Interact: um modelo de interação para interfaces 2D por manipulação direta

MARCELO MEDEIROS CARNEIRO[1,2]
MARCELO GATTASS[1]
CARLOS HENRIQUE LEVY[1]
ENIO EMANUEL RAMOS RUSSO[1,3]

[1] TeCGraf - Grupo de Tecnologia em Computação Gráfica, Depto. de Informática, PUC-Rio
Rua Marquês de São Vicente 255, 22453-900 Rio de Janeiro, RJ, Brasil
{mmc,gattass,levy,enio}@tecgraf.puc-rio.br

[2] IPRJ/UERJ - Instituto Politécnico, Campus Regional da UERJ
Rua Alberto Rangel s/nº , 28601-970 Nova Friburgo, RJ, Brasil
mmc@iprj.uerj.br

[3] PETROBRAS - Petróleo Brasileiro S.A.
CENPES - Centro de Pesquisas e Desenvolvimento Leopoldo A. Miguez de Mello
Cidade Universitária, Quadra 7, Sala 9020, 21949-900 Rio de Janeiro, RJ, Brasil
enio@cenpes.petrobras.com.br

Abstract. We discuss interaction techniques in systems based on direct manipulation of the objects in the canvases and propose a model to manager such interactions. This model reduces the programming effort by providing an appropriate abstraction of the interaction tasks. A class library that implements these abstractions is also presented. Two application programs developed with this class library allow us to draw some conclusions.

Keywords: Computer Graphics, User Interface, Interaction Tasks, Direct Manipulation.

Introdução

Utilizando os recursos dos atuais sistemas de interface, o desenvolvimento de aplicações baseadas em manipulação direta é ainda uma tarefa complexa e sujeita a erros. Nesta classe de problemas, estão os editores gráficos, programas que permitem a criação e manipulação de objetos gráficos que possuem representação visual e comportamentos específicos.

A complexidade na implementação de tais programas deve-se principalmente ao fato de que eles requerem a utilização de ferramentas mais adequadas. *Toolkits* tradicionais de construção de interfaces (por exemplo, Motif [OSF, 1991]) oferecem apenas *widgets*, tais como botões, menus, janelas etc., mas não oferecem os objetos gráficos necessários para compor os desenhos. Por outro lado, os sistemas gráficos (por exemplo, X Window System [Nye, 1990]) oferecem apenas primitivas para o desenho dos objetos, mas usualmente não oferecem suporte abstrato para sua manipulação.

Para exemplificar, a Figura 1 mostra uma situação em que o usuário deseja construir um objeto definido por um conjunto de vértices e representado visualmente por uma simples *polyline*. Para isto, o usuário utiliza dispositivos de entrada do *hardware* (usualmente *mouse* e teclado), que geram diversos eventos. Estes eventos são transmitidos para o componente de *Input* (I) do sistema (usualmente o *toolkit* de interface). Efetivamente, os eventos são repassados para o componente *Computation* (C) (usualmente a própria aplicação), que fica responsável pelo seu tratamento. Finalmente, C envia ordens para que o componente *Display* (D) faça a atualização da tela, à medida em que os eventos são reconhecidos e tratados pelo componente C.

Neste exemplo, C é responsável por boa parte da implementação da interação, visto que trata todas as ações do usuário, inclusive as transientes, que não fazem parte do resultado final desejado. Ou seja, apesar de estar basicamente interessada apenas na lista de coordenadas da *polyline*, a aplicação deve tratar a situação em que o usuário marca um ponto errado, volta atrás e corrige o erro. Este nível de detalhe torna a programação de aplicações muito pouco produtiva. Isto ocorre principalmente porque C recebe de I eventos em sua forma elementar, isto é, sem nenhum tipo de tratamento.

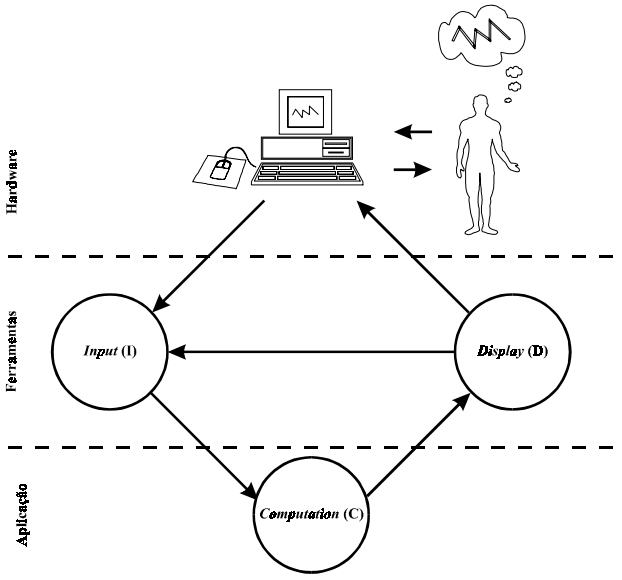

Figura 1: Criação de uma *polyline* com ferramentas tradicionais.

