EGOLib — Uma Biblioteca Orientada a Objetos Gráficos

Eduardo Aguiar Patrocínio
Pedro Jussieu de Rezende

Relatório Técnico DCC–24/93

Setembro de 1993

EGOLib — Uma Biblioteca Orientada a Objetos Gráficos

Eduardo Aguiar Patrocínio* Pedro Jussieu de Rezende†

Departamento de Ciência da Computação
Universidade Estadual de Campinas
13081-970 Campinas, SP

Sumário

Apresentamos a descrição de uma biblioteca de funções (**EGOLib**) construída sobre o sistema *X*-Window para manipulação de objetos gráficos, que provê facilidades para atualização de tais objetos enquanto modificações de atributos dos objetos são realizadas. Esta biblioteca constitui um nível acima à biblioteca *Xlib* permitindo ao usuário um acesso de mais alto nível a modificações de vários atributos de maneira uniforme e homogênea, resultando em código mais elegante do que é possível usando-se puramente funções de *Xlib*.

Utiliza-se o conceito de hierarquia de classes, existentes nas linguagens orientadas a objetos, para a criação de classes de objetos gráficos, desde alguns muito simples até objetos compostos como, por exemplo, árvores binárias.

Aplicações para a **EGOLib** são abundantes, e em particular, esta biblioteca vem sendo utilizada para a implementação de animações de algoritmos.

*Pesquisa desenvolvida com suporte financeiro parcial do CNPq.

†Pesquisa desenvolvida com suporte financeiro parcial do CNPq através dos auxílios 300157/90-8 e 500787/91-3.

Abstract

We present a description of a library of functions (**EGOLib**) built on the *X*-Window system for the manipulation of graphics objects, which provides facilities for the update of such objects while modifications of attributes of the objects are made. This library constitutes a level above the *Xlib* library, allowing the user a higher level access to modifications of various attributes on a homogeneous and uniform way, resulting in more elegant code than is possible using purely *Xlib* functions.

The concept of class hierarchy, present in object oriented languages, is used for creating the classes of graphics objects, from some simple ones up to composed objects such as binary trees.

Applications of **EGOLib** are abundant, and in particular, this library has been used for the implementation of algorithm animations.

1 Introdução

A biblioteca *Xlib* dispõe de um conjunto extenso de funções, manipulando atributos de janelas, desenhos de gráficos e de textos, cores, eventos, teclado, mouse etc. A generalidade que é uma das características mais importantes da *Xlib*, é por outro lado um entrave à simplicidade de sua utilização para aplicações específicas.

A biblioteca **EGOLib** (**E**lementary **G**raphical **O**bjects **Lib**rary) consiste de funções para a manipulação de objetos gráficos cujo objetivo é o de criar uma camada acima à da *Xlib*. **EGOLib** se destina basicamente ao desenho de gráficos e de textos e à manipulação de atributos relacionados. Ademais, a **EGOLib** utiliza a classe **EGOView**, para a criação de um sistema de coordenadas virtuais. A classe **EGOView** provê um "mascaramento" das funções de *Xlib* de modo que todas as funções relevantes de *Xlib* são redefinidas nesta classe.

EGOLib é desenvolvida na linguagem C++ utilizando-se exaustivamente o conceito de hierarquia de generalização/especialização. Referiremos o leitor interessado ao documento completo de descrição da **EGOLib**, uma vez que a limitação de espaço aqui não nos permite

detalhamento maior. (A documentação completa contém cerca de 80 páginas.)

1.1 Motivação: Animação de Algoritmos

A ilustração do funcionamento de algoritmos através de sua simulação gráfica é provavelmente o meio de maior eficácia para a compreensão dos seus procedimentos e até mesmo de sua complexidade.

É porém um trabalho altamente não trivial, a produção de animações de algoritmos. Para facilitar esta tarefa, alguns ambientes já foram desenvolvidos. Em particular, está em fase final de desenvolvimento, o ambiente **AnimA**, veja [AR93].

Uma das partes mais laboriosas do processo de animação de algoritmos é a criação de visualizadores gráficos.

Inicialmente, com o objetivo de atender a necessidades específicas do ambiente **AnimA**, especificamos e desenvolvemos a biblioteca **EGOLib** de objetos gráficos e um conjunto funções para manipulação destes objetos.

Posteriormente, diante da amplitude de aplicações desta biblioteca, nos pareceu oportuno torná-la acessível a programadores de aplicativos diversos.

Dentre as vantagens do uso desta biblioteca especificamente para animação de algoritmos, está uma completa padronização da interface gráfica e dos mecanismos de visualização.

O propósito deste artigo é o de descrever de forma compacta os objetos, atributos e funções que constituem a **EGOLib**. Para maior detalhamento, veja [PR93].

2 Background: Interfaces Gráficas

Sugerimos ao leitor familiarizado com conceitos básicos de computação gráfica e com o sistema *X*-Window, que continue a leitura a partir da seção 2.5.

2.1 Sistema *X*

O sistema gráfico *X* controla um *display* baseado em *bitmaps*, onde cada pixel é controlável individualmente, permitindo o desenho de figuras, e textos. Assim como outros sistemas gráficos, *X* divide a tela em janelas, as quais podem trabalhar independentemente rodando aplicações baseadas em texto, ou aplicações projetadas para aproveitar as capacidades gráficas dos dispositivos.

Para a entrada de dados, o sistema *X* utiliza, além do teclado, o *mouse* que permite o posicionamento preciso do cursor em qualquer ponto da tela e que possui botões para o controle de programas sem a utilização do teclado.

