UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
FACULDADE DE ARQUITETURA
ESCOLA DE ENGENHARIA
PROGRAMA DE PÓS-GRADUAÇÃO EM DESIGN
DISSERTAÇÃO DE MESTRADO

SÉRGIO LEANDRO DOS SANTOS
ORIENTADOR: PROF. DR. FÁBIO GONÇALVES TEIXEIRA

CONCEPÇÃO E DESENVOLVIMENTO DE UMA INTERFACE GRÁFICA PARA INTERAÇÃO TRIDIMENSIONAL

Porto Alegre
Maio de 2009.

SÉRGIO LEANDRO DOS SANTOS

CONCEPÇÃO E DESENVOLVIMENTO DE UMA INTERFACE GRÁFICA PARA INTERAÇÃO TRIDIMENSIONAL

Dissertação de Mestrado apresentada como parte dos requisitos para obtenção do título de Mestre em Design pelo Programa de Pós-Graduação em Design da Universidade Federal do Rio Grande do Sul - UFRGS, área de concentração: Design e Tecnologia.

ORIENTADOR: PROF. DR. FÁBIO GONÇALVES TEIXEIRA

Porto Alegre
Maio de 2009.

S237c Santos, Sérgio Leandro dos

Concepção e desenvolvimento de uma interface gráfica para interação tridimensional / Sérgio Leandro dos Santos. – 2009.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Escola de Engenharia e Faculdade de Arquitetura. Programa de Pós-Graduação em Design. Porto Alegre, BR-RS, 2009.

Orientador: Prof. Dr. Fábio Gonçalves Teixeira

1. Interface gráfica. 2. Computação gráfica. 3. Interação homem-computador. I. Teixeira, Fábio Gonçalves, orient. II. Título.

CDU-681.32(043)

SÉRGIO LEANDRO DOS SANTOS

CONCEPÇÃO E DESENVOLVIMENTO DE UMA INTERFACE GRÁFICA PARA INTERAÇÃO TRIDIMENSIONAL

Dissertação de Mestrado julgada e aprovada como parte dos requisitos para obtenção do título de Mestre em Design pelo Programa de Pós-Graduação em Design da Universidade Federal do Rio Grande do Sul - UFRGS, área de concentração: Design e Tecnologia.

Aprovada em: 13 de Julho de 2009.

Banca Examinadora:

_____

Prof. Dr. Fábio Ferreira da Costa Campos
Universidade Federal de Pernambuco

_____

Prof. Dr. Régio Pierre da Silva
Universidade Federal do Rio Grande do Sul

_____

Prof. Dr. José Luis Farinatti Aymone
Universidade Federal do Rio Grande do Sul

_____

Prof. Dr. Fábio Gonçalves Teixeira - Orientador
Universidade Federal do Rio Grande do Sul

À Ká, MVC.

AGRADECIMENTOS

Gostaria de agradecer sinceramente a todos aqueles que de alguma forma participaram positivamente de mais esta experiência em minha vida e especialmente:

Ao meu orientador, Professor Fábio Teixeira, pela confiança no meu trabalho, por me incentivar na pesquisa e na docência e pela orientação precisa no momento certo, muito obrigado.

Ao Professor Fábio Campos e José Luis Farinatti Aymone, que participaram da banca examinadora e muito contribuíram com suas observações enriquecedoras, fundamentais para a conclusão deste trabalho.

Aos Professores Régio Pierre da Silva e Tânia Luisa Koltermann da Silva, pela companhia nos almoços no Antônio, pelos sabios conselhos quanto a vida acadêmica e pela ajuda constante ao longo de todo o trabalho.

Aos amigos do núcleo de pesquisa Virtual Design e do Departamento de Design e Expressão Gráfica que me acompanharam nesta jornada, pelo companheirismo na hora do trabalho e pela diversão nos intervalos.

Ao PGDesign e a UFRGS, pela oportunidade de aprendizagem, formação e qualificação.

À minha família, pelo apoio e compreensão em todos os momentos, em especial quando que não pude estar com eles.

Aos meus Pais, pelos valores espirituais, morais e principalmente pelo exemplo ao longo de toda vida.

À Karina, pelo amor, carinho, compreensão, amizade e muita, muita, muita paciência.

A Deus, pela oportunidade de evolução.

RESUMO

DOS SANTOS, Sérgio Leandro. **Concepção e Desenvolvimento de uma Interface Gráfica para Interação Tridimensional**. Porto Alegre, 2009. 108p Dissertação (Mestrado em Design) Programa de Pós-Graduação em Design, UFRGS, 2009.

Este trabalho tem por objetivo desenvolver um protótipo de uma interface gráfica para interação tridimensional que permita a manipulação de objetos e a navegação tridimensional com desempenho gráfico adequado às necessidades dos usuários de aplicativos 3D. A construção do protótipo utiliza o ambiente Delphi e é aplicada na plataforma de desenvolvimento T-CADE, visando à melhoria da usabilidade deste aplicativo a partir da melhoria do desempenho gráfico e da interatividade. Para a concepção do trabalho são investigados os conceitos de interação homem-computador, engenharia de usabilidade, engenharia de software, metodologia de projeto, computação gráfica, interação 3D e bibliotecas gráficas. Novas classes gráficas baseadas em programação orientada a objetos são implementadas, bem como algoritmos para resolver o problema de seleção de objetos e reconhecimento do ponto tridimensional da seleção sobre a superfície selecionada. Além disto, são desenvolvidas ferramentas gráficas, para facilitar o movimento de câmera e o controle de visualização da cena, e técnicas de manipulação de objeto mais interativas e diretas. São mostrados os novos modos de visualização possíveis com a criação da nova interface e as diversas intervenções no sentido de melhorar a usabilidade do software. A criação das novas ferramentas baseia-se nos critérios básicos de usabilidade. O protótipo da nova interface mostra ótimos resultados com relação ao desempenho gráfico apresentando grande melhoria na facilidade e qualidade da interação tridimensional.

Palavras-chave: Interface Gráfica. Computação Gráfica. Tridimensional. Usabilidade. Interatividade.

ABSTRACT

DOS SANTOS, Sérgio Leandro. **Concepção e Desenvolvimento de uma Interface Gráfica para Interação Tridimensional**. Porto Alegre, 2009. 108p Dissertação (Mestrado em Design) Programa de Pós-Graduação em Design, UFRGS, 2009.

The objective of this work is to develop a prototype of a graphic interface for tridimentional interaction that allows the object manipulation and tridimentional navigation with adequate graphic performance to the needs of users of 3D software. The construction of the prototype was done using the Delphi environment aplied to the T-CADE Development Plataform ainming the improvement of usability of this software through the improvement of graphic performance and interativity. In order to do this work the concepts of human-computer interaction, usability engeneering, software emgeneering, design methods, Computer graphics, 3D interaction and Graphics librarys are investigated. New graphics classes are implemented based on object oriented programming. Algorithms also are created to solve the problem of object selection and recognition of the tridimentional point of selection over the surface selected. Beyond that, graphics tools are created to facilitate the movement of the camera and the control of scene visualization, techniques of object manipulation more interactive and direct are implemented. The new visualization modes, made possible with the creation of the new interface, and many interventions in order to improve usability are shown. The new interface prototype has shown very good results on the graphics performance presenting great improvement regarding to simplicity and quality of tridimentional interaction as well.

Key-words: Graphic interface. Computer Graphics. Tridimentional. Usability. Interactivity.

SUMÁRIO

1 INTRODUÇÃO 12

1.1 DESCRIÇÃO DAS OCORRÊNCIAS OBJETIVAS 13

1.2 DEMARCAÇÃO DO NÍVEL DE INVESTIGAÇÃO DO FENÔMENO 14

1.3 PROBLEMA DE PESQUISA 14

1.4 HIPÓTESE BÁSICA 15

1.5 VARIÁVEIS 15

1.6 OBJETIVOS 15

1.6.1 Objetivo Geral 15

1.6.2 Objetivos Específicos 15

1.7 JUSTIFICATIVA 15

2 FUNDAMENTAÇÃO TEÓRICA 18

2.1 PROCESSO DE DESENVOLVIMENTO DE PRODUTO 19

2.1.1 O Projeto Informacional 21

2.1.2 O Projeto Conceitual 24

2.2 INTERAÇÃO HOMEM-COMPUTADOR 32

2.2.1 Interação 3D 33

2.3 ENGENHARIA DE USABILIDADE 38

2.3.1 Critérios básicos de Usabilidade 39

2.3.2 Engenharia de Usabilidade com desconto 41

2.4 ENGENHARIA DE SOFTWARE 45

2.4.1 Inter-relação entre Sistemas computacionais 47

2.4.2 Prototipagem 48

2.4.3 Programação orientada a objeto 51

2.5 COMPUTAÇÃO GRÁFICA 53

2.6 BIBLIOTECAS GRÁFICAS 56

2.6.1 A Biblioteca DirectX 60

2.6.2 A biblioteca OpenGL 61

2.6.3 A Biblioteca Glscene 65

3 PROJETO DO SISTEMA 70

3.1 ESCOPO DO PROJETO 70

3.2 REQUISITOS DO PROJETO 71

3.3 SELEÇÃO DE CONCEITOS 75

3.4 RESTRIÇÕES DE PROJETO 79

3.5 A ESTRUTURA DO T-CADE 79

3.6 A NOVA INTERFACE GRÁFICA 80

3.6.1 As Novas Classes Gráficas 81

4 IMPLEMENTAÇÃO COMPUTACIONAL 83

4.1 SELEÇÃO DE OBJETOS 83

4.2 LOCALIZAÇÃO DO PONTO 3D 85

4.3 CRIAÇÃO INTERATIVA DE OBJETOS 87

4.4 MANIPULAÇÃO DIRETA DE OBJETOS 88

4.5 VISUALIZAÇÃO 3D 89

4.6 MODOS DE VISUALIZAÇÃO 93

4.7 BIBLIOTECA DE MATERIAIS 95

4.8 COMUNICAÇÃO COM O USUÁRIO 96

5 RESULTADOS 98

6 CONSIDERAÇÕES FINAIS E CONCLUSÕES 108

6.1 TRABALHOS FUTUROS 110

REFERÊNCIAS BIBLIOGRÁFICAS 111

LISTA DE ABREVIATURAS E SIGLAS

CAE	- *Computer-Aided Engineering*
CAD	- *Computer-Aided Design*
OpenGL	- *Open Graphics Library*
2D	- Bidimensional
3D	- Tridimensional
IHC	- Interação Homem-computador
CG	- Computação Gráfica
API	- *Application Programming Interface*
QFD	- *Quality Function Deployment* – Desdobramento da Função Qualidade
SQFD	- *Software Quality Function Deployment* – Desdobramento da Função Qualidade de software.
SQA	- *Software Quality Assurance* - Garantia da Qualidade de Software
NC	- Necessidades do Cliente.
RQ	- Requisitos de Projeto.
PC	- *Personal Computer* – Computador pessoal.
CPU	- *Central Processing Unit* – Unidade de Processamento Central
GPU	- *Graphics* Processing *Unit* – Unidade de Processamento Gráfico
GKS	- *Graphical Kernel System*
PHIGS	- *Programmer's Hierarchical Interactive Graphics System*
ARB	- *OpenGL Architecture Review Board*
GLU	- *OpenGL Utility Library*
RAD	- *Rapid Application Development*
COM	-*Component Object Model*
DLL	-*Dynamic Link Libraries*

LISTA DE FIGURAS

Figura 1: Superfície de Revolução no T-CADE.....13
Figura 2: Modelo de processo de desenvolvimento de produtos. 20
Figura 3: Matriz da Casa da Qualidade. 23
Figura 4: Matriz Morfológica.....27
Figura 5: Geração de conceitos a partir dos princípios de solução. 29
Figura 6: Exemplo de Matriz de Pugh.....30
Figura 7: Manipulação direta de um retângulo.....36
Figura 8: O Paradigma de abstração dos quatro universos.....55
Figura 9: Algumas Interfaces de um Sistema. 57
Figura 10: Qualidade da Imagem da DirectX versão 10. 61
Figura 11: Primitivas OpenGL. 64
Figura 12: Interface do GLScene com os diversos objetos disponíveis. 67
Figura 13: A linha vermelha evidencia a área de trabalho da plataforma T-CADE. 70
Figura 14: Necessidades do usuário. 72
Figura 15: Matriz de inter-relações (requisitos de projetoX necessidades do usuário). 73
Figura 16: Legenda de Correlação.....73
Figura 17: Correlação entre requisitos de usuário e requisitos de projeto.....74
Figura 18: Matriz de Pugh, triagem de conceitos. 77
Figura 19: Matriz de avaliação por comparação dos pesos dos critérios de seleção. 77
Figura 20: Valoração de Critérios Qualitativos. 78
Figura 21: Valor da função Utilidade. 78
Figura 22: Hierarquia de classes de objetos geométricos.....80
Figura 23: As novas classes Gráficas. 82
Figura 24: Janela Azul Cheia, da Esquerda para a Direita. 84
Figura 25: Janela verde tracejada, da direita para a esquerda. 85
Figura 26: Esfera e Seta 3D, indicando o ponto 3D e o vetor normal à superfície. 86
Figura 27: Arco Início, Centro e fim na Interface Original. 87
Figura 28: Arco Centro, Início e fim na Nova Interface. 87
Figura 29: Manipulação direta na Nova Interface. 89
Figura 30: O Cubo de orientação e visualização. 90
Figura 31: Vista a partir da aresta do cubo. 90
Figura 32: Clique sobre o vértice do cubo.....91
Figura 33: QuadMenu e cursores das ferramentas de câmera. 92
Figura 34: Acionamento em um único movimento, clicar e arrastar na direção da ferramenta desejada. 93
Figura 35: Modos de visualização: Sem malha, WireFrame, Volume Wire e Hide. 94
Figura 36: Modos de Visualização: Shaded Lines, FlatShade, SmoothShade e Ghost. 94
Figura 37: Mudando a resolução da malha. 95
Figura 38: As cores dos objetos evidenciam os diferentes materiais. 96
Figura 39: Texto informativo no cursor. 97
Figura 40: Representação de uma superfície de Revolução na interface original. 99

Figura 41: Três passos necessários para a geração da malha de visualização da superfície na interface original. 100
Figura 42: Criação de uma superfície de Revolução na nova interface. 101
Figura 43: Modo Ghost mostrando a interseção entre superfícies. 102
Figura 44: Modelagem com interseção de superfícies. 102
Figura 45: As linhas vermelhas mostram a interseção. 103
Figura 46: No modo ShadedLines é possível visualizar os elementos da malha. 104
Figura 47: Interseção entre asa e corpo do avião 104
Figura 48: Objeto movido para esquerda, sua malha permanece no local de origem. 105
Figura 49: Superfície sendo movida na nova interface. 106

LISTA DE TABELAS

Tabela 1: Exemplo de matriz para valoração de critérios. 31
Tabela 2: Exemplo de matriz para determinação da função Utilidade. 31
Tabela 3: Desempenho gráfico medido em Quadros por Segundo (FPS). 98

1 INTRODUÇÃO

Um dos principais usos da computação gráfica está no processo de design, tanto nos sistemas de design de produtos em geral como nos projetos de engenharia e arquitetura. Geralmente referidos como CAD (*Computer-Aided Design*), os métodos de projeto auxiliado por computador são parte da rotina no processo de projeto de produtos nos mais diversos setores: prestação de serviço, automotivo, naval, aeroespacial e outros. Os sistemas CAD são artefatos digitais que permitem desde a concepção e modelagem tridimensional até a execução do detalhamento do projeto e dos desenhos técnicos, economizando tempo e aumentando consideravelmente a precisão do projeto. Da mesma forma, a Engenharia auxiliada por computador (em inglês, *Computer-Aided Engineering*, ou simplesmente CAE) é o uso da computação auxiliando em processos como análises, simulações, verificação de interferências, manufaturas, diagnósticos e reparos. Ferramentas computacionais que foram desenvolvidas para dar suporte a estas atividades são consideradas ferramentas CAE.

De acordo com Teixeira *et al*. (2008b), a tecnologia atual permite que praticamente todas as etapas de um projeto sejam realizadas através de ferramentas digitais, com o uso de modelos virtuais. Isto inclui desde a modelagem bi e tridimensional, processos de simulação, dimensionamento, otimização, prototipagem virtual até o projeto executivo. O uso de artefatos digitais que auxiliam o projeto permite ao designer a possibilidade de visualizar os modelos virtuais e prever problemas e interferências que, de outra forma, dificilmente seriam detectados antes que se construísse algum modelo físico do objeto. Neste caso, um protótipo físico só é construído quando o projeto já está em um estágio avançado, servindo para testes e aperfeiçoamentos.

Os sistemas computacionais interativos se tornam cada vez mais presentes no cotidiano dos designers. Através da interface gráfica do sistema, ocorre o processo de interação homem-computador, neste processo, a usabilidade da interface é o fator determinante no nível de satisfação, na eficácia e na eficiência com que os designers executam as tarefas típicas do sistema no processo de desenvolvimento de produto.

Segundo Teixeira e*t al.* (2008a), as ferramentas CAD que oferecem recursos tridimensionais (3D) permitiram substituir os ensaios físicos por simulações virtuais, transformando o processo de projeto. Conforme Teixeira *et al*. (2008b), a possibilidade de realizar todas as etapas de um projeto de uma forma totalmente virtual traz diversas vantagens, como a redução de custos e do tempo de desenvolvimento e, principalmente, a otimização do projeto em si.

É neste contexto que a plataforma de desenvolvimento T-CADE se insere. O software T-CADE foi concebido como uma plataforma de desenvolvimento para o design virtual de produtos com algoritmos para a modelagem paramétrica de superfícies tridimensionais e para a geração de malhas estruturadas e não estruturadas, para análise e simulação por elementos finitos (TEIXEIRA, 2004; TEIXEIRA e CREUS, 2003 e 2008).

Esta pesquisa é parte de um projeto coordenado pelo grupo Virtual Design da Universidade Federal do Rio Grande do Sul (UFRGS) que teve início em 2004, com a criação

da Plataforma de Desenvolvimento T-CADE (TEIXEIRA, 2004). Um dos objetivos deste projeto é integrar os esforços de pesquisa dos mestrandos do programa de pós-graduação em Design da UFRGS em torno de um objetivo comum: o Desenvolvimento da Plataforma T-CADE, acrescentando assim uma significação prática ao objeto de pesquisa, além de desenvolver tecnologia de ponta na área do design virtual.

1.1 DESCRIÇÃO DAS OCORRÊNCIAS OBJETIVAS

O T-CADE foi implementado com programação orientada a objetos na linguagem Delphi e sua interface foi criada a partir da utilização de algoritmos de visualização próprios. Estas rotinas foram desenvolvidas com o objetivo específico de permitir ao usuário a seleção e manipulação dos objetos tridimensionais da forma mais simples e direta possível. Assim, os procedimentos de interseção de superfícies e geração de malhas, alvo da pesquisa de doutorado do autor do software (TEIXEIRA, 2004), poderiam ser comprovados. Esta interface original serviu aos seus propósitos naquele determinado momento, resolvendo bem os problemas até então apresentados, mas tem limitado o desenvolvimento e as possibilidades de criação de novas ferramentas em função da complexidade das operações gráficas exigidas. A Figura 1 mostra a interface original do programa T-CADE com o modo de visualização "Aramado + Shade", evidencia-se neste caso a área de trabalho de interface (a parte com fundo branco onde o objeto é mostrado) que é o objeto desta pesquisa.

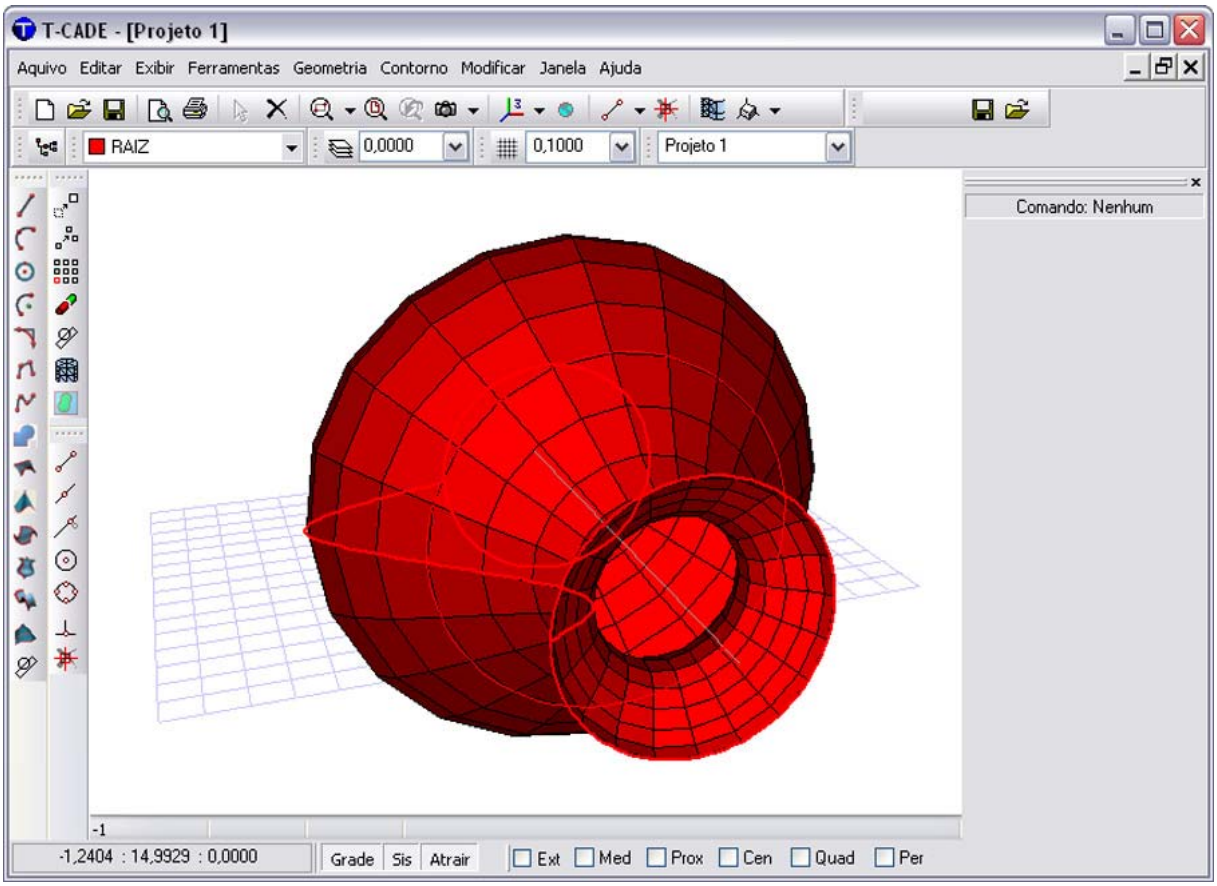


Figura 1: Superfície de Revolução no T-CADE.
Fonte: O Autor.

A interface original não permite a seleção de objetos clicando sobre a superfície do objeto. A seleção só acontece através das linhas de construção deste objeto. O ideal, neste caso, seria selecionar o objeto através de sua superfície e, ainda, reconhecer as coordenadas paramétricas da parte do objeto em que o cursor está apontando (coordenadas em 3D do ponto na superfície). Com a melhoria da seleção dos objetos, novas ferramentas de edição e manipulação de objetos poderiam ser implementadas. Além da seleção unitária (um objeto de cada vez), é necessário também que se possa selecionar um grupo de objetos de uma única vez, utilizando uma janela de seleção.

O desempenho gráfico é outro limitante. Nos casos em que existem muitos polígonos representados na tela, o nível de interatividade com o usuário cai drasticamente quando o ponto de vista da cena é movimentado, mostrando a imagem na tela de modo intermitente, podendo parecer que o programa não está respondendo por um curto período de tempo.

Os modos de visualização da interface original também são limitados, não suportando as opções de transparência e suavização da superfície, entre outras, o que poderia melhorar a representação da superfície e facilitar o entendimento da cena tridimensional.

Sendo assim, percebe-se a necessidade de melhorar esta interface para que o software permita seleção e manipulação de objetos 3D com mais eficiência e interatividade, permitindo assim que novas ferramentas de manipulação, edição e criação de objetos 3D sejam desenvolvidas. A criação destas ferramentas já é possível em termos conceituais e em termos de estrutura de dados, mas ficam ainda limitadas em termos práticos em função do desempenho gráfico da interface original (velocidade de atualização da tela para visualização de um número muito grande de polígonos) e pela interatividade (reconhecimento e seleção de objetos na tela). Em função destas limitações da interface original, a usabilidade do artefato digital em questão também fica comprometida.

1.2 DEMARCAÇÃO DO NÍVEL DE INVESTIGAÇÃO DO FENÔMENO

Quando se trata da interface de um artefato digital, são muitos os aspectos que podem ser estudados em termos de interação com o usuário e muitos os enfoques de abordagem para os problemas de usabilidade. Sendo assim, é preciso definir o que, na interface, é o objeto de estudo deste trabalho. A interface do software pode ser considerada como a camada (barra de ferramentas, menus, ícones, janelas, caixas de diálogo, entre outros) que permite estabelecer uma comunicação com o usuário para receber ou fornecer instruções (dados). Esta pesquisa estará limitada à interface de interação tridimensional que lida diretamente com os objetos 3D (área de trabalho 3D) e às rotinas necessárias à visualização e manipulação destes objetos na plataforma T-CADE. Assim, as barras de ferramentas, menus, ícones, caixas de diálogos e os arquivos de ajuda do programa, normalmente considerados no contexto mais amplo da usabilidade, não fazem parte do escopo deste trabalho.

1.3 PROBLEMA DE PESQUISA

Como melhorar a usabilidade de uma interface de interação tridimensional em softwares para desenvolvimento de produtos a partir da melhoria do desempenho gráfico e da interatividade?

1.4 HIPÓTESE BÁSICA

O uso de uma biblioteca gráfica e a criação de ferramentas gráficas para visualização, seleção e manipulação de objetos melhoram o desempenho gráfico e a interatividade, melhorando, consequentemente, a usabilidade do software.

1.5 VARIÁVEIS

- A usabilidade da interface 3D é a variável dependente.
- O desempenho gráfico da interface é uma das variáveis independentes e será manipulada a partir da implementação de uma biblioteca gráfica 3D.
- A interatividade é a segunda variável independente e será manipulada através da criação de ferramentas gráficas para visualização, seleção e manipulação de objetos 3D.

1.6 OBJETIVOS

1.6.1 OBJETIVO GERAL

Desenvolver um protótipo de uma interface gráfica para interação tridimensional que permita a manipulação de objetos e a navegação tridimensional com bom desempenho gráfico, interatividade e usabilidade.

1.6.2 OBJETIVOS ESPECÍFICOS

- Estudar os conceitos relativos à interação homem-computador sob o ponto de vista da usabilidade;
- Estudar conceitos relativos à engenharia de software;
- Pesquisar referencial teórico sobre as bibliotecas gráficas disponíveis;
- Estudar conceitos relativos ao processo de desenvolvimento de produtos;
- Projetar a interface segundo os conceitos estudados;
- Integrar a biblioteca gráfica selecionada ao software T-CADE.
- Investigar as ferramentas e técnicas mais utilizadas em termos de interação 3D, incluindo a visualização 3D, a manipulação e seleção de objetos virtuais 3D a partir de dispositivos 2D como *mouse* e monitor;
- Criar duas ferramentas gráficas para controle da visualização da cena;
- Criar condições de interatividade para que o usuário manipule os objetos diretamente com o uso do mouse.
- Desenvolver um protótipo funcional da interface proposta.
- Identificar as melhorias em relação à situação atual no que diz respeito à usabilidade, interatividade e desempenho gráfico.

1.7 JUSTIFICATIVA

Segundo Teixeira e*t al.* (2008a), atualmente, os recursos tecnológicos são utilizados com o objetivo de melhorar o processo de desenvolvimento de produtos através da sua virtualização. Os ambientes virtuais se originaram da convergência dos avanços tecnológicos da eletrônica digital dos computadores com os da tecnologia de apresentação visual. O sucesso de

programas de computadores ou sistemas computacionais para design de produto, raramente depende apenas do desempenho dos computadores e dos sistemas operacionais. Os programas devem, acima de tudo, atender as expectativas dos usuários e suas necessidades nas tarefas de projeto.

A grande maioria das soluções CAD/CAE disponíveis hoje no mercado são ferramentas proprietárias e, comumente, pertencentes a empresas multinacionais estrangeiras. Além do custo elevado destes sistemas, apenas seus criadores têm acesso ao seu código fonte, sendo que qualquer mudança, adaptação ou inovação depende do interesse comercial destas empresas. Neste sentido, é importante que um país tenha o domínio da tecnologia que necessita e, principalmente neste caso, do código fonte de suas aplicações.

De acordo com Teixeira e*t al.* (2008b), apesar das vantagens obtidas a partir do Design Virtual, o seu uso sistemático ainda é restrito às grandes corporações, principalmente àquelas que enfrentam a acirrada concorrência do mercado globalizado, como a indústria automobilística, naval e aeroespacial. Este tipo de indústria faz grandes investimentos em pesquisa tecnológica voltada à inovação, procurando incorporar tecnologia em seus produtos, para agregar valor aos mesmos. Neste sentido, as vantagens proporcionadas pelo Design Virtual no desenvolvimento de produtos garantem a competitividade e a permanência das corporações no mercado. Adotar o conceito de Design Virtual significa a condição *cinequanon* para a otimização do processo de desenvolvimento de produtos.

O T-CADE vem sendo desenvolvido dentro desta perspectiva, como uma alternativa de sistemas CAD/CAE para o design de produtos com a vantagem de ser um sistema desenvolvido no Brasil. Este sistema facilita a geração de alternativas na fase conceitual de desenvolvimento de produtos de forma rápida e interativa. O domínio do código fonte permite que o T-CADE seja adaptado, ou direcionado para a solução de problemas específicos de projeto mostrando um grande potencial para a indústria nacional.

Este aplicativo 3D foi criado pelo grupo de pesquisa Virtual Design com o intuito de ser uma plataforma de desenvolvimento onde as diferentes pesquisas dentro do campo de design e tecnologia pudessem ser aplicadas. Assim, ao invés de cada pesquisa começar e encerrar-se em si mesma, os trabalhos de pesquisas seriam direcionados a um objetivo comum e cada pequeno esforço colocado no projeto estaria ajudando a alcançar este objetivo, a criação de uma Plataforma 3D para o desenvolvimento de produto.

Atualmente, o T-CADE já comporta a tecnologia necessária para a criação de novas ferramentas que ajudariam no processo de modelagem virtual. Infelizmente, a implementação destas ferramentas ainda está impedida pela interface original que não suporta determinadas operações gráficas. O desenvolvimento de uma nova interface gráfica permitirá a criação de novas ferramentas de modelagem e de edição de objetos que dependem da correta visualização, interatividade (*feedback* imediato) e da manipulação e seleção de objetos tridimensionais.

O T-CADE é parte de um projeto que propõe o desenvolvimento de um sistema computacional para o design virtual de produtos, o Projeto Virtus. O Virtus será um sistema modular que deverá contemplar as principais etapas do projeto de produto em um ambiente virtual. O sistema terá um módulo principal *on-line* que fará o gerenciamento do processo de projeto e módulos complementares, como: o módulo de projeto conceitual, de detalhamento, de simulação, de documentação e o módulo de biblioteca de componentes. O T-CADE é a base deste módulo de projeto conceitual que contempla a modelagem tridimensional de forma amigável e intuitiva, permitindo testar novos conceitos formais rapidamente.

Neste sentido, este trabalho tem importância fundamental no projeto de desenvolvimento do T-CADE. Para que esta plataforma possa continuar a ser desenvolvida, é imprescindível que a interface com o usuário não apenas tenha um bom desempenho como atenda as expectativas dos usuários.

Sendo assim, a plataforma T-CADE se encontra atualmente com seu futuro desenvolvimento comprometido pela usabilidade, desempenho e interatividade da interface original. A melhoria da interface gráfica do T-CADE, em termos de usabilidade, pode ser alcançada através de duas intervenções específicas. Primeiro, através da criação de ferramentas gráficas para manipulação de objetos e controle da visualização da cena para a melhoria da interatividade. Segundo, pela melhoria do desempenho gráfico quanto à velocidade de atualização da tela através do uso de uma biblioteca gráfica. Isto permite que um maior número de polígonos seja visível e manipulável em tempo real, possibilitando uma correta representação de superfícies, resolvendo problemas complexos de visibilidade, transparência, sombreamento, seleção de objetos e iluminação. Finalmente, uma interface gráfica com tais capacidades permite a criação de novas ferramentas e do contínuo desenvolvimento no software T-CADE, ampliando significativamente o potencial desta plataforma.

2 FUNDAMENTAÇÃO TEÓRICA

Como a pesquisa em design ainda é recente, especialmente na área de desenvolvimento de artefatos digitais, muito do referencial teórico desta pesquisa baseia-se nos conceitos já estabelecidos na área de conhecimento da tecnologia da informação. Parte deste referencial se encontra na área de interação homem-computador (IHC) com ênfase na engenharia de usabilidade e outra parte na área de computação gráfica (CG), além dos conceitos de engenharia de software e processo de desenvolvimento de produtos (PDP) que são fundamentais para o projeto de sistemas computacionais segundo métodos controlados e eficazes.

Em linhas gerais, a área de interação homem-computador (IHC) investiga o design, avaliação e implementação de sistemas computacionais interativos para uso humano, juntamente com os fenômenos associados a este uso (HEWETT, *et al.* 1992). O estudo de sistemas interativos tem como foco o usuário e sua experiência no uso do artefato digital. A interface deve tornar esta experiência o mais fácil, prática e agradável possível. Neste sentido, é importante buscar uma fundamentação na área de engenharia de usabilidade, para que o projeto da interface seja adequado as necessidades dos usuários.

Do ponto de vista técnico, a criação de uma interface 3D não é uma tarefa trivial, requer um planejamento prévio e uma investigação dos requisitos de projeto para que sua implementação seja bem sucedida. A área de engenharia de software vem auxiliar no projeto do sistema e decidir a melhor abordagem para a solução dos problemas encontrados.

Tanto a engenharia de usabilidade como a engenharia de software utilizam-se de métodos para agrupar requerimentos, desenvolver e testar protótipos, avaliar projetos alternativos, analisar problemas, propor soluções e fazer testes. A diferença entre elas está, basicamente, nos objetivos. Assim como a engenharia de software busca uma maior qualidade no desenvolvimento do software, com um foco em sua estrutura, implementação e funcionalidade, a engenharia de usabilidade também busca esta qualidade com foco na experiência do usuário. Tratam-se, portanto, de abordagens metodológicas semelhantes que objetivam a concepção de um produto com o melhor design possível, tanto do ponto de vista técnico como ergonômico.

O processo de desenvolvimento de produtos, assim como a engenharia de software vem contribuir para a tomada de decisões no projeto do produto, utilizando ferramentas e técnicas que organizam e facilitam o processo de projeto.

Genericamente, a computação gráfica é a disciplina que trata das técnicas e dos métodos computacionais que convertem dados em imagens para dispositivos gráficos. As ferramentas de visualização e manipulação propostas neste trabalho baseiam-se tanto na área de IHC como na computação gráfica.

A criação de uma interface para interação tridimensional requer ainda uma investigação sobre as bibliotecas gráficas disponíveis, quais suas principais funções, vantagens e desvantagens para sua utilização neste trabalho, assim como algumas

considerações sobre o problema da interação 3D, em termos de visualização e manipulação de objetos, a partir de dispositivos bidimensionais (2D), como mouse e monitor.

2.1 PROCESSO DE DESENVOLVIMENTO DE PRODUTO

O desenvolvimento de novos produtos é uma necessidade na economia globalizada, onde consumidores ávidos por novos e melhores produtos e serviços tornam-se cada vez mais exigentes. Isto exige dos designers criatividade e inovação para atender a estas demandas. A atividade de projeto tem especial importância diante deste cenário, pois é esta o vetor da inovação. Para realizarem as atividades de projeto de produto, os designers necessitam de diversas competências, dado o perfil inter e multidisciplinar destas atividades, principalmente no que tange ao trabalho em equipe, comunicação e criatividade.

Assim como a engenharia de software, o processo de desenvolvimento de produto (PDP) pode guiar o designer no processo de projeto de forma sistemática, auxiliando na tomada de decisões de projeto e amparando a criatividade com ferramentas e técnicas quantitativas e qualitativas. Nesse contexto, o desenvolvimento de produtos se insere como um dos processos que potencializa a competitividade das organizações, sendo sua eficiência e eficácia diretamente relacionadas às competências dos profissionais envolvidos.

Neste trabalho será adotada a definição de Rozenfeld *et al*. (2006) para o processo de desenvolvimento de produto, o PDP consiste em um conjunto de atividades por meio das quais se busca, a partir das necessidades do mercado e das possibilidades e restrições tecnológicas, e considerando as estratégias competitivas e de produto da empresa, chegar as especificações de projeto de um produto e de seu processo de produção, para que a manufatura seja capaz de produzi-lo.

O modelo de PDP apresentado, proposto por Rozenfeld *et al*. (2006), é dividido em macrofases, subdivididas em fases e atividades. Conforme apresentado na Figura 2, as três macrofases são: pré-desenvolvimento, desenvolvimento e pós-desenvolvimento.

As macrofases de pré e pós-desenvolvimento são mais genéricas. A macrofase de desenvolvimento enfatiza os aspectos tecnológicos correspondentes à definição do produto em si, suas características e forma de produção. Portanto, tais atividades são dependentes da tecnologia envolvida no produto.

A macrofase de pré-desenvolvimento deve garantir que o direcionamento estratégico definido a priori pela empresa no planejamento estratégico da corporação, as idéias de todos os atores internos e externos envolvidos com os produtos, e as oportunidades e restrições sejam sistematicamente mapeados e transformados em um conjunto de projetos bem definidos, isto é, o portfólio dos projetos que deverão ser desenvolvidos.

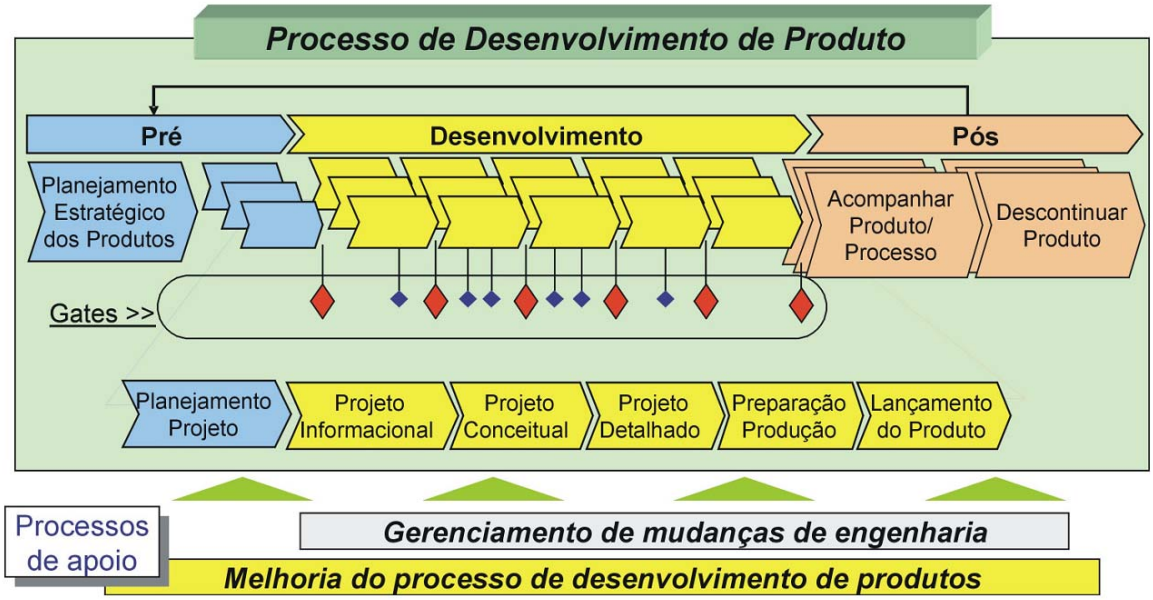


Figura 2: Modelo de processo de desenvolvimento de produtos.
Fonte: (ROZENFELD et al., 2006)

A macrofase de desenvolvimento está integrada às macrofases de pré e pós-desenvolvimento. Nesta etapa, ressaltar-se a importância das fases iniciais do desenvolvimento. No início do projeto, o grau de incerteza é grande, porém, é neste momento que são realizadas as escolhas de soluções de projeto (materiais, conceitos, processos de fabricação, etc), que determinam a maior parte do custo final do produto.

Em razão do grau de incerteza inicial, ocorrem modificações de produto nas fases subseqüentes do desenvolvimento, quando informações mais precisas estão disponíveis. O custo das modificações é cada vez mais elevado, conforme avançam as fases do desenvolvimento do produto.

A macrofase de pós-desenvolvimento compreende o acompanhamento sistemático e a documentação correspondente das melhorias de produto ocorridas durante o seu ciclo de vida. Ela compreende ainda a retirada sistemática do produto do mercado e, finalmente, uma avaliação de todo o ciclo de vida do produto, para que as experiências contrapostas ao que foi planejado anteriormente sirvam de referência a desenvolvimentos futuros.

Cada uma dessas macrofases é subdivida em fases. O que determina uma fase é a entrega de um conjunto de resultados (*deliverables*) que, juntos, determinam um novo patamar de evolução do projeto de desenvolvimento.

É na macrofase de desenvolvimento que ocorre o projeto do produto. Segundo Back *et al.*(2008), o projeto constitui-se num processo de transformação de informações, onde o projetista modifica estas informações, inicialmente na forma de problemas, para uma possível solução, isto tudo ocorrendo sob influência de um determinado ambiente ou meio. Segundo o autor, o projeto de produto em uma abordagem sistemática pode ser dividido em quatro etapas: projeto informacional, projeto conceitual, projeto preliminar e projeto detalhado.

