UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO

(Bacharelado)

ESTUDO DO PROCESSO DE DESENVOLVIMENTO EM PALM USANDO C++ NA PLATAFORMA LINUX

TRABALHO DE CONCLUSÃO DE CURSO SUBMETIDO À UNIVERSIDADE REGIONAL DE BLUMENAU PARA A OBTENÇÃO DOS CRÉDITOS NA DISCIPLINA COM NOME EQUIVALENTE NO CURSO DE CIÊNCIAS DA COMPUTAÇÃO — BACHARELADO

EDUARDO HILDEBRANDT

BLUMENAU, JUNHO/2003

2003/1-19

ESTUDO DO PROCESSO DE DESENVOLVIMENTO EM PALM USANDO C++ NA PLATAFORMA LINUX

EDUARDO HILDEBRANDT

ESTE TRABALHO DE CONCLUSÃO DE CURSO, FOI JULGADO ADEQUADO PARA OBTENÇÃO DOS CRÉDITOS NA DISCIPLINA DE TRABALHO DE CONCLUSÃO DE CURSO OBRIGATÓRIA PARA OBTENÇÃO DO TÍTULO DE:

BACHAREL EM CIÊNCIAS DA COMPUTAÇÃO

_____

Prof. Dr. Paulo César Rodacki Gomes — Orientador na FURB

_____

Prof. José Roque Voltolini da Silva — Coordenador do TCC

BANCA EXAMINADORA

_____

Prof. Dr. Paulo César Rodacki Gomes

_____

Prof. Paulo de Tarso Mendes Luna

_____

Prof. Antonio Carlos Tavares

RESUMO

Este trabalho apresenta um estudo sobre o processo de desenvolvimento de aplicações para a plataforma Palm, utilizando a linguagem C++ com o compilador prc-tools na plataforma Linux. Para a validação do estudo, foi desenvolvido um protótipo simples de editor gráfico 2D que realiza operações simples tais como desenhar e apagar polígonos não-convexos utilizando a tela do Palm como principal elemento de interface gráfica interativa com o usuário. Para seleção de polígonos, foi implementado um algoritmo clássico da Geometria Computacional.

ABSTRACT

This work presents a study about the stages of the development process of Palm applications in C++ language using the prc-tools compiler in Linux. In order to validate the study, a simple software prototype of a 2D graphics editor was implemented. The refered graphic editor has simple features such as non-convex polygon drawing and deletion using the Palm screen as the main graphical interactive user interface. The polygon selection implements a classical Computational Geometry algorithm.

LISTA DE FIGURAS

LISTA DE QUADROS

SUMÁRIO

1 INTRODUÇÃO

Desde que a Palm Inc., em 1996, lançou seu primeiro computador de mão chamado “palmtop”, visando a visualização de informações em qualquer local, em qualquer momento, novos recursos vem sendo incorporados a este tipo de computador. Os primeiros computadores eram monocromáticos, tinham pouca quantidade de memória e muitas limitações, hoje já estão disponíveis computadores coloridos com uma quantidade maior de memória, uma capacidade maior de processamento, e mais recursos. Recursos estes que devem ser explorados, a fim de conseguir extrair o máximo com uma aplicação de um palmtop (Palm, 2003).

O presente trabalho apresenta um estudo do processo de desenvolvimento de aplicações em C++ para a plataforma Palm, com ênfase nos recursos gráficos do Palm. Visando explorar tais recursos, este trabalho também apresenta o desenvolvimento de um editor gráfico que permita desenhar polígonos e depois selecioná-los.

A seleção de polígonos não-convexos é um problema comum em muitas aplicações gráficas. Este problema não é solucionado apenas testando se um ponto está depois do lado direito e antes do lado esquerdo, abaixo do topo e acima da parte de baixo do polígono. Conforme Figueiredo (1993) para solucionar este problema existe um algoritmo relativamente simples, que consiste em traçar uma semi-reta partindo do ponto que se quer testar para um dos lados e contar a quantidade de intersecções dos lados do polígono com esta semi-reta. Se esta quantidade for ímpar significa que o ponto é interior ao polígono, caso contrário, é exterior.

Toda a idéia de construção de um editor não traz resultados concretos se não for conhecida a forma de construção de um aplicativo para a plataforma. Para suprir esta necessidade, este trabalho apresenta uma investigação sobre o desenvolvimento de aplicativos para a plataforma Palm em linguagem *C++*, a linguagem nativa do sistema operacional. Além desse estudo, é apresentada a implementação de um protótipo de editor 2D de polígonos que utiliza recursos gráficos da plataforma.

1.1 OBJETIVOS DO TRABALHO

O objetivo principal deste trabalho de conclusão de curso é apresentar um estudo do processo completo de desenvolvimento de uma aplicação em C++ para computadores Palm.

Os objetivos específicos do trabalho são:

- Apresentar as principais características da plataforma Palm, no que diz respeito aos aspectos relativos ao desenvolvimento de aplicativos para a plataforma;
- Apresentar as ferramentas de desenvolvimento para Palm disponíveis na plataforma Linux;
- Desenvolver um protótipo de um editor gráfico 2D que permita desenhar polígonos;
- Implementar o algoritmo de ponto em polígono, para conseguir fazer seleção de polígonos desenhados pelo software.

Outros objetivos seriam demonstrar a forma de desenvolver um aplicativo para a plataforma Palm e explorar as capacidades gráficas da plataforma.

1.2 ESTRUTURA DO TRABALHO

O capítulo 2 apresenta uma introdução a geometria computacional, enfatizando o algoritmo de ponto em polígono. No capítulo 3 serão abordados detalhes sobre a plataforma Palm e a forma de desenvolvimento de um aplicativo para Palm. No capítulo 4 será apresentada a especificação e a implementação do protótipo de editor gráfico. Por fim, o capítulo 5 traz as considerações finais e sugestões de continuidade do trabalho.

2 GEOMETRIA COMPUTACIONAL

Segundo Freitas (2003), a geometria computacional emergiu da área de desenvolvimento e análise de algoritmos em meados dos anos 70 e constitui uma ferramenta fundamental em diversas áreas da computação que necessitam de uma abordagem geométrica, tais como Computação Gráfica, Robótica, Sistemas de Informações Geográficas, Visão Computacional, Otimização Combinatória, Processamento de Imagens, entre outras. O objetivo da Geometria Computacional é estudar problemas geométricos sob o ponto de vista algorítmico, focando o interesse em solucionar um problema utilizando o menor número possível de operações elementares de modo a trazer eficiência no cálculo da solução.

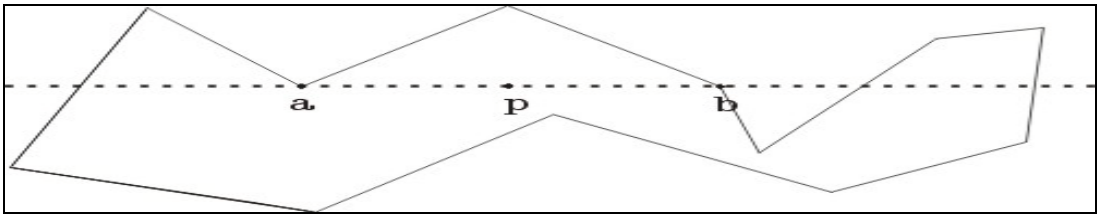
2.1 ALGORITMO DE PONTO EM POLÍGONO

Há programas gráficos que possuem a capacidade de interação com uma cena gráfica, utilizam este algoritmo para encontrar os objetos da cena. Como exemplificado na introdução, o teste de seleção é mais complexo do que apenas testar os lados do polígono para identificar a sua seleção ou não, o algoritmo de ponto em polígono e bastante simples e resolve o problema.

Um exemplo comum de aplicação deste algoritmo, é um programa que desenha mapas de cidades. O mapa geralmente é dividido em quadras, para o usuário selecionar estas quadras o programa pode aplicar o algoritmo de ponto em polígono.

Segundo Figueiredo (1993), este algoritmo define que para solucionar o problema de determinar se um ponto está dentro ou fora de um polígono, deve-se traçar uma semi-reta partindo do ponto que se quer testar para um dos lados, depois testar todas as intersecções desta semi-reta com os lados do polígono. Se o número de intersecções for ímpar, o ponto é interior ao polígono, caso contrário exterior.

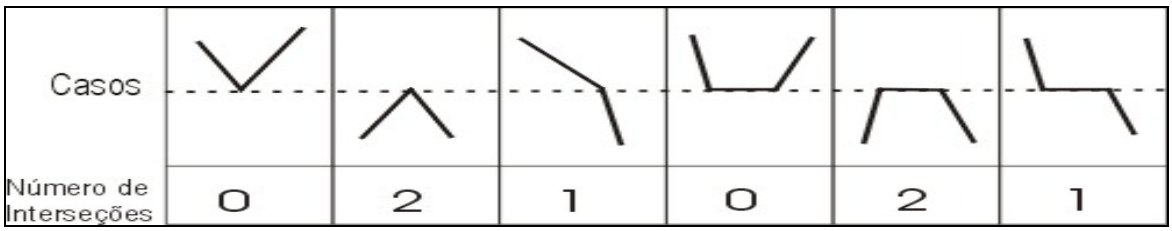
Figura 1 – EXEMPLO DE POLÍGONO



Contudo este algoritmo tem algumas particularidades que devem ser observadas, no caso da figura 1, por exemplo, o ponto "a*"* não deve ser contado como ponto de intersecção, enquanto "b" deve.

Segundo Figueiredo (1993), uma forma prática de solucionar os conflitos acima é a seguinte: a intersecção de um lado é contada somente se ela ocorrer em um ponto que não seja de ordenada mínima com a lado testado. A figura 2 mostra cada caso possível e o número de intersecções correspondentes.