O que difere o sistema *X* de outros sistemas gráficos é que ele é baseado em um protocolo de redes de forma que este sistema pode ser utilizado em diferentes tipos de equipamentos permitindo a comunicação entre estes equipamentos.

Neste sistema, cada aplicação pode consistir de mais de uma janela facilitando o processo de entrada e saída da aplicação. No sistema *X*, as janelas são em geral retangulares e orientadas ao longo dos eixos coordenados e suas dimensões podem ser alterados pelo usuário.

Para permitir que programas que estejam sendo executados em uma máquina, sejam exibidos em outra, o sistema *X* utiliza dois processos: um servidor e um cliente. O servidor controla os recursos da máquina e os torna disponíveis para as aplicações (clientes) do usuário.

2.2 *X Window*

X Window é um sistema de interface gráfica desenvolvida no MIT (Massachusetts Institute of Technology, Cambridge, MA). Existem diversas formas de escrever aplicações utilizando *X*; algumas delas são via *Xlib* ou *XView* (vide abaixo).

2.3 *Xlib*

Xlib (*X* Library) é a interface de programação da linguagem C para a

versão 11 do sistema *X Window*, sendo o nível mais baixo de programação de interfaces gráficas no sistema *X*.

A tarefa principal da *Xlib* é traduzir as estruturas de dados e procedimentos escritos em C para eventos do protocolo *X*. Ele envia e recebe pacotes que são convertidos em estrutura de dados. *Xlib* provê acesso a todas as capacidades do sistema *X*, mas faz pouco para facilitar a programação. O conjunto de funções da *Xlib* é tão grande que se o programador é responsável pela manipulação de todos os níveis diretamente, os aplicativos gráficos ficam tão enormes que tornam-se muitíssimo propensos a degradação de performance e com grande possibilidade de erros. Daí a criação de *toolkits*, para modularizar as funções mais comuns que manipulam uma porção da interface com o usuário de uma aplicação.

2.4 *XView*

XView é uma biblioteca de funções que constitui um sistema de interface com o usuário para suportar aplicações gráficas interativas rodando sobre o sistema *X Window*.

XView provê um conjunto de objetos pré-definidos como *canvases*, *scrollbars*, *botões*, *menus* etc., e é construído sobre *Xlib* como um *toolkit* orientado a objetos que provê reusabilidade de componentes configuráveis da interface com o usuário.

2.5 EGOLib

Apesar de que tanto a *XView* quanto **EGOLib** serem baseados em *Xlib*, elas não são bibliotecas competidoras, pois foram feitas para finalidades diferentes. Enquanto que o objetivo da *XView* é formar um conjunto de funções para a criação de interface gráfica, o objetivo da **EGOLib** é criar um conjunto de funções para a manipulação de objetos gráficos, utilizando-se da interface gráfica definida através de *XView*, ou mesmo, de *Xlib*.

EGOLib é uma biblioteca capaz de desenvolver aplicações sem a necessidade de ferramentas auxiliares. Para a utilização da **EGOLib**, não

é necessário um conhecimento prévio de *Xlib*, pois **EGOLib** incorpora todas as funções relevantes de desenho desta interface a um nível de abstração mais alto.

3 Vantagens da EGOLib

A biblioteca **EGOLib** é desenvolvida sobre a biblioteca *Xlib*, e apresenta as seguintes vantagens em relação a esta:

- Na maioria dos casos, para a visualização de um único objeto gráfico utilizando *Xlib*, é necessária a realização de diversas chamadas de funções de *Xlib*. Por exemplo, para desenhar um retângulo opaco, sem sombra, com fator de zoom unitário, com o estilo e a largura de seu bordo iguais aos da "linha corrente" e não destacado, é necessária a chamada de pelo menos duas funções *Xlib*: uma para desenhar a borda do retângulo com a cor desejada e outra para desenhar o seu interior, de outra cor. Caso queiramos definir alguns dos parâmetros acima de forma menos simples, o número de chamadas cresce *enormemente*.
- Há ainda objetos que, a princípio não existem na *Xlib* e que são definidos na **EGOLib** a partir de objetos já existentes. Note que estamos nos referindo aqui à constituição de objetos manipuláveis em sua forma integral e não apenas ao mero desenho bitmap de sua imagem, que é facilmente realizável.

  Um programador, ao desenvolver a interface gráfica do seu programa não deveria ter que ocupar-se com detalhes como fazer uma seta a partir de primitivas de desenho e preenchimento de polígonos. Para estes casos, **EGOLib** define um objeto, com um conjunto de parâmetros, e o programador em uma simples linha de código, consegue especificar os valores desejados, com um trabalho muito menor. Veja seção 4.
- Ademais, certos objetos gráficos dificilmente são colocados em uma representação gráfica, dada a dificuldade de manipulação destes objetos, como um *string* ou um som. Contudo, a

biblioteca **EGOLib** provê ao usuário estes objetos, com a mesma facilidade de manipulação que a de um retângulo. (Som é tratado como um objeto "gráfico" devido à sua aplicação natural como efeitos especiais na confecção de visualizadores para animações de algoritmos.)

