Interact

Biblioteca de interação gráfica com usuário sobre canvas IUP

Autor
Marcelo Medeiros Carneiro
Revisado por
Newton Cunha Sanches

Índice

Introdução

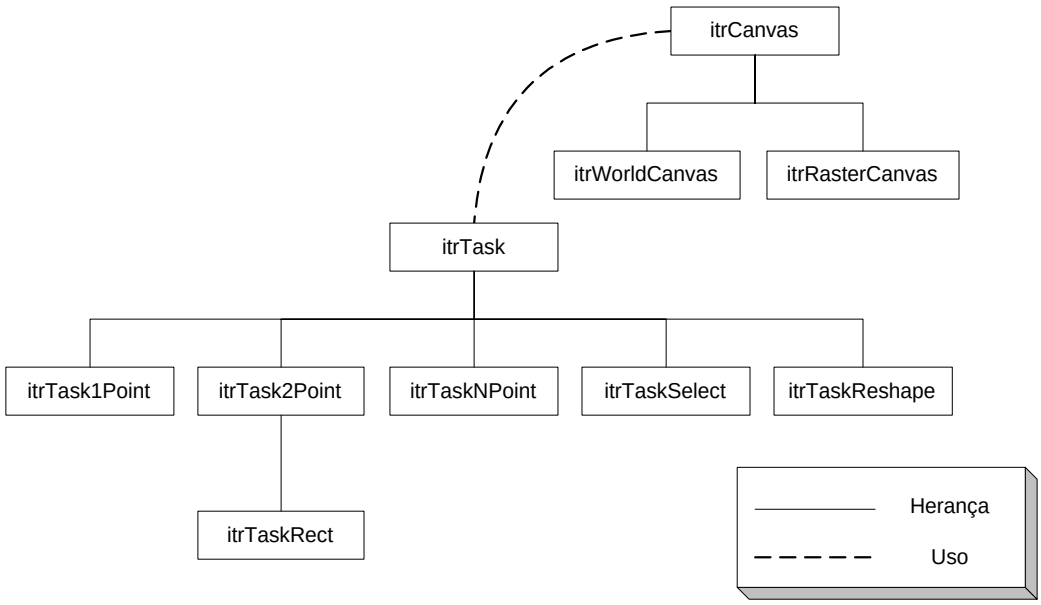
Interact é uma biblioteca de classes em C++ destinada a auxiliar programadores que queiram prover, em suas aplicações gráficas, interações entre o usuário e objetos contidos no canvas. Para tanto o Interact tenta aumentar a granularidade da comunicação entre aplicação e sistema de interface. Este aumento de granularidade é conseguido através da introdução do conceito de **tarefa**. Uma tarefa nada mais é do que um agrupamento lógico de reações aos eventos de interface gerados por um canvas IUP visando algum objetivo em especial. Ao utilizar uma tarefa a aplicação se livra do trabalho de tratar cada evento (callback) IUP individualmente. Este tratamento passa a ser feito pela tarefa. A aplicação só é consultada ou notificada quando a tarefa que está sendo executada precisa de alguma informação ou quando esta possui algum resultado para entregar. Além disso, as tarefas Interact providenciam muito dos feedbacks visuais necessários na interação entre o usuário e os objetos no canvas.

Exemplo 1:
O usuário quer desenhar um quadrado. Para tanto ele deve marcar um ponto e arrastar o *mouse*, soltando-o sobre um segundo ponto. A aplicação "diz" ao Interact que o usuário irá descrever um quadrado. Isso é feito indicando que a tarefa corrente é a tarefa de captura de retângulo (itrTaskRectangle). Ao ser ativada, esta tarefa fica esperando pelo primeiro ponto (Button1Pressed). Uma vez recebido, o Interact irá desenhar retângulos que partem do primeiro ponto dado e terminam na posição corrente do cursor, fornecendo feedback ao usuário. Depois que o usuário fornece o segundo ponto (Button1Released) o Interact informa à aplicação os dois pontos marcados, que definem o retângulo. Aplicação deverá tratar esta informação da forma que lhe for conveniente. Em seguida o Interact passa a um novo estado de "espera por retângulos".

A proposta do Interact é que o programador apenas diga que tipo de tarefa deve ser executada num dado instante, seja uma tarefa de capturar um ponto, tarefa de capturar uma *polyline* com vários pontos, tarefa de seleção, etc. O Interact executa a tarefa e, quando encerrada, passa para aplicação as informações relevantes. Ou seja, a aplicação se preocupa somente com o resultado final da interação, despreocupando-se da captura de pontos e de *feedbacks*. Embora o comportamento do *feedback*, quando e como ele é atualizado, seja de responsabilidade do Interact, o conteúdo do *feedback*, ou seja, que desenho é usado para representá-lo, pode (e muitas vezes deve) ser redefinido pela aplicação.

As tarefas Interact se destinam a serem usadas primariamente por derivação. Isto quer dizer que as tarefas Interact são classes abstratas, com métodos virtuais vazios ou virtuais puros que se destinam a particularizar seu comportamento de acordo com a aplicação em que são usadas. Assim sendo, o que acontece quando, no **exemplo 1**, o usuário acabar de construir o retângulo é definido pela aplicação através da redefinição de um método virtual (no caso EndObject()). Este método virtual acaba agindo como uma *callback* para a aplicação.

A hierarquia de classes do Interact é indicada na figura abaixo:



Inicialização

Antes de se instanciar qualquer das classes interact, a biblioteca interact deve ser inicializada por meio da função ItrOpen(), que deve ser chamanda após a inicialização do IUP (IupOpen).

void ItrOpen(void)
Função: Abre a biblioteca.

Os Canvases

Como base onde as tarefas se ancoram foi definido o conceito de canvas Interact (**itrCanvas**). Este canvas encapsula o canvas IUP interceptando todas as suas *callbacks* e transformando-as em chamadas a métodos virtuais. As atribuições deste canvas são as seguintes:

- Acumular as tarefas ativas (pilha de tarefas ativas).
- Controlar a pintura do canvas em si e das tarefas que estão ativas.
- Controlar a pintura incremental do canvas (interceptar a *callback* IUP_IDLE), evitando que esta pintura incremental interfira com os *feedbacks* do canvas. Convém aqui lembrar que, como estes *feedbacks* são desenhados em modo XOR, eles não podem ser sobrescritos por desenhos em modo COPY.

A cada instante só existe uma tarefa ativa em cada canvas. Isto quer dizer que apenas uma tarefa responderá ao eventos do canvas IUP. No entanto pode haver outras tarefas "dormentes" no canvas. Estas tarefas, embora não recebam eventos de mouse ou teclado, terão seus feedbacks redesenhados adequadamente. Na verdade o canvas contém uma pilha de tarefas. Nesta pilha apenas a tarefa do topo está ativa, estando as demais dormentes. Através deste mecanismo de pilha de tarefas é possível a qualquer momento empilhar (PushTask()) uma nova tarefa, digamos, para capturar uma região que queremos visualizar em detalhe, sem perder o estado da tarefa ativa anteriormente. Assim que o detalhe tenha sido obtido, basta desempilhar (PopTask()) a tarefa responsável por sua obtenção. A tarefa que estava ativa antes do detalhe voltará a estar no topo da pilha, voltando a estar ativa, exatamente no estado em que estava antes do empilhamento da tarefa de detalhe.

itrCanvas

A classe **itrCanvas** encapsula as chamadas decorrentes de eventos ocorridos no canvas IUP. Entre seus atributos há uma pilha de tarefas cuja tarefa de topo recebe todos os eventos de interface. A comunicação entre o canvas e estas tarefas é feita constante e automaticamente, e inclui o repasse de todos os eventos IUP. Ainda assim, a aplicação não está restrita a realizar todas as suas manipulações via tarefas do Interact. Ela pode receber os eventos do IUP diretamente, bastando redefinir os métodos virtuais vazios (na verdade Repaint() é virtual puro):

virtual void Button1Pressed (), virtual void Button1Motion (), virtual void Button1Released (), virtual void Button3Pressed (), virtual void Button3Released (), virtual void MouseMove (), virtual void EnterWindow (), virtual void LeaveWindow (), virtual void Keyboard (int key), virtual void Repaint (), virtual void Resize ().

O que é altamente desaconselhado e pode causar mal funcionamento do Interact é a aplicação tentar usar o eventos do canvas IUP diretamente, isto é, redefinir as callbacks do IUP.

Como visto na hierarquia de classes, o Interact possui dois tipos de canvas, itrRasterCanvas e itrWorldCanvas. Isto ocorre devido a necessidade de suportar tanto aplicações que desejem trabalhar com o canvas em pixels quanto aquelas que trabalham com o conceito de *world* e *window*.