A etapa de projeto informacional consiste em série de procedimentos de pesquisa e de coleta de dados, a fim de determinar o problema de projeto, finalizando com a identificação dos requisitos e determinação das especificações do projeto.

O projeto conceitual é a fase onde se elabora a concepção do projeto a partir de uma série de ciclos de análise e síntese, onde a criatividade é assistida por procedimentos sistemáticos, resultando em uma concepção refinada a partir de conceitos gerados durante um processo iterativo.

O projeto preliminar define a configuração final do produto e a sua viabilidade técnica e econômica. Nesta fase a idéia é satisfazer as funções do produto configurando-se a forma dos componentes, leiaute, processos de fabricação e materiais apropriados para a concepção selecionada.

Finalmente, o projeto detalhado é a última etapa onde são definidas e detalhadas as especificações dos componentes, dos planos de manufatura e a documentação do projeto do produto. Nesta fase finaliza-se o projeto preliminar, estabelecendo-se as descrições definitivas para as soluções dos elementos construtivos, formas, dimensões, acabamentos superficiais, materiais e processos de fabricação.

Todas as etapas do projeto são importantes. No entanto, a etapa do projeto conceitual determina cerca de 85% do custo final do produto (BARTON *et al*., 2001 e BACK *et al*., 2008), o que demonstra a importância das decisões tomadas nesta fase. Portanto, o projeto conceitual deve ser organizado de forma a buscar a otimização do processo para que o conceito de produto resultante seja a melhor solução possível para os requisitos.

Baxter (1998) cita uma maior atenção nos estágios iniciais do desenvolvimento do produto. Os estágios iniciais são os mais importantes no processo de desenvolvimento de novos produtos. Quando o projeto conceitual estiver pronto deve-se definir o seu mercado potencial, seus princípios operacionais e os principais aspectos técnicos. Um grande número de decisões terá sido tomado e um considerável volume de recursos financeiros alocados. Contudo, os gastos com o desenvolvimento ainda são relativamente pequenos – a pesquisa ocorreu só no papel e os trabalhos de projeto consistem de desenhos e modelos baratos. A introdução de mudanças em etapas posteriores, como na fase de engenharia de produção, pode implicar em refazer matrizes de elevadíssimos custos.

O mesmo autor cita que, “a chave do sucesso no desenvolvimento do produto consiste então, em investir mais tempo e talento durante os estágios iniciais, quando custa pouco”, ou seja, nas fases de projeto informacional e conceitual, o tempo deve ser bem aproveitado, para evitar retrabalhos (BAXTER, 1998).

A seguir serão detalhadas as etapas do projeto informacional e do projeto conceitual.

2.1.1 O PROJETO INFORMACIONAL

O projeto informacional é a primeira etapa na fase de desenvolvimento de produto, que corresponde a fase de clarificação ou identificação da tarefa de projeto; é a fase responsável

pela captura e tratamento da informação sobre o problema onde a equipe de projeto precisa entendê-lo e especificá-lo mais detalhadamente (ROZENFELD *et al*., 2006).

Nessa fase, evolui-se das necessidades dos clientes (declarações diretas, geralmente em linguagem subjetiva), passando pelos requisitos dos clientes (necessidades expressas em linguagem de engenharia), até as especificações do projeto, que compõem uma lista de objetivos a que o produto a ser projetado deve atender. A fase do projeto informacional envolve, basicamente, a identificação do problema, a geração de ideias preliminares, a definição de requisitos e a especificação do projeto.

O conjunto de informações elaboradas durante o projeto informacional deve refletir as características que o sistema deverá ter para atender às necessidades dos usuários finais e dos desenvolvedores. A precisão e correção destas informações afetarão a qualidade de todos os projetos subsequentes no processo de desenvolvimento. Uma das metodologias mais bem aceitas para a garantia de qualidade no processo de projeto é o QFD (*Quality Function Deployment* ou Desdobramento da Função Qualidade), também conhecido como Casa da Qualidade. A definição do QFD assumida atualmente foi criada em 1972, com aplicações bem sucedidas nas empresas Mitsubishi e Toyota, sendo rapidamente difundido no Japão. Convém, no entanto, ressaltar que a corrente da Toyota (normalmente a utilizada nas indústrias) constitui-se de uma tabela bidimensional denominada matriz “O que/Como”, não expressando a totalidade do QFD (OHFUJI, 1997).

Conforme Back *et al. (*2008) o QFD é eficiente para transladar vontades de clientes (natureza abstrata) em dados concretos de projeto. Recentemente, várias empresas, entre elas IBM e HP, têm adaptado o método QFD, que permite planejar a qualidade de produtos e serviços, para o desenvolvimento de Software. Esta adaptação é denominada de Desdobramento da Função Qualidade do Software (SQFD - *Software Quality Function Deployment*).

O SQFD enfatiza que a Garantia da Qualidade de Software (SQA – *Software Quality Assurance*) deve ser iniciada a partir da fase de especificação de requisitos, onde se procura ouvir as necessidades dos clientes (qualidades intangíveis) para traduzi-las em características técnicas (mensuráveis) do produto (HAAG, 1996). Essa abordagem está relacionada ao desenvolvimento de software centrado no usuário, onde a participação dos usuários do sistema é um fator imprescindível para se determinar a usabilidade do sistema, além de outros fatores de qualidade, como a correção dos requisitos de projeto.

Um requisito de projeto compreende uma característica ou funcionalidade que o sistema deve possuir para atender uma necessidade do usuário, qualificado por condições mensuráveis e limitado por restrições.

O método QFD permite definir e quantificar os requisitos críticos do cliente, base para um bom projeto de software. Sendo assim, esta é uma boa ferramenta para aumentar a probabilidade de satisfazer às necessidades dos usuários e efetuar eficientemente as conversões que transformam as exigências do cliente para código executável em máquina.

Basicamente, o QFD representa uma diminuição de problemas no início da produção, menos mudanças no projeto, e encurta os ciclos de desenvolvimento do artefato. Com isso, é consequente o aumento da produtividade e a redução de custos. A principal vantagem em longo prazo é a satisfação do usuário.

2.1.1.1 QFD – DESDOBRAMENTO DA FUNÇÃO QUALIDADE

O QFD surgiu por meio da aplicação e desenvolvimento dos contemporâneos conceitos da gerência da qualidade no Japão, criado principalmente pelos professores Mizuno e Akao (1994). Existem diversas definições e abordagens do QFD, uma vez que a metodologia QFD oferece um amplo espectro de aplicações. Neste trabalho será utilizada uma simplificação, focando nos aspectos mais importantes para a definição de requisitos de projeto.

A primeira etapa (1) da construção da casa da qualidade é a determinação das necessidades dos usuários (NC ou os "O quê", como é usualmente conhecido). No caso do *software*, essas são obtidas junto aos usuários de todos os tipos, que podem ser iniciantes, intermediários ou avançados. A seguir, determina-se o grau de importância (2) que os usuários dão a cada necessidade. A partir disto (3), busca-se a determinação dos requisitos da qualidade (RQ ou os "Como"), que se constitui, basicamente, por transformar as necessidades coletadas diretamente dos usuários (geralmente com características abstratas e de difícil mensuração) em linguagem técnica. Nessa etapa, tem-se um número expressivo de necessidades dos clientes e requisitos da qualidade, sendo que alguns estão mais inter-relacionados do que outros. A Figura 3 mostra como fica a matriz simplificada da casa da qualidade. Um exemplo mais detalhado da utilização da matriz da casa da qualidade é apresentado na Figura 17 (pág. 74).

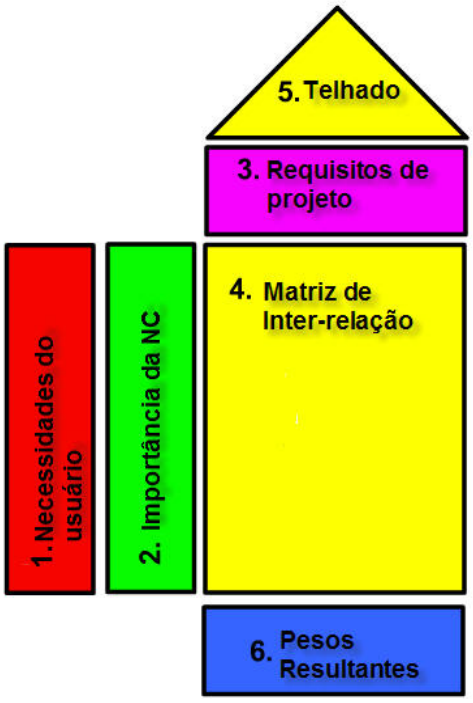


Figura 3: Matriz da Casa da Qualidade.
Fonte: O Autor.

Através de técnicas coletivas, como o *brainstorming*, por exemplo, uma equipe multidisciplinar objetiva indicar, de forma qualitativa, o quanto cada RQ se relaciona com cada NC (4), utilizando para isso convenções de sinais ou uma escala de valores, por exemplo: 1, 3 e 9 para indicar pouca, média ou alta relação. No Telhado da casa da qualidade (5) encontra-se a matriz de correlações. Esta matriz cruza as características de qualidade entre si, sempre duas a duas, permitindo identificar como elas se relacionam. Estas relações podem ser de apoio mútuo - quando o desempenho favorável de uma característica ajuda o desempenho favorável da outra característica, ou de conflito - quando o desempenho favorável de uma característica prejudica o desempenho favorável da outra característica. Finalmente a partir da matriz de inter-relações entre RQ's e NC's definem-se os pesos absolutos e relativos de cada requisito (6) ou a característica do projeto no alcance das necessidades dos usuários.

Desta forma, a utilização do QFD se mostra como uma alternativa eficiente para ouvir o que dizem os clientes, possibilitando descobrir exatamente o que eles querem, permitindo também determinar a melhor maneira de satisfazer suas necessidades, criando um plano de ação que busca garantir que os produtos desenvolvidos ou aperfeiçoados tenham as características exigidas pelos clientes.

2.1.2 O PROJETO CONCEITUAL

O projeto conceitual, por sua importância no custo final do produto, deve seguir procedimentos metodológicos que garantam o desenvolvimento otimizado das soluções que atendem aos requisitos e restrições do projeto. Esta etapa consiste em identificar os problemas essenciais, o estabelecimento da estrutura de funções, a busca e combinação de princípios de soluções, a obtenção de variantes de concepções, sua concretização e finalmente a avaliação das soluções segundo critérios técnicos e econômicos.

O projeto conceitual trata dos princípios do processo de projeto, mas é inevitável pensar também em alguns aspectos da configuração e seus componentes. Isso é essencial para se provar que o conceito selecionado é viável. O projeto conceitual é concluído quando se chega a um conjunto de princípios funcionais e formais para o produto como um todo, satisfazendo as especificações de projeto.

O conceito deve evidenciar como o produto atenderá às necessidades dos consumidores e se diferenciará dos concorrentes ou similares. O projeto da configuração começa com a divisão dos componentes para a fabricação. Ainda no nível conceitual, é explorada uma variedade de formas e funções para cada componente, selecionando sistematicamente a melhor.

Aqui, o processo criativo deve ser orientado e administrado para que não haja desvio de foco, evitando desperdício de tempo e recursos. É possível dividir o processo em etapas e, dependendo do autor, pode haver variação no número e na denominação das mesmas, mas, em geral, há correspondência do processo.

A partir de diversas referências (ASIMOV, 1962; ULRICH e EPINGER, 2007; PAHL *et al.*, 2005; BACK *et al.*, 2008) o projeto conceitual pode ser organizado nas seguintes etapas:

- Definição da estrutura funcional, onde são definidas a função global e as subfunções a partir dos requisitos e restrições;
- Definição de princípios de soluções através de pesquisa e de processos criativos;
- Geração de conceitos a partir da combinação dos princípios de solução de forma criativa e sistemática;
- Seleção de conceitos a partir de análises qualitativas e quantitativas;
- Elaboração do conceito final a partir da avaliação comparada dos conceitos gerados.

O projeto conceitual tem a criatividade como um motor, mas tão importante quanto a criatividade são as decisões tomadas nesta fase, pois estas determinam a eficiência do processo. Assim, ferramentas de apoio à tomada de decisão são fundamentais para garantir as escolhas corretas e coerentes durante o desenvolvimento do projeto. Em cada uma das etapas, há ferramentas analíticas que podem orientar a tomada de decisão. A seguir são apresentadas mais detalhadamente as etapas do projeto conceitual.

2.1.2.1 DEFINIÇÃO DA ESTRUTURA FUNCIONAL

A definição da estrutura funcional, também conhecida como síntese funcional (KOLLER *apud* BACK *et al.*,2008; PAHL e BEITZ, 1996; BAXTER, 1998), é uma poderosa ferramenta que orienta o processo de projeto conceitual. Nesta fase, a partir das especificações, dos requisitos e das restrições de projeto, são definidas a função global do produto e as funções secundárias ou subfunções em uma estrutura hierárquica, cuja topologia descreve o funcionamento do produto.

A função global do produto ou sistema é aquela que é a razão primordial que justifica a sua existência. Por exemplo, a função global de um automóvel é o transporte de pessoas com segurança e autonomia. No entanto, dificilmente pode-se encontrar uma solução direta para a função global quando esta é complexa. Assim, a função é subdividida em funções mais simples (subfunções), através de uma abordagem *top down*, até que seja possível estabelecer princípios de solução para as subfunções individuais.

Este conceito de desdobramento também é utilizado no projeto para modularidade, que é um enfoque de projeto de produto em que um projeto é dividido em unidades menores, ou módulos, cada um dos quais podendo ser desenvolvido, testado e concluído de forma independente, e depois integrado ao produto final. Segundo Arnheiter e Harren (2006) o desenvolvimento e padronização de partes intercambiáveis foi o precursor da modularidade e era utilizado na indústria bélica há alguns séculos, mas o projeto modular surgiu como vantagem competitiva na indústria de computadores apenas na década de 1960, demonstrando grande importância no processo de desenvolvimento de produto.

Esta abordagem pode ser usada para simplificar e facilitar o projeto do sistema de produção ou produtos. O desenvolvimento de produtos baseado em uma estrutura modular permite que seus módulos possam ser adaptados e reutilizados sob demanda em diferentes

projetos. É comum, na indústria automobilística, oferecer uma variedade de modelos de automóveis pela combinação de diversos subsistemas, por exemplo: direção hidráulica, ar condicionado e *air bag*.

Este conceito de modularidade pode ser relacionado com o conceito da inter-relação entre sistemas da engenharia de software, onde um sistema maior é divido em sistemas menores diminuindo sua complexidade (pág. 33). A programação orientada a objetos, utilizada neste trabalho (abordada em mais detalhes no item 2.4.3 da engenharia de software), também é baseada na utilização de módulos (unidades e classes) para minimizar a complexidade da função “programar” e facilitar a reutilização dos módulos de código.

Segundo Back *et al.* (2008), para produtos modulares, é conveniente distinguir dois conceitos: módulos funcionais e módulos construtivos. Um módulo funcional é uma subdivisão de uma função global (mais geral) em subfunções específicas. Um módulo construtivo é uma subdivisão física que pode conter um ou mais módulos funcionais.

Assim na definição da função global de um produto, a estrutura funcional encontrada não é, necessariamente, única, pois a definição das subfunções também envolve processos criativos, podendo resultar em diversas estruturas alternativas. Portanto, nesta fase como nas seguintes, há a necessidade de seleção de alternativas e de ferramentas analíticas para auxiliar a tomada de decisão.

Esta é uma etapa que depende fundamentalmente das informações disponíveis e da inter-relação entre as mesmas. Assim, a confiabilidade das informações é um fator determinante para o sucesso do processo. Por outro lado, as informações geradas nesta etapa são fundamentais para o sucesso da etapa seguinte que definirá os princípios de solução.

2.1.2.2 PRINCÍPIOS DE SOLUÇÃO

A partir da estrutura funcional e formal do produto, devem ser definidos os princípios de solução ou princípios funcionais utilizando técnicas de pesquisa, de análise de similares e de criatividade a fim de estabelecer as soluções para todas as subfunções e, assim, atender à função global do produto.

Os princípios de solução podem ser gerados a partir de técnicas de criatividade intuitivas e de técnicas sistemáticas, que podem ser utilizadas isoladamente ou em conjunto como ferramentas complementares. Um exemplo disto é o uso de uma matriz morfológica (técnica sistemática) para especificar princípios funcionais obtidos a partir de *brainstorm* ou técnicas de analogia (técnicas intuitivas). A Figura 4 apresenta a matriz morfológica com os princípios de solução onde a situação problema é a de levantar um veículo para realizar manutenção na parte inferior.

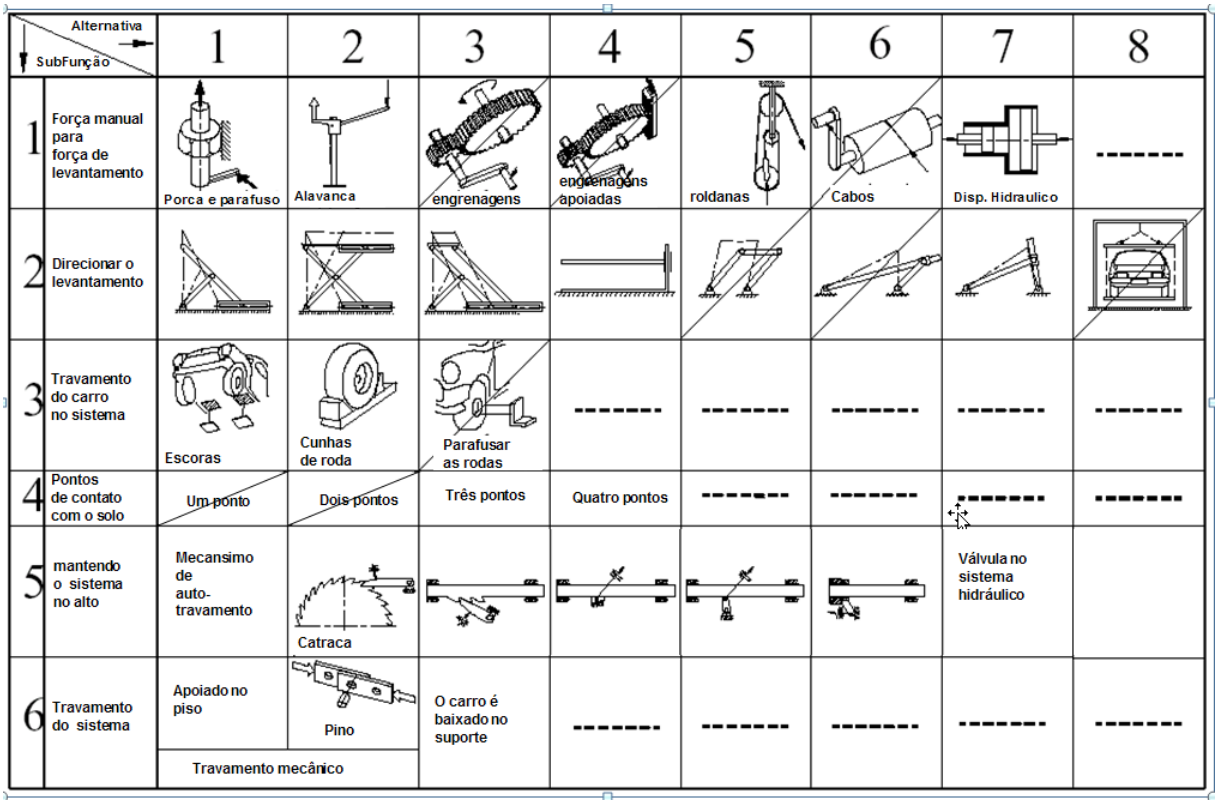


Figura 4: Matriz Morfológica
Fonte: Adaptado de NASERI e YEKTAEE, s.d.

A seguir são apresentadas algumas técnicas de criatividade para a geração de ideias e alternativas para a solução de problemas.

2.1.2.2.1 Técnicas de Criatividade

A criatividade é a habilidade de ter ideias novas e úteis para resolver o problema proposto ou de sugerir soluções para a concepção do produto (BACK *et al.*, 2008). As ideias ou soluções criativas devem ser únicas, úteis e simples. A criatividade não é um dom de alguns poucos escolhidos, mas sim uma habilidade inata de todo ser humano e que pode ser estimulada e guiada utilizando técnicas apropriadas.

Já foram catalogadas 105 técnicas para a geração de ideias (VAN GUNDY, *apud* BAXTER, 1998). Muitas destas são similares ou possuem pequenas variações entre si. Back *et al*. (2008) classifica estas técnicas em dois grupos: métodos intuitivos e métodos sistemáticos.

Dos métodos intuitivos, o mais conhecido é o brainstorm, de origem inglesa. Este método é realizado em sessões com um grupo de pessoas, onde um coordenador propõe um problema e os participantes dispõem de um tempo para a solução (BONSIEPE et al., 1984). As sugestões de solução devem ser registradas e as críticas às ideias propostas devem ser evitadas. Em uma etapa seguinte, o grupo seleciona as melhores ideias.

Este tipo de reunião pode ser feita em várias rodadas a fim de refinar as soluções encontradas que, no final, são atribuídas ao grupo e não apenas a um indivíduo. Existem

variações, como o método 635 e o *brainstorm* escrito. Este tipo de técnica pode ser utilizado em qualquer etapa do projeto e tem a vantagem de integrar os membros da equipe.

O método das analogias é outra técnica muito utilizada. Nesta técnica, buscam-se soluções a partir de analogias com elementos de outras áreas do conhecimento, fora do domínio do problema, como em outros tipos de produtos, na natureza, na literatura, no cinema. A biônica é uma forma importante de uso desta técnica, onde as soluções são inspiradas na fisiologia, estruturas e mecanismos dos seres vivos.

Podem ser citados, ainda, o método sintético (HOUIAS e VILLAR, *apud* BACK *et al*. 2008), que usa o princípio das analogias, porém mais elaborado; o método da listagem de atributos, onde o produto é descrito através de atributos que serão melhorados e modificados; o método MESCRAI (BAXTER, 1998) que consiste nas iniciais das palavras: modificar, substituir, combinar, rearranjar, adaptar e inverter, as quais constituem os princípios para tentar refinar um produto ou solução.

Os métodos sistemáticos seguem uma sequência lógica e estruturada de atividades que geram soluções alternativas para um determinado problema (BACK *et al.*, 2008). Soluções criativas podem ser obtidas a partir de combinação de ideias existentes ou não. Este é o princípio da matriz morfológica. A partir das funções e subfunções do produto, procuram-se soluções ou princípios funcionais das soluções. Aqui, podem ser utilizadas técnicas de pesquisa da área de conhecimento específica e técnicas intuitivas, como o *brainstorm*.

A combinação dos diversos princípios funcionais pode gerar diferentes concepções de produto. A análise de valor é uma técnica que busca otimizar o projeto em função dos custos das soluções levando-se em conta todo o ciclo de vida do produto. Nesta técnica, atribuem-se valores para as funções e utilizam-se ferramentas de criatividade para a redução dos custos nas soluções propostas.

O método dos princípios inventivos é outra técnica que procura sistematizar o processo de geração de soluções e foi baseado nos chamados princípios inventivos, os quais constituem princípios de soluções obtidos através de uma busca em mais de 1,5 milhões de patentes. O método é baseado em uma matriz de contradições e princípios inventivos, os quais recebem pesos variados a fim de encontrar a solução dos conflitos e a solução.

2.1.2.3 Geração de Conceitos

A geração dos conceitos é feita a partir da combinação das diferentes soluções, verificando a compatibilidade e a exequibilidade das mesmas. Cada combinação de soluções corresponde a uma concepção ou conceito do produto. O conjunto de todas as combinações corresponde às alternativas de conceito que atendem aos requisitos de projeto. A Figura 5 apresenta mesma matriz morfológica da Figura 4, porém com uma série de concepções geradas a partir das soluções propostas.

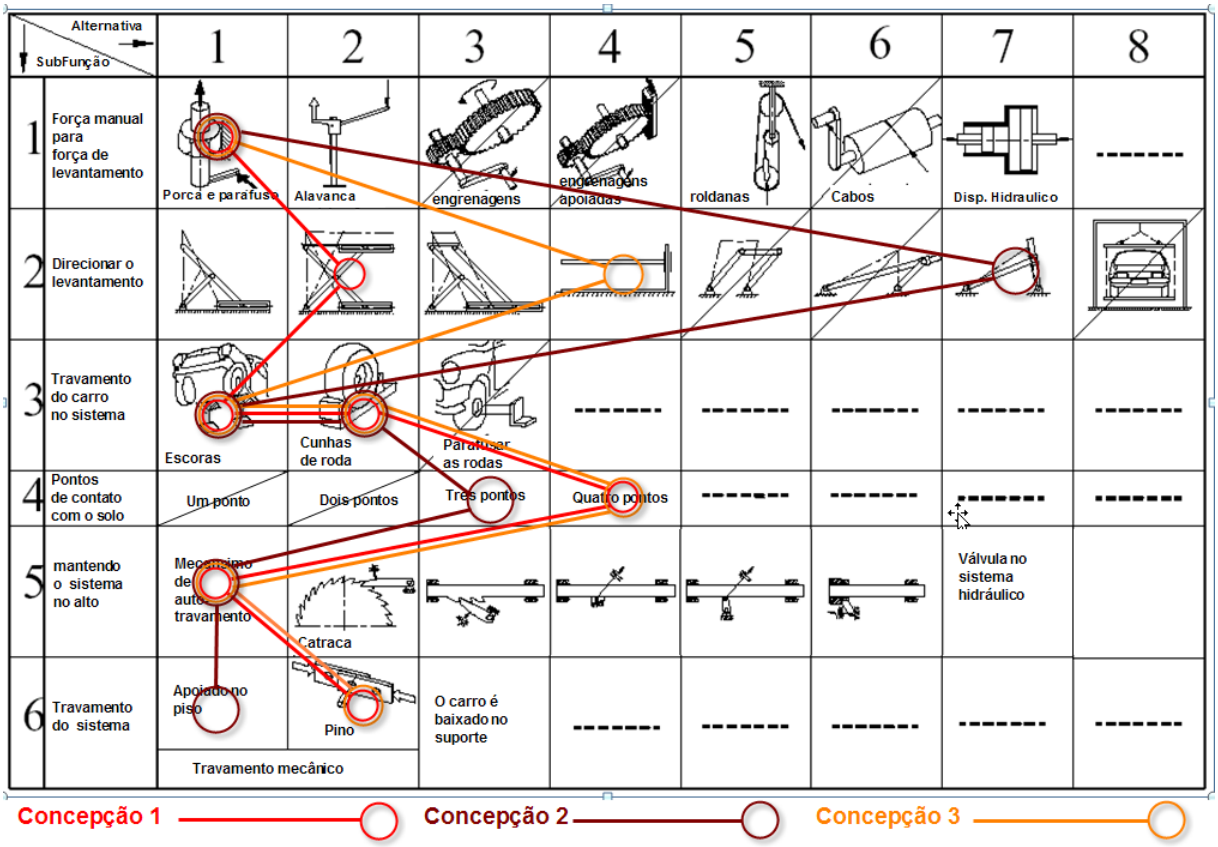


Figura 5: Geração de conceitos a partir dos princípios de solução.
Fonte: Adaptado de Naseri e Yektaee, s.d.

2.1.2.4 SELEÇÃO DOS CONCEITOS

A seleção dos conceitos gerados na etapa anterior é tão importante quanto a geração das alternativas de conceito, pois as decisões tomadas nesta etapa irão determinar a concepção do produto. A seleção dos conceitos deve levar em conta o atendimento aos requisitos de projeto e sua viabilidade técnica e econômica. A partir deste ponto, Back *et al.*(2008) sugere uma seleção em etapas.

Na primeira etapa, é feita uma triagem das concepções através da comparação com um produto de referência, com o objetivo de gerar um conjunto menor de concepções que tenham um potencial real de viabilidade. Para tanto, pode ser utilizado o método de Pugh (PUGH, 1990), onde em uma matriz de avaliação são atribuídos valores que relacionam atributos das concepções e do produto de referência (*Datum*).

É possível atribuir pesos aos atributos, mas, em geral, utilizam-se os mesmos pesos nesta triagem inicial. A avaliação de cada atributo pode ser: (+) se é considerado superior à referência, (-) se é considerado inferior à referência e (S) ou (0) se o atributo é considerado similar à referência. Os somatórios dos (+) e (-) definem um valor para cada conceito. Somente devem ser selecionadas aquelas concepções com valores (+) maiores que valores (-), o que significa que são superiores ao produto de referência. A Figura 6 apresenta um exemplo de uma matriz de Pugh para a seleção de perfis, onde apenas os conceitos 3, 5, 8 e 10 apresentam uma valoração positiva em relação ao produto de referência (7).

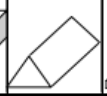
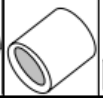
SOLUÇÕES											
CRITÉRIO	1	2	3	4	5	6	7	8	9	10	11
A	+	-	+	-	+	-	D	-	+	+	+
B	+	S	+	S	-	-		+	-	+	-
C	-	+	-	-	S	S	A	+	S	-	-
D	-	+	+	-	S	+		S	-	-	S
E	+	-	+	-	S	+	T	S	+	+	+
F	-	-	S	+	+	-		+	-	+	S
∑ (+)	3	2	4	1	2	2	U	3	2	4	2
∑ (-)	3	3	1	4	1	3		1	3	2	2
∑(S)	0	1	1	1	3	1	M		1	0	2

Figura 6: Exemplo de Matriz de Pugh
Fonte: (PUGH, 1990).

Na segunda etapa, com um conjunto reduzido de conceitos, os atributos ou critérios de comparação devem ser melhor explicitados e desdobrados em critérios mais detalhados e específicos, a fim de refinar o processo de comparação. Para uma avaliação mais aprofundada e metódica, Pahl e Beitz (1996) propõem que as variantes de solução factíveis devem ser comparadas antes de estabelecer uma concepção do produto, por meio da aplicação de critérios de avaliação e, na medida do possível, quantificáveis.

Nesta proposta, são definidos critérios, os quais são valorados a partir de uma comparação de critério contra critério em uma matriz de avaliação que resulta na definição dos pesos de cada critério, traduzindo o seu grau de importância na concepção do produto. A Tabela 1 apresenta um exemplo de matriz de valoração de critérios. Na primeira linha e na primeira coluna são dispostos os critérios a serem comparados. A seguir, é comparado o critério da linha A com os critérios de cada coluna (A – E). São atribuídos valores que definem a importância relativa entre cada critério. Os valores são um (1) para mais importante, zero (0) para menos importante e meio (0.5) para igual importância. Assim, se o critério A for menos importante que o critério B é colocado zero (0) na célula correspondente a AB (linha de A) e um (1) na célula correspondente a BA (linha de B). Após as sucessivas comparações, critério a critério, são somados os valores de cada linha e os valores são normalizados pela soma total das linhas. Assim, os critérios adquirem pesos (*wi*) cuja soma total é um (1) ou cem por cento (100%).

Estes pesos são então utilizados para comparar os conceitos gerados a fim de estabelecer as melhores concepções em relação aos diversos critérios de seleção. Esta

abordagem, que tem alto potencial computacional, consiste em uma ferramenta poderosa para a análise tanto quantitativa como qualitativa das concepções de projeto.

Tabela 1: Exemplo de matriz para valoração de critérios.

	A	**B**	**C**	**D**	**E**	**SOMA**	**PESO (*Wi*)**
A	X	0	0,5	1	0	1,5	0,14
B	1	X	1	1	1	4	0,38
C	0,5	0	X	1	0,5	2,5	0,24
D	0	0	0	X	1	1	0,10
E	1	0	0,5	0	X	1,5	0,14
						10,5	1,00

Fonte:O Autor.

Partindo-se dos pesos dos critérios e das soluções pré-aprovadas na triagem pode-se submeter estas soluções a uma segunda etapa no processo de seleção, a definição do valor da função utilidade (BACK *et al.*, 2008).

A Tabela 2 mostra a matriz de determinação da função utilidade, onde "V" indica a valoração do critério qualitativo (de 1 a 5, sendo 1= satisfatório e 5=excelente) e "W" é o peso de cada critério obtido a partir da comparação na matriz de Pugh do exemplo anterior. A valoração destes critérios nesta etapa é sempre positiva, uma vez que as soluções que não atendiam satisfatoriamente aos critérios já foram descartadas na etapa anterior.

Tabela 2: Exemplo de matriz para determinação da função Utilidade.

Critérios de Seleção	Pesos W	Concepções Geradas							
		Concepção AB		Concepção LM		Concepção CD		Concepção XZ	
		V	V * W	V	V * W	V	V * W	V	V * W
A	0,14	2	0,28	3	0,42	2	0,28	2	0,28
B	0,38	3	1,14	5	1,9	2	0,76	1	0,38
C	0,24	3	0,72	4	0,96	3	0,72	1	0,24
D	0,1	2	0,2	1	0,1	2	0,2	2	0,2
E	0,14	3	0,42	2	0,28	2	0,28	1	0,14
Valor da Função Utilidade		Somatório	2,76	Somatório	3,66	Somatório	2,24	Somatório	1,24
Ordenação das Concepçòes		2ª Posição		1ª Posição		3ª Posição		4ª Posição	

Fonte: Adaptado de Back et al. (*2008).*

Adotando-se uma escala numérica para uma valoração qualitativa dos critérios combinados com os respectivos pesos destes critérios, pode-se determinar uma ordenação da qualidade das soluções, concluindo o processo de seleção. A esquerda da tabela estão os

critérios de seleção (A,B,C,D e E) e seus respectivos pesos (w). Cada uma das concepções tem duas colunas. Na primeira coluna, a concepção recebe um valor numérico (v) segundo uma avaliação qualitativa (que também pode ser quantitativa) em um determinado critério. Na coluna ao lado, este valor é multiplicado pelo peso do critério (v*w). Do somatório destes valores da segunda coluna, resulta a determinação do valor da função utilidade, onde o valor mais alto deste somatório indica a melhor solução segundo aqueles critérios.

2.1.2.5 ELABORAÇÃO DO CONCEITO FINAL

O conceito final poderá ser uma das concepções, a que obteve melhor avaliação final, ou uma concepção híbrida gerada a partir das melhores soluções existentes nas concepções geradas.

Frequentemente as decisões de projeto baseiam-se em critérios altamente subjetivos, as necessidades intangíveis do cliente, deixando os designers em uma situação muito arriscada ao fazer determinadas escolhas, qualquer erro de interpretação destas necessidades pode levar o projeto todo ao fracasso.

Sendo assim, o processo de desenvolvimento de projeto traz ferramentas e técnicas importantes para apoiar as escolhas de projeto auxiliando a tomada de decisão e não, simplesmente, justificando tais decisões como fruto da criatividade ou inspiração divina. Neste trabalho, estas ferramentas serão utilizadas para o auxílio nas escolhas de projeto, tornando o processo de tomada de decisão mais sistemático e criterioso.

2.2 INTERAÇÃO HOMEM-COMPUTADOR

A interação homem-computador (IHC) como área de conhecimento teve início na década de 1960, juntamente com as primeiras interfaces gráficas, como uma subárea da ergonomia cognitiva, que estuda a adaptação dos produtos e ambientes de trabalho aos processos psicológicos do ser humano.

A IHC estuda de forma interdisciplinar a interação do homem com o computador, para melhor adaptar o último ao primeiro. Deste ponto de vista, a IHC estaria para o design de interfaces, assim como a ergonomia estaria para o design de produtos, sendo a base sólida onde o designer pode se apoiar.

Em qualquer interface, gráfica ou não, o usuário valoriza a facilidade com que consegue executar as tarefas pretendidas e a comodidade ao executá-las. Para que se possam atender às expectativas do usuário, é fundamental conhecer os elementos envolvidos no processo de trabalho. Na criação da interface de um aplicativo, os principais elementos são os recursos disponíveis (os equipamentos, por exemplo), a finalidade da interface e os seus usuários finais.

Uma interface gráfica adequada aproveita ao máximo as potencialidades do computador em questão, desde a CPU aos periféricos, de modo a tornar um programa computacional mais fácil de usar. Assim, as escolhas e decisões a tomar ao longo do processo de criação da interface devem ser feitas com base na compreensão dos usuários e nos recursos do sistema. É preciso ter em consideração as especialidades e dificuldades das pessoas, suas necessidades e objetivos ao utilizar a ferramenta digital e considerar o que poderá ajudar os

usuários no modo como executam as suas tarefas. É importante conhecer o tipo de periféricos e que recursos o sistema tem disponível, pensar sobre o que poderá dar qualidade às interações e adotar técnicas testadas pelos usuários ao longo de todo o desenvolvimento da interface (PREECE, ROGERS e SHARP, 2002).

Os estudos em IHC preocupam-se em entender como as pessoas utilizam os sistemas informatizados, de modo a projetar sistemas que preencham melhor os objetivos e necessidades dos usuários. A partir destes conceitos percebe-se que é preciso identificar as necessidades do usuário em seu contexto de trabalho, a partir daí, criar uma interface prática, útil e agradável.

A ABNT – Associação Brasileira de Normas Técnicas – (NBR 9241-11/2002) define usabilidade como a medida na qual um produto pode ser usado por usuários específicos para alcançar objetivos específicos como eficácia, eficiência e satisfação num contexto específico de uso. Neste sentido, o estudo da usabilidade é parte essencial no entendimento da interação homem-computador.

2.2.1 INTERAÇÃO 3D

Interação é o processo de comunicação entre pessoas e sistemas interativos (PREECE *et al.* 1994). Neste processo, usuário e sistema trocam frequentemente de papéis em uma relação onde um fornece informações específicas e o outro as recebe, interpreta e realiza uma determinada ação. A interação do usuário com programas CAD/CAE ocorre pela interface gráfica do software que compreende também (além dos menus e barras de ferramentas) a área de trabalho virtual.

De acordo com McGraw-Hill (2005), a interação homem-computador se caracteriza, em um primeiro nível, pelas capacidades do homem e do computador de aceitarem dados de entrada, processá-los, e gerarem dados de saída. Em outro nível, a interação consiste em um software para a interface do usuário que governa o significado das entradas e saídas para o computador, assim como as regras e expectativas correspondentes que o usuário aplica para gerar ações significativas. O modelo interno de interação do usuário se apóia nas pistas visuais da interface projetadas de acordo com princípios ergonômicos e semióticos (fatores humanos).

Interface é o nome dado a toda porção de um sistema com a qual um usuário mantém contato ao utilizá-lo, tanto ativa quanto passivamente. A interface engloba o software e o hardware (dispositivos de entrada e saída, tais como: teclados, mouse, *tablets*, monitores, impressoras e etc.). Uma interface tridimensional em um software para o desenvolvimento de produto deve levar em conta a usabilidade do programa, permitindo interatividade e um bom desempenho gráfico.

O desenvolvimento da tecnologia na ciência da computação aumentou significativamente o poder de processamento dos computadores enquanto diminuiu seu tamanho. Este desenvolvimento tecnológico proporcionou a criação de uma variedade de interações homem-computador. Utilizando as definições de Kettner (1995), uma operação de interação é o que o usuário quer realizar (rotacionar, escalar, mover um objeto, etc.) e uma

técnica de interação são os meios utilizados para se atingir uma determinada operação de interação.

Considerando os requisitos de usabilidade da interface, para facilitar a aprendizagem do usuário e facilitar a lembrança do funcionamento da interface 3D, metáforas são utilizadas. Assim como é feito em outros softwares do cotidiano (por exemplo, um processador de texto utiliza a metáfora da máquina de escrever). Neste trabalho, se considera apenas a interação 3D a partir de dispositivos 2D, originando a necessidade do uso de metáforas para interação. Portanto, metáforas 3D são técnicas de interação 3D associadas a conceitos metafóricos.

Segundo Kettner (1995), uma metáfora 3D explica a manipulação de uma operação 3D utilizando um dispositivo de entrada 2D. Ela consiste em duas partes: o conceito metafórico, que descreve o mapeamento da estrutura do modelo mental do usuário na interface, e a implementação, que traduz os eventos da interface para o modelo matemático tridimensional do aplicativo. Para facilitar o estabelecimento de um modelo mental, os conceitos metafóricos se relacionam diretamente à experiência tridimensional anterior do usuário.

Para o desenvolvimento das ferramentas necessárias à melhoria da interatividade, neste trabalho foi conduzida uma pesquisa nos principais softwares utilizados no segmento CAD/CAE, aproveitando os conceitos já estabelecidos por estes programas e a experiência anterior dos usuários.

2.2.1.1 VISUALIZAÇÃO 3D

Após a criação dos objetos tridimensionais na forma de dados matemáticos, o próximo passo é efetuar a sua apresentação. Se comparado com o processo de visualização bidimensional, a visualização de objetos tridimensionais é bastante mais complexa. Esta complexidade deve-se, basicamente, ao fato de que os dispositivos gráficos existentes (monitores) são adequados à apresentação de imagens planas, bidimensionais.

Assim, o principal problema a ser resolvido é o de apresentar uma entidade tridimensional em um meio bidimensional (2D). Este processo é resolvido em computação gráfica através do uso de projeções (FOLEY *et al.*, 1992). Outro componente importante no processo de visualização tridimensional em computadores, além das projeções, é a existência de um "observador virtual" das cenas. Este "observador" é quem define de que ponto de vista a cena será exibida. Um ponto importante na definição do "observador virtual" é sua orientação, ou seja, para onde ele está olhando dentro do universo. Esse observador virtual é, na verdade, uma câmera virtual ou sintética.

A tela do computador fornece uma janela (tela bidimensional) para olhar dentro de um mundo virtual tridimensional. É natural pensar na janela gráfica como sendo a lente de uma câmera fotográfica através da qual nós vemos uma imagem, então o processo de manipulação do ponto de vista é análogo ao movimento da câmera no espaço.

Manipular interativamente uma câmera virtual através do espaço tridimensional é uma das principais características de todos os aplicativos gráficos 3D. Para que um sistema de modelagem interativo possa mostrar ao usuário uma cena realística, com um bom senso de

tridimensionalidade dos objetos, é essencial que o sistema forneça uma maneira efetiva de controlar o ponto vista da câmera.

