Projeto
Anáglifos em *Assembly*

1 Introdução

1.1 O que são anáglifos?

Um anáglifo é um efeito tridimensional que se obtém a partir de uma combinação de imagens estereoscópicas e que pode ser observado através de um filtro especial (p. ex., óculos com uma lente de cada cor[1]). Imagens estereoscópicas (ou estereogramas) nada mais são do que imagens captadas com uma ligeira distância horizontal entre elas, semelhante ao que acontece com os olhos dos seres humanos. A Figura 1 exemplifica a obtenção de um estereograma, a criação de um anáglifo a partir do estereograma obtido e a visualização do anáglifo.

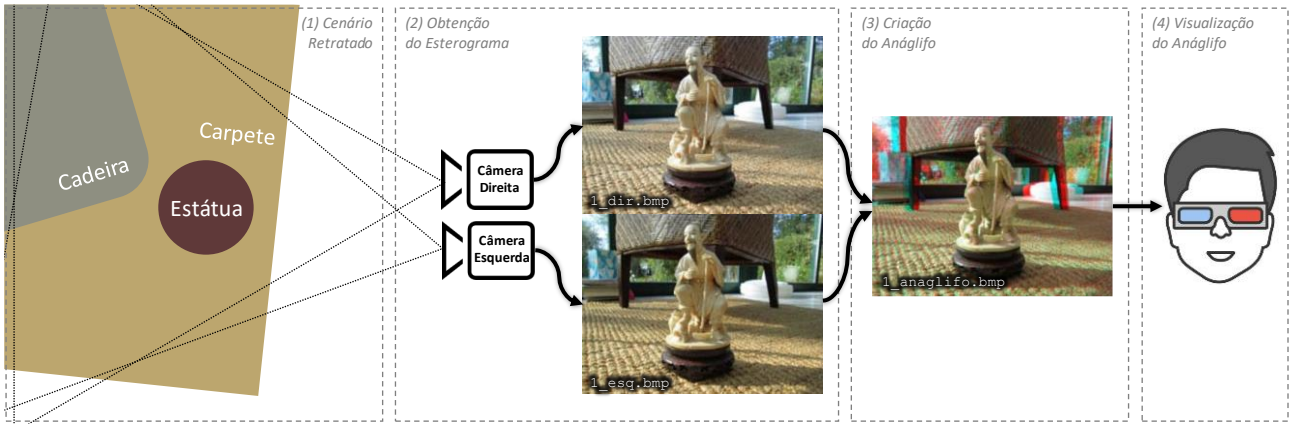

Figura 1: Exemplo da obtenção de um estereograma e a criação e visualização de um anáglifo.

1.2 Exemplos práticos do uso de anáglifos

Durante algum tempo, os anáglifos foram amplamente utilizados em videogames, filmes e marketing como uma solução barata para a obtenção de efeitos tridimensionais através de imagens bidimensionais modificadas. A título de curiosidade, este efeito é possível principalmente porque o córtex visual é capaz de reconhecer a profundidade relativa entre objetos através da estereopsia (visão binocular).

Esta técnica acabou por se tornar obsoleta na indústria do entretenimento devido à redução do custo de outras técnicas tridimensionais com resultados visualmente mais agradáveis, com menos requisitos para a visualização e com menos efeitos colaterais[2] aos seus utilizadores. Mais recentemente, esta técnica tem sido empregada em aplicações científicas que se beneficiam da sua relação custo-benefício para atingir os seus objetivos. Dois exemplos recentes destes usos científicos são a exploração de outros planetas (e.g., *Mars Curiosity Rover* [NASA22a, NASA22b]) e a análise tridimensional de órgãos em pleno funcionamento (e.g., [Iza22]).

[1] A última página deste enunciado contém um molde para imprimir e recortar um óculo que poderá ser utilizado juntamente com os papéis celofane vermelho e azul que serão entregues aos alunos nas próximas aulas teóricas.
[2] Alguns utilizadores são mais sensíveis quando expostos a estereopsia por longos períodos e podem relatar casos de tonturas e náuseas. Nestes casos, interrompa imediatamente a visualização de anáglifos com os filtros.

2 Contexto

O objetivo deste projeto será **criar um programa em linguagem de programação *Assembly* x86 de 64 bits**, no formato Intel para o `nasm`, **que crie um anáglifo a partir de duas imagens de um estereograma**. Na Figura 1, este programa estará localizado entre os passos *(2) obtenção do estereograma* e *(3) criação do anáglifo*.

Nas próximas secções serão detalhados o formato de imagem *bitmap* (BMP, Secção 2.1), o processo sobre como criar um anáglifo a partir de um estereograma (Secção 2.2), a apresentação dos ficheiros de apoio ao projeto fornecidos juntamente com este enunciado (Secção 2.3) e exemplos de como utilizar o `hexdump` para visualizar os bytes de um ficheiro BMP (Secção 2.4).