Assim sendo, a classe básica itrCanvas é uma classe abstrata. Todas as ações que envolvem sistemas de coordenadas são delegadas a classes que derivem dela. É por isso que os métodos _MoveWindow(), _ResizeWindow(), _PrepareCoords(), _PrepareToRepaint() são virtuais puros.

itrCanvas::itrCanvas(Ihandle *h, cdCanvas *cc)

Função: Construtor da classe
Entrada: h - Handle do canvas Iup ao qual o itrCanvas vai ser associado.
cc - canvas do CD, se o usuário já tiver criado um para o Canvas.
NULL se o usuário deseja que o construtor crie seu próprio canvas CD.

void itrCanvas::SetCursor (itrCursor crsr, int snap)
Função: Ajusta o tipo de *cursor* do Canvas
Entrada: **crsr** - Tipo do *cursor*, dentre os valores listados nas tabelas abaixo. Os nomes físicos são são nomes de cursor que relamente descrevem a aparência dos cursores. Os nomes funcionais descrevem uma semântica que o cursor vai exercer. Os nomes funcionais são associados a nomes físicos como visto na tabela. Dentro de tarefas é preferível utilizar nomes funcionais.

Nomes físicos de cursores			
ITR_C_ARROW	ITR_C_BUSY	ITR_C_RESIZE_N	ITR_C_RESIZE_S
ITR_C_RESIZE_E	ITR_C_RESIZE_W	ITR_C_RESIZE_NE	ITR_C_RESIZE_NW
ITR_C_RESIZE_SE	ITR_C_RESIZE_SW	ITR_C_MOVE	ITR_C_HAND
ITR_C_NONE	ITR_C_IUP	ITR_C_CROSS	ITR_C_PEN
ITR_C_TEXT	ITR_C_RESIZE_C	ITR_C_OPENHAND	ITR_C_CROSSHAIR
ITR_C_CROSS_BUSY	ITR_C_CROSS_ARROW	ITR_C_CROSS_MOVE	ITR_C_CROSS_HAND

Cursor funcional	Equivalente físico (*default*)
ITR_C_NORMAL	ITR_C_ARROW
ITR_C_OVER	ITR_C_HAND
ITR_C_DRAGOBJECT	ITR_C_MOVE
ITR_C_DRAGNODE	ITR_C_MOVE
ITR_C_DRAGEDGE	ITR_C_MOVE
ITR_C_MARK	ITR_C_CROSS
ITR_C_FORBIDEN	ITR_C_BUSY
ITR_C_MOVEABLE	ITR_C_MOVE
ITR_C_SELECTABLE	ITR_C_CROSS

snap - Determina se o cursor vai aderir à grade ou não. Isto é valido para os cursores que são desenhados sobre o canvas, tais como ITR_C_CROSSHAIR.
Observação: Os cursores ITR_C_CROSS_* e ITR_C_CROSSHAIR são formados por uma linha vertical e outra horizontal que cortam todo o canvas, cuja interceção determinda o *hotspot* do cursor.

void itrCanvas::Refresh()
Função: Redesenha o canvas, isto é, prepara o canvas CD (e WD se for o caso), chama a callback de Repaint e os repaints das tasks.
Observação: Esta (Refresh()) deve ser a função chamada quando se deseja o desenho total do canvas. Embora seja a função Repaint() que deva ser redefinida para se fazer o redesenho do canvas não convém chamá-la diretamente para redesenhar o canvas.

void itrCanvas::PaintToCanvas(cdCanvas* altcnv)
Função: Faz toda a rotina de pintura, em um canvas alternativo fornecido pelo usuário. Útil para enviar o conteúdo do canvas para uma impressora, um arquivo ou outro dispositivo suportado pelo CD. Basta a aplicação abrir um canvas para o dispositivo adequado e passá-lo para **PaintToCanvas**. Os desenhos transitórios, isto é, desenhos das tarefas, **não são emitidos para este canvas**.
Entrada: **atlcnv** - Canvas no qual o desenho será feito.

void itrCanvas::HideCursor()
Função: Esconde o *cursor* CrossHair (WIRE).

void itrCanvas::ShowCursor()
Função: Mostra o *cursor* CrossHair (WIRE).

void itrCanvas::PushTask(itrTask* t)
Função: Põe um novo task no topo da pilha.
Entrada: t - a tarefa
Observação: A tarefa empilhada (a do topo) torna-se ativa, passando a receber os eventos de *mouse* e teclado. As demais tarefas da pilha terão seus feedbacks desenhados, mas não receberão eventos. Elas ficarão em "estado suspenso" até que se tornem novamente a tarefa do topo.

void itrCanvas::PopTask()
Função: Retira a tarefa to topo da pilha.

inline itrTask* itrCanvas::GetTask ()
Função: Retorna um ponteiro para o Task no topo da pilha.

inline void itrCanvas::SetNear (int on)
Função: Liga/desliga a flag que indica se o ponto atual está próximo do ponto inicial do *drag* ou não.

inline int itrCanvas::IsNear ()
Função: Consulta a flag que indica se o ponto atual esta próximo do ponto inicial do *drag.*

inline Ihandle* itrCanvas::GetHandle ()
Função: Retorna o handle do canvas IUP associado ao itrCanvas.

inline cdCanvas* itrCanvas::GetCanvas ()
Função: Retorna o cdCanvas* do canvas CD associado ao itrCanvas.

inline itrCursor itrCanvas::GetCursor()
Função: Retorna o tipo de *cursor* atual.

inline char* itrCanvas::GetEventStatus()
Função: Retorna o char* de status dos botões e teclas modificadoras.
Observação: Esta consulta só faz sentido, isto é, só retorna um valor correto quando chamada dentro de *callbacks* (fçs. virtuais) que se relacionem a eventos de *mouse*, ou funções chamadas apenas por estas. Essas *callbacks* são:
virtual void Button1Pressed (), virtual void Button1Motion(),
virtual void Button1Released (), virtual void Button3Pressed (),
virtual void MouseMove ().

void itrCanvas::EnableIdleRepaint()
Função: Habilita o desenho incremental (desenho no Idle).
Observação: Uma vez habilitado o redesenho incremental, a callback IdleRepaint() será chamada freqüentemente para que a aplicação possa realizar, em etapas, o seu redesenho. A freqüência com que esta callback será chamada é indeterminada, uma vez que o Interact se vale do recurso IUP_IDLE.
É recomendado que a função IdleRepaint() não execute código muito demorado, sob pena de prejudicar a interatividade da aplicação. Se o desenho for feito de forma incremental, é de responsabilidade da aplicação guardar as variáveis de estado que indiquem quanto do desenho já foi executado.
Se houver mais de um desenho sendo repintado incrementalmente, suas funções IdleRepaint() serão chamadas alternadamente de forma a que os dois desenhos progridam ao mesmo tempo.
É responsabilidade da aplicação indicar ao Interact, através do método DisableIdleRepaint() quando seu desenho incremental já terminou.

void itrCanvas::DisableIdleRepaint()

Função: Desabilita o desenho no Idle. Deve ser chamada sempre que a aplicação detectar que não é mais necessário chamar o seu IdleRepaint(). Por exemplo, ao detectar que o seu desenho incremental já foi completado a aplicação pode, de dentro de IdleRepaint() chamar DisableIdleRepaint().

int itrCanvas::IsIdleRepaintEnabled()

Função: Verifica se o repaint idle esta habilitado.

Saída: 0 - Desabilitado 1 - Habilitado (o estado *default* é desabilitado).

void itrCanvas::DisableFeedBack()

Função: Desabilita feedback. Todas as tarefas testam a flag de feedback do canvas antes de pintarem os seus feedbacks. Assim, ao se desligar a flag de feedback, tem-se certeza de que, até que se chame EnableFeedBack(), as tarefas não terão feedback.

Observação: O efeito de DisableFeedBack() é cumulativo. Assim sendo, se forem executados diversos DisableFeedBack()'s, apenas após um igual número de EnableFeedBack()'s o feedback votará a estar habilitado.

void itrCanvas::EnableFeedBack()

Função: Habilita feedback.

int itrCanvas::IsFeedBackEnabled()

Função: Verifica se o feedback está habilitado.

Saída: 0 - Desabilitado 1 - Habilitado (o estado *default* é habilitado).

Os métodos virtuais que fazem o papel de *callbacks* nesta classe e que aplicação pode definir são:

```
virtual void Button1Pressed ();
virtual void Button1Motion ();
virtual void Button1Released ();
virtual void Button3Pressed ();
virtual void Button3Released ();
virtual void MouseMove ();
virtual void EnterWindow ();
virtual void LeaveWindow ();
virtual void Keyboard (int key);
virtual void Repaint () = 0;
virtual void Resize ();
```

Estas funções são quase equivalentes às *callbacks* do IUP, a exceção de *Button1Motion* e *MouseMove*, que são todas geradas pelo mesmo evento IUP_MOTION_CB. A *Button1Motion* ocorre quando o *mouse* é movido enquanto o botão 1 estiver apertado e a *MouseMove* é chamada quando não.

