteração, em relação às suas posições iniciais. Como descrito anteriormente, este posicionamento é feito dentro do *software* de modelagem, antes da exportação das malhas, conforme foi mostrado na figura 7.9.b.

Assim, o processo de transformação geométrica da roda dianteira, da sua posição inicial para uma posição genérica dentro do cenário virtual, compreende a seguinte seqüência de etapas:

- Rotação de $\theta + \theta'$ em torno de ${}^{U}\hat{E}_{v}$.
- Rotação de χ em torno de ${}^{U}\hat{T}_{t}$.
- Rotação de $dAng_{d}$ em torno de ${}^{U}\hat{D}_{t}$.
- Rotação de 15° em torno de ${}^{U}\hat{T}_{t}$ (inclinação do conjunto guidom + garfo).
- Rotação de ψ em torno de ${}^{U}\hat{E}_{h}$ (inclinação aproximada da bicicleta).
- Translação de ${}^{U}\vec{D}_{cont}$.*
- Translação de ${}^{U}\vec{P}_{d}$ (translação da roda da origem para a posição desejada).

(*) Uma vez rotacionada a roda em torno de ${}^{U}\hat{E}_h$, torna-se necessário transladá-la de modo que seu ponto de contato permaneça sobre o terreno. A figura 7.26 ilustra o processo:



$$^{U}\vec{D}_{cont} = rR_{\psi,\hat{E}h}.{}^{U}\hat{E}_v - r{}^{U}\hat{E}_v$$

Figura 7.26 - Translação para colocar novamente os pneus em contato com o terreno

Desta forma, cada vértice da roda dianteira é transformado, a cada iteração, pela composição matricial abaixo:

$$^{U}\vec{P}_{transformado} = T_{Pd}.T_{Dcont}.R_{\psi,\hat{E}h}.R_{\delta,\hat{T}t}.R_{dAngd,\hat{D}t}.R_{\chi,\hat{T}t}.R_{\theta+\theta',\hat{E}v}.{}^{U}\vec{P}_{inicial}$$

Os demais componentes da bicicleta seguem processos análogos, onde são consideradas as transformações geométricas intrínsecas ao posicionamento de cada um.

As etapas envolvidas no processo de transformação do conjunto guidom + garfo são:

- Rotação de $\theta + \theta'$ em torno de ${}^{U}\hat{E}_v$.
- Rotação de χ em torno de ${}^{U}\hat{T}_t$.
- Rotação de 15° em torno de ${}^{U}\hat{T}_t$ (inclinação do conjunto guidom + garfo).
- Rotação de ψ em torno de ${}^{U}\hat{E}_h$ (inclinação aproximada da bicicleta).
- Translação de ${}^{U}\vec{D}_{cont}$.
- Translação de ${}^{U}\vec{P}_d$ (translação da roda da origem para a posição desejada).

Assim, cada vértice do conjunto guidom + garfo deve ser transformado, a cada iteração, pela seguinte composição matricial:

$$^{U}\vec{P}_{transformado} = T_{Pd}.T_{Dcont}.R_{\psi,\hat{E}h}.R_{\delta,\hat{T}t}.R_{\chi,\hat{T}t}.R_{\theta+\theta',\hat{E}v}.^{U}\vec{P}_{inicial}$$

Como descrito anteriormente, as rodas traseira e dianteira, por serem idênticas, são transformadas e apresentadas a partir de uma mesma malha. Sendo assim, as etapas envolvidas no processo de transformação da roda traseira são:

- Rotação de θ' em torno de $^{U}\hat{E}_{v}$.
- Rotação de χ em torno de $^{U}\hat{T}_{t}$.
- Rotação de $dAng_{t}$ em torno de $^{U}\hat{T}_{t}$.
- Rotação de ψ em torno de $^{U}\hat{E}_{h}$ (inclinação aproximada da bicicleta).
- Translação de $^{U}\vec{D}_{cont}$.
- Translação de $^{U}\vec{P}_{t}$ (translação da roda da origem para a posição desejada).

A composição matricial que transforma, a cada iteração, os vértices da roda traseira, é representada por:

$$^{U}\vec{P}_{transformado} = T_{Pt}.T_{Dcont}.R_{\psi,\hat{E}h}.R_{dAngt,\hat{T}t}.R_{\chi,\hat{T}t}.R_{\theta',\hat{E}v}.^{U}\vec{P}_{inicial}$$

As etapas envolvidas no processo de transformação do quadro são:

- Rotação de θ' em torno de $^{U}\hat{E}_{v}$.
- Rotação de χ em torno de $^{U}\hat{T}_{t}$.
- Rotação de ψ em torno de $^{U}\hat{E}_{h}$ (inclinação aproximada da bicicleta).
- Translação de $^{U}\vec{D}_{cont}$.
- Translação de $^{U}\vec{P}_{d}$ (translação da roda da origem para a posição desejada).

Cada vértice do quadro é transformado, a cada iteração, pela seguinte composição matricial:

$$^{U}\vec{P}_{transformado} = T_{Pd}.T_{Dcont}.R_{\psi,\hat{E}h}.R_{\chi,\hat{T}t}.R_{\theta',\hat{E}v}.^{U}\vec{P}_{inicial}$$

7.3 Modelos dinâmicos deformáveis

Os modelos dinâmicos deformáveis não são tratados como corpos rígidos, uma vez que não conservam, ao longo do tempo, as distâncias relativas entre os seus vértices. Neste projeto, somente o modelo da criança é classificado nesta categoria. A figura 7.27 ilustra a concepção, modelagem, mapeamento e posicionamento da criança sobre a bicicleta.

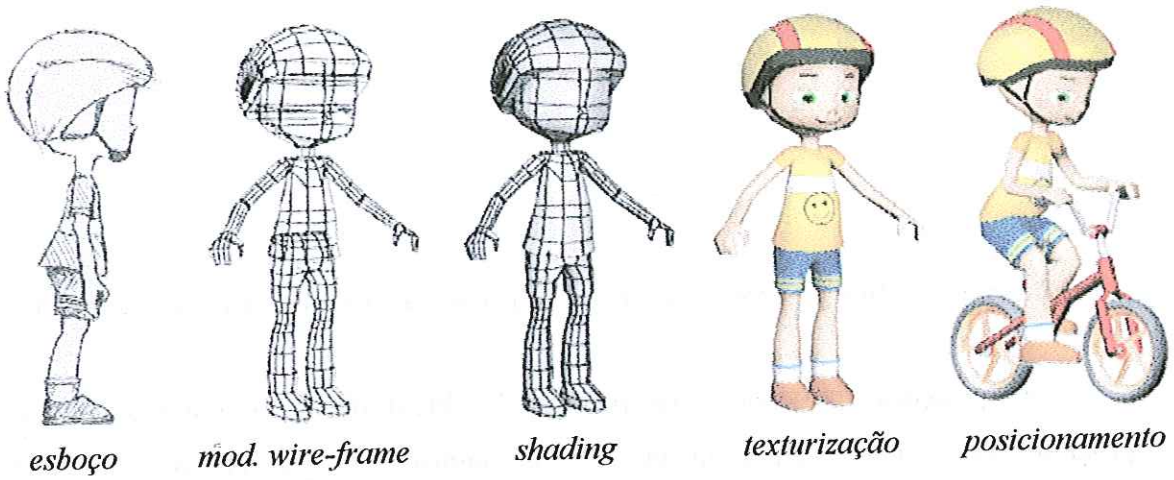


Figura 7.27 - Concepção, modelagem, mapeamento e posicionamento da criança sobre a bicicleta

Como a movimentação dos braços é independente, ao longo do tempo, da movimentação das pernas, a criança foi modelada a partir de duas malhas distintas: a malha superior e a inferior. A malha superior, correspondente ao conjunto formado pelo tronco, pelos braços e pela cabeça, acompanha o movimento do guidom, enquanto que a malha inferior, correspondente às pernas, é responsável pela simulação da pedalada. A movimentação destas malhas é obtida por meio de animações previamente concebidas, as quais são aplicadas utilizando-se os recursos de animação do *software* de modelagem. Assim, para se simular os braços movimentando o guidom, foram gravadas 100 posições diferentes para a malha superior, como ilustra a seguinte figura:

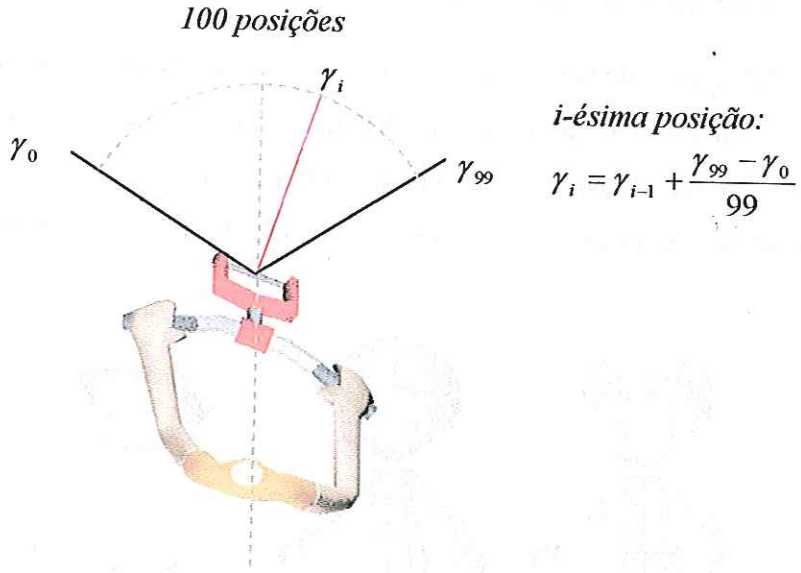


Figura 7.28 – Movimentação dos braços (100 *key-frames* diferentes para a malha superior)

As posições apresentadas na figura 7.28 foram obtidas e gravadas dentro do programa de modelagem, por um processo de animação seqüencial. Deve-se, contudo, efetuar tal processo de forma que os limites angulares γ_0 e γ_{99} coincidam respectivamente com os valores θ_{min} e θ_{max}, definidos no ambiente de programação. Uma vez finalizados os trabalhos de modelagem e animação, todas as 100 posições são exportadas, de forma que cada uma esteja associada a um índice de identificação. Deste modo, o *key-frame* (posição da malha na seqüência de animação) correspondente ao deslocamento angular θ pode ser obtido pela seguinte interpolação linear:

$$k_i = \frac{99}{\theta_{max} - \theta_{min}}(\theta - \theta_{min})$$

onde k_i é o índice referente à posição γ_i.

Obviamente, o índice k_i deve ser arredondado para um valor inteiro. Em linguagem C, a simples declaração deste índice como uma variável inteira automaticamente já proporciona o arredondamento do valor calculado.