Em muitos programas de aplicação, as tarefas de interação são semelhantes e poderiam ser melhor suportadas se houvesse uma preocupação com a abstração dos mecanismos de interação no *canvas*. Em particular, nas aplicações que manipulam modelos geométricos bidimensionias do tipo de sistemas CAD e de grafos geométricos, a definição dos objetos faz-se por meio de vértices de controle. A maior dificuldade de programar a interface com o usuário nesta classe de aplicações está em permitir a manipulação direta sobre o *canvas* para criar e/ou editar estes objetos. Estas tarefas podem ser realizadas por meio da criação e do arrasto de vértices de controle e alças de *bounding boxes*.

Este trabalho propõe um modelo, denominado Interact, que define um conjunto genérico de tarefas de interação que pretende atender adequadamente a classe de aplicações baseadas em modelos geométricos bidimensionais. Nesta proposta, a arquitetura ilustrada na Figura 1 é substituída por outra, mostrada na Figura 2.

Nesta nova arquitetura, I agora possui um certo poder de computação (C_I), o que lhe permite tratar os eventos que recebe e enviar para C apenas informações mais essenciais como, por exemplo, a lista de coordenadas da *polyline*, ao final da interação. Além disso, com algum poder de *display* (D_I), I é capaz de apresentar seus próprios *feedbacks* e *prompts* sem nenhuma (ou quase nenhuma) interferência de C. Pelo mesmo motivo, D também possui um pouco de poder de computação (C_D).

Esta estratégia de agregar mais poder em I e D permite obter uma abstração maior no tratamento da interação, como será visto posteriormente. No entanto, não é trivial estabelecer um limite nesta relação. A situação extrema, onde muito poder de *display* e *computation* é incorporado em I, causa um efeito negativo, pois não promove a independência entre os componentes, ou seja, a separação necessária entre os componentes não fica clara.

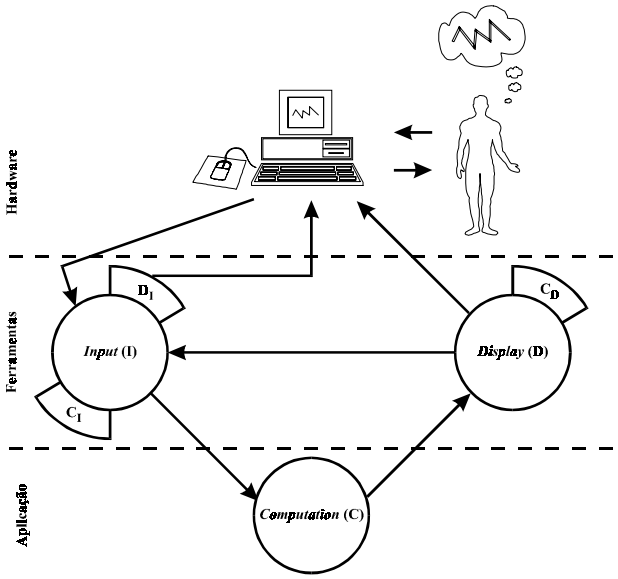

Figura 2: Criação de uma *polyline* no modelo do Interact.

Trabalhos anteriores

Um estudo mais detalhado sobre os aspectos discutidos na seção anterior é encontrado em [Hartson, 1989]. A simbologia utilizada nas Figuras 1 e 2 segue a mesma convenção adotada neste trabalho, para facilitar as comparações e realçar que os nossos objetivos são comuns.

Hartson faz um resumo das estratégias de *design* de um sistema interativo, especialmente no caso de sistemas baseados em eventos, por serem mais flexíveis e completos.

Outro ponto importante levantado é o problema da comunicação entre os componentes. Como a separação entre componentes é fundamental para promover o reuso de *software*, deve existir um mecanismo de comunicação de forma a garantir sua independência.

Um exemplo típico ocorre quando o usuário seleciona um determinado objeto da aplicação. Normalmente, este objeto é apresentado de forma diferente dos demais, para deixar bem clara a seleção. Mas, em que componente isto será tratado? Em certos casos mais simples, o próprio *Input* pode fazer isto. Porém, no caso de manipulações mais complexas, como, por exemplo, *dragging*, não é

desejável que I tenha tanto poder de computação, pois isto pode violar a separação dos componentes.

Em [Szekely e outros, 1993], encontra-se uma discussão mais aprofundada sobre aspectos de *design* de interfaces. Neste trabalho, os autores sugerem que uma ferramenta de *design* de interfaces não deve apenas dar suporte à construção de interfaces (*interface builders*), tais como menus e caixas de diálogo, mas também deve permitir a construção dos aspectos específicos da aplicação, como, por exemplo, a criação, manipulação e visualização dos objetos em um editor gráfico.

O primeiro passo no *design* de uma interface é exatamente modelar todas as funcionalidades da aplicação, isto é, os objetos e os comandos oferecidos. Este ponto é bastante polêmico, pois introduz na ferramenta de *design* um poder de expressão muito grande, às vezes até desnecessário.

Os autores descrevem este sistema (HUMANOID) e apresentam os aspectos mais importantes de uma interface, que são: *presentation*, *behaviour* e *dialogue*. Estes aspectos são modelados utilizando uma linguagem baseada em *templates*.

Em [Vlissides e Linton, 1990], os autores apresentam o sistema Unidraw. Esta ferramenta oferece um *framework* para a construção de editores gráficos em domínios específicos, tais como desenhadores comuns e editores de circuitos eletrônicos.