Figura 2 - CASOS DE CRUZAMENTO E NÚMERO DE INTERSECÇÕES



Abaixo é apresentado o algoritmo de ponto em polígono. Observando-se este algoritmo percebe-se que ele nunca conta uma intersecção com um lado que está na horizontal e também não conta nenhuma intersecção na ordenada mínima, isto garante que todos os casos especiais apresentados na figura 2 sejam atendidos.

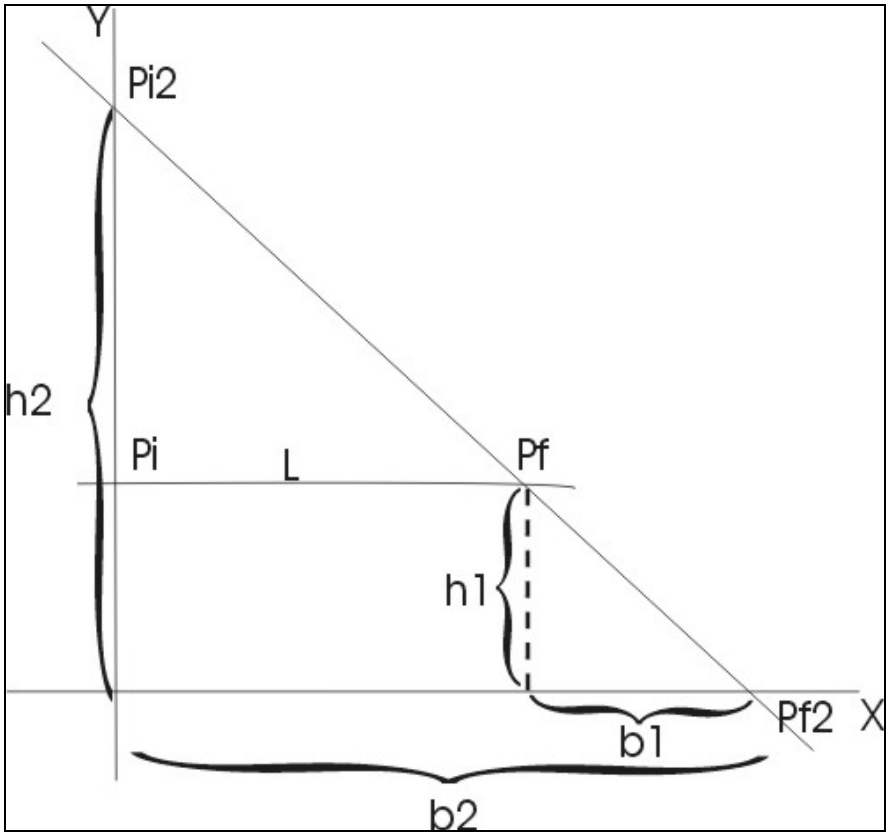
Quadro 1 - ALGORITMO DE PONTO EM POLIGONO

dados: $p = (x_0, y_0)$; $p_i = (x_i, y_i)$, $i = 1,..., n$; $p_{n+1} = p_1$.
inicialização: $N = 0$;
para $i = 1..n$
    se $y_i \neq y_{i+1}$ entao
        (x, y) = a intersecção de $p_i p_{i+1}$ com L
        se $x = x_0$ entao
            p_0 é da fronteira: PARE.
        senao, se $x > x_0$ e $y > \min(y_i, y_{i+1})$ entao
            $N = N + 1$.
    senão, se p pertence ao lado horizontal $p_i p_{i+1}$ entao
        p é da fronteira: PARE.
se N é impar, então p é interior; senão, p é exterior.

2.2 INTERSECÇÃO DE SEMI-RETAS, UTILIZANDO SEMELHANÇA DE TRIÂNGULOS

Para a aplicação do algoritmo de ponto em polígono é necessário um algoritmo auxiliar que retorne o ponto de intersecção de duas semi-retas e devido ao fato de uma das semi-retas ser uma semi-reta horizontal (paralela ao eixo das abscissas), o cálculo do ponto de interseção da reta com cada lado do polígono pôde ser simplificado e realizado através de semelhança de triângulos. Segundo Giovanni (1988), dois triângulos são semelhantes se, e somente se, têm dois ângulos respectivamente congruentes. No caso deste trabalho tendo uma semi-reta horizontal, a coordenada y desta reta já é conhecida, permitindo obter as medidas dos triângulos para calcular a coordenada x do ponto de intersecção. Observando-se a figura 3 temos um exemplo da aplicação de semelhança de polígonos.

Figura 3 - EXEMPLO DE INTERSECÇÃO



No quadro 2 são demonstradas as fórmulas para o cálculo das variáveis necessárias para se obter o valor da coordenada x do ponto de intersecção das duas retas. Sendo que só existirá intersecção se a coordenada y da semi-reta L ficar entre os valores das coordenadas y final e inicial da outra semi-reta, e os valores das coordenadas x inicial e final da semi-reta L abrangerem os valores das coordenadas x inicial e final da outra semi-reta, caso isto não seja verdade e se as duas semi-retas não forem paralelas o seu ponto de intersecção não pertencerá a semi-reta, o que indica que não há intersecção das semi-retas.

Quadro 2 – FORMULA PARA CALCULO DA INTERSECÇÃO

variáveis

h2 = Pf2.Y - Pi2.Y;

h1 = Pf2.Y - Pi.Y;

b2 = Pf2.X - Pi2.X;

b1 = (b2 * h1) / h2

x = x0 – b1

onde:

b1, b2 = base do triângulo

h1, h2 = altura do triângulo

x0 = coordena inicial da reta

x = coordena x do ponto de intersecção

y = será igual ao y da reta horizontal

2.3 TOLERÂNCIA NUMÉRICA PARA GEOMETRIA

Conforme Corney (1998), devido ao número limitado de dígitos que o computador oferece para representar um número, há a necessidade de determinadas aplicações gráficas determinarem um intervalo válido para um valor, sendo definido quantos dígitos serão significativos para a comparação. Sem um nível de tolerância a usabilidade do programa fica comprometida, pois fica muito custoso para o usuário informar o ponto com muita precisão, também o dispositivo que o usuário está utilizando para a seleção pode não fornecer um nível de precisão tão grande quanto o utilizado no sistema de coordenadas do *software* e assim, o máximo que o usuário consegue informar é um valor aproximado ao valor que ele deveria selecionar. Deve-se então determinar um nível de tolerância que possibilite, por exemplo, que o usuário informe o ponto 1,5 e estar selecionando o ponto 1,004.

Outro exemplo de controle de tolerância numérica para geometria seria tentar avaliar o ponto que o usuário deseja selecionar, como no caso em que o usuário tem que

ligar duas retas, então o programa já tentaria adivinhar qual seria o provável ponto onde o usuário quer fazer a ligação.

2.4 ESTRUTURA DE DADOS PARA COMPUTAÇÃO GRÁFICA VETORIAL

Segundo Foley (1990), as estruturadas de dados para a geometria computacional são um grupo lógico de primitivas, atributos e outras informações, que podem estar relacionadas a geometria (posição, orientação, tamanho), aparência (cor, estilo), como também pode ser informações do aplicativo que está utilizando estas estruturas. Um exemplo que utiliza esta estrutura é um programa de mapas, que armazenaria nas estruturas de dados as informações necessárias para poder apresentar os nomes das ruas e terrenos representados no mapa. Há ainda uma preocupação com a utilização de memória, e a atualização dos dados, a redundância de informações causaria um grande desperdício de memória e a atualização de dados teria um custo de processamento muito grande.

Para montar uma estrutura que seguisse estas preocupações, utilizou-se de uma lista contendo pontos 2D (x, y), e outra contento polígonos. Os polígonos contem uma lista que mantém referência a que pontos o polígono possui, formando assim seu vértices. Assim todos os pontos que são comuns a diferentes polígonos estarão armazenados na memória somente uma vez e todos os polígonos irão referenciar a esta estrutura.

3 PLATAFORMA PALM

Em 1996, a Palm Inc. lançou o primeiro computador de mão, chamado “palmtop”. Um computador de mão não é um *laptop* e nem um computador de mesa. O foco deste tipo de plataforma computacional está na administração e no acesso às informações, em vez da criação e edição de documentos. Os usuários dos computadores de mão podem carregá-los para todos os lados para acessar informações em qualquer momento e em qualquer local (Palm, 2003).

Os primeiros computadores de mão lançados eram monocromáticos, hoje já estão disponíveis computadores de mão coloridos, com grande capacidade de processamento e armazenamento e fácil sincronização com um computador de mesa comum.

Para a interação do usuário com um palmtop, é disponibilizada uma *“caneta”*, com ela se interage com os programas, também está disponível nos *palm* quatro botões de acesso rápido, mais dois para rolagem de informações.

3.1.1 CANETA

Utilizada para a interação do usuário com os palmtop, é o dispositivo indicador para seleção, escrita, manipulação de informações. Para a seleção o usuário toca a tela do *palm* no ponto que se quer selecionar e o sistema operacional do *palm* responde a esta ação. Para a escrita o usuário utiliza um conjunto pré-determinado de caracteres que o palmOS compreende e traduz para o alfabeto do idioma selecionado no *palm*.

3.1.2 BOTÕES DE ACESSO RÁPIDO

São atalhos para os programas mais utilizados em um palmtop. Em nível de programa é possível alterar a ação deste programas, mas por padrão as suas ações são:

- Botão agenda, inicia a agenda padrão do sistema operacional palmOS;
- Botão telefone, inicia uma programa de lista de telefones;
- Botão tarefa, inicia um programa para registrar tarefas;
- Botão nota, inicia um programa para fazer anotações;

- Botões de rolagem, fazer a rolagem das informações vistas na tela.