- Mais significativo ainda é o mecanismo para construção de objetos compostos que permite que seja agrupado um conjunto de objetos, definidos parâmetros e realizadas transformações no conjunto resultante. Desta forma, caso tenhamos uma árvore binária e queiramos mudar um atributo para toda a árvore, basta mudarmos o atributo do objeto composto "Árvore Binária", sem ter que se preocupar com o fato de que sua representação utiliza vários outros objetos.
- Com relação aos atributos dos objetos, a biblioteca *Xlib* nos provê um conjunto muito grande de parâmetros que podem ser definidos em seu Contexto Gráfico (GC). A manipulação destes parâmetros é feita de forma muito pouco natural. A **EGOLib** por sua vez vem justamente resolver este problema, dando ao usuário a possibilidade da manipulação destes atributos da forma mais natural possível. Um exemplo é a colocação de níveis de transparência em objetos. Embora *Xlib* forneça inúmeras possibilidades para a definição do nível de transparência, através de uma matriz de valores, a colocação destes "detalhes" numa interface torna a sua criação muito custosa. Daí a utilização da **EGOLib**, que, embora só defina um nível de translucidez, isto já é em geral suficiente para a maioria das aplicações.
- Além disso, **EGOLib** provê um conjunto de atributos que não são disponíveis em *Xlib*, como o destaque (highlight) e as sombras de objetos.
- A **EGOLib** também implementa o conceito de camadas de objetos, de modo que os objetos nas camadas mais altas são desenhados superpostos aos objetos nas camadas mais baixas. A mudança da camada de um objeto é feita através da chamada de uma única função, que redesenha a nova situação destes objetos.

- Além disso, a camada **EGOView** provê algumas facilidades, como a realização de zoom ou scroll de toda a região de desenho e a definição de um sistema de coordenadas virtuais. Desta forma, caso tenhamos uma imagem e queiramos mudar o seu tamanho, basta redefinirmos o sistema de coordenadas virtuais para que os objetos passem a ser desenhados no novo tamanho.
- O **EGOView** também apresenta uma lista dos objetos gráficos associados a uma região de desenho. Desta forma, caso queiramos redesenhar todos os objetos (como no caso de *refresh* da janela), basta chamarmos uma função, que percorre esta lista e redesenha cada um destes objetos, dispensando assim a necessidade da manutenção da lista de objetos dentro do próprio aplicativo do usuário.

Por estas razões, **EGOLib** é uma biblioteca extremamente útil para a criação de aplicações gráficas, proporcionando a criação de código compacto, simples e eficiente. Veja exemplos na seção 4.

3.1 Orientação a Objetos

O paradigma da programação orientada a objetos introduz uma série de conceitos muito importantes para a realização desta biblioteca. Dado que os objetos gráficos apresentam muitos atributos em comum, o conceito de generalização/especialização é extremamente útil e conveniente e é muito empregado. Deste modo, os atributos comuns a todos os objetos são definidos em uma classe genérica, e as classes especializadas definem apenas os atributos que são específicos a elas.

Este conceito é importante também para a concepção das classes, pois como estas apresentam diferentes complexidades, elas são construídas de forma hierárquica, onde as classes mais complexas derivam das classes mais simples. A possibilidade de ser fazer *overload* de funções é utilizada para que uma função possa ser chamada com diferentes sintaxes pelo usuário.

Sem este paradigma, a **EGOLib** perderia muito de sua homogeneidade e uniformidade, pois todas as características apresentadas acima seriam

prejudicadas, deixando de ser transparentes ao usuário.

4 Descrição Funcional

Para a criação dos objetos gráficos, foi criada uma classe genérica, chamada **GraphObj** da qual todos os tipos de objetos descendem. Nela estão contidos todos os atributos comuns a todos os tipos de objetos. Estes atributos são os seguintes:

- coordenada (x,y) de referência
- *egoview* associada ao objeto
- cor da borda e do fundo dos objetos
- visibilidade
- transparência
- fator de zoom
- estilo da linha
- tamanho da linha
- objeto em highlight ou não
- tamanho da sombra
- objeto selecionado ou não

A partir da classe descrita acima, foram definidas as demais classes. Estas classes foram divididas em dois tipos: classes simples e compostas. As classes simples são as seguintes:

- pixel (**Dot**)
- ponto (**Point**)
- retângulo (**Rectangle**)
- segmento (**Segment**)
- círculo (**Circle**)
- arco de círculo (**Arc**)
- elipse (**Oval**)
- arco de elipse (**OvalArc**)
- retângulo ovalado (**RoundRectangle**)
- seta (**Arrow**)
- texto (**String**)
- som (**Sound**)

Estas classes possuem atributos próprios que estão detalhados em [PR93].

As classes compostas não definem nenhum objeto gráfico novo, mas apenas agrupam objetos, formando um grupo de objetos os quais são tratados como um objeto gráfico único. Existe uma classe genérica,

chamada **Group**, das quais as demais classes descendem. Todas as funções aplicadas a um grupo de objetos são aplicadas a cada objeto pertencente ao grupo. Estas classes são:

- grupo (**Group**)
- objeto gráfico com um texto (**Text**)
- vetor vertical (**VertVector**)
- vetor horizontal (**HorizVector**)
- linha poligonal (**Polygonal**)
- árvore binária (**BinaryTree**)

Para estes objetos gráficos existem funções para definir e obter os atributos dos objetos, bem como funções para mover um objeto, rotacionar, realizar zoom e calcular a distância de um ponto a um objeto.

O conceito de camada de objeto gráfico foi definido de modo que em uma mesma *egoview*, cada objeto gráfico está em uma camada e objetos que estão em camadas superiores são mostrados superpostos a objetos em camadas inferiores. Existem funções para definir e obter este atributo, bem como uma função para mover um objeto um número de camadas para cima ou para baixo.