A malha inferior é tratada de forma análoga. Porém, neste caso, foram gravadas 360 posições distintas para as pernas da criança, de forma que seja simulada uma pedalada completa. A figura a seguir ilustra o processo.

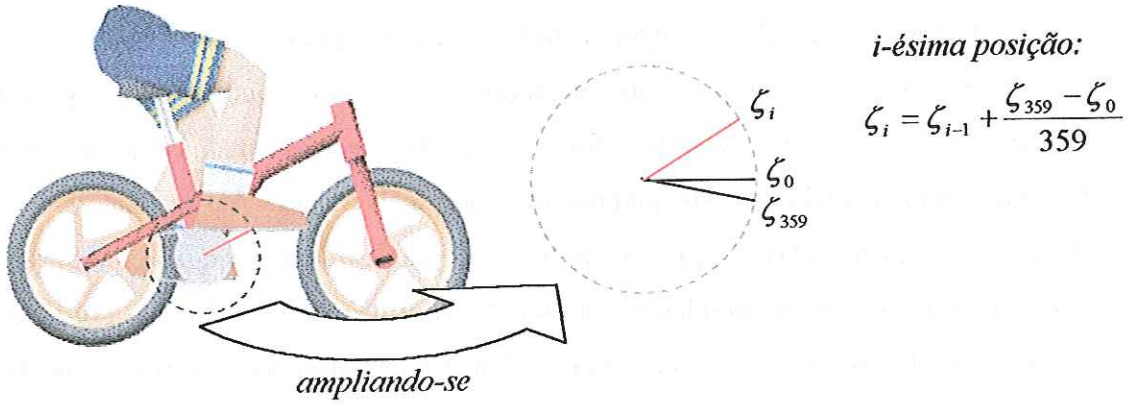


Figura 7.29 - Movimentação das pernas (360 *key-frames* diferentes para a malha inferior)

Perceba que o deslocamento angular entre duas posições sucessivas é de apenas 1°, o que fornece uma boa precisão à representação do movimento.

Finalizados os processos de modelagem e animação, a malha inferior é exportada, juntamente com suas 360 posições. Se o controle da pedalada for feito pelo teclado, então pode-se determinar um *key-frame* linearmente proporcional ao deslocamento angular da roda traseira:

$$k_i = N\frac{180}{\pi}dAng_t, \qquad \text{se } dAng_t \geq 0$$

$$k_i = 360 + N\frac{180}{\pi}dAng_t, \qquad \text{se } dAng_t < 0$$

onde k_i é o índice referente à posição ζ_i e N é a relação de transmissão entre a coroa e a catraca.

Para que as formulações acima forneçam resultados adequados, $dAng_t$ deve pertencer ao intervalo $(-2\pi, 2\pi)$. Assim, é necessário impor as seguintes condicionais:

Se $dAng_t \geq 2\pi$ então

$dAng_t \leftarrow dAng_t - 2\pi$;

Se $dAng_t \leq -2\pi$ então

$dAng_t \leftarrow dAng_t + 2\pi$;

Se, no entanto, o controle da pedalada for feito a partir dos dados de um *encoder*, então o *key-frame* correto é determinado efetuando-se um cálculo proporcional ao número

de pulsos enviados pelo mesmo. Este problema será abordado no próximo capítulo, quando será estudado o processo de aquisição de dados fornecidos por um *encoder*.

Note, contudo, que a movimentação geral da criança não é proveniente exclusivamente das animações que foram aplicadas às suas malhas, já que estas se deslocam pelo cenário virtual juntamente com a bicicleta. Assim, antes de serem exportadas, as malhas da criança são posicionadas relativamente ao quadro da bicicleta, de modo que possam ser submetidas às mesmas transformações geométricas. Portanto, além da aplicação das animações, as malhas da criança também devem ser transformadas pela seguinte composição matricial:

$$^{U}\vec{P}_{transformado} = T_{Pd}.T_{Dcont}.R_{\psi,\hat{E}h}.R_{\chi,\hat{T}t}.R_{\theta',\hat{E}v}.{}^{U}\vec{P}_{inicial}$$

8 ENTRADA E SAÍDA DE DADOS

No capítulo anterior foi considerado, para fins didáticos, que o controle da bicicleta virtual no cenário tridimensional era feito pelo teclado do computador. Considerando-se, contudo, os objetivos deste projeto, é fundamental que tal controle seja efetuado por meio de um dispositivo especial, cujas funcionalidades sejam cabíveis ao tratamento de reabilitação motora. Desta forma, desenvolveu-se um primeiro protótipo, ainda bastante elementar, objetivando-se estudar o grau de interatividade e controle que ele é capaz de fornecer. O protótipo, apresentado na figura 8.1, compreende uma bicicleta instrumentada, na qual estão acoplados sensores (para a captura dos movimentos do guidom e da coroa) e um motor (para propiciar o *force-feedback*).

Figura 8.1 - Bicicleta instrumentada com sensores e um motor

Obviamente, o dispositivo apresentado na figura 8.1 não se mostra adaptado à criança portadora deficiência. Ele não impede, no entanto, que testes iniciais sejam feitos com crianças normais (ou até mesmo com crianças deficientes, que poderiam se equilibrar sobre o protótipo com a ajuda de um terapeuta), a fim de que seja avaliado o grau de motivação das mesmas.

8.1 Captura do movimento do guidom

Para a captura do movimento do guidom acoplou-se um potenciômetro à bicicleta, de forma que a variação de sua resistência elétrica fosse proporcional ao deslocamento angular do guidom. A figura 8.2 ilustra o acoplamento.

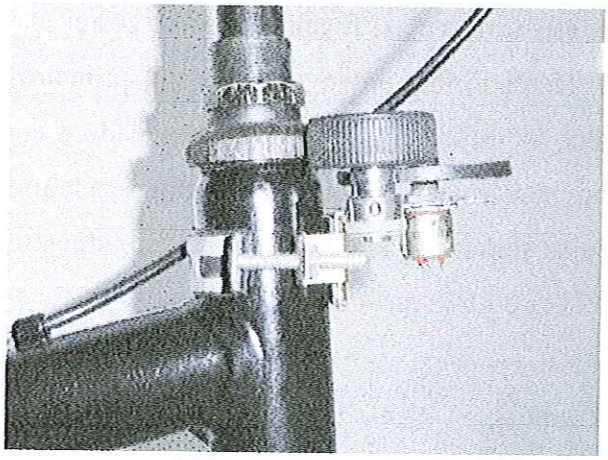
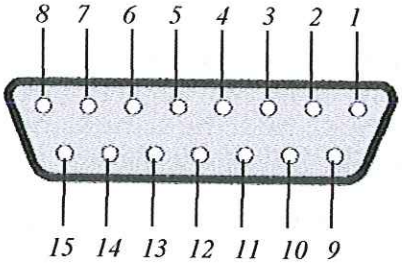


Figura 8.2 - Potenciômetro conectado ao guidom da bicicleta

A comunicação do computador com o potenciômetro é feita pela porta de *joystick*. Diferentemente das outras portas de comunicação, nas quais circuitos externos enviam ou recebem sinais digitais (pulsos de tensão), a porta de *joystick* funciona de modo que a resistência do potenciômetro constitua uma parte do circuito interno da placa.

Abaixo é apresentada a pinagem do conector macho desta porta. Por meio dela é possível conectar dois *joysticks*, cada um com dois potenciômetros e dois botões.



1- potencial de tensão de 5V
2- botão 1
3- potenciômetro 1
4- terra
5- terra
6- potenciômetro 2
7- botão 2
8- não usado
9- potencial de tensão de 5V
10- botão 3
11- potenciômetro 3
12- midi txd
13- potenciômetro 4
14- botão 4
15- midi rxd

Figura 8.3 - Pinagem do conector macho da porta de *joystick*

A porta de *joystick* permite a transferência de um pequeno fluxo de dados (8 bits) e geralmente está integrada à placa de som do computador, cuja comunicação com o

processador principal é feita através do barramento ISA (endereço 201h). Os 8 bits são distribuídos segundo a relação mostrada na tabela abaixo, onde o valor de cada bit define o estado (aberto ou fechado) de cada componente conectado.

Tabela 8.1 - Relação dos bits associados a cada componente

bit	*componente*
0	potenciômetro 1
1	potenciômetro 2
2	potenciômetro 3
3	potenciômetro 4
4	botão 1
5	botão 2
6	botão 3
7	botão 4

O processo de leitura da resistência do potenciômetro é efetuado com a utilização de um multivibrador monoestável (normalmente o CI 558). A partir de um pulso de entrada retangular, o multivibrador monoestável emite um pulso de saída com duração proporcional aos valores dos componentes capacitivos e resistivos aos quais ele está ligado. Neste contexto, a *CPU* envia um trem de pulsos ao CI 558, através do barramento ISA, e este, por sua vez, envia um outro trem de pulsos a um *buffer* de memória. A largura dos pulsos emitidos pelo multivibrador é proporcional ao valor da resistência do potenciômetro conectado à porta de *joystick*. A cada intervalo de tempo (previamente definido pelo programador), faz-se a leitura do número de pulsos armazenados no *buffer*, cujo valor é proporcional ao deslocamento angular da haste do potenciômetro. A figura a seguir esquematiza o processo de captura do movimento do guidom.