2.1 O formato de imagem *bitmap* (BMP)

Os ficheiros *bitmap* (BMP) são ficheiros de imagens onde o valor exato de cada píxel (*px*) da imagem é apresentado explicitamente. Os ficheiros no formato BMP são divididos em duas partes: o **cabeçalho** e a secção dos **píxeis**. Estas duas partes são apresentadas na Figura 2.

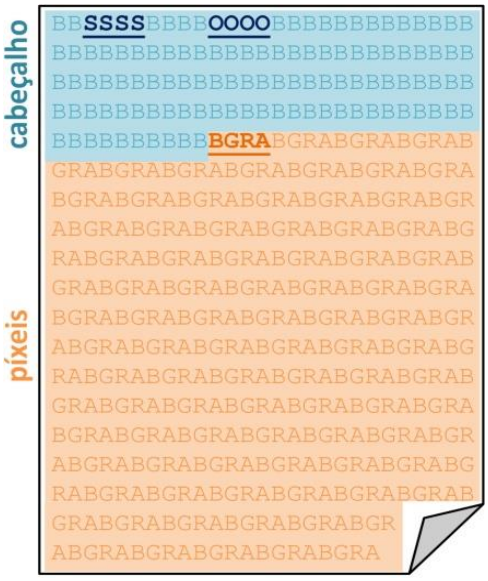

Figura 2: Estrutura de um ficheiro BMP

O cabeçalho pode seguir uma de diversas especificações existentes e pode ter um tamanho variável. Porém, para este trabalho, será utilizada a especificação ARGB32 (que será explicada a seguir) e as únicas informações que serão importantes de ler do cabeçalho serão o tamanho do ficheiro (*size*), o qual está indicado como **SSSS** na Figura 2, e o deslocamento (*offset*), o qual está indicado como **OOOO** na mesma figura. O *size* é um número do tipo *double word* (com tamanho 4 bytes de tamanho) que começa no 3º byte do ficheiro e indica o tamanho total do ficheiro BMP em bytes. O *offset* também é um número do tipo *double word* (de 4 bytes de tamanho), porém começa no 11º byte do ficheiro e indica exatamente em qual byte a secção dos píxeis inicia.

A secção dos píxeis começa a partir desta posição de deslocamento (*offset*) e é composta por conjuntos também de 4 bytes (i.e., 32 bits) por píxel, representados por **BGRA** na Figura 2. Cada conjunto **BGRA** contém 1 byte para a intensidade (um número entre 0 e 255) da cor azul (**B**), 1 byte para o verde (**G**), 1 byte para o vermelho (**R**) e 1 byte para a transparência (**A**, que representa o chamado canal *alfa* e vai de 0 para totalmente transparente a 255 para totalmente opaco).

Esta especificação é chamada ARGB32 e em arquiteturas *little-endian* utilizam esta sequência **BGRA** para cada píxel. Portanto, um píxel composto pelos bytes **0x0000FFFF** é um píxel vermelho puro sem transparência, um **0x00FF00FF** é um verde puro, um **0xFF0000FF** é um azul puro, um **0x00FFFFFF** é um píxel amarelo e assim por diante. Qualquer píxel do tipo **0x??????00** é um píxel transparente. Neste trabalho, vamos sempre utilizar o valor **0xFF** (i.e., 255) para a transparência (o byte **A** de cada píxel). O fim da secção dos píxeis (e do ficheiro BMP) corresponde ao byte na posição indicada pelo número **SSSS** da Figura 2.

2.2 Como criar um anáglifo a partir de um estereograma?

Para o caso específico deste projeto, vamos utilizar dois algoritmos semelhantes para gerar anáglifos (COLOR a cores e MONO em escala de cinza) sempre a partir de um estereograma composto por duas imagens (esquerda e direita) com as mesmas dimensões. A ideia básica para criar o anáglifo com ambos algoritmos será, para cada píxel do anáglifo utilizar o byte para o vermelho (**R**) da imagem da esquerda e utilizar os bytes para o verde (**G**) e azul (**B**) da imagem da direita. O que difere entre os algoritmos COLOR e MONO são os escalares em vírgula flutuante utilizados para multiplicar cada componente de cor dos píxeis [Wim22].

2.2.1 Definição das variáveis iniciais

Mais formalmente, vamos assumir que os valores RGB da imagem do anáglifo que queremos calcular são representados pelas variáveis r_a, g_a e b_a. Vamos também assumir que os valores RGB do pixel correspondente na imagem da esquerda do estereograma são representados por r_e, g_e e b_e, assim como os da imagem da direita por r_d, g_d e b_d.