Para que a câmera virtual seja controlada com eficiência, é preciso oferecer meios para controlar, pelo menos, seus seis (6) graus de liberdade (6DOF – *Degrees of Freedom*) a partir dos dispositivos 2D (mouse, *tablet*, monitor). Os seis graus de liberdade são a translação e rotação nos três (3) eixos cartesianos (X, Y e Z). É possível ainda considerar um sétimo grau que seria a profundidade de campo da câmera. O controle do ponto de vista é especialmente importante durante o processo de manipulação dos objetos, pela necessidade de se ver o que se está fazendo de um ângulo apropriado. Se o usuário não consegue ver o objeto adequadamente, então a manipulação deste objeto será muito difícil. A própria noção de interatividade requer que o usuário veja o que ele está fazendo durante a execução da tarefa, tendo um *feedback* imediato dos resultados de suas ações.

O controle destes graus de liberdade, por sua importância e complexidade, exige grande sofisticação e maturação dos algoritmos gráficos. Isto induz a utilização de uma biblioteca gráfica 3D, que traz em seu código a solução (ou a base da solução) para estes problemas das transformações geométricas tridimensionais, as quais resultam nas suas projeções bidimensionais mostradas na tela.

2.2.1.2 MANIPULAÇÃO DIRETA 3D A PARTIR DE DISPOSITIVOS 2D

A eficácia de sistemas 3D depende das ferramentas e técnicas disponíveis ao usuário para uma rápida e amigável definição e modificação do modelo tridimensional. De acordo com Ketner (1994), o desenvolvimento de técnicas eficazes de interação 3D para dispositivos de entrada e saída em 2D tem sido um desafio clássico na computação gráfica. Embora já existam tecnologias como a realidade virtual e a realidade aumentada que solucionam esse problema de forma satisfatória com a utilização de dispositivos especiais de interação como luvas e óculos 3D, entre outros, ainda é usual que os usuários em geral tenham acesso apenas aos dispositivos de entrada e saída 2D populares, tais como *joystick*, mouse e monitor.

Os dispositivos 2D têm ainda várias vantagens sobre os dispositivos 3D. Os dispositivos 2D de entrada são largamente aceitos e conhecidos pelos usuários em geral para o uso em interfaces gráficas tradicionais, então os dispositivos 2D são mais familiares ao usuário do que os dispositivos 3D. Poucos são ainda os grupos de pesquisas e empresas capazes de arcar com os custos de uma interação 3D, seja pelos preços dos dispositivos em si, seja pelos preços dos sistemas necessários ao processamento de tal interação, enquanto que os dispositivos 2D e um PC se encontram acessíveis ao usuário comum.

Além disto, com o uso do *mouse*, normalmente o antebraço pode descansar sobre a mesa em uma postura mais adequada em termos ergonômicos, enquanto que os dispositivos de interação 3D (como luvas, óculos e sensores com cabos), além de mais pesados, exigem uma movimentação maior, fatigando o usuário. Dependendo da aplicação, interações em 3D são mais bem executadas se tiverem um ou dois graus de liberdade de interação restringidos. A dificuldade dos usuários em manipular um grande número de graus de liberdade pode introduzir erros. Então, o tempo gasto para executar uma tarefa pode ser reduzido ao limitarem-se as possibilidades de movimento para evitar erros.

O tamanho e a portabilidade dos dispositivos também pode ser um problema. Enquanto, atualmente, a manipulação em dispositivos 2D encontra-se até mesmo em telefones celulares, os dispositivos 3D ainda precisam de um espaço maior, como as cavernas de imersão virtual.

O grau de interatividade é um requisito importante para a eficiência de manipulação tridimensional de um sistema. As modificações no modelo e no ambiente durante o processo de design devem ser visualizadas imediatamente, em tempo real. Um importante paradigma na interatividade das interfaces gráficas é o conceito de “manipulação direta”, a qual consiste em um estilo de interação homem-computador que envolve a contínua representação do objeto de interesse com um imediato, reversível e sucessivo *feedback* das ações e operações exercidas sobre o mesmo. O objetivo é permitir que o usuário manipule diretamente os objetos apresentados a ele, realizando ações e operações que correspondem às metáforas do mundo real.

Este tipo de abordagem para objetos e ações pode tornar mais fácil para o usuário aprender e utilizar a interface (pode-se dizer que a interface torna-se mais natural e intuitiva). Um imediato e sucessivo *feedback* permite que o usuário cometa menos erros e complete as tarefas em menos tempo, porque ele pode conferir os resultados de uma ação antes mesmo de completá-la. Um exemplo prático de manipulação direta é o redimensionamento de um objeto gráfico, como um retângulo, onde, ao arrastar um de seus cantos, pode-se notar o resultado esperado (a nova forma e tamanho) imediatamente (SHNEIDERMAN, 1983). A Figura 7 mostra a manipulação do tamanho de um retângulo a partir de um de seus cantos.

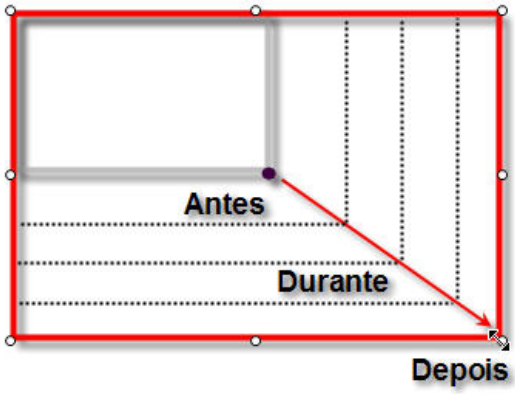


Figura 7: Manipulação direta de um retângulo.
Fonte: O Autor.

Entretanto, o uso de manipulação direta para modelagem tridimensional torna-se mais complexo pelo fato de que os parâmetros para as transformações em 3D devem ser especificados por uma interação em representações projetadas em 2D. Uma especificação gráfica para os parâmetros de posição, orientação e dimensões de um objeto é um pré-requisito para uma interface de manipulação direta. Com programas de desenho em 2D, o usuário pode manipular estes parâmetros movendo “pontos de controle” com um cursor gráfico. A posição XY do ponto de controle pode determinar, por exemplo, o tamanho do

objeto. A manipulação direta em 3D é mais complexa, uma vez que os movimentos do cursor em 2D devem ser convertidos em transformações em 3D no objeto. Muitas aplicações 3D resolvem este problema oferecendo, além da vista tridimensional do objeto em perspectiva, outras três vistas ortográficas nas quais o objeto pode ser manipulado com as devidas restrições em 2D. Conforme Emmerik (1990), ainda que este tipo de interface seja utilizado em muitas aplicações tridimensionais, ela traz no mínimo três desvantagens:

- É necessário utilizar muito espaço de tela para mostrar o mesmo objeto por quatro ângulos diferentes: três vistas ortogonais, mais a vista em perspectiva;
- A interação em 2D nos pontos de controle em vistas ortogonais pode ser adequada para transformações como escala, mas requerem outros métodos para translação e rotação;
- A representação por vistas ortogonais e suas interações não são exatamente manipulação direta, uma vez que requerem do usuário um grau de abstração maior para entender os sistemas projetivos e, ainda, a manipulação do objeto se dá em uma vista ortográfica enquanto o resultado esperado é visualizado na vista em 3D.

A manipulação direta de objetos em 3D a partir de dispositivos 2D requer o controle de seis graus de liberdade: a translação (mover) ao longo dos três eixos perpendiculares de um sistema de coordenadas, X, Y e Z e a rotação ao redor destes mesmos eixos. Uma das maneiras de resolver o problema da manipulação direta em 3D a partir de dispositivos 2D é a restrição destes graus de liberdade. Utilizando ferramentas ou opções que permitam estas restrições. Desta forma, a manipulação se torna mais eficiente e produz menos erros.

É possível que uma aplicação disponibilize uma opção para que o usuário determine quais os eixos do sistema de coordenadas estão ativos para edição. Assim, o usuário pode impedir a movimentação no eixo Z, por exemplo, permitindo que o mapeamento do movimento do cursor em 2D (XY da tela) seja facilmente traduzido para o movimento do objeto no plano XY do mundo virtual. Tais restrições podem ser feitas através de teclas de atalho, menus de contexto (*Pop-up*), barras de ferramentas com botões persistentes (que permanecem selecionados após o clique), grupo de seleção de opções ou, ainda, objetos visuais em 3D criados especialmente para a manipulação direta dos objetos, como mostrar os eixos de coordenadas locais dos objetos, um *pivot* para a origem das transformações, esfera virtual para rotação, *bounding box* (caixa envolvente) do objeto selecionado, etc.

Assim, boas soluções de interface homem-computador que compatibilizem os dispositivos de entrada 2D e a interação em 3D tem o potencial de melhorar a usabilidade, aprimorar as técnicas de interação e tornar as aplicações 3D mais fáceis e populares.

A partir destas observações, entende-se que o uso de dispositivos bidimensionais como mouse e monitor para a interação tridimensional é a melhor escolha neste projeto. Além de aproveitar o conhecimento prévio dos usuários com a utilização destes equipamentos, permite que a plataforma T-CADE seja independente de hardwares específicos, possa ser utilizada nos computadores mais populares e ainda venha a contribuir com o estudo da interatividade e usabilidade neste tipo de equipamento.

2.3 ENGENHARIA DE USABILIDADE

O princípio da ergonomia é estudar o conforto e a adaptação aos objetos e interfaces visando a produtividade e à satisfação das pessoas. De acordo com a definição oficial adotada pela *International Ergonomics Association* – IEA (IEA site s.d.) no ano de 2000:

> *"Ergonomics (or human factors) is the scientific discipline concerned with the understanding of interactions among humans and other elements of a system, and the profession that applies theory, principles, data and methods to design in order to optimize human well-being and overall system performance."*

"Ergonomia (ou fatores humanos) é a disciplina científica que investiga o entendimento das interações entre homens e outros elementos de um sistema, e a aplicação da teoria, princípios, dados e métodos ao design buscando aperfeiçoar o bem-estar humano e o desempenho geral do sistema". (Tradução do autor).

No contexto da ergonomia, existe uma parte específica da ciência da computação intimamente relacionada ao design e ao campo de IHC, que trata da questão de como projetar artefatos digitais que sejam fáceis de usar, conhecida como Engenharia de usabilidade.

A Engenharia de Usabilidade é a disciplina que fornece métodos estruturados para a obtenção da usabilidade durante o desenvolvimento de sistemas interativos (MAYHEW, 1999). Usabilidade é um atributo de qualidade que avalia a facilidade de uso das interfaces de usuários (NIELSEN e TAHIR, 2002).

Existem muitos motivos para se investir ou se preocupar com a usabilidade de uma interface. Os softwares desenvolvidos recentemente têm devotado em média 48% do código á implementação da interface com usuário (MYERS e ROSSON, 1992). Parece então ser justificável que se dedique uma porção razoável do esforço no desenvolvimento de software para assegurar uma boa usabilidade destas interfaces.

Usabilidade é considerada como um atributo de qualidade crucial em qualquer sistema interativo. A usabilidade está diretamente ligada à produtividade. O tempo que os funcionários de uma empresa gastam tentando, sem sucesso, executar uma determinada tarefa é um dos valiosos recursos desperdiçados pela empresa, que lhes paga para estarem trabalhando sem conseguir fazer seu trabalho propriamente, além da qualidade de vida e satisfação do próprio funcionário. Melhorar a usabilidade também significa diminuir custos de treinamento, aumentar a satisfação dos usuários, a aceitação do produto e, por consequência, aumentar as vendas.

De acordo com Nielsen (1993), "a aceitabilidade global de um sistema está dividida entre aceitabilidade social e aceitabilidade prática", onde a primeira se caracteriza pela aceitação, por parte dos usuários, da necessidade e da relevância do papel social proposto por um determinado sistema. Quanto a sua aceitação prática, ela se subdivide em critérios como

custo, confiança, segurança, compatibilidade, flexibilidade, dentre os quais se encontra a qualidade de uso.

Assim, a usabilidade tem o potencial de melhorar a aceitação do software no mercado (o que é bom para a empresa) e também pode diminuir o tempo de execução da tarefa e torná-la mais fácil (o que é bom para o usuário).

2.3.1 CRITÉRIOS BÁSICOS DE USABILIDADE

A usabilidade tem como objetivo elaborar interfaces capazes de permitir uma interação fácil, agradável, com eficácia e eficiência. Ela deve capacitar a criação de interfaces "transparentes", de maneira a não dificultar a execução das tarefas, permitindo ao usuário pleno controle do ambiente sem se tornar um obstáculo durante a interação (NIELSEN, 1993).

A qualidade de uma interface pode ser medida como sendo a propriedade de poder alcançar o resultado desejado, ou seja, que o sistema tenha a capacidade de solucionar o problema para o qual foi elaborado. Conforme Nielsen (1993), a qualidade se divide em dois aspectos importantes do software: se ele é útil, satisfazendo uma real necessidade capaz de justificar seu desenvolvimento, e sua usabilidade, que busca aplicar características que facilitem sua interação com o usuário.

Segundo Nielsen (1993), a usabilidade pode ser dividida em cinco critérios básicos:

- **Intuitividade;**
- **Eficiência;**
- **Memorização;**
- **Erro;**
- **Satisfação.**

2.3.1.1 INTUITIVIDADE

O sistema deve apresentar facilidade de uso permitindo que, mesmo um usuário sem experiência, seja capaz de produzir algum trabalho satisfatoriamente. Entre os atributos que compõem a usabilidade, o mais importante é a intuitividade.

Para que ocorra minimamente uma interação, a interface deve apresentar características que facilitem sua utilização, permitindo que usuários iniciantes ou avançados possam aprender seus recursos de forma clara e objetiva. A aprendizagem de um sistema é um processo contínuo, cujo desempenho melhora a cada nova interação, não podendo ser considerado como uma distinção entre "aprendido" / "não aprendido" (NIELSEN, 1993).

Para a avaliação da usabilidade, é preciso utilizar uma amostragem de usuários pertencentes ao grupo a que se destina o software e que não tenha utilizado o sistema, medindo-se o tempo necessário para a execução de uma determinada tarefa. Assim, é avaliado o nível de controle da interface em sua utilização. Essa medição tem como objetivo informar a eficiência do nível de interação. Quanto menor for o tempo gasto, melhor será a qualidade da

interface. Pode-se, ainda, filmar o usuário durante a utilização, detectando suas ações e seus erros mais comuns (NIELSEN, 1993).

2.3.1.2 EFICIÊNCIA

O sistema deve ser eficiente em seu desempenho, apresentando um alto nível de produtividade. A eficiência é determinada através da medição do tempo gasto na utilização do software por usuários experientes.

2.3.1.3 MEMORIZAÇÃO

Um sistema deve apresentar facilidade de memorização permitindo que usuários ocasionais consigam utilizá-lo mesmo depois de um longo intervalo de tempo. A memorização pode ser medida através do registro do tempo decorrido na última interação e do tempo gasto para re-executar uma mesma tarefa específica algum tempo depois. Outra forma de medi-la é verificar, após um determinado tempo de interação, se o usuário é capaz de reconhecer comandos e ações específicas através de questionário.

2.3.1.4 ERROS

A quantidade de erros apresentados pelo sistema deve ser a mais reduzida possível. Além disso, eles devem apresentar soluções simples e rápidas, mesmo para usuários iniciantes. Erros graves ou sem solução não podem ocorrer. É considerado erro qualquer ação que leve o usuário a não executar determinada tarefa. Sua medição não é baseada no tempo, e sim na quantidade de suas ocorrências durante a execução de uma tarefa específica.

Alguns aplicativos apresentam recursos de ajuda com o objetivo de solucionar erros causados por desatenção do usuário como, por exemplo, caixas de mensagem dizendo ao usuário o que está sendo feito errado, ou o que falta fazer, e como ele pode solucionar aquele erro. Alguns erros são considerados catastróficos quando não podem ser detectados pelo usuário, resultando em uma produção defeituosa ou causando perdas irreversíveis de informações. Esse tipo de erro deve ser tratado separadamente e com um maior grau de atenção.

2.3.1.5 SATISFAÇÃO

O sistema deve agradar aos usuários, sejam eles iniciantes ou avançados, permitindo uma interação agradável. A satisfação representa o quão agradável deve ser a interação do usuário com o sistema. É importante observar que o atributo de satisfação é bastante subjetivo e diferente dos demais critérios de usabilidade, pois cada usuário apresenta características individuais que podem resultar em diferentes atitudes diante do computador.

Por ser uma característica subjetiva da usabilidade, a satisfação é medida através da aplicação de questionários individuais, devendo ser levada em consideração a média das respostas obtidas de um determinado grupo de usuários. Embora as respostas individuais sejam subjetivas, quando realizadas com diversos usuários, podem-se obter medidas objetivas de satisfação em relação ao sistema.

Para Nielsen (1993), esses cinco atributos compõem a natureza multidimensional da usabilidade. A preocupação com usabilidade deve estar presente durante todo o ciclo de

projeto e não apenas quando o produto já está finalizado. Assim, podem-se fazer as modificações necessárias sem que elas impliquem em um custo elevado de re-projeto.

É importante salientar que o designer precisa saber administrar seus recursos para atingir seus objetivos da melhor maneira possível. Nielsen (1993) advoga ainda que os especialistas vão sempre propor os melhores métodos possíveis. Infelizmente, o ótimo é inimigo do bom, na medida em que insistir em usar os melhores métodos pode levar a não se usar método algum.

2.3.2 ENGENHARIA DE USABILIDADE COM DESCONTO

Muitas vezes, os longos processos de avaliação de usabilidade realizados por especialistas que buscam a "realização do ótimo" nas interfaces podem intimidar os designers, extrapolar orçamentos e prazos e dificultar a aplicação realista dos princípios de usabilidade para um projeto pequeno. Entretanto, é importante ter algum trabalho de usabilidade sendo feito, antes a realização do "bom", do que nenhum trabalho, mesmo que os métodos empregados não tenham sido os melhores e não necessariamente vão produzir os melhores resultados. Métodos cuidadosos são mais dispendiosos, tanto em termos de custos e tempo, como da experiência necessária, o que pode levar aos problemas acima referenciados.

Em função disto, Nielsen (1993) propõe a engenharia de usabilidade com "desconto", que é baseada no uso das seguintes técnicas "baratas":

- **Observação do usuário e da tarefa;**
- **Cenários de uso;**
- **Verbalização simplificada;**
- **Avaliação heurística.**

2.3.2.1 OBSERVAÇÃO DO USUÁRIO E DA TAREFA;

A usabilidade tem o foco no usuário como um de seus princípios básicos. As regras principais para a "análise da tarefa com desconto" são simples: observar os usuários, manter silêncio e deixá-los trabalhar como eles fariam normalmente, sem interferências.

Conforme Nielsen (2002), o método mais básico e útil para a avaliação de usabilidade é o teste com usuários:

- Formar um grupo representativo dos usuários do programa;
- Definir tarefas a serem cumpridas que possam avaliar as ferramentas propostas;
- Observar o que os usuários fazem, o que eles tentam fazer sem sucesso, onde eles acertam, onde eles erram e, o mais importante, não interferir!

De acordo com Nielsen, o sistema pode ser testado com sucesso por um grupo de três a cinco avaliadores (NIELSEN, 2008). Um ponto importante da avaliação é testar os usuários individualmente e deixar que eles resolvam os problemas sozinhos. Se eles forem ajudados ou tiverem sua atenção direcionada para uma área em particular da tela, os resultados foram contaminados.

2.3.2.2 CENÁRIOS DE USO

Para que se entenda o conceito de cenários, é necessário que se introduza o conceito de prototipagem. Mais adiante neste trabalho, no capítulo da engenharia de software, existe um item específico sobre prototipagem, mas, neste momento, algumas considerações são importantes.

Os protótipos são implementações concretas, mas parciais, do projeto do sistema. A ideia por trás da prototipação é a de diminuir a complexidade da implementação pela eliminação de partes do sistema. Protótipos horizontais reduzem o nível de funcionalidade e resultam em uma camada superficial de interface com o usuário (permite avaliar como a interface se encaixa como um todo; não se pode interagir com dados reais; é visual), enquanto que protótipos verticais implementam toda a funcionalidade de apenas uma função (permite testar a funcionalidade de uma pequena parte do sistema; usuários não podem se mover livremente pelo sistema; oferecem poucas tarefas).

Um cenário é uma narrativa, textual ou pictórica, de uma situação (de uso de uma aplicação), envolvendo usuários, processos e dados reais ou potenciais (CARROLL, 1995). Os cenários ou os casos de uso apenas simulam uma interação do sistema com o usuário ao longo de um caminho previamente definido. Podem representar uma redução radical de recursos na prototipagem em ambas direções (horizontal e vertical). Eles podem ser usados para implementar estudos baseados na verbalização do usuário. E, por serem simples e baratos, podem ser alterados facilmente.

2.3.2.3 VERBALIZAÇÃO SIMPLIFICADA

A verbalização simplificada envolve, basicamente, o usuário pensando em voz alta enquanto usa o sistema. Assim, o observador pode conhecer o porquê das ações dos usuários e identificar elementos de interface que levem a entendimentos equivocados e que devam ser revisados.

Tradicionalmente, estes estudos são conduzidos por psicólogos e especialistas em usabilidade, mais especificamente em experimentos gravados em vídeo, que realizam análises detalhadas do protocolo verbal. Este tipo de estudo com especialistas pode não ser acessível (em função da escassez de recursos) aos desenvolvedores comuns.

Os estudos de Nielsen (1993) mostram que programadores e analistas podem aplicar técnicas de verbalização eficientemente para avaliar interfaces com o usuário com um mínimo de treinamento e que, mesmo os experimentos metodologicamente simples, irão ter sucesso em encontrar problemas de usabilidade.

2.3.2.4 AVALIAÇÃO HEURÍSTICA

A avaliação heurística é um método informal de inspeção de interfaces, onde especialistas de usabilidade julgam cada elemento da interface tendo como referência princípios heurísticos de usabilidade comumente aceitos. Nielsen e Molich (1990) recomendam 10 critérios de qualidade na avaliação heurística:

1. Visibilidade do status sistema: O sistema deve sempre manter o usuário informado sobre o que está acontecendo com um adequado *feedback* em um tempo razoável.

2. Linguagem familiar ao usuário: O sistema deve usar palavras, frases e conceitos familiares aos usuários, ao invés de usar termos orientados ao sistema sem significado ao usuário. Seguir convenções do mundo real, fazendo informações aparecerem em uma ordem lógica e natural.

3. Controle do usuário: Os usuários, frequentemente, escolhem funções por engano e irão necessitar de uma "saída de emergência" claramente marcada para cancelar a ação indesejada sem ter que passar por uma caixa de diálogo extensa. Suportar "Desfazer" e "Refazer" é importante.

4. Consistência: Os usuários não devem ter que se preocupar se diferentes palavras, situações ou ações significam a mesma coisa. Ações similares devem provocar reações similares.

5. Prevenção de erros: Ainda melhor do que fornecer mensagens de erro bem explicativas é projetar uma interface cuidadosamente para preveni-los de acontecerem em primeiro lugar.

6. Memorização mínima: Deve ser minimizada a sobrecarga da memória do usuário, criando objetos, ações e opções que sejam visíveis e facilmente identificáveis.

7. Uso eficiente e flexível: Aceleradores (como teclas de atalho), normalmente negligenciados pelos usuários novatos, podem melhorar muito o desempenho dos usuários mais experientes. O sistema deve estar preparado para os diferentes tipos de usuários e permitir aos mesmos personalizar o uso das ações mais frequentes.

8. Projeto minimalista, simples: a interface não deve conter informação irrelevante, cada informação desnecessária colocada em uma interface compete com a informação relevante, diminuindo sua visibilidade e tomando mais tempo do usuário para processá-la.

9. Ajuda aos usuários para reconhecer, diagnosticar e recuperar-se de seus erros: As mensagens de erro devem ser expressas em uma linguagem simples (sem códigos), indicando precisamente o problema e, construtivamente, sugerindo uma solução.

10. Ajuda e documentação: Ainda que seja possível usar o sistema sem precisar da documentação, é melhor fornecer ajuda e documentação completa. A informação deve ser fácil de encontrar, com o foco na tarefa a ser desempenhada pelo usuário, com uma lista concreta sobre os passos necessários para completá-la e, se possível, evitar documentos muito longos.

Embora esses critérios demandem alguma experiência para serem aplicados em todos os casos, mesmo não-especialistas podem encontrar muitos problemas de usabilidade em avaliações heurísticas. Os problemas restantes podem ser encontrados pela técnica de verbalização simplificada e de cenários de uso. Também se recomenda que diferentes pessoas

envolvidas com o desenvolvimento realizem avaliações heurísticas de modo a encontrar mais e diferentes problemas (NIELSEN e MOLICH, 1990).

A avaliação heurística se destaca por ser de fácil aplicação e por requerer poucos recursos e um pequeno número de pessoas envolvidas. O modo mais direto de se obter informações sobre as necessidades e expectativas de um usuário é observar as suas ações e intenções na tentativa de interação com uma determinada interface, observando suas tomadas de decisões, informações, acionamentos, deslocamentos, comunicações e perguntar como e porque o fazem.

A avaliação da usabilidade pode ser utilizada como um procedimento para aquisição de informação sobre a forma que um sistema é utilizado a fim de aprimorar recursos em uma interface em construção e também para avaliar uma interface já finalizada. De acordo com o momento em que for realizada, a avaliação de usabilidade pode ser formativa ou somativa (HARTSON e HIX, 1993).

A **avaliação formativa** ou construtiva (*formative evaluation*) acontece antes da implementação e tem participação na formação do sistema, com influência sobre as características do produto em desenvolvimento, uma vez que auxilia o projetista a formar e a refinar o projeto. Este tipo de avaliação deve ser utilizado sempre que os projetistas precisem compreender melhor o que os usuários querem e precisam, ou quando precisem verificar se suas ideias estão atendendo às necessidades conhecidas dos usuários. A avaliação formativa tem por objetivo:

- Prever a usabilidade (de alguns aspectos) do produto;
- Verificar a compreensão da equipe de projeto sobre os requisitos dos usuários, examinando como um sistema existente está sendo utilizado;
- Testar ideias de forma rápida e informal.

A **avaliação somativa** ou conclusiva (*summative evaluation*) acontece após a implementação, com o objetivo de testar o funcionamento apropriado do sistema final. Em outras palavras, os principais objetivos da avaliação somativa são:

- Identificar problemas de interação e interface;
- Propor alterações para aumentar a qualidade de uso de um sistema interativo.

A forma mais comum de se avaliar a usabilidade de um software é observando a sua interação com o usuário, o que pode ser feito em laboratório, com uma quantidade mínima de usuários para o qual o sistema foi desenvolvido, ou no próprio ambiente de trabalho onde o sistema será implantado. Segundo os autores (NIELSEN e MOLICH, 1990), o mais importante nesse processo avaliativo é que, sempre que possível, deve-se utilizar o usuário certo para as tarefas certas, a fim de se obter o máximo de desempenho avaliativo.

Sendo assim, a usabilidade é definida ou medida para um contexto em que um sistema é operado. Um sistema pode proporcionar boa usabilidade para um usuário experiente, mas péssima para novatos, ou vice e versa. Pode ser fácil de operar uma vez ou outra, mas difícil,

se for utilizado no dia a dia. Portanto, é fundamental que se considere o contexto de uso para o qual uma interface está sendo concebida e no qual ela será testada. É imperativo que a interface seja testada com os usuários adequados, fazendo o que o programa é projetado para fazer.

O entendimento destes princípios de usabilidade é fundamental para o projeto de uma interface gráfica cujo objetivo primeiro é a melhoria da usabilidade. Assim, neste projeto a observação destes critérios de usabilidade durante o desenvolvimento do sistema foi foco constante de mudanças na implementação e um dos principais critérios avaliados para a tomada de decisão nas soluções propostas.

2.4 ENGENHARIA DE SOFTWARE

O processo de desenvolvimento de um artefato digital é uma atividade de design. Quanto maior a complexidade do software, maior a complexidade do processo de seu desenvolvimento. Diante disto, para o correto projeto do produto, neste caso, um artefato digital, é necessário seguir um conjunto de métodos e técnicas que possam ajudar a garantir a qualidade do software e diminuir os riscos inerentes ao processo, como desperdício de recursos, descumprimento dos prazos, entre outros.

A engenharia de software surgiu em meados da década de 1970 em uma tentativa de dar um tratamento de engenharia (mais sistemático e controlado) ao desenvolvimento de sistemas de software complexos e oferecer mecanismos para se planejar e gerenciar o processo de desenvolvimento. Antes da engenharia de software, este processo era “artesanal”, no sentido em que cada software era único, desenvolvido de uma maneira diferente e o programador seguia métodos próprios adquiridos ao longo de sua experiência pessoal (PFLEEGER, 2004).

Neste trabalho é utilizado o conceito mais geral de software, que compreende todo o conjunto de programas, procedimentos, dados e documentação associados a um sistema de computador, e não somente ao programa em si. Já um sistema é um conjunto de entidades, um conjunto de atividades, a descrição das relações entre entidades e atividades e a definição das fronteiras do sistema (PFLEEGER, 2004).

Bauer (1972) conceitua a engenharia de software como: "O estabelecimento e uso de sólidos princípios de engenharia para que se possa obter um software economicamente viável, que seja confiável e que funcione eficientemente em máquinas reais". Atualmente, notam-se dificuldades no desenvolvimento de softwares, tais como: uma crescente complexidade dos problemas a serem resolvidos pelos softwares; as estimativas de prazo e de custo de desenvolvimento, frequentemente, são imprecisas; a produtividade das pessoas da área de software não tem acompanhado a demanda por seus serviços e a qualidade de software, muitas vezes, é inferior à adequada, resultando na insatisfação do usuário.

Segundo Pressman (1995), a chave para vencer esses problemas e dificuldades é a larga utilização de uma abordagem de engenharia ao desenvolvimento de software, aliada a

uma contínua melhoria das técnicas e ferramentas, no intuito, também, de melhorar a produtividade da equipe.

A engenharia de software utiliza um conjunto de quatro elementos fundamentais na solução de um problema: métodos, ferramentas, procedimentos e paradigmas. Os **métodos** são técnicas formais para produzir um determinado resultado, eles detalham "como fazer" para se construir o software. As **ferramentas** são instrumentos ou sistemas que proporcionam apoio automatizado ou semi-automatizado para realizar uma determinada tarefa da melhor maneira (métodos). Os **procedimentos** são uma combinação de ferramentas e métodos que, em harmonia, produzem um resultado específico. Por fim, um **paradigma** representa uma abordagem ou estilo em particular para a construção de um software, possibilitando, assim, um processo de desenvolvimento claro e eficiente, visando garantir ao desenvolvedor e seus clientes a produção de um software de qualidade (PFLEEGER, 2004).

Destacam-se alguns paradigmas: Ciclo de vida clássico (*Waterfall*); o modelo incremental; a prototipação; o modelo espiral, entre outros. Conforme Pressman (1995), um modelo de processo para engenharia de software é escolhido de acordo com a natureza do projeto e da aplicação dos métodos e ferramentas disponíveis e dos resultados que são esperados.

Um grande número de diferentes modelos de processos para engenharia de software já foram propostos, cada um mostrando pontos fortes e fracos, mas todos eles trazem uma série de etapas genéricas em comum. De acordo com Pfleeger (2004), o desenvolvimento de software inclui as seguintes atividades:

- Análise e definição dos requisitos;
- Projeto do sistema;
- Projetos do programa;
- Escrever os programas (implementação do programa);
- Teste das unidades;
- Teste de integração;
- Teste do sistema;
- Manutenção.

Os designers de artefatos digitais são engenheiros de software, mas cada designer pode se especializar em um aspecto específico do desenvolvimento. Assim, uma equipe de desenvolvimento pode ter um ou mais analistas de requisitos para trabalhar com o cliente, especificando em requisitos aquilo que o usuário quer. Uma vez que os requisitos são conhecidos e documentados, os analistas trabalham com os projetistas para gerar uma descrição, no nível do sistema, do que o sistema deve fazer. Por sua vez, os projetistas trabalham com os programadores para descrever o sistema, de modo que os programadores possam gerar as linhas de código que efetivamente constroem o que foi especificado nos requisitos (PFLEEGER, 2004).

Depois que o código é gerado, ele deve ser testado. Geralmente, o primeiro teste é feito pelos próprios programadores. Algumas vezes outros testadores são utilizados para

ajudar a encontrar defeitos que os programadores não identificaram. Quando as unidades de código são integradas em grupos funcionais, uma equipe de testadores trabalha com a equipe de implementação para verificar se ele funciona de maneira adequada e de acordo com as especificações, uma vez que o sistema foi construído a partir da integração das partes. Quando a equipe de desenvolvimento está satisfeita com a funcionalidade e a qualidade do sistema, a equipe de teste e o cliente trabalham juntos para verificar se o sistema completo está de acordo com as especificações do cliente. Em seguida, os instrutores treinam os usuários do sistema.

No caso da plataforma T-CADE, essas funções se confundem, uma vez que o cliente foi o desenvolvedor em outras etapas do projeto e atualmente é também o principal testador e usuário, tornando todo o processo de desenvolvimento muito mais dinâmico a partir das constantes sugestões e reavaliações em todos os módulos do código.

Segundo (PFLEEGER, 2004), a qualquer tempo o projeto do sistema pode ser modificado, seja pela equipe de desenvolvimento, seja pelos clientes, em função dos requisitos levantados não refletirem as aspirações dos clientes, ou mesmo um projeto inicial se mostrar inviável na hora da implementação do código.

A engenharia de software utiliza seu conhecimento sobre computadores e computação para ajudar a resolver problemas. Os projetos não são desenvolvidos a partir do nada. É preciso considerar a interação do software com o hardware existente, com os usuários, com outras tarefas de software, com o sistema operacional, banco de dados já existentes ou, até mesmo, com outros softwares similares. Consequentemente, é fundamental estabelecer o contexto de cada projeto, conhecendo suas fronteiras, o que está incluído no projeto e o que não está (PFLEEGER, 2004).

2.4.1 INTER-RELAÇÃO ENTRE SISTEMAS COMPUTACIONAIS

O conceito de fronteira é importante quando se fala de sistemas computacionais inter-relacionados. Pouquíssimos sistemas são independentes de outros sistemas. Na verdade, é muito frequente um sistema conter outros subsistemas, cada um deles responsável pela execução de uma determinada função. As interdependências podem ser complexas. Devem-se reconhecer as fronteiras de um sistema, de tal modo que a solução não se sobreponha aos subsistemas que interagem com o sistema em questão. Uma vez definida a fronteira do sistema, ou seus limites de atuação, fica fácil definir o que está dentro e o que está fora dele, assim como o que atravessa sua fronteira e, por fim, solucionar o problema.

Decompor um sistema maior em seus subsistemas menores é importante, uma vez que resolver o sistema maior como um todo pode ser tarefa árdua. Dividindo o problema em sistemas menores, o grau de complexidade exigido para se encontrar a solução daquele sistema específico é menor, facilitando a solução. Da mesma forma, o sistema maior se torna cada vez mais complexo na medida em que vão sendo acrescentados novos sistemas menores responsáveis por tarefas específicas (PFLEEGER, 2004).

Por exemplo, pensar no corpo humano como um único sistema é muito mais complexo do que dividir suas funções vitais em sistemas menores, como o sistema circulatório, o

sistema respiratório, o sistema nervoso, etc. Cada um destes sistemas menores pode, ainda, ser composto por outros sistemas ainda mais simples até que o problema esteja em um nível de complexidade que possa ser resolvido. “Reconhecer que um sistema contém outros é importante, pois isso reflete o fato de que um objeto ou atividade em um sistema é parte dos demais.” (PFLEEGER, 2004).

Essa ideia pode ser utilizada ao se construir um sistema compuatacional para substituir uma versão mais antiga. Construir o novo sistema computacional com uma série gradativa de sistemas intermediários ajuda muito na construção e síntese do grande sistema. Uma abordagem de desenvolvimento incremental pode incorporar uma série de estágios. Essa abordagem permite, de forma simultânea, analisar o sistema de duas maneiras diferentes: estática e dinamicamente. “A visão estática nos diz como o sistema está funcionando hoje, enquanto a visão dinâmica nos mostra como o sistema está mudando para o que ele, finalmente, se tornará. Uma visão completa outra” (PFLEEGER, 2004).

Ao invés de ir direto do sistema “A” para o sistema “B”, em um único passo, o que pode acarretar em incompatibilidades não previstas no projeto do sistema B, que seriam de difícil identificação e solução, pode-se fazer esta transição aos poucos, substituindo-se os sistemas internos, verificando e solucionando os erros e incompatibilidades daquele sistema específico para que o todo continue funcionando. Sendo assim, o desenvolvimento de sistemas pode, inicialmente, incorporar um conjunto de mudanças a um sistema já existente buscando adaptá-lo às do novo conjunto de alterações mais significativas a fim de gerar um esquema de projeto completo, em vez de tentar ‘passar do presente para o futuro’ de uma só vez. O processo é repetido para cada sistema menor até que o sistema maior esteja todo modificado e reflita o projeto do novo sistema.

Esta transição entre o sistema atual e o novo não é um processo trivial. Dificilmente os designers conseguem de antemão prever todas as implicações e modificações necessárias a tal mudança. Após uma definição inicial dos requisitos, é fundamental o uso de protótipos para avaliar e validar as decisões iniciais para que, a partir daí, se possa prosseguir ou redefinir o projeto. Além de auxiliar a especificação de requisitos, o uso de protótipos pode servir de base para subsidiar tomadas de decisão e como forma de ganhar experiência prática.

2.4.2 PROTOTIPAGEM

O conceito de protótipos já foi abordado anteriormente no item 2.3.2.2 (cenários de uso da engenharia de usabilidade). A prototipagem tem influência em duas atividades do processo de engenharia de software, na atividade de identificação de requisitos e na atividade de validação de requisitos. A experiência dos usuários analisarem o funcionamento do sistema computacional poderá traduzir-se em novos requisitos (atividade de identificação requisitos) e poderá, igualmente, revelar a correção ou incorreção dos requisitos propostos (atividade de validação de requisitos). Nesse processo de fases e validação dos protótipos, torna-se possível efetuar o treinamento dos usuários e também realizar testes – ambos ocorrem gradativamente, ao mesmo passo do desenvolvimento do protótipo.

Quando os usuários ou clientes expressam suas necessidades aos designers em termos de requisitos do programa, estes requisitos nem sempre são claros ou facilmente quantificáveis, como por exemplo: - O programa deve ser “fácil de usar”. Cabe aos designers traduzir estas aspirações dos clientes em características de projeto que satisfaçam a necessidade dos mesmos. Outro motivo para recorrer à prototipagem é que, geralmente, os clientes não conseguem especificar o que pretendem, mas, perante um sistema computacional e após uma breve utilização, facilmente especificam o que não pretendem. A experiência permite concluir que o sistema final será tanto melhor quanto mais iterativo for o processo de desenvolvimento do protótipo (PREECE, ROGERS e SHARP, 2002). O protótipo permite demonstrar conceitos, opções de design, aumentar o conhecimento sobre os problemas e sobre as possíveis soluções.

Um protótipo deve ser empregado como um instrumento de análise, com finalidade de superar as dificuldades de comunicação existentes entre o analista e o usuário do sistema, esta técnica pode ser empregada em pequenos ou grandes sistemas computacionais onde os requisitos não estão claramente definidos. A análise dos requisitos gera uma lista de itens desejados, mas talvez alguns deles não sejam factíveis, considerando-se os recursos do projeto, não sejam realistas ou mesmo não solucionem o problema inicial do cliente. Com a prototipagem pode-se investigar se os requisitos de projeto estão sendo atendidos. Preece, Rogers e Sharp (2002) defendem que uma cultura de efetuar prototipagem origina uma cultura de busca pela inovação.

Os protótipos podem ser classificados em protótipos de baixa fidelidade ou de alta fidelidade (PREECE, ROGERS e SHARP, 2002). Os protótipos de baixa fidelidade não se assemelham ao produto final. Eles são úteis à exploração e testes na fase inicial de desenvolvimento do sistema. São protótipos simples, baratos e de fácil produção e alteração, facilitando deste modo a exploração e validação de ideias. Um exemplo muito usado neste tipo de protótipo é na definição de interfaces gráficas, muito cedo no projeto e com baixíssimos custos, protótipos feitos em papel podem definir ou descartar opções de *Layout*, dando uma boa noção da organização e relação entre os objetos da interface.

Alguns aspectos positivos do uso de protótipos de baixa fidelidade são:

- Custos reduzidos;
- Reduzido tempo de desenvolvimento;
- Eficiente para definição de requisitos;
- Facilita múltiplos testes de opções de design.

Por outro lado, como aspectos negativos têm-se:

- Reduzida ou nenhuma utilidade após a definição do documento de requisitos (ex: na fase de testes do sistema final);
- Definição incompleta do esquema navegacional;
- Permite testes limitados (ex: usabilidade).

Os protótipos de alta fidelidade são aqueles que se assemelham e utilizam as mesmas técnicas e materiais que o sistema final (PREECE, ROGERS e SHARP, 2002). São os protótipos indicados quando a intenção é a apresentação do sistema ou o teste de problemas técnicos. A prototipagem de alta fidelidade também tem como característica o fato do protótipo ter algumas de suas funcionalidades limitadas.

Os aspectos positivos da prototipagem de alta fidelidade são, entre outros:

- Possui funcionalidades semelhantes às do sistema final;
- Permite uma definição completa do esquema navegacional;
- Permite um elevado grau de interatividade com os usuários;
- Permite a exploração e testes diversos com um elevado grau de realismo;
- O protótipo é um documento de requisitos (registro inicial do projeto);
- Facilita a visualização do sistema final por ser um protótipo realista;
- Permite a reutilização do protótipo no sistema final.

