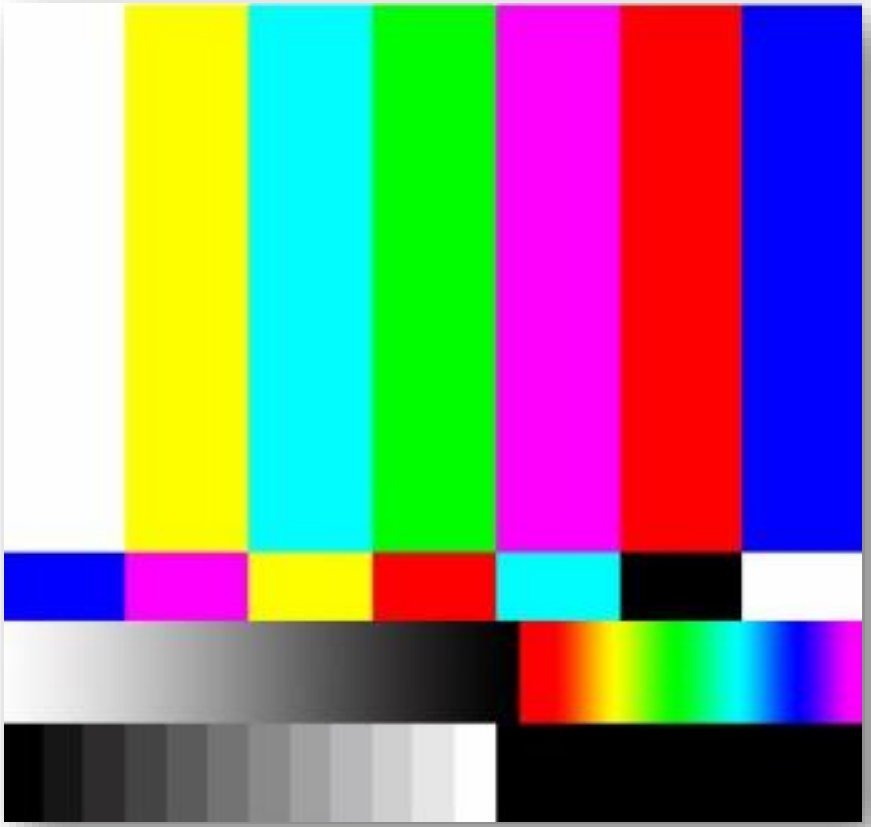
VISÃO POR COMPUTADOR

Engenharia de Sistemas Informáticos

Duarte Duque | dduque@ipca.pt

IPCA
INSTITUTO POLITÉCNICO DO CÁVADO E DO AVE
ESCOLA SUPERIOR DE TECNOLOGIA

- **Espaços de Cor**



- **Cor**

Os sinais medidos pelas células sensíveis à cor (cones do olho humano) ou pelos sensores sensíveis a diferentes radiações de um circuito CCD, ou CMOS, são combinados (no cérebro ou numa unidade de processamento) de modo a fornecer diferentes sensações de cor.

Vejamos as principais **propriedades da cor**:

- **Brilho (Brightness):**

Propriedade relacionada com a intensidade da energia electromagnética.
Esta relacionada com a sensação de maior ou menor intensidade de luz.

- **Tonalidade (Hue):**

Propriedade que corresponde ao comprimento de onda predominante no espectro resultante da interação entre a fonte luminosa e o objecto.
Está relacionada com a sensação de similaridade a uma cor.

- **Cor**

- **Saturação (Saturation):**

Propriedade determinada pela combinação da intensidade de luz, e de como essa intensidade é distribuída pelo espectro de cores.

As **cores** mais "puras", isto é, **mais saturadas**, são obtidas quando a intensidade de luz recebida se concentra apenas num determinado comprimento de onda (como acontece, por exemplo, numa luz laser).

Espaços de Cor

- **O que é um espaço de Cor?**

Um **espaço de cor é um método** pelo qual se torna possível **especificar**, **criar** ou **visualizar cor**.

No monitor de um computador a cor de um pixel pode, por exemplo, ser descrita pela quantidade de energia emitida pelos leds vermelho, verde e azul.

A **cor** é usualmente **especificada utilizando três parâmetros** (ou coordenadas).