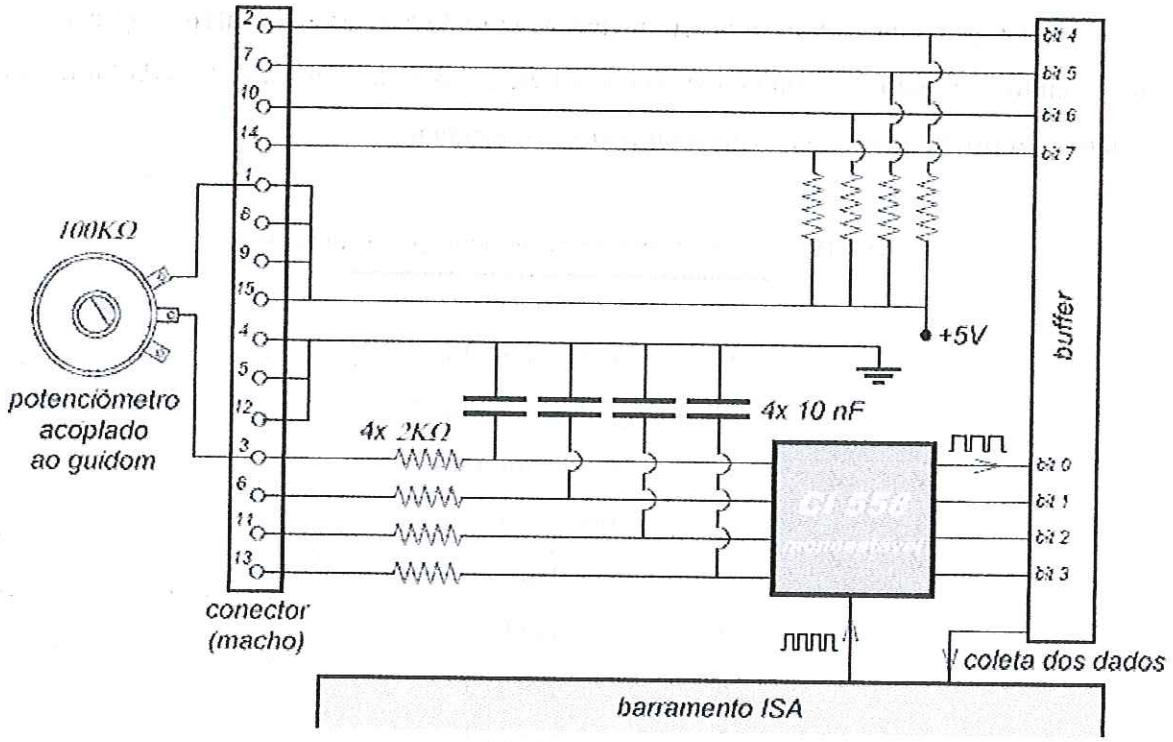


Figura 8.4 – Esquematização do processo de leitura da resistência do potenciômetro

A aquisição dos dados provenientes do circuito apresentado na figura 8.4 é feita com o uso da *API Microsoft DirectInput*, que é chamada por meio do ambiente de programação. Esta *API* proporciona rápido acesso aos dispositivos de entrada, pois possibilita a comunicação direta com os seus *drivers* (sem usar os recursos do sistema operacional).

Embora eficiente, o acesso ao dispositivo de entrada por meio do *DirectInput* não é simples. Sua configuração é complexa e seu entendimento requer um maior aprofundamento no assunto. Resumidamente, pode-se dizer que o uso do *DirectInput* envolve a implementação das seguintes etapas:

- Criar uma variável de interface para representar o dispositivo de entrada.
- Configurar o formato dos dados do dispositivo de entrada.
- Configurar os estados do dispositivo de entrada.
- Acessar o dispositivo de entrada.
- Receber os dados do dispositivo de entrada.

Cada etapa mencionada anteriormente envolve uma série de outras etapas que, por serem muito específicas e relacionadas à programação, não foram detalhadas aqui. Para maiores informações, consulte *Microsoft DirectX 9.0 SDK Documentation* (2002).

O processo de contagem dos pulsos emitidos pelo CI 558 já está implícito em uma função específica da *API DirectInput*, dispensando a necessidade de se implementar tal contagem.

Fisicamente, o guidom pode ser rotacionado de ângulos maiores do que θ_{max} e menores do que θ_{min}. No entanto, valores fora do intervalo $[\theta_{min}, \theta_{max}]$ ocasionariam problemas de sincronismo entre o movimento do guidom e as posições (*key-frames*) dos braços da criança virtual, pois sua seqüência animada não foi definida fora deste intervalo. Assim, por um processo de calibração, devem ser estipulados os valores de resistência correspondentes a θ_{min} e θ_{max}. Neste sentido, se para uma dada posição angular do guidom é retornado um valor R, basta rotacionar o guidom de forma que sejam retornados os valores R_{min} e R_{max} correspondentes respectivamente aos valores θ_{min} e θ_{max}. O gráfico abaixo ilustra a relação entre a posição angular do guidom e o valor retornado pelo *DirectInput*.

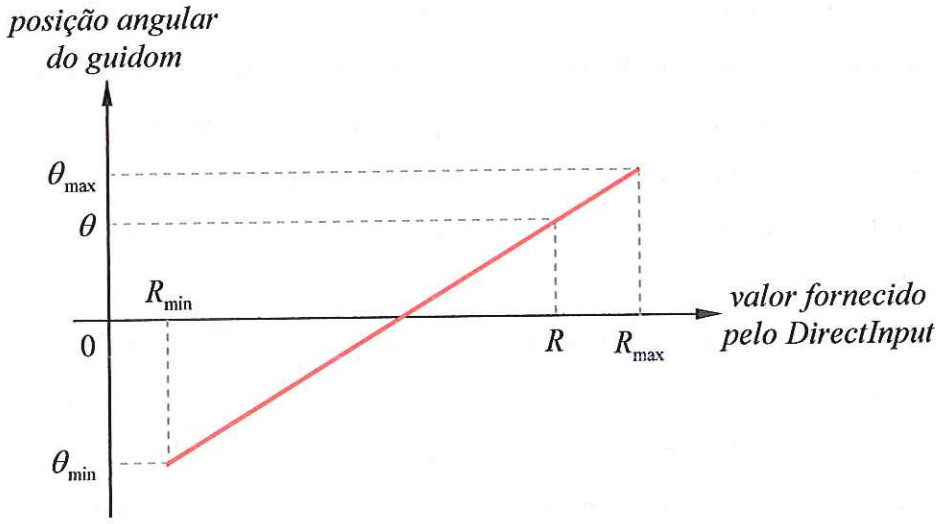


Figura 8.5 - Relação entre a posição angular do guidom e o valor retornado pelo *DirectInput*

onde:

$$\theta = \theta_{min} + \frac{\theta_{max} - \theta_{min}}{R_{max} - R_{min}}(R - R_{min})$$

8.2 Captura do movimento da coroa (pedalada)

A captura do movimento de pedalar da criança é feito a partir dos dados fornecidos por dois pequenos sensores ópticos acoplados à coroa da bicicleta. Um rolete de borracha transmite o movimento da coroa para um *encoder* posicionado entre os emissores de luz infravermelha (LEDs) e os sensores. A figura 8.6 ilustra este sistema:

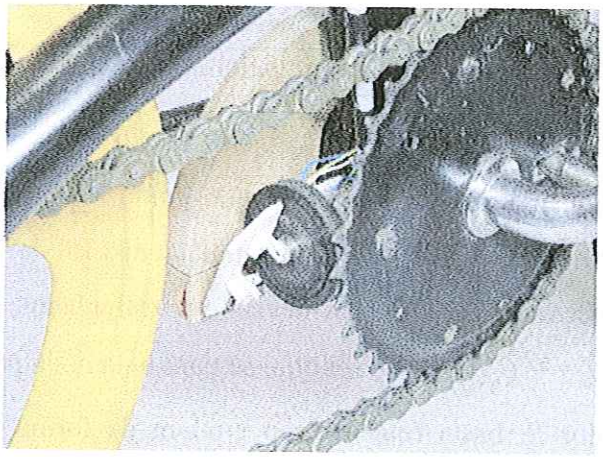


Figura 8.6 - Sistema de captura da rotação da coroa da bicicleta

O *encoder* é um pequeno disco contendo diversos furos, os quais possibilitam a passagem da luz infravermelha emitida pelos LEDs rumo aos sensores ópticos. Assim, quando o *encoder* é rotacionado, ora a luz emitida é bloqueada, ora não, de forma que cada sensor óptico receba uma seqüência de pulsos luminosos. Considerando-se que o sensor óptico fornece +5V quando iluminado e 0V quando não, cada sensor retorna um sinal de tensão na forma de uma onda, cujo número de pulsos é proporcional ao deslocamento angular do *encoder*.

Em princípio, um único conjunto emissor/sensor já seria suficiente para perceber o movimento do *encoder*, porém não seria possível identificar o seu sentido de rotação. Por este motivo, são usados dois conjuntos ópticos posicionados em relação ao *encoder* de forma que seus sinais de tensão estejam defasados entre si de 90°, como mostra a seguinte figura:

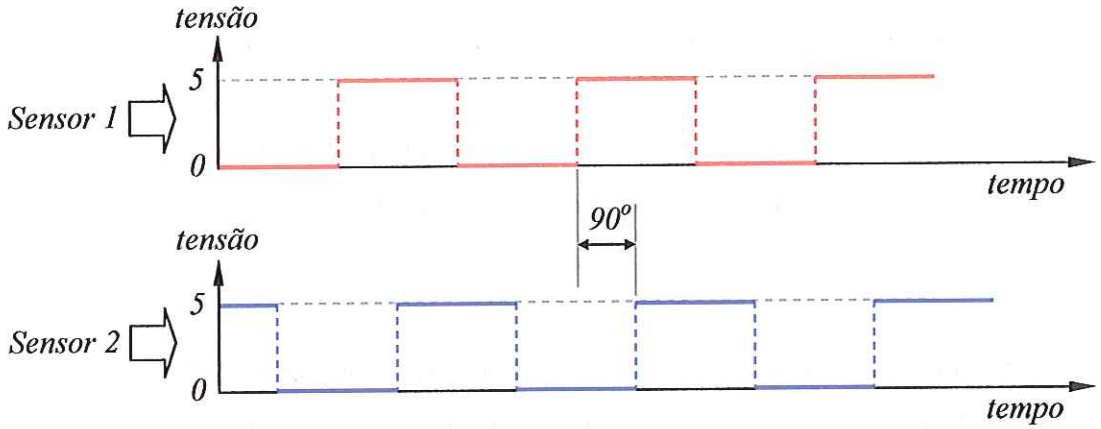


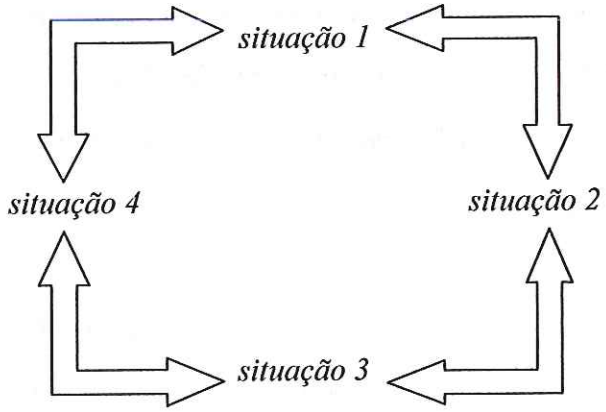
Figura 8.7 – Sensores ópticos emitem sinais de tensão defasados de 90° um do outro

A tabela abaixo mostra as quatro situações possíveis de ocorrer a cada instante de tempo:

Tabela 8.2 – Espaço amostral das situações possíveis

situações	*sensor 1 (tensão)*	*sensor 2 (tensão)*	*Descrição*
1	*5V*	*5V*	*ambos os sensores recebem luz*
2	*5V*	*0V*	*o sensor 1 recebe luz e o sensor 2 não recebe*
3	*0V*	*0V*	*ambos os sensores não recebem luz*
4	*0V*	*5V*	*o sensor 2 recebe luz e o sensor 1 não recebe*

onde os sinais de tensões enviados simultaneamente pelos dois sensores obedecem obrigatoriamente à seguinte seqüência cíclica:



Desta forma, transitando-se de uma situação para outra, é fácil estabelecer algumas condicionais lógicas para definir o sentido de rotação do *encoder*. Por exemplo:

Se situação (1 para 2) *Ou* (2 para 3) *Ou* (3 para 4) *Ou* (4 para 1)
Então encoder gira no sentido horário;
Senão encoder gira no sentido anti-horário;

Por meio de um pequeno microcontrolador, estas relações lógicas podem ser facilmente decodificadas e convertidas em informações compreensíveis pelo computador.