3.2 RECURSOS / LIMITAÇÕES

Muitas empresas disponibilizam modelos de acessórios para palmtop, e cada um possui alguns recursos que são comuns a todos os modelos e outros que são somente de alguns modelos, dentre isso, pode-se citar modelos que vêm com antenas para comunicação via rádio, câmera digitais, dentre outros. Mas por ser um computador portátil, os modelos são pequenos e alimentados por bateria, possuem um pequeno visor e pouca memória disponível em comparação com um computador de mesa.

Devido a estas características, algumas considerações devem ser feitas no desenvolvimento de uma aplicação para um computador palmtop. A aplicação precisa ser pequena, com uma interface bem simples que possibilite acionar os recursos da mesma sem muita interação. Uma interface mais elaborada utilizaria mais espaço da tela que não é muito grande isto comprometeria o objetivo do aplicativo que seria mostrar informação e não a interface. Também há o compromisso com a quantidade de memória disponível e a utilização de processamento, pois existe um tempo de utilização imposto pela bateria.

Hoje os palmtops estão limitados a 16 megabytes de memória, mas já está sendo desenvolvido o suporte para 128 megabytes e processadores de 400 Mhz (PalmSource, 2003), o que permitirá aplicações mais robustas, mas não tirando a preocupação com as antigas limitações.

Muitos modelos de palmtop disponibilizam um soquete para encaixe de cartões de expansão, estes aumentam a quantidade de memória ou disponibilizam outro recurso para o palmtop, como por exemplo uma câmera digital e o modem.

3.2.1 RECURSOS GRÁFICOS

Hoje há modelos de palmtop que possuem uma tela gráfica de 160x160 *pixels*, e suportam 65 milhões de cores, alguns modelos mais novos estão disponibilizando um tela de 320x320 *pixels* (Palm, 2003).

3.3 PALM-OS

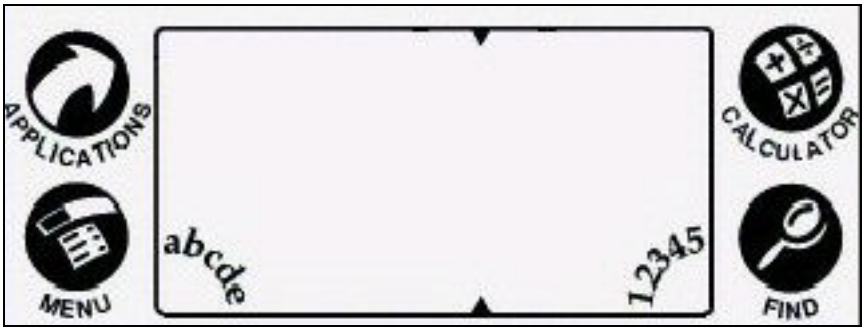
PalmOS é o sistema operacional dos computadores de mão palmtop. Segundo Foster (2002), o palmOS é multi-tarefa, mas a interface do palmOS permite executar somente um aplicativo por vez, isto não impede que o programa cria mais de um processo. O sistema operacional palmOS, oferece uma interface simples, sendo composta por uma tela principal onde estão os programas e pode ser visualizada na figura 4, e uma parte fixa, onde estão os botões de comando, e a área de escrita do palmOS, que pode ser visualizada na figura 5.

3.3.1 GERENCIAMENTO DE MEMÓRIA

Conforme Foster (2002), no palmOS existe somente uma memória RAM física que o sistema operacional divide em duas unidades lógicas, a memória para programa, que é a memória RAM dinâmica e a memória RAM para armazenamento, ambas são voláteis, ou seja se a fonte de energia do computador *palm* for cancelada, todas as informações que estão na memória serão perdidas, incluindo programas instalados. A memória para programa tem a mesma finalidade que uma memória RAM de um computador de mesa, é a parte onde são declaradas as variáveis temporárias e variáveis globais e ainda qualquer dado que não precisa ser salvo. Já a memória de armazenamento funciona como a memória secundária de um computador de mesa, nela são salvos os programas e os dados permanentes. Para armazenar os dados, o palmOS disponibiliza um banco de dados onde cada aplicação cria os seus registros. A maioria dos computadores *palm* possui a capacidade de adicionar um cartão que expande a capacidade de memória do computador. Para o cartão valem as mesmas regras da memória nativa do computador, esta memória é dividida em duas partes lógicas, uma para RAM dinâmica outra para armazenamento e também é volátil.

Figura 4 - TELA PRINCIPAL DO PALMOS

Figura 5 - PARTE FIXA



O ícone *applications*, mostra a lista de programas, o ícone *calculator* inicia uma calculadora, o ícone *menu*, chama o menu da aplicação corrente, o ícone *find*, chama uma caixa de diálogo de procura. A moldura central é a área de escrita do palmOS chamada de *graffiti*, ela é dividida em duas partes, uma reconhece a escrita de letras e a outra reconhece a escrita de números.

A Application Program Interface (API) do palmOS disponibiliza um conjunto de funções para a construção de aplicações gráficas, todas as funções gráficas trabalham em relação a uma janela ativa.

3.3.2 FUNÇÕES PARA PRIMITIVAS GRÁFICAS

Na API do palmOS pode-se encontrar um conjunto de funções que permite a criação de pontos, linhas, retângulos ou caracteres. Com este conjunto básico de funções é possível disponibilizar inúmeras funcionalidades a um programa, como desenhar figuras geométricas, desenhar mapas, dentre outras aplicações. Com o tratamento dos eventos que o palmOS gera é possível saber, por exemplo se o usuário arrastou a caneta pelo *palm*, o que nos possibilitaria construir uma rotina que se movimenta um objeto da tela, o palmOS gera um evento repetitivo que vai indicando o deslocamento da caneta pela tela do *palm*, com este deslocamento é possível calcular o quanto deve ser movimentado o objeto gráfico. Agregando os controles visuais que o palmOS disponibiliza, o programa pode ter à sua disposição praticamente todos os recursos necessários para construção de aplicativos para inúmeras áreas de conhecimento. A lista de controles visuais que o palmOS suporta e o seu funcionamento pode ser encontrado em Foster (2002) e em Palmsource (2003).

Existem muitas outras funções na API gráfica, mas estas se destinam a mudanças na forma de funcionamento das funções básicas de desenho. O detalhamento completo das funções da API do palmOS, assim como dos eventos que o palmOS gera, podem ser encontrado na *Software Development Kit* (SDK) do palm e pode ser obtido em Palmsource (2003). A seguir, são listadas as principais funções da biblioteca gráfica do palmOS.

3.3.2.1 WINDRAWPIXEL

Recebe como parâmetro dois valores que seria a coordenada x e y da tela onde o ponto deve ser desenhado.

3.3.2.2 WINDRAWLINE

Recebe como parâmetro quatro valores que seriam os pontos inicial e final da tela onde deve ser desenhada uma linha.

3.3.2.3 WINDRAWRECTANGLE

Recebe uma estrutura que indica as coordenadas de onde deve ser desenhado um retângulo. A estrutura que é passada por parâmetro é composta por duas coordenadas que indicam o valor do canto superior direito, e do canto inferior esquerdo, com estas duas coordenadas é possível calcular todos os cantos do retângulo.

3.3.2.4 WINDRAWCHARS

Recebe como parâmetro um conjunto de caracteres para desenhar, a quantidade de caracteres e as coordenadas x e y de onde os caracteres devem ser desenhados.

3.3.2.5 WINRGBTOINDEX

Esta função é utiliza para mapear uma cor no formato RGB para o índice interno da tabela de cores do palmOS. Para este procedimento é necessário criar uma estrutura que possui três valores de 8 bits para a formarão o código RGB, cada um destes valores corresponde a uma cor base que forma a cor final, um para vermelho, outro para verde e o último para azul, ambos podem aceitar valores 0 a 255, o que indica a quantidade de vermelho, verde e azul que terá na composição da cor RGB. Esta função é utilizada para montar a cor que será utiliza pelas funções que definem as cores de desenho do palmOS.

3.3.2.6 WINSETBACKCOLOR

Define qual a cor de fundo que será utilizada pelas funções de desenho, esta função recebe como parâmetro o valor retornado pela função WinRGBToIndex.

3.3.2.7 WINSETFORECOLOR

Define qual a cor de primeiro plano que será utilizada pelas funções de desenho, esta função recebe como parâmetro o valor retornado pela função WinRGBToIndex.

3.3.2.8 WINSETTEXTCOLOR

Define qual a cor do texto será utilizada, esta função recebe como parâmetro o valor retornado pela função WinRGBToIndex.

3.3.2.9 OUTRAS FUNÇÕES

Existe ainda um número muito grande de funções na API, tanto para a parte gráficas, quanto para a manipulação de informações, para um melhor esclarecimento sobre elas dever ser consultado Palmsource (2003).

3.4 FORMA DE DESENVOLVIMENTO DE PROGRAMA PALM

Nesta seção serão descritos os procedimentos necessários para desenvolver um programa para a plataforma Palm. Assim como o que um programa palm necessita ter para permitir interação do usuário com o mesmo.

3.4.1 LINGUAGENS E AMBIENTES DISPONÍVEIS:

Hoje há algumas linguagens disponíveis para o desenvolvimento de aplicações palm, mas como o *C* é a linguagem nativa do sistema operacional será dada mais ênfase a esta linguagem. Devido à exploração de alguns recursos gráficos da plataforma, a adoção de outra linguagem que não seja a linguagem nativa do sistema operacional poderia ter dificultado a implementação deste trabalho, visto que algumas funcionalidades da API do palmOS poderiam não estar implementada ainda na linguagem.

Algumas linguagens e seus respectivos ambientes:

a) C/C++
   - CodeWarrior, Prc-Tools, Falch.net,VFDIDE, etc.

b) Basic Like
   - Satellite forms, Pendragon forms, NS Basic, etc.

c) Java
   - IBM VisualAge Micro Edition

d) Outras
   - Smalltalk, Lisp, Assembly, etc.