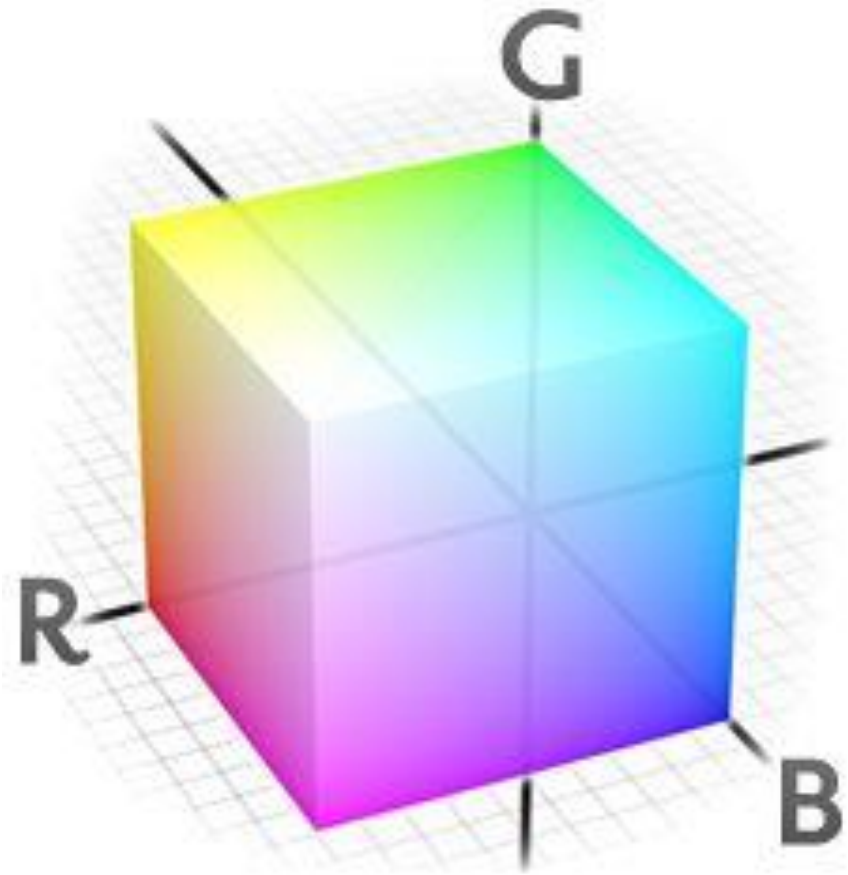
Estes parâmetros descrevem a **posição da cor** no **espaço de cor utilizado**.

Duarte Duque | dduque@ipca.pt

IPCA
INSTITUTO POLITÉCNICO DO CÁVADO E DO AVE
ESCOLA SUPERIOR DE TECNOLOGIA

Espaços de Cor

- **Espaço de Cor RGB**



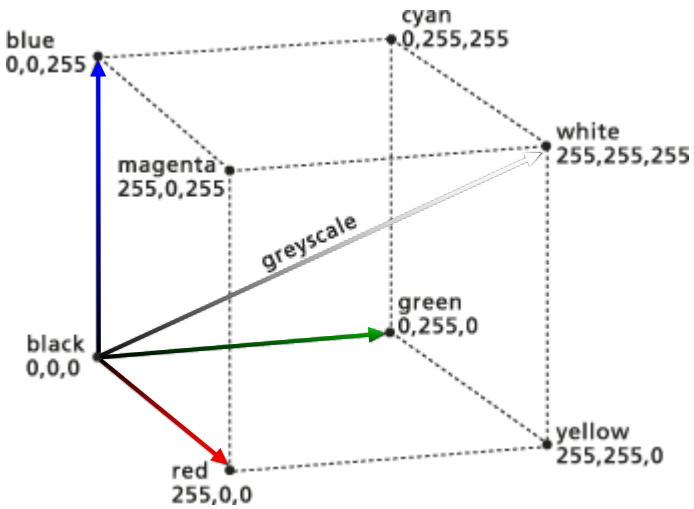
Duarte Duque | dduque@ipca.pt

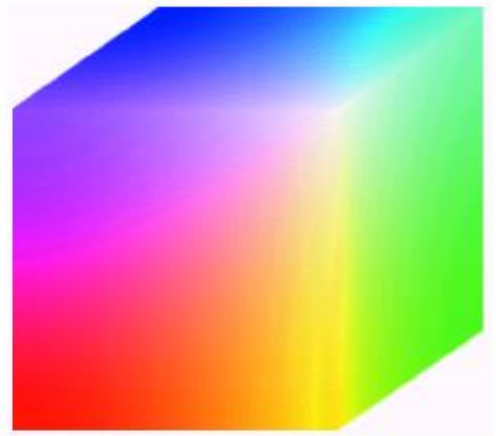
IPCA
INSTITUTO POLITÉCNICO DO CÁVADO E DO AVE
ESCOLA SUPERIOR DE TECNOLOGIA

• Espaço de Cor RGB

O espaço de cor **RGB** faz uso de três cores primárias (vermelho, verde e azul) para produzir o conjunto de cores visíveis.

