Universidade de Brasília – UnB

Aluno: Guilherme David Branco **Mat:** 11/0012470

Curso: Organização e Arquitetura de Computadores - CiC

Professor: Ricardo Pezzoul Jacobi

Objetivos:

O objetivo deste projeto é estudar, desenvolver e implementar as primitivas gráficas na linguagem assembly MIPS, assim como testar o mesmo no simulador MARS.

Indíce:

Convenção de código

Todas as funções que podem ser chamadas por código externo contém uma breve explicação em forma de comentário antes da declaração de sua “label”, esta explicação explicíta os registradores que são utilizados pelo método assim como o significa real destes.

Declaração de métodos

Existem métodos “private” e métodos “public” no código, por convenção decidida pelo autor, os que podem ser utilizados externamente não possuem em sua declaração o caractér “.” antes da primeira letra. Todo “label” que contém “.” como primeiro caractér é uma função ajudante.

Ex.:

point:

...código

.p_sair:

...código

Neste caso, point é o método que deve ser chamado e .p_sair é um método ajudante para o método point.

Labels

Como já descrito “labels” com início em “.” equivalem a funções ajudantes, para que não haja repetição de nomes entre estes foi adotado uma convenção de que os ajudantes de um certo método terão o começo do nome do método abreviado.

Ex.:

plotLine:

.pl_sair:

rect:

.r_loop:

Esta convenção garante que não haverão “labels” com mesmo nome, pois o seu prefixo será sempre correspondente ao método que as utilizam.

Implementação das primitivas de desenho

Altura, Largura e endereço de memória

Os valores de altura, largura são baseados nos valores padrões que o mars coloca ao ser aberto. Já o valor de endereço de memória inicial para o display deve ser colocado em heap(0x10040000), para que não entre em conflito.

Método: point

O método coloca um pixel na tela numa cor especificada, para tal é necessário saber o endereço em memória de onde o pixel estará, assim ao escrever o valor da cor neste endereço o pixel irá aparecer. O calculo do endereço é dado pela fórmula:

$$p_{end} = 4(y * largura + x) + end\_inicial$$

Método: plotLine

O método desenha uma linha de pixeis entre dois pontos a partir de uma cor especifica, utilizando o altoritmo de Bresenham's para linhas.

A implementação deste algoritmo foi baseada em:

```
function line(x0, y0, x1, y1)
  dx := abs(x1-x0)
  dy := abs(y1-y0)
  if x0 < x1 then sx := 1 else sx := -
  if y0 < y1 then sy := 1 else sy := -
  err := dx-dy

  loop
    plot(x0,y0)
    if x0 = x1 and y0 = y1 exit loop
    e2 := 2*err
    if e2 > -dy then
      err := err - dy
      x0 := x0 + sx
    end if
    if e2 < dx then
      err := err + dx
      y0 := y0 + sy
    end if
  end loop
```

Disponível em <http://en.wikipedia.org/wiki/Bresenham%27s_line_algorithm#Simplification>. Acesso em: 12/04/2014

Método: rect

Desenha um retângulo, o canto superior esquerdo é dado pelas coordenadas x0 e y0, com altura, largura e cor também especificados pelo usuário. Para sua implementação foi utilizado um loop e um contador para decidir qual das 4 linhas esta sendo desenhada no momento, assim traçando as bordas do retângulo.

Método: fillRect

Desenha um retângulo sólido, primeiro desenha as bordas do retângulo para o usuário ver a partir do mesmo anterior, após isto preenche a parte de dentro do retângulo com linhas com uma lógica parecida com a anterior, porém o contador ao invés de ir até 4 vai até a altura do retângulo e as linhas de preenchimento respeitam os limites previamente desenhados.

Método: circle

Desenha um círculo com centro em x0 e y0, assim como com cor e raio especificados pelo usuário. Utiliza o algoritmo “Midpoint circle” abaixo:

```
procédure tracerCercle (entier rayon, entier x_centre, entier y_centre)
        déclarer entier x, y, m ;
        x ← 0 ;
        y ← rayon ;             // on se place en haut du cercle
        m ← 5 - 4*rayon ;       // initialisation
        Tant que x <= y         // tant qu'on est dans le second octant
                tracerPixel( x+x_centre, y+y_centre ) ;
                tracerPixel( y+x_centre, x+y_centre ) ;
                tracerPixel( -x+x_centre, y+y_centre ) ;
                tracerPixel( -y+x_centre, x+y_centre ) ;
                tracerPixel( x+x_centre, -y+y_centre ) ;
                tracerPixel( y+x_centre, -x+y_centre ) ;
                tracerPixel( -x+x_centre, -y+y_centre ) ;
                tracerPixel( -y+x_centre, -x+y_centre ) ;
                si m > 0 alors       //choix du point F
                        y ← y - 1 ;
                        m ← m - 8*y ;
                fin si ;
                x ← x + 1 ;
                m ← m + 8*x + 4 ;
        fin tant que ;
fin de procédure ;
```

Disponível em <http://fr.wikipedia.org/wiki/Algorithme_de_trac%C3%A9_d%27arc_de_cercle_de_Bresenham>.Acesso em: 12/04/2014

Problemas encontrados:

O código não foi amplamente testado, afinal é apenas um trabalho de estudo, como a especificação não pedia muita coisa além do básico existem algumas falhas no programa.

O código não trata as entradas do usuário de maneira a achar erros, como altura e largura dos retângulos serem 0 ou x0 e y0 estarem fora das bordas da tela bitmap, ou se algum dos pontos da linha estarão fora da tela.