Como aspectos negativos é possível citar:

- Elevados custos de desenvolvimento (em relação ao de baixa fidelidade);
- Elevado tempo de desenvolvimento (em relação ao de baixa fidelidade);
- Risco de um *feedback* inadequado dos usuários, por exemplo, preocupação com detalhes menores (fonte, alinhamento, etc.) ao invés da estrutura do aplicativo;
- Resistência a mudanças por parte dos projetistas;
- Risco de passar uma falsa impressão de que falta pouco para a conclusão da implantação do produto;
- Risco em sessões de testes, por exemplo, um único engano de digitação no código pode paralisar todo o programa.

A escolha entre protótipos de baixa ou alta fidelidade depende dos objetivos a serem alcançados e dos tipos de requisitos que estão sendo avaliados.

Pfleeger (2004) coloca que a prototipagem poderá ser implementada segundo dois métodos: a evolutiva e a descartável. Ambas as técnicas são chamadas de prototipagem rápida, porque elas constroem partes do sistema proposto, a fim de determinar a necessidade e a viabilidade dos requisitos ou o quanto eles são desejáveis. Na prototipagem rápida, as opções são avaliadas antes que o projeto seja criado; o propósito do protótipo rápido é ajudar a entender os requisitos e a decidir sobre o projeto final.

Um protótipo descartável é um software desenvolvido para aprender mais sobre um problema, ou para explorar a viabilidade das possíveis soluções ou verificar o quanto elas são desejáveis. Um protótipo descartável é exploratório, e não se pretende utilizá-lo como uma parte real do sistema final. Protótipos descartáveis (*throw-away*), após sua utilização, são deixados de lado ou passam a servir apenas para consultas.

Neste tipo de prototipagem, o processo de desenvolvimento do software real é totalmente independente do processo utilizado para a elaboração do protótipo. Quando de

baixa fidelidade, serve para especificar os requisitos e escolher opções de layout, quando de alta fidelidade serve para testar a funcionalidade de algoritmos e ferramentas separadamente, permitindo que seus problemas sejam solucionados antes de incorporá-las ao sistema final.

A própria intenção dos protótipos é ajudar a clarear ou definir melhor os requisitos do software a ser desenvolvido. Esse fato se evidencia mais na prototipagem descartável, onde os requisitos não precisam estar tão claros inicialmente. Por outro lado, quando determinada necessidade dos usuários ou funcionalidade do sistema é totalmente conhecida, o protótipo gerado se torna mais robusto e consistente, favorecendo, principalmente, a prototipagem evolutiva (KOTONYA e SOMMERVILLE, 1998).

Um protótipo evolutivo é desenvolvido para se aprender mais sobre um problema e se ter a base de uma parte ou de todo o sistema final. Protótipos evolutivos ou evolucionários (*evolutionary*) são criados nas fases iniciais do projeto e refinados ao longo do decorrer do processo de desenvolvimento do software. Incrementos de funcionalidade são incorporados ao protótipo, que, tendo sua fidelidade gradualmente aumentada, se torna o software final. Assim, os processos de desenvolvimento do protótipo e do sistema real são essencialmente os mesmos (KOTONYA e SOMMERVILLE, 1998).

Neste sentido, a prototipagem evolutiva vem ao encontro ao conceito plataforma de desenvolvimento utilizado neste trabalho. O desenvolvimento de uma interface 3D para o T-CADE deve ser entendida como um protótipo evolutivo, onde melhorias e aperfeiçoamentos constantes serão implementados na medida em que novas necessidades e novos requisitos surgirem das avaliações e interações com este protótipo.

O projeto deste protótipo deve considerar os conceitos da interação entre sistemas e subsistemas vistos anteriormente, o que corrobora o uso da programação orientada a objetos utilizada neste trabalho.

Finalmente, esta visão geral do desenvolvimento de um artefato digital sob a ótica da Engenharia de Software mostra a importância do projeto no processo de desenvolvimento e da participação constante de todos os agentes envolvidos neste desenvolvimento, seja o cliente, designer, programador, testador ou usuários.

2.4.3 PROGRAMAÇÃO ORIENTADA A OBJETO

De acordo com Pfleeger (2004), a programação orientada a objetos (POO) é uma abordagem para desenvolvimento de software que organiza os problemas e suas soluções como um conjunto de objetos distintos. Em alguns contextos, prefere-se usar modelagem orientada ao objeto, ao invés de programação, uma vez que é uma estratégia para a solução de problemas de design e não apenas de problemas de estruturação de códigos de softwares.

A POO é um dos muitos paradigmas de modelagem de dados. É importante diferenciar a orientação a objetos da programação estruturada, outro paradigma muito usado no desenvolvimento de softwares. A POO é um estilo de programação baseado em objetos, e não em procedimentos e funções, utilizados na programação estruturada. Na POO usam-se objetos

com estruturas fechadas de dados, e os próprios objetos resolvem seus problemas internamente. A estrutura e o comportamento dos dados estão incluídos na sua representação.

Conforme Booch (2007), ao utilizar a POO, desenvolve-se programas menos propícios a mudanças e com economia de código. Aumenta-se a confiança na correção do software através da separação do problema em problemas menores. Conforme o autor, a POO reduz os riscos inerentes ao desenvolvimento de sistemas computacionais complexos.

Alguns conceitos básicos da POO merecem destaque:

- Classes- uma classe é uma estrutura que abstrai um conjunto de objetos com características similares. É uma generalização de um objeto. Por exemplo, as pessoas são diferentes, mas tem características físicas em comum (dois olhos, nariz, boca) e executam as mesmas tarefas (comer, dormir, falar), assim, pode-se generalizar uma classe “humanos” com todas estas características. Novas classes podem ser definidas a parir de classes ancestrais.
- Objetos- Um objeto é uma instância de uma classe. No exemplo acima, pode-se criar um objeto “João”, com todas as características da classe “humanos” (um indivíduo de uma espécie).
- Atributos- São as características de um objeto. São os dados referentes a uma classe. Por sua vez, os atributos podem receber valores. Por exemplo, um dos atributos de uma pessoa é a cor dos cabelos, e um dos valores possíveis é o castanho.
- Métodos- É a descrição das ações dos objetos, de como seus atributos são manipulados ou o que os objetos podem fazer. Por exemplo, as pessoas podem ver, sorrir, chorar, comer, etc. Estes são métodos da classe pessoa.

Além destes conceitos, existem algumas características da orientação a objetos que devem ser evidenciadas:

- Herança- É a capacidade de herdar as características de uma classe ancestral. Por exemplo, pode-se criar uma classe “Chineses” derivada da classe “humanos” que contenha todas as características da classe ancestral. Pode-se acrescentar mais algumas características específicas desta classe ou definir valores específicos para os atributos existentes.
- Polimorfismo- é a capacidade de executar um determinado método de formas diferentes, conforme o contexto. Por exemplo, o método “falar” da classe “humanos” é herdado na classe “chineses” e quando executado falará a língua chinesa. O mesmo método “falar” é também herdado na classe “Brasileiros” e quando executado falará a língua portuguesa.
- Encapsulamento- é a capacidade de esconder (tornar privado) ou mostrar (tornar público) os atributos e métodos de um objeto. Esta característica permite simplificar o uso de um determinado objeto e aumentar a proteção do código (O'DOCHERTY, 2005). Por exemplo, as pessoas se alimentam, todo o processo digestivo e a absorção dos nutrientes são resolvidos internamente (métodos privados), tudo o que elas sentem

é fome ou saciedade (métodos públicos). O encapsulamento é uma das maneiras de proteger o programador dele mesmo.

- Associação- é o mecanismo pelo qual um objeto utiliza os recursos de outro. Por exemplo, um dos atributos dos humanos são os olhos. Os olhos humanos podem ser definidos como objetos (de uma classe “olhos”) com seus próprios atributos e métodos.

Em função destas características, o uso de POO traz alguns benefícios no desenvolvimento de artefatos digitais. Os conceitos de estrutura de classes e herança forçam o desenvolvedor a fazer uma análise da estrutura de dados mais criteriosa e planejada, isto assegura uma codificação mais precisa e facilita a manutenção. Uma classe define e manipula apenas os seus próprios dados, com a possibilidade de tornar públicos ou esconder determinados dados e métodos, isto evita o acesso acidental de dados por outras partes do código, aumentando a segurança do sistema computacional e evitando erros inesperados. A definição de classes é reutilizável, não apenas no próprio sistema computacional, mas também na criação de outros sistemas, o que reduz o tempo de desenvolvimento destes.

Portanto, uma abordagem orientada a objetos permite uma melhor estruturação dos dados, dividindo problemas maiores em problemas menores, permitindo o reuso das classes e facilitando a manutenção e, caso necessário, a mudança da estrutura de classes de maneira rápida e eficiente.

2.5 COMPUTAÇÃO GRÁFICA

De acordo com Persiano e Oliveira (1989), a computação gráfica (CG) é a área da ciência da computação que estuda a geração, manipulação e interpretação de modelos e imagens de objetos utilizando computador. A CG é utilizada em uma variedade de áreas como design, engenharia, arquitetura, publicidade, educação, ciências, medicina, pesquisa militar, física, matemática, cinema, jogos, etc. De um modo geral, a CG pode ser considerada como qualquer atividade computacional que resulta em imagens gráficas. Mais especificamente, a CG em 3D (em contraste com CG em geral) cria, através de cálculos matemáticos, uma representação geométrica de dados tridimensionais armazenados no computador, a qual é mostrada através de imagens digitais bidimensionais.

Atualmente, a CG é altamente interativa: o usuário controla o conteúdo, a estrutura e a aparência dos objetos e suas imagens visualizadas na tela, usando dispositivos como o teclado e o mouse. Entretanto, até o início dos anos 80, a CG era uma disciplina restrita e altamente especializada. Devido, principalmente, ao alto custo do hardware, poucos aplicativos exploravam gráficos. O advento dos computadores pessoais de baixo custo, como o PC-IBM e o Macintosh Apple, com terminais gráficos de varredura (*Raster Graphics Displays*) ou displays gráficos, popularizou o uso de gráficos na interação Homem-computador.

Os displays gráficos de baixo custo possibilitaram o desenvolvimento de inúmeros aplicativos de custo acessível e fáceis de usar, que dispunham de interfaces gráficas, processadores de imagem, programas de desenho (imagens e vetoriais), entre outros. Atualmente a CG não é encontrada apenas nos softwares científicos de grandes laboratórios,

centros de pesquisas ou criadas por grandes empresas de softwares comerciais, é parte essencial de qualquer interface com o usuário, de fácil criação e indispensável para a visualização de dados em 2D e 3D.

Considerando genericamente a CG como a visualização da Informação, percebe-se a necessidade de especificar este conceito. A computação gráfica trabalha basicamente com dois tipos de informação: visual e descritiva. Informação visual é, por exemplo, a imagem vista na tela do computador. Informação descritiva refere-se ao modelo matemático que representa os objetos visualizados.

Pode-se relacionar a computação gráfica com três subáreas de conhecimento (PERSIANO e OLIVEIRA, 1989):

• **Processamento de imagens**: envolve as técnicas de transformação de imagens, em que tanto a imagem original, quanto a imagem resultado apresentam-se sob uma representação visual (geralmente matricial). Estas transformações visam melhorar as características visuais da imagem (aumentar contraste, foco, ou mesmo diminuir ruídos e/ou distorções). As imagens produzidas por esta subárea são armazenadas em *raster-files*, ou arquivos *bitmap.* O Processamento de Imagens abrange operações que são realizadas sobre imagens cujos resultados também são imagens. Os principais softwares comerciais nesta área são o Adobe Photoshop ®(ADOBE s.d.) e o Corel Photo-Paint ® (COREL s.d.), tendo ainda o software livre Gimp (GIMP s.d.) de código aberto (Open Source).

• **Síntese de imagens**: área que se preocupa com a produção de representações visuais a partir das especificações geométricas e visuais de seus componentes. É frequentemente confundida com a própria computação gráfica. As imagens produzidas por esta subárea são geradas a partir de dados mantidos em *display-Files*. A síntese de imagens parte da descrição matemática de objetos, tais como segmentos de reta, polígonos, poliedros, esferas, entre outros, produzindo imagens que atendem a certas especificações para representação e visualização de modelos e que, em última instância, podem ser visualizados em algum dispositivo (terminal de vídeo, plotter, impressora). As ferramentas CAD e CAE, bem como softwares de animação e jogos 3D, são os principais representantes desta categoria.

• **Análise de imagens**: área que procura obter a especificação dos componentes de uma imagem a partir de sua representação visual. Ou seja, a partir da informação pictórica da imagem (a própria imagem!), produz-se uma informação não pictórica da mesma (por exemplo, as primitivas geométricas elementares que a compõem). A área de visão computacional abrange as operações de análise dos objetos contidos na imagem e a geração de modelos matemáticos desses objetos. É o processo inverso da síntese de imagens, muito utilizado na visão robótica, na detecção de movimento, análise de imagens de satélite e na medicina (análise de imagens de ressonância magnética).

Na última década, somou-se a esse contexto a área de Visualização de Dados, também chamada visualização científica (MINGHIM e OLIVEIRA, 1997), (SCHROEDER, MARTIN e LORENSEN, 1997) que usa técnicas de CG para representar informação, de forma a facilitar o entendimento de conjuntos de dados numéricos de alta complexidade. Exemplos de

áreas de aplicação são: visualização de imagens médicas, meteorologia, dados financeiros, dinâmica dos fluidos, entre outras. Em todos estes exemplos, a representação gráfica (superfícies, partículas, ícones) é gerada automaticamente a partir do conjunto de dados. Ao usuário cabe definir parâmetros e atributos da imagem para melhor "navegar" em seu conjunto de dados. Sendo assim, a visualização de dados partilha de características da síntese, do processamento e da análise de dados.

Desta maneira, os artefatos digitais das diversas subáreas devem ser tratados de forma diferente de acordo com o objetivo a ser alcançado. Esta diferença de objetivos vai levar a modelos de trabalho diferentes. No processamento de imagens, usa-se o modelo matricial, onde os dados do arquivo são descritos em termos da posição dos pixels em uma matriz e suas propriedades. A síntese de imagens, geralmente, se baseia no modelo de objetos vetoriais. Neste modelo, os objetos são armazenados apenas a partir da descrição das coordenadas de seus vértices, sejam elas espaciais ou planas (três ou duas dimensões, respectivamente). Dessa maneira, utiliza-se um sistema de coordenadas cartesiano, onde os objetos podem ser escalados, rotacionados e transladados com maior liberdade. A visualização destes objetos se dá através da geração de imagens baseada em um modelo matricial, mas seus cálculos e sua manipulação são realizados a partir do modelo vetorial.

O modelo matricial utiliza uma matriz de dados para armazenar a informação de cor em cada ponto da imagem, onde o sistema de coordenadas é uma grade de números inteiros que descrevem a posição do pixel na matriz. Portanto, no modelo matricial não há distinção dos objetos contidos na imagem. Além disso, armazenar a matriz que contém a imagem geralmente exige muito mais memória que armazenar sua descrição vetorial. A diferença de modelos fica explícita quando se apresenta o paradigma dos quatro universos (VELHO e GOMES, 1997), que ajuda a entender este processo de sair do mundo físico, onde os sinais são contínuos, e ir para o mundo digital, onde a configuração é discreta. A Figura 8 ilustra a inter-relação entre estes quatro universos.

Figura 8: O Paradigma de abstração dos quatro universos.
Fonte: Adaptado de Velho e Gomes, 199).

Em computação gráfica, um paradigma de abstração que se aplica em geral, consiste em estabelecer quatro universos (conjuntos). No universo físico, estão os objetos a serem representados. No universo matemático, são formuladas descrições abstratas desses objetos. O universo de representação vai permitir trazer essas descrições abstratas para o mundo digital, e é onde se dará a discretização dos sinais contínuos. O universo de implementação é onde é feita a codificação do sinal discretizado na memória do computador através de uma estrutura de dados. O termo discretizar é aqui utilizado da mesma forma utilizada na matemática para designar a individualização (discretização) de uma distribuição contínua em unidades individuais (discretas), como por exemplo, uma imagem (contínua) sendo representada por uma grade de pixels (discreta).

Assim, para estudar um determinado fenômeno ou objeto do mundo real no computador, é preciso associá-lo a um modelo matemático e, então, encontrar uma representação discreta para esse modelo, que pode ser implementada no ambiente computacional. A partir deste modelo, o objeto virtual pode ser manipulado e modificado, para então ser novamente representado descritivamente para a construção no mundo real.

O conceito de síntese de imagens é o que melhor se aplica neste trabalho com relação a computação gráfica, onde o objetivo inicial da nova interface é representar (através de imagens) em um monitor bidimensional o modelo matemático tridimensional e a partir destas imagens, interagir com o modelo virtual.

2.6 BIBLIOTECAS GRÁFICAS

Frequentemente as pessoas dependem de outras para realizar funções que não podem fazer por si mesmas, ou não têm permissão para isso, por exemplo, ao abrir uma conta bancária. Da mesma forma, praticamente todos os softwares dependem de outros softwares para executar determinadas tarefas. Para isto, o software que está requisitando o serviço (que no exemplo bancário corresponde ao cliente) usa um conjunto de funções (formulário de abertura de conta) que foi definido pelo software que prestará o serviço (banco). A este conjunto de funções chamamos API (*Application Programming Interfaces*) ou interface de programação da aplicação.

A grande maioria dos softwares depende de APIs para o sistema operacional que os suporta para executar as mais básicas funções, como acessar o sistema de arquivos ou abrir uma caixa de diálogo por exemplo. Essencialmente, a API de um programa define a maneira apropriada para o desenvolvedor requisitar um determinado serviço daquele programa. Neste caso, ela serve como uma interface entre o desenvolvedor e o programa. Esta interface com os dispositivos gráficos refere-se ao protocolo de conversão de comandos independentes de dispositivos (comandos da API) para comandos do próprio equipamento (Hardware).

No caso específico da computação gráfica, as APIs gráficas (ou bibliotecas gráficas) são utilizadas como uma interface entre o software e o hardware gráfico para uma melhor visualização dos dados. As primeiras APIs gráficas especializadas e padronizadas foram criadas com o objetivo de eliminar os principais problemas existentes na área de computação gráfica (WRIGHT, LIPCHAK e HAEMEL, 2007). Uma vez que os programas da década de

1980 utilizavam comandos específicos do próprio terminal, escreviam diretamente na memória gráfica e utilizavam rotinas fornecidas pelo fabricante do hardware. Como os desenvolvedores tinham acesso direto ao hardware para o qual eles estavam programando, como por exemplo, dispositivos de entrada (como mouse e teclado), placas de vídeo e controladores de vídeo, eles podiam fazer praticamente qualquer operação gráfica. Mas esta metodologia gerava problemas para transferência de programas gráficos de uma instalação para outra, ou seja, os programas eram dependentes dos equipamentos.

Com a popularização dos Computadores pessoais (PC – *Personal Computers*) e, consequentemente, a maior competição entre os fabricantes de hardware, havia centenas de possíveis configurações diferentes para um PC comum. Com o objetivo de minimizar o problema, o programador precisava desenvolver algoritmos específicos para uma interminável lista de hardware. Com isso, mais tempo de desenvolvimento era necessário, o que deixava cada vez menos tempo para trabalhar nas funções próprias do aplicativo. Logo, houve necessidade de elaborar programas gráficos compatíveis com qualquer equipamento e com ela o surgimento das bibliotecas gráficas.

A Figura 9 mostra o papel da biblioteca gráfica nas diversas interfaces de um sistema. Nota-se que, ao longo do percurso da informação, existem, entre outras, as seguintes interfaces:

1. Do usuário com os dispositivos de hardware (mouse, teclado, monitor);
2. Dos dispositivos de hardware com as funções gráficas (CPU, placa gráfica, memória);
3. Das funções gráficas com o software;
4. Do software com os dados.

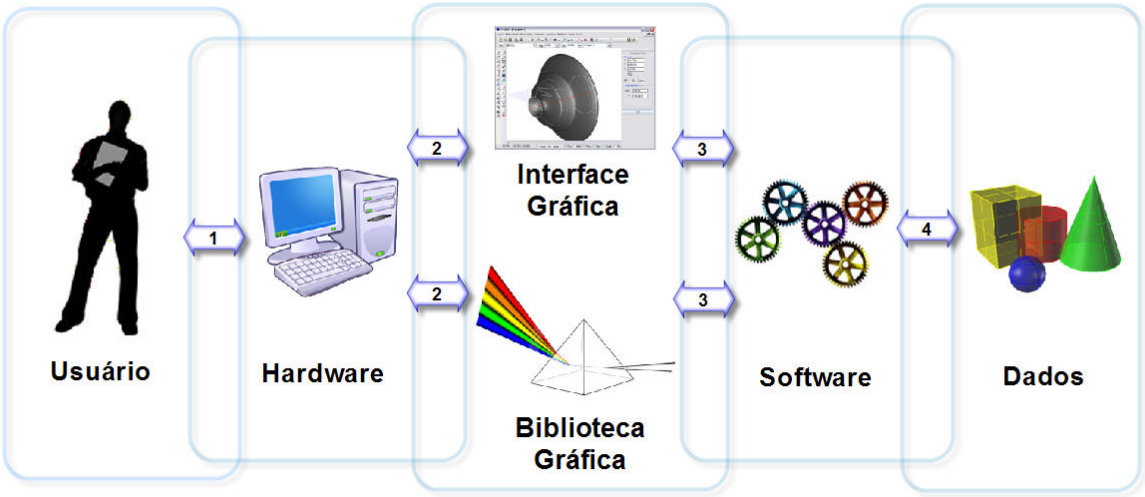


Figura 9: Algumas Interfaces de um Sistema.
Fonte: O Autor.

As bibliotecas gráficas são as melhores ferramentas para possibilitar generalizar o processo de programação gráfica para diferentes hardwares. Por outro lado, as bibliotecas gráficas e seus algoritmos especializados são constantemente atualizados pelos consórcios

mantenedores, utilizando todo o potencial da tecnologia disponível. Sendo assim, uma biblioteca gráfica ajuda a renderizar (mostrar/criar) uma imagem digital em um monitor qualquer, com qualquer placa gráfica e da melhor forma possível.

A programação gráfica envolve, normalmente, funções que podem ser simples, usando puramente os recursos da CPU (*Central Processing Unit* em inglês, ou Unidade Central de Processamento), ou mais complexas, envolvendo aceleração de hardware com versões otimizadas de funções que utilizam também os recursos da placa Gráfica – GPU (*Graphics Processing Unit)* ou Unidade de processamento gráfico. GPU é um processador da placa gráfica específico para os cálculos de visualização. As bibliotecas gráficas também se tornaram vitais para os fabricantes de hardware, que podem determinar especificações que permitam aos desenvolvedores ter acesso ao hardware de uma forma abstrata enquanto utilizam todos os recursos especiais de uma placa gráfica.

Na década de 1980, os padrões gráficos GKS (*Graphical Kernel System)* (ANSI 1985a) e o PHIGS (*Programmer's Hierarchical Interactive Graphics System)* (ANSI 1985b) tiveram um importante papel, inclusive ajudando a estabelecer o conceito de uso de padrões mesmo fora da área gráfica, tendo sido implementados em diversas plataformas. O GKS suporta um conjunto de primitivas gráficas inter-relacionadas, tais como: desenho de linhas, polígonos, caracteres, etc., bem como seus atributos. Mas não suporta agrupamentos de primitivas hierárquicas de estruturas 3D. O PHIGS (e seu descendente, PHIGS+) é uma evolução do GKS permitindo manipular e desenhar objetos 3D encapsulando descrições de objetos e seus atributos. Uma desvantagem do PHIGS e PHIGS+ (e GKS) é que eles não têm suporte a recursos avançados como mapeamento de textura.

Nenhuma destas APIs, no entanto, conseguiu ter grande aceitação (Segal e Akeley, 1994). A biblioteca OpenGL é o sucessor da biblioteca gráfica conhecida como IRIS GL, desenvolvida pela *Silicon Graphics* como uma interface gráfica independente de hardware (Kilgard, 1994) e independente de plataforma (sistema operacional). A maioria das funcionalidades da IRIS GL foi removida ou reescrita na OpenGL e as rotinas e os símbolos foram renomeados para evitar conflitos (todos os nomes começam com "gl" ou "GL_). Em 1992, foi formado o ARB (*OpenGL Architecture Review Board*), um consórcio independente que administra o uso da OpenGL, formado por diversas empresas da área, inclusive a Microsoft®, que alguns anos depois, saiu do consórcio e criou sua própria Biblioteca gráfica (Direct3D, atualmente,fazendo parte do DirectX).

Nos dias atuais, a OpenGL (em sua versão 3.0) e a Direct3D (em sua versão 11) são as APIs mais usadas em programas CAD, realidade virtual, visualização científica, visualização da informação, simuladores de vôo e vídeo games. No entanto, OpenGL é a mais usada em aplicações científicas, CAD e também em outros sistemas operacionais que não o Windows, enquanto que a Direct3D é a mais utilizada em Jogos (somente na plataforma Windows). As duas APIs fornecem suporte para desenhar pontos, linhas, triângulos, quadrados, entre outros, bem como suporte à aplicação de texturas sobre planos. As duas guardam as informações dos objetos no espaço virtual e, a partir da posição da câmera, renderizam os mesmos na tela. Como a maioria das aplicações é dinâmica e realiza mudanças na câmera que devem ser

passadas rapidamente ao usuário, as duas APIs se utilizam das capacidades do Hardware (GPU) para realizar os processamentos dos vértices e faces. Logo, para que se possa trabalhar com objetos tridimensionais renderizados em tempo real, é desejável o uso de uma boa placa de vídeo aceleradora 3D. Atualmente, os melhores PCs disponíveis ao usuário comum já estão equipados com placas de vídeo com um desempenho satisfatório para jogos e aplicativos 3D.

Do ponto de vista do programador para plataforma Windows, a API do Direct3D é mais interessante, pois sua construção é feita a partir de um modelo orientado a objetos e acompanha outras APIs (*DirectDraw*, *DirectMusic*, *DirectPlay*, *DirectSound*, para citar algumas incluídas no DirectX). Enquanto que no OpenGL (apenas a API gráfica), o programador terá que escolher dentre inúmeras funções, o que não é exatamente um problema pois todas as funções têm nomes indicativos. Talvez uma das vantagens esteja nesse ponto, pois para um desenvolvedor que deseja criar sua própria *engine 3D* (gerenciador físico, temporal e organizacional do mundo 3D) e que busca um certo grau de autonomia, o fato de poder controlar exatamente como os objetos serão renderizados pode resultar num ganho de desempenho ou mesmo qualidade visual.

Além destas já citadas, algumas outras bibliotecas gráficas 3D merecem destaque:

- RenderMan (Pixar studios);
- RenderWare (Eletronic Arts);
- Glide API (3dfx – placas gráficas Voodoo);
- QuickDraw 3D (Apple® – lançada em 1995, abandonada em 1998).

Existem também bibliotecas gráficas implementadas em alto-nível que proporcionam funcionalidades adicionais às APIs de baixo-nível, como por exemplo:

- Glscene (OpenGL);
- Quesa (QuickDraw 3D);
- Java 3D (OpenGL/Direct3D);
- OpenGL Performer (OpenGL);
- Mesa3D (OpenGL);

Com o uso de bibliotecas gráficas, além da possibilidade de se poder aproveitar ao máximo o hardware disponível a partir de suas funções otimizadas, o desenvolvedor pode liberar-se da função de criar e otimizar funções gráficas para se concentrar nos objetivos e na funcionalidade do software a ser desenvolvido. Embora as bibliotecas gráficas proporcionem uma maneira fácil e ágil de resolver os problemas gráficos, as mesmas podem limitar a liberdade de programação dos desenvolvedores, como é o caso de bibliotecas proprietárias, por exemplo, a DirectX da Microsoft, que vem sendo a API mais usada na indústria de jogos para criar efeitos realísticos. No entanto, o desenvolvedor não tem acesso aos códigos fonte.

As bibliotecas gráficas de código aberto (*Open Source code*) fornecem (liberam) todo o código, permitindo maior flexibilidade ao desenvolvedor para entender os detalhes dos algoritmos, ou personalizar alguma função para resolver um problema específico. Muitos artigos científicos e excelentes publicações podem ser encontrados utilizando a OpenGl ou

bibliotecas baseadas em OpenGL de código aberto, mas o mesmo não pode ser dito da DirectX. Esta é sem dúvida uma limitação em aprender e utilizar DirectX no meio acadêmico.

A seguir, são apresentadas com mais detalhes as principais bibliotecas utilizadas, a biblioteca DirectX, OpenGL e a biblioteca GLScene.

2.6.1 A BIBLIOTECA DIRECTX

Em meados da década de 1990, a Microsoft® criou suas próprias APIs, como a *DirectSound* (API para som), *DirectPlay* (API para jogos em rede) e Direct3D (API gráfica 3D), entre outras. Mais tarde todas essas APIs foram incorporadas em um único pacote chamado DirectX, que atualmente se encontra na versão 10. A versão 11, que já foi mostrada no evento *Gamefest* 2008 em Seattle, foi lançada para o Windows Vista.

Diferentemente das bibliotecas de código aberto, DirectX é uma API proprietária para aceleração de hardware e é especifica para o sistema operacional Windows. Outra diferença importante é que DirectX não é apenas uma API gráfica como a OpenGL, ela permite as aplicações multimídia além do acesso à GPU, às placas de som (e uma vasta gama de equipamentos de som), a dispositivos como *Joystick*, teclado, mouse, controladores de rede para jogos online (com múltiplos jogadores) e controle sobre os vários formatos de arquivos multimídia.

As principais APIs que fazem parte do pacote DirectX (com suas respectivas funções) são:

- *DirectDraw* – Objetos 2D (hoje incorporada na Direct3D);
- *Direct3D* – Imagens e Objetos 3D;
- *DirectSound* – Som;
- *DirectSound3D* – Som 3D;
- *DirectMusic* – Músicas;
- *DirectPlay* – Rede/Múltiplos Jogadores;
- *DirectInput* – Dispositivos de entrada.

A DirectX opera quase em nível de hardware para praticamente todos os dispositivos. Isto é possível através de uma tecnologia chama COM (*Component Object Model*) e por conjuntos de *drivers* e bibliotecas DLL (*Dynamic Link Libraries*) escritos pela Microsoft® e pelos próprios fabricantes de hardware. Essas bibliotecas são distribuídas em código binário (já compiladas) juntamente com as diretrizes de como utilizá-las. Atualmente, a DirectX já vem incluída na instalação do sistema operacional Windows.

A Figura 10 mostra uma imagem gerada pela DirectX em sua versão 10, imagem obtida no Jogo ‘*Age of Conan: Hyborian Adventures*’.

Figura 10: Qualidade da Imagem da DirectX versão 10.
Fonte:(http://www.gamesforwindows.com/en-US/Games/Pages/AgeofConanHyborianAdventures.aspx)

Existem alternativas ao pacote de APIs da DirectX, algumas são mais completas que outras. Embora não exista uma solução unificada que faça tudo que a DirectX é capaz de fazer, com uma combinação de bibliotecas, é possível implementar uma solução comparável a esta biblioteca com as vantagens de ser multi-plataforma e, frequentemente, de código aberto.

2.6.2 A BIBLIOTECA OPENGL

O desenvolvimento de aplicações baseadas em manipulação direta de objetos virtuais em 3D é ainda uma tarefa complexa e sujeita a erros, mesmo utilizando os diversos recursos disponíveis (bibliotecas, componentes, ferramentas, entre outros) para criação de interfaces gráficas. Um exemplo disto é a correta interpretação das três dimensões do mundo virtual e sua respectiva representação em duas dimensões (monitor). Outro exemplo é a manipulação de um objeto virtual com seis graus de liberdade (6 *degrees of freedom* – 6DOF), a translação e rotação nos três eixos cartesianos, utilizando um mouse com 2 graus de liberdade (2 *degrees of freedom* – 2DOF), movimento do cursor na horizontal e vertical.

Estes problemas complexos de representação e manipulação costumam exigir um grande poder de processamento do hardware. Felizmente, a tecnologia atual já atende tal demanda a custos suportados por um usuário doméstico. A variedade de equipamentos para computação gráfica, como placas gráficas 3D, canetas digitais e monitores, exigem o uso de bibliotecas gráficas para utilizá-los de maneira transparente e eficiente (WRIGHT, LIPCHAK e HAEMEL, 2007).

Um exemplo destas bibliotecas é a OpenGL (Open Graphics Library) (OPENGL s.d.), uma biblioteca gráfica que serve de interface ao hardware gráfico, permitindo que o desenvolvedor crie aplicações 3D interativas, gerando imagens realísticas com performance em tempo real. A biblioteca OpenGL já traz diversas implementações que atendem aos principais requisitos gráficos para aplicativos 3D, incluindo o desempenho gráfico otimizado e o uso dos recursos adicionais de processamento das placas gráficas.

A biblioteca OpenGL é uma biblioteca gráfica de modelagem e visualização 3D que é altamente portável entre diferentes plataformas e com um desempenho gráfico otimizado, uma vez que seus algoritmos foram cuidadosamente criados e desenvolvidos pelas empresas líderes em computação gráfica e animação no mundo (WRIGHT, LIPCHAK e HAEMEL, 2007). A primeira versão da OpenGL foi criada em 1992 e é baseada na biblioteca IRIS GL das estações de trabalho da *Silicon Graphics*.

Atualmente, um consórcio de mais de cem (100) indústrias (Grupo Khronos) é responsável pelo gerenciamento da evolução do OpenGL. Assim como as outras bibliotecas gráficas, a OpenGL oferece uma interface entre o software e o hardware gráfico e é reconhecida como um padrão API para desenvolvimento de aplicações gráficas 3D em tempo real. Sua última versão (versão 3.0) já está implementada nas principais plataformas.

Entre outras vantagens, ela possui uma arquitetura bem definida, bom desempenho, vasta documentação e é portável (está disponível em diversas plataformas). Por ser um padrão destinado somente à renderização (SEGAL e AKELEY, 1996), a biblioteca OpenGL não possui funções para gerenciamento de janelas, interação com o usuário ou arquivos de entrada/saída. Cada plataforma possui suas próprias funções para estes propósitos. A OpenGL pode ser utilizada em qualquer sistema de janelas (exemplo: XWindow System, MSWindows), aproveitando-se dos recursos disponibilizados pelos diversos hardwares gráficos existentes. Esta característica faz da OpenGL a API mais adequada para equipamentos diversos como celulares, PDAs, GPS, etc. que apresentam uma grande variedade de tecnologias de hardware (OPENGL s.d.).

Esta biblioteca gráfica consiste em mais de 250 funções diferentes que podem ser utilizadas para modelar cenas tridimensionais complexas e é bastante utilizada em ferramentas CAD, realidade virtual, simulações e visualizações científicas, programas de modelagem usados para criar personagens e efeitos especiais para o cinema e na indústria de jogos para PC e videogames (OPENGL s.d.).

A biblioteca OpenGL não é uma linguagem de programação, é uma poderosa e sofisticada API para criação de aplicações gráficas. Normalmente se diz que um programa é baseado em OpenGL ou é uma aplicação OpenGL, o que significa que ele é escrito em alguma linguagem de programação (como C, C++, Visual Basic, Object Pascal, etc.) que faz chamada a uma ou mais bibliotecas OpenGL.

A biblioteca OpenGL disponibiliza um controle simples e direto sobre um conjunto de rotinas, permitindo ao programador especificar os objetos e as operações necessárias para a produção de imagens gráficas de alta qualidade. Para tanto, a OpenGL funciona como uma

máquina de estados, onde o controle de vários atributos é realizado através de um conjunto de variáveis de estado (ligado/desligado ou uma lista de opções) que, inicialmente, possuem valores padrão, podendo ser alterados caso seja necessário. Por exemplo, todo objeto será traçado com a mesma cor até que seja definido um novo valor para a variável que controla a cor. Os softwares quando utilizam a biblioteca OpenGL renderizam uma imagem a partir da definição de primitivas e especificação de seus vértices. A Biblioteca OpenGL fornece dez tipos diferentes de primitivas (MARTZ, 2006) para o desenho de pontos, linhas e polígonos interpretados seguindo as seguintes regras:

- GL_POINTS— Utilizada para desenhar pontos. Desenha um ponto para cada vértice especificado;
- GL_LINES— Utilizada para desenhar segmentos de retas não conectadas. Desenha um segmento de reta para cada par de vértices especificados. Se o software especificar “n” vértices, a biblioteca desenhará “n/2” linhas. Se o software especificar um número impar de vértices o último vértice é ignorado;
- GL_LINE_STRIP— Utilizada para desenhar uma linha com uma sequência conectada de segmentos de reta. Desenha uma linha entre o primeiro e o segundo vértice, entre o segundo e o terceiro, o terceiro e quarto e assim por diante. Se o software especifica “n” vértices, será desenhada uma linha com “n-1” segmento de retas;
- GL_LINE_LOOP— Utilizada para desenhar uma poligonal fechada. Desenha esta primitiva exatamente como a GL_LINE_STRIP e por fim fecha a poligonal desenhando um segmento de reta entre o último e o primeiro vértice;
- GL_TRIANGLES— Utilizada para desenhar triângulos individuais. Desenha um triângulo para cada grupo de três vértices. Se o software especifica “n” vértices, serão desenhados “n/3” triângulos. Se o número de vértices não é múltiplo de três, os vértices em excesso são ignorados;
- GL_TRIANGLE_STRIP— Utilizada para desenhar uma sequência de triângulos que compartilham arestas. Desenha o primeiro triângulo usando os três primeiros vértices, o segundo triângulo usando o segundo, o terceiro e o quarto vértices, e assim por diante. Se o software especifica “n” vértices, serão desenhados “n-2” triângulos conectados. Se “n” não é múltiplo de três, os excessos são ignorados e com menos de três vértices nada é desenhado;
- GL_TRIANGLE_FAN— Utilizada para desenhar uma sequência de triângulos que compartilham arestas e também o primeiro vértice especificado (como as fatias de uma pizza). O software especifica uma sequência de vértices “v” (v0, v1, v2,v3, etc.) e a OpenGL desenha o primeiro triângulo utilizando V0,V1 e V2, o segundo triângulo com V0, V2 e V3 e assim por diante. Para “n” vértices são desenhados “n-2” triângulos conectados. Se “n” não é múltiplo de três, os excessos são ignorados e com menos de três vértices nada é desenhado;
- GL_QUADS— Utilizada para desenhar quadriláteros convexos. Desenha um quadrilátero para cada grupo de quatro vértices. Se o software especifica “n” vértices são desenhados “n/4” quadriláteros. Se “n” não é múltiplo de quatro, os excessos são ignorados e com menos de quatro vértices nada é desenhado;

- GL_QUAD_STRIP— Utilizada para desenhar uma sequência de quadriláteros que compartilham arestas. O software especifica uma sequência de vértices “v” (v0, v1, v2, v3, etc.), OpenGL desenha o primeiro quadrilátero usando v0, v1, v 3, e v 2; Outro quadrilátero usando v3, v2, v4, e v5; outro com v4, v5, v7 e v6; e assim por diante. Se o software especifica “n” vértices são desenhados “(n-2)/2” quadriláteros. Se “n” não é múltiplo de quatro, os excessos são ignorados e com menos de quatro vértices nada é desenhado;
- GL_POLYGON— Utilizado para desenhar um polígono convexo de “n” lados a partir de uma sequência de “n+1” vértices. Com menos de três vértices nada é desenhado.

A sequência de desenho dos vértices e o resultado do desenho das primitivas em OpenGL é mostrado na Figura 11.

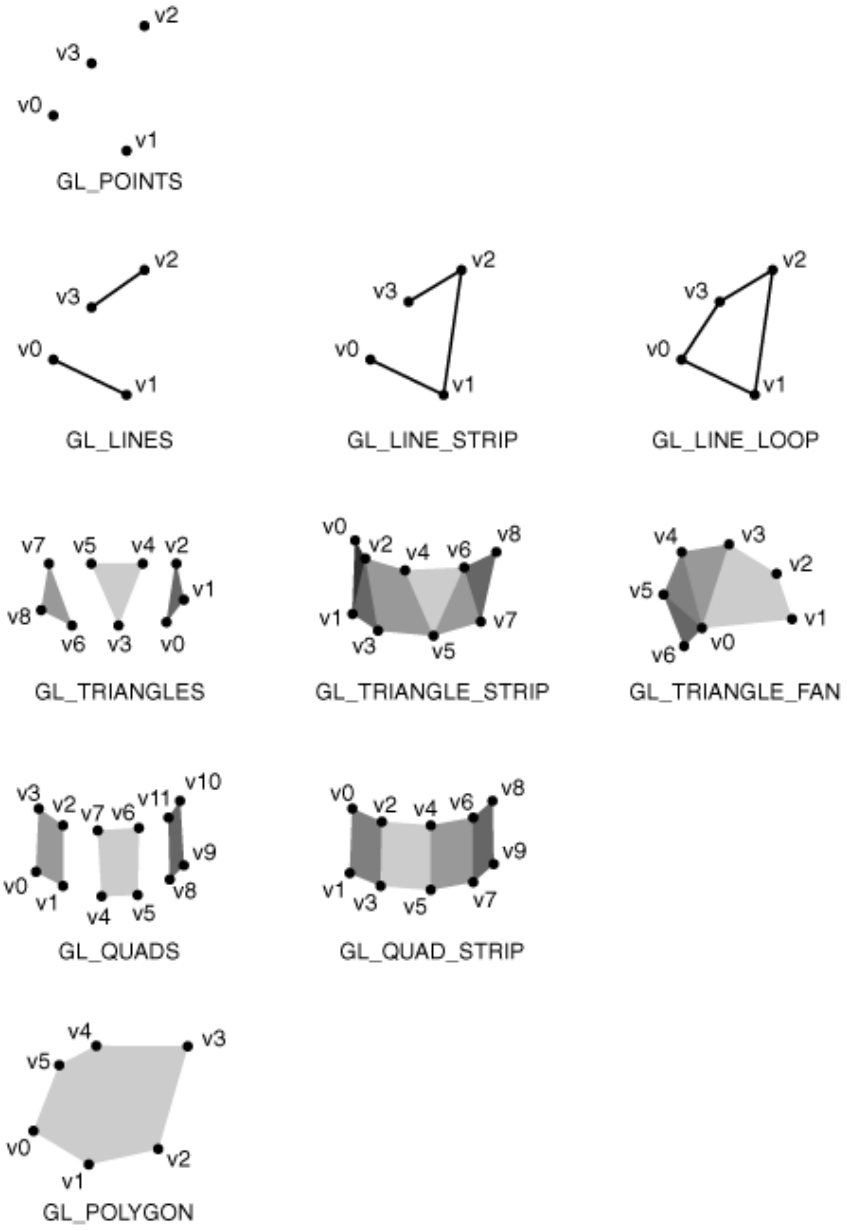


Figura 11: Primitivas OpenGL.
Fonte: (Martz, 2006)

Além do desenho de primitivas gráficas básicas, como por exemplo linhas e polígonos, a OpenGL dá suporte à iluminação, colorização, mapeamento de textura, transparência, animação, entre outros efeitos especiais. A OpenGL intencionalmente fornece apenas rotinas de renderização de baixo-nível, permitindo ao desenvolvedor mais controle e flexibilidade. As rotinas fornecidas podem, facilmente, ser usadas para construir uma biblioteca de alto-nível. A biblioteca GLU (*OpenGL Utility Library*), que é incluída na maioria das distribuições OpenGL, faz exatamente isto. A biblioteca GLscene, utilizada neste trabalho e que usa uma combinação de bibliotecas para ser tão completa como a DirectX, é outro exemplo destas bibliotecas de alto-nível.