2.2.2 Algoritmo COLOR

A partir das definições iniciais (Secção 2.2.1), a obtenção do anáglifo no algoritmo COLOR dá-se através da implementação do seguinte conjunto de equações [Wim22]:

$$\begin{pmatrix} r_a \\ g_a \\ b_a \end{pmatrix} = \begin{pmatrix} 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{pmatrix} \cdot \begin{pmatrix} r_e \\ g_e \\ b_e \end{pmatrix} + \begin{pmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 1.0 \end{pmatrix} \cdot \begin{pmatrix} r_d \\ g_d \\ b_d \end{pmatrix}$$

Mais especificamente, pode-se ignorar os escalares 0.0 das equações e assumir que $1.0 \times i = i$, o que resultará no seguinte sistema:

$$\begin{aligned} r_a &= r_e \\ g_a &= g_d \\ b_a &= b_d \end{aligned}$$

Este algoritmo é simples e obtém uma boa reprodução das cores no efeito tridimensional, para além de permitir a perceção da rivalidade binocular (i.e., ao fechar apenas o olho esquerdo com o filtro vê-se uma imagem diferente do que se fechar apenas o olho direito com o filtro).

No algoritmo COLOR é necessário ter atenção com a ordem em que os bytes das cores dos píxeis do anáglifo são obtidos a partir das imagens (esquerda e direita) do estereograma. A Figura 3 ajuda a ilustrar esta ordem, a qual resulta no seguinte padrão:

- O 1º byte (índice 0, B) do 1º píxel do anáglifo é obtido a partir do 1º byte (índice 0, B) do 1º píxel da imagem da direita.
- O 2º byte (índice 1, G) do 1º píxel do anáglifo é obtido a partir do 2º byte (índice 1, G) do 1º píxel da imagem da direita.
- O 3º byte (índice 2, R) do 1º píxel do anáglifo é obtido a partir do 3º byte (índice 2, R) do 1º píxel da imagem da esquerda.
- O 4º byte (índice 3, A) do 1º píxel do anáglifo terá o valor FF (sem transparência).
- O 1º byte (índice 0, B) do 2º píxel do anáglifo é obtido a partir do 1º byte (índice 0, B) do 2º píxel da imagem da direita.
- O 2º byte (índice 1, G) do 2º píxel do anáglifo é obtido a partir do 2º byte (índice 1, G) do 2º píxel da imagem da direita.
- O 3º byte (índice 2, R) do 2º píxel do anáglifo é obtido a partir do 3º byte (índice 2, R) do 2º píxel da imagem da esquerda.
- E assim por diante conforme é apresentado na Figura 3.

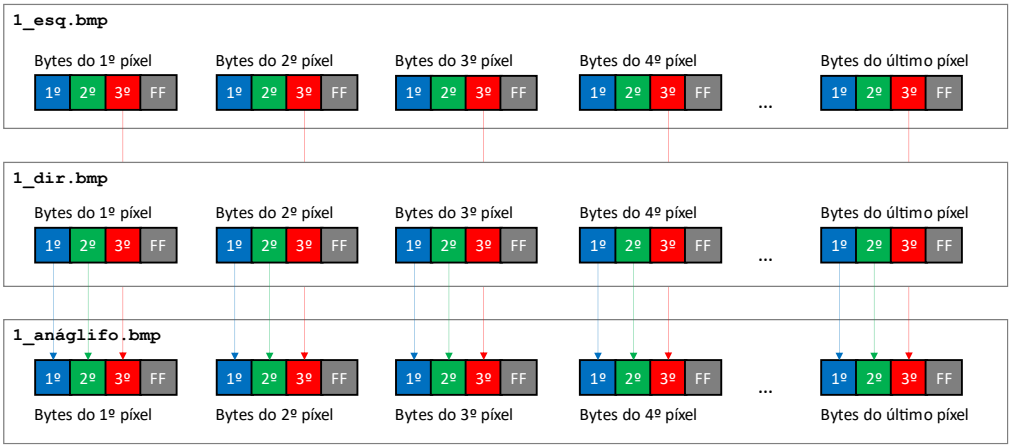

Figura 3: Diagrama para exemplificar a obtenção dos bytes de cor para os píxeis do anáglifo a partir dos bytes de cor dos píxeis das duas imagens do estereograma segundo o algoritmo COLOR.