Um exemplo da criação de um destes objetos é apresentado na figura 1. Note que é possível definir os atributos de um objeto na sua própria construção. Na figura 2, construída a partir da figura anterior, são definidos alguns atributos destes objetos. A figura 3 apresenta a utilização de funções de movimentação para estes mesmos objetos.

Cada objeto é desenhado apenas uma vez de forma que se um objeto simples pertence a um objeto composto, ele será desenhado em apenas uma de suas ocorrências. Se o objeto composto estiver numa camada superior ao próprio objeto, ele será exibido quando o objeto composto for desenhado; caso contrário, ele será exibido quando o objeto simples for desenhado.

5 EGOView

A classe *egoview* implementa um sistema de coordenadas virtuais e encapsula as funções de desenho providas pela biblioteca *Xlib*,

```
Rectangle *rectangle[4];

void create (Egoview *egoview)
{
    rectangle[0] = new Rectangle (egoview, 50, 100, 50, 150);
    rectangle[1] = new Rectangle (egoview, 100, 100, 50, 100,
                    FILLCOLOR, BLACK,
                    0);
    rectangle[2] = new Rectangle (egoview, 150, 100, 50, 50);
    rectangle[3] = new Rectangle (egoview, 200, 100, 50, 200,
                    FILLCOLOR, GRAY,
                    0);
}  /* create */
```

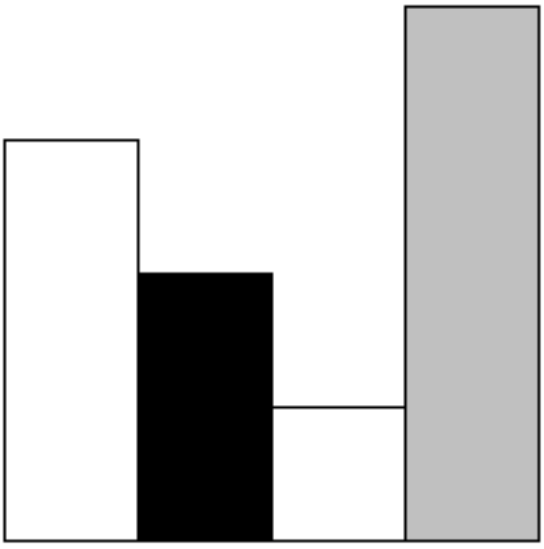

Figura 1: Exemplo da criação de objetos gráficos elementares

```
void define (Rectangle *rectangle)
{
    rectangle[0]→Set (TRANSPARENCY, TRANSLUCENT,
                // Define atributo X
                X , 125,
                // Define retangulo 0 sobre  retangulo 1
                LAYER , 2,
                0);
    rectangle[2]→SetTransparency (TRANSLUCENT);
    // Define atributo X do retangulo 2
    rectangle[2]→SetX (75);
}  /* define */
```

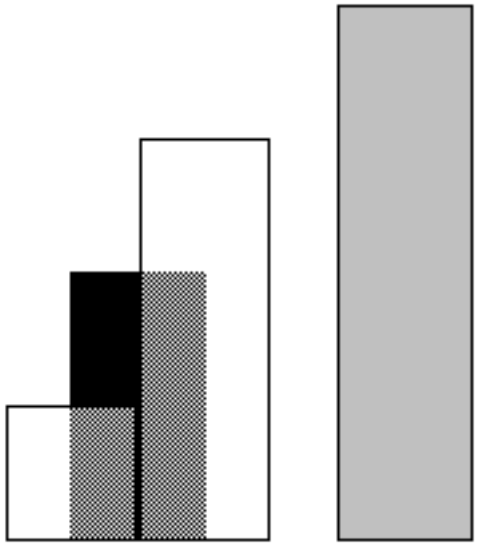

Figura 2: Exemplo da definição de alguns atributos de objetos gráficos

```
void move (Rectangle *rectangle)
{
    rectangle[0]→Move (-25);
    rectangle[2]→Move (25);
}  /* move */
```

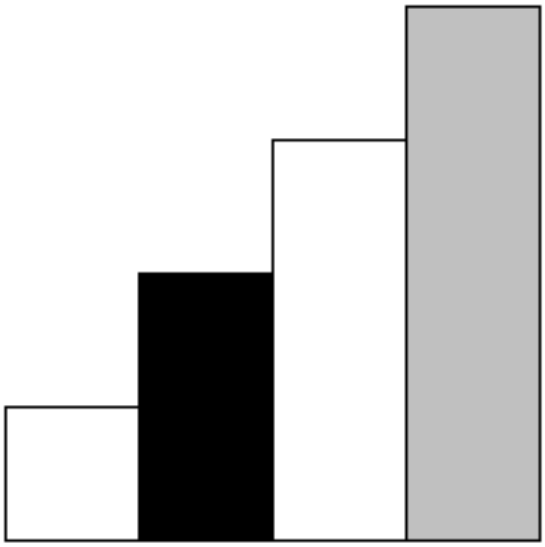

Figura 3: Exemplo da movimentação destes objetos

convertendo automaticamente as coordenadas virtuais para coordenadas reais.

Para o construtor da classe *egoview*, é necessário passar um *canvas*. Como existe uma relação natural entre um *canvas*, um *display* e uma *window*, nas chamadas das funções dos métodos da classe, não é necessário passar nenhum destes parâmetros.