Esta ferramenta simplifica a construção de tais editores, pois procura explorar e abstrair os aspectos comuns que existem em tais programas. Para isto, os autores definem quatro abstrações: *components* (aparência e semântica dos objetos), *tools* (manipulação direta dos objetos), *commands* (operações entre *components*) e *external representations* (por exemplo, uma saída em *PostScript*). Novamente, o comportamento dos objetos da aplicação fica inserido dentro do próprio modelo.

Por último, sistemas do tipo OpenInventor [Wernecke, 1994] promovem a abstração de tarefas de interação em 3D, através de um mecanismo de composição, também baseado em um ambiente dirigido por eventos.

Primeiro, é definido um conjunto básico de mecanismos de interação (*Draggers*) que, quando agregados, permitem formas mais sofisticadas de interação (*Manipulators*). Os mecanismos de interação são, na verdade, objetos "espertos" capazes de tratar eventos.

Todos os objetos (chamados *nodes*) de uma cena no Inventor são armazenados em uma única estrutura (um grafo dirigido acíclico) contendo, inclusive, os objetos da aplicação.

Neste sistema, procura-se fornecer ao programador uma ampla variedade de objetos, deste primitivas 3D (cubos, esferas, *nurbs* etc.) até câmeras, luzes e materiais diversos (a maiorida destes aspectos são baseados no OpenGL [Neider e Woo, 1993]). Com isto, o Inventor fornece, quase que por exaustão, um elevado grau de automação das tarefas de interação, além de um conjunto mais amplo de recursos (por exemplo, sensores de cena e suporte a animações). Entretanto, muitos destes recursos existem porque toda a estrutura de dados está "dentro" do Inventor, e não na própria aplicação. Recentemente, a Silicon Graphics (responsável pela concepção do Inventor) está investindo em uma nova API, chamada OpenGL Optimizer. Esta ferramenta procura compatibilizar os requisitos de desempenho, fundamentais para as aplicações de CAD/CAM/CAE.

O Modelo de Interação Proposto

Os trabalhos estudados até o momento procuram, em geral, criar uma representação visual ativa dos objetos (a exemplo do que ocorreu com botões e menus), colocando em C_I e D_I da Figura 2 estruturas de dados que implementam tais representações. Na opinião dos autores, as componentes de interação (I) e *display* (D) não devem definir estas estruturas para os objetos da aplicação. Toda esta semântica deve estar embutida na própria aplicação e comunicação deve ser realizada em um nível mais abstrato, através da definição e manipulação de vértices de controle e *bounding boxes*.

A idéia central é que um programa de aplicação possa especificar uma tarefa a ser executada e só se preocupar com o seu resultado. Todos os eventos que ocorrem durante a interação ficam a cargo dos componentes I, C_I e D_I do *toolkit* proposto. Uma tarefa de interação, ao ser associada a um *canvas*, torna-se ativa e passa a receber todos os eventos de *mouse* e teclado.

Para dar maior flexibilidade, o modelo pressupõe que a aplicação possa suspender temporariamente a execução de uma tarefa, associando outra ao *canvas* e retornando à anterior assim que a última terminar. Assim, por exemplo, a tarefa de se fornecer uma *polyline* pode ser interrompida enquanto o usuário navega pelo desenho, tornando visível a posição do novo ponto a ser inserido na *polyline*. Assim, em um dado momento, cada *canvas* só pode estar executando uma tarefa de interação por vez. Isto faz sentido, pois usualmente só dispomos de um *mouse* e um teclado. Nada, entretanto, impede de se ter diversas tarefas suspensas em um *canvas*, prontas para ganharem execução do ponto onde foram suspensas. Isto ocorre porque cada tarefa tem um estado interno próprio.

A formalização do modelo utiliza os conceitos de programação orientada para objetos. O Interact define duas classes principais (Figura 3):

- `Canvas`. É a abstração do conceito de área de desenho, onde a aplicação exibe seus objetos. O *Canvas* é responsável pela captura e tratamento de todos os eventos gerados pelo sistema de interface, tais como movimento do *cursor*, pressionamento de teclas (*mouse* e teclado) etc;
- `Task`. É a abstração de tarefas de interação, tais como captura de uma linha, posicionamento, seleção e transformação de objetos, controle do nível de *zoom* etc.

Uma idéia importante é que as tarefas mais usuais de interação já devem estar definidas no modelo, porém o programador pode criar as suas próprias tarefas específicas, normalmente através de herança. A Figura 3 ilustra a relação entre os objetos `Canvas` e `Task` (as setas indicam somente o fluxo da interação).

O modelo de interação aqui proposto visa atender principalmente aplicações onde o usuário realiza as seguintes tarefas:

- construção: criação de novos objetos definidos a partir de vértices de controle (linhas, quadrados, círculos etc.);
- seleção e transformação: escolha e alteração de atributos dos objetos criados (cor, tamanho, posição, forma etc.);
- visualização: controle da área visível do desenho (*zoom*, *pan* etc.).

Estas tarefas são as mais freqüentes para a classe de aplicações que este trabalho enfatiza.