É um **sistema de cores aditivo**, baseado na teoria tricromática.





• Espaço de Cor RGB

O espaço de cor **RGB** é assim constituído por 3 componentes:

- **Vermelho** (**Red**);
- **Verde** (**Green**);
- **Azul** (**Blue**).

Numa profundidade de cor de **24bpp**, cada um dos três componentes **R**, **G**, e **B** é definida por uma variável de **8 bits**, que pode tomar valores entre **0** e **255**.

O valor **0** (zero) indica que a referida componente de cor não tem qualquer contribuição para a representação da cor final. O valor **255** indica que a contribuição dessa componente será máxima.

• Espaço de Cor RGB

Assim, os valores:

- **0,0,0** representam a cor **preto**;
- **255,0,0** representam a cor **vermelho**;
- **0,255,0** representam a cor **verde**;
- **0,0,255** representam a cor **azul**;
- **255,255,255** representam a cor **branco**;
- **127,127,127** representam uma das tonalidades de **cinzento**.

VISÃO POR COMPUTADOR

Espaços de Cor

- **Espaço de Cor RGB**

R	G	B	Cor
0	0	0	Preto
255	0	0	Vermelho
0	255	0	Verde
0	0	255	Azul
255	255	0	Amarelo
255	0	255	Magenta
0	255	255	Ciano
64	64	64	Cinzento Escuro
220	220	220	Cinzento Claro