Como apresentado acima temos algumas opções de linguagens para programação em palmOS, mas a linguagem nativa do sistema operacional é *C*, está foi escolhida para o desenvolvimento do presente trabalho.

Um programa escrito na linguagem *C*, deve incluir o cabeçalho "PalmOS.h" depois deve implementar uma função *Pilotmain* que recebe como parâmetros um código de inicialização, um ponteiro e um valor inteiro com informações adicionais passadas pelo sistema operacional. O programa para sua execução normal deve iniciar o funcionamento quando receber o código *sysAppLaunchCmdNormalLaunch*, outros códigos de inicialização podem ser utilizados pelo palmOS, e cada um tem a sua finalidade, a tabela 1 mostra alguns outros códigos de inicialização que o palmOS pode utilizar.

Tabela 1- CÓDIGOS DE INICIALIZAÇÃO DO PALMOS

Código de inicialização	Descrição
sysAppLaunchCmdAddRecord	Sinaliza que outra aplicação quer adicionar um registro a base de dados do aplicativo
sysAppLaunchCodeDisplayAlarm	Indica para a aplicação apresentar um diálogo de alerta ou outra diálogo
sysAppLaunchCmdFind	Indica para aplicação procurar um texto especificado na sua base de dados
sysAppLaunchCmdGoto	Indica para a aplicação ir a um determinado registro da sua base de dados
sysAppLaunchCmdNormalLaunch	Inicialização normal da aplicação
sysAppLaunchCmdSystemReset	Permite a aplicação responder a um reinicialização do sistema.

Segundo Foster (2002), o PalmOS suporta outros códigos de inicialização, mas se este tipo de inicialização não é necessário para uma determinada aplicação, então

simplesmente neste caso não há necessidade de se efetuar qualquer tratamento para este código de inicialização.

3.5 EXEMPLO GENÉRICO DE PROGRAMA PALM

Um programa palm não diferencia muito de um programa para a plataforma PC, o sistema operacional comunica-se com o programa através de eventos que devem ser recebidos e interpretados pelo programa. No caso da linguagens *C*, tem que ser implementada uma função principal, chamada de *Pilotmain*, que será a primeira função a ser executada quando o programa inicia. Nesta função deve ser feita a inicialização do programa e implementação do tratador de eventos, que irá receber as mensagens passadas para o programa e chamar a função específica que trata aquele determinado evento.

Como pode ser observado no quadro 3, temos um exemplo de programa para palm que apresenta a mensagem "Hello Word!!!" e desenha uma linha do canto superior esquerdo da tela até o canto inferior direito. Neste exemplo somente é realizado um teste para a inicialização normal de um programa palm, não há nenhum tratamento para os outros códigos de inicialização do palmOS, visto que os mesmos não têm nenhuma influência no programa em questão.

Quadro 3 - PROGRAMA EXEMPLO PARA PALM

```
/*
 Programa teste n 1
  Objetivo: descobrir o funcionamento básico de um programa na plataforma Palm OS
 Eduardo Hildebrandt
 10 mar 2003
*/
#include "PalmTypes.h"
#include "PalmOS.h"
/* funcao principal de um programa palm */
UInt32
PilotMain (UInt16 cmd, void *pbp, UInt16 flags) {
 if (cmd == sysAppLaunchCmdNormalLaunch) {
   EventType e;
   WinDrawChars("Hello Word!!!", 13/*quatidade de caracteres a ser escrita*/,
                                    100/*posicao x a ser escrita*/,
                                    00/*posicao y a ser escrita*/);
   /*desenhar uma linha do inicio da tela ate o final*/
   WinDrawLine(0, 0, 320, 320);
   /* tratador de eventos */
   do {
     EvtGetEvent (&e, evtWaitForever);
     SysHandleEvent (&e);
    } while (e.eType != appStopEvent);
   }
 return 0;
 }
```

Um outro exemplo de programa palm é um programa que possua formulários. Para construção de formulário e elementos de interface deve ser criado um arquivo de recurso que possui a definição das interfaces, este arquivo de recursos será compilado com o compilador *pilrc* e os arquivos gerados serão compilador pelo compilador *build-prc*, juntamente com o arquivo gerado pelo compilador *gcc,* para gerar o executável final que irá executar no *palm*. O capítulo 4, a seguir, aborda em maiores detalhes os compiladores *pilrc, gcc e build-prc*. A API do palmOS disponibiliza funções para a manipulação das interfaces, funções para construções das primitivas gráficas e funções para manipulação de registros. Existem diferentes formas de mostrar uma janela nas aplicações para palmOS, mas a que foi utilizada no protótipo segue os seguintes passos:

- na inicialização do programa, é chamada a função “FrmGotoForm(MainForm);”;

- para que as ações do usuário incidam sobre o programa, deve ser implementado o tratador de eventos. No caso da apresentação de formulários, o tratador de eventos precisa interpretar os eventos de carga e apresentação dos formulários. Este tratador é executado para todas as apresentações de formulários do aplicativo, portanto dever ser identificado qual o formulário está sendo apresentado e devem ser realizadas as ações pertinentes àquele formulário.

Quadro 4 - TRATADOR DE EVENTOS DO APLICATIVO

```
static Boolean AppHandleEvent(EventType *event){
        FormType *form;
        UInt16 formID;
        Boolean handled = false;
        switch (event->eType){
                case frmLoadEvent:
                        formID = event->data.frmLoad.formID;
                        form = FrmInitForm(formID);
                        FrmSetActiveForm(form);
                        switch (formID){
                                case MainForm:
                                        FrmSetEventHandler(form, DrawFormHandleEvent);
                                        break;
                                default:
                                        break;
                        }
                        handled = true;
                        break;
                case frmOpenEvent:
                        form = FrmGetActiveForm();
                        FrmDrawForm(form);
                        WinDrawRectangleFrame(rectangleFrame, &gRect);
                        handled = true;
                        break;
                default:
                        break;
        }
        return (handled);
}
```

Neste tratador, apresentado no quadro 4, duas opções são tratadas, a leitura da tela e a apresentação da tela. Na leitura da tela, que seria o “case frmLoadEvent”, carrega-se o formulário para a memória, define-se que ele será o formulário ativo e define-se qual será a função que irá interpretar os eventos gerados por este formulário. Na apresentação do formulário desenha-se a tela através da função “FrmDrawForm(form)”. A chamada da função “WinDrawRectangleFrame...” tem por finalidade desenhar um retângulo na janela

ativa. Como na implementação deste trabalho foi utiliza apenas uma janela, não há a necessidade de se fazer um tratamento a respeito da janela de destino dos comandos de desenho.

3.5.1 TRATADOR DE EVENTOS DO FORMULÁRIO

O tratador de eventos do aplicativo consiste na parte do programa que irá executar o código para cada ação realizada em um formulário. Se esta parte de código não for implementada, todas as ações que o usuário fizer no software não serão executadas, ou seja, se não for colocado um tratamento para o evento de seleção de um determinado botão, a seleção do mesmo não irá promover nenhuma alteração no aplicativo. No quadro 5, pode ser observado um exemplo de tratador de eventos do formulário, no qual são testados eventos gerados por botões e pela interação do usuário na tela com a caneta.

Quadro 5 - TRATADOR DE EVENTOS DO FORMULÁRIO

```
static Boolean DrawFormHandleEvent(EventType *event){
        Boolean handled = false;
    switch (event->eType){
                case ctlSelectEvent:
                        switch (event->data.ctlSelect.controlID){
                                case BTN_NEW:
                                        break;
                                case BTN_CLOSE:
                                        break;
                                case BTN_INFO:
                                        break;
                                case BTN_DELETE:
                                        break;
                                default:
                                        break;
                        }
                        break;
                case penDownEvent:
                        break;
                default:
                        break;
   }
   return (handled);
}
```

Esta parte do código identifica cada ação realizada na janela ativa e executa o código correspondente. Por exemplo, o case BTN_NEW corresponde à ação na qual o

usuário seleciona o botão New da interface do protótipo. Já o penDownEvent corresponde a uma ação do usuário com a "caneta". Muitos outros tipos de eventos podem ser gerados, como, por exemplo, o movimento da caneta pela tela do *palm*, ações que indicam o estado do computador (se ele está sendo desligado, se está terminando a bateria, etc). Nota-se que apesar de existirem muitas possibilidades de intaração, é interessante que no tratador de eventos trate somente aquilo que for relevante ao aplicativo, todos os demais eventos podem ser ignorados.

3.6 EMULADOR

Para a execução do protótipo foi utilizado um programa que emula um computador palmtop, ao qual se dá o nome de Emulador. Para o funcionamento do programa é necessário ter a imagem do sistema operacional que é instalado no palm, a esta imagem dá se o nome de ROM e pode ser obtida em Palm (2003). O emulador oferece quase todos os recursos existentes em um palmtop, e oferece um outro recurso que é a possibilidade de depurar os programas. A seção .4. Desenvolvimento do Protótipo apresenta em maiores detalhes o funcionamento do emulador juntamente com o protótipo desenvolvido neste trabalho de conclusão de curso.

4 DESENVOLVIMENTO DO PROTÓTIPO

Após um estudo das capacidades gráficas da plataforma palmtop e uma ferramenta de desenvolvimento para a mesma o presente trabalho resultou na criação de um protótipo de um editor gráfico 2D para palmtop.

4.1 REQUISITOS PRINCIPAIS DO PROTÓTIPO

O objetivo deste trabalho é desenvolver um protótipo de editor gráfico 2D para palm, sendo um protótipo de editor gráfico, ele deve permitir desenhar polígonos e depois através da aplicação do algoritmo de ponto em polígono, deve permitir a seleção destes polígonos. Tendo os polígonos selecionados pode-se removê-los do editor.