Esta classe *egoview* foi desenvolvida para ser utilizada com a **EGOLib** que pode também ser vista como uma camada acima de **EGOView**, no que tange às funções de desenho. Além disso, a classe aqui apresentada interage com a **EGOLib**, armazenando uma lista de todos os objetos gráficos associados.

A **EGOView** provê métodos para: realizar *zoom*, *scroll*; obter o *canvas*, *paint window*, *display*, e a *window* a ele associada; obter as coordenadas limitantes do *canvas*, do *view*, e da *paint window*; redefinir as coordenadas virtuais da **EGOView**; transformar coordenadas virtuais em reais e vice-versa.

Além disso, todas as funções de desenho da biblioteca *Xlib*, que são encapsuladas pela **EGOView** são declaradas como *private*, já que, a priori, esta classe foi feita para ser utilizada com a biblioteca **EGOLib**, e as funções de **EGOLib** provêm uma camada acima desta, para o desenho.

6 Descrição dos atributos

A seguir descrevemos o conjunto de atributos existentes nesta biblioteca. Estes atributos são divididos em atributos comuns a todos os objetos e atributos específicos de alguns objetos.

6.1 Atributos comuns a todos os objetos

X, Y - Para todos os objetos gráficos, exceto som, está definido o conceito de ponto de referência. Cada classe o define de uma forma particular, mas geralmente é a coordenada de um dos vértices do retângulo que encapsula o objeto. Ao se alterar a

coordenada do ponto de referência, estaremos movendo o objeto de lugar.

BorderColor - Cor da borda do objeto.

FillColor - Cor do fundo do objeto. Para algumas classes, como segmento, este atributo não tem finalidade

Visible - Quando é verdadeiro, o objeto é visível.

Highlight - Todos os objetos possuem uma forma para se destacar. Cada classe define este destaque de uma forma diferente. (No caso de um retângulo, este destaque é feito, exibindo um outro retângulo, um pouco maior que o objeto, com linhas fracamente tracejadas.)

Selected - Este atributo, embora seja imperceptível graficamente, define se um objeto está selecionado. A priori, este atributo não tem nenhuma ligação com o atributo highlight.

Transparency - Os objetos podem apresentar três níveis de transparência: opaco, transparente e translúcido. No nível opaco, não é possível ver os objetos que estão em camadas inferiores à deste objeto. No nível transparente, é exibido apenas a borda do objeto. Portanto, é possível ver os objetos abaixo. No nível translúcido, embora o objeto seja totalmente mostrado, é possível ver um esboço dos objetos que estão sob este objeto.

LineStyle - Estilo da linha. Pode ser sólida, tracejada ou duplamente tracejada.

LineWidth - Define o número de pixels a ser utilizado no na largura das linhas que compões o objeto.

Zoom - Define de quanto os atributos devem ser ampliados.

ShadowX, ShadowY - Todos os objetos, podem ser desenhados com sombra. Estes atributos definem o número de pixels a serem utilizados para o desenho da sombra nas direções x e y.

Layer - Camada em que o objeto é desenhado.

AutoFlush - quando este atributo está ligado, todas as manipulações dos objetos serão refletidas no canvas, pois é feita uma chamada de um

comando da **EGOView** automaticamente. Caso contrário, esta chamada deve ser feita explicitamente.

6.2 Atributos específicos de alguns objetos

A seguir apresentamos os atributos que são específicos de algumas classes. Os nomes entre parênteses indicam as classes nas quais estes atributos estão definidos.

Para a classe composta **Group**, todos os atributos abaixo estão definidos. Ao se definir um destes atributos num objeto do tipo **Group**, é feita uma chamada da função para cada objeto pertencente ao grupo.

Height (Arrow, Oval, OvalArc, Rectangle, RoundRectangle, Segment) - define a altura do retângulo que encapsula o objeto;

Width (Arrow, Oval, OvalArc, Rectangle, RoundRectangle, Segment) - define o comprimento do retângulo que encapsula o objeto;

Size (Arc, Circle) - define o tamanho do lado do quadrado que encapsula o objeto;

Radius (Arc, Circle, RoundRectangle) - define o raio do objeto;

XRadius (Oval, OvalArc) - define o tamanho do raio no eixo x do objeto;

YRadius (Oval, OvalArc) - define o tamanho do raio no eixo y do objeto;

Angle (Arc, OvalArc) - define o ângulo total do objeto;

InitialAngle (arc, OvalArc) - define o ângulo inicial do objeto;

Arrow (Arrow) - define a presença da ponta da seta em um dos extremos;

ArrowWidth (Arrow) - define o tamanho da ponta da seta; se o módulo deste valor for menor que 1, representa um valor proporcional ao tamanho da seta; se for maior que 1, este valor é absoluto;

ArrowHeight (Arrow) - define a largura da ponta da seta; segue a mesma convenção que o atributo ArrowWidth;

ArrowRewind (Arrow) - se este valor for zero, a ponta da seta será um triângulo. Caso

contrario, este atributo define quantos pixels os dois extremos simétricos da ponta da seta estarão deslocados em relação ao triângulo. Utiliza a mesma convenção que o atributo ArrowWidth;

Family (String) - define o tipo de fonte de letra a ser utilizado;

FontSize (String) - define o tamanho da fonte da letra;

String (String) - define a cadeia a ser escrita;

Style (String) - define o estilo da letra (itálico, negrito etc.);

GroupMode (Group) - define o modo como uma chamada de uma função a um grupo será propagada pelos elementos pertencentes a este grupo.