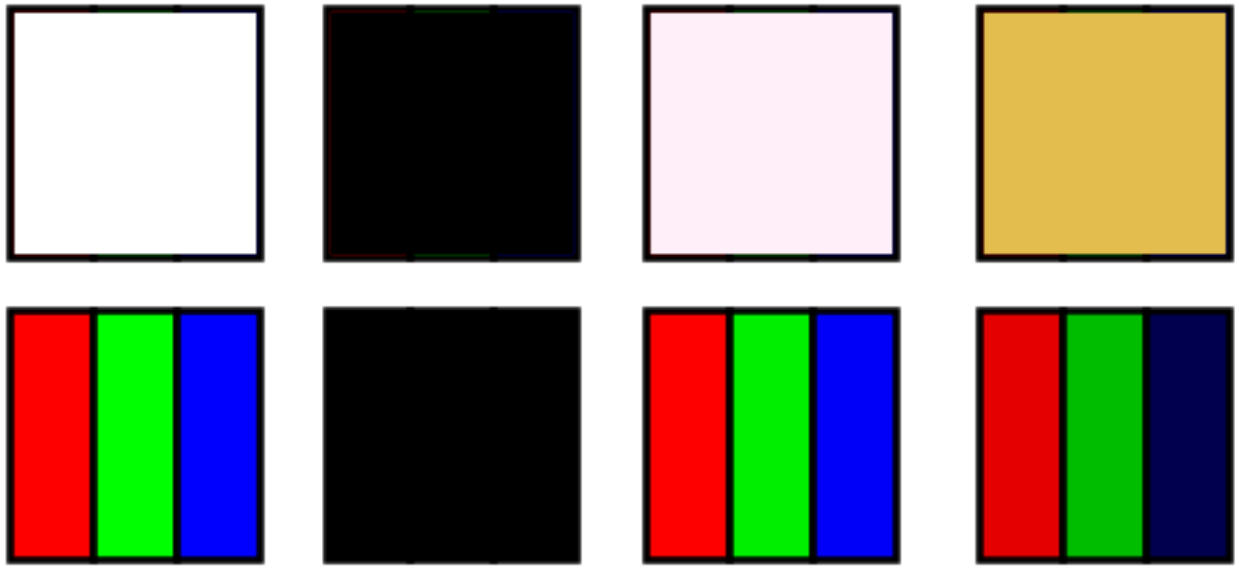
Duarte Duque | dduque@ipca.pt

IPCA
INSTITUTO POLITÉCNICO DO CÁVADO E DO AVE
ESCOLA SUPERIOR DE TECNOLOGIA

VISÃO POR COMPUTADOR

Espaços de Cor

- **Espaço de Cor RGB**



RGB
255, 255, 255

RGB
0, 0, 0

RGB
255, 239, 248

RGB
228, 189, 79

Duarte Duque | dduque@ipca.pt

IPCA
INSTITUTO POLITÉCNICO DO CÁVADO E DO AVE
ESCOLA SUPERIOR DE TECNOLOGIA

- **Aritmética no Espaço de Cor RGB**

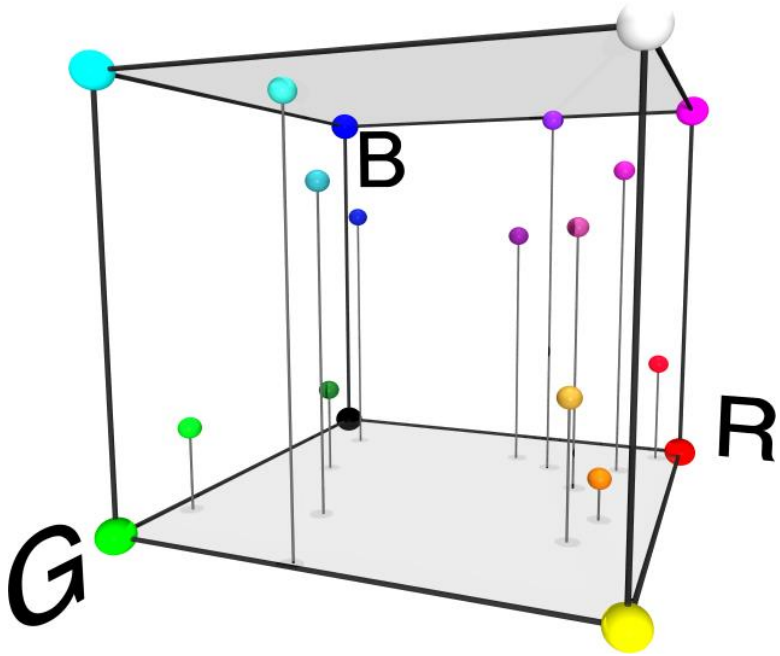
É possível efectuar diversos efeitos de cor através de operações sobre os valores RGB:

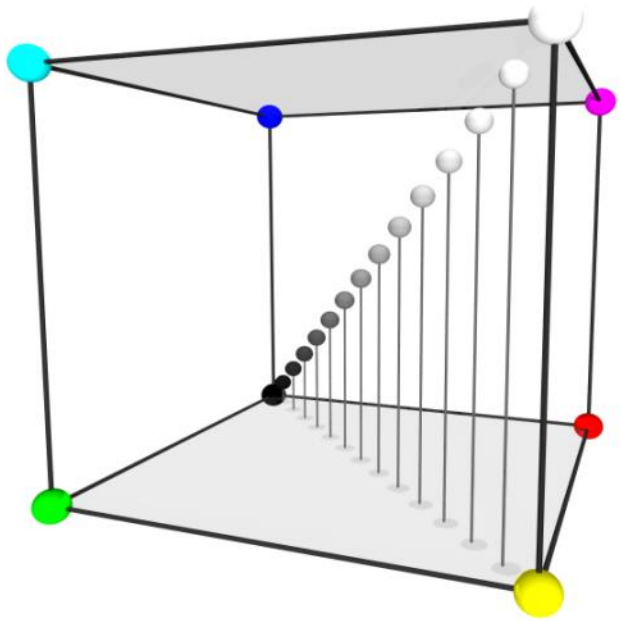
Operação	Fórmula	Efeito
Negativo	C = 255 – C	Calcula a cor oposta, por exemplo, o preto torna-se branco, o vermelho em ciano, etc.
Escurecer	C = C / P C = C – P	Divide a componente de cor por uma qualquer constante (maior que 1), ou subtrai por uma constante, para a tornar mais escura.
Clarear	C = C * P C = C + P	Multiplica a componente de cor por uma qualquer constante (maior que 1), ou adiciona uma constante, para a tornar mais clara.
Cinzento	(R + G + B) / 3	Calcula a média dos três canais (componentes) de cor, de modo a obter uma tonalidade cinzenta.
Remover Canal	R = 0, G = 0 e/ou B = 0	Colocando um ou mais canais a 0 (zero), remove-se completamente a contribuição dessa componente de cor.
Trocar Canais	R = G, G = R, …	Trocar valores entre dois canais. Esta operação altera a cor.