4.2 ESPECIFICAÇÃO

O protótipo desenvolvido neste trabalho visa explorar as capacidades gráficas da plataforma palm. Então para ilustrar a funcionalidade do editor foram especificados quatro casos de uso. No caso de uso chamado “Criar polígono”, o usuário que desenhar um polígono no editor, para isso ele informa os pontos que farão parte do polígono. No caso “Selecionar polígono”, o usuário deseja selecionar um polígono, para isso ele utiliza a caneta e indica um ponto na tela. Em “Remover todos”, o usuário deseja remover todos os polígonos que existem desenhados no editor. E por fim, em “Remover Selecionado”, o usuário deseja remover todos os polígonos selecionados.

A especificação deste protótipo foi baseada na técnica de orientação a objetos *Unified Modeling Language* (UML) com base em Furlan (1998), utilizando a ferramenta *Rational Rose* com base em Quatrani (2001). Este ferramenta foi escolhida por ser uma ferramenta que dá suporte a todos os diagramas UML utilizados nesta especificação, e por ter sido abordado durante o período acadêmico. Os diagramas utilizados foram: diagrama de classes, diagrama de casos de uso ou *“use cases”* e diagrama de seqüência.

4.2.1 DIAGRAMA DE CLASSES

Para a implementação do protótipo foram modeladas as classes: *Point, Poligon, Poligons*.

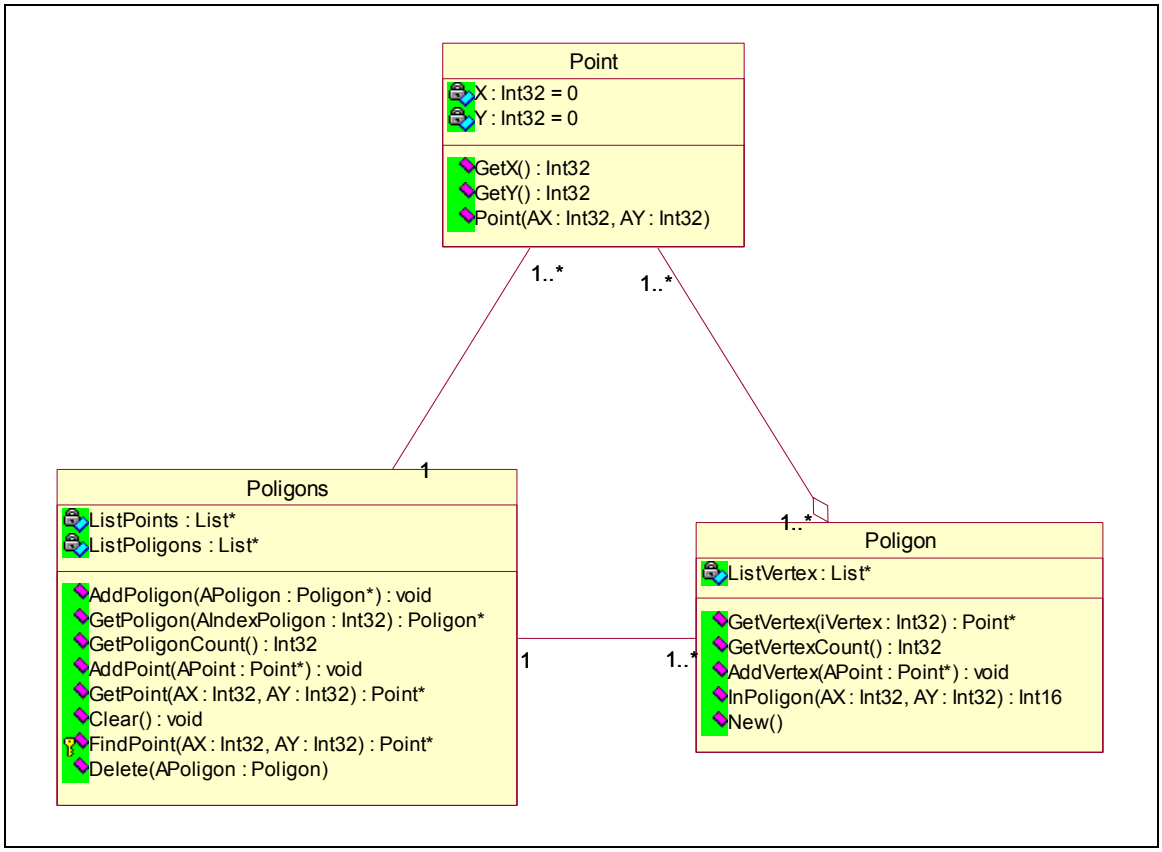
A classe *Point* é a responsável por armazenar as coordenadas x e y de cada vértice dos polígonos desenhados, e possui três métodos simples, um construtor que recebe os valores de x e y inicializando o objeto com os mesmos. O método *GetX* retorna o valor de x associado ao ponto. O método *GetY* retorna o valor de y associado ao ponto.

A classe *Poligon* é a responsável por armazenar as informações de cada polígono desenhado dentro do editor. Os seus métodos são destinados a adicionar vértices ao polígono e ao teste de inclusão ou exclusão de um ponto no polígono. O método *GetVertex* retorna o vértice indicado pelo valor de *iVertex*, já o método *GetVertexCount* retorna a quantidade de vértices que o polígono possui, o método *AddVertex* destina-se a adicionar mais um vértice ao polígono. O método *New* serve para iniciar a criação de um novo polígono. Finalmente o método *InPoligon* é a implementação do algoritmo de ponto em polígono, ele é responsável por determinar se um ponto é interior ou exterior ao polígono.

A classe *Poligons* destina-se a controlar todos os polígonos e pontos criados no editor. Para isso ela mantém uma lista com os pontos e outra lista com os polígonos. Antes de cada novo ponto ser desenhado, um objeto desta classe é consultado para verificar se o ponto em questão já existe na estrutura de dados, se existir a classe retorna a referência a este ponto, caso contrário cria um novo ponto e retorna a sua referência. O método *AddPoligon* adiciona um polígono que foi criado no editor, já o método *GetPoligon* retorna um polígono e o método *GetPoligonCount* retorna a quantidade de polígonos criados no editor. O método *GetPoint* chama o método *FindPoint*, que se destina a procurar na lista de pontos se o ponto já existe e retornar a referência ao ponto, se o ponto existir o método *GetPoint* retorna o sua referência, caso contrário é criado um objeto *Point*, passado para o método *AddPoint* que adiciona o ponto lista de ponto e depois retorna a sua referência. Independentemente de o ponto existir ou não, o método *GetPoint* sempre retorna uma referência a um ponto. O método *Clear* destina-se a

remover todos os polígonos desenhados no editor. O método *Delete* destina-se a remover somente um polígono do editor.

Figura 6 – DIAGRAMA DE CLASSES



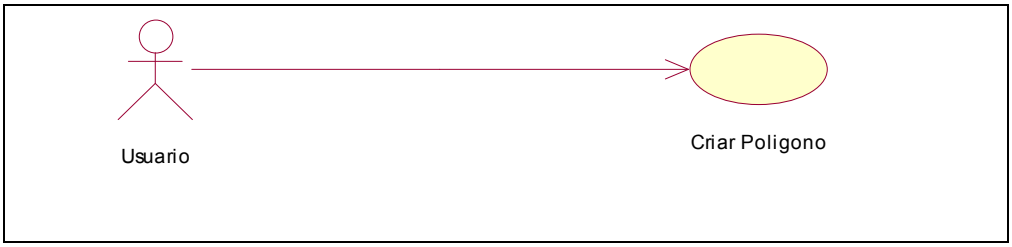
4.2.2 CASOS DE USO

Nesta seção será descrito conceitualmente o que o usuário poderá fazer com o protótipo. Para esta descrição utilizou-se do diagrama de casos de uso da *UML*.