Como visto a função Repaint() deve obrigatoriamente ser definida. Os parâmetros de posição do cursor de estado dos botões e das teclas modificadoras estão ausentes. Eles podem ser obtidos através dos métodos consulta GetEventPos(), GetEventUnsnapPos() e GetEventStatus(). O protótipo destas funções segue:

```
virtual void GetEventPos (int*, int*)=0;
virtual void GetEventPos (double*, double*)=0;
virtual void GetEventUnsnapPos (int *,int *)=0;
virtual void GetEventUnsnapPos (double*, double*)=0;
inline char* GetEventStatus  ();
```

Como já visto, a classe itrCanvas não trata sistemas de coordenadas. Assim, as classes dela derivadas que cuidam de sistema de coordenadas (como itrWorldCanvas e itrRasterCanvas) deverão definir as funções GetEventPos(), GetEventUnsnapPos(). Estas também deverão definir as funções de conversão:

virtual void Raster2World (int, int, double*, double*)=0;
virtual void World2Raster (double, double, int*, int*)=0;

A existência de versões de GetEventPos e GetEventUnsnapPos, que retornam *double* ou *int*, serve para proporcionar uniformidade para aplicações que trabalhem com sistemas de coordenadas inteiras (usualmente *Raster*) ou em ponto flutuante (usualmente valendo-se dos conceitos de *World* e *Window*).
Isto será aprofundado quando apresentarmos as classes itrRasterCanvas e itrWorldCanvas.

Assim sendo, uma aplicação não deve declarar uma classe que herde diretamente de itrCanvas. Esta deve herdar de itrRasterCanvas ou itrWorldCanvas, conforme o caso.

itrRasterCanvas

Esta classe define um sistema de coordenadas raster para itrCanvas. Por isso esta classe não manipula nem reage a eventos vindos do *scroll bar* do canvas IUP. Além disso, ela permite a definição de um mecanismo de Snap.

itrRasterCanvas redefine as funções de itrCanvas da seguinte forma:

itrRasterCanvas::itrRasterCanvas(Ihandle* h, cdCanvas* cc)
Função: Construtor
Entrada: h - handle do canvas do IUP
cc - ptr. para o canvas do CD (NULL cria um novo canvas)

void itrRasterCanvas::GetEventPos(int *x, int *y)
Função: Fornece para a aplicação o ponto onde ocorreu um evento de *mouse* em coordenadas raster do canvas.
Saída: (x,y) - posição do cursor
Observação: Esta consulta só faz sentido, isto é, só retorna um valor correto quando chamada dentro de *callbacks* (fçs. virtuais) que se relacionem a eventos de *mouse*, ou funções chamadas apenas por estas.

void itrRasterCanvas::GetEventPos(double *x, double *y)
Função: Fornece para a aplicação o ponto onde ocorreu um evento de *mouse*. Como itrRasterCanvas implementa coordenadas raster, tudo que esta função faz é retornar o ponto fornecido por GetEventPos(int*,int*) convertido para *double*.
Saída: (x,y) - posição do cursor
Observação: Idem.

void itrRasterCanvas::GetEventUnsnapPos(int* x, int* y)
Função: Fornece para a aplicação o ponto onde ocorreu um evento de *mouse*, antes de passar pelo algoritmo de Snap.
Saída: (x,y) - Posição do cursor
Observação: Idem.

v**oid itrRasterCanvas::GetEventUnsnapPos(double* x, double* y)**
Função: Fornece para a aplicação o ponto onde ocorreu um evento de *mouse*, antes de passar pelo algoritmo de Snap. Como itrRasterCanvas implementa coordenadas raster, tudo que esta função faz é retornar o ponto fornecido por GetEventUnsnapPos(int*,int*) convertido para *double*.
Saída: (x,y) - Posição do cursor
Observação: Idem.

void itrRasterCanvas::Raster2World(int ix, int iy, double* x, double* y)

Função: Converte de coordenadas raster para coordenadas do mundo. Como itrRasterCanvas implementa coordenadas raster, tudo que esta função faz é converter **ix** e **iy** para *double*.

Entrada: (ix,iy) - coordenadas raster

Saída: (x,y) - coordenadas do mundo

void itrRasterCanvas::World2Raster(double x, double y, int* ix, int* iy)

Função: Converte de coordenadas do mundo para coordenadas raster. Como itrRasterCanvas implementa coordenadas raster, tudo que esta função faz é converter **x** e **y** para *int*, arredondando.

Entrada: (x,y) - coordenadas do mundo

Saída: (ix,iy) - coordenadas raster

itrRasterCanvas define os seguintes novos métodos:

void itrRasterCanvas::GetCanvasSize(int *dx, int *dy)

Função: Fornece o tamanho do canvas (ou, da área usada) em pixels

Saída: (dx,dy) - Tamanho do canvas

void itrRasterCanvas::SetUsedArea(double xmin, double ymin, double xmax, double ymax)

Função: Ajusta a posição da área que realmente será usada por itrRasterCanvas, dentro do canvas.

Entrada: (xmin,ymin)-(xmax,ymax) - Posição dos cantos da área útil com relação às dimensões totais do canvas. Os valores xmin, ymin, xmax, ymax devem estar entre 0.0 e 1.0. Nestas coordenadas, (0,0) é o canto inferior esquerdo do canvas e (1,1) o canto superior direito.

Observação: O viewport do CD será ajustado para cobrir apenas a parcela do canvas especificada. Também o clip será ajustado para não permitir o desenho fora desta área.

void itrRasterCanvas::GetUsedArea(double* xmin, double* ymin, double* xmax, double* ymax)

Função: Obtém a posição da área que realmente é usada por itrRasterCanvas, dentro do canvas.

Saída: (xmin,ymin)-(xmax,ymax) - Posição dos cantos da area util com relação às dimensões totais do canvas. Os valores xmin, ymin, xmax, ymax devem estar entre 0.0 e 1.0. Nestas coordenadas, (0,0) é o canto inferior esquerdo do canvas e (1,1) o canto superior direito.

Uma classe derivada de itrRasterCanvas pode redefinir a função **virtual void Snap(int* x , int* y)** para criar um mecanismo de *snap to grid* no canvas. Dados **x** e **y** esta função deve retornar os novos **x** e **y** que representam a posição do evento após a operação de *snap*. Os ponto (x,y) original, isto é, aquele que foi passado para Snap é guardado dentro de itrRasterCanvas para tornar possível GetUnsnapPos().

itrWorldCanvas

A classe itr*World*Canvas define um sistema de coordenadas com o conceito de mundo e janela visível do mundo. Para isso além de definir os métodos virtuais dependentes de coordenadas de itrCanvas, ela oferece vários outros métodos para a manipulação do sistema de coordenadas.

As alterações nas dimensões do canvas do IUP podem agir nas dimensões da janela visível de algumas formas distintas:

1. O conteúdo da janela pode ter a sua razão de aspecto alterada para que a janela se adeqüe a forma da superfície de visualização na qual ela será apresentada;
2. A janela de visualização pode ser aumentada adequadamente de forma manter as razão de aspecto do mundo de usuário constante e unitária;
3. O tamanho do desenho continua o mesmo e a alteração do tamanho do canvas só faz uma parte maior ou menor do desenho ficar visível.

A escolha entre estes diversos comportamentos é feita através dos métodos SetKeepAspect() e SetResizeBehaviour().