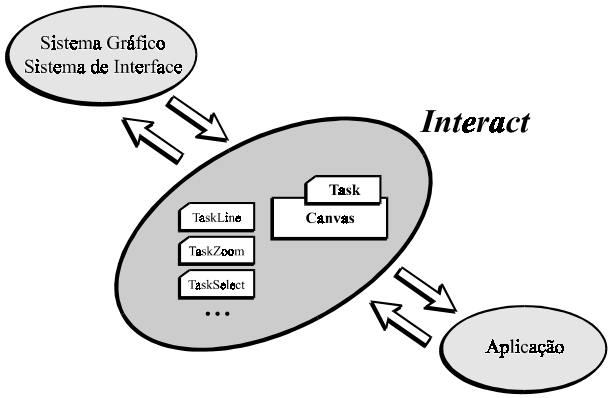

Figura 3: Interact Canvas e Task.

Tarefas de Construção

As tarefas de construção no modelo do Interact são divididas em três categorias, baseadas na quantidade de pontos necessários para definir o objeto:

- construção de um ponto;
- construção de dois pontos;
- construção de N pontos.

A idéia é que as tarefas de construção permitam que o usuário instancie diversos tipos objetos na aplicação. Normalmente, os objetos são mantidos em alguma estrutura de dados da aplicação, para que possam ser manipulados posteriormente. No entanto, o Interact desconhece quaisquer estruturas de dados da aplicação. O único papel das tarefas de construção é definir a geometria dos objetos que estão sendo instanciados. Por exemplo, quando a aplicação precisa desenhar uma linha, a tarefa de linha cuida de toda a interação, porém a aplicação só está interessada no ponto inicial e final da linha. Esta é a única informação que a tarefa passa para a aplicação. Todas as tarefas de construção possuem a *callback* `EndObject`, que é chamada assim que a geometria do objeto estiver completamente definida.

Uma outra característica importante é que as tarefas de construção não desenham a forma definitiva do objeto. Por exemplo, após a construção de um retângulo, nada é desenhado pela tarefa. A aplicação é que deve desenhá-lo utilizando os atributos (cor, espessura de linha e preenchimento) adequados. A tarefa desenha apenas retângulos temporários (*feedback*) durante a construção do objeto, isto é, no decorrer da interação. Após sua conclusão, este retângulo temporário é removido e a aplicação recebe suas coordenadas. A partir deste momento, a aplicação fica responsável por decidir o que fazer com o objeto.

Construção de um ponto

As construções de um ponto são utilizadas para instanciar objetos que possam ser definidos através de um único par de coordenadas (*x*,*y*).

Para construir um ponto no modelo do Interact, o usuário deve posicionar o *cursor* no local desejado do *canvas* e pressionar o botão esquerdo do *mouse*, finalizando a tarefa. Enquanto o *cursor* é movido, a tarefa desenha um *feedback* simples (um pequeno retângulo de tamanho ajustável). Este *feedback* permite que o usuário visualize o tamanho real do objeto que está sendo instanciado. Ao finalizar a tarefa (através do botão esquerdo do *mouse*), a *callback* `EndObject` é automaticamente chamada (Figura 4).

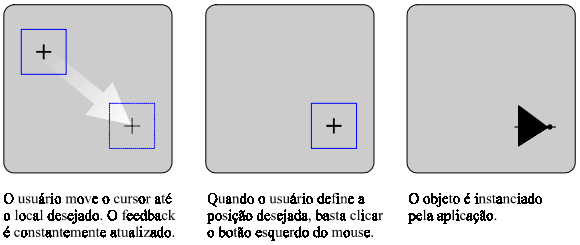

Figura 4: Construção de um ponto.

Construção de dois pontos

As formais mais usuais de construção de dois pontos são os segmentos de reta (linhas) e os retângulos. A diferença entre essas formas está unicamente no *feedback*.

As construções de dois pontos no modelo do Interact permitem que sejam aplicadas restrições de movimento (pressionando a tecla CONTROL enquanto o *cursor* é arrastado). Com isto, o usuário pode desenhar linhas horizontais ou verticais. A tarefa é do tipo *click-and-drag*:

- pressionando o botão esquerdo do *mouse*, a construção é iniciada (o primeiro ponto é definido na posição corrente do *cursor*);
- o usuário arrasta o *cursor* até o segundo ponto (o *feedback* é desenhado constantemente);
- soltando o botão do *mouse*, a construção é terminada (o segundo ponto é definido na posição corrente do *cursor* e a *callback* `EndObject` é chamada).

Construção de N pontos

É utilizada basicamente para construir *polylines*. A tarefa é do tipo *click-and-move*:

- pressionando o botão esquerdo do *mouse*, um vértice é definido na posição corrente do *cursor*;
- movendo o *cursor* (sem pressionar os botões do *mouse*), um segmento é ajustado (o *feedback* é constantemente desenhado);
- pressionando o botão da direita do *mouse*, a construção é terminada (a *callback* `EndObject` é chamada).

Além das facilidades para definir os diversos vértices da *polyline*, o modelo também permite que o usuário volte atrás removendo os vértices (pressionando a tecla DEL), cancele a construção (pressionando a tecla ESC) e também crie segmentos horizontais ou verticais (pressionando a tecla CONTROL). Após o término da interação, a *callback* `EndObject` é automaticamente chamada.