4.2.2.1 CRIAR POLÍGONO

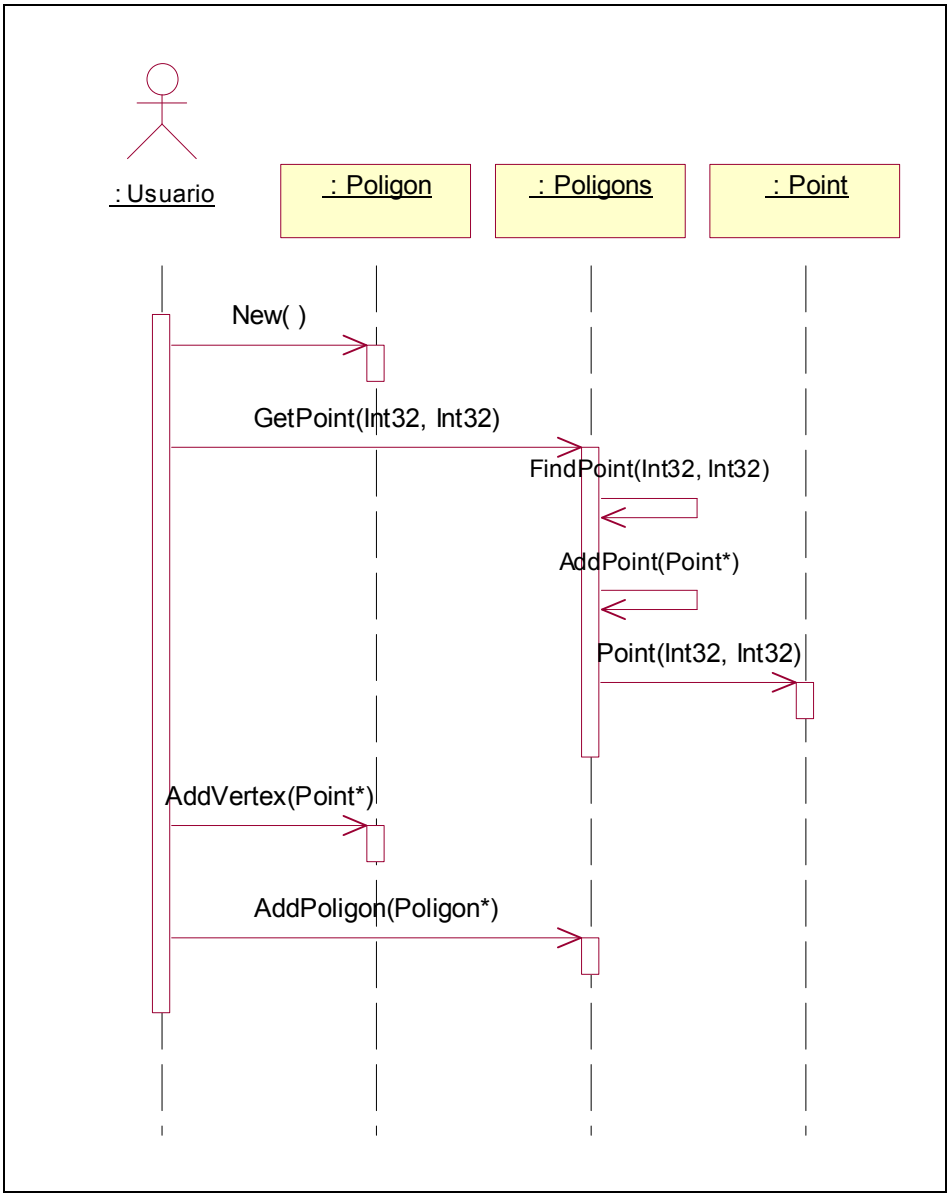
Este é o processo de criação de um polígono no editor, consiste na definição dos pontos que formarão o polígono. A figura 7 apresenta o diagrama de caso de uso, na qual o usuário cria um polígono no editor.

Figura 7 – CASO DE USO "CRIAR POLÍGONO"



O processo pode ser melhor compreendido observando-se o diagrama de seqüência da figura 8. Sempre que for necessário criar um polígono, será instanciado um objeto da classe *poligon,* depois será informado os pontos que este polígono possui. Para cada ponto informado, será consultado um objeto da classe *Poligons* o qual possui a lista de todos os pontos utilizados no editor. Se o ponto não foi utilizado ainda, será criado um objeto da classe *Point* e retornado a referência deste ponto para compor o polígono.

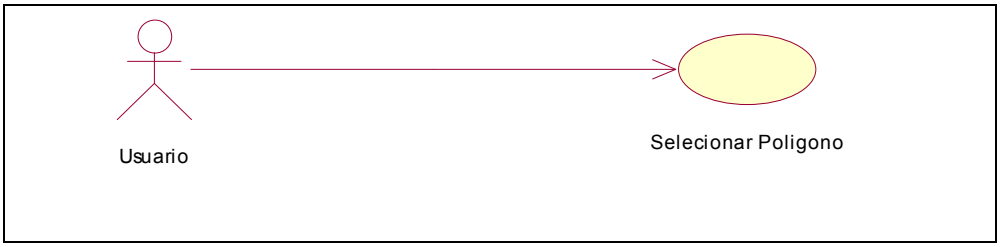
Figura 8 – DIAGRAMA DE SEQUÊNCIA "CRIAR POLIGONO



4.2.2.2 SELECIONAR POLÍGONO

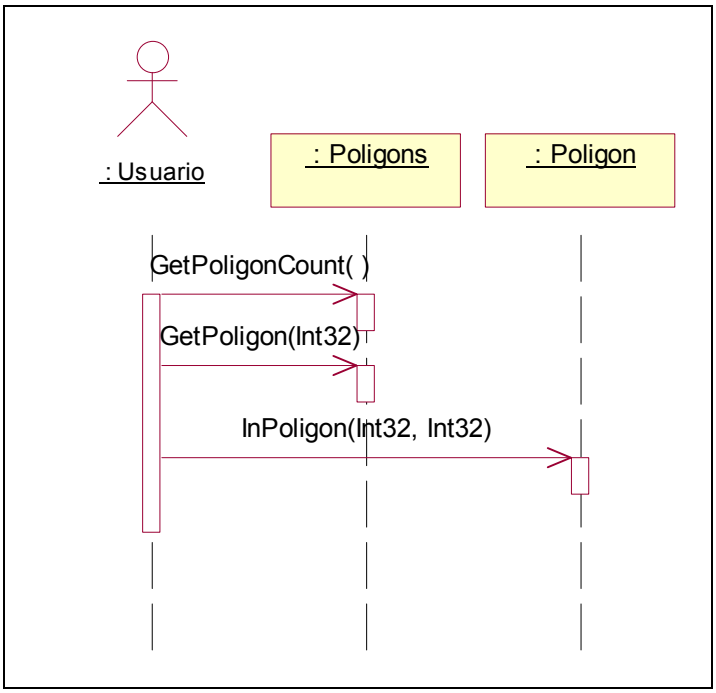
Este é o processo de seleção de polígonos no editor, neste processo é que se aplica o algoritmo de ponto em polígono. No caso, será informado um ponto, depois será solicitado ao objeto da classe *Poligons* a lista de polígonos que foram desenhadas no editor. Para cada polígono será perguntado se o ponto informado está dentro ou fora do polígono, caso esteja dentro este polígono será selecionado. A figura 9 apresenta o caso do uso no qual o usuário seleciona polígono.

Figura 9 – CASO DE USO "SELECIONAR POLÍGONO"



O processo pode ser melhor compreendido visualizando o diagrama de seqüência para a da figura 10.

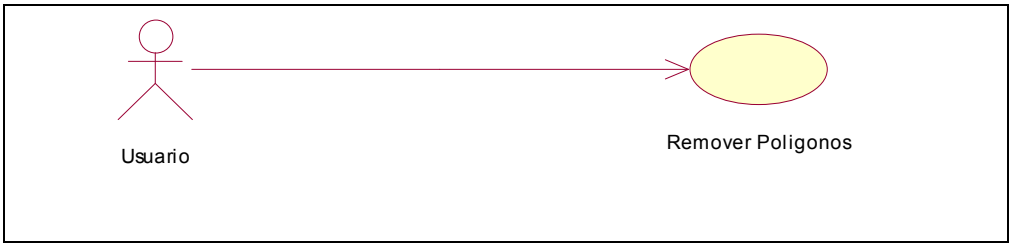
Figura 10 – DIAGRAMA DE SEQUENCIA "SELECIONAR POLIGONO"



4.2.2.3 CASO DE USO "REMOVER POLIGONOS"

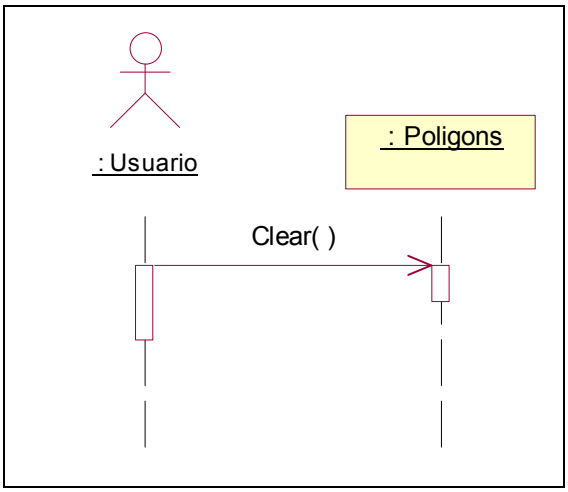
Este é o processo em que se deseja remover todos os polígonos desenhados no editor. A figura 11 apresenta o caso de uso no qual o usuário remove todos os polígonos.

Figura 11 – CASO DE USO "REMOVER POLÍGONOS"



O processo consiste em solicitar ao objeto da classe *Poligons* para remover todos os polígonos desenhados, e apagá-los da tela do editor, este processo pode ser melhor visualizado observando-se o diagrama de seqüência da figura 12.

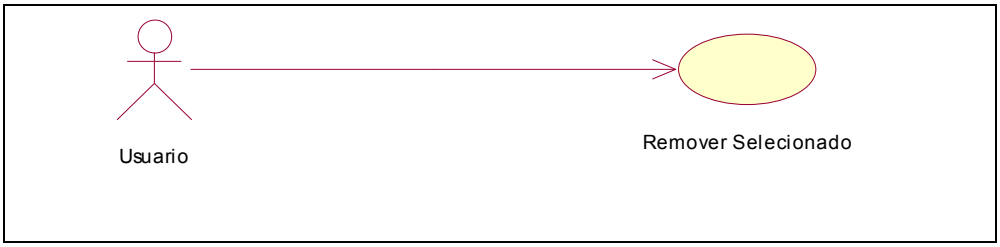
Figura 12 – DIAGRAMA DE SEQUENCIA "REMOVER POLIGONOS"



4.2.2.4 CASO DE USO "REMOVER SELECIONADO"

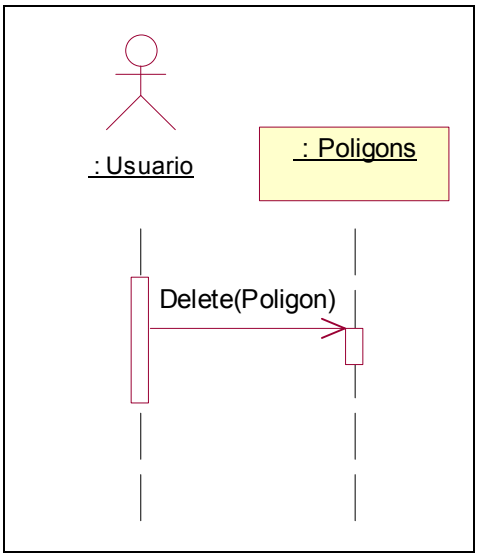
Este processo consiste em remover somente os polígonos que o usuário selecionou, para isso será informado ao objeto da classe *Poligons* quais são os polígonos selecionados, e os mesmos serão apagados da tela do editor. A figura 13 apresenta o caso de uso no qual o usuário remove os polígonos selecionados.