Considerando-se os objetivos deste projeto, não é necessário que a captura do movimento da coroa seja efetuada com extrema precisão. Neste sentido, foram usados os componentes de um mouse serial padrão na construção deste sistema, cujas vantagens consistem no baixo custo do equipamento e na conveniência em se poder utilizar um protocolo de comunicação já implementado.

Assim como na captura do movimento do guidom, a aquisição dos dados provenientes dos sensores ópticos também é feita com o uso da *API Microsoft DirectInput*.

Os potenciômetros usados em sistemas de captura de movimento normalmente fornecem posições absolutas, ou seja, onde a leitura é feita em relação a uma posição fixa. Já os *encoders* comuns, como os dos mouses, fornecem posições relativas. Neste caso, a leitura é feita em relação à última posição medida.

A cada leitura do estado do dispositivo de entrada, as funções do *DirectInput* retornam o número de pulsos na forma de valores inteiros positivos ou negativos, dependendo do sentido de rotação do *encoder*.

Por meio de testes empíricos pode-se determinar uma constante de transmissão (K_{trans}) que fornece o valor da velocidade angular da catraca ($\omega_{catraca}$) a partir do número de pulsos (n_{pulsos}) lidos pelo *DirectInput*. Assim, a velocidade angular da catraca pode ser obtida pela expressão:

$$\omega_{catraca} = K_{trans} \cdot n_{pulsos}$$

Se a velocidade angular da catraca for maior ou igual à velocidade angular da roda traseira ($\omega_{catraca} \geq \omega_t$), então haverá tração. No presente caso, onde o deslocamento da

bicicleta virtual é obtido pela captura do movimento físico da coroa, a tração imposta à bicicleta não mais é efetuada pela aplicação de uma aceleração no centro da roda traseira (como ocorrera no controle pelo teclado), mas pela atribuição do valor $\omega_{catraca} \cdot r$ à variável V_t (velocidade escalar do centro da roda traseira).

No controle da bicicleta pelo teclado, a seqüência animada, correspondente à movimentação das pernas da criança, foi definida a partir do deslocamento angular da roda traseira ($dAng_t$). No presente caso, porém, tal animação acompanha o deslocamento angular da catraca:

$$dAng_{catraca} \leftarrow dAng_{catraca} + \omega_{catraca} * dt;$$

Deste modo, as pernas da criança virtual acompanham o movimento das pernas da criança real, de forma que se a bicicleta estiver deslocando-se para frente e a criança real pedalar para trás (não impondo tração), a criança virtual também irá pedalar para trás.

8.3 Resistência controlada à ação de pedalar

Ao se movimentar pelo cenário virtual, a criança se depara com terrenos com diversas inclinações. A cada iteração, o programa envia sinais a um motor (acoplado à roda traseira da bicicleta) solicitando a sua atuação de acordo com o estado interativo da bicicleta com o cenário 3D.

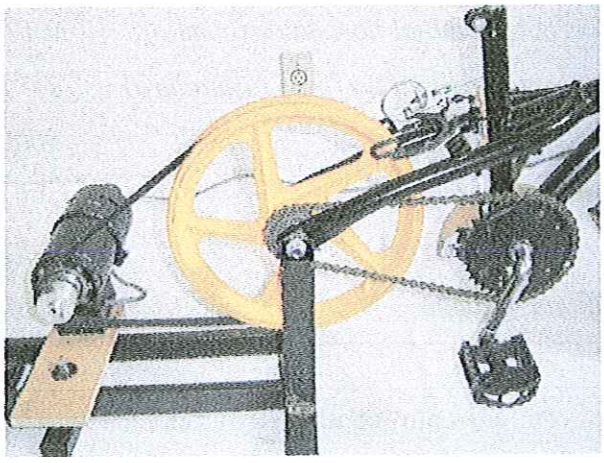


Figura 8.8 - Sistema de transmissão entre o motor e a coroa

A função principal do motor é atuar como um freio eletromagnético, oferecendo resistência ao movimento nas subidas. No entanto, mesmo que o motor não esteja sendo alimentado, os atritos presentes entre os componentes do sistema de transmissão já

oferecem, por si só, uma pequena resistência à ação de pedalar da criança. Considerando-se que a criança portadora de deficiência normalmente possua pouca coordenação motora, pode-se afirmar que essa pequena resistência é desejada, pois, neste caso, é importante que seus pés encontrem um mínimo de firmeza durante a pedalada. Por outro lado, tal resistência não contribui para o realismo da simulação nos movimentos descendentes, onde a resistência à pedalada deve ser mínima. Para solucionar esse inconveniente, o sentido da corrente elétrica na armadura é invertido nas descidas, de forma que o motor passe a fornecer torque a favor da pedalada.

Em crianças cujas pernas são muito fracas ou totalmente inoperantes, o motor pode ser usado para conduzir a pedalada, cabendo à criança controlar a direção do movimento da bicicleta. Neste caso, porém, para que o torque do motor seja transmitido à coroa, é necessário que a catraca esteja travada.

8.3.1 O motor

O motor usado no projeto opera em corrente contínua e apresenta as seguintes especificações técnicas:

Tabela 8.3 – Especificação técnicas do motor

Tensão nominal	*190 VCC*
Corrente nominal no estator (campo)	*0,12 A*
Corrente nominal no rotor (armadura)	*1,7 A*
Rotação nominal	*5000 rpm*
Potência nominal	*190 W*
Grau de proteção	*IP23*
Tipo de ligação	*Shunt*

Nos motores de corrente contínua, a velocidade angular da armadura varia linearmente com sua tensão de alimentação. Na ligação *shunt*, se não ocorrer a saturação do núcleo da armadura, ou seja, se a corrente que passa por esta não for muito elevada, a relação entre o torque e a velocidade angular do eixo do motor é linear. Caso contrário, a queda de tensão provocada pela indutância da armadura torna-se significativa e a relação

anterior deixa de ser linear. Desta forma, o torque gerado pelo motor é diretamente proporcional à sua tensão de alimentação e varia linearmente com esta caso não ocorra a saturação do núcleo da armadura.

8.3.2 A porta paralela

A porta paralela *SPP* (*Standard Parallel Port*) é uma interface de comunicação entre o computador e um periférico capaz de transferir dados a uma taxa de 150Kb/s. Ela se comunica com a *CPU* por meio de um *BUS* de dados de oito bits. O computador nomeia sua porta paralela padrão como *LPT1* e seu endereço hexadecimal para o envio de um *byte* de dados é 378h. Seu conector, conhecido como DB25, possui 25 pinos, dos quais somente oito são usados neste projeto. A figura 8.9 ilustra a pinagem deste conector.

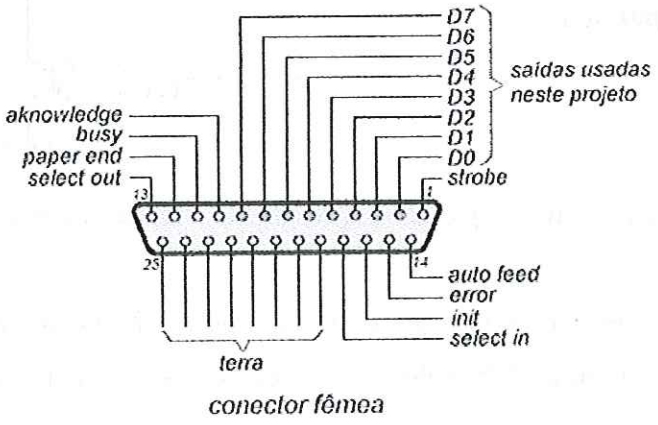


Figura 8.9 - Pinagem do conector fêmea da porta paralela

No DB25, se um pino apresentar tensão dentro do intervalo [0V , 0.4V], o valor lógico 0 é atribuído ao mesmo. Se, no entanto, sua tensão estiver dentro do intervalo [3.1V , 5V], seu valor lógico será de 1.

8.3.3 Controle do motor

Dos oito bits emitidos pelo computador, um é usado para inverter o sentido da corrente na armadura, e os outros sete são convertidos, por meio de um conversor digital/analógico (DAC0800), em um sinal analógico (de 0 a 10V), cujo valor é usado como referência no controle da tensão de alimentação do motor. A inversão do sentido da

corrente na armadura é feita por meio de um *relé*, cujo chaveamento está associado ao valor lógico do bit designado para esta tarefa. A figura 8.10 esquematiza este processo.

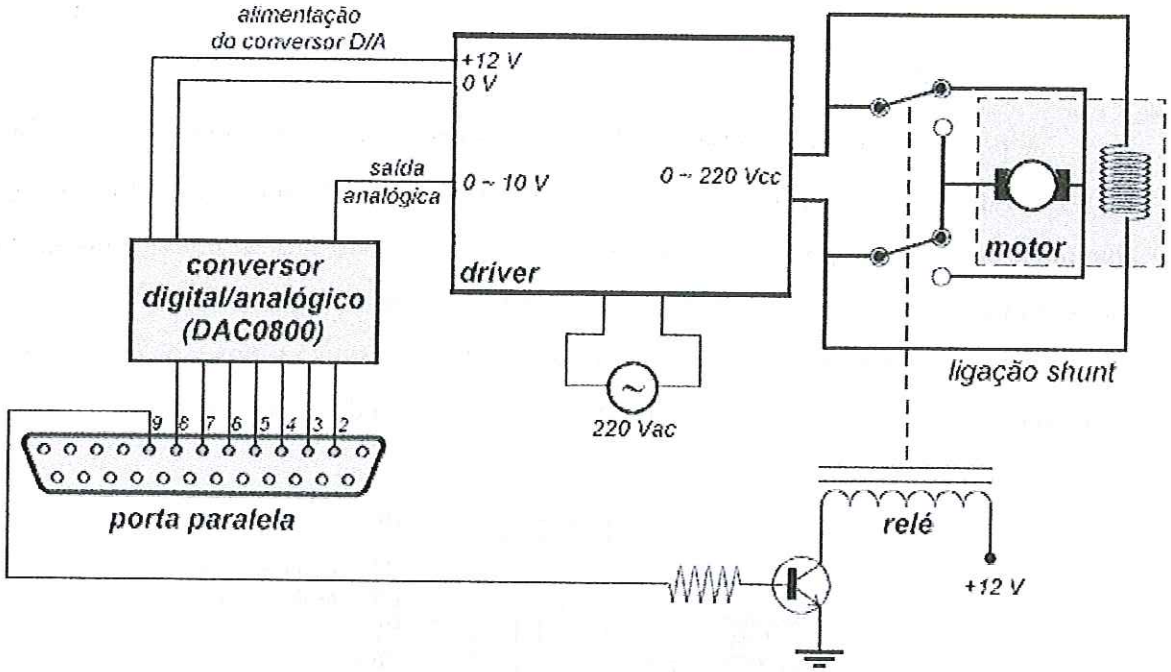


Figura 8.10 – Esquematização do processo de comunicação com o motor

Assim, considerando-se que o controle de tensão no motor é feito a partir de sete bits, pode-se configurar 128 valores distintos de tensão para cada um dos sentidos de rotação.