Os métodos virtuais de itrCanvas redefinidos em itrWorldCanvas são:

void itrWorldCanvas::GetEventPos(int *x, int *y)
Função: Retorna a posição do evento de *mouse*, em coordenas raster do canvas
Saída: (x,y) - posição do cursor
Observação: Esta consulta só faz sentido, isto é, só retorna um valor correto quando chamada dentro *callbacks* (fçs. virtuais) que se relacionem a eventos de *mouse*, ou funções chamadas apenas por estas. Essas *callbacks* são:
virtual void Button1Pressed (), virtual void Button1Motion(),
virtual void Button1Released (), virtual void Button3Pressed (),
virtual void MouseMove ().

void itrWorldCanvas::GetEventPos(double *x, double *y)
Função: Retorna a posição do evento de *mouse*, em coordenas do mundo.
Saída: (x,y) - posição do cursor
Observação: Idem.

void itrWorldCanvas::GetEventUnsnapPos(int *x, int *y)
Função: Retorna a posição do evento de *mouse*, em coordenas raster, antes da operação de Snap.
Saída: (x,y) - posição do cursor
Observação: Idem.

void itrWorldCanvas::GetEventUnsnapPos(double *x, double *y)
Função: Retorna a posição do evento de *mouse*, em coordenas do mundo, antes da operação de Snap.
Saída: (x,y) - posição do cursor
Observação: Idem.

void itrWorldCanvas::Raster2World(int ix, int iy, double* x, double* y)
Função: Converte de coordenadas raster para coordenadas do mundo.
Entrada: (ix,iy) - coordenadas raster
Saída: (x,y) - coordenadas do mundo

void itrWorldCanvas::World2Raster(double x, double y, int* ix, int* iy)
Função: Converte de coordenadas do mundo para coordenadas raster.
Entrada: (x,y) - coordenadas do mundo
Saída: (ix,iy) - coordenadas raster

Os novos métodos criados em itr*World*Canvas são:

itrWorldCanvas::itrWorldCanvas(Ihandle* h, cdCanvas* cc)
Função: Construtor
Entrada: h - handle do canvas do IUP
cc - ptr. para o canvas do CD (NULL cria um novo canvas)

void itrWorldCanvas::SetWorld(double xmin, double ymin, double xmax, double ymax)
Função: Muda o tamanho do mundo
Entrada: xmin, ymin, xmax, ymax - limites do mundo

void itrWorldCanvas::SetWindow(double xmin, double ymin, double xmax, double ymax)
Função: Muda o tamanho da janela, atualizando a scrollbar. Gera um repaint automático.
Entrada: xmin, ymin, xmax, ymax - limites da janela
Observações: Se o mecanismo de *KeepAspect* estiver ligado, não é garantido que o valor de *window* retido pelo canvas seja exatamente o valor passado. Isto ocorre pois a janela retida é definida como sendo a menor possível capaz de conter a janela pedida pelo usuário centralizada em seu interior e que apresente a mesma razão de aspecto da superfície de visualização.

void itrWorldCanvas::SetLimits(double xmin, double ymin, double xmax, double ymax)
Função: Essa operação visa informar ao canvas qual o tamanho dos limites do desenho que se deseja mostrar. Atua em conjunto com a Função Fit().

void itrWorldCanvas::Fit()
Função: A partir do limite do desenho ajustado em SetLimits(), ajusta conjuntamente o *world* e a *window*. A *window* é ajustada para compreender toda a área de desenho especificada em SetLimits() e o *world* para ter dimensões três vezes maiores que a *window*. Isto é, a nova *window* fica assentada no centro de um novo mundo que tem área 9 vezes maior que esta *window*.

void itrWorldCanvas::GetWorld(double *xmin, double *ymin, double *xmax, double *ymax)
Função: Informa o *world* atual. Qualquer ponteiro NULL é ignorado
Saída: xmin, ymin, xmax, ymax - limites do *World*

void itrWorldCanvas::GetWindow(double *xmin, double *ymin, double *xmax, double *ymax)
Função: Informa a *window* atual. Qualquer ponteiro NULL e' ignorado
Saída: xmin, ymin, xmax, ymax - limites do *Window*

void itrWorldCanvas::GetLimits(double *xmin, double *ymin, double *xmax, double *ymax)
Função: Informa os limites atuais do desenho. Qualquer ponteiro NULL é ignorado
Saída: xmin,ymin,xmax,ymax - limites do desenho.

int itrWorldCanvas::SetKeepAspect(int on)
Função: Liga/desliga o flag KeepAspect
Entrada: on - novo valor do flag (0 ou 1)
Saída: retorna o valor antigo do flag.

itrResizeBehaviour itrWorldCanvas::SetResizeBehaviour(itrResizeBehaviour bh)
Função: Modifica a forma como a window é ajustada quando o canvas muda de tamanho.
Entrada: bh - novo comportamento
Saída: retorna o valor antigo do atributo. Os valores possíveis são:

ITR_SCALE	Tenta fazer com que a janela atual ocupe toda a área do canvas. Se isso for mudar a razão de aspecto, pode (segundo indicado em SetKeepAspect) aumentar a janela de forma a enquadrar nas novas dimensões do canvas toda a janela anterior.
ITR_KEEPORIGIN_UL	Mantém o desenho do mesmo tamanho. Devido ao aumento ou diminuição da área do canvas, a área visível do desenho também muda. O canto **superior esquerdo** do desenho visível antes da alteração do tamanho do canvas é mantido fixo.
ITR_KEEPORIGIN_UR	Igual ao anterior. A diferença é que o canto **superior direito** do desenho anterior ou resize é mantido fixo.
ITR_KEEPORIGIN_LL	Igual ao anterior. A diferença é que o canto **inferior esquerdo** do desenho anterior ou resize é mantido fixo.
ITR_KEEPORIGIN_LR	Igual ao anterior. A diferença é que o canto **inferior direito** do desenho anterior ou resize é mantido fixo.
ITR_KEEPORIGIN_CENTER	Igual ao anterior. A diferença é que o **centro** desenho anterior ou resize é mantido fixo.

void itrWorldCanvas::GetCanvasSize(int *dx, int *dy)

Função: Retorna as dimensões do canvas (ou da área usada), em coordenadas raster

Saída: (dx,dy) - dimensões

void itrWolrdCanvas::SetUsedArea(double xmin, double ymin, double xmax, double ymax)

Função: Ajusta a posição da área que realmente será usada por itrWorldCanvas, dentro do canvas.

Entrada: (xmin,ymin)-(xmax,ymax) - Posição dos cantos da área útil com relação às dimensões totais do canvas. Os valores xmin, ymin, xmax, ymax devem estar entre 0.0 e 1.0. Nestas coordenadas (0,0) é o canto inferior esquerdo do canvas e (1,1) o canto superior direito.

Observação: O viewport do CD será ajustado para cobrir apenas a parcela do canvas especificada. Também o clip será ajustado para não permitir o desenho fora desta área.

void itrWorldCanvas::SetUsedArea(double* xmin, double* ymin, double* xmax, double* ymax)

Função: Obtém a posição da área que realmente será usada por itrRasterCanvas, dentro do canvas.

Saída: (xmin,ymin)-(xmax,ymax) - Posição dos cantos da área útil com relação às dimensões totais do canvas. Os valores xmin, ymin, xmax, ymax devem estar entre 0.0 e 1.0. Nestas coordenadas (0,0) é o canto inferior esquerdo do canvas e (1,1) o canto superior direito.

void itrWorldCanvas::UpdateScrollBar()

Função: Atualiza as posições e tamanhos do thumb do scroll bar para se adequar ao *window* e *world* correntes.

void itrWorldCanvas::Zoom(double x, double y, double factor)

Função: Amplia o objeto visualizado 'factor' vezes, centrando a nova vista no ponto (x,y).

Entrada: (x,y) - novo ponto central
factor - factor > 1.0 = Zoom In; factor <1.0 = Zoom Out

void itrWorldCanvas::Zoom(double factor)

Função: Amplia o objeto visualizado 'factor' vezes, centrando no centro do canvas

Entrada: factor - factor > 1.0 = Zoom In; factor <1.0 = Zoom Out

void itrCanvas::PaintWindowToCanvas(cdCanvas* altcnv,
int xmin, int ymin,
int xmax, int ymax,
double wxmin, double wymin,
double wxmax, double wymax)

Função: Faz toda a rotina de pintura, em um canvas alternativo fornecido pelo usuário. Útil para enviar o conteúdo do canvas para uma impressora, um arquivo ou outro dispositivo suportado pelo CD. Basta a aplicação abrir um canvas para o dispositivo adequado e passá-lo para **PaintToCanvas**. Os desenhos transitórios, isto é, desenhos das tarefas, **não são emitidos para este canvas**.

Entrada: **atlcnv** - Canvas no qual o desenho será feito.
(xmin, ymin) - (xmax, xmax) - Viewport do canvas alternativo que será usada.
(wxmin, wymin) - (wxmax, wxmax) - Área do mundo (obviamente, em coordenadas do mundo) que será pintada no canvas alternativo.