Tarefas de Seleção e Transformação

Para realizar qualquer operação nos objetos criados na aplicação, é bastante comum os programas gráficos interativos pós-fixarem suas operações: primeiro, o usuário deve escolher, isto é, selecionar um ou vários objetos da aplicação e depois aplicar a operação.

Quando o usuário deseja escolher um objeto individualmente, pode utilizar a técnica de *pick*: basta posicionar o *cursor* sobre o objeto e *clicar* o botão do *mouse*. Em uma outra situação, se o usuário deseja selecionar um conjunto de objetos, pode utilizar a técnica de *fence*: basta definir uma região (normalmente retangular) no *canvas* da aplicação e todos os objetos que estão dentro são automaticamente selecionados.

Um ponto importante é a ordem de seleção. Muitas vezes alguma operação necessita que os objetos sejam marcados em uma determinada ordem. Um típico exemplo é a operação de alinhamento: algum objeto servirá como referência e os outros serão alinhados em relação à esse objeto. Quando se utiliza o *pick*, a ordem de seleção fica clara. No entanto, quando se realiza um *fence*, isto não acontece.

Um outro aspecto importante é o *feedback* de seleção. Vários tipos podem ser utilizados (Figura 5).

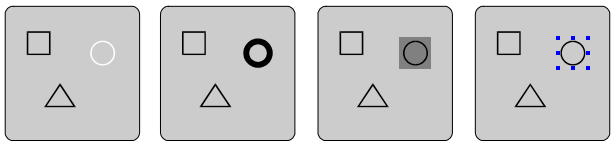

Figura 5: Tipos de *feedback* de seleção.

Várias das formas mostradas têm graves problemas. Nem sempre existe uma cor disponível ou estilo de linha para mostrar quais os objetos estão selecionados. O *feedback* deve sempre utilizar um padrão único, senão o reconhecimento não será eficiente.

Outro ponto importante ocorre quando vários objetos estão selecionados simultaneamente. Se o *feedback* for colocado em cada objeto, ganha-se na facilidade de identificar quem está selecionado. No entanto, isto não acontece quando a quantidade de objetos torna-se grande. Por outro lado, se o *feedback* for no grupo, a tela da aplicação fica bem mais limpa, porém pode ser difícil perceber quem está selecionado (Figura 6).

No modelo do Interact, a seleção e a transformação de objetos são implementadas por duas tarefas. A primeira é responsável pela seleção propriamente dita e pelas transformações lineares afins (translação, escala, rotação e cisalhamento). A segunda faz transformações locais

(*reshape*) de manipulação de vértices (ou pontos de controle) em *polylines*.

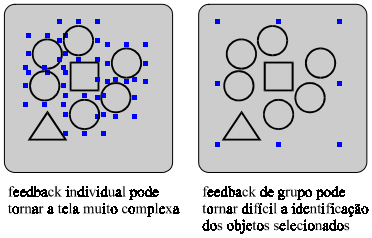

Figura 6: *Feedback* individual ou de grupo.

Seleção e transformações lineares

Todas as transformações realizadas por esta tarefa (translação, escala, rotação e cisalhamento) utilizam manipulação direta. As translações são realizadas através de *dragging* e podem ser restritas apenas na vertical ou horizontal. Para isto, deve-se manter a tecla CONTROL pressionada durante o *dragging*.

Para realizar as outras transformações (escala, rotação e cisalhamento), é utilizada a metáfora das alças (*handles*), que são pequenos retângulos colocados ao redor dos objetos selecionados. No entanto, não é possível realizar as três transformações simultaneamente. Para isto, a tarefa de seleção pode estar em dois modos: um que realiza escala e um outro que faz rotação e cisalhamento. A diferença entre estes dois modos está no *feedback* (Figura 7). Para alternar de modo, basta *clicar* em qualquer objeto selecionado. A translação pode ser realizada em qualquer modo, pois não utiliza as alças.

O mecanismo de *feedback* também permite diferenciar o primeiro objeto selecionado: ele fica envolvido por um retângulo contínuo, enquanto os demais são tracejados. Isto pode ser importante para algumas operações, onde um objeto é utilizado como referência (por exemplo, operação de alinhamento).

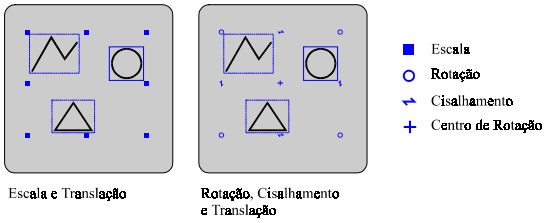

Figura 7: Modos de transformação.

As escalas são realizadas arrastando-se os *handles*. Podem ser feitas na horizontal, vertical ou em ambas as direções. No último caso, o tamanho do objeto é alterado proporcionalmente, sem distorções. O uso dos *handles* permite que a escala seja realizada sem que o usuário indique qual o centro da transformação: a tarefa de seleção mantém fixo o *handle* oposto. Também é possível realizar a escala em relação ao centro do grupo de objetos. A Figura 8 mostra a situação em que os objetos aumentam de tamanho, mas o centro do grupo permanece fixo. Para realizar a escala centrada, é necessário manter a tecla SHIFT pressionada durante o arrasto de um *handle*.