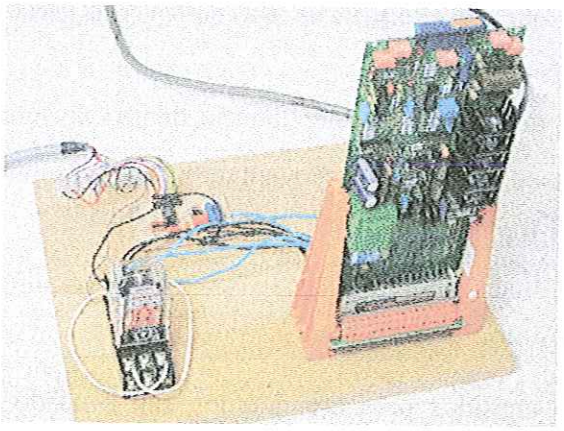


Figura 8.11 – Placa controladora (*driver*) do motor (à direita)

O *driver* do motor utiliza tiristores para efetuar o controle por ângulo de fase da tensão de alimentação. Após a retificação da tensão, os tiristores atuam de modo que, a cada semi-ciclo da rede, o motor seja alimentado somente quando a fase atingir um determinado ângulo ϕ, cujo valor é diretamente proporcional ao sinal de tensão (0 a 10V) enviado pelo conversor digital/analógico. Quando a tensão da rede for nula, os tiristores desligam o motor e o mantém neste estado até que a fase seja novamente igual ϕ. Assim, a cada semi-ciclo, o motor permanece conectado à rede durante um intervalo de tempo menor ou igual ao de um semi-ciclo (1/120 segundos, no caso da rede de 60Hz). A figura 8.12 ilustra o processo.

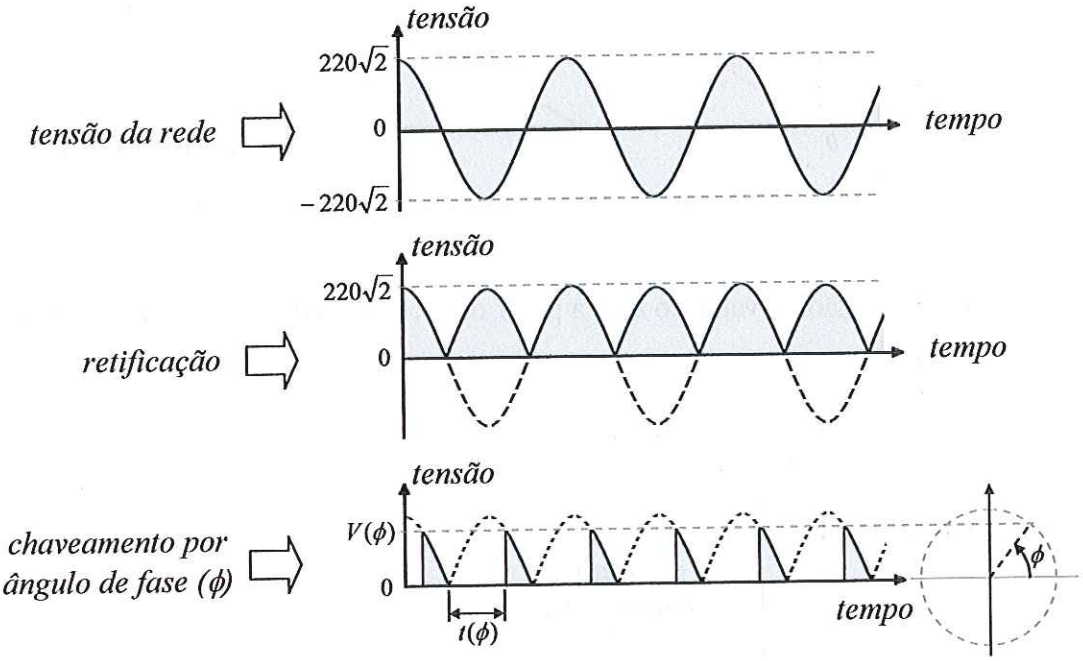


Figura 8.12 - Controle da tensão de alimentação por ângulo de fase

8.3.4 Interação da bicicleta com as inclinações do terreno

A variável tomada como referência para especificar, a cada iteração, o grau da inclinação do terreno, é a componente da aceleração gravitacional na direção do deslocamento da bicicleta virtual (A_g), cuja determinação foi apresentada no capítulo 7.

O envio dos bits através da porta paralela é feito com o uso da função:

```
_outp(0x378, vByte);
```

onde:

- *0x378* indica que o acesso à porta paralela foi solicitado.
- *vByte* é o valor do *byte* enviado, especificado na função acima na forma de um número inteiro entre 0 e 255.

A tensão de saída do conversor digital/analógico foi configurada para variar em relação ao *byte* emitido pela porta paralela segundo o gráfico abaixo:

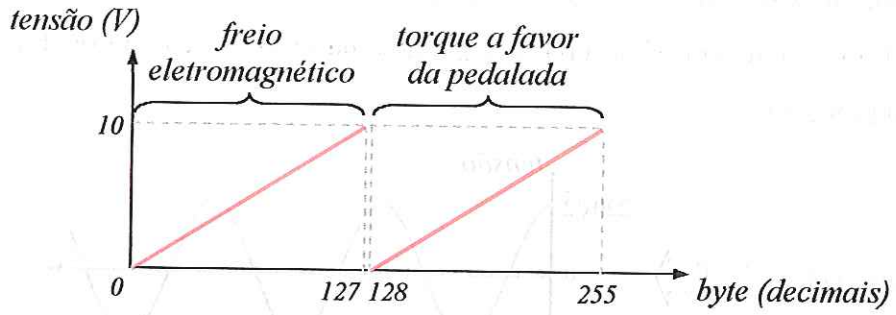


Figura 8.13 - Variação da tensão de saída do DAC em relação ao *byte* emitido

A determinação do valor do *byte* a partir da inclinação do terreno é feita com base no gráfico abaixo:

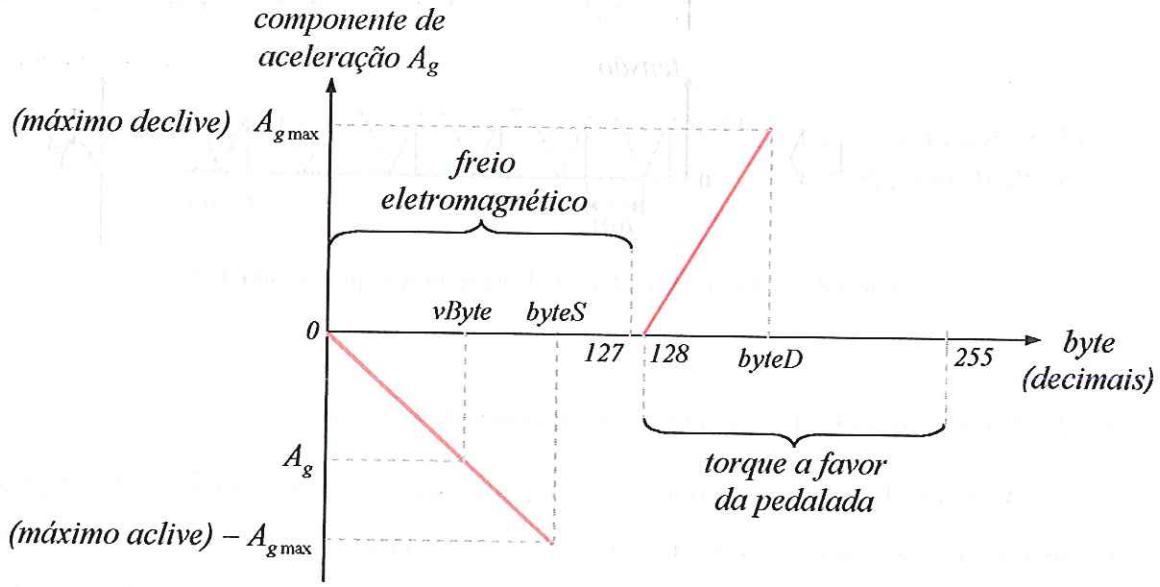


Figura 8.14 – Relação entre o valor do *byte* e a inclinação do terreno

de onde são obtidas as seguintes interpolações lineares:

a) Motor atuando como freio eletromagnético:

$$vByte = \frac{byteS}{-A_{g\max}} A_g$$

b) Motor exercendo torque a favor da pedalada:

$$vByte = \frac{(byteD - 128)}{A_{g\max}} A_g + 128$$

O valor $A_{g\max}$, referente à componente da aceleração gravitacional na direção do deslocamento da bicicleta, está associado ao maior declive existente no cenário virtual. Conseqüentemente, o maior aclive está associado ao valor $-A_{g\max}$.

Os valores *byteS* e *byteD* estabelecem limites para o torque fornecido pelo motor, respectivamente contra e a favor da pedalada.

Por meio de uma interface gráfica amigável, o programa possibilita ao terapeuta configurar o limite *byteS* de acordo com a força muscular da criança que estará interagindo com o ambiente virtual. O limite *byteD*, no entanto, é definido com o intuito de reduzir a resistência à pedalada (gerada pelo sistema de transmissão) nos movimentos descendentes. Seu valor é determinado empiricamente.

Considerando-se que a variável A_g foi declarada como real, o valor de *vByte* obtido pelas interpolações anteriores pode não ser inteiro. No entanto, a simples declaração de *vByte* como uma variável inteira automaticamente já proporciona o arredondamento do valor calculado para o inteiro mais próximo.