VISÃO POR COMPUTADOR

Espaços de Cor

- **Espaço de Cor RGB**





Duarte Duque | dduque@ipca.pt

IPCA
INSTITUTO POLITÉCNICO DO CÁVADO E DO AVE
ESCOLA SUPERIOR DE TECNOLOGIA

- **Exercícios:**

- Construa uma função que calcule o negativo de uma imagem Gray.

```
int vc_gray_negative(IVC *srcdst);
```

- Construa uma função que calcule o negativo de uma imagem RGB.

```
int vc_rgb_negative(IVC *srcdst);
```

- **Exercícios:**

```
// Gerar negativo da imagem Gray
int vc_gray_negative(IVC *srcdst)
{
    unsigned char *data = (unsigned char *) srcdst->data;
    int width = srcdst->width;
    int height = srcdst->height;
    int bytesperline = srcdst->width * srcdst->channels;
    int channels = srcdst->channels;
    int x, y;
    long int pos;

    // Verificação de erros
    if((srcdst->width <= 0) || (srcdst->height <= 0) || (srcdst->data == NULL)) return 0;
    if(channels != 1) return 0;

    // Inverte a imagem Gray
    for(y=0; y<height; y++)
    {
        for(x=0; x<width; x++)
        {
            pos = y * bytesperline + x * channels;

            data[pos] = 255 - data[pos];
        }
    }

    return 1;
}
```

- **Exercícios:**

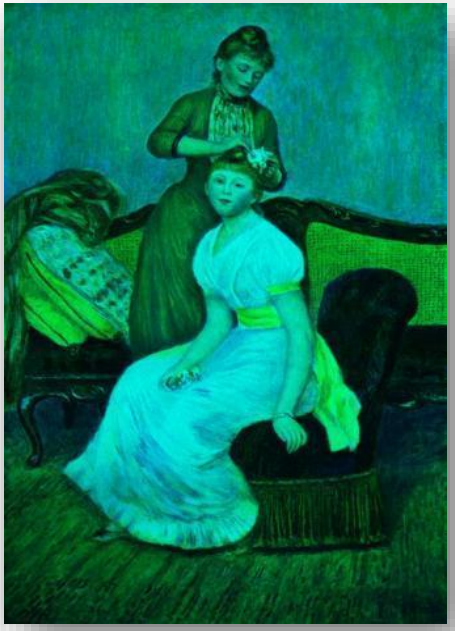
```
// Gerar negativo da imagem RGB
int vc_rgb_negative(IVC *srcdst)
{
    unsigned char *data = (unsigned char *) srcdst->data;
    int width = srcdst->width;
    int height = srcdst->height;
    int bytesperline = srcdst->width * srcdst->channels;
    int channels = srcdst->channels;
    int x, y;
    long int pos;

    // Verificação de erros
    if((srcdst->width <= 0) || (srcdst->height <= 0) || (srcdst->data == NULL)) return 0;
    if(channels != 3) return 0;

    // Inverte a imagem RGB
    for(y=0; y<height; y++)
    {
        for(x=0; x<width; x++)
        {
            pos = y * bytesperline + x * channels;

            data[pos] = 255 - data[pos];
            data[pos + 1] = 255 - data[pos + 1];
            data[pos + 2] = 255 - data[pos + 2];
        }
    }

    return 1;
}
```