2.6.3 A BIBLIOTECA GLSCENE

A capacidade de criar e manipular objetos gráficos em 3D tornou-se uma propriedade altamente desejável nas aplicações, desde modeladores sólidos em 3D (CAD e CAE) e sistemas para animação a softwares educacionais ou jogos 3D. A diversidade dos diferentes campos destas aplicações cria a necessidade de uma biblioteca gráfica que pode ser aprendida e usada produtivamente por programadores gráficos ocasionais, ao mesmo tempo em que proporciona a flexibilidade e extensão necessária aos programadores especializados (CELES e CORSON-RIKERT, 1997).

Designers de aplicações gráficas 3D enfrentam vários desafios em programação, em decorrência disto, interfaces gráficas são frequentemente negligenciadas, pelo grau de habilidade e esforço necessários para construí-las. Aprender a escrever aplicações que usam a biblioteca OpenGL eficientemente não é uma tarefa trivial. A OpenGL é uma biblioteca gráfica que usa uma lista de vértices para desenhar linhas e polígonos simples o mais rápido possível. Para fazer algo mais prático como “desenhar uma peça”, por exemplo, o programador precisa decodificar o objeto em uma série de instruções simples em OpenGL para que sejam desenhadas.

Um dos problemas com essa abordagem é que o desempenho da OpenGL é sensível à maneira como estas instruções são enviadas ao sistema, necessitando que o programador saiba quais instruções executar e a ordem exata de envio para obter o melhor desempenho possível. Isto força o programador a analisar cuidadosamente os dados para evitar que objetos que nem sequer serão visíveis na imagem final sejam enviados e analisados pelo sistema. Mesmo em programas simples, um enorme esforço de programação precisa ser empregado apenas para começar a mostrar algum resultado na tela.

Considerando que a biblioteca OpenGL é uma especificação de baixo nível, necessitando que o programador especifique os passos exatos para a renderização de uma cena, é importante o uso de uma ferramenta que trabalhe em um alto nível de abstração para fornecer ao desenvolvedor uma plataforma gráfica 3D apropriada ao rápido desenvolvimento da aplicação. Trabalhar em um alto nível de abstração é desejável mesmo para designers experientes em aplicações gráficas 3D.

Existem muitas bibliotecas de alto-nível disponíveis que são implementadas a partir da OpenGL, mas poucas delas atendem as especificações de projeto definidas nesta pesquisa. A

biblioteca GLscene (GLSCENE s.d.) atende a estas necessidades, facilitando o processo de desenvolvimento e melhorando a integração de uma interface OpenGL com a linguagem Delphi (CANTÚ,1999).

Esta biblioteca facilita a programação da interface 3D uma vez que muitas das rotinas de baixo nível na programação OpenGL já estão resolvidas nesta biblioteca fornecendo componentes e objetos visuais que permitem a descrição e renderização de cenas 3D de uma maneira visual e interativa, possibilitando a determinação das propriedades dos componentes em tempo de projeto e em tempo de execução. A GLscene é uma biblioteca baseada em OpenGL de código aberto (*open source*) para Delphi, que facilita a descrição e renderização de cenas 3D(GLSCENE s.d.).

Implementada em *Object Pascal* (linguagem do Delphi), a GLscene enfoca a criação de objetos 3D e oferece uma interface orientada a objetos com alto nível de abstração para o desenvolvimento de aplicações interativas gráficas 3D, concentrando suas atividades na construção e manipulação dos objetos. Seu principal objetivo é facilitar a programação gráfica 3D, tanto para designers experientes quanto para iniciantes, fornecendo acesso rápido as funcionalidades OpenGL em alto-nível, em um ambiente propício ao aumento da eficiência e produtividade.

A GLScene é uma biblioteca de funções gráficas que aproveita de forma significativa o conceito RAD (*Rapid Application Development)* introduzido pela linguagem de programação Delphi, esta de larga preferência entre os estudantes de Sistemas de Informação. Mais do que isso, ela é uma API 3D (*Engine*) completa para criar aplicações rápidas e eficientes, sem precisar conhecer aspectos intrínsecos da API OpenGL, além de possuir uma enorme coleção de aplicações de demonstração que possibilitam entender rapidamente o seu funcionamento.

O desenvolvimento da biblioteca original começou em 1999 por Mike Lischke. Na versão 0.5, a biblioteca GLscene foi transformada em código aberto, licenciada pela Mozilla Public License 1.1 (MPL 1.1) e colocada sob os cuidados de Eric Grange que administra as versões oficiais.

A GLScene utiliza um sistema de distribuição conhecido como CVS (*concurrent version system* ou sistema de versões concorrentes) que permite que se trabalhe com diversas versões de arquivos organizados em um diretório e localizados local ou remotamente, mantendo-se suas versões antigas e os registros de quem manipulou os arquivos, quando isto foi feito e quais alterações foram feitas (CVS s.d.). A última versão oficial é a versão 1.0.0.0714, lançada em Julho de 2006, adaptando a biblioteca ao Delphi 2007, mas versões não oficiais já circulam pela internet, como a versão da GLscene para o Delphi 2009, criadas por integrantes de sua comunidade de usuários.

A GLScene é um software livre de alta qualidade que atende as necessidades de qualquer desenvolvedor de softwares gráficos, sejam eles científicos, aplicações comerciais ou mesmo jogos 3D Interativos. Esta biblioteca permite aos programadores criar uma grande variedade de objetos 3D e controles adicionais para visualização, animação e som em tempo

de projeto, usando sua própria interface. A Figura 12 mostra a interface do GLscene no Delphi em tempo de projeto, mostrando a diversidade de objetos que podem ser criados.

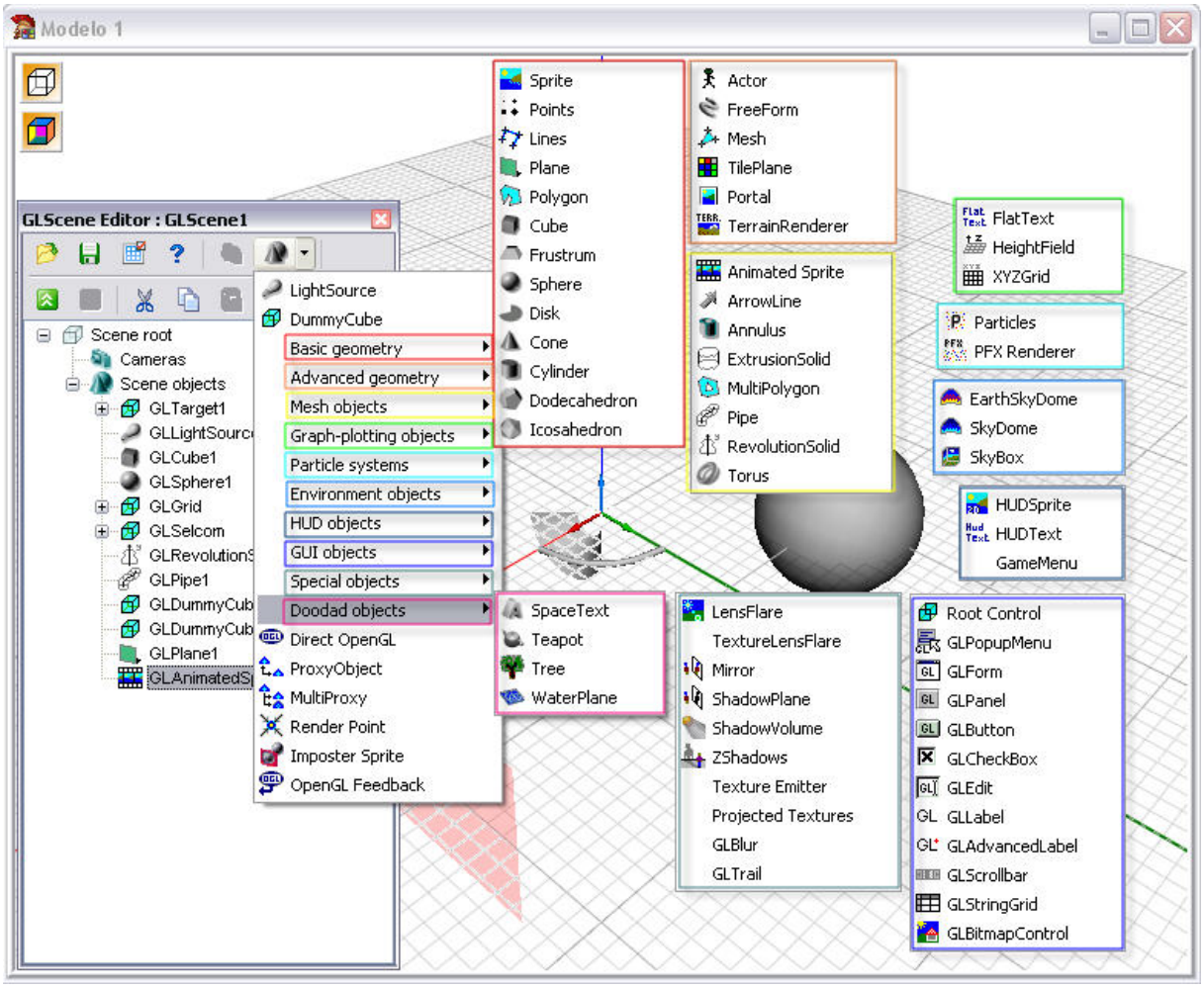


Figura 12: Interface do GLScene com os diversos objetos disponíveis.
Fonte: O Autor.

A seguir, são descritas algumas características da biblioteca (GLScene site oficial, s.d.):

- Programação Orientada a objetos com código fonte disponível;
- Gerenciamento interativo de cena: um cenário na biblioteca GLscene é definido por uma estrutura hierárquica de objetos que fazem parte do ambiente virtual facilmente expansível. Estes objetos são adicionados à cena pela própria ferramenta, conforme a escolha do programador, tanto em tempo de projeto como em tempo de execução;
- Podem ser criados objetos complexos pré-definidos:
    - *Basic Objects*: Apresenta suporte para objetos 2D e 1D como pontos, linhas com suporte a *Spline*, planos e discos. Além destes, traz também primitivas 3D como cubos, pirâmides, troncos de pirâmides, esferas, cones, cilindros, entre outros;
    - *Advanced Objects*: *ArrowLine* (uma composição de cilindro e cone para representar setas ou eixos), *Torus* (toróide), *Anulus* (cilindro vasado), *Pipe*,

RevolutionSolid (sólido de revolução), *ExtrusionSolid* (sólido por extrusão), entre outros;

- *Mesh Objects*: Permite ainda a importação de arquivos para serem usados como um objeto *Actor* (personagens em jogos). Outros objetos desta categoria são os objetos *Mesh* (uma malha de planos definida a partir dos vértices destes planos), *FreeForm*, *TerrainModeler*, entre outros;
- Objetos estruturais (*dummy cube*, *proxy object*) para manuseio de objetos compostos;
- Objetos especiais como *Teapot* (bule de chá), *SpaceText* (texto em 3D), *Tree* (árvore) e *WaterPlane* (simulação de um plano d´água);
- Objetos HUD (imagens renderizadas em 2D dentro de um mundo 3D);
- Objetos utilitários como Grades e abóboda celeste (*grids*, *skydome*);

- Permite, ainda, a importação de arquivos do tipo 3DS e MD2 (formatos comuns em aplicativos 3D e CAD), importação de texturas também é suportada, sendo sua limitação o máximo de objetos que a memória permitir. Outros formatos para importação incluem: OBJ/OBJF, SMD, STL, TIN e PLY;
- Fácil aplicação de rotação, translação e escala para cada objeto, sejam eles em uma, duas ou três dimensões;
- Permite a inserção e manipulação de objetos de câmera e luz que podem ser usados em qualquer lugar da cena, em qualquer número;
- Permite a criação de efeitos especiais com suporte a sistemas de partículas, *LensFlare* (brilho na lente) e projeção de texturas;
- Materiais- podem-se otimizar materiais disponíveis pela ferramenta e adicioná-los a esta biblioteca. A aplicação de materiais, com efeitos do tipo difusão, contraste, especular, brilho e matização, torna os componentes do programa mais realçados, aproximando-os da realidade. A ferramenta também disponibiliza o suporte a modelos polimórficos com texturas, remoção das texturas e propriedades (independente das coordenadas) e imagens de 32 bits;
- Renderização- uma das grandes vantagens da GLScene é a habilitação automática de *driver* OpenGL (caso exista) para a placa gráfica. Além disto, podem-se ajustar diversas câmeras em um único cenário, permitindo diferentes visualizações em diversos ângulos, além de efeitos de neblina e sombra, reflexão de objetos (espelhos) e suporte a tela cheia;
- Animação- a animação de objetos requer cadência dos mesmos. A biblioteca fornece esta cadência de maneira automática, com propagação de eventos progressivos no tempo e definição de comportamentos dos objetos. Pode-se também interpolar frames e permitir metamorfoses;
- Aplicação de física dinâmica: inércia, aceleração, frenagem, força e detecção de colisões;
- Interface 2D- possui a possibilidade da criação de interfaces em 2D dentro do mundo virtual como menus, grades e caixas de seleção, muito útil na criação de jogos;
- Interface 3D- dispõe de funções para seleção de objetos, as quais ajudam na movimentação das câmeras e na translação de objetos. A interface de aplicação

também auxilia na conversão entre telas e coordenadas de mundos virtuais, utilizando a técnica *raycasting*;

- Áudio- podem receber a habilitação de som 3D em movimentos, objetos e a própria cena, com atualização automática de posicionamento, com orientação e velocidade de entrada e saída de som;
- Utilitários- compostos por funções geométricas otimizadas (vetores, matrizes e quaternários), determinação da velocidade exata de frames, relógio assíncrono (multitarefa) e suporte a *joystick*.

A biblioteca GLscene combina poder e versatilidade. É uma ferramenta útil para os desenvolvedores com um nível de conhecimento intermediário em programação que queiram criar aplicações interativas em 3D, ao mesmo tempo em que oferece suporte aos desenvolvedores especialistas usando uma linguagem orientada a objetos e a possibilidade da customização do código para atender a qualquer necessidade específica graças ao seu código aberto. (GLSCENE s.d.).

Considerando-se todas as vantagens da utilização de bibliotecas gráficas para a criação de uma interface para interação tridimensional e ainda as vantagens e desvantagens apontadas anteriormente na comparação das bibliotecas pesquisadas, este trabalho utiliza a biblioteca GLscene, por oferecer uma estrutura orientada a objetos, fornecer ferramentas versáteis para criar e gerenciar aplicações em 3D, utilizar a OpenGL para a renderização das cenas, pela integração com o ambiente de programação Delphi, além de ser de código aberto, permitindo o acesso a sua estrutura e possibilitando a customização caso necessário.

3 PROJETO DO SISTEMA

Para uma utilização otimizada de recursos, neste caso principalmente o tempo, a implementação de um software necessita de um planejamento prévio. O planejamento envolve tarefas minuciosas e detalhadas desde a demanda do usuário, passando por levantamento de requisitos, detalhamento de escopo, até processo de testes e finalmente o lançamento (*deployment*). Considerando ainda o software T-CADE como um conceito de plataforma de desenvolvimento é importante que se sigam determinadas restrições de projeto, para que outros desenvolvedores possam dar continuidade ao trabalho já iniciado.

3.1 ESCOPO DO PROJETO

O termo interface gráfica pode dar margem para diversas abordagens de interação com o usuário, especialmente quando o foco do trabalho trata de usabilidade. A interação com o usuário pode ocorrer tanto por menus, ícones, barras de ferramentas e caixas de diálogo como a própria interação com os objetos 3D dentro da área de trabalho, conforme mostrado na Figura 13.

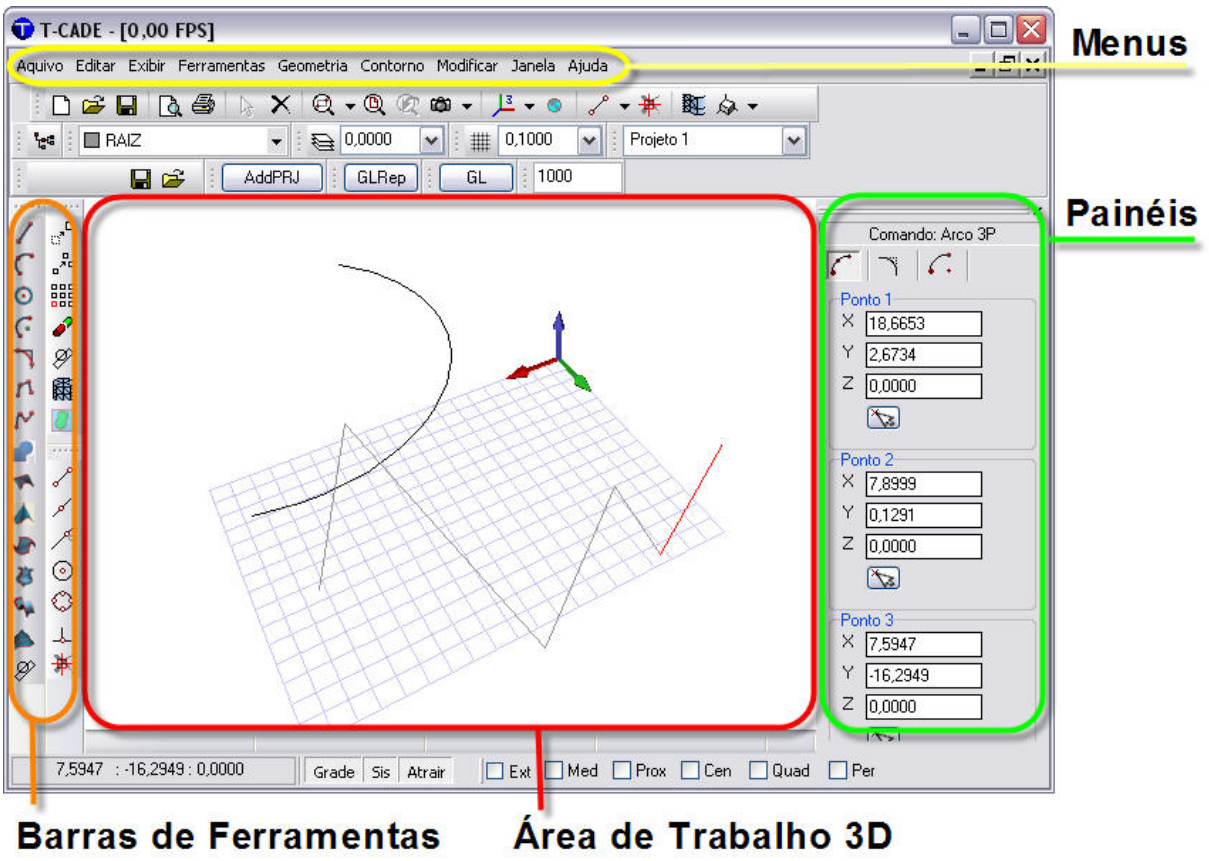


Figura 13: A linha vermelha evidencia a área de trabalho da plataforma T-CADE.
Fonte: O Autor.

Este trabalho se refere especificamente a esta interação 3D da área de trabalho, seja no desenvolvimento de algoritmos para interação com biblioteca OpenGL, seja na criação de ferramentas gráficas para melhorar esta interação.

Uma vez que o T-CADE foi concebido com o conceito de plataforma de desenvolvimento, onde receberá constantes acréscimos, reavaliações e modificações, a interface 3D desenvolvida aqui não pode ser considerada como um produto final, ao invés disto deve ser considerada como um protótipo evolutivo. Tal protótipo permitirá a definição de novas técnicas de interação, a implementação de novas ferramentas e a utilização do aplicativo durante o seu desenvolvimento.

Para tanto, as avaliações formativas, realizadas ao longo do desenvolvimento do protótipo da interface, tiveram grande importância na tomada de decisões de design. As avaliações somativas ou conclusivas, como por exemplo, os extensivos testes de usabilidade, que deveriam ser realizados após a conclusão do sistema final, não fazem parte do escopo deste trabalho, em função do nível de aprofundamento da pesquisa (nível de protótipo) e dos recursos disponíveis.

3.2 REQUISITOS DO PROJETO

Como o T-CADE é uma plataforma de desenvolvimento, um software essencialmente acadêmico e não voltado diretamente ao mercado, considerou-se como clientes o criador do software e a própria equipe de desenvolvimento. Após algumas reuniões foram levantados como objetivos ou metas a serem alcançadas a facilidade de uso e interatividade encontradas nos softwares mais representativos destes segmentos: 3D, CAD e CAE.

Assim, para a implementação de uma nova interface em um programa 3D, foram investigados os softwares para projeto mais amplamente utilizados no mercado atualmente: Rhinoceros®, AutoCAD® e Inventor®. Estes programas foram estudados, principalmente, sob o ponto de vista das ferramentas utilizadas para manipulação direta dos objetos, das ferramentas de visualização e movimentação da câmera no mundo tridimensional, e dos métodos e técnicas de trabalho voltados à produtividade e usabilidade.

Na avaliação comparativa entre estes softwares, foram encontradas algumas características importantes comuns a todos:

- O uso de manipuladores para restrições de eixos para a manipulação direta de objetos, podendo-se, diretamente com o uso do cursor em um único movimento, manipular os objetos com as restrições desejadas.
- O uso de teclas de atalho para facilitar e acelerar o uso das ferramentas pelos usuários mais experientes.
- O uso de vistas pré-definidas mais frequentemente utilizadas ou mais representativas para a visualização em 3D.
- A possibilidade de se trabalhar em planos auxiliares de trabalho, como sistemas de coordenadas criados pelo usuário com uma posição no espaço distinta do sistema de coordenadas globais, bem como os sistemas locais dos objetos e das vistas (ou câmeras).
- O uso transparente dos comandos de visualização, ou seja, deve ser possível usar funções da câmera, como *zoom* (tamanho da imagem na tela), *pan* (translação da inagem), ou *orbit* (girar a câmera ao redor do alvo), enquanto executa um determinado

comando de edição, facilitando a visualização completa necessária para melhor entendimento e execução precisa da tarefa.

- Seleção direta de objetos com um clique sobre o objeto, e a possibilidade de selecionar os objetos através de janelas de seleção, com dois métodos distintos, selecionando apenas objetos inteiramente contidos na Janela de seleção ou incluir também os objetos parcialmente contidos na janela.

Além destas observações, algumas características foram apontadas como importantes para um software de modelagem tridimensional:

- A importância de que quando um objeto é selecionado, seja possível identificar sobre a superfície deste objeto o ponto 3D apontado pelo cursor; o reconhecimento do ponto 3D, sobre a superfície que é clicada ao se selecionar um objeto, é importante para o desenvolvimento de alguns algoritmos, como o de concordância de superfícies, que atualmente em termos de estrutura de dados já poderia ser implementado, mas necessita deste reconhecimento do ponto 3D clicado na superfície para funcionar conforme o esperado.
- Deve ter um bom desempenho gráfico quando trabalhar com vários objetos. O desempenho gráfico é importante em termos de usabilidade e da garantia da interatividade com o usuário. O desempenho gráfico não pode ser limitador das ações do usuário. Uma nova interface gráfica só se justifica se ela apresentar um desempenho gráfico melhor que a interface original.
- A interatividade é fundamental na criação dos objetos, uma retroalimentação (*feedback*) mais natural e precisa pode ser fornecida enquanto o usuário está interagindo.
- A nova interface deve ser mais fácil de usar; deveria oferecer uma melhor a usabilidade em relação à interface original.

A partir do exposto acima e dos conceitos e princípios da usabilidade almejados para este trabalhado, pode-se começar a entrada de dados na matriz QFD ou Casa da Qualidade (item 2.1.1.1, pág. 23). Estas intenções ou necessidades do usuário são sintetizadas e classificadas em termos de requisitos do usuário, como mostra a Figura 14. Ainda nesta etapa, é também atribuído um grau de importância a cada um destes requisitos em uma escala de 1 a 5, sendo 1= pouco importante e 5= muito importante.

	Importância
Fácil de usar para usuários iniciantes e avançados	3
Usuário no controle	5
Boa Interatividade	4
Bom desempenho Gráfico	3

Figura 14: Necessidades do usuário.
Fonte: O Autor.

Na etapa seguinte, os requisitos de projeto são definidos e, a partir da matriz de inter-relações, são determinados os pesos de cada um na busca pelo alcance das necessidades do usuário. Para a valoração do quanto determinado requisito de projeto atende a uma necessidade do usuário é utilizada uma escala com três valores, 1, 3 e 9, conforme recomendado pela literatura (GUAZZI, 1999). A matriz mostra uma variação entre os pesos relativos resultantes entre 6,7% e 16%. A Figura 15 mostra a matriz de inter-relações e os pesos resultantes.

	Importância	Restrições de eixos	Utilização de Planos de trabalho	Manipulação da visualização da cena	Manipulação de objetos	Uso de teclas de atalho	Opções para Seleção de objetos	Uso de Biblioteca Gráfica 3D	Atualização da tela	Criação de Modos de visualização	Comunicação com o usuário
Fácil de usar para usuários iniciantes e avançados	3	3	1	3	3	9	9				9
Usuário no controle	5	9	9	9	9	3	3		1	3	3
Boa Interatividade	4	1		3	9	1	1	3	9		1
Bom Desempenho Gráfico	3			9	1			9	9	9	
Pesos abolutos		58	48	93	93	46	46	39	68	42	46
Pesos Relativos %		10	8,3	16	16	7,9	7,9	6,7	12	7,3	7,9

Figura 15: Matriz de inter-relações (requisitos de projetoX necessidades do usuário).
Fonte: O Autor.

No telhado da matriz QFD, foram relacionados os requisitos um com o outro para determinar as mútuas interferências, positivas ou negativas. Foi utilizada a legenda conforme Figura 16:

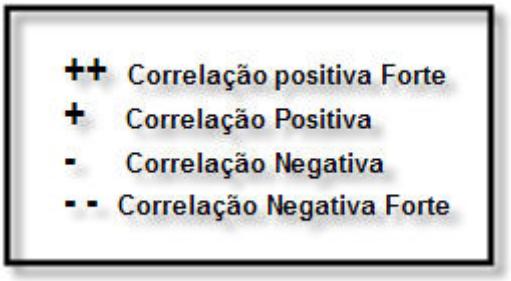
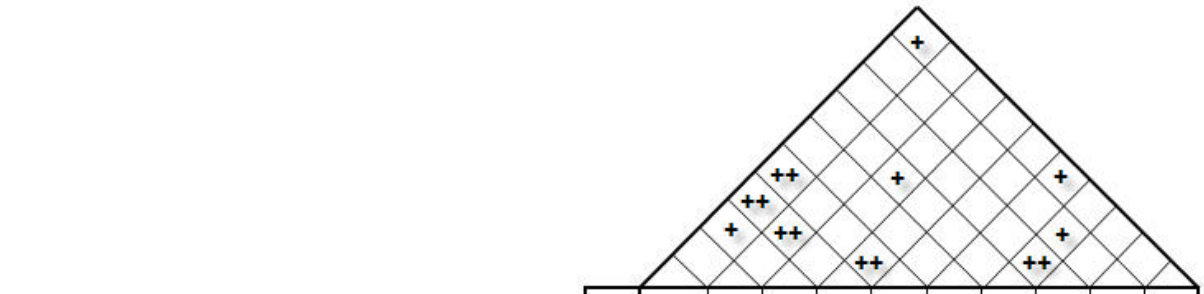


Figura 16: Legenda de Correlação.
Fonte: O Autor.

A Figura 17 mostra como se correlacionam os requisitos de projeto; não foi encontrada nenhuma relação negativa entre eles.



	Importância	Restrições de eixos	Utilização de Planos de trabalho	Manipulação da visualização da cena	Manipulação de objetos	Uso de teclas de atalho	Opções para Seleção de objetos	Uso de Biblioteca Gráfica 3D	Atualização da tela	Criação de Modos de visualização	Comunicação com o usuário
Fácil de usar para usuários iniciantes e avançados	3	3	1	3	3	9	9				9
Usuário no controle	5	9	9	9	9	3	3		1	3	3
Boa Interatividade	4	1		3	9	1	1	3	9		1
Bom Desempenho Gráfico	3			9	1			9	9	9	
Pesos abolutos		58	48	93	93	46	46	39	68	42	46
Pesos Relativos %		10	8,3	16	16	7,9	7,9	6,7	12	7,3	7,9

Figura 17: Correlação entre requisitos de usuário e requisitos de projeto.
Fonte: O Autor.

A partir da matriz QFD, percebeu-se que:

- Seria necessária a criação de ferramentas para o controle da visualização da cena, tanto de controle dinâmico da câmera como para vistas pré-definidas;
- O uso de teclas de atalho poderia controlar eficientemente as restrições de eixos na manipulação de objetos e que estas restrições deveriam levar em consideração os eixos dos planos de trabalho.
- As ferramentas deveriam ser de fácil utilização tanto por usuários iniciantes quanto pelos avançados.
- A pequena variação dos pesos relativos de cada uma das intervenções torna todos os requisitos de projeto importantes para o sucesso da pesquisa.

A comunicação com o usuário é importante para os usuários iniciantes, facilitando o uso do aplicativo e mantendo o usuário no controle da situação.

Nesta fase inicial, o QFD é importante para decidir como agir e avaliar realisticamente a importância de cada item no contexto geral de projeto. Por exemplo, foi uma surpresa perceber que o uso de uma biblioteca gráfica 3D, apesar de fundamental para a realização do trabalho, não tinha grande impacto direto no objetivo final da pesquisa, melhorar a usabilidade do T-CADE. No entanto, outros requisitos como a atualização de tela e os modos de visualização dependem da biblioteca gráfica diretamente e sua não implementação inviabilizaria o projeto. Também surpreendeu o quanto pequenas ações, como a comunicação com o usuário e a seleção de objetos, têm o potencial de melhorar a experiência do usuário.

3.3 SELEÇÃO DE CONCEITOS

Com base na etapa anterior, as ferramentas de controle de câmera (manipulação da visualização da cena) e manipulação de objetos são as que mais influenciam nos requisitos do usuário e, portanto, merecem uma maior atenção.

O controle de câmera era o problema que oferecia um maior número de possíveis ferramentas a serem implementadas para a sua solução. Para a definição de qual método de controle das funções da câmera a ser adotado, são utilizadas as técnicas de seleção de conceitos do processo de desenvolvimento de projeto. Em um primeiro momento, é realizada uma triagem das possíveis soluções com base nos critérios estabelecidos. Em seguida realizou-se uma valoração destes critérios para, só então, selecionar a melhor alternativa dentre as aprovadas na triagem inicial.

Em função dos requisitos de usabilidade, dos recursos disponíveis para pesquisa e de seus objetivos, foram elencados alguns critérios através dos quais as possíveis alternativas de métodos para manipulação da câmera virtual foram selecionadas.

Os principais critérios determinantes foram:

- Ser de fácil implementação- A solução deve ser simples para que possa ser implementada em tempo hábil e que esteja dentro do domínio de conhecimento do programador responsável;
- Ser de fácil utilização- Deve ser de fácil utilização pelos usuários novatos, assim como pelos especialistas;
- Proporcionar um acesso rápido- A ferramenta deve ser acessada diretamente, sem a necessidade de diversos cliques, caixas de diálogos, ou sintaxes que demandem tempo do usuário;
- Ser de fácil memorização- seu funcionamento deve privilegiar a memória do usuário;
- Possibilitar o controle total da câmera pelo usuário- Neste caso, é importante o controle de sete graus de liberdade (7 DOF- translação nos três eixos, rotação nos três eixos e da profundidade de campo). O controle de zoom da câmera pode ser dado pelo afastamento da câmera no eixo perpendicular à tela, bem como pelo controle da profundidade de campo (ajuste das lentes);
- Utilizar dispositivos 2D- A solução deve utilizar apenas hardware convencional como mouse, teclado e monitor;

- Possibilitar o uso transparente- A câmera deve ser controlada de modo transparente aos outros comandos, ou seja, deve ser possível a movimentação da câmera mesmo durante a execução de algum outro comando pelo usuário;
- Manter o foco na área de trabalho- a ferramenta deve, sempre que possível, manter a atenção do usuário na área de trabalho.

Para as possíveis soluções de manipulação da câmera, algumas alternativas foram consideradas:

- *Viewports* adicionais (VPA)- a partir da criação de *viewports* (janelas de visualização) adicionais, pode-se controlar a posição da câmera movendo-se diretamente o objeto ou ícone da câmera através de *viewports* ortogonais enquanto a imagem resultante é mostrada em outra *viewport*;
- Uso de um *TrackBall* (TKB)- uso de um dispositivo de controle externo para controle dos movimentos da câmera;
- Controles 2D (C2D)- Em um painel lateral, pode-se criar dispositivos 2D para controlar os movimentos da câmera, como barras deslizantes horizontais e verticais, caixas de textos para entrada de ângulos e deslocamentos, entre outros;
- Controles 3D virtuais (C3D)- Na área de trabalho, objetos tridimensionais podem ser manipulados representando o movimento da câmera, como por exemplo, esferas sendo rotacionadas por sua superfície ou setas sendo movidas ao longo de seu eixo;
- Linha de comando (LDC)- As instruções da câmera podem ser acionadas por opções em uma linha de comando;
- Barra de ferramentas (BDF)- Os comandos de visualização podem ser agrupados em barras de ferramentas como os demais comandos do programa;
- Mouse (MOU)- Os comandos de visualização podem ser mapeados e acionados a a partir de determinados botões do mouse.

Como solução de referência (ORI) foi utilizada a técnica existente na interface original, uma solução híbrida que utiliza o movimento do mouse associado a teclas de modificação (Ctrl) para a rotação da câmera, um painel lateral com controles 2D para posicionamento e um menu *DropDown* para o controle de zoom.

Uma vez que os critérios estavam definidos e as possíveis soluções foram consideradas, realizou-se uma triagem para que somente as soluções mais viáveis fossem estudadas. Esta triagem foi realizada a partir do método de Pugh (PUGH, 1990). A Figura 18 mostra a matriz de Pugh relacionando os critérios de triagem aos conceitos de solução da ferramenta de visualização (item 2.1.2.4, pág. 29).

A partir desta matriz é possível perceber que os conceitos 2, 4, 6 e 7 apresentam uma valoração positiva em relação à interface existente, que são respectivamente o uso de um *TrackBall*, de controles 3D virtuais, de barras de ferramentas e utilização dos botões do mouse.

Soluções	0	1	2	3	4	5	6	7
critérios	ORI	VPA	TKB	C2D	C3D	LDC	BDF	MOU
Fácil Implementação		-	-	s	-	-	+	-
Fácil utilização		-	-	-	+	-	+	+
Acesso rápido		-	+	-	+	+	-	+
Fácil memorização	D	-	+	s	+	-	+	+
Controle Total	A	+	+	s	+	+	+	+
Dispositivos 2D	T	s	-	s	s	s	s	s
Uso Transparente	U	-	+	s	s	-	-	s
Foco na área de trabalho	M	+	+	-	+	-	-	+
Somatório +		2	5	0	5	2	4	5
Somatório -		-5	-3	-3	-1	-5	-3	-1
Saldo		-3	**2**	-3	**4**	-3	**1**	**4**

Figura 18: Matriz de Pugh, triagem de conceitos.
Fonte: O Autor.

Estas soluções são investigadas em maior detalhe e selecionadas utilizando o método da valoração dos critérios e da função utilidade (BACK *et al.*, 2008). O método de valoração dos critérios é realizado pela comparação individual entre os mesmos. Comparando-se cada critério a cada um dos outros, é possível estabelecer qual dos dois é mais importante a cada comparação. Assim, podem-se estabelecer os pesos da importância relativa de cada critério no projeto. A Figura 19 mostra a avaliação dos pesos de importância.

		Critérios								Soma da Linha	Pesos (%)
		Fácil Implementação	Fácil Utilização	Acesso Rápido	Fácil Memorização	Controle Total	Dispositivos 2D	Uso Transparente	Foco na Área de Trabalho		
Critérios	Fácil Implementação	-	0	0	1	0,5	0	0	0	2,5	8,62
	Fácil Utilização	1	-	0	1	1	0,5	0,5	0,5	4,5	15,51
	Acesso Rápido	1	1	-	1	0,5	0	1	0,5	5	17,24
	Fácil Memorização	0	0	0	-	0,5	0,5	0	1	2	6,9
	Controle Total	0,5	0	0,5	0,5	-	0	1	0	2,5	8,62
	Dispositivos 2D	1	0,5	1	0,5	1	-	0,5	0,5	5	17,24
	Uso Transparente	1	0,5	0	1	0	0,5	-	0,5	3,5	12,07
	Foco na Área de Trabalho	1	0,5	0,5	0	1	0,5	0,5	-	4	13,8
		Somatório								29	100

Figura 19: Matriz de avaliação por comparação dos pesos dos critérios de seleção.
Fonte: O Autor.

Para que se utilize o método da função utilidade é necessário utilizar uma escala numérica para valoração dos critérios qualitativos. A valoração adotada neste trabalho segue os conceitos e os respectivos valores conforme mostrado na Figura 20.

Conceitos Qualitativos	Valoração Numérica dos critérios
Satisfatório	1
Regular	2
Bom	3
Muito Bom	4
Excelente	5

Figura 20: Valoração de Critérios Qualitativos.
Fonte: O Autor.

Concluídas a valoração e a determinação dos pesos de importância dos critérios, o passo seguinte é o cálculo do valor da função utilidade das concepções para obter a ordenação das mesmas. A matriz da Figura 21 mostra a determinação dos valores da função utilidade das concepções aprovadas na triagem inicial.

Critérios de Seleção	Pesos (%)	Concepções Geradas							
		TKB		C3D		BDF		MOU	
		Val. Qual.	*VQ * pesos*	*Val. Qual.*	*VQ * pesos*	*Val. Qual.*	*VQ * pesos*	*Val. Qual.*	*VQ * pesos*
Fácil Implementação	8,62	1	8,62	1	8,62	5	43,1	1	8,62
Fácil Utilização	15,51	1	15,51	2	31,02	4	62,04	3	46,53
Acesso Rápido	17,24	4	68,96	4	68,96	1	17,24	4	68,96
Fácil Memorização	6,9	3	20,7	5	34,5	4	27,6	3	20,7
Controle Total	8,62	5	43,1	4	34,48	4	34,48	4	34,48
Dispositivos 2D	17,24	1	17,24	4	68,96	5	86,2	4	68,96
Uso Transparente	12,07	5	60,35	5	60,35	1	12,07	5	60,35
Foco na Área de Trabalho	13,8	4	55,2	3	41,4	1	13,8	5	69
Valor da Função Utilidade			289,68		348,29		296,53		377,6
Ordenação das Concepções		4ª Posição		2ª Posição		3ª Posição		1ª Posição	

Figura 21: Valor da função Utilidade.
Fonte: O Autor.

A partir desta análise, pode-se identificar as opções de implementação da ferramenta de controle de câmera. Na primeira posição ficou o mapeamento das funções nos botões do mouse. A criação de controles 3D na área de trabalho também pode ser considerada como uma boa solução. Neste trabalho, as duas soluções foram implementadas. O QuadMenu que controla os movimentos da câmera a partir do uso do botão direito do mouse e o ViewCube que controla a definição da posição da câmera em vistas pré-definidas, frontal, superior, laterais, vistas compostas por duas faces vizinhas, e isométricas.

3.4 RESTRIÇÕES DE PROJETO

Considerando que a plataforma T-CADE já vem sendo desenvolvida há alguns anos, a principal restrição de projeto era sua estrutura original. Qualquer nova implementação deveria respeitar a estrutura original, sendo assim, podem ser adicionados novos atributos e métodos às classes já existentes, podem ser criadas novas classes dentro da estrutura original, mas não devem ser modificadas as classes pré-existentes.

Uma segunda restrição é que ambas as interfaces devem funcionar perfeitamente até o final deste projeto, para só depois se decidir pela retirada da interface original (ou não). Sendo assim, qualquer que seja a solução de um determinado problema, ela não pode alterar significativamente a maneira como as coisas funcionam atualmente, na melhor das hipóteses pode-se criar um caminho paralelo ao primeiro.

Finalmente, deve-se considerar apenas soluções que contemplem exclusivamente a utilização de dispositivos 2D de entrada e saída.

3.5 A ESTRUTURA DO T-CADE

De acordo com Teixeira (2004) o T-CADE foi criado utilizando-se a linguagem de programação Delphi e uma programação orientada a objetos. Entre outros objetos criados no T-CADE estão as classes de objetos geométricos, que foram organizados em quatro classes fundamentais que definem hierarquia e complexidade: *TPonto, TObjetoGeométrico, Tlinha e TSuperfície*.