9 RESULTADOS E DISCUSSÃO

Como mencionado no início do capítulo anterior, o atual protótipo ainda não se mostra adequado ao ensaio envolvendo crianças portadoras de deficiência e, portanto, foi testado apenas com crianças normais. Contudo, já está sendo concebido um novo protótipo, contendo dispositivos mais seguros e confortáveis, para possibilitar a realização de experimentos envolvendo crianças deficientes.

Doze crianças, na faixa de cinco a oito anos de idade, foram submetidas ao teste. Todas, sem exceção, revelaram grande fascínio e motivação durante o experimento. A figura 9.0 mostra alguns desses ensaios.

Figura 9.0 – Ensaios do protótipo com crianças normais

O experimento revelou que cada criança tem uma afinidade maior com uma determinada característica do ambiente virtual. Algumas dão mais importância aos objetivos (tarefas) do jogo enquanto outras buscam explorar mais o cenário virtual. Esse é

um fator relevante, pois a aparência, os objetivos e a "jogabilidade" do ambiente virtual podem atrair ou afastar o interesse da criança deficiente. Aparentemente, um ambiente virtual com uma vasta diversidade de opções seria a solução mais adequada para o caso.

Outra importante observação decorreu do fato de algumas crianças mostrarem uma coordenação motora mais refinada do que outras e, conseqüentemente, uma maior facilidade para controlar a direção do seu movimento no ambiente virtual. Outras, no entanto, encontram dificuldades significativas, quase que comprometendo o seu entusiasmo pelo jogo. Para que isso seja evitado, faz-se necessário desenvolver um sistema que apresente possibilidades de controle em níveis crescentes de dificuldade, de forma que a coordenação motora da criança seja gradualmente aprimorada no decorrer do tratamento.

Notou-se também que algumas crianças possuíam maior força muscular do que outras (devido principalmente à diferença de idade), o que contribuiu para o sucesso do teste de configuração da potência do motor em cada caso ensaiado.

O trabalho foi apresentado aos profissionais da equipe de fisioterapia do Núcleo de Reabilitação Municipal de São José do Rio Preto, que se mostraram muito otimistas quanto às expectativas do projeto. Atentaram, porém, para a diversidade de níveis de deficiência existente em crianças, categorizando a aplicação deste projeto somente àquelas que já apresentam um mínimo de força muscular nas pernas e alguma coordenação motora. Ficou claro que este projeto trata casos específicos e que seria interessante expandir a tecnologia para o tratamento de outros níveis de deficiência.

O projeto também contou com o apoio do Dr. Paulo Sérgio Rodriguez, médico neuro-pediatra que atua junto à APAE e à Associação Renascer (instituições que atendem crianças e jovens portadores de deficiência). Suas expectativas para o projeto foram muito boas, citando que todo método de estimulação sensório motora pode ter grande aplicação na reabilitação de indivíduos portadores de deficiências múltiplas.

Segundo ele, "o projeto apresentado possui um objetivo bem definido e importante: integrar a estimulação motora à excitação das áreas cerebrais relacionadas à atenção e à concentração. Certamente, o portador de deficiência encontrará neste sistema uma forma prazerosa e interessante para a realização das atividades físicas. Com o passar do tempo, poder-se-á adequar o protótipo ao tratamento dos demais distúrbios motores causados por

outros tipos de paralisia cerebral". Como complemento, sugeriu o uso da bicicleta também na estimulação de pacientes portadores de hemiparesias e miopatias (doenças musculares).

10 CONCLUSÃO

O projeto atendeu às expectativas. Embora, até o presente momento, os testes tenham sido feitos com crianças normais, devido às condições do atual protótipo ainda serem inadequadas ao ensaio com crianças deficientes, comprovou-se que o sistema desperta grande fascínio e motivação em crianças, incentivando-as, de forma divertida, à prática das atividades físicas decorrentes da interação com o ambiente virtual. Como próximo passo, será projetado um novo protótipo, contendo recursos que possibilitem o experimento seguro e confortável com crianças portadoras de deficiência.

Os experimentos mostraram que a aparência do ambiente virtual e sua "jogabilidade" constituem importante fator para estímulo da motivação na criança. A implementação de uma grande diversidade de opções e objetivos para o jogo contribui para o sustento do seu interesse. Porém, dependendo do grau de comprometimento cognitivo causado pela lesão cerebral, a criança deficiente pode não demonstrar interesse para os padrões de divertimento aceitos pela maioria das outras crianças. Neste caso, uma possível alternativa seria acompanhar de perto o tratamento desta criança e observar o que lhe desperta curiosidade. Assim, poder-se-ia implementar um ambiente virtual especificamente relacionado aos objetos e ações causadores desta curiosidade.

Outro fator importante constatado nos experimentos foi a diversidade de níveis de coordenação motora existente entre as crianças. Crianças com pouca coordenação encontraram dificuldades significativas para comandar a direção do seu movimento no cenário virtual. Em se tratando de uma criança deficiente, uma grande dificuldade poderia comprometer o seu interesse pelo jogo, ou até mesmo lhe induzir uma sensação de incapacidade, abalando o seu estado psicológico. É então de suma importância que o projeto ofereça opções de controle em níveis crescentes de dificuldade, de forma que a coordenação motora da criança seja gradualmente aprimorada ao longo do tratamento.

Por fim, vale citar que, além do conhecimento tecnológico, o acompanhamento do tratamento de reabilitação em um grupo de crianças deficientes, sob a orientação de uma equipe clínica capacitada, constitui importante recurso para o sucesso de futuros projetos na área.

11 REFERÊNCIAS BIBLIGRÁFICAS

ALBUQUERQUE, A. L. P. (1999). *Cenários virtuais com um estudo de sincronismo de câmera*. 95 p. Dissertação (Mestrado) - Departamento de Informática, PUC- RIO, Rio de Janeiro. 1999.

AMATE, F. C. *et al.* (2004). *Jogos computadorizados no auxílio da reabilitação de atividades motoras*. Departamento de Engenharia Elétrica. Escola de Engenharia de São Carlos, São Carlos, 2004. Disponível em: <http://www.cbcomp.univali.br/anais/pdf/2003/for284.pdf>. Acesso em Maio de 2004.

ARAÚJO, R. B. (1996). *Especificação e análise de um sistema distribuído de realidade virtual*. 144 p. Tese (Doutorado) - Escola Politécnica, Universidade de São Paulo, São Paulo. 1996.

ASOCIAÇÃO DE ASSISTÊNCIA À CRIANÇA DEFICIENTE (2004). *Fisioterapia Infantil*. Disponível em: < www.aacd.org.br/centro_setores_fisio.asp>. Acesso em Maio de 2004.

BOLAS, M. T. (1994). *Human factors in the design of an immersive display*. IEEE Computer Graphics & Application, p. 55-59, Jan., 1994.

BOTELHO, Luiz. Jogos educacionais aplicados ao *e-learning*. Disponível em: <http://www.elearningbrasil.com.br/news/artigos/artigo_48.asp.> Acesso em: janeiro de 2004.

BURDEA, G. (1996). *Force and touch feedback for virtual reality*. John Wiley & Sons. New York, 1996.

BURDEA, G. (2000). *Haptic Feedback for Virtual Reality*. Rutgers University, NJ, EUA. CAIP Center. Disponível em: <http://www.caip.rutgers.edu/vrlab/publications/papers_2000.html>. Acesso em Julho de 2004.

BURDEA, G. C *et al.* (2002). *The Rutgers Master II—New Design, Force-Feedback Glove*. Rutgers University, NJ, EUA. Disponível em: <http://www.caip.rutgers.edu/vrlab/publications/papers_2002.html>. Acesso em Junho de 2004.

BURDEA, G. C. *et al.* (2003). *Haptic Effects for Virtual Reality-based Post-Stroke Rehabilitation*. Rutgers University, NJ, EUA. Disponível em: <http://www.caip.rutgers.edu/vrlab/publications/papers_2003.html>. Acesso em: Junho de 2004.

CARRARA, V. (2002). *Computação Gráfica.* Disponível em: <http://www.valcar.net>. Acesso em Janeiro de 2004.

CENTER FOR ADVANCED INFORMATION PROCESSING'S. (2003). *The Rutgers Ankle Rehabilitation Interface*. Rutgers University, NJ, EUA. Disponível em: <http://www.caip.rutgers.edu/vrlab/projects/ankle/ankle.html>. Acesso em Junho de 2004.

DELFT UNIVERSITY OF TECHNOLOGY. (2003). *Virtual Reality and Phobias.* Disponível em: <http://graphics.tudelft.nl/~vrphobia/index.html>. Acesso em: Junho de 2004.

DURAND, F. (2004). *Stereoscopy*. Disponível em: <www.mnemocine.com.br/fotografia/stereo.htm>. Acesso em Maio de 2004.

HANCOCK, D. (1995). *Virtual reality in search of middle ground.* IEEE Spectrum, January, 1995.

HAND, C. (1994). *Other faces of virtual reality.* First International Conference MHVR'94 - Lecture Notes in Computer Science n.1077, p. 107-116, Ed. Springer, Moscow, Russia, Sept., 1994.

IMMERSION. (2003). Disponível em: <http://www.immersion.com/>. Acesso em Junho de 2004.

INITION. (2004). *Inovative Graphics Solution.* Disponível em: <http://www.inition.co.uk/inition/product_hmd_cybermind_hires900.php>. Acesso em: Junho de 2004.

INSTITUTO BENJAMIN CONSTANT - IBC (2004). *Os conceitos de deficiência.* Disponível em: <www.ibcnet.org.br>. Acesso em Maio de 2004.

INSTITUTO BRASILEIRO DE GEOGRAFIA E ESTATÍSTICA - IBGE (2000). *Censo de 2000*. Disponível em: <www.ibge.gov.br>. Acesso em Maio de 2004.

IREX (2003). *Virtual Reality Technologies*. Toronto, Canadá, 2003. Disponível em: <www.irexonline.com>. Acesso em Dezembro de 2003.

JOHANSSON, A. (2000). *Notas de Aula.* Disciplina Aerofoto e Fotointerpretação. UNIFAP. Disponível em: <http://www.aryjohansson.pop.com.br/notas/naero08.htm>. Acesso em Fevereiro de 2004.

KALAWSKY, R. S. (1993). *The Science of Virtual Reality and Virtual Environments.* Addison-Wesley. 1993.

KISNER, C. (1999). *Exercícios terapêuticos, fundamentos e técnicas*. São Paulo, 3ª ed., Editora Manole.

KUHLEN. T; DOHLE, C. (2000). *Virtual reality for physically disabled people*. Institute of Technical Computer Science. Aachen, Germany, 2000.