As Tarefas

itrTask

Classe base da qual descendem todas as tarefas do Interact. Define funções virtuais que formam uma interface comum para a comunicação dos mais diversos tipos de tarefas com o itrCanvas.
As únicas funções realmente definidas em itrTask são:

inline void itrTask::SetCanvas(itrCanvas* c)
Função: Associa o Task a um canvas
Entrada: c - o canvas a ser associado
Observação: É executada automaticamente dentro de itrCanvas::PushTask(). Em geral, nunca precisará ser chamada pela aplicação.
Repare que **uma mesma tarefa não pode estar simultaneamente em dois canvases.**

inline itrCanvas* itrTask::GetCanvas()
Função: Retorna o Canvas associado ao task
Saída: ptr. para o Canvas

void itrTask::GetEventPos (int* x, int* y)
Função: Retorna a posição do cursor no momento do evento (coordenadas raster). Shortcut para GetCanvas()->GetEventPos(x,y)
Saída: (x,y) o ponto

void itrTask::GetEventPos (double* x, double* y)
Função: Retorna a posição do cursor no momento do evento (coordenadas do mundo). Shortcut para GetCanvas()->GetEventPos(x,y)
Saída: (x,y) o ponto

char* itrTask::GetEventStatus ()
Função: Retorna o status dos botões e modificadores para o evento (coordenadas do mundo). Shortcut para GetCanvas()->GetEventStatus()

As três últimas funções são de pouco uso para aqueles que vão apenas utilizar as tarefas e colher seus resultados, pois os dados a elas pertinentes podem ser consultados de forma mais direta e estruturada através de métodos oferecidos em cada uma (normalmente, GetObjectData()). Assim sendo, estas funções acabam sendo úteis apenas para aqueles que quiserem criar novas tarefas. Ver **Construindo Novas Tarefas**.

itrTask1Point

Esta tarefa destina-se a capturar um ponto no canvas e passá-lo à aplicação. Os métodos definidos são:

itrTask1Point::itrTask1Point ()
Função: Construtor

void itrTask1Point::Init ()
Função: Inicializador - É chamado por PushTask.

void itrTask1Point::Done ()
Função: Terminador - É chamado por PopTask.

void itrTask1Point::GetObjectData(int *x, int* y)
Função: Retorna o ponto que foi escolhido pelo usuário
Saída: (x,y) - ponto em coordenadas raster
Observação: Esta função só retorna valores válidos se chamada dentro de Feedback(), EndObject() ou qualquer função chamada somente dentro destas duas funções.

void itrTask1Point::GetObjectData(double *x, double* y)
Função: Retorna o ponto que foi escolhido pelo usuário
Saída: (x,y) - ponto em coordenadas do mundo
Observação: Idem.

Uma aplicação que deseje utilizar esta tarefa deve obrigatoriamente definir as seguintes funções virtuais puras:

virtual void Feedback ()
Função: Indica que a task precisa se redesenhar. Para tal a aplicação deve desenhar (**em modo XOR**) o *feedback* adequado. Para saber o ponto atual da interação deve ser consultada a função GetObjectData().
Observação: Para manter-se uniformidade, deve-se usar a cor ITR_TASK_COLOR como *cdForeground* ao desenhar em modo XOR.

virtual void EndObject ()
Função: Indica que a task finalizou uma interação. A aplicação deve responder atualizando suas estruturas internas e, se for o caso, fazendo o desenho definitivo. Para saber a posição do ponto final da interação deve ser consultada a função GetObjectData().

itrTask2Point

Esta tarefa destina-se a capturar dois pontos no canvas e passá-los à aplicação. Os métodos definidos nela são:

itrTask2Point::itrTask2Point ()
Função: Construtor

void itrTask2Point::Init ()
Função: Inicializador - É chamado por PushTask.

void itrTask2Point::Done ()
Função: Terminador - É chamado por PopTask.

void itrTask2Point::GetObjectData(int* x1, int* y1, int* x2, int* y2)
Função: Retorna os pontos que estão sendo/foram escolhidos pelo usuário
Saída: (x1,y1) - 1o. ponto em coordenadas raster
(x2,y2) - 2o. ponto em coordenadas raster
Observação: Esta função só retorna valores válidos se chamada dentro de *Feedback*(), EndObject() ou qualquer função chamada somente dentro destas duas funções.

void itrTask2Point::GetObjectData (double* x1, double* y1, double* x2, double* y2)
Função: Retorna os pontos que estão sendo/foram escolhidos pelo usuário
Saída: (x1,y1) - 1o. ponto em coordenadas do mundo
(x2,y2) - 2o. ponto em coordenadas do mundo
Observação: Idem.

Uma aplicação que deseje utilizar esta tarefa deve obrigatoriamente definir as seguintes funções virtuais puras:

virtual void Feedback ()
Função: Indica que a task precisa se redesenhar. Para tal a aplicação deve desenhar (**em modo XOR**) o *feedback* adequado. Para saber os pontos atuais da interação deve ser consultada a função GetObjectData().
Observação: Para manter-se uniformidade, deve-se usar a cor ITR_TASK_COLOR como *cdForeground* ao desenhar em modo XOR.

virtual void EndObject ()
Função: Indica que a task finalizou uma interação. A aplicação deve responder atualizando suas estruturas internas e, se for o caso, fazendo o desenho definitivo. Para saber a posição dos pontos finais da interação deve ser consultada a função GetObjectData().

itrTaskRectangle

Esta é uma herdeira direta da classe itrTask2Point. A única função que é (re-)definida nessa classe é

void itrTaskRect::Feedback ()
Função: *Feedback* de retângulo em modo XOR.

Esta tarefa disponibiliza o *feedback* mais comum que se pode dar à itrTask2Point.

itrTaskNPoint

Esta task espera que o usuário defina uma série de pontos a medida que usa o botão esquerdo do *mouse* e encerra a tarefa quando o usuário pressiona o botão direito ou o botão do meio. Durante a coleta de pontos a tarefa desenha segmentos de linhas. Quando *cursor* sai do canvas o último segmento é apagado, mas os anteriores são mantidos. Quando botão da direita ou do meio são pressionados a task é considerada encerrada e gera um evento EndObject().

void itrTaskNPoint::Init ()
Função: Inicializador - É chamado por PushTask.

void itrTaskNPoint::Done ()
Função: Terminador - É chamado por PopTask.

void itrTaskNPoint::GetObjectData (int* n, int* x[], int* y[])
Função: Retorna os pontos da *polyline* em coordenadas raster
Saída: n - no. de pontos
x - ptr. para o vetor das coordenadas x
y - ptr. para o vetor das coordenadas y
Observação: Esta função só retorna valores válidos se chamada dentro de *Feedback*(), EndObject() ou qualquer função chamada somente dentro destas duas funções.
Inclusive o ponto atualmente apontado pelo cursor é retornado. Assim se uma aplicação quiser, por exemplo, ignorar o ponto atual do cursor no momento do EndObject, ela deverá usar apenas os n-1 primeiros pontos retornados, ou seja de (x[0],y[0]) até (x[n-2],y[n-2]).

void itrTaskNPoint::GetObjectData (int* n, double* x[], double* y[])
Função: Retorna os pontos da *polyline* em coordenadas do mundo
Saída: n - no. de pontos
x - ptr. para o vetor das coordenadas x
y - ptr. para o vetor das coordenadas y
Observação: Idem.

Uma aplicação que deseje utilizar esta tarefa deve obrigatoriamente definir as seguintes funções virtuais puras:

virtual void Feedback ()

Função: Indica que a task precisa se redesenhar. Para tal a aplicação deve desenhar (**em modo XOR**) o *feedback* adequado. Para saber os pontos atuais da interação deve ser consultada a função GetObjectData().

Observação: Para manter-se uniformidade, deve-se usar a cor ITR_TASK_COLOR como *cdForeground* ao desenhar em modo XOR.

virtual void EndObject (int mode)

Função: Indica que a task finalizou uma interação. A aplicação deve responder atualizando suas estruturas internas e, se for o caso, fazendo o desenho definitivo. Para saber a posição dos pontos finais da interação deve ser consultada a função GetObjectData().

Entrada: mode - Indica se o usuário determinou que a polyline será uma primitiva fechada ou aberta. os valores possíveis são:

ITR_CLOSED - Terminou a primitiva com o botão do meio do *mouse.*

ITR_OPEN - Terminou a primitiva com o botão da direita do *mouse.*

itrTaskSelect

Ao contrário das tarefas anteriores que se destinam tipicamente a criação de objetos gráficos, a **itrTaskSelect** se dedica a selecionar e manipular estes objetos. A tarefa funciona da seguinte forma:

Quanto o usuário clica algum ponto do canvas, **TaskSelect** pergunta à aplicação se aquele ponto pertence a algum objeto (Pick()). Em caso afirmativo a aplicação deve informar ao **TaskSelect** qual a *bounding box* do objeto. Com esta informação o **TaskSelect** executará o *feedback*, que será desenhar oito marcadores distribuídos ao redor da *bounding box*. O **TaskSelect** também permite a seleção de mais de um objeto simultaneamente, através do *fence* ou *pick* com a tecla *shift* pressionada. Depois de visíveis, os marcadores podem ser usados para redimensionar os objetos. Ao final de uma manipulação destes marcadores a aplicação é notificada, sendo-lhe fornecida a matriz de transformação que traduz a manipulação. Com esta matriz a aplicação poderá atualizar o(s) objeto(s) selecionado(s).

O *feedback* de rotação é exibido se ocorrer um segundo *click* do *mouse* sobre o objeto que já esteja selecionado. Os marcadores, que anteriormente eram apenas quadrados, são transformados em círculos e setas (cisalhamento), além de uma cruz que indica o centro de rotação (este último marcador também pode ser movido).