LeftSegment (BinaryTree) - define um objeto do tipo Segment a ser tomado como a aresta que liga um nó da árvore ao seu filho esquerdo;

LeftTree (BinaryTree) - define a sub-árvore esquerda do objeto atual;

Object (BinaryTree) - define o objeto a ser tomado como o nó da árvore;

RightSegment (BinaryTree) - define um objeto do tipo Segment a ser tomado como a aresta que liga um nó da árvore ao seu filho direito;

RightTree (BinaryTree) - define a sub-árvore direita do objeto atual;

Sound (Sound) - define um arquivo, contendo um som, que será tocado quando este objeto for "exibido";

Synchronized (Sound) - define se o fluxo do programa irá continuar enquanto é tocado o som, ou se espera até o som terminar para dar prosseguimento;

Volume (Sound) - define o volume em que será tocado um som;

7 Descrição das Funções

Distance - calcula a distância de um ponto ao objeto;

GetAngle - obtém o ângulo do objeto;

GetArrow - obtém a presença da ponta da seta em um dos extremos;

GetArrowHeight - obtém a altura da ponta da seta;

GetArrowRewind - obtém o recuo da ponta da seta;

GetArrowWidth - obtém o tamanho da ponta da seta;

GetAutoFlush - obtém valor do indicador de flush automático;

GetBorderColor - obtém cor da borda;

GetFamily - obtém a família de fontes a ser utilizada nos caracteres;

GetFillColor - obtém cor do fundo;

GetFontSize - obtém o tamanho do fonte a ser utilizado nos caracteres;

GetGroupMode - obtém o modo como os atributos serão propagados para o grupo;

GetHeight - obtém a altura de um objeto;

GetHighlight - obtém o highlight do objeto;

GetInitialAngle - obtém o ângulo inicial dos objetos;

GetLayer - obtém a camada em que o objeto é desenhado;

GetLeftSegment - obtém o segmento associado ao filho esquerdo do nó atual;

GetLeftTree - obtém o apontador do objeto que é o filho esquerdo do nó atual;

GetLineStyle - obtém o estilo da linha;

GetLineWidth - obtém o tamanho da linha;

GetObject - obtém o apontador do objeto associado ao nó da árvore;

GetRadius - obtém o raio do círculo;

GetRightSegment - obtém o segmento associado ao filho direito do nó atual;

GetRightTree - obtém o apontador do objeto que é o filho direito do nó atual;

GetSelected - obtém um valor indicando se o objeto está selecionado;

GetShadowX - obtém o tamanho da sombra no eixo x;

GetShadowY - obtém o tamanho da sombra no eixo y;

GetSize - obtém o tamanho do objeto;

GetSound - obtém o nome do arquivo de som;

GetString - obtém a cadeia de caracteres a ser exibida;

GetStyle - obtém o estilo do fonte a ser utilizado na cadeia de caracteres a ser exibida;

GetSynchronized - obtém um valor indicando se deve haver sincronismo quando se toca um som ou não;

GetTransparency - obtém a transparência de um objeto;

GetVisible - obtém a visibilidade do objeto;

GetVolume - obtém o volume em que será tocado o som;

GetWidth - obtém o comprimento de um objeto;

GetX - obtém a coordenada x do ponto de referência do objeto;

GetXRadius - obtém o tamanho do raio no eixo x da elipse;

GetY - obtém a coordenada y do ponto de referência do objeto;

GetYRadius - obtém o tamanho do raio no eixo y da elipse;

GetZoom - obtém o fator de zoom de um objeto;

Move - move o ponto de referência de um objeto, relativamente à posição atual;

MoveLayer - desloca o objeto algumas camadas acima ou abaixo da camada atual;

Set - define diversos atributos do objeto;

SetAngle - define o ângulo dos objetos;

SetArrow - define a presença da ponta da seta em um dos extremos;

SetArrowHeight - define a altura da ponta da seta;

SetArrowRewind - define o recuo da ponta da seta;

SetArrowWidth - define o tamanho da ponta da seta;

SetAutoFlush - define se as alterações são realizadas sincronizadamente à sua chamada ou podem ser feitas de modo assíncrono;

SetBorderColor - define a cor da borda do objeto;

SetFillColor - define a cor de preenchimento do objeto;

SetFamily - define a família de fontes a ser utilizado nos caracteres;

SetFontSize - define o tamanho do fonte a ser utilizado nos caracteres;

SetGroupMode - define o modo como os atributos serão propagados para o grupo;

SetHeight - define a altura de um objeto;

SetHighlight - define o highlight do objeto;

SetInitialAngle - Define o ângulo inicial dos objetos;

SetLayer - define a camada em que o objeto é desenhado;

SetLeftSegment - define o objeto do tipo Segment que ligará o nó atual a seu filho esquerdo;

SetLeftTree - define o objeto do tipo BinaryTree que será o filho esquerdo do nó atual;

SetLineStyle - define o estilo da linha;

SetLineWidth - define o tamanho da linha;

SetObject - define o objeto que será tratado como nó atual;

SetRadius - define o raio do círculo;

SetRightSegment - define o objeto que ligará o nó atual ao seu filho direito;

SetRightTree - define o objeto do tipo BinaryTree que será tratado como filho direito do nó atual;

SetSelected - define se o objeto está selecionado;

SetShadowX - define o tamanho da sombra no eixo x;

SetShadowY - define o tamanho da sombra no eixo y;

SetSize - define o tamanho do objeto;

SetSound - define o arquivo de som;

SetString - define a cadeia de caracteres a ser exibida;

SetStyle - define o estilo do fonte a ser utilizado na cadeia de caracteres a ser exibida;

SetSynchronized - define se deve haver sincronismo quando se toca um som ou não;

SetTransparency - define a transparência de um objeto;

SetVisible - define a visibilidade do objeto;

SetVolume - define o volume em que será tocado o som;

SetWidth - define o comprimento de um objeto;

SetX - define a coordenada x do ponto de referência do objeto;

SetXRadius - define o tamanho do raio no eixo x da elipse;

SetY - define a coordenada y do ponto de referência do objeto;

SetYRadius - define o tamanho do raio no eixo y da elipse;

SetZoom - define o fator de zoom de um objeto;

Zoom - realiza um zoom em um objeto.