VISÃO POR COMPUTADOR

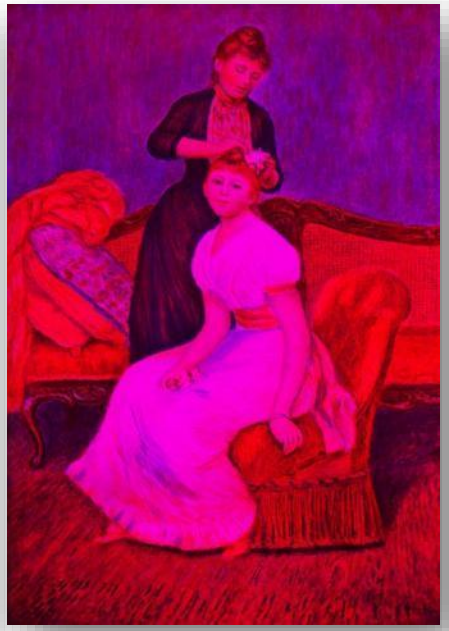
Espaços de Cor

- **Decomposição de uma Imagem RGB**

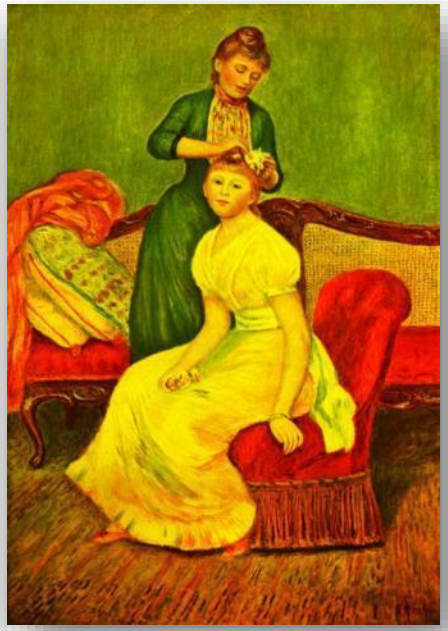
Original



Sem Vermelho



Sem Verde



Sem Azul

Duarte Duque | dduque@ipca.pt

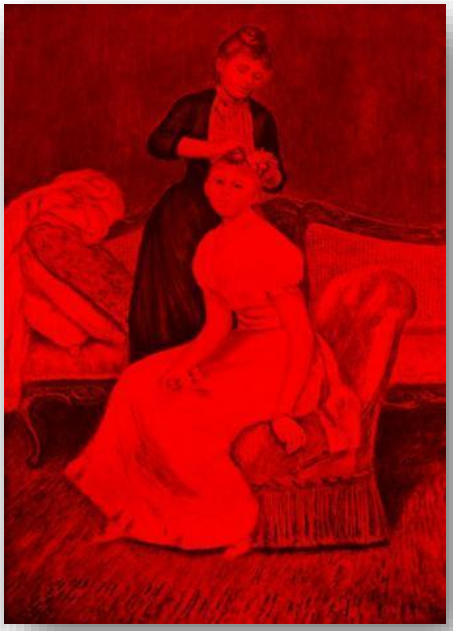
IPCA
INSTITUTO POLITÉCNICO DO CÁVADO E DO AVE
ESCOLA SUPERIOR DE TECNOLOGIA

VISÃO POR COMPUTADOR

Espaços de Cor

- **Decomposição de uma Imagem RGB**

Original



Componente Vermelho

Componente Verde

Componente Azul

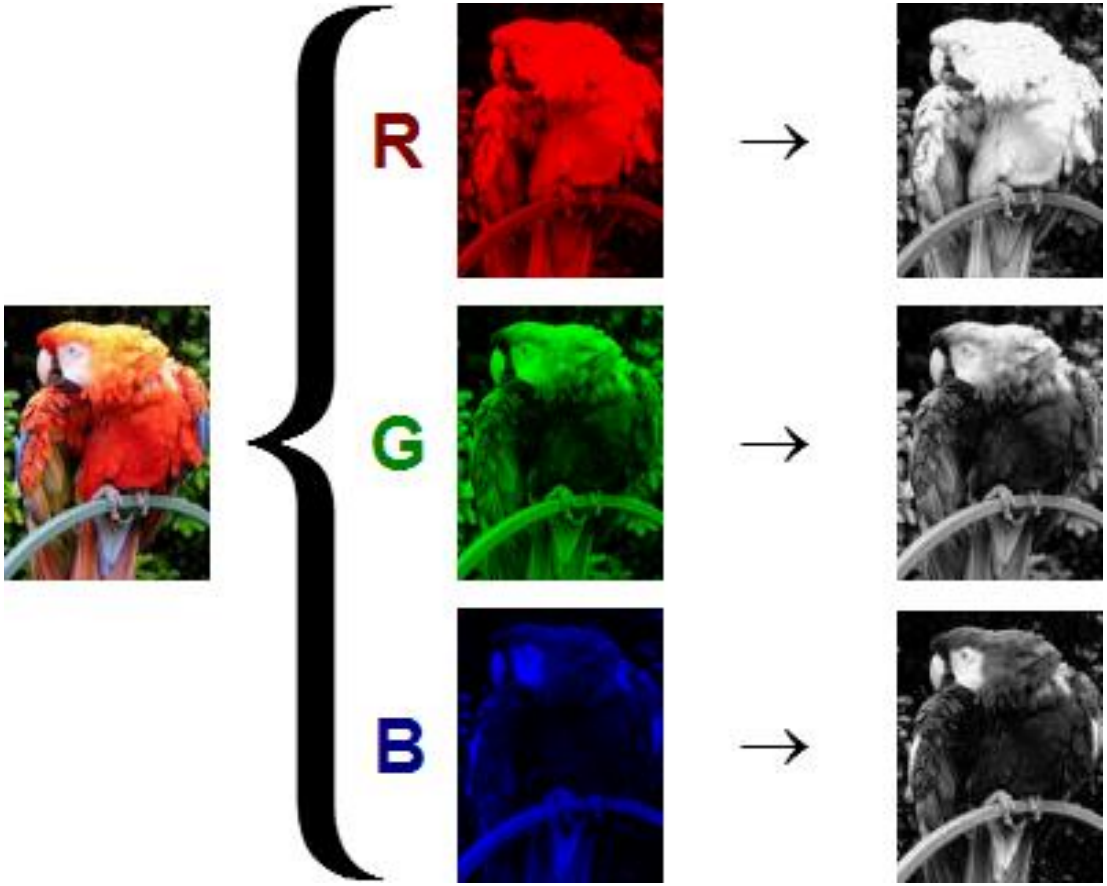
Duarte Duque | dduque@ipca.pt

IPCA
INSTITUTO POLITÉCNICO DO CÁVADO E DO AVE
ESCOLA SUPERIOR DE TECNOLOGIA

VISÃO POR COMPUTADOR

Espaços de Cor

- **Decomposição de uma Imagem RGB**



Duarte Duque | dduque@ipca.pt

IPCA
INSTITUTO POLITÉCNICO DO CÁVADO E DO AVE
ESCOLA SUPERIOR DE TECNOLOGIA

- **Exercícios:**

- Construa funções que extraiam as componentes R, G e B de uma imagem RGB, para imagens em tons de cinzento.

```
int vc_rgb_get_red_gray(IVC *srcdst);
int vc_rgb_get_green_gray(IVC *srcdst);
int vc_rgb_get_blue_gray(IVC *srcdst);
```

- **Exercícios:**

```
// Extrair componente Red da imagem RGB para Gray
int vc_rgb_get_red_gray(IVC *srcdst)
{
    unsigned char *data = (unsigned char *) srcdst->data;
    int width = srcdst->width;
    int height = srcdst->height;
    int bytesperline = srcdst->width * srcdst->channels;
    int channels = srcdst->channels;
    int x, y;
    long int pos;

    // Verificação de erros
    if((srcdst->width <= 0) || (srcdst->height <= 0) || (srcdst->data == NULL)) return 0;
    if(channels != 3) return 0;

    // Extrai a componente Red
    for(y=0; y<height; y++)
    {
        for(x=0; x<width; x++)
        {
            pos = y * bytesperline + x * channels;

            data[pos + 1] = data[pos]; // Green
            data[pos + 2] = data[pos]; // Blue
        }
    }

    return 1;
}
```

- **Intensidade de Luz**

A intensidade de luz é uma função pesada dos valores **R**, **G** e **B**.

O olho humano não percepciona de igual forma cada componente de cor:

$$\textbf{Intensidade} = R * 0.299 + G * 0.587 + B * 0.114$$

Assim, assumindo três fontes de luz com igual intensidade, mas cores distintas (vermelho, verde e azul), a luz verde irá aparecer mais brilhante, seguida pelas componentes vermelha e azul.