LABORATÓRIO DE SISTEMAS INTEGRÁVEIS. (2002). *Realidade Virtual – Caverna Digital*. Disponível em: <http://www.lsi.usp.br/interativos/nrv/caverna.html>. Acesso em Julho de 2004.

LATTA, J. N. e OBERG, D. J. (1994). *A conceptual virtual reality model*. IEEE Computer Graphics & Applications. p. 23-29, Jan., 1994.

LINDSTEDT, E. (2003). *How well does child see?* Disponível em: <http://home.swipnet.se/~w-94583/Elisyn/hwman.html>. Acesso em Maio de 2004.

LUSTED, H. S.; KNAPP, R. B. (1994). *Medical Applications for Biocontroller Technology*. Proceedings of Medicine Meets Virtual Reality, San Diego, 1994.

MACHADO *et al.* (2002). *Realidade Virtual: Fundamentos e Aplicações*. Editora Visual Books, 2002.

MAHONEY, D. P. (1995). *Driving VR, Computer Graphics World*. p.22-33, May, 1995.

MICROSOFT C. (2002). *Microsoft DirectX 9.0 SDK Documentation*. Microsoft Corporation, 2002.

MICROVISION. (2002). *Augmented Reality*. Disponível em: <http://www.mvis.com>. Acesso em Julho de 2002.

MOLENDI, G.; PATRIARCA, M. (1992). *Virtual Reality: Medical Researches*. Technical Report. Universita degli Studi di Milano. Jan. 1992.

NAPOLI, J. D. (2003). *Informações básicas sobre deficiência física*. Disponível em: <www.acadef.com.br>. Acesso em Abril de 2004.

NATIONAL REABILITATION HOSPITAL (2004). *Research & Development*. Disponível em: <http://www.atnrc.org/r+d/r+d.html>. Acesso em Abril de 2004.

NETTO, A. V. (1998). *Prototipação de um torno CNC utilizando realidade virtual*. 131p. Dissertação (Mestrado) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos. 1998.

NETTO, A. V. *et al.* (2002). *Realidade Virtual - Definições, Dispositivos e Aplicações*. Notas Didáticas, n.34, ICMC-USP, São Carlos – SP, 2002.

OLIVEIRA, A. I. A. (2002). *Tecnologia Assistiva: Abordagem inovadora do Terapeuta Ocupacional*. Revista COFFITO, Edição nº 15, Junho de 2002.

OLIVEIRA, C. E. N. *et al.* (2000). *Fatores ambientais que influenciam a plasticidade do SNC*. Acta Fisiátrica, v.8, n.1, Abril de 2001.

PIMENTEL, K. e TEIXEIRA, K. (1995). *Virtual reality - through the new looking glass*. 2.ed. New York, McGraw-Hill, 1995.

REDE SARAH DE HOSPITAIS DE REABILITAÇÃO - SARAH (2004). *Informações sobre doenças tratadas*. Disponível em: <www.sarah.br>. Acesso em Maio e 2004.

REDE SOLIDARIEDADE, APOIO, COMUNICAÇÃO E INFORMAÇÃO - SACI (2002). *Questões amplas sobre a deficiência em geral*. Disponível em: <www.saci.org.br>. Acesso em Maio de 2004.

ROBERTSON, G. G. *et al.* (1993). *Non-immersive virtual reality*. IEEE Computer Graphics & Applications, p. 81-83, Feb.1993.

SABBATINE, R. M. E. (1993). *Realidade virtual e Medicina*. Revista Informédica, n.1, Nov de 1993.

SANTAROSA, L. M. C. *et al.*. (2002). *O processo de alfabetização de crianças com dificuldades de aprendizagem em ambientes lúdicos computacionais*. Disponível em: <http://www.niee.ufrgs.br/publicacoes/artigos/art_lu93.html>. Acesso em Maio de 2004.

SANTOS, S. M. S. *et al.* (2002). *O Papel da Plasticidade Cerebral na Fisioterapia*. Revista Cérebro & Mente, Junho de 2002.

SILICON GRAPHICS INCORPORATION (2004). Disponível em: <http://www.sgi.com/realitycenter/>. Acesso em Junho de 2004.

SOARES, T. S. (2002). *Paralisia Cerebral*. Disponível em: <http://thelma.soares.sites.uol.com.br/fisiothe.htm>. Acesso em Maio de 2004.

STURMAN, D. J.; ZELTZER, D. (1994). *A survey of glove-based input*. IEEE Computer Graphics & Application, p. 30-39, Jan., 1994.

TAUROCO, L. M. R.; ROLAND, L. C.- Jogos educacionais, CINTED-UFRGS, Novas Tecnologias na Educação, Março de 2004.

TELEMEDICINA. (2003). *Cirurgia à Distância e Telepresença*. Disponível em: <http://www.virtual.epm.br/material/tis/curr-med/temas/med5/>. Acesso em Julho de 2004.

THE AMERICAN SOCIETY OF MECHANICAL ENGINEERS. (2003). Disponível em: <http://www.memagazine.org/backissues/nov03/features/touching.html>. Acesso em Junho de 2004.

UNIVERSIDADE NACIONAL DE BRASÍLIA (2003). *LabRedes Virtual*. Disponível em: <http://www.redes.unb.br/labredesvirtual/>. Acesso em: Junho de 2004.

UNIVERSITY OF MICHIGAN (2003). *Virtual Reality Laboratory*. Disponível em: <http://www-vrl.umich.edu/intro/>. Acesso em Junho de 2004.

VICON MOTION SYSTEMS. (2003). *Gait Analysis*. Disponível em: <http://www.vicon.com/main/applications/gait_analysis.shtml>. Acesso em: Junho de 2004.

VINCE, J. (1995). *Virtual reality systems*. Cambridge, Addison-Wesley, 1995.

VIRTUAL & REALLY (2002). Disponível em: <http://www.really.ru/review/russian_project.html>. Acesso em: Junho de 2004.

VR LAB. (2002). *Research*. Tsukuba University. Disponível em: http://intron.kz.tsukuba.ac.jp/vrlab_web/research/research_e.html>. Acesso em: Junho de 2004.

WANN, J. P. (1993). *Motor Skill Learning in Cerebral Palsy: Movement, Action and Computer-Enhanced Therapy*. Baillière's Clinical Neurology 2, 1993.

WARNER D. *et al.* (1994). *Bio-Cybernetics – A Biologically Responsive Interactive Interface*. Proceedings of Medicine Meets Virtual Reality, San Diego, 1994.

WATT, A. H. (1992). *Advanced animation and rendering techniques: theory and practice*. San Francisco, CA : ACM Press, 455 p. il. (006.6 W344a).

WATT, A. H. (2000). *3D Computer Graphics*. 3rd ed. Addison-Wesley, 2000.

APÊNDICE A - RECURSOS MATEMÁTICOS DO *DIRECTX GRAPHICS*

Estruturas que definem um vetor 3D no *DirectX Graphics*

Sintaxe:

```
typedef struct D3DXVECTOR3 : public D3DVECTOR {
public:
    D3DXVECTOR3() {};
    D3DXVECTOR3( CONST FLOAT * );
    D3DXVECTOR3( CONST D3DVECTOR& );
    D3DXVECTOR3( FLOAT x, FLOAT y, FLOAT z );

    // casting
    operator FLOAT* ();
    operator CONST FLOAT* () const;

    // assignment operators
    D3DXVECTOR3& operator += ( CONST D3DXVECTOR3& );
    D3DXVECTOR3& operator -= ( CONST D3DXVECTOR3& );
    D3DXVECTOR3& operator *= ( FLOAT );
    D3DXVECTOR3& operator /= ( FLOAT );

    // unary operators
    D3DXVECTOR3 operator + () const;
    D3DXVECTOR3 operator - () const;

    // binary operators
    D3DXVECTOR3 operator + ( CONST D3DXVECTOR3& ) const;
    D3DXVECTOR3 operator - ( CONST D3DXVECTOR3& ) const;
    D3DXVECTOR3 operator * ( FLOAT ) const;
    D3DXVECTOR3 operator / ( FLOAT ) const;

    friend D3DXVECTOR3 operator * ( FLOAT, CONST struct D3DXVECTOR3& );

    BOOL operator == ( CONST D3DXVECTOR3& ) const;
    BOOL operator != ( CONST D3DXVECTOR3& ) const;

} D3DXVECTOR3, *LPD3DXVECTOR3;

typedef struct D3DXVECTOR3 {
    FLOAT x;
    FLOAT y;
    FLOAT z;
} D3DXVECTOR3;
```

Estruturas que definem uma matriz 4x4 no *DirectX Graphics*

Sintaxe:

```
typedef struct D3DXMATRIX : public D3DMATRIX {
public:
    D3DXMATRIX() {};
    D3DXMATRIX( CONST FLOAT * );
    D3DXMATRIX( CONST D3DMATRIX& );
    D3DXMATRIX( FLOAT _11, FLOAT _12, FLOAT _13, FLOAT _14,
                FLOAT _21, FLOAT _22, FLOAT _23, FLOAT _24,
                FLOAT _31, FLOAT _32, FLOAT _33, FLOAT _34,
                FLOAT _41, FLOAT _42, FLOAT _43, FLOAT _44 );

    // access grants
    FLOAT& operator () ( UINT Row, UINT Col );
    FLOAT  operator () ( UINT Row, UINT Col ) const;

    // casting operators
    operator FLOAT* ();
    operator CONST FLOAT* () const;

    // assignment operators
    D3DXMATRIX& operator *= ( CONST D3DXMATRIX& );
    D3DXMATRIX& operator += ( CONST D3DXMATRIX& );
    D3DXMATRIX& operator -= ( CONST D3DXMATRIX& );
    D3DXMATRIX& operator *= ( FLOAT );
    D3DXMATRIX& operator /= ( FLOAT );

    // unary operators
    D3DXMATRIX operator + () const;
    D3DXMATRIX operator - () const;

    // binary operators
    D3DXMATRIX operator * ( CONST D3DXMATRIX& ) const;
    D3DXMATRIX operator + ( CONST D3DXMATRIX& ) const;
    D3DXMATRIX operator - ( CONST D3DXMATRIX& ) const;
    D3DXMATRIX operator * ( FLOAT ) const;
    D3DXMATRIX operator / ( FLOAT ) const;

    friend D3DXMATRIX operator * ( FLOAT, CONST D3DXMATRIX& );

    BOOL operator == ( CONST D3DXMATRIX& ) const;
    BOOL operator != ( CONST D3DXMATRIX& ) const;

} D3DXMATRIX, *LPD3DXMATRIX;

typedef struct _D3DMATRIX {
    union {
        struct {
            float        _11, _12, _13, _14;
            float        _21, _22, _23, _24;
            float        _31, _32, _33, _34;
            float        _41, _42, _43, _44;
        };
        float m[4][4];
    };
} D3DMATRIX;
```

Funções do *DirectX Graphics* para operações com vetores e matrizes

❖ *Adição de dois vetores*

Sintaxe:

```
D3DXVECTOR3 *D3DXVec3Add(
    D3DXVECTOR3 *pOut,
    CONST D3DXVECTOR3 *pV1,
    CONST D3DXVECTOR3 *pV2
);
```

Parâmetros:

- *pOut*
  [entrada/saída] Ponteiro para um vetor que será o resultado da operação.
- *pV1* e *pV2*
  [entradas] Ponteiros para os vetores que serão somados.