2.2.3 Algoritmo MONO

A partir das definições iniciais (Secção 2.2.1), a obtenção do anáglifo no algoritmo MONO dá-se através da implementação do seguinte conjunto de equações [Wim22]:

$$\begin{pmatrix} r_a \\ g_a \\ b_a \end{pmatrix} = \begin{pmatrix} 0.299 & 0.587 & 0.114 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{pmatrix} \cdot \begin{pmatrix} r_e \\ g_e \\ b_e \end{pmatrix} + \begin{pmatrix} 0.0 & 0.0 & 0.0 \\ 0.299 & 0.587 & 0.114 \\ 0.299 & 0.587 & 0.114 \end{pmatrix} \cdot \begin{pmatrix} r_d \\ g_d \\ b_d \end{pmatrix}$$

Novamente, ao ignorar os escalares 0.0 das equações, obter-se-á um sistema simplificado como o que segue:

$$\begin{aligned} r_a &= 0.299 \times r_e + 0.587 \times g_e + 0.114 \times b_e \\ g_a &= 0.299 \times r_d + 0.587 \times g_d + 0.114 \times b_d \\ b_a &= 0.299 \times r_d + 0.587 \times g_d + 0.114 \times b_d \end{aligned}$$

Este algoritmo é um pouco mais complexo que o primeiro pois envolve multiplicações por escalares em vírgula flutuante diferentes de 1.0. O anáglifo resultante deste algoritmo tem a particularidade de minimizar ou até mesmo eliminar a reprodução de cores no efeito tridimensional, passando a apresentar uma imagem 3D em escala de cinzas.

No algoritmo MONO é necessário ter ainda mais atenção com a ordem em que os bytes das cores dos píxeis do anáglifo são obtidos a partir das imagens (esquerda e direita) do estereograma. A Figura 4 ajuda a ilustrar esta ordem, a qual resulta no seguinte padrão:

- O 1º byte (índice 0, B) do 1º píxel do anáglifo é obtido a partir do somatório dos 3 bytes de cor (índices 0/B, 1/G e 2/R) do 1º píxel da imagem da direita multiplicados pelos devidos escalares (respetivamente 0.114, 0.587 e 0.299).
- O 2º byte (índice 1, G) do 1º píxel do anáglifo é obtido a partir do somatório dos 3 bytes de cor (índices 0/B, 1/G e 2/R) do 1º píxel da imagem da direita multiplicados pelos devidos escalares (respetivamente 0.114, 0.587 e 0.299).
- O 3º byte (índice 2, R) do 1º píxel do anáglifo é obtido a partir do somatório dos 3 bytes de cor (índices 0/B, 1/G e 2/R) do 1º píxel da imagem da esquerda multiplicados pelos devidos escalares (respetivamente 0.114, 0.587 e 0.299).
- O 4º byte (índice 3, A) do 1º píxel do anáglifo terá o valor FF (sem transparência).
- O 1º byte (índice 0, B) do 2º píxel do anáglifo é obtido a partir do somatório dos 3 bytes de cor (índices 0/B, 1/G e 2/R) do 2º píxel da imagem da direita multiplicados pelos devidos escalares (respetivamente 0.114, 0.587 e 0.299).
- O 2º byte (índice 1, G) do 2º píxel do anáglifo é obtido a partir do somatório dos 3 bytes de cor (índices 0/B, 1/G e 2/R) do 2º píxel da imagem da direita multiplicados pelos devidos escalares (respetivamente 0.114, 0.587 e 0.299).
- O 3º byte (índice 2, R) do 2º píxel do anáglifo é obtido a partir do somatório dos 3 bytes de cor (índices 0/B, 1/G e 2/R) do 2º píxel da imagem da esquerda multiplicados pelos devidos escalares (respetivamente 0.114, 0.587 e 0.299).
- E assim por diante conforme a Figura 4.

É de se notar alguns aspetos importantes para facilitar o entendimento e a implementação do algoritmo MONO:

1. O somatório dos escalares multiplicativos em vírgula flutuante resulta no valor 1.0, pois $0.299 + 0.587 + 0.114 = 1.0$, o que leva a que o resultado de cada byte de cor do anáglifo continue a ser um valor entre 0 e 255.
2. Multiplicar os bytes de cor por um escalar em vírgula flutuante pode resultar em números com algumas casas decimais. Não esqueça de converter o resultado em vírgula flutuante para um byte em número natural (entre 0 e 255) para os bytes de cor do anáglifo resultante.
3. Os bytes de cor g_a e b_a resultam no mesmo valor pois utilizam os mesmos bytes dos píxeis da imagem da direita e os multiplicam pelos mesmos escalares, pelo que se pode fazer este cálculo apenas uma vez e usar o resultado tanto em g_a como em b_a.

