SSS – Slide Show System

Licenciatura em Ciências da Computação

Programação Imperativa 2009

Segundo Projecto

Versão de 21 de Março de 2009

Alberto Simões José C. Ramalho

Instruções

Este projecto deve ser realizado tendo em conta que:

- os grupos de trabalho poderão ter até três elementos;
- serão realizadas avaliações intercalares, pelo que o não cumprimento das entregas e/ou apresentações correspondem à avaliação nula;
- cada grupo deverá preparar uma solução computacional desenvolvida na linguagem C usando a plataforma gráfica OpenGL/GLUT;
- a aplicação deverá ser entregue em cada avaliação intercalar em **código fonte**, e deverá compilar em ambiente Linux;
- o pacote entregue deverá ser em formato `tar.gz`, incluindo uma directoria com o código fonte, um documento `README` com a identificação do grupo, e um ficheiro `Makefile` com as regras de compilação;
- deverá ser entregue um relatório **impresso** que documente o problema, discuta a solução e apresente os resultados obtidos;
- **VISITE SEMANALMENTE A PÁGINA DA DISCIPLINA E CONSULTE O ENUNCIADO!** Embora o enunciado não vá sofrer alterações de fundo, algumas dicas ou informações extras podem vir a surgir.

1 Descrição do *SSS*

Pretende-se desenvolver uma ferramenta para a animação de *slide-show*, apresentando imagens, transições e texto. A este sistema chamou-se *Slide Show System* e será de agora a diante designado por *SSS*.

Este sistema baseia-se numa linguagem textual, descrita num ficheiro `.sss`. Este ficheiro descreve as várias operações que devem ser realizadas. Estas operações podem ser de vários tipos:

- mostrar uma imagem;
- manipular graficamente a imagem;
- adicionar texto sobre uma imagem;
- animar uma imagem;
- transitar para outra imagem mediante uma animação;

Quando o sistema acaba de interpretar todo o ficheiro poderá voltar ao início, repetindo toda a animação, ou sair, de acordo com as instruções presentes.

1.1 Execução do *SSS*

O sistema *SSS* é invocado a partir da linha de comando, passando-lhe como parâmetro o nome do ficheiro em que estão descritas as operações para a realização do *slide-show*.

```
1        [user@machine PI]$ ./sss slideshow.sss
```

Este sistema funciona com as seguintes premissas:

- a área de apresentação tem as dimensões de 800×600 pixeis;
- as imagens maiores que estas dimensões serão truncadas;
- as imagens menores que estas dimensões serão mostradas, preenchendo a área restante com a cor preta;
- o sistema deverá indicar uma mensagem de erro no caso de uma linha da linguagem *SSS* estar incorrecta. No entanto, não deverá sair da apresentação, passando a interpretar a próxima linha;

A apresentação deverá iniciar com uma imagem preta.

1.2 A linguagem *SSS*

A notação *SSS* é descrita num ficheiro textual. Cada linha corresponde a uma operação gráfica (que pode demorar mais ou menos tempo). Segue-se a descrição de cada uma das operações suportadas.

1.2.1 Comandos básicos

`clear`

O comando mais simples de apresentação é o `clear` que permite apagar toda a área de desenho. Este comando suporta duas formas:

- apenas o comando `clear`, preencherá a área de apresentação com a cor preta;
- o comando `clear` seguido de uma das cores `black`, `white`, `red`, `green`, `blue`, apagará a área de apresentação com a cor especificada.

Quando o sistema *SSS* inicia o comando `clear` deverá ser chamado implicitamente para limpar a área de desenho para a cor preta.

`show`

Para carregar uma imagem é usado o comando `show` seguido do nome do ficheiro a apresentar:

```
1        show whitehouse.pnm
```

Os tipos de imagem suportados pelo sistema serão descritos na secção 3.

Todos os comandos de manipulação de imagens que se sigam são acções sobre esta imagem. A imagem deve ser mantida até aparecer um novo comando `show`, ou até surgir um comando de transição (ver mais à frente).

`sleep`

Para obrigar o sistema a aguardar será usado o comando `sleep` que é seguido por um número natural:

```
1        sleep 20
```

Este valor não corresponde ao número de segundos como é habitual nas linguagens de programação. Corresponde, sim, ao número de iterações que devem ser ignoradas. A secção 2 apresenta o código base para a aplicação *SSS*, e descreve com mais detalhe o que se entende por uma iteração.

`end`

`loop`

A qualquer instante pode surgir um comando de fim de apresentação. Este comando pode tomar duas formas: indicar que a apresentação deve voltar ao início, ou indicar que a apresentação terminou. Em qualquer um dos casos, todos os comandos que apareçam nas linhas seguintes serão ignorados.