A aparência dos marcadores de seleção é dependente das permissões de manipulação do(s) objeto(s) selecionado(s). Por exemplo, se um objeto puder ser cisalhado aparecerão no meio das laterais do bounding box setinhas (indicando cisalhamento). Se o cisalhamento estiver desabilitado, ao invés das setinhas aparecerá um pequeno 'x'. O mesmo acontece com as outras marcas.

void itrTaskSelect::Init ()

Função: Inicializador - É chamado por PushTask.

void itrTaskSelect::Done ()

Função: Terminador - É chamado por PopTask.

void itrTaskSelect::RegisterBox (void *id, int x0, int y0, int x1, int y1, int perm)

Função: Função que registra um novo objeto. Versão em coordenadas raster

Entrada: id - identificador do objeto. Este parâmetro será armazenado pelo Interact e disponibilizado para a aplicação quando necessário, não tendo significado algum para o Interact. A única exigência que se faz é que ele seja único para cada objeto da aplicação. Um bom candidato a **id** é um ponteiro para a estrutura de dados da aplicação que representa o objeto em questão.

(x0,y0),(x1,y1) - bounding box do objeto.

perm - Permissão dada ao objeto, para cada tipo de manipulação. Pode ser qualquer combinação (bitwise OR) das constantes abaixo:

ITR_PNONE	permissão para nenhuma operação
ITR_PTRANSFORM	permissão para tudo (default)
ITR_PTRANSLATE	permissão para translação
ITR_PRESIZE	permissão para alterar tamanho mantendo a razão e aspecto.
ITR_PVDISTORT	permissão para alterar a razão de aspecto puxando os manipuladores em cima e embaixo da caixa de seleção.
ITR_PHDISTORT	permissão para alterar a razão de aspecto puxando os manipuladores laterais da caixa de seleção.
ITR_PDISTORT	permissão para alterar a razão de aspecto (combinação de ITR_PHDISTORT e ITR_PVDISTORT)
ITR_PROTATE	permissão para rodar
ITR_PSHEAR	permissão para cisalhar.

A permissão real que vai ser dada num dado instante vai ser a interseção das permissões de cada um dos objetos selecionados.

Observação: Ver descrição de Pick() e Fence().

void itrTaskSelect::RegisterBox (void *id, double x0, double y0, double x1, double y1, int perm)

Função: Função que registra um novo objeto. Versão em coordenadas do mundo

Entrada: id - identificador do objeto
(x0,y0), (x1,y1) - bounding box do objeto.
perm - Idem.

Observação: Idem.

void itrTaskSelect::ChangeBox (void *id, int x0, int y0, int x1, int y1)

Função: Função que modifica a bounding box de um objeto. Versão em coordenadas raster

Entrada: id - identificador do objeto
(x0,y0),(x1,y1) - nova bounding box do objeto

Observação: Ver descrição de Transform().

void itrTaskSelect::ChangeBox (void *id, double x0, double y0, double x1, double y1)

Função: Função que modifica a bounding box de um objeto. Versão em coordenadas do mundo.

Entrada: id - identificador do objeto
(x0,y0),(x1,y1) - nova bounding box do objeto

Observação: Idem.

inline void itrTaskSelect::ResetBoxes ()

Função: "Limpa" a lista de objetos selecionados, removendo as referências a todos os objetos selecionados.

void itrTaskSelect::GetPickPos(int* x, int* y)

Função: Retorna a coordenada do *pick* corrente.

Saída: (x,y) - ponto "picked", em coordenadas raster.

Observação: Esta função só retorna valores válidos se chamada dentro de Pick(), QuickPick() ou qualquer função chamada somente dentro destas duas funções.

void itrTaskSelect::GetPickPos(double* x, double* y)

Função: Retorna a coordenada do pick corrente.

Saída: (x,y) - ponto "picked", em coordenadas do mundo.

Observação: Idem.

void itrTaskSelect::GetFenceArea(int* x1, int* y1, int* x2, int* y2)

Função: Retorna a área do fence.

Saída: (x1,y1),(x2,y2) - área do fence, em coordenadas raster

Observação: Esta função só retorna valores válidos se chamada dentro de Fence() ou qualquer função chamada somente dentro desta função.

void itrTaskSelect::GetFenceArea(double* x1, double* y1, double* x2, double* y2)
Função: Retorna a área do fence.
Saída: (x1,y1),(x2,y2) - área do fence, em coordenadas do mundo.
Observação: Idem.

inline void *itrTaskSelect::GetFirstId ()
Função: GetFirst do iterador de objetos selecionados (retorna o primeiro objeto selecionado).
Saída: retorna o identificador do dito objeto. (NULL, se não houver objetos selecionados).

inline void *itrTaskSelect::GetNextId ()
Função: GetNext do iterador de objetos selecionados (retorna o próximo objeto selecionado).
Saída: retorna o identificador do dito objeto (NULL, se não houver mais objetos selecionados).

Uma aplicação que deseje utilizar esta tarefa deve obrigatoriamente definir as seguintes funções virtuais puras:

virtual void* Pick ()
Função: Quando o usuário clica o botão esquerdo do *mouse* no canvas esta função é chamada para obter da aplicação se o ponto clicado está sobre um objeto gráfico da aplicação. Ao definir esta função a aplicação deve realizar as seguintes tarefas:
1. Obter o ponto onde ocorreu o pick através de GetPickPos();
2. Se este ponto acertou um de seus objetos cadastrar o objeto e sua *bounding box* através de RegisterBox(). Mesmo que o ponto acerte em mais de um objeto a convenção de pick manda que apenas um seja registrado;
3. Ainda, se o ponto acertou um objeto retornar o identificador (ou ponteiro) do objeto. Se o ponto não acertou objeto algum retornar NULL.

virtual void Transform(itrTransf* t)
Função: Esta função é chamada ao fim de interações de mover, escalar, rodar e cisalhar para indicar à aplicação que um (ou mais) objetos seus foram transformados. A aplicação deve definir esta função da seguinte forma:
1. Para cada objeto selecionado (obtido através de GetFirstId() e GetNextId()), transformá-lo segundo o especificado em na matriz de transformação **t**;
2. Para cada objeto selecionado atualizar sua bounding box no Interact, através de ChangeBox().

Entrada: t - A transformação realizada sobre os objetos.
Observação: Ver classe **itrTransf**.

Também é recomendável redefinir as funções abaixo, embora não sejam virtuais puras:

virtual void Fence()
Função: Ao final de uma operação de fence realizada pelo usuário no canvas, é gerada uma chamada a Fence(). A aplicação deve responder registrando os objetos gráficos que estão dentro da área de fence. Para isto deve:
1. Obter a área demarcada pelo Fence() através de GetFenceArea();
2. Para cada objeto dentro da área de fence aplicação deve registrá-lo através de RegisterBox().

virtual int QuickPick ()
Função: A medida que passa sobre o canvas, o Interact chama esta QuickPick() para saber se o *mouse* está sobre algum objeto. Para redefinir esta a aplicação deve:
1. Obter o ponto onde ocorreu o Pick() através de GetPickPos();
2. Se este ponto acertou um de seus objetos, QuickPick deve retornar 1. Caso contrário, ela deve retornar 0. Não é necessário nem é correto registrar objetos nesse momento.

Observação: Como esta função é chamada com muita freqüência, é importante que o seu processamento seja o mais rápido possível.

itrMatrix e itrTransf

itrMatrix e **itrTransf** não são tarefas, mas sim classes auxiliares de **itrTaskSelect**. Além de serem usadas nas atividades internas de itrTaskSelect, itrTransf também é de interesse da aplicação por ser passada no método itrTaskSelect::Transform(), e itrMatrix por fazer parte de itrTransf.

itrMatrix é uma matrix em coordenadas homogêneas (3x2) que serve para representar a transformação sofrida por um objeto durante a execução de uma tarefa de seleção. Ela dispõe do seguinte método:

inline itrTMatrix operator*(const itrTMatrix &a, const itrTMatrix &b)
Função: Redefinição do operador *vezes* significando concatenação (multiplicação) de matrizes

Já **itrTransf** é uma classe que engloba uma itrMatrix e oferece mais algumas facilidades. Ela dispõe dos seguintes métodos:

void itrTransf::Identity
Função: Atribui à transformação a matriz identidade

void itrTransf::Translate(double tx, double ty)
Função: Concatena uma translação à matriz de transformação
Entrada: (tx,ty) - Distância da translação

void itrTransf::Rotate(double angle)
Função: Concatena uma rotação à matriz de transformação
Entrada: angle - ângulo da rotação em radianos

void itrTransf::Scale(double sx, double sy)
Função: Concatena um aumento/diminuição à matriz de transformação
Entrada: (sx,sy) - fatores de escala

void itrTransf::Shear(double sx, double sy)
Função: Concatena um cisalhamento à matriz de transformação.
Entrada: (sx,sy) - distâncias do deslizamento ao longo dos eixos x e y.

void itrTransf::SetMatrix(itrTMatrix *tm)
Função: Atribui uma matriz à transformação
Entrada: tm - a matriz

void itrTransf::AccMatrix(itrTMatrix *tm)
Função: Concatena uma matriz à transformação
Entrada: tm - a matriz

itrTMatrix itrTransf::GetMatrix()
Função: retorna a matriz de transformação

void itrTransf::TransformPoint(double *x,double *y)
Função: Transforma o ponto x,y
Entrada: (x,y) - ponto original
Saída: (x,y) - ponto transformado.