Figura 13 – CASO DE USO "REMOVER SELECIONADO"



O processo pode ser melhor compreendido observando-se o diagrama de seqüência da figura 14.

Figura 14 – DIAGRAMA DE SEQUÊNCIA "REMOVER SELECIONADO"



4.3 IMPLEMENTAÇÃO

Nesta seção será descrita a implementação do protótipo de editor gráfico, serão abordados detalhes mais técnicos da implementação, descrevendo detalhes das ferramentas utilizadas.

4.3.1 TÉCNICAS E FERRAMENTAS UTILIZADAS

Para a implementação do protótipo foi utilizada a técnica de programação orientada a objetos, utilizando o compilador *gcc* com base nas informações fornecidas por Foster (2002). Não foi utilizado nenhum ambiente específico para este compilador, os

códigos fontes foram editados no editor de texto *vi*, para depuração foi utilizado o depurador *gdb* com base nas informações de Foster (2002).

4.3.1.1 COMPILADOR GCC

O compilador gcc para palm consiste em um conjunto de programas que interpretam código C/C++ e geram código objeto.

4.3.1.2 COMPILADOR PILRC

Conforme Foster (2002), consiste em um compilador de recursos que transforma uma descrição textual de uma interface com o usuário e outros recursos em formato binário entendido pelo palmOS.

Quadro 6 - EXEMPLO DE DESCRIÇÃO DE UMA TELA

```
#include "resmain.h"
FORM ID MainForm AT (0 0 160 160)
USABLE
BEGIN
   BUTTON "Novo" ID BTN_NEW AT (0 1 AUTO AUTO)
   BUTTON "Term" ID BTN_CLOSE AT (PREVRIGHT + 5 1 AUTO AUTO)
   BUTTON "Delete" ID BTN_CLOSE AT (PREVRIGHT + 5 1 AUTO AUTO)
   BUTTON "Info" ID BTN_INFO AT (PREVRIGHT + 5 1 AUTO AUTO)
END
```

Como resultado desta descrição textual temos a seguinte tela.

Figura 15 - TELA RESULTADO DO PILRC

4.3.1.3 COMPILADOR BUILD-PRC

Conforme Foster (2002), este é o utilitário que converte o código objeto gerado pelo compilador *gcc* em um executável entendido pelo palmOS, ele também combina as informações de interfaces geradas pelo compilador *Pilrc*gerando assim o executável final que irá executar no palm.

4.3.1.4 DEPURADOR GDB

Conforme Foster (2002) é ferramenta de depuração para aplicativos para palmOS, que compõem o conjunto de programas do compilador *gcc*, funciona em modo texto. Ele trabalha integrado com o emulador de palm.

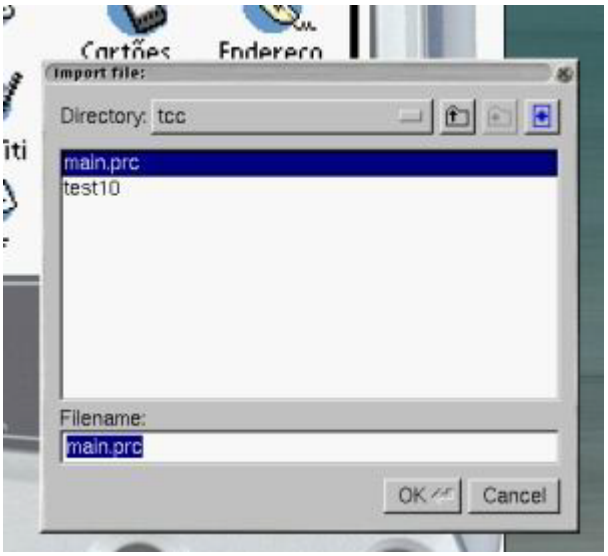
Para depuração de um programa é necessário ter o código objeto gerado pelo *gcc* e o programa executando dentro do *emulador*, nisto deverá ser executando o seguinte comando “m68k-palmos-gdb <arquivo.o>”, depois dentro do gdb deve ser executado o comando “target palmos”, apartir deste momento quando o programa for executado no emulador ele irá começar a ser depurado dentro o gdb.

4.3.2 OPERACIONALIDADE DA IMPLEMENTAÇÃO

Para a execução do protótipo o primeiro passo é instalar o aplicativo dentro do emulador, para isso é necessário o arquivo main.prc, que é gerado apartir da compilação do protótipo. No emulador é necessário acessar a função “Install Application/Database”, selecionar qual o local onde está o aplicativo.

Figura 16 – INSTALAR APLICATIVO

Figura 17 - SELEÇÃO DO APLICATIVO



Após este processo o aplicativo irá aparecer na lista de todos os programas do palm. Selecionado o ícone TCC irá ser iniciada a execução do protótipo.

Figura 18 - APLICATIVO INSTALADO

Neste ponto chega-se a tela principal do editor, que é composta por botões que executam as funções mais comuns do editor.

Figura 19 - TELA PRINCIPAL DO PROTÓTIPO

Agora se deve criar um novo polígono selecionando o botão novo, e utilizando a caneta do palm. Tocando a área central do editor defini-se os pontos que formarão o polígono.

Figura 20 - NOVO POLÍGONO

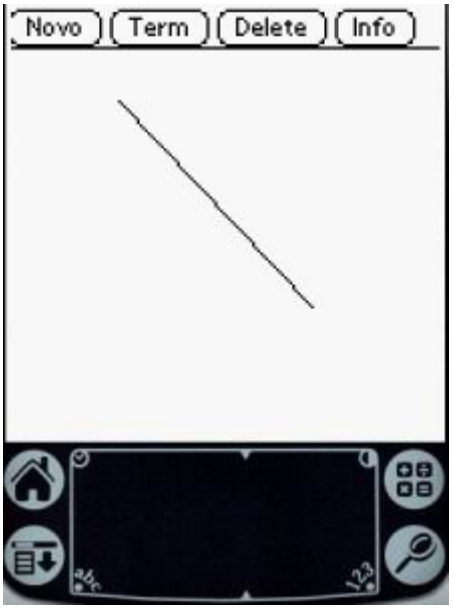
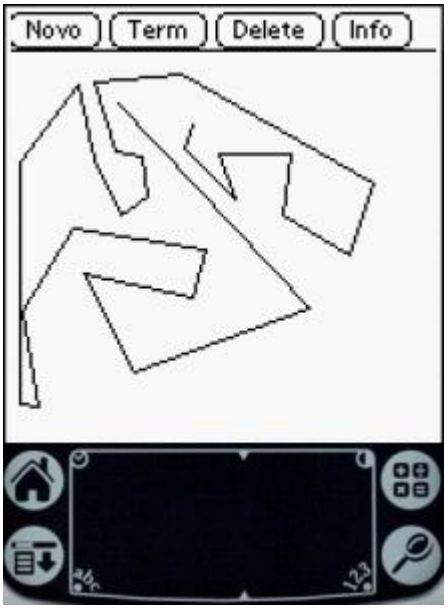
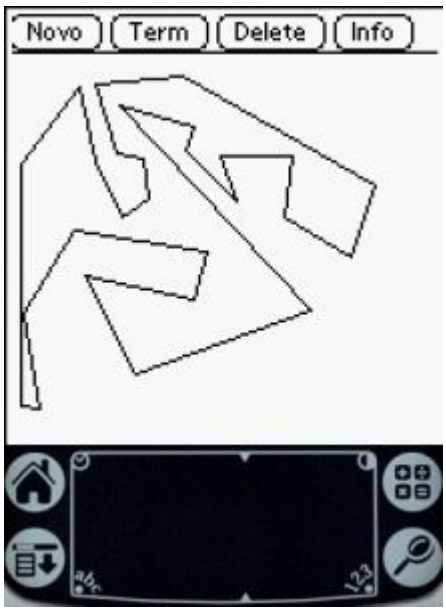


Figura 21 - PONTO DEFINIDOS



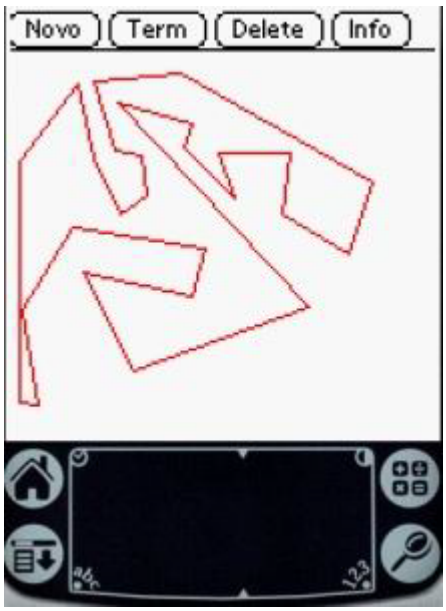
Depois de ter definido uma quantidade de pontos deseja-se terminar o polígono, para isso deve ser selecionado o botão *term*, este procedimento cria um novo ponto no polígono que é o ponto inicial fechando assim o polígono.