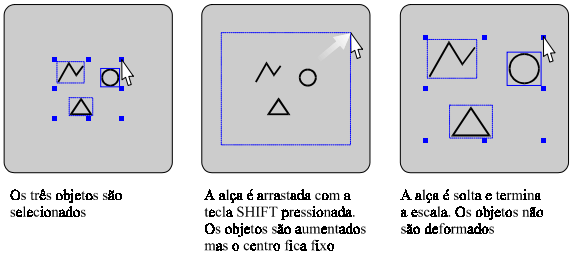

Figura 8: Mudança centrada de escala.

Se o usuário pressionar a tecla CONTROL durante o arrasto de um *handle*, a mudança de escala é realizada em passos de 100%. Além disso, nada impede que o usuário arraste os *handles* para lado oposto (Figura 9). Nesta situação, os objetos são espelhados.

As rotações e cisalhamentos são realizados de forma análoga. Neste modo, aparece mais uma alça (além das oito já existentes), simbolizada por "+", que indica o centro de rotação. Esta nova alça é inicialmente colocada no centro do grupo e, através de *dragging*, pode-se arrastá-la para qualquer outro lugar. As rotações e cisalhamentos também podem ser realizadas em passos discretos (pressionando a tecla CONTROL).

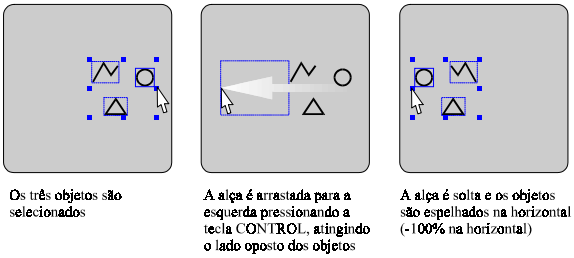

Figura 9: Espelhamento na horizontal.

Para que a aplicação utilize a tarefa de seleção, é preciso apenas que três *callbacks* sejam definidas (Figura 10):

- *Pick*. É chamada sempre que a tarefa de seleção precisa saber se existe algum objeto da aplicação em uma determinada posição do *canvas*. Caso exista, a *callback* deve registrar a *bounding box* do objeto e retornar um identificador para o objeto (usualmente um ponteiro para ele);

- *Fence*. É chamada sempre que a tarefa de seleção precisa saber quais os objetos da aplicação estão dentro de uma determinada região retangular do *canvas*. Todos estes objetos devem ser registrados;
- *Transform*. É chamada sempre que o usuário realiza alguma transformação (translação, rotação etc.). A *callback* recebe a matriz que transforma os pontos da aplicação. A operação deve ser aplicada em todos os objetos selecionados, o que é fornecido pelo Interact. Deve-se não só aplicar a transformação, como também atualizar a *bounding box* dos objetos transformados.

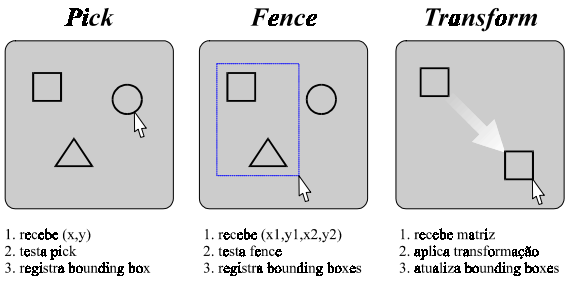

Figura 10: *Callbacks* de responsabilidade da aplicação.

Mudança de forma

A tarefa de mudança de forma (*reshape*) pode ser utilizada em objetos representados por um conjunto de vértices (nós). Pode-se mover nós, inserir novos nós ou remover nós já existentes. Um objeto por vez pode ser remodelado. A seleção é realizada através de *clique* sobre o objeto desejado. Quando ele é selecionado, todos os seus nós aparecem. Os nós podem ser selecionados através de *pick* e *fence* (Figura 11).

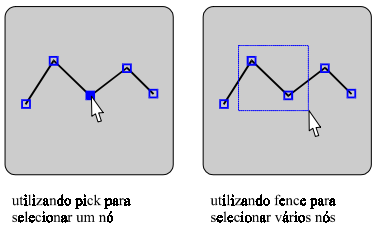

Figura 11: Seleção de nós através de *pick* e *fence*.

Toda a manipulação dos nós é implementada unicamente através de manipulação direta. Por exemplo, para mover os nós, é utilizada a técnica de *dragging* (Figura 12). Todas as operações de manipulação dos nós estão rigorosamente formalizadas no modelo, sendo que sua utilização simplifica bastante a programação da interação. Para maiores detalhes, consulte [Carneiro, 1995].

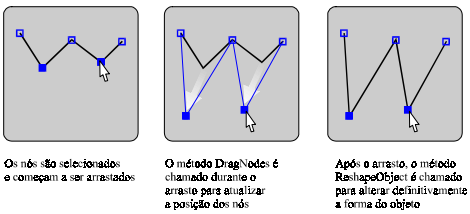

Figura 12: Movendo nós.