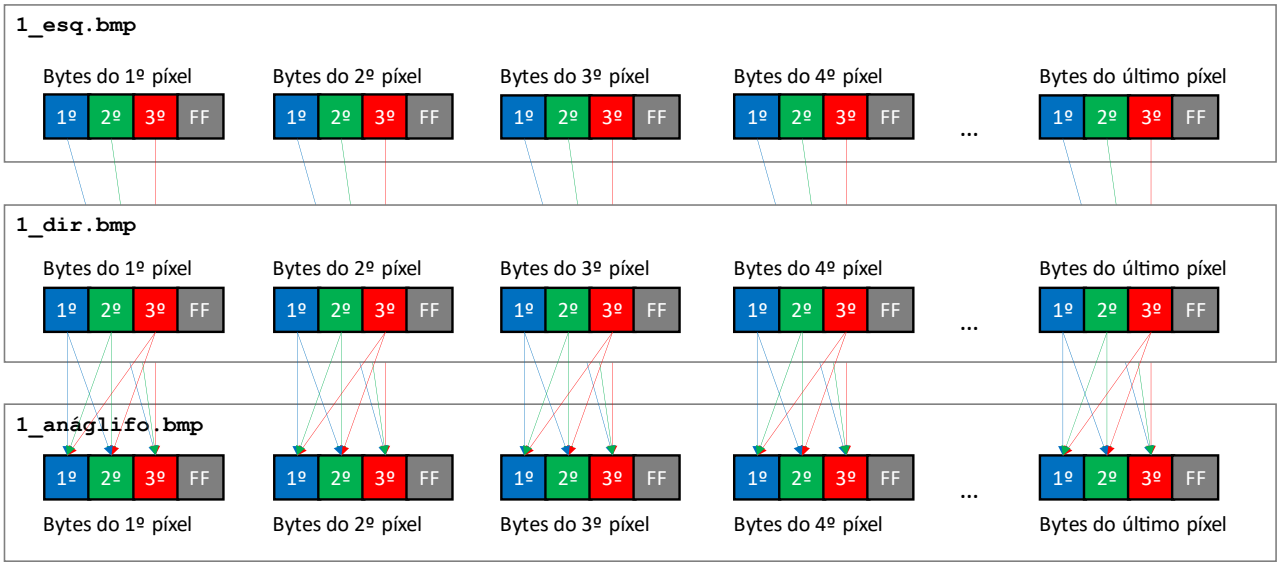

Figura 4: Diagrama para exemplificar a obtenção dos bytes de cor para os píxeis do anáglifo a partir dos bytes de cor dos píxeis das duas imagens do estereograma segundo o algoritmo MONO.

2.3 Ficheiros de apoio ao projeto

Junto ao enunciado, é disponibilizado um ficheiro (***apoio_projeto.tar.gz***) com diversos ficheiros BMP e uma biblioteca de funções para auxiliar a implementação do projeto:

- **Ficheiro básico:** O ficheiro ***básico.bmp*** serve para auxiliar a compreensão da estrutura dos ficheiros BMP e auxiliar nos estágios iniciais da implementação, pois possuem uma complexidade baixa ao nível de cores. Este ficheiro possui um tamanho de `10×10`*px*, onde todos os píxeis possuem a cor *Lapis Lazuli* sem transparência (i.e., `0xFF336699`), pois esta facilita a visualização dos componentes das cores dos seus píxeis.
- **Ficheiros de anáglifos prontos (em JPG):** Os ficheiros ***anaglifo_pronto_X.jpg***, onde ***X*** vai de ***a*** a ***n***, servem apenas como bons exemplos de anáglifos prontos para serem observados tridimensionalmente através dos óculos com os filtros (lentes) vermelho e azul. Além disso, esta diretoria possui um ficheiro que permite perceber o conceito de rivalidade binocular.
- **Ficheiros de estereogramas (em BMP):** Estes ficheiros servem para auxiliar a implementação dos programas e realizar testes para as fases da implementação (as quais serão descritas na Secção 3.1). São disponibilizados dois conjuntos de ficheiros. Cada conjunto possui três ficheiros (um obtido pela câmara esquerda, outro pela câmara direita e outro com um exemplo do anáglifo resultante a partir de um algoritmo desconhecido). O objetivo é os alunos utilizarem os dois ficheiros das câmaras estereoscópicas para tentar construir um anáglifo semelhante (não necessariamente idêntico) ao exemplo do conjunto.
- **Biblioteca de funções:** Esta biblioteca contém algumas funções úteis para o desenvolvimento do projeto. As principais funções implementadas nela incluem a leitura de um ficheiro de imagem BMP para a memória (`readImageFile`) e a escrita de um ficheiro de imagem BMP a partir de um buffer (i.e., espaço) na memória (`writeImageFile`). Algumas funções secundárias também são disponibilizadas, nomeadamente uma para finalizar a execução do programa (`terminate`) e outra para escrever uma *string* no terminal de saída (`printStrLn`).

Para além destes ficheiros, os alunos podem futuramente criar ou utilizar quaisquer outras imagens BMP com o código implementado, desde que a especificação utilizada nos ficheiros BMP seja a ARGB32 (conforme mencionado na Secção 2.1). O código pode ainda ser modificado para aceitar outras especificações, mas isto é um trabalho fora do escopo deste projeto.