Valor retornado pela função:

Ponteiro para um vetor que será o resultado da operação.

❖ *Subtração de vetores*

Sintaxe:

```
D3DXVECTOR4 *D3DXVec3Subtract(
    D3DXVECTOR3 *pOut,
    CONST D3DXVECTOR3 *pV1,
    CONST D3DXVECTOR3 *pV2
);
```

Parâmetros:

- *pOut*
  [entrada/saída] Ponteiro para um vetor que será o resultado da operação.
- *pV1* e *pV2*
  [entradas] Ponteiros para os vetores envolvidos na operação **pV1 - *pV2*.

Valor retornado pela função:

Ponteiro para um vetor que será o resultado da operação.

❖ *Normalização de um vetor*

Sintaxe:

```
D3DXVECTOR3 *WINAPI D3DXVec3Normalize(
    D3DXVECTOR3 *pOut,
    CONST D3DXVECTOR3 *pV
);
```

Parâmetros:

- *pOut*
  [entrada/saída] Ponteiro para um vetor que será o resultado da operação.
- *pV*
  [entrada] Ponteiro para o vetor que será normalizado.

Valor retornado pela função:

Ponteiro para um vetor que será o resultado da operação.

❖ *Módulo de um vetor*

Sintaxe:

```
FLOAT D3DXVec3Length(
    CONST D3DXVECTOR3 *pV
);
```

Parâmetros:

- *pV*
  [entrada] Ponteiro para o vetor do qual se deseja extrair o módulo.

Valor retornado pela função:

Módulo do vetor.

❖ ***Produto escalar***

Sintaxe:

```
FLOAT D3DXVec3Dot(
    CONST D3DXVECTOR3 *pV1,
    CONST D3DXVECTOR3 *pV2
);
```

Parâmetros:

- *pV1* e *pV2*
  [entradas] Ponteiros para os vetores envolvidos no produto escalar.

Valor retornado pela função:

Produto escalar dos vetores.

❖ ***Produto vetorial***

Sintaxe:

```
D3DXVECTOR3 *WINAPI D3DXVec3Cross(
    D3DXVECTOR3 *pOut,
    CONST D3DXVECTOR3 *pV1,
    CONST D3DXVECTOR3 *pV2,
);
```

Parâmetros:

- *pOut*
  [entrada/saída] Ponteiro para um vetor que será o resultado da operação.
- *pV1* e *pV2*
  [entradas] Ponteiros para os vetores envolvidos na operação **pV1* x **pV2*.

Valor retornado pela função:

Ponteiro para um vetor que será o resultado do produto vetorial.

❖ ***Multiplicação de um vetor por um escalar***

Sintaxe:

```
D3DXVECTOR3 *D3DXVec3Scale(
    D3DXVECTOR3 *pOut,
    CONST D3DXVECTOR3 *pV,
    FLOAT s
);
```

Parâmetros:

- *pOut*
  [entrada/saída] Ponteiro para um vetor que será o resultado da operação.
- *pV*
  [entrada] Ponteiro para o vetor que será multiplicado pelo escalar.
- *s*
  [entrada] Valor escalar.

Valor retornado pela função:

Ponteiro para um vetor que será o resultado da operação.

❖ ***Produto de duas matrizes***

Sintaxe:

```
D3DXMATRIX *D3DXMatrixMultiply(
    D3DXMATRIX *pOut,
    CONST D3DXMATRIX *pM1,
    CONST D3DXMATRIX *pM2
);
```

Parâmetros:

- *pOut*
  [entrada/saída] Ponteiro para uma matriz que será o resultado da operação.
- *pM1* e *pM2*
  [entradas] Ponteiros para as matrizes envolvidas na operação **pM1* . **pM2*.

Valor retornado pela função:

Ponteiro para uma matriz que será o resultado da operação.

❖ ***Determinante de uma matriz***

Sintaxe:

```
FLOAT D3DXMatrixDeterminant(
    CONST D3DXMATRIX *pM
);
```

Parâmetros:

- *pM*
  [entrada] Ponteiro para uma matriz que da qual se deseja extrair o determinante.

Valor retornado pela função:

Determinante na matriz.

❖ ***Inversa de uma matriz***

Sintaxe:

```
D3DXMATRIX *D3DXMatrixInverse(
    D3DXMATRIX *pOut,
    FLOAT *pDeterminant,
    CONST D3DXMATRIX *pM
);
```

Parâmetros:

- *pOut*
  [entrada/saída] Ponteiro para uma matriz que será o resultado da operação.
- *pDeterminant*
  [entrada/saída] Ponteiro para uma variável real que contém o determinante da matriz que será invertida.
- *pM*
  [entrada] Ponteiro para a matriz que será invertida.

Valor retornado pela função:

Ponteiro para uma matriz que será o resultado da operação. Se a inversão falhar, será retornado o valor NULL.

❖ ***Matriz Identidade***

Sintaxe:

```
D3DXMATRIX *D3DXMatrixIdentity(
    D3DXMATRIX *pOut
);
```

Parâmetros:

- *pOut*
  [entrada/saída] Ponteiro para uma matriz que será a identidade.

Valor retornado pela função:

Ponteiro para uma matriz que será a identidade.

❖ ***Transposta de uma matriz***

Sintaxe:

```
D3DXMATRIX *D3DXMatrixTranspose(
    D3DXMATRIX *pOut,
    CONST D3DXMATRIX *pM
);
```

Parâmetros:

- *pOut*
  [entrada/saída] Ponteiro para uma matriz que será a transposta.
- *pM*
  [entrada] Ponteiro para uma matriz da qual se deseja obter a transposta.

Valor retornado pela função:

Ponteiro para uma matriz que será a transposta.

❖ ***Matriz para efetuar a transformação de translação***

Sintaxe:

```
D3DXMATRIX *D3DXMatrixTranslation(
    D3DXMATRIX *pOut,
    FLOAT x,
    FLOAT y,
    FLOAT z
);
```

Parâmetros:

- *pOut*
  [entrada/saída] Ponteiro para a matriz que efetuará a transformação de translação.
- *x*
  [entrada] Deslocamento na direção do eixo *x*.
- *y*
  [entrada] Deslocamento na direção do eixo *y*.
- *z*
  [entrada] Deslocamento na direção do eixo *z*.

Valor retornado pela função:

Ponteiro para a matriz que efetuará a transformação de translação.

❖ ***Matriz para efetuar a transformação de rotação em torno do eixo x***

Sintaxe:

```
D3DXMATRIX *D3DXMatrixRotationX(
    D3DXMATRIX *pOut,
    FLOAT Angle
);
```

Parâmetros:

- *pOut*
  [entrada/saída] Ponteiro para a matriz que efetuará a transformação de rotação em torno do eixo *x*.
- *Angle*
  [entrada] Ângulo de rotação em radianos.

Valor retornado pela função:

Ponteiro para a matriz que efetuará a transformação de rotação em torno do eixo *x*.

❖ ***Matriz para efetuar a transformação de rotação em torno do eixo y***

Sintaxe:

```
D3DXMATRIX *D3DXMatrixRotationY(
    D3DXMATRIX *pOut,
    FLOAT Angle
);
```

Parâmetros:

- *pOut*
  [entrada/saída] Ponteiro para a matriz que efetuará a transformação de rotação em torno do eixo *y*.
- *Angle*
  [entrada] Ângulo de rotação em radianos.

Valor retornado pela função:

Ponteiro para a matriz que efetuará a transformação de rotação em torno do eixo *y*.

❖ ***Matriz para efetuar a transformação de rotação em torno do eixo z***

Sintaxe:

```
D3DXMATRIX *D3DXMatrixRotationZ(
    D3DXMATRIX *pOut,
    FLOAT Angle
);
```

Parâmetros:

- *pOut*
  [entrada/saída] Ponteiro para a matriz que efetuará a transformação de rotação em torno do eixo *z*.
- *Angle*
  [entrada] Ângulo de rotação em radianos.

Valor retornado pela função:

Ponteiro para a matriz que efetuará a transformação de rotação em torno do eixo *z*.

❖ ***Matriz para efetuar a transformação de rotação em torno de um eixo arbitrário***

Sintaxe:

```
D3DXMATRIX *D3DXMatrixRotationAxis(
    D3DXMATRIX *pOut,
    CONST D3DXVECTOR3 *pV,
    FLOAT Angle
);
```

Parâmetros:

- *pOut*
  [entrada/saída] Ponteiro para a matriz que efetuará a transformação de rotação em torno do eixo definido.
- *pV*
  [entrada] Ponteiro para um vetor que define o eixo de rotação.
- *Angle*
  [entrada] Ângulo de rotação em radianos.

Valor retornado pela função:

Ponteiro para a matriz que efetuará a transformação de rotação em torno do eixo definido.

❖ ***Matriz para efetuar a transformação de variação de escala***

Sintaxe:

```
D3DXMATRIX *D3DXMatrixScaling(
    D3DXMATRIX *pOut,
    FLOAT sx,
    FLOAT sy,
    FLOAT sz
);
```

Parâmetros:

- *pOut*
  [entrada/saída] Ponteiro para a matriz que efetuará a transformação de variação de escala.
- *sx*
  [entrada] Fator de escalonamento na direção *x*.
- *sy*
  [entrada] Fator de escalonamento na direção *y*.

- *sz*
  [entrada] Fator de escalonamento na direção *z*.

Valor retornado pela função:

Ponteiro para a matriz que efetuará a transformação de variação de escala.

❖ ***Transformação de um vetor 3D por uma operação matricial***

Sintaxe:

```
D3DXVECTOR3 *WINAPI D3DXVec3Transform(
    D3DXVECTOR3 *pOut,
    CONST D3DXVECTOR3 *pV,
    CONST D3DXMATRIX *pM
);
```

Parâmetros:

- *pOut*
  [entrada/saída] Ponteiro para um vetor que será o resultado da operação matricial.
- *pV*
  [entrada] Ponteiro para o vetor que será multiplicado pela matriz.
- *pM*
  [entrada] Ponteiro para a matriz de transformação.

Valor retornado pela função:

Ponteiro para um vetor que será o resultado da operação matricial.