Tarefas de Visualização

As técnicas de visualização permitem ao usuário "navegar" por toda área de desenho. Em geral, a região de desenho é bem maior que a área do *canvas*, onde as primitivas são exibidas. Por isto, utiliza-se bastante o conceito de espaço virtual. Primeiro, é definido o conceito de *world*, que é todo o espaço destinado para o desenho. A parte visível do *world* é chamada de *window*, que é um subespaço do *world*. Todas as primitivas dentro da *window* são mapeadas para o *canvas* da aplicação.

A função das tarefas de visualização é permitir o controle da *window*. O modelo define dois mecanismos distintos de interação: *zoom* e *pan*. Após o término de uma interação de visualização, a aplicação deve ser redesenhada. A utilização do *scroll bar* também é uma forma de visualização; no entanto, ele é controlado diretamente pelo Canvas.

A tarefa de *zoom* permite realizar *zoom in* e *zoom out*. O *zoom* pode ser feito em relação ao centro do *canvas* ou então em algum outro ponto qualquer. Pode-se também realizar *zoom* de área (*zoom in*). Já a tarefa de *pan* permite mover a *window* nas duas direções simultaneamente. Este movimento está restrito dentro dos limites do *world*.

As tarefas de visualização são ortogonais em relação às demais tarefas. Enquanto a construção, seleção e transformação interagem com os objetos da aplicação, a visualização interage diretamente com o espaço de visualização. A visualização está preocupada com a manipulação do *canvas* da aplicação e não com objetos contidos nele.

Características de Implementação

O sistema Interact utiliza a linguagem de programação C++ e o sistema IUP/LED de interface com o usuário [Levy e outros, 1996] desenvolvido pelo TeCGraf, Grupo de Tecnologia em Computação Gráfica da PUC-Rio. O Interact está diretamente associado ao elemento de interface *canvas* do sistema IUP/LED. A Figura 13 mostra a organização em classes do Interact.

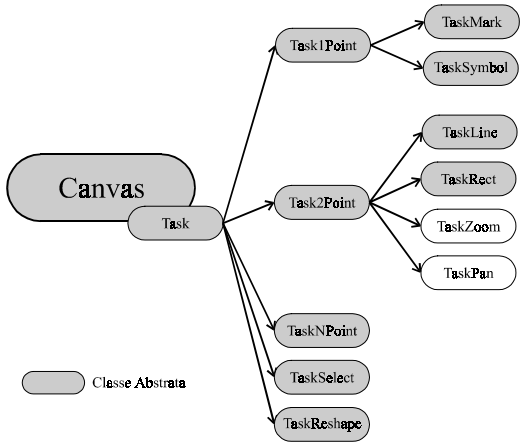

Figura 13: Hierarquia de classes do Interact.

A grande maioria das classes é abstrata. Isto requer que o programador defina os métodos virtuais puros para utilizá-la. Estes métodos são, em geral, a implementação das *callbacks* definidas pelo modelo, por exemplo, `EndObject`.

As principais tarefas de interação já estão definidas e a criação de uma nova forma de interação é facilitada através de herança e redefinição de métodos virtuais. Com isto, o modelo pode ser facilmente estendido.

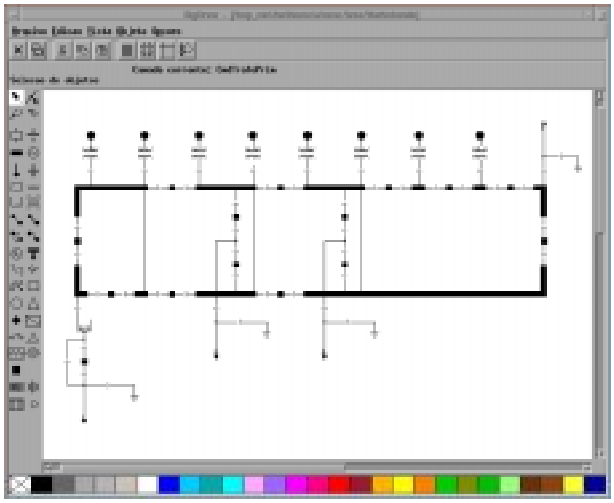

Figura 14: O sistema SigDraw.

A Figura 14 mostra um exemplo de aplicação que utiliza o Interact. O sistema SigDraw [SigDraw, 1993] é um editor gráfico de diagramas de sistemas de energia elétrica desenvolvido no projeto "Nova Geração de Centros de Controle" [CEPEL, 1992], através do convênio CEPEL/TeCGraf. Apesar de ser um editor de domínio específico (manipula equipamentos elétricos), muitos recursos são comuns aos editores gráficos convencionais.

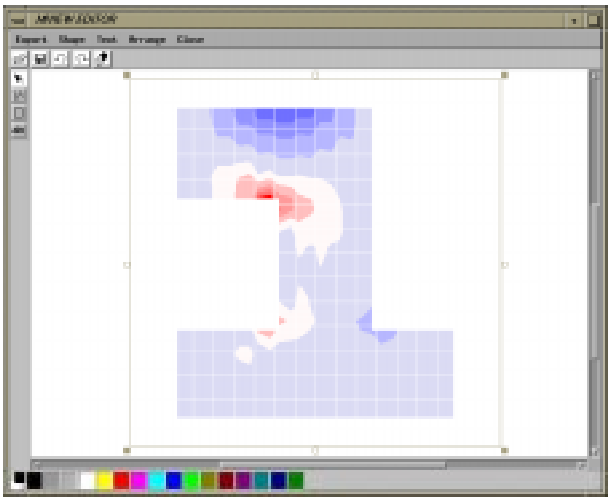

Figura 15: O sistema MVIEW.