A Classe *TPonto* representa um ponto com suas coordenadas 3D. É uma classe fundamental para o sistema de posicionamento, visualização e geração de todos os objetos, incluindo os sistemas de referência, câmeras e objetos geométricos. Esta classe não possui nenhum método associado, possuindo somente as propriedades correspondentes às componentes em cada eixo (*x*,*y* e *z).*

A classe *TObjetoGeométrico* apresenta as propriedades e métodos fundamentais a todas as classes de objetos gráficos. Esta classe fundamental possui duas grandes classes filhas: classe *TLinha* e *TSuperfície*. Essas por sua vez possuem diversas classes filhas, uma para cada objeto gráfico específico.

A Classe *TLinha* é a classe mãe da qual derivam todas as classes de linhas e contém métodos comuns a todas as linhas. Da mesma forma, a classe *TSuperfície* é a classe mãe de todas as superfícies e apresenta todas as propriedades e métodos comuns as demais classes herdeiras de superfícies.

Todos os objetos geométricos criados no projeto são armazenados em um arranjo de objetos da classe *TObjetoGeométrico* pertencente à classe *TProjRec*. Esta classe além de armazenar todos os objetos criados no projeto também contém os métodos gerais para acrescentar e retirar objetos do projeto, definir cores, materiais, vínculos, cargas e propriedades dos objetos. Por exemplo, toda vez que um objeto é selecionado, este arranjo é percorrido para que o objeto seja encontrado e tenha sua propriedade *SetSel* (seleciona o objeto) modificada.

A Figura 22 ilustra a hierarquia das classes geométricas do T-CADE.

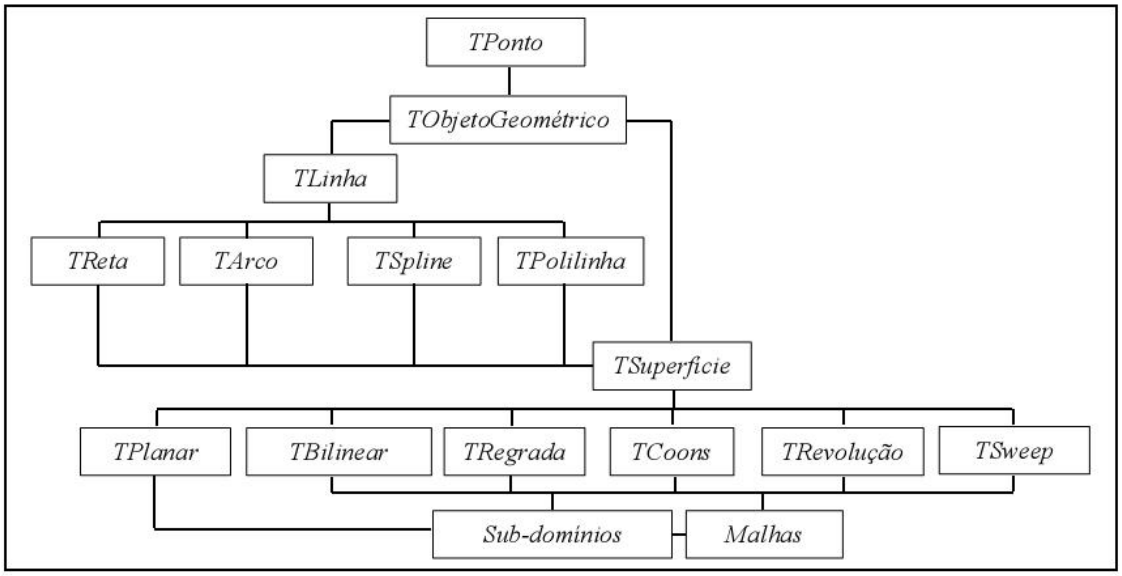


Figura 22: Hierarquia de classes de objetos geométricos
Fonte: (Teixeira, 2004)

A classe *TObjetoGeométrico* tem, entre outras propriedades, uma propriedade chamada *Retas*, que recebe uma classe de objetos chamada *TRetaVideo.* A *TRetaVideo* é a projeção das retas 3D em 2D na tela, são as linhas em 2D na tela que representam um objeto 3D no mundo virtual. Estas retas eram utilizadas pela interface original que as desenhava diretamente sobre a Tela (Canvas) do formulário.

Um método importante para a visualização do *TLinha* (Derivada do *TObjetoGeométrico*) é o método *GeraSeg_Rep* que lê um arranjo de pontos do objeto para gerar as Retas de Tela (*TRetaVideo*) daquele objeto para aquela posição de câmera especifica. Sendo assim, toda linha tem, além de suas coordenadas 3D, um objeto 2D associado a ela que é sua representação de tela. Cada vez que se muda a posição da câmera, essas retas são redesenhadas. Quando se seleciona uma linha, seleciona-se na verdade sua Reta de tela e ,só então, o software seleciona a reta 3D correspondente àquela reta de tela.

Todas as operações de visualização e transformações geométricas feitas pela interface original são feitas pela CPU, não levando em conta os recursos gráficos do hardware de vídeo disponível, o que reduz significativamente o desempenho na visualização de superfícies mais complexas.

3.6 A NOVA INTERFACE GRÁFICA

Considerando que a nova interface, que utiliza a biblioteca GLScene, deve ser implementada de maneira que a interface atual continue funcionando, optou-se por uma intervenção incremental que funcionasse paralelamente às duas interfaces. Assim, como na interface

original os objetos geométricos tem uma propriedade que é sua representação na tela (retas de tela), foi criada então outra propriedade que é sua representação em OpenGL.

Acrescentando-se uma propriedade na classe mãe de todos objetos geométricos, todas as classes derivadas dela teriam também esta propriedade, restando apenas o trabalho de implementação desta representação. Foi vinculada a esta propriedade de representação OpenGL dos objetos, uma nova classe gráfica chamada *TGLRep*, a qual será descrita mais adiante.

Como a biblioteca Glscene possui sua própria estrutura de dados 3D, e todos os objetos criados no mundo OpenGL ficam armazenados nesta estrutura, tem-se então um "Objeto GL" (uma representação em OpenGL) para cada objeto na estrutura de dados do T-CADE. Isto significa na prática que, ao invés de existir objetos 3D relacionados a objetos 2D que os representam (como era feito até então), agora existem objetos 3D (matemáticos) relacionados a outros objetos 3D (gráficos) que os representam.

Assim os objetos gráficos 3D são propriedades dos objetos matemáticos, isto é, estão vinculados a eles de tal maneira que, ao se criar o objeto matemático, o objeto gráfico tridimensional também é criado, e quando ocorre a destruição do matemático, o mesmo acontece com sua representação gráfica 3D.

3.6.1 AS NOVAS CLASSES GRÁFICAS

Para a representação em OpenGL dos objetos geométricos, foi criada uma classe chamada *TGLRep*. Trata-se de uma classe genérica que traz os métodos e propriedades necessários para a representação de linhas e superfícies. Esta classe pode conter diversas linhas e superfícies para representar um objeto. Para que isso seja possível, ela contém um arranjo de linhas e um arranjo de superfícies, além disto, é preciso que tenha métodos de seleção, para adicionar linhas, adicionar superfícies e propriedades como ID (número de identificação) para conectar os objetos OpenGL aos objetos da base de dados do T-CADE.

Para representar as linhas, foi criada uma classe chamada *TRepLIn*. Para representar as superfícies, foi criada uma classe de objetos chamada *TRepSup*. Ambas as classes são derivadas de uma classe genérica chamada *TRepBase*, que contém apenas as propriedades comuns às duas classes.

A Figura 23 mostra as classes gráficas criadas e como elas se relacionam. A classe *TGLRep* contém um arranjo de linhas que pode receber diversos objetos do tipo *TRepLin* e um arranjo de superfícies que pode receber diversos objetos do tipo *TRepSup*.

Uma superfície no T-CADE, para ser representada corretamente, é preciso mostrar a própria superfície (ou superfícies) e mostrar também suas linhas de construção. Estas linhas podem ser: a curva geradora, os paralelos da revolução, linhas importantes para sua representação, ou linhas que poderão ser criadas posteriormente, como por exemplo, linhas de interseção com outra superfície. Por isso, é importante que um objeto de representação como o *TGLRep* seja composto por outros objetos (tantos quantos forem necessários) para sua correta representação.

Figura 23: As novas classes Gráficas.
Fonte: O Autor.

Tanto o desenho de linhas, quanto o desenho das superfícies são resolvidos internamente em suas próprias classes: o *TRepLin* e o *TRepSup*.

A classe *TRepLin* representa as linhas na nova interface. Uma de suas propriedades mais importantes é conter uma classe de linhas já existente no GLscene: *TGLLinha* que é o que efetivamente é visto na tela. Além disto, são resolvidos internamente o número único de identificação do objeto (ID), se a linha está ou não selecionada, sua cor, espessura, número de segmentos, número de nós, entre outros.

A classe *TRepSup* é mais complexa. Uma de suas principais propriedades contém um objeto do GLScene chamado *TGLFreeForm*. Este objeto permite a definição de uma ou mais malhas de polígonos (triangulares ou quadrangulares) para sua representação. Toda a criação destes objetos, assim como sua destruição (para liberar a memória utilizada por eles) também é resolvida internamente na classe.

A classe *TRepSup* tem ainda como propriedade uma biblioteca de materiais (*TGLMaterialLibrary*) que permite a criação e configuração de materiais e texturas, limitado somente pela quantidade de memória do computador. Esta qualidade, aliada às classes de materiais do T-CADE, permite que os objetos apareçam com suas cores correspondentes (tanto nas linhas quanto nas superfícies).

Além disto, a classe *TRepSup* tem internamente rotinas para fazer a otimização da malha (evitar nós redundantes) e, ainda, rotinas para administrar os modos de visualização (*wireframe*, *smoothshade*, *ghost*, etc.).

A nova estrutura de classes criada para a representação de objetos tridimensionais no T-CADE permite que superfícies complexas, com um ou mais de um domínios (partes), sejam representadas corretamente, com suas respectivas cores (representando os materiais) e com um desempenho gráfico que permite uma boa interação e usabilidade.

4 IMPLEMENTAÇÃO COMPUTACIONAL

Para implementar as melhorias propostas na interface 3D, foi utilizado o software Delphi 2007 juntamente com a Biblioteca Gráfica GLscene, especialmente projetada para a utilização no ambiente de programação Delphi. A implementação da nova interface foi direcionada para duas questões específicas: a melhoria do desempenho gráfico e a melhoria da interatividade. A partir destes pontos busca-se, então, uma melhor usabilidade do software.

Foi criada uma nova estrutura de classes de representação dos objetos 3D utilizando a biblioteca gráfica GLscene buscando a melhoria do desempenho gráfico. As novas classes gráficas são propriedades dos objetos existentes na estrutura do T-CADE. Objetos lineares (uma dimensão) têm um tipo de representação diferente de objetos planos (duas dimensões) e objetos tridimensionais, portanto, estas classes gráficas devem atender às diferentes necessidades de representação dos objetos existentes e dos objetos ainda a serem criados.

Com a introdução das funcionalidades de uma biblioteca gráfica, foi possível a implementação de algoritmos de seleção de objetos mais robustos, que permitem uma melhor interação, possibilitando não só a melhoria da interatividade como o desenvolvimento de novas ferramentas que dependem de uma seleção de objetos rápida, eficiente e precisa.

Uma vez implementada a nova biblioteca gráfica, foi possível também a criação de novas ferramentas gráficas que permitem uma melhor interatividade com os objetos 3D e, consequentemente, melhor usabilidade. Foram implementadas técnicas de manipulação direta dos objetos selecionados, permitindo a restrição da movimentação e da cópia do objeto selecionado aos eixos cartesianos, facilitando assim sua manipulação em 3D. Outra intervenção foi a criação de uma ferramenta de visualização semelhante ao *ViewCube* (Khan, et al. 2008), uma ferramenta para visualização 3D que facilita a definição de vistas pré-estabelecidas na cena. Também foi criado o QuadMenu, uma ferramenta de visualização do tipo “menu de contexto” para o controle e seleção das ações da câmera.

Um dos requisitos de projeto era que as duas interfaces (a original e a nova) pudessem funcionar simultaneamente, para isso, foi preciso criar novas propriedades nas classes de objetos existentes para que fossem representados na nova interface, além disto, foi preciso adaptar algumas das rotinas atuais da geração da geometria aos novos algoritmos, sem que elas perdessem suas funcionalidades.

4.1 SELEÇÃO DE OBJETOS

Na OpenGL, a seleção de objetos se dá pela intersecção de vetores no espaço tridimensional. A Biblioteca GLscene já traz algoritmos de seleção de objetos baseados neste princípio. Mesmo assim, ainda é preciso muito trabalho para implementá-los na interface gráfica, para que funcionem conforme o esperado.

As classes gráficas (assim como as classes de objetos geométricos do T-CADE) trazem, entre outras propriedades, uma propriedade booleana (verdadeiro/falso) para a seleção dos objetos, ou seja, quando um objeto é selecionado, sua propriedade “*Sel*” fica verdadeira, e um índice referente à identificação daquele objeto é colocado em um arranjo de objetos

selecionados. Esta propriedade fica vinculada a eventos (métodos) que modificam a aparência dos objetos selecionados, fornecendo um feedback visual ao usuário sobre sua ação de seleção.

A seleção de objetos pode ser feita de três maneiras: clicando diretamente sobre o objeto, através de uma janela Cheia (*Window selection*) e através de uma Janela tracejada (*Crossing selction*). Ao se clicar sobre um objeto, este fica selecionado e sua aparência se modifica, suas linhas e suas superfícies, se houver, ficam imediatamente alaranjadas, indicando a seleção.

O modo de seleção de janela cheia é feito da esquerda para a direita e uma janela (quadro) azul transparente com borda de linha contínua é desenhada na tela. O primeiro clique inicia a janela e um segundo clique no canto oposto da janela termina a seleção. Neste modo, somente os objetos totalmente contidos dentro da janela são selecionados. A Figura 24 ilustra este conceito.

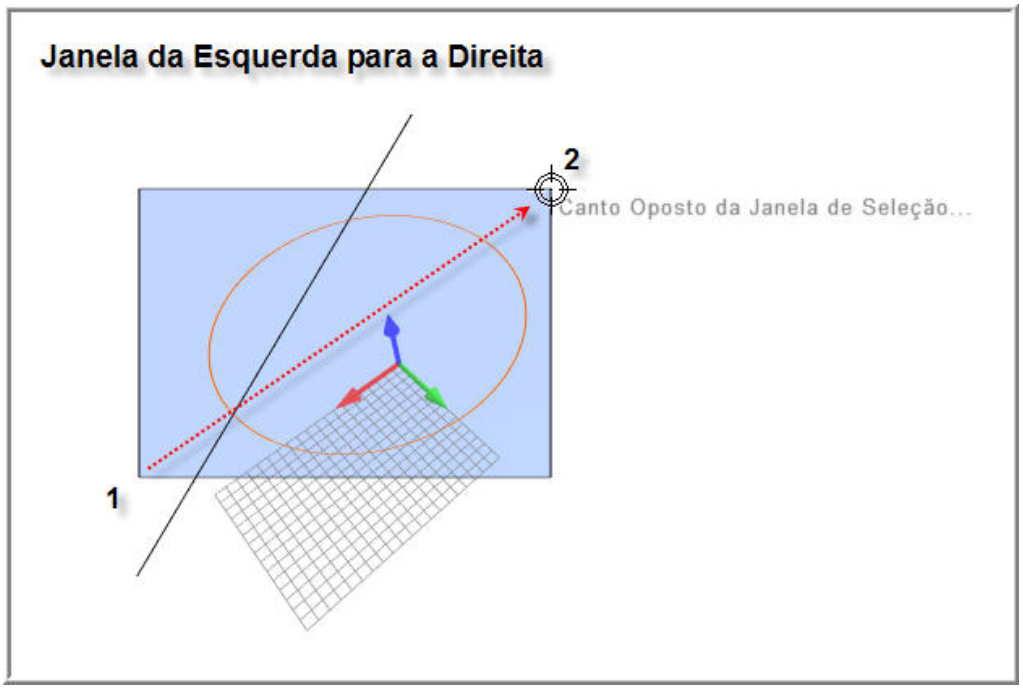


Figura 24: Janela Azul Cheia, da Esquerda para a Direita.
Fonte: O Autor.

No modo de seleção de Janela tracejada, a criação da janela é feita da direita para esquerda e uma janela verde transparente com borda pontilhada é desenhada na tela. Neste modo, qualquer objeto tocado pela janela é selecionado. A Figura 25 mostra uma janela tracejada.

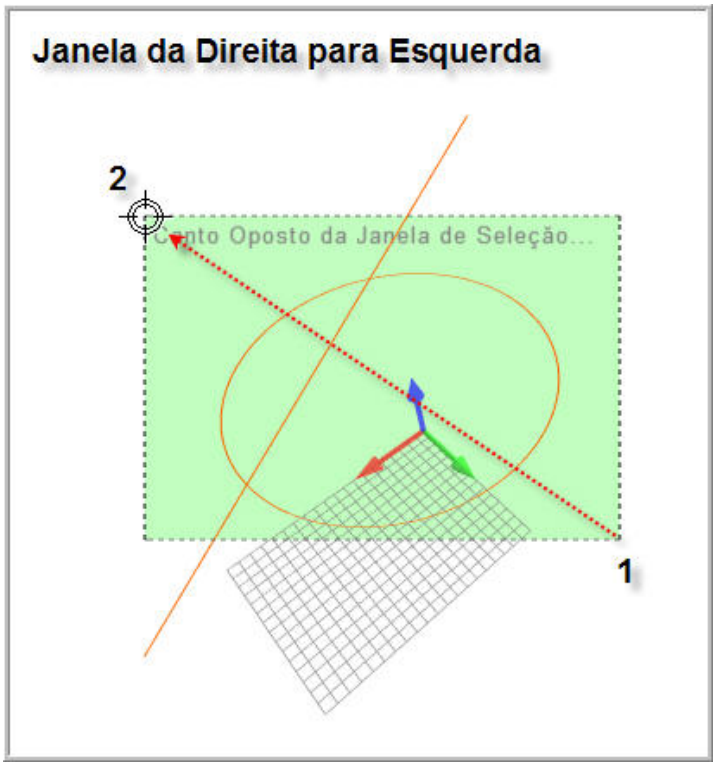


Figura 25: Janela verde tracejada, da direita para a esquerda.
Fonte: O Autor.

A escolha das cores, direções da criação da janela de seleção e opções de seleção (*Crossing ou window*) foram baseadas no software AutoCAD®, um dos mais utilizados no desenvolvimento de projetos, aproveitando, desta forma, a cultura do usuário e seus modelos mentais já estabelecidos. O software Rhinoceros® também utiliza os mesmos conceitos. Utilizar conceitos já estabelecidos em outros programas para a seleção de objetos facilita a memorização e a intuitividade, alguns dos conceitos básicos da usabilidade conforme Nielsen (1993).

Estas opções de seleção, além de tornarem o processo mais ágil, aumentam o controle do usuário na interação com o programa, principalmente quando se trabalha em cenas complexas com dezenas de objetos.

4.2 LOCALIZAÇÃO DO PONTO 3D

Os algoritmos de intersecção da biblioteca GLscene (RayCast) têm, entre outras características interessantes, a capacidade de saber a posição 3D do ponto clicado sobre uma superfície, o que é fundamental para a implementação de novos algoritmos com uma maior interatividade. Esta era, até então, uma limitação do T-CADE. Infelizmente esta capacidade se aplica somente a superfícies e não às linhas (uma curva qualquer). Mas, no T-CADE, já foram implementados algoritmos de intersecção entre retas, que funcionam perfeitamente em qualquer posição do espaço e poderão suprir esta deficiência.

Na interface original do T-CADE, na seleção de retas e curvas já são identificadas as posições paramétricas do ponto clicado sobre o objeto e, também, suas coordenadas globais.

Usando então uma solução semelhante na nova interface, um objeto do tipo *TReta* (derivado do *TLinha*) é colocado no espaço da câmera na posição do cursor e normal à tela de projeção. A partir desta reta, pode-se calcular a intersecção com qualquer outra curva no espaço 3D e saber exatamente o ponto clicado sobre o objeto. Este é um recurso importante em comandos como dividir uma linha em um determinado ponto ou, mesmo, desenhar uma nova linha a partir de um ponto qualquer pertencente à linha existente.

Para as superfícies, foi criado um algoritmo específico para a identificação do ponto 3D sobre a superfície, de tal maneira que fossem mostrados tanto o ponto 3d apontado pelo cursor quanto o vetor normal à superfície naquele ponto. Para demonstrar esta capacidade, foi criada temporariamente uma representação visual que consiste de uma esfera que é criada com centro no ponto 3D sobre a superfície e uma seta 3D indicando o sentido do vetor normal conforme mostrado na Figura 26.

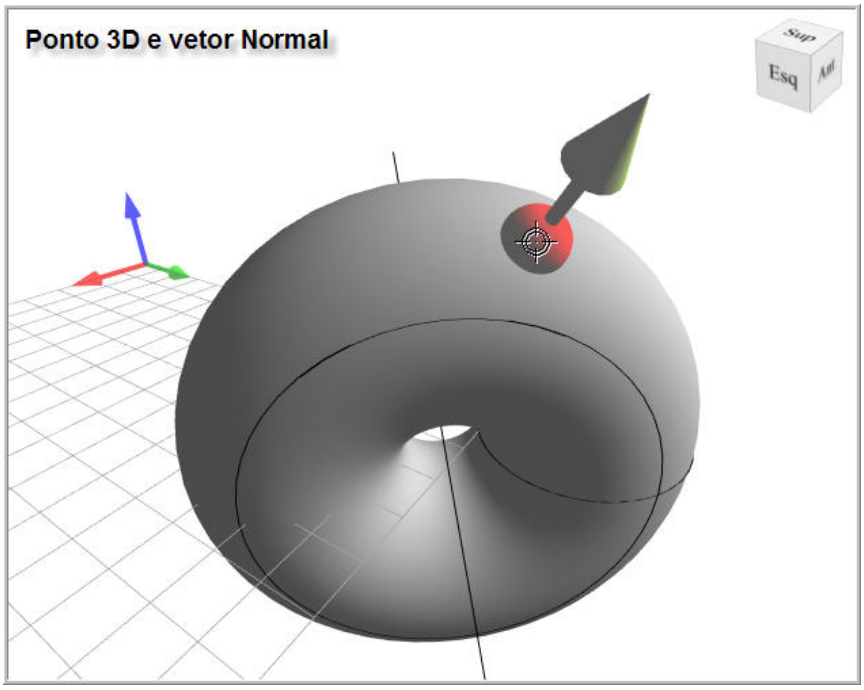


Figura 26: Esfera e Seta 3D, indicando o ponto 3D e o vetor normal à superfície.
Fonte: O Autor.

Na interface original a superfície não podia ser selecionada, somente suas linhas de representação, isto causava a impressão de erro na interação além da insatisfação do usuário. Segundo Nielsen (1993), é preciso minimizar a ocorrência de erros na interação. Os erros frustram os usuários e afetam a eficiência da interface de modo negativo.

Esta inovação está relacionada a três dos cinco critérios básicos da usabilidade apontados por Nielsen (1993): a minimização de erros, eficiência da interação e a satisfação do usuário. Esta inovação é uma melhoria na usabilidade e interatividade, permitindo a seleção de objetos diretamente sobre a sua superfície bem como possibilitando a criação de novas ferramentas que dependem desta precisão.

4.3 CRIAÇÃO INTERATIVA DE OBJETOS

No desenho de objetos, a interatividade é fundamental para o controle das ações do usuário. Ao desenhar uma linha, a linha deve ser mostrada durante a operação de desenho, para que usuário possa avaliar se o resultado mostrado é o resultado esperado por sua ação e, então, decidir aceitar ou não aquele resultado.

Um exemplo disto é a melhoria da rotina de desenho do Arco utilizando a opção "Centro-Início-Fim". Na interface original, somente depois de fornecidos os três pontos (início, centro e fim do arco) é que o arco era desenhado, frequentemente causando uma surpresa ao usuário com o resultado obtido, uma vez que para os mesmos três pontos, existem duas soluções possíveis (o arco de ângulo maior e o de ângulo menor).

A Figura 27 mostra duas situações onde os ângulos iniciais e finais são os mesmos e os resultados além de diferentes são apenas mostrados quando o comando é finalizado.

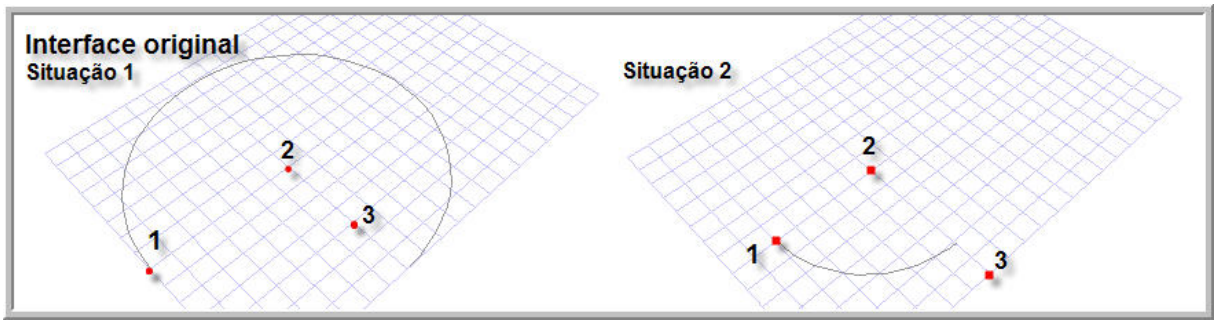


Figura 27: Arco Início, Centro e fim na Interface Original.
Fonte: O Autor.

Na nova interface, este problema foi resolvido através da interatividade. Para um melhor funcionamento do comando, a ordem de inserção dos dados também foi alterada. Ao fornecer o primeiro ponto (centro) uma linha já fica presa ao cursor indicando a linha de início do arco (ângulo inicial), ao fornecer o segundo ponto (definindo o início do arco), o arco começa a ser desenhado acompanhando o movimento do cursor.

A Figura 28 mostra o mesmo comando na nova Interface, o arco é desenhado durante o processo e acompanha o movimento do cursor.

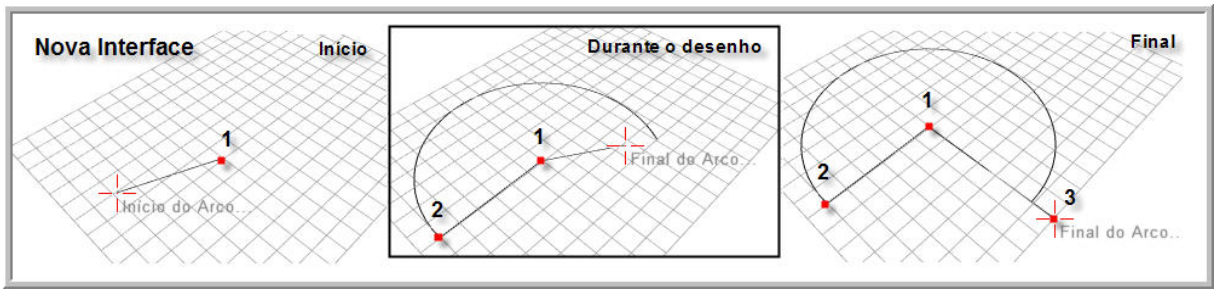


Figura 28: Arco Centro, Início e fim na Nova Interface.
Fonte: O Autor.

O sentido do arco é alterado cada vez que o arco (acompanhando o cursor) passa pelo seu início, permitindo que o usuário controle o resultado da ação. Ao fornecer o último ponto, o comando é então finalizado.

Um dos critérios da avaliação heurística da usabilidade segundo Nielsen e Molich (1990) é a visibilidade do sistema, isto significa que se deve manter o usuário informado do resultado de suas ações, dando um feedback imediato às ações do usuário, facilitando a interação e minimizando os erros.

4.4 MANIPULAÇÃO DIRETA DE OBJETOS

Outra aplicação importante da interatividade é na modificação e manipulação direta de objetos. Atualmente, o T-CADE oferece duas possibilidades de modificação de objetos: mover e copiar. Para utilizá-las, é necessário acionar o respectivo comando, selecionar os objetos, terminar a seleção, indicar o ponto de base e, finalmente, indicar o novo ponto (relativo ao primeiro) para onde o objeto vai ser movido ou copiado. Este estilo de interação é importante por uma questão de precisão e controle na modelagem de objetos, mas torna-se frustrante quando a precisão não é tão importante quanto a agilidade do processo.

Para suprir esta necessidade de interação, foi criada a possibilidade de mover e/ou copiar os objetos enquanto selecionados, tornando a manipulação dos objetos mais ágil e direta. Por exemplo, ao clicar sobre um objeto, pode-se movê-lo mantendo o botão esquerdo do mouse pressionado e movimentando o cursor em qualquer direção. Assim, em um único movimento (clicar, arrastar e soltar), o objeto é selecionado e movido para a nova posição, aumentando a produtividade do usuário.

O uso combinado deste movimento com teclas modificadoras para a restrição de eixos amplia seu potencial de manipulação, inclusive em termos de precisão. Pode-se mover um objeto ao longo do eixo X ou Y, mantendo-se a tecla SHIFT pressionada, é desenhada uma linha de cor vermelha ou verde, indicando os eixos X e Y respectivamente, informando imediatamente ao usuário o resultado de suas ações.

Pode-se ainda mover um objeto ao longo do eixo Z utilizando, para isto, a combinação com a tecla CTRL, neste caso uma linha azul é desenhada, indicando o movimento ao longo do eixo Z. Para fazer uma cópia, utilizam-se os mesmos procedimentos descritos anteriormente combinados também com a tecla ALT.

Quando o modo copiar está ativado, o cursor se modifica, é desenhado um sinal de adição ao lado do cursor original. A Figura 29 mostra esta situação combinada com o uso de teclas de atalho.

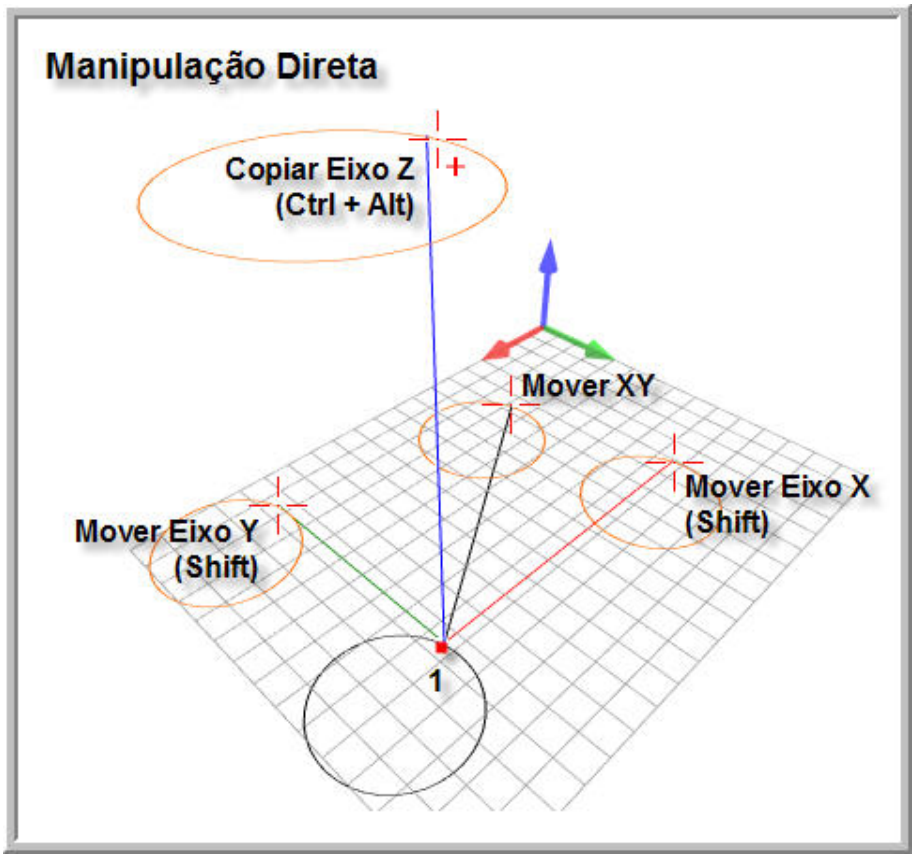


Figura 29: Manipulação direta na Nova Interface.
Fonte: O Autor.

Uma das ações mais intuitivas na interação tridimensional é clicar e arrastar um objeto. De acordo com Nielsen (1993), a intuitividade é um dos critérios mais importantes da usabilidade. Esta técnica afeta também, além da intuitividade, a eficiência da interação tridimensional.

Assim, o uso de teclas de modificação combinados a uma interação direta com objeto selecionado, sem a necessidade de invocar comandos específicos de transformação, torna o trabalho do usuário muito mais prático e rápido, aumentando sua produtividade e, o mais importante neste caso, sua satisfação com a ferramenta digital.

4.5 VISUALIZAÇÃO 3D

Para o controle da visualização 3D, foi criada uma versão do ViewCube (KHAN *et al.*, 2008). Esta ferramenta permite a mudança de vistas pré-estabelecidas de maneira simples e intuitiva, evitando a desorientação comum em ambientes abstratos 3D.

A partir de um cubo no canto superior direito da tela, com o nome das vistas nas faces, o usuário tem o controle e a indicação da orientação da cena. Quando usado como controlador da posição de visada, clicando sobre as faces, arestas ou cantos do cubo, tanto o cubo quanto a cena mudam sua orientação ajustando-se à vista correspondente. Quando a cena é modificada por outros métodos de movimentação de câmera, o cubo se ajusta para indicar a nova orientação. A transição entre as mudanças de vistas é feita de modo animado, de maneira a

suavizar a mudança e evitar a desorientação do usuário, causada por mudanças bruscas da cena. As vistas foram chamadas de **Sup** (Superior), **Inf** (Inferior), **Ant** (Anterior), **Pos** (Posterior), **Esq** (Lateral Esquerda) e **Dir** (Lateral Direita). A Figura 30 mostra o cubo e como ele funciona, por exemplo, clicando sobre a face Anterior do cubo, tanto o cubo quanto a cena se ajustam à nova vista em 3D.

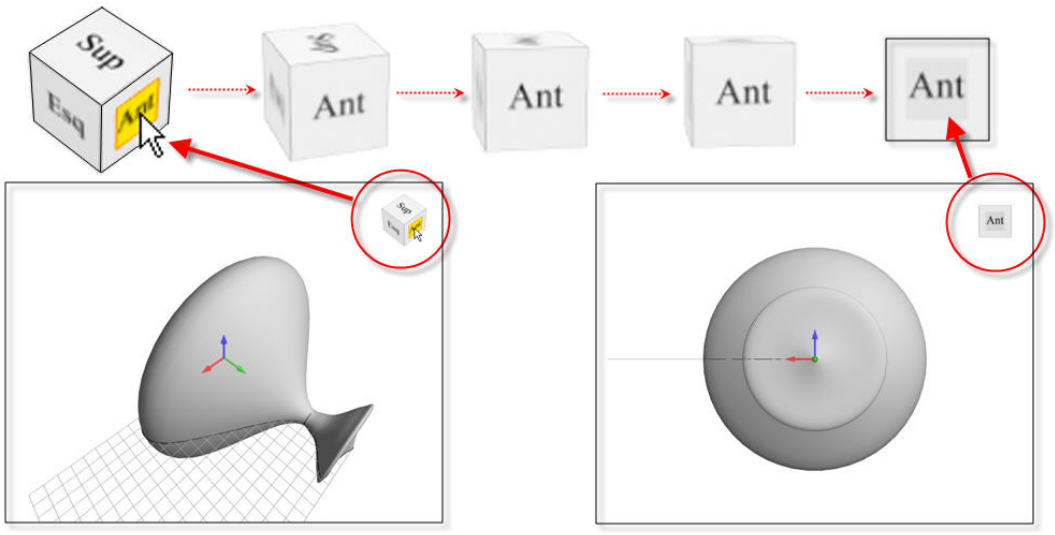


Figura 30: O Cubo de orientação e visualização.
Fonte: O Autor.

Além das vistas ortogonais nomeadas anteriormente, é possível clicar sobre as arestas do cubo para conseguir a vista intermediária entre duas vistas. A Figura 31 mostra a transição entre as a posição original e nova vista escolhida. Após o clique na aresta entre a vista esquerda e a vista anterior, a vista e o cubo se ajustam para refletir a nova posição de câmera.

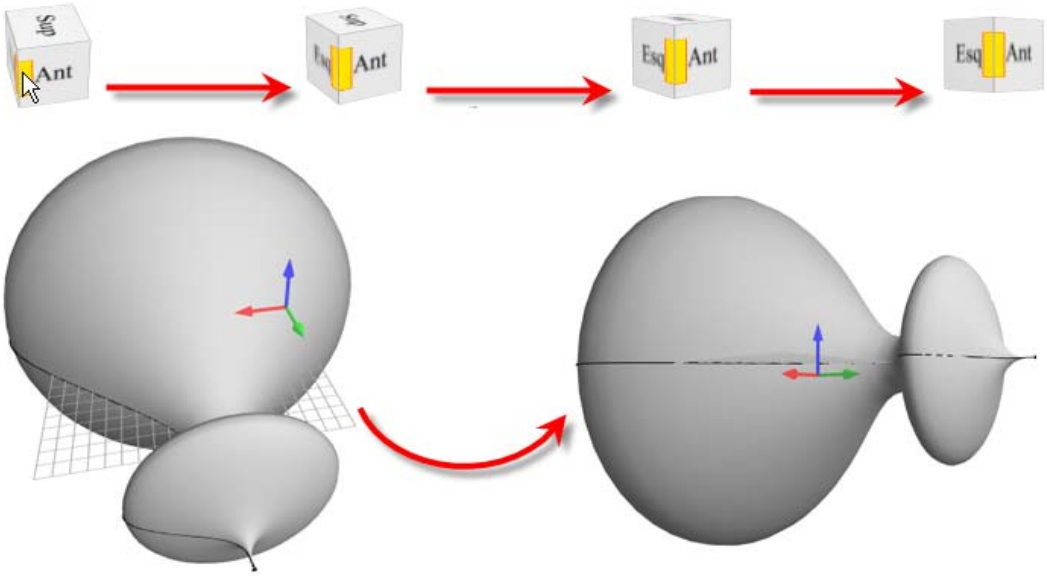


Figura 31: Vista a partir da aresta do cubo.
Fonte: O Autor.

Ao apontar um dos vértices do cubo, tanto o cubo quanto a cena se posicionam igualmente em uma vista axonométrica, segundo aquela orientação. A Figura 32 ilustra esta situação.

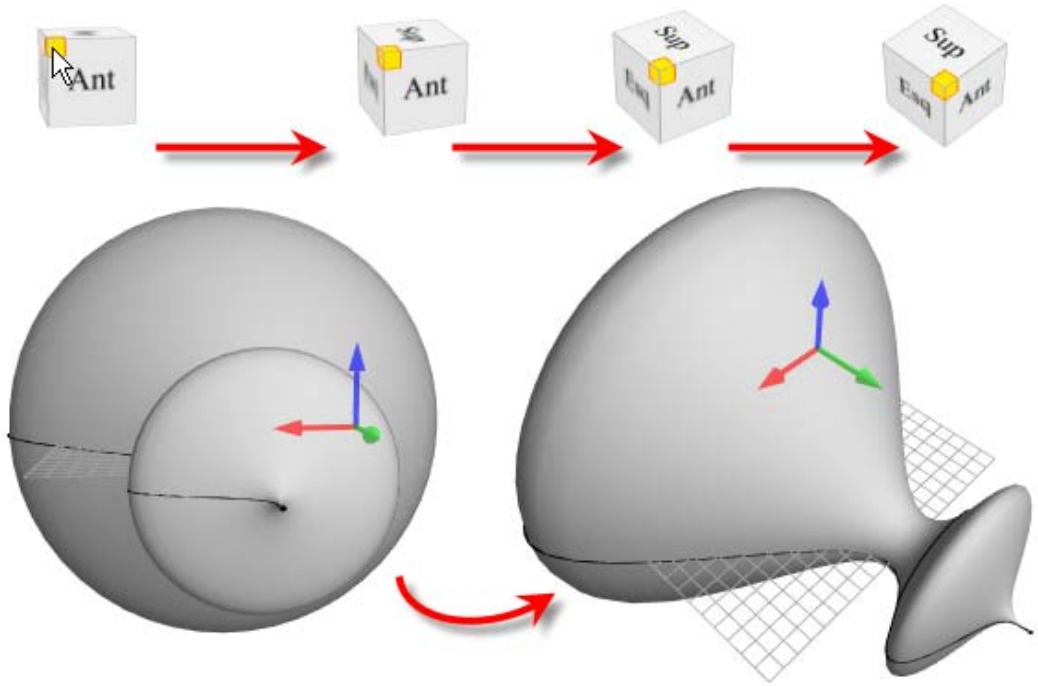


Figura 32: Clique sobre o vértice do cubo.
Fonte: O Autor.

O cubo é formado por vinte e seis (26) objetos, seis (6) objetos representando as faces do cubo, oito (8) objetos representando os vértices do cubo e doze (12) objetos representando as arestas do cubo. A Cada um destes objetos esta mapeada uma determinada posição de câmera. A posição original da câmera fica registrada em uma variável, e assim que a posição da câmera é atualizada pelo clique em um destes objetos, a interface faz a animação da posição anterior para a nova posição, minimizando a desorientação do usuário.

Entende-se que um design interativo tende a melhorar a qualidade da experiência do usuário. Outra melhoria importante foi a criação do “QuadMenu”, uma ferramenta de controle de câmera a partir de um menu circular onde os comandos são dispostos em quadrantes, facilitando a seleção dos comandos e sua memorização. Seu funcionamento é baseado na interação gestual do mouse (ZELEZNIK , 1999), mas com um diferencial: a seleção acontece, não através de gestos pré-definidos, mas da direção do movimento e distância percorrida pelo cursor, inovando pela simplificação e pelo menu visual para a escolha de comandos.