Se a aplicação só usar esta classe para responder ao evento itrTaskSelect::Transform(), ela provavelmente só precisará do método itrTransf::TransformPoint(), que pode ser usada para transformar diretamente os pontos da primitiva a aplicação.

itrTaskReshape

Esta tarefa é responsável pela remodelagem dos objetos da aplicação. Ao contrário de itrTaskSelect que só permite uma manipulação do objeto através da aplicação de uma transformação homogênea (mais deslocamento), itrTaskReshape permite manipular os nós do objeto (ou pontos de controle para o objeto) individualmente, permitindo operações não lineares.

A tarefa funciona da seguinte forma:
Ao clicar em um ponto do canvas, TaskReshape pergunta à aplicação se o cursor está sobre um objeto (Pick()). Em caso afirmativo a aplicação deve registrar todos os nós ou pontos de controle (BeginRegister(), RegisterNode(), EndRegister()) da primitiva, juntamente com um identificador do objeto. A partir daí, a tarefa desenha o feedback. O feedback default é uma polyline ligando todos os nós cadastrados pela aplicação, bem como marcas nos próprios nós. Então, o usuário pode selecionar, mover, remover ou inserir nós.

- Para selecionar um nó, basta o usuário clicar sobre o nó. A marca do nó que era vazada passará a cheia, indicado seleção. Mais de um nó pode ser selecionado através de click com a tecla *shift* ligada ou de fence.
- Para mover os nós selecionados, basta o usuário arrastar um dos nós selecionados. Todos os demais nós selecionados moverão juntos. A própria tarefa cuidará de chamar o método de feedback. Se o usuário arrastar um nó não selecionado apenas este nó será movido. Ao final da movimentação dos nós, a aplicação será notificada através de ReshapeObject(). Assim ela poderá atualizar o estado de seu objeto com base nas novas posições dos nós.
- Para remover nós, o usuário deve pressionar a tecla DEL. TaskReshape tentará remover os nós selecionados. Para isso ele executará o método VerifyDelNode() para cada candidato a deleção. Através deste método a aplicação poderá permitir ou não a deleção dos nós em questão.
- Para inserir um novo nó, o usuário deve clicar sobre uma primitiva, mas fora dos pontos de controle. O método VerifyNewNode() pode permitir ou não a inserção e, eventualmente, alterar a posição do ponto de inserção para um ponto pertencente á polyline. Se VerifyNewNode() permitir a inserção TaskReshape colocará uma marca indicando o ponto de inserção. O usuário deverá confirmar a inserção pressionando a tecla INS. Neste momento a aplicação será notificada através de NotifyNewNode() podendo então alocar espaço para o novo nó em suas estruturas internas. Esta notificação será seguida de um ReshapeObject() desta vez já incluindo o novo nó.

void itrTaskReshape::Init ()
Função: Inicializador - É chamado por PushTask.

void itrTaskReshape::Done ()
Função: Terminador - É chamado por PopTask.

void itrTaskReshape::FeedBackNodes ()
Função: Pinta os quadrados indicando os pontos de controle da primitiva. Se os nós estão selecionados usa retângulos cheios. Se não usa retângulos vazados.
Observação: Pode ser redefinida para modificar parcialmente o feedback.

void itrTaskReshape::FeedBackLines ()
Função: Desenha uma poly-line entre os nós.
Observação: Item.

void itrTaskReshape::BeginRegister(void* id,int type)
Função: Inicia o registro de um objeto.
Entrada: id - Identificador do objeto selecionado
type - Tipo de objeto os valores possíveis são
ITR_CLOSED - É uma primitiva fechada.
ITR_OPEN - É uma primitiva aberta.
Observação: O tipo da primitiva serve para dar uma dica às versões default de feedback e do método de inserção de novos nós (FeedBackLines() e VerifyNewNode()) para testar se saber se é

válido conectar o primeiro ao último nó da lista de nós. Se a aplicação resolver redefinir estes métodos é sua responsabilidade respeitar ou não este protocolo.

void itrTaskReshape::RegisterNode(double x, double y, int st=ITR_NORMAL)
Função: Registra mais um nó da primitiva
Entrada: (x,y) - posição do novo nó em coordenada do mundo
st - estado do novo nó. O valor de st pode ser um os abaixo ou combinações (bitwise OR):

ITR_NORMAL	Não marcado nem bloqueado
ITR_MARKED	Marcado
ITR_VBOUNDED	Cercado verticalmente. Isto é, só pode ter valores da coordenada vertical entre os valores das de seus vizinhos.
ITR_HBOUNDED	Cercado horizontalmente. Isto é, só pode ter valores da coordenada horizontal entre os valores das de seus vizinhos.
ITR_BOUNDED	Cercado vertical e horizontalmente
ITR_VLOCKED	Bloqueado verticalmente. Só pode ser movido horizontalmente.
ITR_HLOCKED	Bloqueado horizontalmente. Só pode ser movido verticalmente.
ITR_LOCKED	Totalmente bloqueado. Simplesmente não pode ser movido.

Observação: Em geral se valores conflitantes forem combinados, o mais restritivo (mais abaixo nesta lista) será o considerado. Por exemplo se for especificado **ITR_HBLOCKED | ITR_HBOUNDED** o que ficará valendo é **ITR_HBLOCKED**.

void itrTaskReshape::RegisterNode(int x, int y, int st)
Função: Registra mais um no na primitiva
Entrada: (x,y) - posição do novo nó em coordenadas raster.
st - estado do novo nó.

void itrTaskReshape::EndRegister()
Função: Termina o processo de registro de uma primitiva.

Função: void* itrTaskReshape::GetId()
Função: Retorna o identificador da primitiva selecionada

int itrTaskReshape::GetNumNodes()
Função: Retorna o número de nós na primitiva correntemente selecionada.

int itrTaskReshape::GetNodeData(int i, double* x, double *y, int* state)
Função: Retorna os dados sobre o i-ésimo nó da primitiva selecionada
Saída: (x,y) - posição do nó (world).
state - estado do nó (marcado, bloqueado...).

int itrTaskReshape::GetNodeData(int i, int* x, int *y, int* state)
Função: Retorna os dados sobre o i-ésimo nó da primitiva selecionada.
Saída: (x,y) - posição do nó (raster).
state - estado do nó (marcado, bloqueado...).

void itrTaskReshape::GetPickPos(double* x, double* y)
Função: Retorna a coordenada do pick corrente.
Saída: (x,y) - ponto "picked", em coordenadas do mundo.
Observação: Idem.

void itrTaskReshape::GetPickPos(int* x, int* y)
Função: Retorna a coordenada do *pick* corrente.
Saída: (x,y) - ponto "picked", em coordenadas raster.
Observação: Esta função só retorna valores válidos se chamada dentro de Pick(), QuickPick() ou qualquer função chamada somente dentro destas duas funções.

inline void itrTaskReshape::GetNewNodePos(double* x, double* y)
Função: Retorna a posição onde a tarefa pretende inserir um novo nó.
Saída: (x,y) - posição do nó em coordenadas do mundo.
Observação: Idem.

inline void itrTaskReshape::GetNewNodePos(int* x, int* y)
Função: Retorna a posição onde a tarefa pretende inserir um novo nó.
Saída: (x,y) - posição do nó em coordenadas raster.
Observação: Esta função só retorna valores válidos se chamada dentro de VerifyNewNode() ou qualquer função chamada somente dentro desta.

void itrTaskReshape::SetNewNodePos(double x, double y)
Função: Ajusta a posição onde um novo nó deve ser inserido
Entrada: (x,y) - Novas coordenadas do novo nó (world)
Observação: Esta função só deve ser chamada dentro de VerifyNewNode() ou qualquer função chamada somente dentro desta.

void itrTaskReshape::SetNewNodePos(int x, int y)
Função: Ajusta a posição onde um novo nó deve ser inserido
Entrada: (x,y) - Novas coordenadas do novo nó (raster)
Observação: Idem.