Para indicar que a apresentação terminou é usado o comando `end`, enquanto que o comando `loop` é usado para indicar que a apresentação deve voltar ao início.

```
1       show whitehouse.pnm
2       sleep 20
3       show obama.pnm
4       sleep 20
5       loop
6       esta linha nunca é interpretada...
```

No exemplo, a primeira linha indica que deve ser mostrada a fotografia da casa branca. Segue-se uma espera de 20 iterações, para depois mostrar a fotografia do presidente Obama. Depois de se esperar mais 20 iterações, a apresentação é re-iniciada mostrando de novo a fotografia da casa branca.

Note que qualquer linha depois do comando `end` ou `loop` deve ser ignorada.

1.2.2 Manipulação de Imagem

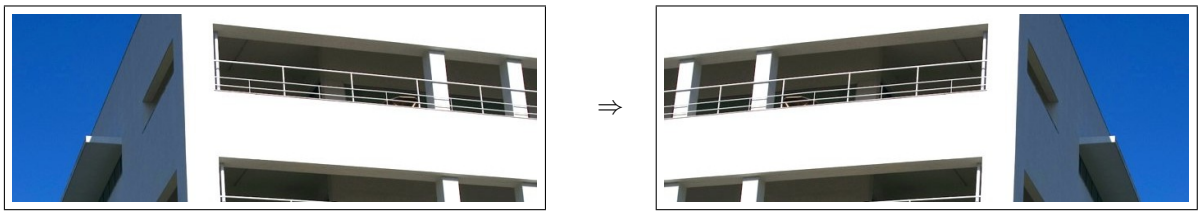
Esta secção apresenta um conjunto de comandos que serão aplicados sobre toda a área de apresentação (seja qual for o tamanho da imagem).

`invert`

A inversão de uma imagem corresponde ao cálculo da imagem negativa. O processamento é bastante simples, bastando para cada ponto calcular a cor inversa. A cor inversa é calculada calculando o inverso[1] de cada componente (vermelha, verde e azul).

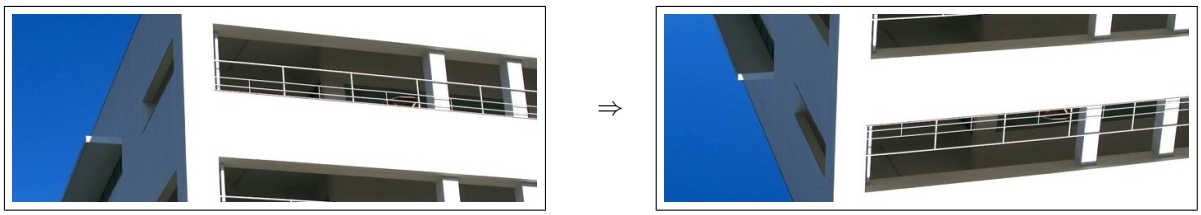
`flop`

Este comando não recebe qualquer argumento, e serve para rodar a imagem sobre o eixo vertical.



`flip`

Este comando não recebe qualquer argumento, e serve para rodar a imagem sobre o eixo horizontal.



[1] $1 - valor$

`blur`

Este comando serve para desfocar uma imagem. Recebe como argumento um inteiro que corresponde ao raio do processamento. O valor deve ser um inteiro, entre 1 e 3.

v_3	v_3	v_3	v_3	v_3	v_3	v_3
v_3	v_2	v_2	v_2	v_2	v_2	v_3
v_3	v_2	v_1	v_1	v_1	v_2	v_3
v_3	v_2	v_1	X	v_1	v_2	v_3
v_3	v_2	v_1	v_1	v_1	v_2	v_3
v_3	v_2	v_2	v_2	v_2	v_2	v_3
v_3	v_3	v_3	v_3	v_3	v_3	v_3

Considerando que a imagem representa um pixel (X), a imagem apresenta os vizinhos no raio de 1, 2 e 3 pixeis.

O cálculo de uma imagem desfocada corresponde a processar cada pixel da imagem. Cada pixel será transformado para a média das cores dos vizinhos.

Por exemplo, no caso do blur de raio 2, cada componente RGB é calculada com base em:

$$X = \frac{\sum_{v=1}^{2} v_i + X}{25}$$

1.2.3 Animação de Transições

A transição entre imagens numa apresentação pode ser feita apenas redesenhando a imagem actual (com o comando `show`) ou através de uma transição suave/animada.