2.4 Visualização de uma imagem BMP com o comando `hexdump`

Pode-se utilizar o comando `hexdump` para visualizar o conteúdo binário (em hexadecimal, octal, decimal ou ASCII) de um ficheiro de imagem BMP (p. ex., o ficheiro ***basico.bmp*** descrito na Secção 2.1). A seguir são apresentados alguns exemplos de comandos `hexdump` que serão úteis para o desenvolvimento deste projeto:

1. Imprimir todo o conteúdo de um ficheiro, byte a byte, onde cada byte é apresentado com 2 dígitos hexadecimais na ordem em que os bytes são armazenados em arquiteturas *little-endian*:

```
$ hexdump -v -e '1/1 "%02X "' basico.bmp
42 4D 1A 02 00 00 00 00 00 00 8A 00 00 00 7C 00 00 00 0A 00 00 00 0A 00 00 00 01
00 20 00 03 00 00 00 90 01 00 00 23 2E 00 00 23 2E 00 00 00 00 00 00 00 00 00 00
00 00 FF 00 00 FF 00 00 FF 00 00 00 00 00 00 FF 42 47 52 73 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 02 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 99 66 33 FF 99 66 33 FF 99 66 33 FF …
```

2. Imprimir o tamanho (em bytes) do ficheiro BMP, o qual está guardado no seu cabeçalho. Esta informação está armazenada nos 4 bytes a partir do 3º byte do ficheiro. Para visualizá-la, basta saltar 2 bytes (com o argumento `-s`, de *skip*) e definir o tamanho da leitura para 4 bytes (com o argumento `-n`):

```
$ hexdump -v -s 2 -n 4 -e '1/4 "Tamanho (em hexadecimal): %08X\n"' basico.bmp
Tamanho (em hexadecimal): 0000021A
$ hexdump -v -s 2 -n 4 -e '1/4 "Tamanho (em decimal): %d\n"' basico.bmp
Tamanho (em decimal): 538
```

3. Pode-se validar o tamanho obtido no comando anterior através do comando `du`:

```
$ du -b basico.bmp
538
```

4. Imprimir o deslocamento (*offset*) para o início da secção dos píxeis da imagem (i.e., o fim do cabeçalho) no ficheiro BMP, o qual também está presente no cabeçalho do mesmo. Esta informação está armazenada nos 4 bytes a partir do 11º byte do ficheiro. Para visualizá-la, basta saltar 10 bytes (i.e., `-s 10`) e definir o tamanho da leitura para 4 bytes (i.e., `-n 4`):

```
$ hexdump -v -s 10 -n 4 -e '1/4 "Offset (em hexadecimal): %08X\n"' basico.bmp
Offset (em hexadecimal): 0000008A
$ hexdump -v -s 10 -n 4 -e '1/4 "Offset (em decimal): %d\n"' basico.bmp
Offset (em decimal): 138
```

5. Imprimir somente os píxeis do ficheiro de imagem BMP, a cada 4 bytes (i.e., 32 bits) no formato **ARGB**. Note que agora o salto (`-s`) deve ser igual ao tamanho do *offset* obtido com o comando anterior e que a ordem dos dados apresentados difere da ordem em que eles estão armazenados (ver o primeiro exemplo do comando `hexdump`):

```
$ hexdump -v -s 138 -e '1/4 "%02X "' basico.bmp
FF336699 FF336699 FF336699 …
```

3 Implementação

3.1 Fases da implementação

A entrega do projeto consiste numa única data (ver Secção 3.1), porém recomenda-se que a sua implementação seja feita em três fases separadas:

1) **Tratamento dos argumentos:** A primeira fase consiste em começar por implementar o programa ***fcXXXXX_Anaglifo.asm***, onde XXXXX é o número do aluno, para que este receba os argumentos passados pela linha de comandos (ver Secção 3.2) e os interprete apropriadamente.

2) **Algoritmo COLOR:** A segunda fase consiste em começar por implementar o algoritmo COLOR no programa ***fcXXXXX_Anaglifo.asm***. Este é o algoritmo mais simples e permitirá aos alunos contextualizarem-se melhor com o formato BMP e com algumas funcionalidades que serão úteis também na fase seguinte. Como descrito na Secção 2.3, é vos dado dois conjuntos de ficheiros BMP com imagens da esquerda e da direita do estereograma, as quais o vosso programa deverá conseguir utilizar como ficheiros de entrada do vosso programa (como será descrito na Secção 3.2).
3) **Algoritmo MONO:** A terceira fase consiste em complementar o mesmo programa ***fcXXXXX_Anaglifo.asm***, com o segundo algoritmo (MONO) pois este é mais complexo que o anterior.