Para que estas ferramentas pudessem existir, foram implementados comandos de controle de câmera como zoom interativo, zoom window, translação da câmera, rotação da câmera, entre outros. A Figura 33 mostra a ferramenta QuadMenu e os diferentes cursores para cada uma das ferramentas de visualização.

Figura 33: QuadMenu e cursores das ferramentas de câmera.
Fonte: O Autor.

A nova técnica propõe que sejam concentrados os comandos comuns de visualização em um único evento do botão. Isto facilita a memorização e agiliza a utilização da ferramenta. Além disto, a atenção do usuário permanece na área de trabalho, sem a necessidade de desviar o olhar ou procurar o botão da ferramenta correspondente àquela função.

Uma vez pressionado o botão da direita do mouse, um menu de seleção aparece ao redor do cursor do programa em forma de um alvo, que é claramente dividido em quatro quadrantes. O menu circular é mostrado com quatro opções para o controle da câmera, orbitar ao redor do alvo, mover-se sobre o plano XY, mover a altura da câmera ou mover a cena no plano da tela.

A zona circular definida na parte interna do cursor forma uma zona neutra onde o usuário poderá decidir que direção tomar e, portanto, qual ferramenta de visualização escolher. Ainda com o botão pressionado, o usuário leva o cursor em direção ao comando escolhido e este fica selecionado imediatamente.

Uma vez que o cursor for arrastado para fora da zona neutra, alcançando o ícone do comando no quadrante escolhido, a ferramenta é selecionada e seus efeitos passam a ser visíveis imediatamente. A partir deste momento, o menu desaparece e o cursor é mudado para indicar a ferramenta que está sendo utilizada. A mudança do cursor para indicar a ferramenta escolhida ajuda a comunicação com o usuário.

O menu de escolha só aparece depois de ¼ de segundo (250 milissegundos) que o botão seja mantido pressionado, liberando assim a função do simples clique do botão da direita para um futuro menu de contexto. Para os usuários experientes, que já conhecem a posição correta (direção) das ferramentas, é possível simplesmente clicar e arrastar em uma determinada direção para selecionar a ferramenta, sem a necessidade de esperar que o menu apareça. Depois de acionado o comando, o ícone do quadrante se transforma no respectivo cursor dauqela ferramenta, indicando o início da ação. Neste caso, da ação já ser escolhida diretamente, o menu nem sequer é mostrado.

A Figura 34 ilustra o funcionamento desta ferramenta.

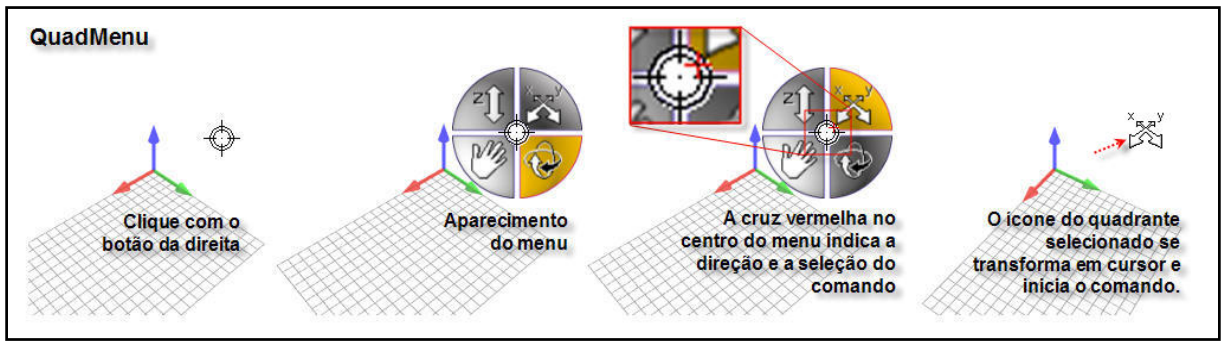


Figura 34: Acionamento em um único movimento, clicar e arrastar na direção da ferramenta desejada.
Fonte: O Autor.

As ferramentas de visualização Cubo e QuadMenu matêm o usuário constantemente informado da posição da cena, facilitam a seleção dos comandos de visualização, aumentam a eficiência da interface e favorecem a memorização das posições dos comandos no menu. Todos estes fatores foram direcionados à melhoria da usabilidade.

4.6 MODOS DE VISUALIZAÇÃO

A nova biblioteca gráfica permitiu que novos modos de visualização fossem implementados, aumentando o controle das opções de visualização do usuário, melhorando a clareza e o entendimento da cena tridimensional.

É possível mostrar os objetos sem a malha de visualização, em modo aramado (*WireFrame*), em modo aramado com Volume (*VolumeWire*), em modo aramado com linhas ocultas (*Hide*), em modo Volumétrico Facetado com linhas e sem linhas (*ShadedLines* e *FlatShade*), em modo Volumétrico Suavizado (*SmoothShade*) e em modo Transparente (*Ghost*).

Os diferentes modos de visualização são importantes durante o processo de modelagem tridimensional. Frequentemente, se precisa editar objetos que estão ocultos por outros objetos na cena, quando se muda o ponto de vista para tentar corrigir este problema, outros objetos se colocam entre a câmera e o objeto em questão, dificultando a edição e aumentando o tempo de trabalho e a frustração do usuário por não conseguir realizar a tarefa.

Com diferentes modos de visualização é possível, por exemplo, utilizar um dos modos que representam os objetos em estrutura de arame (sem malha, *wireframe* ou *volumewire*) para conseguir ver e editar o objeto através de outros objetos. O modo transparente também pode ajudar na visualização destes objetos ocultos em outros modos de visualização, esta possibilidade melhora o entendimento da cena mostrando a relação de distancia entre todos os objetos e a perspectiva da cena.

A Figura 35 mostra os quatro primeiros modos de visualização.

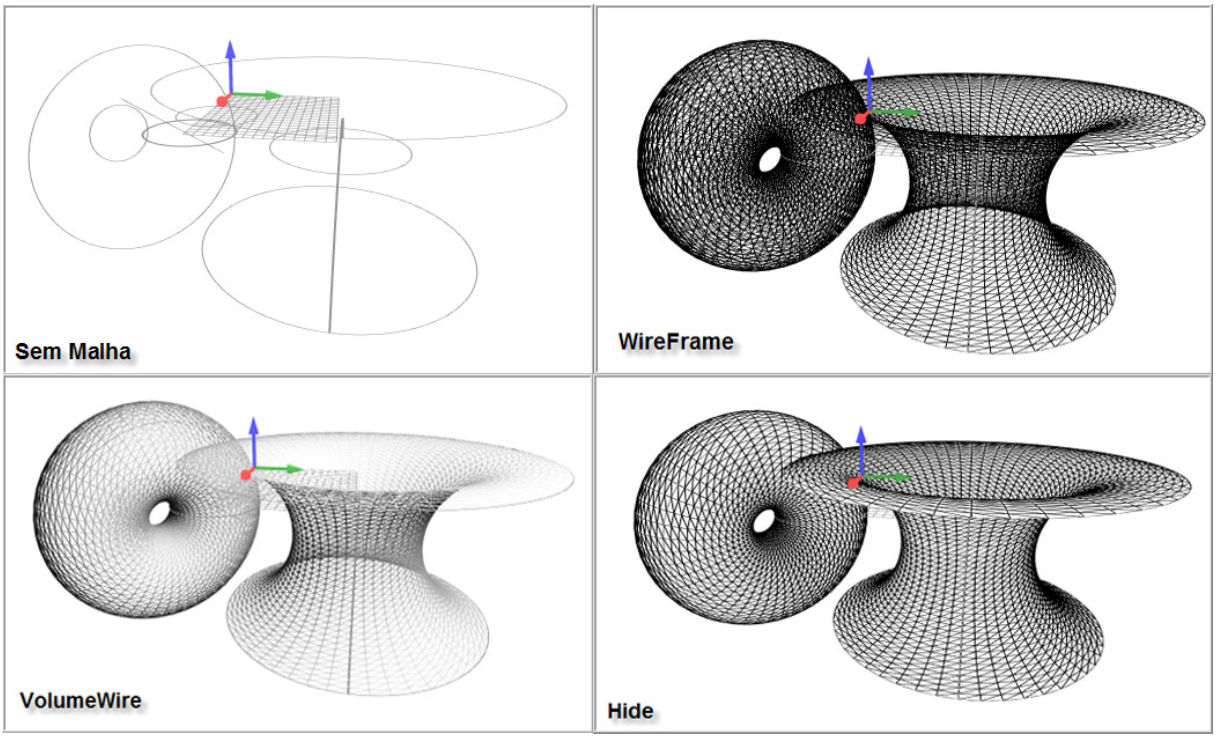


Figura 35: Modos de visualização: Sem malha, WireFrame, Volume Wire e Hide.
Fonte: O Autor.

A Figura 36 mostra os modos de visualização restantes.

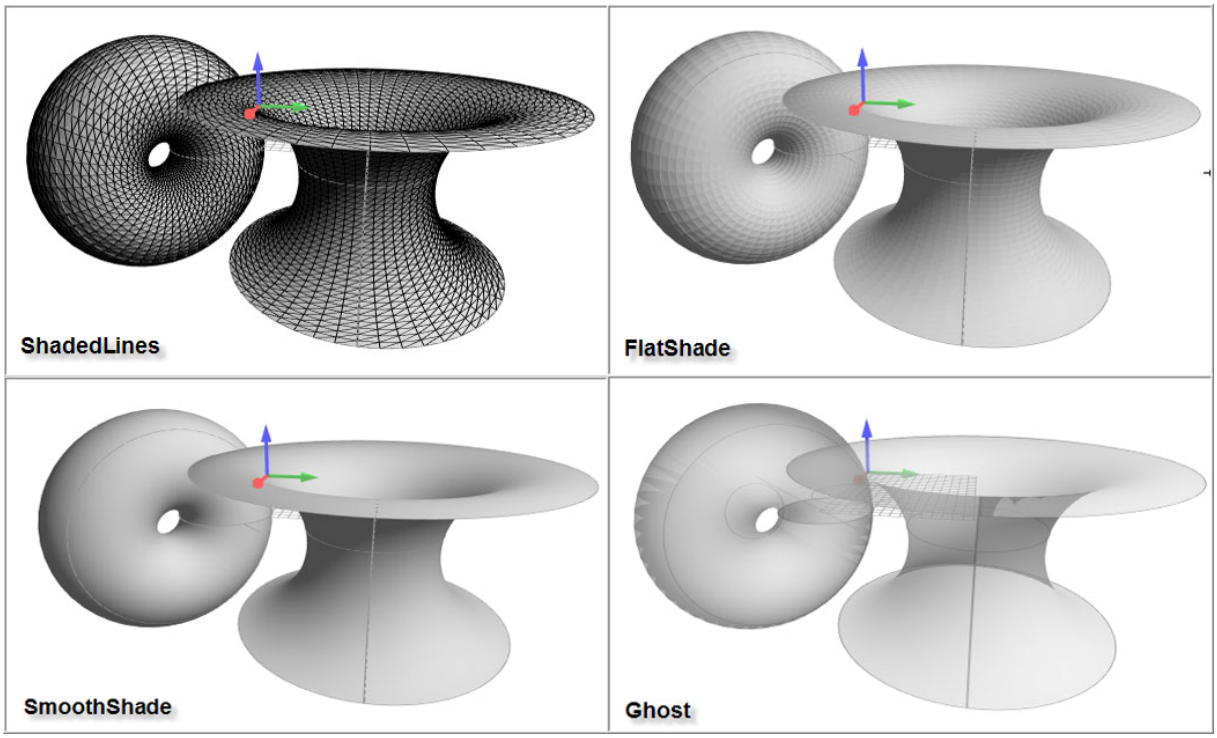


Figura 36: Modos de Visualização: Shaded Lines, FlatShade, SmoothShade e Ghost.
Fonte: O Autor.

Quando um objeto é criado, especialmente as superfícies, sua representação 3D aparece imediatamente conforme o modo de visualização corrente e a cor do material utilizado; os diferentes modos de visualização permitem diferentes percepções dos objetos; é possível ainda controlar a resolução da malha de visualização, controlando assim, caso necessário, o desempenho gráfico do aplicativo.

A Figura 37 mostra uma superfície de revolução com dois níveis diferentes de resolução de malha utilizando o modo de visualização *Hide* (remoção de linhas ocultas). Na imagem à esquerda, a superfície paramétrica é divida em vinte (20) linhas e vinte (20) colunas, resultando em uma malha com oitocentos (800) triângulos. Na imagem à direita, a mesma superfície é representada por uma malha com cinqüenta divisões em cada direção, resultando em uma malha com cinco mil (5.000) triângulos.

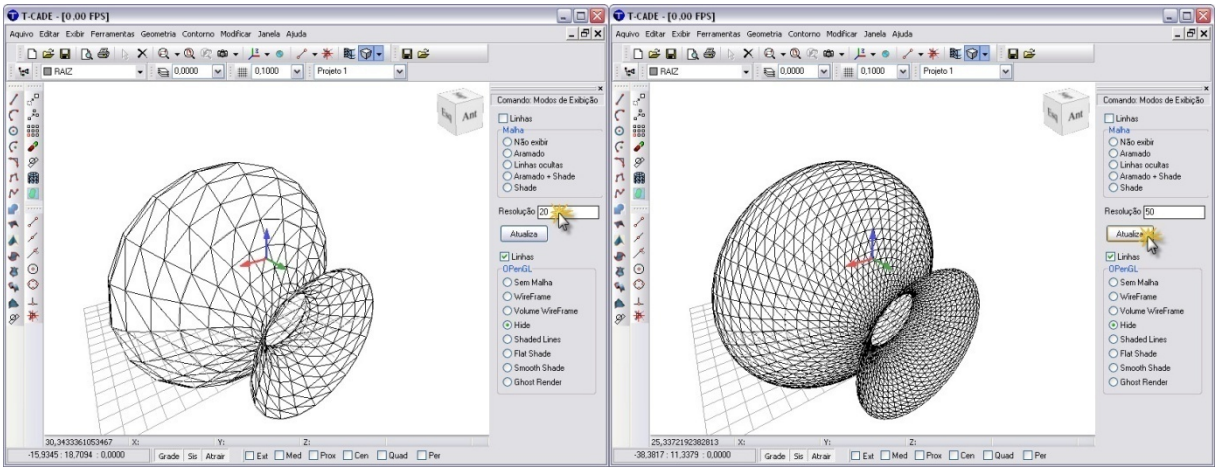


Figura 37: Mudando a resolução da malha.
Fonte: O Autor.

Os diferentes modos de visualização têm desempenhos gráficos diferentes em função dos algoritmos específicos usados em cada implementação. Alguns modos podem facilitar a visualização de objetos que estejam escondidos atrás de outro objeto. Outros podem, também, ter grande influência no desempenho gráfico, principalmente no que diz respeito à eficiência computacional do próprio modo de visualização.

Assim, os modos de visualização são importantes para a representação dos objetos, influenciando diretamente o entendimento da cena tridimensional e o desempenho da interface.

4.7 BIBLIOTECA DE MATERIAIS

O T-CADE já permitia, conceitualmente, a utilização de materiais. Na interface original isto se refletia apenas na cor do objeto. Na nova interface, criou-se uma biblioteca de materiais padrão para representar corretamente os diferentes modos de visualização. Quando o programa é iniciado, esta biblioteca padrão é criada. A partir da definição de um novo material na estrutura de dados do T-CADE, uma nova biblioteca de materiais é criada para

representá-lo e fica associado a ele na estrutura de dados. A Figura 38 mostra a interseção entre duas superfícies definidas com materiais diferentes.

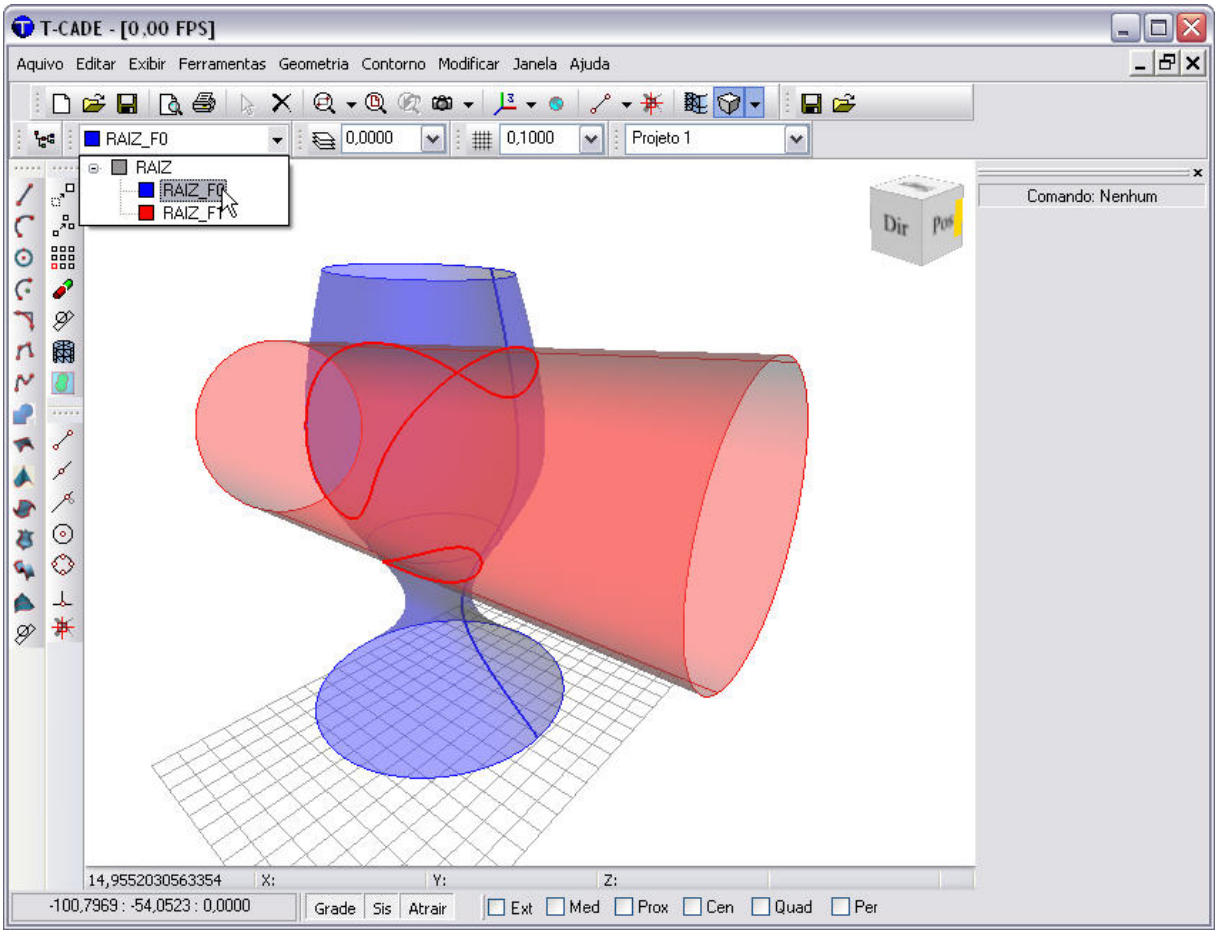


Figura 38: As cores dos objetos evidenciam os diferentes materiais.
Fonte: O Autor.

Com a nova interface, cada vez que um objeto é criado, é associado a um material, e este material esta ligado a uma biblioteca de materiais, com o potencial de aplicação de texturas, inclusive. A aplicação de texturas nos objetos não faz parte dos objetivos desta pesquisa, portanto, não foram criadas as ferramentas necessárias para isto. No entanto, a tecnologia e a estrutura de dados necessários para a criação destas ferramentas já foram implementadas neste trabalho e permitem seu futuro desenvolvimento.

4.8 COMUNICAÇÃO COM O USUÁRIO

A interface de um aplicativo é o mediador entre o aplicativo e seu usuário, assim a comunicação entre eles deve ser clara e eficiente. O usuário deve saber sempre o que o programa espera que ele faça, especialmente no momento de fornecer os dados a um determinado comando.

Considerando que o olhar do usuário acompanha sempre o cursor, ao invés de se usar uma barra de status para instruções do usuário, como é usual em programas 3D, foi criado um

texto informativo logo abaixo do cursor, indicando, então, qual o próximo passo em um comando ou quais dados o programa espera que o usuário forneça naquele momento.

Esta abordagem evita que o usuário não perceba uma determinada informação relevante para a correta execução do comando, uma vez que esta informação está colocada na área de foco visual do usuário. Este texto informativo mostrou-se muito útil no treinamento de novos usuários. A Figura 39 mostra o texto informativo no cursor durante a execução do comando “Arco por três pontos”.

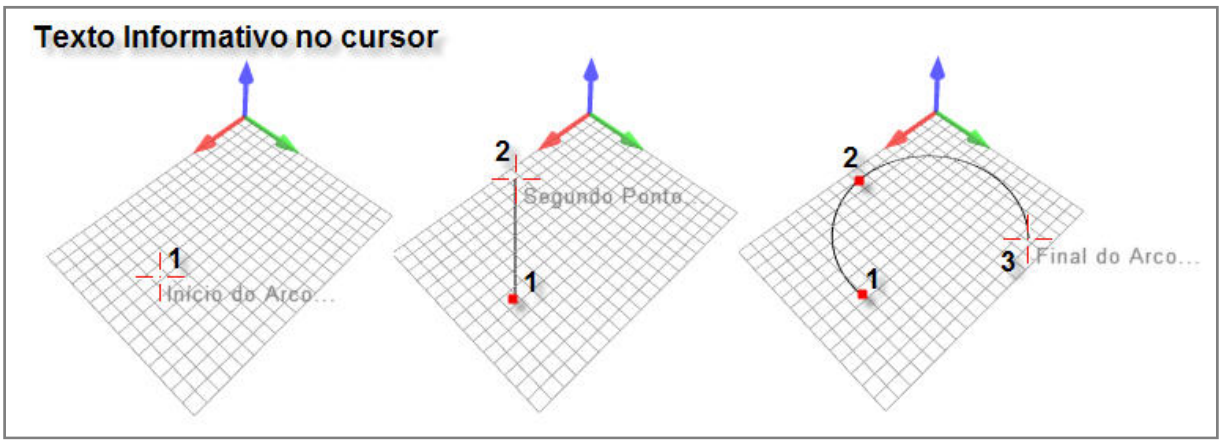


Figura 39: Texto informativo no cursor.
Fonte: O Autor.

O texto informativo, bem como os diferentes cursores, foram criados com o intuito da melhoria da usabilidade no diz respeito à visibilidade do sistema. Manter a comunicação com o usuário facilita o aprendizado da ferramenta, o treinamento de novos usuários, minimiza os erros e consequentemente aumenta a eficiência e a produtividade.

5 RESULTADOS

A implementação de uma biblioteca gráfica na plataforma de desenvolvimento T-CADE trouxe diversas melhorias em termos de desempenho, interatividade e principalmente usabilidade do software.

A utilização de uma interface gráfica baseada em OpenGL permite que o software T-CADE tenha um melhor desempenho gráfico quanto à velocidade de atualização da tela, que um maior número de polígonos seja manipulável e visível em tempo real, uma correta representação de superfícies, resolvendo problemas complexos tais como: visibilidade, transparência, sombreamento, seleção de objetos e iluminação.

Para uma avaliação comparativa do desempenho gráfico em ambas interfaces, foram criadas rotinas com "*Timers*" para a medição da taxa de atualização da tela, medida em quadros por segundo (FPS – *frames per Second*). A Tabela 3 mostra os resultados obtidos:

Tabela 3: Desempenho gráfico medido em Quadros por Segundo (FPS).

Objetos	Interface Original	Nova Interface
Mil Linhas	60 – 80 FPS	30 - 40 FPS
10 Mil Linhas	15 – 25 FPS	10 – 20 FPS
400 Faces	10 – 20 FPS	40 – 50 FPS
900 Faces	05 – 15 FPS	40 – 50 FPS
2500 Faces	02 – 10 FPS	40 – 50 FPS
5 Mil faces	00 – 05 FPS	40 – 50 FPS
10 Mil faces	0 FPS (Mais de 1 seg.)	35 – 45 FPS

Fonte: O Autor.

Considerando a simplicidade da interface original, ela apresenta um melhor desempenho no desenho de linhas, uma vez que as rotinas de otimização da biblioteca gráfica OpenGL na nova interface continuam sendo calculadas, mas têm pouco ou nenhum efeito no desenho de linhas, porque na representação das linhas não tem-se nenhum objeto oculto por outro, tudo é visível. Quando se trata da representação de superfícies, o desempenho da nova interface é muito superior. Além de manter uma taxa de atualização de tela estável, independente das circunstâncias, mantém-se regularmente acima do padrão estabelecido para o cinema, que é de 30 quadros por segundo. Embora utilizada nos testes, é improvável a situação em que existam apenas linhas, pois em um projeto 3D, os objetos são representados por superfícies e/ou sólidos, sendo as linhas utilizadas para a construção destes.

Quanto à interatividade, a interface implementada permite que, durante a criação do objeto, o resultado das ações do usuário seja imediatamente mostrado, antes mesmo de o comando ser finalizado, permitindo que o usuário corrija suas ações caso necessário.

Um dos exemplos mais significativos da melhoria apresentada na nova interface se verifica na criação de superfícies. Na interface original, quando uma superfície é criada, são representadas somente as isolinhas da superfície (linhas que mantêm uma coordenada paramétrica constante), mas sua malha de representação não é mostrada automaticamente. A Figura 40 mostra as linhas representantes de uma superfície de revolução, a curva geradora e os paralelos no início, meio e final da curva na interface original.

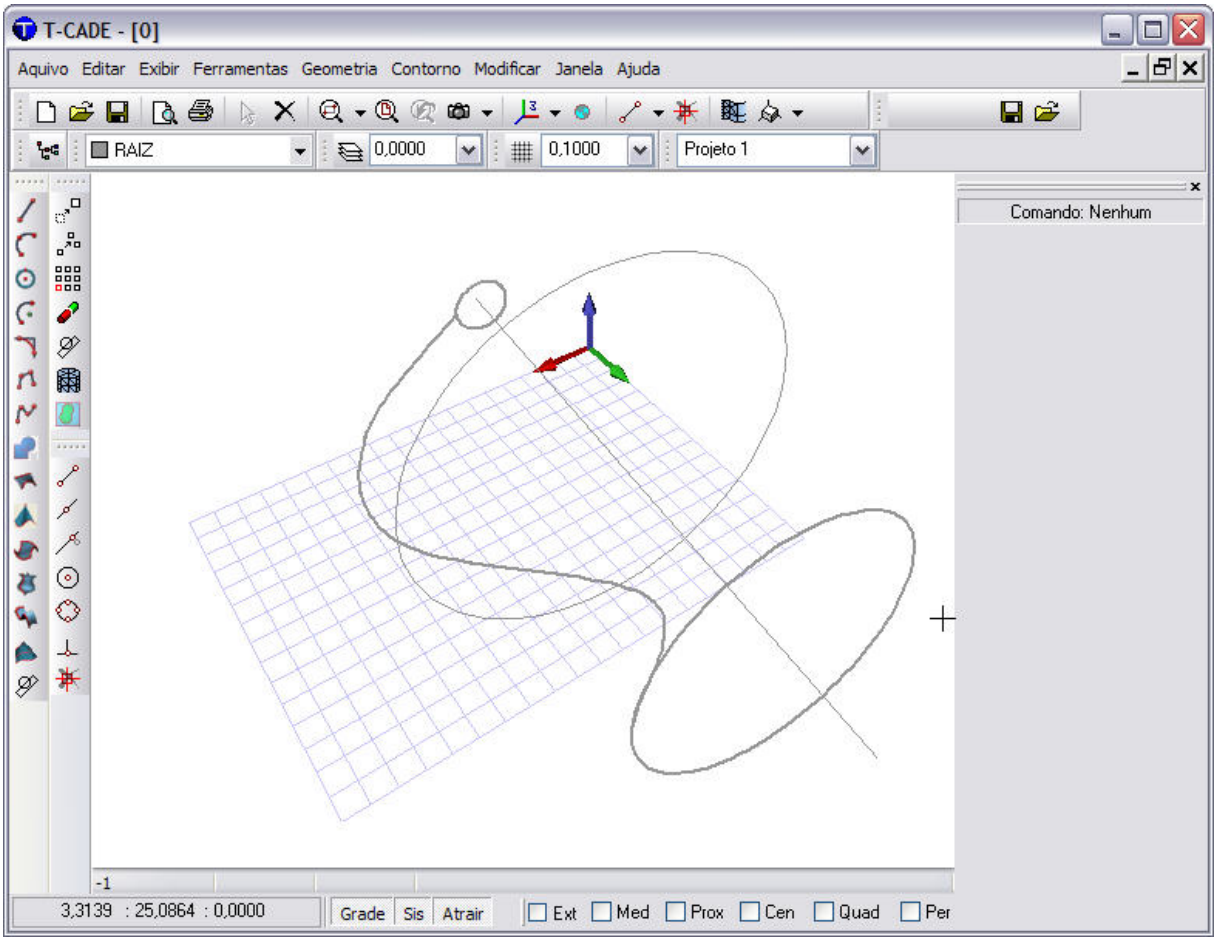


Figura 40: Representação de uma superfície de Revolução na interface original.
Fonte: O Autor

Assim, para que o usuário pudesse visualizar a malha da superfície era necessário o acionamento de outros comandos. Após três cliques com o cursor, a superfície era mostrada como esperado, primeiro para mostrar um painel de modos de visualização e depois para escolher o modo de visualização e, finalmente, para atualizar a malha de representação, conforme mostra a Figura 41. Isto era feito, justamente, para contornar as deficiências no desempenho, uma vez que a representação de superfícies era o que limitava a interface original. É importante ressaltar, ainda, que os modos de visualização da interface original são

modos básicos, não permitem a suavização da superfície nem mesmo a aplicação de transparências.

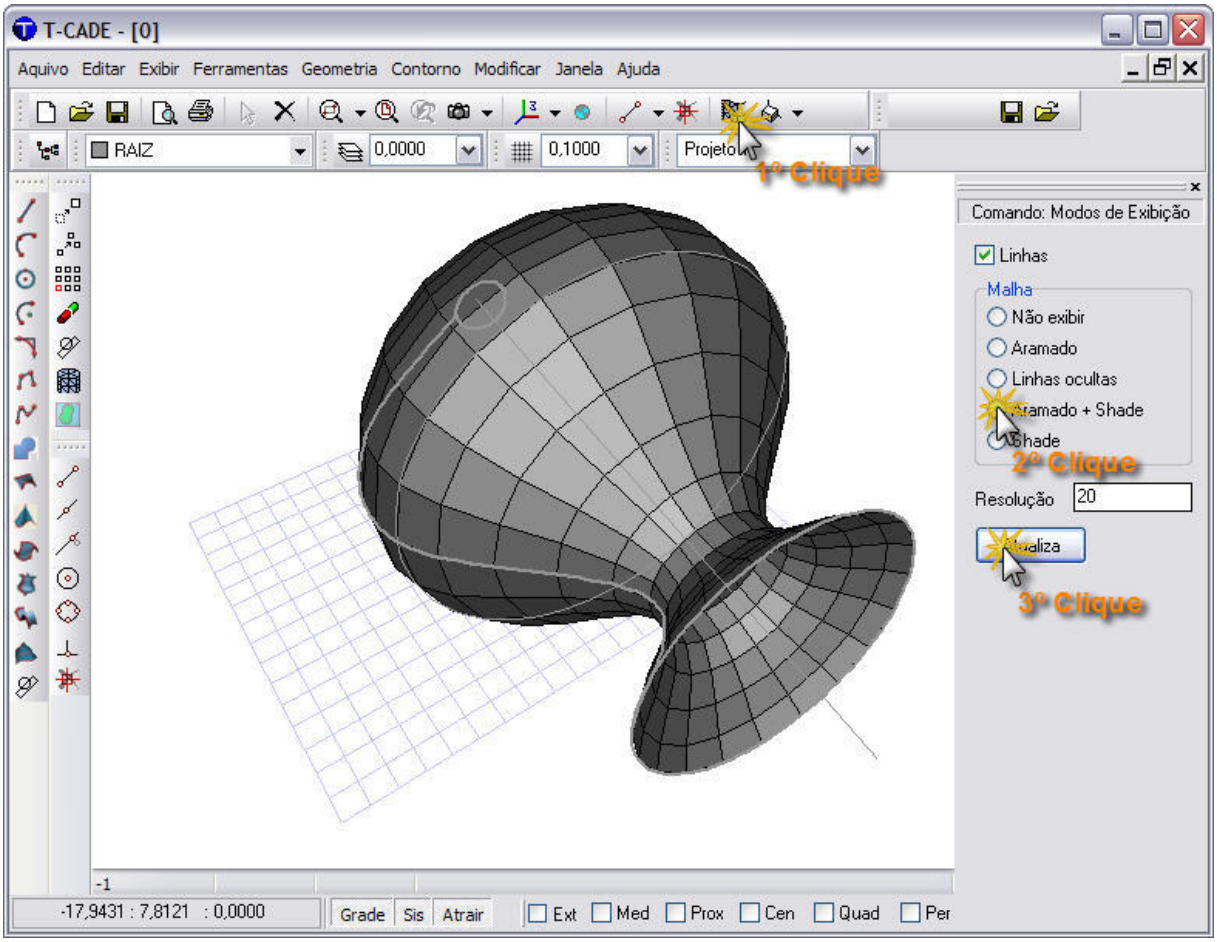


Figura 41: Três passos necessários para a geração da malha de visualização da superfície na interface original.
Fonte: O Autor.

A criação de superfícies na nova interface já mostra automaticamente a malha da superfície, bem como suas linhas de construção, sem a necessidade do acionamento de novos comandos para sua representação. O modo de visualização padrão é o *SmoothShade* (modo Suavizado), conforme mostrado na Figura 42.

Além do modo de visualização suavizado (*SmoothShade*), outra inovação foi o modo *Ghost* (transparente). Este modo permite ao usuário uma melhor compreensão da cena, tanto da geometria dos objetos como, da relação entre elas, uma vez que as superfícies não ocultam completamente outras superfícies, que de outra forma não estariam visíveis. Um bom exemplo do uso do modo de visualização transparente é a visualização das linhas de interseção entre as superfícies.

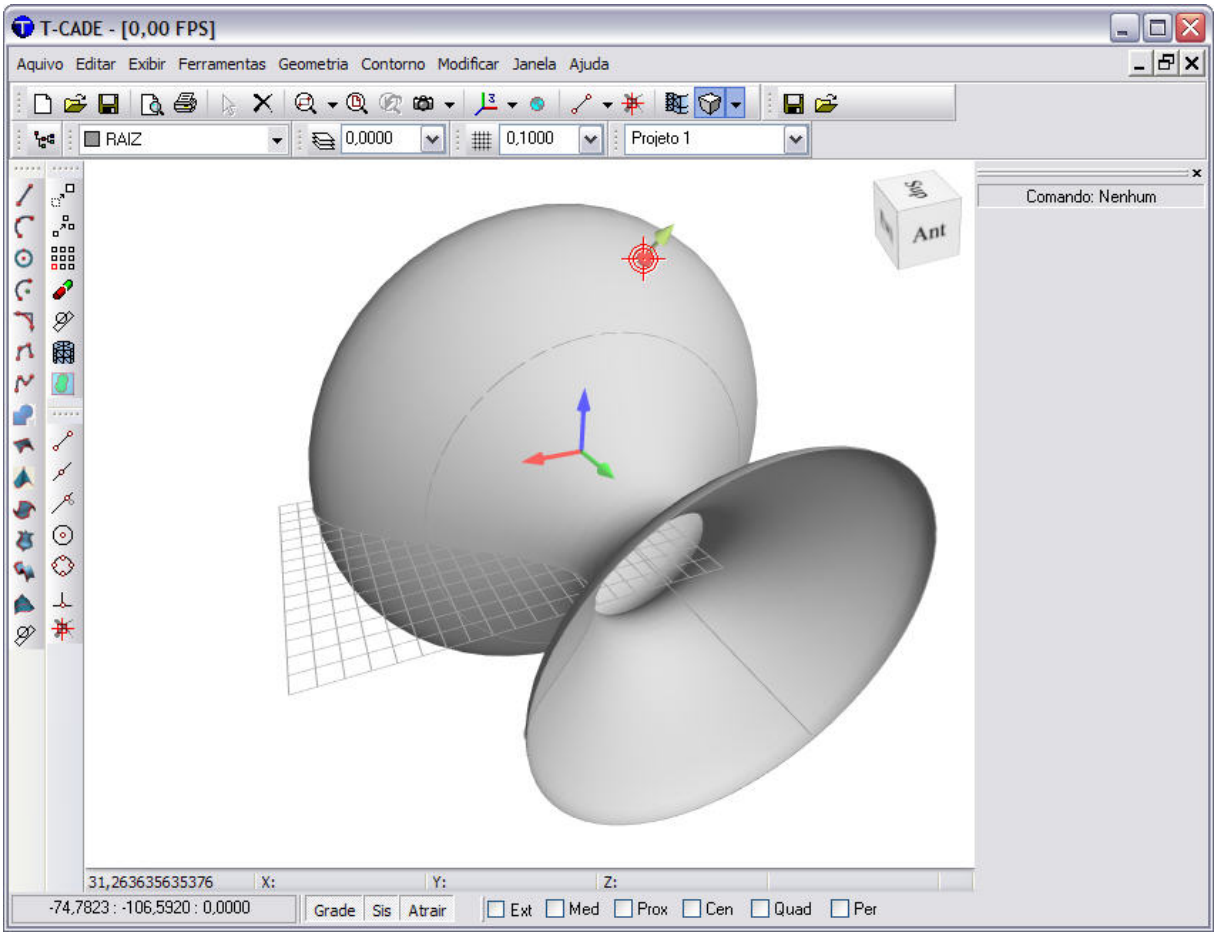


Figura 42: Criação de uma superfície de Revolução na nova interface.
Fonte: O Autor.

A plataforma T-CADE pode determinar as linhas de interseção entre duas superfícies quaisquer e, a partir destas interseções, determinar os domínios da superfície (partes da superfície onde será gerada a malha). A Figura 43 mostra a interseção entre duas superfícies (linha em vermelho) por um ponto de vista que naturalmente ocultaria esta linha se as superfícies não fossem transparentes. Nesta imagem, também é possível ver a deformação em perspectiva da superfície cilíndrica, o que ajuda a perceber a tridimensionalidade da cena.

Utilizando um exemplo mais realístico, a Figura 44 mostra a sequência de modelagem de um avião, partindo-se dos perfis transversais e linhas de base até a geração das superfícies utilizando os comandos *Loft* e Revolução. Evidencia-se na imagem maior, a superfície que marca o corte do vidro da cabine e a superfície das asas penetrando no corpo do avião.

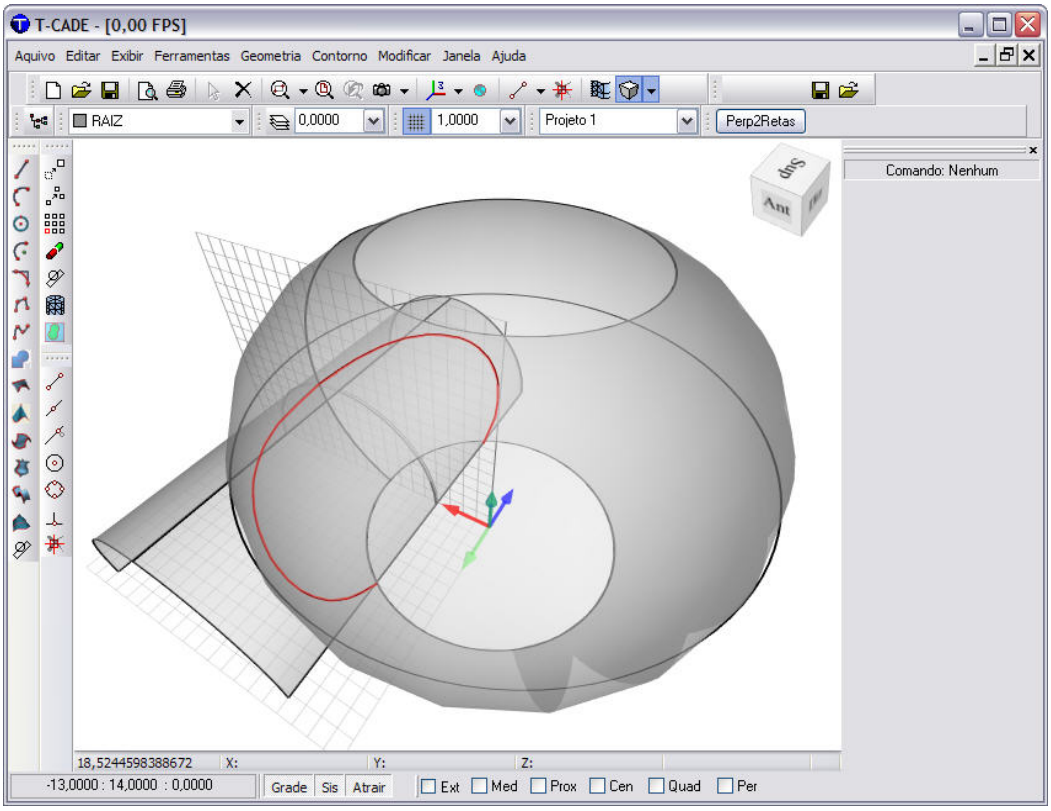


Figura 43: Modo Ghost mostrando a interseção entre superfícies.
Fonte: O Autor.