Figura 22 - POLIGONO TERMINADO



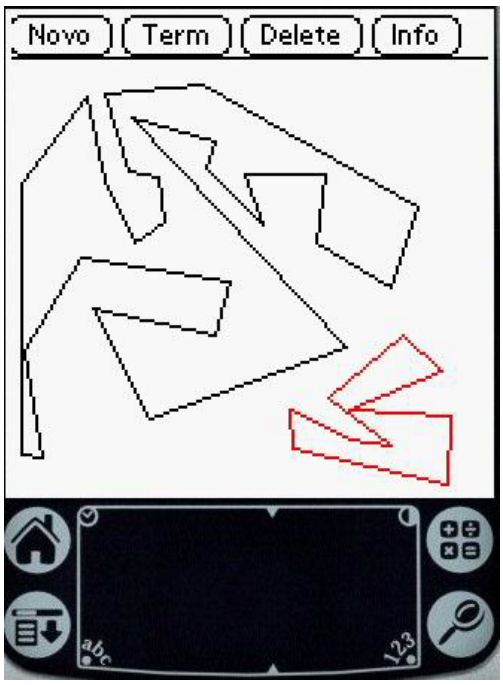
Neste ponto com a caneta, pode-se selecionar o polígono indicando um ponto na tela. Todos os polígonos que possuírem aquele ponto serão selecionados.

Figura 23 - POLIGONO SELECIONADO



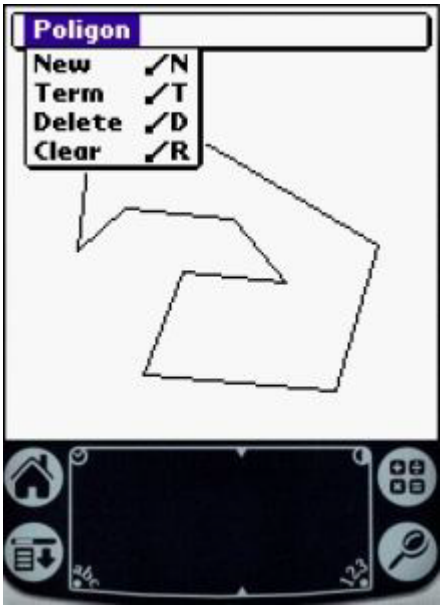
Tendo um conjunto de polígonos selecionados, deseja-se remover-los do editor. Para isso deve ser selecionado o botão *Delete*

Figura 24 - DELETAR POLÍGONO



Todas as funções que estão disponíveis por botões também podem ser acessadas pelo menu. O menu traz mais uma opção que não está disponível nos botões devido ao problema do tamanho da tela, como já falamos o que dever ser apresentado no palm é a informação e não a interface, por este motivo, só os comandos mais acessados estão disponíveis nos botões, a opção Clear destina-se a remover todos os polígonos do editor.

Figura 25 - MENU DO EDITOR



Para a finalização do protótipo, selecione o botão *applications* do palm.

4.4 RESULTADOS E DISCUSSÃO

O desenvolvimento de programas para *palm* não é muito diferente do desenvolvimento para qualquer outra plataforma computacional. Existem algumas limitações impostas pela plataforma quanto ao uso de memória e processamento, mas com a evolução dos computadores *palm* acredita-se que isto será menos relevante nos anos futuros. Entretanto, algumas preocupações permanecem, pois o conceito da plataforma é construir programas com interfaces simples, de fácil uso e que apresentem os resultados esperados, sem muito processamento.

Assim, basicamente o processo de desenvolvimento de programas para *palm* segue as etapas de implementação das funções de controle do programa, ou seja, as rotinas que possuem a "inteligência" do programa. No caso do protótipo apresentado, as rotinas de controle do programa são os fontes C++, onde é realizado o tratamento para o desenho, seleção e exclusão dos polígonos. Para esta implementação foi utilizado o editor

de texto *vi* no *Linux*. Após as rotinas de controle estarem prontas deve-se ser construir as telas do programa. Para esta etapa utiliza-se o compilador de interfaces *pilrc*. Estando as interfaces definidas, deve ser feito o tratamento de todas as ações que o usuário pode realizar na interface. Somente os eventos que são relevantes ao funcionamento do programa precisam ser tratados, ou seja, quando o usuário selecionar um botão na interface, esta ação deverá gerar alguma mudança dentro no programa, caso contrário o botão não necessitaria estar na tela. As ações que o usuário realiza são sinalizadas ao programa através de eventos, os quais devem ser tratados pelo tratador de eventos que identifica qual a origem do evento e executa o procedimento relativo àquele evento.

Após a implementação das rotinas básicas, das interfaces e dos tratadores de eventos, deve-se implementar o *loop* de eventos, que é a parte do programa que recebe os eventos do sistema operacional e executa o tratador de eventos. Quando a aplicação está rodando, este *loop* de eventos continua sendo executado até que o evento de término do programa for gerado.

Em resumo, os passos para implementação de uma aplicação para *palm* usando a linguagem C++ são os seguintes:

- Escrever as rotinas básicas do programa;
- Definir as interfaces;
- Escrever os tratadores de eventos;
- Escrever o *loop* de eventos;
- Fazer a inicialização do programa;
- Executar o formulário principal.

A seguir, são apresentadas as principais conclusões decorrentes da realização deste trabalho.

5 CONCLUSÕES

O presente trabalho teve como objetivo principal construir um protótipo de editor gráfico 2D a fim de explorar as capacidades gráficas da plataforma palm. Durante o processo de desenvolvimento do protótipo, observou-se que a plataforma Palm apresenta bons recursos gráficos, tanto em relação ao hardware quanto em relação à sua API, apesar das limitações quanto ao uso de memória, inerentes a este tipo de equipamento.

Quanto ao algoritmo de ponto em polígono já é tradicional na literatura de Geometria Computacional, e sua eficiência e simplicidade revelaram-se também no protótipo para palm. Porém, quando a quantidade de polígonos for muito grande o tempo de resposta aumenta, visto que para obter a solução deve-se aplicar o algoritmo para cada polígono. Uma solução para este problema é proposta por NEDEL (2003), e consiste em realizar um pré-processamento no teste de polígonos através da subdivisão do espaço R^2 em faixas de pesquisa. Assim, procura-se otimizar a pesquisa dando-se preferência para polígonos dentro de determinadas faixas onde há maior probabilidade de haver polígonos selecionados. A implementação desta melhoria no algoritmo representa uma sugestão de continuação para o presente trabalho. O objetivo principal da implementação deste editor 2D foi compreender e testar o processo completo de desenvolvimento de aplicações para *palm* em C++ na plataforma Linux. O protótipo implementado pode servir de subsídio para o desenvolvimento de outros tipos de aplicações que necessitem de interfaces gráficas interativas de desenho, tais como aplicações educacionais para o ensino de matemática e geometria, ou até o desenvolvimento de jogos educativos.

O compilador utilizado mostrou-se satisfatório, suprindo todas as necessidades encontradas na implementação do protótipo. Tanto implementações em C quanto em C++ não apresentaram problemas. Visto que a presente implementação não utilizou todos os recursos da plataforma, outra sugestão seria a exploração dos demais recursos que a plataforma possui, tais como recursos de som, permitindo uma análise melhor do compilador.

Quanto ao editor muitas melhorias ainda podem ser feitas, como permitir a modificação dos polígonos após eles terem sido construídos, e também permitir salvá-los,

além de implementar funcionalidades mais interativas tais como mover/arrastar polígonos interativamente com o auxílio da caneta do *palm*. A opção de salvar torna-se interessante não somente por permitir ter estas informações depois que o editor é finalizado, mas também por permitir explorar outro recurso da plataforma que é a sincronização, processo em que dois computadores *palm* trocam informações, ou um computador *palm* troca informações com um computador de mesa, ou qualquer outro dispositivo. Esta é uma sugestão que permitiria sincronizar modelos gerados no palmtop para algum outro software no computador comum e vice-versa.

6 REFÊRENCIAS

CORNEY, Jonathan. **3D Modeling with the acis kernel and toolkit**. Indianápolis: Wiley, 1998.

FIGUEIREDO, Luiz Henrique; CARVALHO, Paulo C. P. Introdução à geometria computacional. In: COLÓQUIO BRASILEIRO DE MATEMÁTICA, 18., 1993, Rio de Janeiro. **Anais...** Rio de Janeiro: Impa, 1993. p. 30-31.

FOLEY, James D. **Computer graphics : principles and practice**. 2.ed. Reading: Addison-Wesley, c1990.

FOSTER, Lonnon R. **Palm OS programming bible.** 2 ed. Indianápolis: Wiley, 2002.

FREITAS, Eduardo Garcia; **Geometria computacional - IME-USP**, São Paulo, jun, 2003. Disponível em: <http://www.ime.usp.br/~freitas/gc/>. Acesso em: 02 jun 2003.

FURLAN, Davi. **Modelagem de objetos através da UML** – the unified modeling language. São Paulo: Makron Books, 1998. 329 p.

GIOVANNI, José Ruy; BONJORNO, José Roberto; GIOVANNI, José Ruy Junior. **Matemática**. São Paulo: FTP, 1988.

NEDEL, Luciana Porcher; COHEN Marcelo. **Geometria Computacional - Método das Faixas**, [S.l.], jun 2003. Disponível em: <http://www.inf.pucrs.br/cg/Aulas/GeomComp/Slab/Slab.html>. Acesso em: 02 jun 2003.

PALM, **Palm Brasil**, [S.l.], jun. 2003. Disponível em: <http://www.palm.com/br>. Acesso em: 02 jun. 2003.

PALMSOURCE, **Palm OS Desktop Development**, [S.l.], jun. 2003. Disponível em: <http://www.palmos.com/dev/tools/sdk/sdk50.html>. Acesso em: 02 jun. 2003.

QUATRANI, Terry. **Modelagem visual com rational rose 2000 e UML**. Tradução Savannah Hartmann. Rio de Janeiro: Ciência Moderna, 2001. 206 p.