Os comandos de transição são da forma:

```
1   nomeDaTransição   nomeDaNovaImagem
```

em que o *nome da transição* corresponde a uma das seguintes transições[2]:

`wipeLeft`

`wipeRight`

`wipeUp`

`wipeDown`

Estas quatro transições deslizam a nova imagem sobre a actual. Por exemplo, com o `wipeRight`, a imagem começa a entrar na área de desenho pelo lado esquerdo (em direcção ao lado direito), enquanto que a imagem antiga é mantida sempre na mesma posição. Ou seja, a nova imagem vai-se sobrepondo progressivamente à imagem antiga.

`slideLeft`

`slideRight`

`slideUp`

`slideDown`

Estas quatro transições deslizam a nova imagem para o lugar da actual, enquanto que a actual vai deslizando para fora da área de desenho. Por exemplo, com `slideDown`, é desenhada a última linha da nova imagem no topo da área de desenho, e desenhada a imagem actual a partir da segunda linha da área de desenho. Na iteração seguinte, são desenhadas as duas últimas linhas da nova imagem no topo da área de desenho, e desenhada a imagem actual a partir da terceira linha, e assim sucessivamente, até que toda a nova imagem esteja a ser desenhada na área de desenho.

[2]Se estas transições forem realizadas pixel a pixel demoram bastante tempo. Como opção pense numa forma de a tornar mais rápida.

`fadeWhite`

`fadeBlack`

Estas duas transições realizam um *fading* da imagem para uma cor (branco ou preto), e depois o *fading* inverso para a nova imagem.

Para exemplificar o processo de *fading* considere o *fadeWhite*. A cor branca é composta pelas três componentes (vermelha, verde e azul) como valor máximo. Assim, o processo de fazer um *fade* da imagem para a cor branca corresponde a ir adicionando um valor a cada componente de todos os pixeis, mostrar a imagem, voltar a adicionar um valor a cada componente, e assim sucessivamente, até que todos os pixeis estejam com todas as componentes ao máximo. O processo de *fade* para a imagem é inverso, removendo valores aos componentes até que estes sejam iguais às cores definidas pela imagem que se pretende mostrar.

1.2.4 Geração de Texto

Finalmente, pretende-se que se possa mostrar palavras centradas horizontalmente na imagem, na margem inferior (imagine uma legenda de um filme). Esta imagem pode ser vista como a legenda da imagem.

Para desenhar palavras é necessário desenhar caracteres. Para desenhar caracteres é necessário saber como estes se desenham. Para isso, considere que existe uma directoria com 26 ficheiros chamados `a.pnm`, `b.pnm` até `z.pnm` no formato P1 (ver descrição mais abaixo). Considere também que todas as letras têm a mesma largura.

O *SSS* tem um comando para ler o alfabeto destes 26 ficheiros para memória. A partir deste ponto, será possível escrever em qualquer imagem usando o comando `write`. A mensagem será escrita na cor especificada no comando, centrada, e numa única linha. Se a mensagem ocupar mais do que a largura da zona de desenho envie uma mensagem de erro para o STDERR e não desenhe a legenda.

`loadAlphabet`

Este comando é seguido do nome de uma directoria (caminho relativo à pasta actual). Nessa directoria existem os 26 ficheiros pnm com as 26 letras, como descrito acima. Este processo não desenha o quer que seja, simplesmente carrega para memória a descrição das letras:

```
1       loadAlphabet alfabetoGotico
```

`write`

O comando `write` recebe como segundo argumento a cor (`black`, `white`, `red`, `green`, `blue`) e como terceiro argumento a legenda a ser escrita. Note que o espaço também deverá ser considerado!

```
1        write black Kremlin
```

2 O Código OpenGL Disponibilizado

O pacote com código disponibilizado inclui:

- uma `Makefile` com três regras distintas: `mac`, `linux` e `cygwin`. Estas três regras definem como a aplicação se compila nestes três sistemas. Embora esta `Makefile` seja para ser adaptada às necessidades do grupo, estas três regras devem ser mantidas, para que se possa testar com facilidade cada projecto;
- o ficheiro `defs.h` que define a área de apresentação, e o protótipo da função `load_pbm`.
- o ficheiro `whitehouse.pnm` com uma imagem da casa branca, para testes;
- o ficheiro `display.c` onde está o código base:

  main inicializa o ambiente gráfico, criando uma janela. Posteriormente invoca a função `init` para criar a área de desenho;

  init inicializa a área de desenho e, neste momento, usa a função `load_pbm` para ler de um ficheiro uma imagem;

  display é invocada a cada iteração do sistema gráfico para redesenhar a janela de apresentação. Assim, quando ao apresentar o comando `sleep` se diz que espera n iterações, está-se a dizer que depois de lido o comando, as n próximas invocações a esta função deverá simplesmente redesenhar a imagem actual.

load_pbm carrega uma imagem no formato **P3**;

O código tal como foi disponibilizado deverá compilar e, quando executado, mostrar uma janela com a casa branca. Note que para compilar deve, antes de mais, editar a `Makefile` para a adaptar ao seu sistema.