3.2 Detalhes de compilação e execução

- A biblioteca de funções auxiliares ***Biblioteca.asm*** fornecida (ver Secção 2.3) deve ser compilada com o seguinte comando:
```
$ nasm -F dwarf -f elf64 Biblioteca.asm
```
- O programa ***fcXXXXX_Anaglifo.asm*** passará pelo *Assembler* `nasm` e será ligado à biblioteca de funções auxiliares fornecida através dos seguintes comandos:
```
$ nasm -F dwarf -f elf64 fcXXXXX_Anaglifo.asm
$ ld fcXXXXX_Anaglifo.o Biblioteca.o -o Anaglifo
```
- O objeto resultante dos comandos anteriores será executado através de um dos dois comandos a seguir, onde o primeiro argumento indica qual o algoritmo a usar (C para COLOR ou M para MONO) pelo programa e os três argumentos seguintes indicam os nomes dos ficheiros a usar pelo programa. Os primeiros dois nomes de ficheiros serão usados como entradas do programa que deverão conter, respetivamente, as imagens, esquerda e direita, do estereograma, enquanto o terceiro nome será usado como saída que deverá conter o anáglifo resultante.
  Os dois comandos possíveis são:
```
$ ./Anaglifo C 1_esq.bmp 1_dir.bmp 1_anaglifo.bmp
```
  ou
```
$ ./Anaglifo M 2_esq.bmp 2_dir.bmp 2_anaglifo.bmp
```

Note que os comandos apresentados nesta secção assumem que todos os ficheiros de código *Assembly*, os ficheiros das imagens BMP e a biblioteca fornecida devem estar todos na mesma diretoria para que estes funcionem corretamente. Pode haver casos (p. ex., *debug* com o SASM) em que é necessário passar o caminho absoluto de cada ficheiro (p. ex., os caminhos começados com o caractere "/" como em "`/home/aluno-di/ASC/projeto/01_esq.bmp`") como argumentos. **O não cumprimento da ordem e número dos argumentos descritos nesta secção implica no não funcionamento do programa e invalidará o projeto**.

3.3 Detalhes e sugestões para a implementação

Muitas das funcionalidades a serem implementadas já foram tema das aulas de ASC ou são abordadas nas referências bibliográficas da disciplina. Seguem alguns exemplos:

- A validação e obtenção dos argumentos passados a um programa: Permitirá receber, através da linha de comandos, o algoritmo de anáglifo (MONO ou COLOR) e o caminho para os ficheiros a serem utilizados pelos programas. Se os parâmetros não estiverem de acordo com o esperado então o programa deve escrever uma mensagem de informação para a consola e terminar. Este tema foi abordado nos Exercícios 3 do [ASC14] e 2 do [ASC15];
- A leitura de ficheiros: Permitirá ler as imagens BMP (p. ex., as imagens esquerda e direita do estereograma). Este tema é abordado pela Secção 13.8.2 do livro [Jor20]. Note que na parte final da secção "*Read from file*" do código deste exemplo, o registo *rax* guarda na verdade a quantidade de bytes que foram lidos do ficheiro.

- A escrita de ficheiros: Permitirá escrever a imagem BMP gerada com o anáglifo. Este tema é abordado pela Secção 13.8.1 do livro [Jor20];
- Encontrar o tamanho (*size*) e o deslocamento (*offset*) nos ficheiros BMP: Permitirá saber onde começa e onde termina a secção dos píxeis nestes ficheiros. Ver a Secção 2.1.
- Percorrer a memória byte-a-byte: Permitirá obter separadamente cada byte da mensagem ou cada byte dos píxeis da imagem BMP. Este tema foi abordado em diversos exercícios e programas com exemplos de ciclos com endereçamento indexado.
- A conversão entre números naturais e em vírgula flutuante servirá para atribuir os valores dos bytes de cores dos píxeis do anáglifo, a qual foi abordada no Exercício 2 do [ASC11];
- A depuração de código com o GDB auxilia a compreensão do código durante o desenvolvimento, cujo tema foi abordado no [ASC13].

4 Entrega do trabalho

4.1 Data da entrega

O trabalho é **individual** e deverá ser entregue até às **23:59** do dia **23/12/2022**.

4.2 Formato da entrega

A entrega final será feita através da página da disciplina no Moodle, antes do limite do período de entrega, em local assinalado para tal (são permitidas ilimitadas submissões antes do deadline). Caso os alunos não sigam exatamente as regras especificadas a seguir, serão penalizados na nota:

- O ficheiro *Assembly* deve ser gravado exatamente com o nome ***fcXXXXX_Anaglifo.asm***, onde XXXXX é o número do aluno.
- Os alunos devem incluir, no início do ficheiro *Assembly* entregue, uma linha de comentário com o seu número de aluno (exatamente "`; fcXXXXX`").
- O ficheiro *Assembly* deve ser colocado em um ficheiro comprimido ZIP ou TAR.GZ com o número de aluno como o nome do ficheiro (exatamente ***fcXXXXX.zip*** ou ***fcXXXXX.tar.gz***).
- O ficheiro comprimido (***fcXXXXX.zip*** ou ***fcXXXXX.tar.gz***) deve conter **apenas** o ficheiro *Assembly* implementado pelo aluno e **não** deve incluir o ficheiro ***Biblioteca.asm*** fornecido.