A Figura 15 mostra mais uma aplicação que utiliza o Interact. O sistema MVIEW [MVIEW, 1996], desenvolvido através do convênio CENPES/TeCGraf, é um pós-processador gráfico interativo que permite a visualização de resultados de uma análise por elementos finitos. O Interact foi utilizado principalmente para facilitar a geração de relatórios, permitindo que o usuário inclua textos, legendas, decorações etc., além de facilitar a impressão, através de um mecanismo de formatação de página (*page layout*). A experiência comprovou que a utilização do modelo Interact permitiu um expressivo ganho de produtividade na programação de aplicações interativas em geral, não apenas as exemplificadas neste trabalho.

Conclusão

Este trabalho aborda as formas de interação mais usuais em *canvases* de programas baseados em modelos geométricos bidimensionais, do tipo de editores gráficos. O principal objetivo deste estudo é permitir o desenvolvimento de aplicações gráficas interativas com mais facilidade, rapidez e organização. Estes objetivos foram atingidos através da criação de um modelo de interação flexível, de fácil expansão, permitindo reutilização de código.

Vários aspectos positivos merecem ser destacados neste modelo. Primeiro, por ser bastante genérico, pode ser utilizado em diversas aplicações sem mudanças significativas. Segundo, por exigir pouca intervenção da aplicação, requer pouco esforço para ser utilizado. As

intervenções só ocorrem porque o Interact, para não perder a generalidade, desconhece as estruturas de dados da aplicação. Outro aspecto, é permitir a padronização das tarefas de interação sob o ponto de vista do usuário. Duas aplicações que utilizam o mesmo modelo se comportam de maneira semelhante no tratamento da interação, mesmo que seus domínios sejam distintos. Finalmente, foi observado, no desenvolvimento de aplicações, que a utilização do Interact promove o re-uso de um módulo complexo de ser programado, reduzindo de forma significativa o tempo de desenvolvimento.

Agradecimentos

Este trabalho foi desenvolvido no TeCGraf, Grupo de Tecnologia em Computação Gráfica da PUC-Rio. O TeCGraf é suportado financeiramente através de projetos, principalmente com a PETROBRAS/CENPES. O MCT, CAPES e CEPEL também financiaram este trabalho. Este trabalho tem suporte parcial do CNPq através do projeto temático GEOTEC. Os autores agradecem a Juli Huang, Waldemar Filho, Marcelo Tílio, Luiz Corrêa, Gilberto Azevedo e Maria Fernanda pelas inúmeras sugestões durante o desenvolvimento deste trabalho.

Referências

[Carneiro, 1995], Carneiro, M. M., Interact: *Um Modelo de Interação para Editores Gráficos*, Dissertação de Mestrado do Departamento de Informática. PUC-Rio, 1995.

[CEPEL, 1992], CEPEL - Centro de Pesquisas de Energia Elétrica, *Desenvolvimento de Uma Nova Geração de Centros de Controle - Especificação Funcional*, Projeto CEPEL/TeCGraf, Maio 1992.

[Hartson, 1989], Hartson, R., *User-Interface Management Control and Communication*, IEEE Software, 1989, January, 62-70.

[Levy e outros, 1996], Levy, C. H., de Figueiredo, L. H., Gattass, M., Lucena, C., Cowan, D., IUP/LED: *a portable user interface development tool*, Software: Practice & Experience 26 #7 (1996) 737-762.

[MVIEW, 1996], TeCGraf, Grupo de Tecnologia em Computação Gráfica, PUC-Rio. *MVIEW - Bidimensional Mesh View Versão 2.1 - Manual do Usuário*, Abril, 1996.

[Neider e Woo, 1993], Neider, J. e Woo, M., (Open GL Architecture Review Board). *Open GL Programming Guide*, Addison-Wesley, Reading, Massachusetts, USA, 1993.

[Nye, 1990], Nye, A., *The Definitive Guides to the X Window System. Volume I: Xlib Programming Manual for Version 11*, O'Reilly & Associates, Inc, 1990.

[OSF, 1991], Open Software Foundation, *OSF/Motif Programmer's Guide*, Revision 1.1, Prentice Hall, Englewood Cliffs, New Jersey, 1991.

[SigDraw, 1995], Carneiro, M. M., Huang, J., SigDraw: *Documentação de Desenvolvimento*, Relatório técnico TeCGraf/CEPEL, 1995.

[Szekely e outros, 1993], Szekely, P., Luo, P., Neches, R., *Beyond Interface Builders: Model-Based Interface Tools*, CHI'1993, 383-390, April.

[Vlissides e Linton, 1990], Vlissides, J. M. e Linton, M. A., *Unidraw: A Framework for Building Domain-Specific Graphical Editors*, ACM Transactions on Information Systems, Vol 8. No.3, July, 1990, pages 237-268.

[Wernecke, 1994], Wernecke, J., *The Inventor Mentor: Programming Object-oriented 3D Graphics with Open Inventor Release 2*, Addison-Wesley, 1994.