3 Os formatos PNM

Existem vários formatos PNM. Esta secção descreve apenas os que se espera que sejam reconhecidos pelo *SSS*. Para mais informações sobre estes formatos: `http://en.wikipedia.org/wiki/Netpbm_format`

3.1 Formato P3

A função disponibilizada (`load_pbm`) carrega uma imagem no formato **P3**. Este formato é completamente textual e de leitura razoavelmente simples ao ser humano. A primeira linha identifica o formato em causa, e tem apenas os caracteres `P3`. Segue-se uma linha com as dimensões da imagem (número de pixeis na horizontal e na vertical), separadas por um espaço. A terceira linha contém a *profundidade* ou *nível de cor* da imagem, que é indicado por um inteiro (habitualmente com o valor 255, para 255 níveis de cor).

Na quarta linha começam a ser descritos os pixeis da imagem, começando pelo canto superior esquerdo da imagem, linha a linha. Cada pixel é identificado por três valores inteiros que variam entre 0 e a *profundidade* da imagem definida. Estes três valores inteiros correspondem à intensidade das cores Vermelha, Verde e Azul. Habitualmente este modelo de cores é representado por RGB (Red, Green, Blue). Em cada uma das componentes, 0 indica a ausência dessa cor, enquanto que um valor positivo, quanto mais próximo da *profundidade* da imagem, indica maior luminosidade.

É importante referir que a função base disponibilizada executa o carregamento de uma imagem de forma muito simplicista, sem validar o tipo de imagem, as dimensões ou a *profundidade* da cor. Logo, deve adaptar a função para ter em conta estes diferentes problemas.

Segue-se um exemplo de um documento neste formato:

```
1  P3
2  3 2
3  255
4  255 0 0  0 255 0  0 0 255
5  255 255 0  255 255 255  0 0 0
```

3.2 Formato P6

O formato **P6** é semelhante ao formato **P3**, com algumas diferenças. Em primeiro lugar, a primeira linha deverá conter os caracteres `P6`. A segunda e terceira linha são semelhantes ao formato **P3**, mas os pixeis são indicados de modo binário, usando um byte para cada cor. Note que estes valores devem ser lidos usando a função `fread` e não o `fscanf`.

3.3 Formato P1

O formato **P1** usa apenas duas cores (preto e branco, ou preto e ausência de cor). O formato é similar aos anteriores, mas em que não existe uma linha a indicar a *profundidade* da imagem, já que este formato só suporta duas cores.

O seguinte exemplo codifica no formato **P1** um **J**:

```
1  P1
2  6 10
3  0 0 0 0 1 0
4  0 0 0 0 1 0
5  0 0 0 0 1 0
6  0 0 0 0 1 0
7  0 0 0 0 1 0
8  0 0 0 0 1 0
9  1 0 0 0 1 0
10 0 1 1 1 0 0
11 0 0 0 0 0 0
12 0 0 0 0 0 0
```

4 Etapas do Projecto

Sendo este o projecto da disciplina é natural que alguns conhecimentos necessários à sua realização sejam leccionados ao longo do semestre. A apresentação do sistema *SSS* corresponde à lista de requisitos da aplicação. No entanto, o seu desenvolvimento será feito de forma incremental, de acordo com as etapas descritas nesta secção.

4.1 Etapa I

A primeira etapa serve essencialmente para que ganhem confiança com o código disponibilizado. Deste modo, pretende-se que o sistema *SSS* suporte:

- a leitura de um programa a partir do `stdin` (de modo a que possam executar o *SSS* com `./sss < slides.sss`);
- a apresentação de imagens do tipo P3;
- os comandos básicos, `clear`, `show`, `sleep`, `end` e `loop`.
- os comandos de transformação, `flip`, `flop`, `blur`, `invert`,

4.2 Etapa II

A segunda etapa deve suportar transições e escrita de texto, bem como o suporte para as imagens de tipo P6. Assim, a aplicação deve ser capaz de:

- ler um programa a partir de um ficheiro passado como parâmetro na linha de comando;
- carregar imagens em formato P3 ou P6, detectando o seu formato de forma automática;
- realizar as transições entre imagens definidas;
- ler uma especificação de tipo de letras, para escrita sobre a imagem;
- adicionar uma legenda sobre uma imagem;