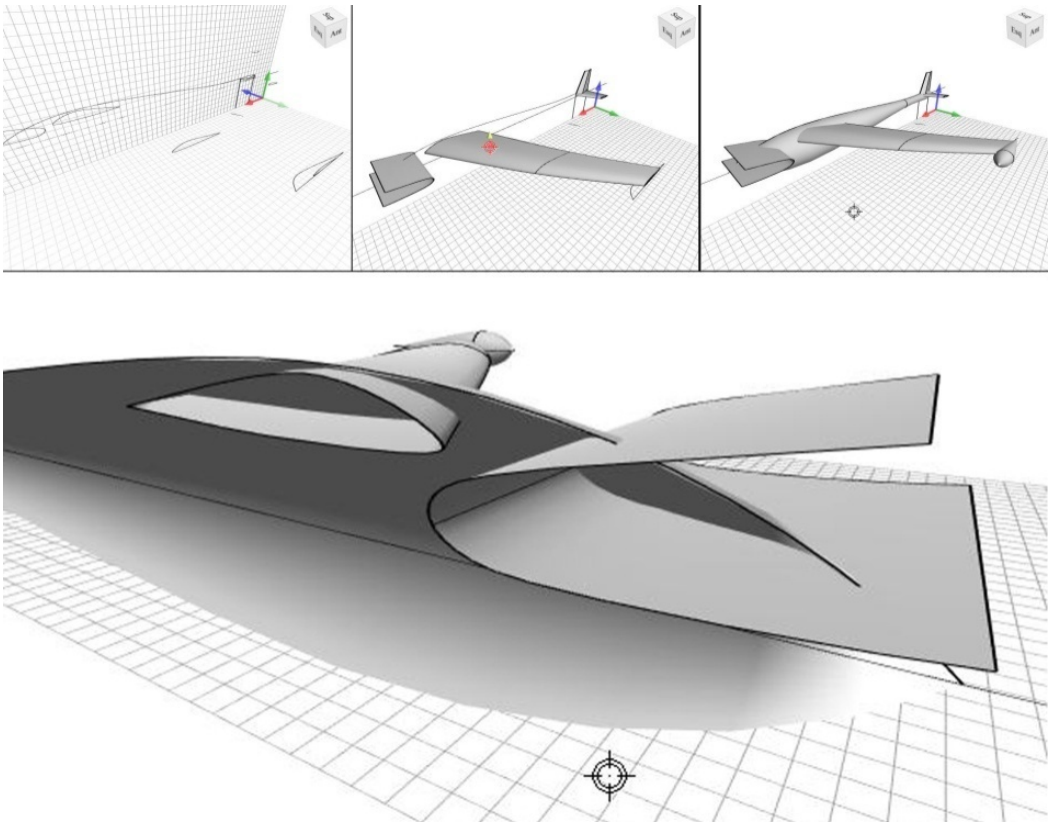


Figura 44: Modelagem com interseção de superfícies.
Fonte: O Autor.

A partir desta modelagem e depois de calculadas as interseções, na Figura 45, é possível perceber as interferências entre as asas e o corpo do avião utilizando-se o modo *Ghost*.

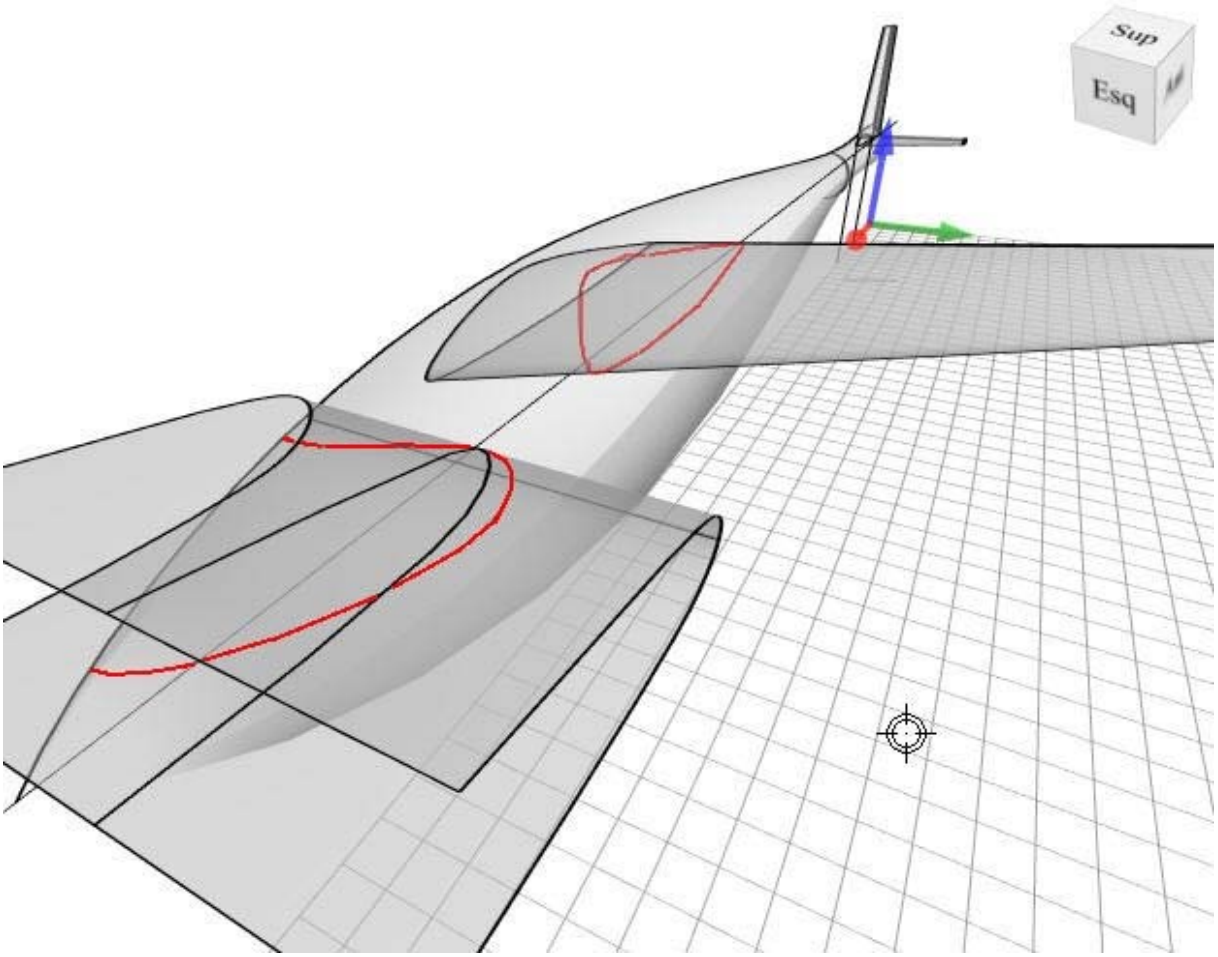


Figura 45: As linhas vermelhas mostram a interseção.
Fonte: O Autor.

Uma das principais funcionalidades do T-CADE é a geração de malhas não-estruturadas para análise de elementos finitos. Na criação das superfícies, uma malha simples de visualização é criada automaticamente conforme a resolução corrente e é apresentada conforme o modo de visualização corrente (*Hide, SmoothShade, Ghost*). A partir das interseções entre as superfícies, é possível gerar uma malha não-estruturada somente na parte relevante da superfície. A geração desta malha segue parâmetros específicos e a visualização dos resultados é fundamental neste processo. A Figura 46 mostra a malha não estruturada (malha para análise / laranja) no corpo do avião em contraste com a malha estruturada (de visualização / cinza) na asa do avião. Esta figura mostra, ainda, a parte do corpo correspondente ao vidro da gabine, já cortada.

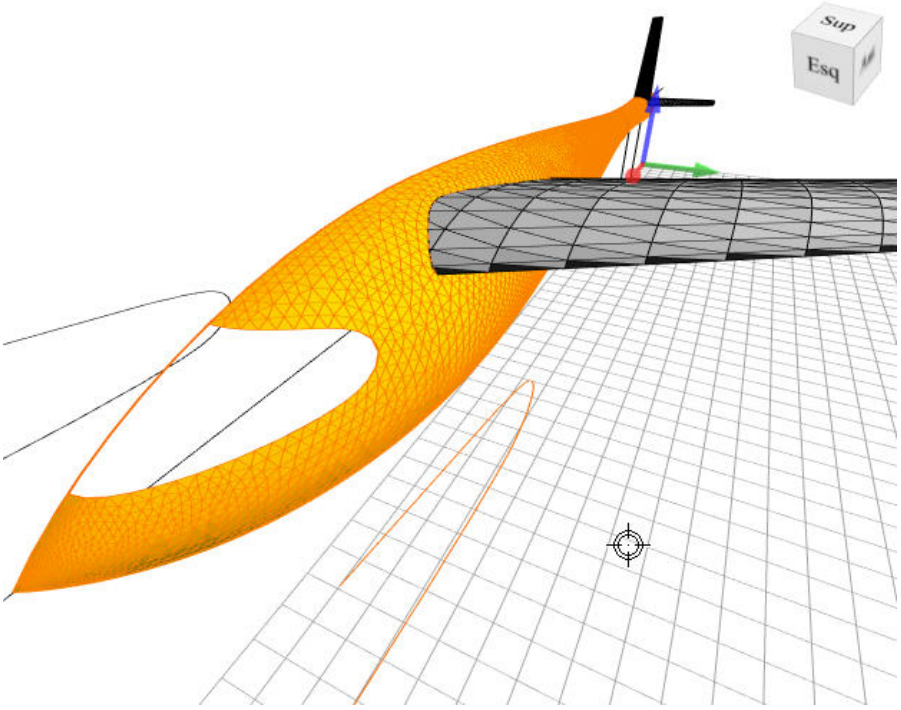


Figura 46: No modo ShadedLines é possível visualizar os elementos da malha.
Fonte: O Autor.

A Figura 47 mostra uma vista interna do corpo do avião, onde se pode notar a junção do corpo do avião com asa e a parte da asa (dentro do corpo) em que a malha não foi gerada. A figura foi criada utilizando o modo *SmoothShade*, que valoriza a suavização e o sombreamento da superfície.

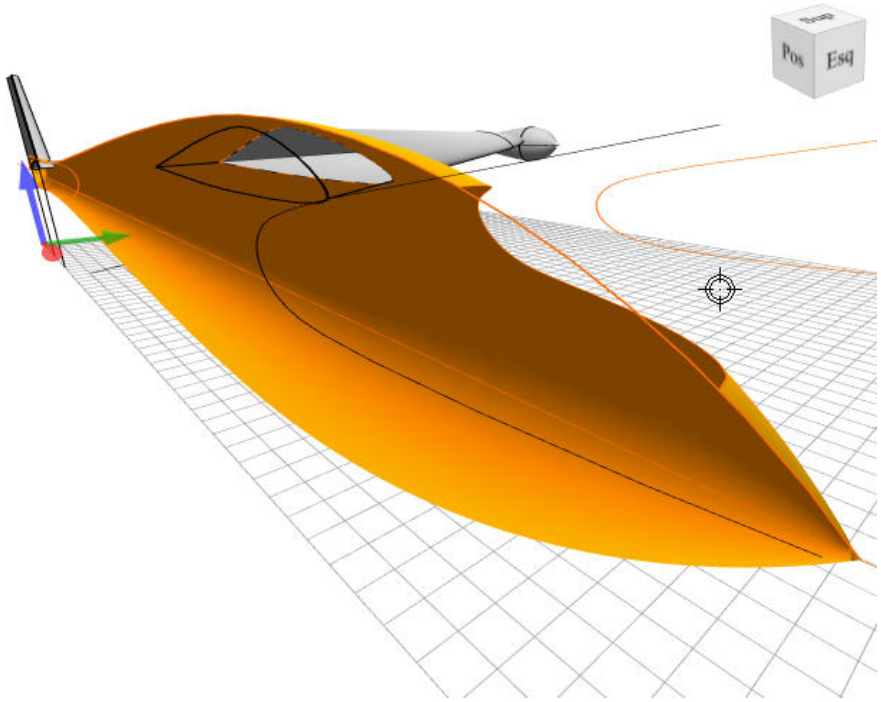


Figura 47: Interseção entre asa e corpo do avião.
Fonte: O Autor.

Outra melhoria na representação de superfícies é percebida na edição de superfícies. Na interface original, quando uma superfície é movida ou copiada, apenas suas linhas de representação acompanham o cursor e são representadas na nova posição, enquanto que sua malha permanece no mesmo local de origem. Em uma primeira instância, isto causava confusão no usuário inexperiente sobre a eficácia do comando. Além disto, a correta representação da superfície exigia que fossem acionados novamente os comandos de atualização da malha, para só então mostrar a malha de representação da superfície no local correto. A Figura 48 ilustra a representação de uma superfície quando movida na interface original.

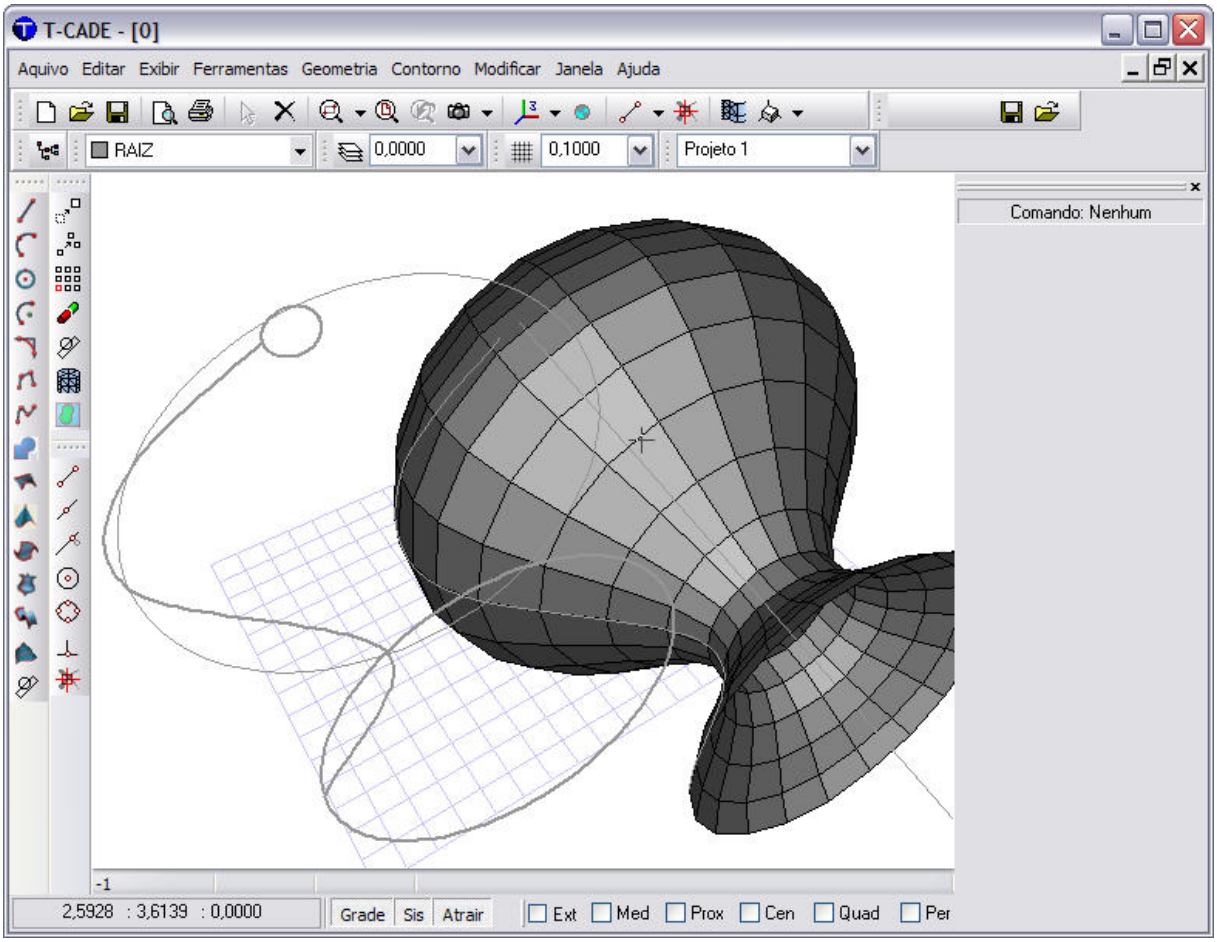


Figura 48: Objeto movido para esquerda, sua malha permanece no local de origem.
Fonte: O Autor.

Tanto para a ação de mover quanto para a de copiar, era necessário o acionamento do comando via barra de ferramentas ou menu, a seleção dos objetos, o término da seleção, a indicação de um ponto de base, e por fim, a indicação de um novo ponto para onde os objetos selecionados seriam movidos. Embora seja importante que existam comandos com esta sintaxe para a manipulação dos objetos com uma maior precisão (definição do ponto de base e do ponto alvo), é melhor que se tenham, além destas, outras opções mais diretas para este tipo de manipulação, por uma questão de praticidade. Frequentemente, o usuário apenas quer fazer uma cópia de um objeto para o lado, não importando tanto a precisão do deslocamento da cópia, mas sim a agilidade com que este processo é feito.

Na nova interface, opções para a manipulação direta de objetos foram criadas para que se pudesse selecionar, mover ou copiar os objetos diretamente a partir de um único movimento. Além desta melhoria, a sua representação também é atualizada constantemente ao longo do processo, mostrando não só a nova posição da superfície selecionada como também a posição original, auxiliando o usuário ao criar esta referência visual. Enquanto a superfície é movida, uma linha é mostrada indicando a trajetória entre o ponto de base e o ponto final, sendo que esta linha também é atualizada interativamente. Ao final do comando, a superfície na posição original desaparece, restando apenas a superfície selecionada na nova posição. A Figura 49 mostra a nova posição da superfície selecionada, a linha da trajetória do cursor e a posição original do objeto.

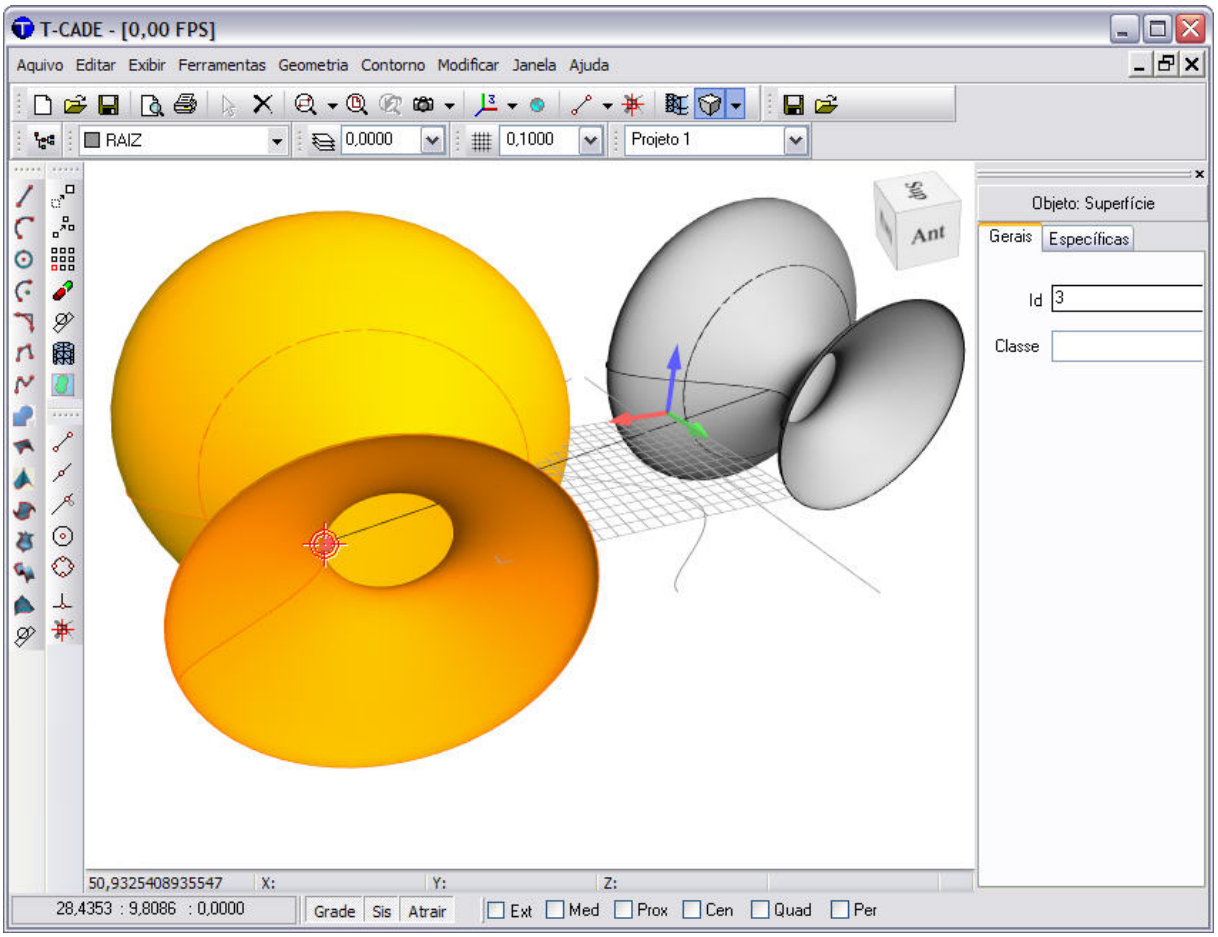


Figura 49: Superfície sendo movida na nova interface.
Fonte: O Autor.

A partir da combinação de teclas de modificação (Ctrl, Alt e Shift), é possível alternar entre o movimento no modo ortogonal (eixos X, Y e Z) e entre mover ou copiar, tudo isto ainda durante o movimento. Tanto o formato do cursor como a cor da linha da trajetória do cursor se adaptam às modificações do comando, indicando seu estado, conforme mostrado anteriormente na Figura 29 (pág. 89).

O uso destas combinações de teclas com os comandos de manipulação de objetos aumentou o controle do usuário, tornou mais ágil e facilitou o processo de transformação da

cena tridimensional. Neste caso, a redução do número de cliques com o mouse tem reflexo direto no tempo de execução do comando, permitindo que o usuário execute sua tarefa mais rapidamente e com maior precisão.

No que se refere à usabilidade, ao invés de fazer um longo estudo com muitos usuários, levando em conta a interface terminada, o que seria, além de demorado, muito dispendioso, optou-se por usar os recursos disponíveis para fazer, ao longo do desenvolvimento do projeto, inspeções de usabilidade nas diversas ferramentas e técnicas implementadas. Desta forma, estas ferramentas e técnicas puderam ser revisadas e melhoradas na medida em que os problemas de usabilidade foram sendo identificados.

6 CONSIDERAÇÕES FINAIS E CONCLUSÕES

Neste trabalho foi desenvolvido um protótipo de interface para interação tridimensional levando em conta conceitos de usabilidade, interatividade e desempenho gráfico. Para a melhoria do desempenho gráfico, foi implementada a biblioteca gráfica Glscene que, nos testes apresentados, manteve uma taxa de atualização de tela acima de 30 quadros por segundo (padrão de cinema) de modo relativamente constante.

Com a melhoria do desempenho gráfico, é possível trabalhar com até um milhão de polígonos (no modo de visualização menos eficiente) sem que a interface seja um limitador do processo de interação em tempo real.

O aumento da interatividade melhora a experiência do usuário e estabelece as condições necessárias para o desenvolvimento de novas ferramentas de criação e edição de objetos, imprescindíveis em programas para modelagem tridimensional.

Para a melhoria da interatividade, foram criadas duas ferramentas de visualização e implementadas técnicas de manipulação direta de objetos. Uma destas ferramentas foi o Viewcube, que permite a rápida mudança entre vistas pré-definidas de modo animado e de fácil utilização. A outra ferramenta de visualização, o QuadMenu, facilita a seleção de comandos de navegação (controle do movimento da câmera) e permite sua utilização de modo transparente (sem interrupção dos comandos acionados).

Além destas ferramentas interativas, a representação das superfícies também mostra uma melhoria significativa em relação à interface original, tanto nas possibilidades dos modos de visualização (suavização e transparência), como na representação interativa dos objetos, nas etapas de criação e na edição de linhas e superfícies. Os diferentes modos de visualização possibilitam ao usuário escolher como as superfícies serão representadas. Esta escolha tem influência direta no desempenho e, por consequência, na interatividade.

Em casos em que a visualização dos triângulos da malha não estruturada (para análise de elementos finitos) seja necessária, pode-se usar, por exemplo, o modo ShadedLines (arestas visíveis e sombreamento). Este modo é o que tem o maior custo computacional em termos de desempenho gráfico. Quando a visualização da superfície for mais importante do que a disposição dos triângulos da malha, pode-se usar o SmoothShade (suavizado) ou o modo Ghost (transparente), melhorando muito o desempenho, a interatividade e a representação visual da superfície.

A criação das ferramentas gráficas em conjunto com as técnicas de manipulação tornou a experiência do usuário mais interativa, indicando que os objetivos foram alcançados de forma satisfatória. Com uma interface para interação tridimensional adequada às necessidades da plataforma T-CADE em termos de desempenho, interatividade e usabilidade, esta poderá agora continuar a ser desenvolvida, uma vez que as inovações pretendidas para a plataforma dependiam de uma interface robusta, ágil e confiável.

Uma destas inovações é a criação de superfícies de concordância entre duas superfícies, ou o arredondamento das arestas comuns a duas superfícies. Embora a tecnologia

necessária à criação desta ferramenta já estivesse ao alcance da plataforma T-CADE, esta inovação dependia do reconhecimento do ponto 3D sobre a superfície selecionada no momento de sua seleção. A nova interface permite este tipo de interação.

Outra possível inovação é a criação de ferramentas de edição e criação de objetos, sendo a nova interface baseada em uma biblioteca gráfica 3D, ela traz algoritmos robustos para resolver problemas geométricos de transformações tridimensionais que, de outra forma, seriam de difícil implementação. Assim, torna-se possível a criação de ferramentas como rotação, escala e espelhamento, essenciais na interação tridimensional e de fácil implementação, considerando a estrutura de dados da nova interface.

O projeto Virtus propõe o desenvolvimento de um sistema computacional para o design virtual de produtos. Este será um sistema modular que deverá contemplar, quando concluído, as principais etapas do projeto de produto em um ambiente virtual. O T-CADE complementa o módulo de projeto conceitual, parte deste sistema computacional, como um sistema CAD para a geração de conceitos no desenvolvimento de produtos a partir das suas ferramentas de modelagem e da capacidade de geração de malhas otimizadas para a análise estrutural por elementos finitos.

Em função do domínio de seu código fonte, é possível fazer uma integração de sua base de dados dos objetos geométricos com uma base de dados *On-line* (um dos módulos do projeto Virtus), tornando os objetos reutilizáveis em novos projetos.

Com a nova interface, a plataforma T-CADE pode, ainda, ser adaptada para uso como ferramenta de ensino em cursos de graduação como Design, Arquitetura e/ou Engenharias, substituindo com vantagens as atuais aplicações que utilizam *plugins* de realidade virtual (VRML) como meio de representação tridimensional.

Todas as intervenções feitas na nova interface visaram à melhoria do desempenho gráfico e da interatividade, levando em conta, principalmente, os critérios básicos de usabilidade apontados por Nielsen (1993) e os critérios para avaliação heurística da usabilidade recomendados por Nielsen e Molich (1990).

Sendo assim, a criação de uma interface gráfica para interação tridimensional na plataforma T-CADE melhorou o desempenho gráfico, aumentou a interatividade e, consequentemente, melhorou a usabilidade do software. A partir destes avanços, a interface da plataforma T-CADE passou a facilitar e criar novas oportunidades de desenvolvimento, dando condições à criação de novas ferramentas e, inclusive, novas aplicações desta plataforma.

Desta forma, a hipótese básica de que: “O uso de uma biblioteca gráfica e a criação de ferramentas gráficas para visualização, seleção e manipulação de objetos melhoram o desempenho gráfico e a interatividade, melhorando, consequentemente, a usabilidade do software”, pode ser considerada verdadeira. A partir das melhorias implementadas na interface, em termos de desempenho e interatividade, pode-se inferir uma melhoria significativa da usabilidade, sendo esta visivelmente perceptível. No entanto, a quantificação

desta melhoria só poderá ser efetivamente estabelecida nos testes de usabilidade, ao término do projeto T-CADE, ou em outra pesquisa com este fim específico.

Além destas conclusões, algumas considerações são importantes a título de registro. O código fonte do programa original, com todas suas funções, continha cerca de vinte e quatro mil (24.000) linhas de código. A implementação da nova interface acrescentou cerca de seis mil e quinhentas (6.500) linhas de código a este programa, correspondendo a mais de 25% do total de código existente.

Como um resultado pessoal, um dos maiores benefícios alcançados com o desenvolvimento deste trabalho, em termos de formação, foi o aprendizado da lógica de programação. Foi um trabalho árduo, mas altamente gratificante.

6.1 TRABALHOS FUTUROS

A partir dos recursos interativos da nova interface, abre-se a possibilidade do desenvolvimento de novas ferramentas e capacidades do T-CADE:

- O desenvolvimento de algoritmos para superfícies de concordância no encontro de arestas.
- Ferramentas para pintura em objetos tridimensionais.
- Aplicação de texturas, tanto diretamente nos objetos quanto nas classes a que eles pertencem.
- Criação de uma ferramenta para gerenciamento da biblioteca de materiais, com possibilidade de criação e edição de materiais.
- Utilização do novo ambiente 3D implementado para a criação de softwares educacionais que necessitam de interação tridimensional.
- Integração da base dados geométrica do T-CADE com uma base de dados *On-line* para facilitar o processo de desenvolvimento virtual de produtos.
- Criar uma estrutura de dados sólido-construtiva.
- Avaliar a usabilidade da nova interface a partir das melhorias identificadas.

REFERÊNCIAS BIBLIOGRÁFICAS

ABADI, M. e CARDELLI, L. ***A Theory of Objects***. Nova York: Springer-Verlag, 1996.

ABNT- **NBR 9241-11** - Requisitos Ergonômicos para Trabalho de Escritórios com Computadores: Parte 11 – Orientações sobre Usabilidade. Rio de janeiro, 2002.

ADOBE Photoshop CS4, Site Ofical. Disponível em : <http://www.adobe.com/br/products/photoshop/photoshop/>. Acesso em: 19 novembro 2008.

ANSI- ***Computer Graphics-Graphical Kernel System (GKS) Function Description.*** *Technical report. American National Standards Institute for Information Processing Systems*, 1985a.

ANSI- ***Programmer's Hierarchical Interactive Graphics System (PHIGS).*** *Technical report. American National Standards Institute for Information Processing Systems*, 1985b.

ARNHEITER, E. D. e HARREN, H. *Quality management in a modular world*. ***The TQM Magazine***. Hartford, Vol. 18 No. 1, 2006.

ASIMOV, M. ***Introduction to design***. Nova Jersey: Prentice Hall, 1962.

BACK, N.; OGLIARI, A.; DIAS, A. e SILVA, J. C. **Projeto Integrado de Produtos**. Barueri-SP: Manole, 2008.

BARTON, J.A.; LOVE, D.M. e TAYLOR, G.D. *Design determines 70% of cost? A review of implications for design evaluation.* ***Journal of Engineering Design***, 12:1, pp 47-58, 2001.

BASTIEN, J.M.C. e D. SCAPIN. ***Ergonomic Criteria for the Evaluation of Human-Computer interfaces***. France: Institut national de recherche en informatique et en automatique, 1993.

BAUER, F.L. ***Software Engineering - Information Processing***. Amsterdan: North Holland, 1972.

BAXTER, M. **Projeto de Produto: Guia prático para o desenvolvimento de novos Produtos.** São Paulo: Edgar Blucher, 1998.

BONSIEPE, G.; KELNNER, P. e POESSNECKER, H. **Metodologia Experimental: Desenho Industrial.** CNPq, 1984.

BOOCH, G.; MAKSIMCHUK, R. A.; ENGLE, M. W.; YOUNG, B. J.; CONALLEN, J. e HOUSTON, K. A. ***Object-Oriented Analysis and Design with Applications***. London: Addison-Wesley, 2007.

BURDEK, B.E. **História Teoria e Prática do Design de Produtos**. São Paulo: Edgar Blucher, 2006.

CANTÚ, M.. **Dominando o Delphi 5: A Bíblia**. São paulo: Makron Books, 2000

CARROLL, J. M. ***Scenario-based design: Envisioning work and technology in system development.*** Nova York: John Wiley & Sons, 1995.

CELES, W., e CORSON-RIKERT, J. Act: An easy-to-use and dynamically extensible 3D graphics. **Proc. of SIBGRAPI '97**, 1997.

CORELDRAW Graphics Suite X4, Site Oficial. Disponível em: <http://www.corel.com.br/> Acesso em: 19 Novembro 2008.

EMMERIK, M. J.G.M. van. *A Direct Manipulation Technique for Specifying 3D Object Transformations with a 2D Input Device*. **Computer Graphics Forum 9**. North-Holland, 1990.

FOLEY, J.D.; VAN DAM, J.; FEINER, S.L. e HUGHES, J.F.. ***Computer Graphics:*** *Principles and Practices*. 2º Ed. London: Addison-Wesley, 1992.

GIMP, Site Oficial. Disponível em: <http://www.gimp.org/>. Acesso em: 19 Novembro 2008.

GLSCENE, Site Oficial. Disponível em: <http://www.glscene.org/>. Acesso em: 24 novembro 2007.

GUAZZI, D. M. **Utilização do QFD como ferramenta de melhoria contínua do grau de satisfação de clientes internos: uma aplicação em cooperativas agropecuárias.** Florianópolis, 1999. (Doutorado – Engenharia de Produção da Universidade Federal de Santa Catarina).

HAAG, S.; RAJA, M. K. e SCHKADE, L. L. *Quality Function Deployment Usage in Software Development*. ***Communications of the ACM***, Vol. 39, No. 1, pgs. 41-49, 1996.

HARTSON, H. R. e HIX, D. ***Developing User Interfaces: ensuring usability through product and process.*** Nova York: John Willey & Sons, 1993.

HEWETT, T. *Curricula for Human-Computer Interaction*. **ACM SIGCHI Report.** 1992. Disponível em: <http://sigchi.org/cdg/>. Acesso em: 25 Agosto de 2008.

IEA - International ergonomics association, Site Oficial. Disponível em: <http://www.iea.cc/browse.php?contID=what_is_ergonomics>. Acesso em: 25 Novembro 2007.

KETTNER, L. *A classification scheme of 3D interaction techniques.* ***Technical report*** *B 95-05*, Berlin: Institute for Computer Science, Department of Mathematics and Computer Science, Freie Universität Berlin, 1995.

KETTNER, L. *Theorical Foundations of 3D-Metaphors.* ***Computer/Human Interaction - CHI'94***. Nova York: ACM, 1994.

KHAN, A; FITZMAURICE, G.; MATEJKA, J.; MORDATCH, I. e KURTENBACH, G. *ViewCube: A 3D Orientation Indicator and Controller*. **SIGGRAPH**. New York: ACM, 2008. p.17-25.

KOTONYA, G. e SOMMERVILLE, I. ***Requirements Engineering: Processes and Techniques***. Nova York: John Willey & Sons. 1998.

KILGARD, M. J. *OpenGL and X, Part 1: An Introduction*. **Technical report, SGI**, 1994.

MARTZ, P. ***OpenGL® Distilled***. Harlow: Addison-Wesley Professional, 2006.

MAYHEW, D.J. ***The Usability Engineering Lifecycle, a practitioner's handbook for User Interface Design.*** São Francisco: Morgan Kaufmann Publishers, 1999.

MCGRAW-HILL encyclopedia of science and technology. **Human-computer interaction**. The McGraw-Hill Companies, Inc., 2005.

MINGHIM, R. e OLIVEIRA, M.C.F. Uma Introdução à Visualização Computacional. **JAI'97 - Jornadas de Atualização em Informática**, XVII Congresso da SBC. Brasília, 1997. pp.85-131.

MIZUNO, S. e AKAO, Y. *Quality Function Deployment*. JUSE. USA, 1994.

MYERS, B.A. e ROSSON, M.B. Survey on user interface programming. **Proc. ACM CHI'92 Con**, 3-7 de May de 1992: p195-202.

NASERI, E. e YEKTAEE, S. **Car Design**. Recursos de Aula – Sharif University of Technology. Disponível em: <http://ie.sharif.edu/en/courses/courses/spring2006/ergonomy/index.php? section=resources&file=resources>. Acesso em: 20 abril 2009.

NIELSEN, J. e TAHIR, M. **Homepage: Usabilidade – 50 Websites desconstruídos.** Rio de Janeiro: Editora Campus, 2002.

NIELSEN, J. e MOLICH, R. *Heuristic evaluation of user interfaces.* **Proc. CHI'90** Conference on Human Factors in Computer Systems., 1990: p249- 256.

NIELSEN, J. *How to conduct a heuristic evaluation*. Disponível em: <http://www.useit.com/papers/heuristic/heuristic_evaluation.html>. Acesso em: 10 novembro 2008.

NIELSEN, J. ***Usability engineering***. São Francisco: Morgan Kaufmann, 1993.

NIELSEN, J. Usability 101: introduction to usability. Disponível em: <http://www.useit.com/alertbox/20030825.html>. Acesso em: 25 Novembro 2007.

O'DOCHERTY, M. ***Object-Oriented Analysis and Design: Understanding System Development with UML 2.0.*** São Francisco: John Wiley & Sons, 2005.

OHFUJI, T. **Verdadeiro Significado do QFD**. Palestra proferida no I Encontro Internacional de QFD, Rio de Janeiro, RJ, 1997.

OPENGL, Site Oficial. Disponível em: <http://www.opengl.org>. Acesso em: 25 Novembro 2007.

PAHL, G.; BEITZ, W.; FELDHUSEN, J. e GROTE, K.H. **Projeto na Engenharia - Fundamentos do Desenvolvimento Eficaz de Produtos - Métodos e Aplicações**. São Paulo: Edgard Blucher, 2005.

PERSIANO, R. M. e OLIVEIRA, A. A. F. **Introdução à Computação Gráfica Usando GKS**. 1.ed. Rio de Janeiro: LTC- Livro Técnico e Científico, 1989. v.1. 223 p.

PFLEEGER, S. L. **Engenharia de Software: Teoria e Prática**. 2º Ed. São paulo: Prentice Hall, 2004.

PREECE, J.; ROGERS, Y. e SHARP. H. ***Interaction Design: beyond human-computer interaction***. Nova York: John Wiley & Sons, 2002.

PREECE, J.; ROGERS, Y.; SHARP, H.; BENYON, D.; HOLLAND, S. e CAREY, T. ***Human-Computer Interaction***. London: Addison-Wesley, 1994.

PRESSMAN, R.S. **Engenharia de Software**. São Paulo: Makron Books, 1995.

PUGH, S. ***Total Design: Integrate Methods for Successful Product Engineering***. London: Addison-Wesley, 1990.

ROGERS, D.F. e ADAMS. J.A. ***Mathematical Elements for Computer Graphics.*** New York: McGraw-Hill Book Company, 1976.

ROZENFELD, H; FORCELLINI, F. A.; TOLEDO, J. C.; et al. **Gestão de desenvolvimento de produto: uma referência para a melhoria do processo**. São Paulo: Saraiva, 2006.

SCHROEDER, W.J.; MARTIN, K. e LORENSEN. W. ***The Visualization Toolkit - An Object-Oriented Approach to 3D Graphics***- 2ª edição. Prentice-Hall, 1997.

SEGAL, M., e AKELEY. K. *The Design of the OpenGL Graphics Interface*. ***Technical report,*** Mountain View: Silicon Graphics Inc, 1996.

SEGAL, M., e AKELEY. K. *The OpenGL Graphics Interface.* ***Technical report***, Mountain View: Silicon Graphics Inc., 1994.

SHNEIDERMAN, B. *Direct manipulation: a step beyond programming languages*. **IEEE Computer 16(8).** San Francisco: Morgan Kaufmann Publishers Inc.,1983: p57-69.

TEIXEIRA, F.G. **Modelamento paramétrico e geração de malha em superfícies para aplicações em engenharia.** Porto Alegre: UFRGS, 2004. (Doutorado, Engenharia mecânica da Universidade Federal do Rio Grande Sul).

TEIXEIRA, F.G.; SILVA, T.L.K.; SILVA, R.P. e AYMONE. J.L.F. *Virtual Design: Concepts.* ***SAE Technical Paper Series*** 2008-36-0332., Warrendale: Society of Automotive Engineers, Inc., 2008a.
TEIXEIRA, F.G.; SILVA, T.L.K.; SILVA, R.P. e AYMONE. J.L.F. *Virtual Design: Technologies.* **SAE Technical Paper Series** 2008-36-0341., Warrendale: Society of Automotive Engineers, Inc., 2008b.

TEIXEIRA, F.G. e CREUS, G.J. Geração automática de malha sobre recortes de superfícies paramétricas com grande curvatura. **Mecânica Computacional**, v. XXII, p. 2305-2319, 2003.

TEIXEIRA, F.G. e CREUS, G.J. *A robust algorithm to determine surface/surface intersection in both parametric spaces*. **Mecânica Computacional**, v. XXVII, p. 3093-3115, 2008.

ULRICH, K. e EPPINGER, S. ***Product Design and Development***. New York: McGraw-Hill, 2007.

VELHO, L. e GOMES. J. M. ***Image Processing for Computer Graphics***. New York: Spriner Verlag, 1997.

CVS. Disponível em: <http://pt.wikipedia.org/wiki/CVS>. Acesso em: 02 novembro 2008.

WRIGHT, R.S.; LIPCHAK, B. e HAEMEL. N. ***OpenGL SuperBible: Compreensive Tutorial and Reference*** – 4th ed. Upper Saddle River: Addison-Wesley, 2007.

WU, S., e Malheiros. M.G. Interactive 3D Geometric Modelers with 2D UI. **Citeseer**. 2002. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.9. 104>. Acesso em: 25 outubro 2008.

ZELEZNIK, R. e FORSBERG. A. *UniCam—2D gestural camera controls for 3D environments.* **SIGGRAPH.** New York: ACM, 1999. p169-173.