4.3 Critérios de avaliação

1. Compilação do código e ligação das bibliotecas a funcionar sem erros. Serão utilizados os comandos apresentados na Secção 3.2 e outros semelhantes.
2. O código realizar a validação do número de argumentos passados ao programa e utilizar corretamente cada argumento na ordem especificada na Secção 3.2.
3. Funcionamento correto do programa *Anaglifo* conforme especificado neste enunciado. Serão executados os comandos apresentados na Secção 3.2 com os ficheiros de teste, para além de outros ficheiros BMP.
4. Conformidade estrita com o formato de entrega descrito na Secção 4.2.
5. Organização dos códigos em *Assembly*. Aspetos que serão valorizados incluem respeitar as convenções de chamada de funções *System V ABI*, aplicar boas práticas de uso da memória, evitar referências à memória desnecessárias, etc.
6. Documentação dos códigos em *Assembly*. Aspetos que serão valorizados incluem comentar o funcionamento de trechos de códigos mais complexos e não-triviais.

As cotações de cada critério apresentado não serão facultadas, porém naturalmente o Critério 3 terá o maior peso. Trabalhos vazios para cumprir apenas com o Critério 1 terão nota zero.

4.4 Plágio

Não é permitido aos alunos partilharem códigos com soluções, ainda que parciais, de nenhuma parte do projeto com outros alunos (nem através do Fórum da disciplina, nem por qualquer outro meio). Além disso, todos os códigos serão testados por um verificador de plágio. Caso alguma irregularidade seja encontrada, os projetos de todos os alunos envolvidos serão anulados e o caso será reportado aos órgãos responsáveis na Ciências@ULisboa.

Por fim, é responsabilidade de cada aluno garantir que a sua *home*, as suas diretorias e os seus ficheiros de código estão protegidos contra a leitura de outras pessoas (que não o utilizador dono dos mesmos). Por exemplo, se os ficheiros estiverem gravados na sua área de aluno nos servidores da Ciências@ULisboa, então todos os itens mencionados anteriormente devem ter as permissões de acesso 700 (ver o último item da Secção 1.3 do Guião [ASC1]).

5 Bibliografia

[ASC1] M. Calha, V. Cogo., S. Signorello. (2022). Guião de Laboratório – Linux. Disponível online em https://moodle.ciencias.ulisboa.pt/mod/resource/view.php?id=200123.

[ASC11] M. Calha, V. Cogo., S. Signorello. (2022). Guião de laboratório – x86: Operações. Disponível online em https://moodle.ciencias.ulisboa.pt/mod/resource/view.php?id=203211.

[ASC13] M. Calha, V. Cogo., S. Signorello. (2022). Guião de GDB. Disponível online em https://moodle.ciencias.ulisboa.pt/mod/resource/view.php?id=203266.

[ASC14] M. Calha, V. Cogo., S. Signorello. (2022). Ficha de exercícios – x86: Pilha. Disponível online em https://moodle.ciencias.ulisboa.pt/mod/resource/view.php?id=203867.

[ASC15] M. Calha, V. Cogo., S. Signorello. (2022). Guião de laboratório – x86: Pilha. Disponível online em https://moodle.ciencias.ulisboa.pt/mod/resource/view.php?id=204064.

[Iza22] Izawa, Yu et al. "Stereogram of the Living Heart, Lung, and Adjacent Structures." Tomography (Ann Arbor, Mich.) vol. 8,2 824-841. Mar. 2022, doi:10.3390/tomography8020068. Disponível online em https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8938811/.

[Jor20] Ed Jorgensen. (2020). x86-64 Assembly Language Programming with Ubuntu. Disponível online em http://www.egr.unlv.edu/~ed/x86.html.

[NASA22a] National Aeronautics and Space Administration (NASA). (2022) "Mars Curiosity Rover". Disponível online em https://mars.nasa.gov/msl/home/.

[NASA22b] National Aeronautics and Space Administration (NASA). (2022) "Mast Camera". Disponível online em https://mars.nasa.gov/msl/spacecraft/instruments/mastcam/.

[Wim22] Peter Wimmer. (2022) "Anaglyph Methods Comparison". Disponível online em https://3dtv.at/knowhow/anaglyphcomparison_en.aspx.

Molde para os óculos:

Os alunos receberão nas aulas teóricas um pedaço de papel celofane azul e outro vermelho para aplicar nas lentes do molde abaixo.