Uma aplicação que deseje utilizar esta tarefa deve obrigatoriamente definir as seguintes funções virtuais puras:

virtual void* itrTaskReshape::Pick ()
Função: Quando o usuário clica o botão esquerdo do *mouse* no canvas esta função é chamada para obter da aplicação se o ponto clicado está sobre um objeto gráfico da aplicação. Ao definir esta função a aplicação deve realizar as seguintes tarefas:
1. Obter o ponto onde ocorreu o Pick() através de GetPickPos();
2. Se este ponto acertou um de seus objetos cadastrar o objeto e sua *bounding box* clicado através de BeginRegister(), RegisterNode() e EndRegister(). Mesmo que o ponto acerte em mais de um objeto a convenção de pick manda que apenas um seja registrado;
3. Ainda, se o ponto acertou um objeto retornar o identificador (ou ponteiro) do objeto. Se o ponto não acertou objeto algum retornar NULL.

virtual void itrTaskReshape::ReshapeObject()
Função: Esta função é chamada ao fim da movimentação de um ou mais nós. A aplicação deve definir esta função da seguinte forma:
1. Obter o objeto que está sendo modificado através de GetId() e seu número de nós (GetNumNodes());
2. Obter as posições, e eventualmente status, de todos os nós deste objeto através de GetNodeData();
3. Atualizar a sua (da aplicação) estrutura interna que representa o objeto.

int itrTaskReshape::NotifyNewNode (int)
Função: Informa à aplicação que o usuário confirmou a criação de um novo nó o indica o índice que ele passará a ocupar na lista de nós.
A aplicação deve responder a este método alocando espaço em sua estrutura interna para comportar mais um nó na primitiva.
Entrada: i - Posição do nó na nodelist.
Saída: Deve retornar o estado do nó recém criado (Marcado, bloqueado ...)
Observação: Esta função não precisa ser definida se a aplicação garantir que VerifyNewNode() retorna sempre -1. Ver definição de VerifyNewNode(), adiante.

Também é recomendável redefinir as funções abaixo, embora não sejam virtuais puras:

virtual int itrTaskReshape::QuickPick ()

Função: A medida que passa sobre o canvas, o Interact chama esta QuickPick() para saber se o *mouse* está sobre algum objeto. Para redefinir esta a aplicação deve:

1. Obter o ponto onde ocorreu o Pick() através de GetPickPos();
2. Se este ponto acertou um de seus objetos QuickPick deve retornar um identificador para este objeto. Caso contrário ela deve retornar NULL. Não é necessário nem é correto registrar nós nesse momento.

Observação: Como esta função é chamada com muita freqüência, é importante que o seu processamento seja o mais rápido possível, como indica o próprio nome, ainda que a precisão do pick seja apenas uma aproximação.

Definir os métodos abaixo é opcional, mas pode influir muito no comportamento da tarefa.

int itrTaskReshape::VerifyNewNode ()

Função: Testa se um novo nó pode ser inserido e, eventualmente, ajusta a posição onde ele será inserido. A aplicação deve redefinir este método da seguinte forma:

1. Obter o ponto onde se pretende inserir o novo nó através de GetNewNodePos();
2. Verificar se este novo nó fica sobre a primitiva que está atualmente selecionada (cujo identificador pode ser obtido com GetId());
3. Se o ponto não pode ser inserido deve retornar -1;
4. Se a posição for considerada próxima o suficiente para permitir a inserção, mas precisar de algum ajuste para ficar realmente sobre a primitiva, ajustá-la com SetNewNodePos();
5. Se realmente o nó pode ser inserido retornar o índice que ele deverá ocupar na lista de nós (0-based).

Saída: Deve retorna o índice que se deseja para o novo nó.
-1 para inibir a inserção.

Observação: Caso a aplicação não queira permitir a inserção de novos pontos de controle (nós) basta definir esta função como retornando -1. Neste caso todos os passos da descrição acima podem se ignorados, exceto o 3.

int itrTaskReshape::VerifyDelNode(int)

Função: Verifica se o i-ésimo nó pode ser removido.
A aplicação deve responder a este método da seguinte forma.

1. Obter o objeto atualmente selecionado através de GetId() e, se necessário, as coordenadas do nó que é candidato a remoção (GetNodeData());
2. Se o nó puder ser apagado, aterar as estruturas internas da aplicação que representam a primitiva, para não mais usar este nó;
3. Se o nó puder ser apagado retornar 1. Senão retornar 0.

Saída: 1 - Permite a deleção.
0 - Não permite a deleção.

Observação: Se o usuário tentar remover vários nós ao mesmo tempo (todos os que estiverem marcados), este método será invocado para cada um deles. Assim a aplicação pode filtrar aqueles que podem e aqueles que não podem ser removidos.

Construindo Novas Tarefas

Para utilizar-se uma tarefa do Interact é necessário derivar uma nova classe de *itrTask**, redefinindo-se tipicamente os métodos *Feedback()* e *EndObject()*. Este tipo de derivação não é considerado como a construção de uma nova tarefa e sim como a especialização de uma tarefa existente, para que atenda às necessidades da aplicação. Este capítulo não trata deste tipo de especialização. Ao contrário, ele trata da construção de novas tarefas, ou seja de formas novas de interação.

A construção de uma nova tarefa também envolve derivação, só que tipicamente da classe abstrata *itrTask*.

Os eventos de *mouse* são comunicados a estas tarefas através de chamadas ao métodos *MouseMove*, *Button*N*Pressed*, *Button*N*Motion* e *Button*N*Released*. Com base nas informações colhidas nestes eventos a tarefa deve guardar informação sobre o estado da interação ou sobre outros modificadores da interação. Também importantes são os métodos *LeaveWindow* e *EnterWindow* são importantes pois a tarefa pode querer remover ou inserir algum *feedback* quando o mouse sair ou entrar na área do canvas. Tipicamente isto ocorrerá se a tarefa mantiver ligado um *rubber band*, ou qualquer outro *feedback* que siga o cursor enquanto não houver nenhum botão de mouse apertado.

Alguns cuidados ou regras de boas maneiras devem ser observados dentro de tarefas:

- Manter o padrão de nomes, como *FeedBack*, *EndObject*, *GetObjectData* nos métodos que se destinarem a ser "callbacks" para a aplicação do usuário;

- Ao chamar o método *FeedBack* ou qualquer outro método que realize feedback em modo XOR, utilizar a macro ITR_CALL_FB. Por exemplo:
  ```
  ITR_CALL_FB(FeedBack);
  ```
  Esta providência impedirá que o método de *Feedback* seja chamado caso o canvas ainda não tenha completado um redesenho incremental que estivesse realizando;

- Sempre que a tarefa necessitar manter um *rubber band* ativo enquanto nenhum botão do mouse estiver apertado (como no caso de *itrTaskNPoint*), chamar *DisableIdleRepaint()* ao iniciar o *rubber band* e *EnableIdleRepaint()* ao terminá-lo. Esta providência é necessária para evitar que primitivas sejam desenhadas no canvas, via *IdleRepaint()*, sobre o *Feedback* transitório (XOR) da tarefa. Não é necessário chamar estes métodos se o *rubber band* só for ficar ativo enquanto algum botão do mouse estiver apertado, como no caso da *itrTask2Point*. Isto ocorre pois sempre que há algum botão do mouse apertado o canvas paralisa o redesenho incremental automaticamente dispensando a chamada a *DisableIdleRepaint();*

- Sempre que for desenhar algum feedback em modo XOR, ajustar a cor de frente do CD para ITR_TASK_COLOR. Esta é uma macro que equivale à variável *itrEnv::FeedBackColor*. Esta variável pode ser ajustada pela aplicação para que contraste melhor com os seus canvases;

- Sempre que trocar de cursor tentar usar um dos valores da coluna da esquerda da tabela abaixo. Estes são chamados nomes funcionais de cursores, e indicam estados da tarefa. Cada cursor funcional equivale a um nome real de cursor (coluna da direita). Os nomes reais de cursores refletem a aparência do ícone do cursor. Ao usar o nome funcional do cursor permite-se que uma aplicação redefina a amarração entre nomes funcionais e nomes físicos de cursores. Por exemplo, se a aplicação executar o seguinte código
  ```
  ITR_C_DRAGOBJECT->ChangeRef(ITR_C_HAND);
  ```
  todas as tarefas que usarem ITR_C_DRAGOBJECT como cursor de movimentação de objetos terão sua aparência modificada uniformemente para um cursos ITR_C_HAND.

Cursor funcional	Equivalente físico (*default*)
ITR_C_NORMAL	ITR_C_ARROW
ITR_C_OVER	ITR_C_HAND
ITR_C_DRAGOBJECT	ITR_C_MOVE
ITR_C_DRAGNODE	ITR_C_MOVE
ITR_C_DRAGEDGE	ITR_C_MOVE
ITR_C_MARK	ITR_C_CROSS
ITR_C_FORBIDEN	ITR_C_BUSY
ITR_C_MOVEABLE	ITR_C_MOVE
ITR_C_SELECTABLE	ITR_C_CROSS