8 Conclusão

Descrevemos a biblioteca **EGOLib** que permite o desenvolvimento de aplicativos que trabalham com objetos gráficos e que requerem manipulação de atributos de tais objetos, sem a necessidade acesso direto a *Xlib* pois incorpora todas as funções relevantes de desenho desta biblioteca a um nível de abstração mais alto.

EGOLib vem sendo utilizada extensivamente e com grande sucesso no desenvolvimento de visualizadores gráficos para animações de algoritmos. Dentre as vantagens de seu uso neste contexto, está uma completa padronização da interface gráfica e dos mecanismos de visualização.

Diante da sua generalidade e da amplitude de aplicações desta biblioteca, nos pareceu adequado torná-la acessível a programadores de aplicativos de outras naturezas. Duas das principais características incorporadas são o gerenciamento interno da lista de objetos gráficos e o objeto do tipo **Group**. O primeiro dispensa que a manutenção da lista de objetos em existência tenha que ser feita pelo aplicativo do usuário e o segundo permite a criação de novos objetos compostos que por sua vez são tratados com a mesma uniformidade de tratamento de atributos que os objetos pré-definidos.

Seguem a esta seção dois exemplos de definição e de utilização de uma classe.

A Exemplo da definição de uma classe

A seguir apresentamos para efeito de ilustração a definição da classe Rectangle. Para maiores detalhes, referenciamos o leitor ao manual de documentação da **EGOLib** [PR93].

A.1 Definição da classe Rectangle

```
/*****************************************************************
 *
 * Modulo:      Rectangle
 *
 * Arquivo:     rectangle.h
 *
 * Funcao:      Implementacao da classe Rectangle.
 *
 * Estrutura:  Como atributo proprio, esta classe so' possui o tamanho
 *              e a largura do retangulo.
 *
 * Autor:       Eduardo Aguiar Patrocinio
 *
 * Versao :                    1.6
 * Data de Criacao:            17/12/91
 *
 * Data da Ultima Alteracao: 04/08/92
 *
 *****************************************************************/

#ifndef _RECTANGLE_DEFINED
#define _RECTANGLE_DEFINED

#include "graphobj.h"

class Rectangle : public GraphObj
{
    public :
      Rectangle (EgoView *, double, double, double, double);
      Rectangle (EgoView *,
```

```
                double,
                double,
                double,
                double,
                va_tdcl (SetOption));
    ~Rectangle (void);

    virtual double Distance (double, double);
    Boolean Visible (void);
    void SetHeight (double);
    void SetWidth (double);
    virtual double GetHeight (void);
    virtual double GetWidth (void);
  private :
    virtual void Border (double, double);
    virtual void Fill (void);
    virtual void GetArea (double &, double &, double &, double &);
    virtual void Highlight (double, double);
    virtual void MagnifyZoomFactor (double);

    double height;
    double width;
};                                                    /* Rectangle */

typedef Rectangle *RectanglePointer;
#endif
```

B Exemplo da utilização de uma classe

A seguir apresentamos o código de um procedimento que recebe como parâmetro o *egoview* a ser utilizado, um vetor e o número de elementos deste e cria um "vetor" de retângulos, com o tamanho proporcional ao valor da respectiva posição no vetor.

Note-se a simplicidade do código abaixo em vista do propósito pretendido, e compare-se com o enorme comprimento de código equivalente escrito exclusivamente com chamadas a funções de *Xlib*.

```
void showelements(Egoview *egoview, int *x, int len)
```

```
{
    double window_min_x;
    double window_min_y;
    double window_max_x;
    double window_max_y;

    length = len;

    egoview→get_values(window_min_x, window_max_y, window_max_x,
window_min_y);
    height = 0.8*((window_max_y-window_min_y) / length);

    int i;

    rect = new RectanglePointer [length];
    p = new int [length];

    width = (window_max_x - window_min_x) / length;

    for (i = 0; i < length; i++)
      {
          p[i] = i;
          rect[i] = new Rectangle (egoview, i*width, 0, width, x[i]*height);
          rect[i]→SetBorderColor (BORDER_COLOR);
          rect[i]→SetFillColor (FILL_COLOR);
          rect[i]→SetVisible (true);
      }
}
```

Referências

[AR93] Rackel V. Amorim e Pedro J. Rezende. Compreensão de algoritmos através de ambientes dedicados a animação. *Anais do XX Semish*, 1993.

[Hel90] Dan Heller. *XView Programming Manual*, volume sete. O'Reilly & Associates, Inc., segunda edição, Julho de 1990.

[Nye90] Adrian Nye. *Xlib Programming Manual (for Version 11)*, volume um. O'Reilly & Associates, Inc., Sebastapol, CA, segunda edição, Abril de 1990.

[PR93] Eduardo A. Patrocínio e Pedro J. Rezende. **EGOLib** — uma biblioteca para manipulação de objetos gráficos. *Relatorio Tecnico 80 pgs.*, 1993.

[Str] Bjarne Stroustrup. *The C++ Programming Language.* Addison-Wesley Publishing Company, primeira edição.

Relatórios Técnicos – 1992

01/92 **Applications of Finite Automata Representing Large Vocabularies,** *C. L. Lucchesi, T. Kowaltowski*

02/92 **Point Set Pattern Matching in d-Dimensions,** *P. J. de Rezende, D. T. Lee*

03/92 **On the Irrelevance of Edge Orientations on the Acyclic Directed Two Disjoint Paths Problem,** *C. L. Lucchesi, M. C. M. T. Giglio*

04/92 **A Note on Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi Diagrams,** *W. Jacometti*

05/92 **An (l, u)-Transversal Theorem for Bipartite Graphs,** *C. L. Lucchesi, D. H. Younger*

06/92 **Implementing Integrity Control in Active Databases,** *C. B. Medeiros, M. J. Andrade*

07/92 **New Experimental Results For Bipartite Matching,** *J. C. Setubal*

08/92 **Maintaining Integrity Constraints across Versions in a Database,** *C. B. Medeiros, G. Jomier, W. Cellary*

09/92 **On Clique-Complete Graphs,** *C. L. Lucchesi, C. P. Mello, J. L. Szwarcfiter*

10/92 **Examples of Informal but Rigorous Correctness Proofs for Tree Traversing Algorithms,** *T. Kowaltowski*

11/92 **Debugging Aids for Statechart-Based Systems,** *V. G. S. Elias, H. Liesenberg*

12/92 **Browsing and Querying in Object-Oriented Databases,** *J. L. de Oliveira, R. de O. Anido*

Relatórios Técnicos – 1993

01/93 **Transforming Statecharts into Reactive Systems,** *Antonio G. Figueiredo Filho, Hans K. E. Liesenberg*

02/93 **The Hierarchical Ring Protocol: An Efficient Scheme for Reading Replicated Data,** *Nabor das C. Mendonça, Ricardo de O. Anido*

03/93 **Matching Algorithms for Bipartite Graphs,** *Herbert A. Baier Saip, Cláudio L. Lucchesi*

04/93 **A lexBFS Algorithm for Proper Interval Graph Recognition,** *Celina M. H. de Figueiredo, João Meidanis, Célia P. de Mello*

05/93 **Sistema Gerenciador de Processamento Cooperativo,** *Ivonne. M. Carrazana, Nelson. C. Machado, Célio. C. Guimarães*

06/93 **Implementação de um Banco de Dados Relacional Dotado de uma Interface Cooperativa,** *Nascif A. Abousalh Neto, Ariadne M. B. R. Carvalho*

07/93 **Estadogramas no Desenvolvimento de Interfaces,** *Fábio N. de Lucena, Hans K. E. Liesenberg*

08/93 **Introspection and Projection in Reasoning about Other Agents,** *Jacques Wainer*

09/93 **Codificação de Seqüências de Imagens com Quantização Vetorial,** *Carlos Antonio Reinaldo Costa, Paulo Lício de Geus*

10/93 **Minimização do Consumo de Energia em um Sistema para Aquisição de Dados Controlado por Microcomputador,** *Paulo Cesar Centoducatte, Nelson Castro Machado*

11/93 **An Implementation Structure for RM-OSI/ISO Transaction Processing Application Contexts,** *Flávio Morais de Assis Silva, Edmundo Roberto Mauro Madeira*

12/93 **Boole's conditions of possible experience and reasoning under uncertainty,** *Pierre Hansen, Brigitte Jaumard, Marcus Poggi de Aragão*

13/93 **Modelling Geographic Information Systems using an Object Oriented Framework,** *Fatima Pires, Claudia Bauzer Medeiros, Ardemiris Barros Silva*

14/93 **Managing Time in Object-Oriented Databases,** *Lincoln M. Oliveira, Claudia Bauzer Medeiros*

15/93 **Using Extended Hierarchical Quorum Consensus to Control Replicated Data: from Traditional Voting to Logical Structures,** *Nabor das Chagas Mendonça, Ricardo de Oliveira Anido*

16/93 **$\mathcal{LL}$ – An Object Oriented Library Language Reference Manual,** *Tomasz Kowaltowski, Evandro Bacarin*

17/93 **Metodologias para Conversão de Esquemas em Sistemas de Bancos de Dados Heterogêneos,** *Ronaldo Lopes de Oliveira, Geovane Cayres Magalhães*

18/93 **Rule Application in GIS – a Case Study,** *Claudia Bauzer Medeiros, Geovane Cayres Magalhães*

19/93 **Modelamento, Simulação e Síntese com VHDL,** *Carlos Geraldo Krüger e Mário Lúcio Côrtes*

20/93 **Reflections on Using Statecharts to Capture Human-Computer Interface Behaviour,** *Fábio Nogueira de Lucena e Hans Liesenberg*

21/93 **Applications of Finite Automata in Debugging Natural Language Vocabularies,** *Tomasz Kowaltowski, Cláudio Leonardo Lucchesi e Jorge Stolfi*

22/93 **Minimization of Binary Automata,** *Tomasz Kowaltowski, Cláudio Leonardo Lucchesi e Jorge Stolfi*

Departamento de Ciência da Computação — IMECC
Caixa Postal 6065
Universidade Estadual de Campinas
13081-970 – Campinas – SP
BRASIL

reltec@dcc.unicamp.br