Duarte Duque | dduque@ipca.pt

IPCA
INSTITUTO POLITÉCNICO DO CÁVADO E DO AVE
ESCOLA SUPERIOR DE TECNOLOGIA

- **Exercícios:**

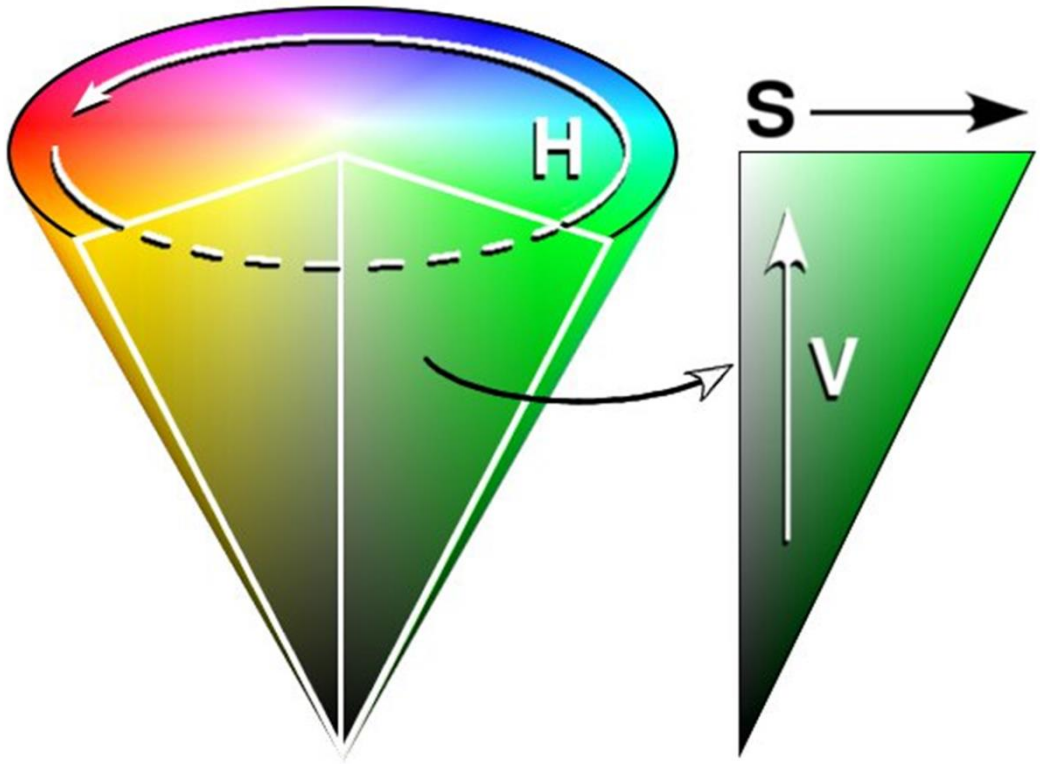
- Construa uma função que converta uma imagem no espaço RGB para uma imagem em tons de cinzento.

```
int vc_rgb_to_gray(IVC *src, IVC *dst);
```

- **Exercícios:**

```
// Converter de RGB para Gray
int vc_rgb_to_gray(IVC *src, IVC *dst)
{
    unsigned char *datasrc = (unsigned char *) src->data;
    int bytesperline_src = src->width * src->channels;
    int channels_src = src->channels;
    unsigned char *datadst = (unsigned char *) dst->data;
    int bytesperline_dst = dst->width * dst->channels;
    int channels_dst = dst->channels;
    int width = src->width;
    int height = src->height;
    int x, y;
    long int pos_src, pos_dst;
    float rf, gf, bf;

    // Verificação de erros
    if((src->width <= 0) || (src->height <= 0) || (src->data == NULL)) return 0;
    if((src->width != dst->width) || (src->height != dst->height)) return 0;
    if((src->channels != 3) || (dst->channels != 1)) return 0;

    for(y=0; y<height; y++)
    {
        for(x=0; x<width; x++)
        {
            pos_src = y * bytesperline_src + x * channels_src;
            pos_dst = y * bytesperline_dst + x * channels_dst;

            rf = (float) datasrc[pos_src];
            gf = (float) datasrc[pos_src + 1];
            bf = (float) datasrc[pos_src + 2];

            datadst[pos_dst] = (unsigned char) ((rf * 0.299) + (gf * 0.587) + (bf * 0.114));
        }
    }

    return 1;
}
```

- **Espaço de Cor HSV**



• Espaço de Cor HSV

O espaço de cor **HSV** proporciona um método intuitivo de especificar a cor.

Neste espaço de cor, cada cor é representada por três componentes:

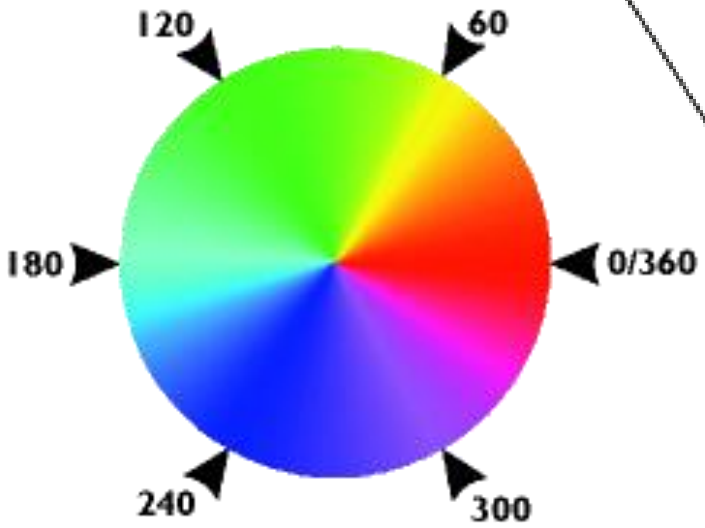
- **Tonalidade ou Matiz** (**Hue**);
- **Saturação** (**Saturation**);
- **Valor** (**Value**).

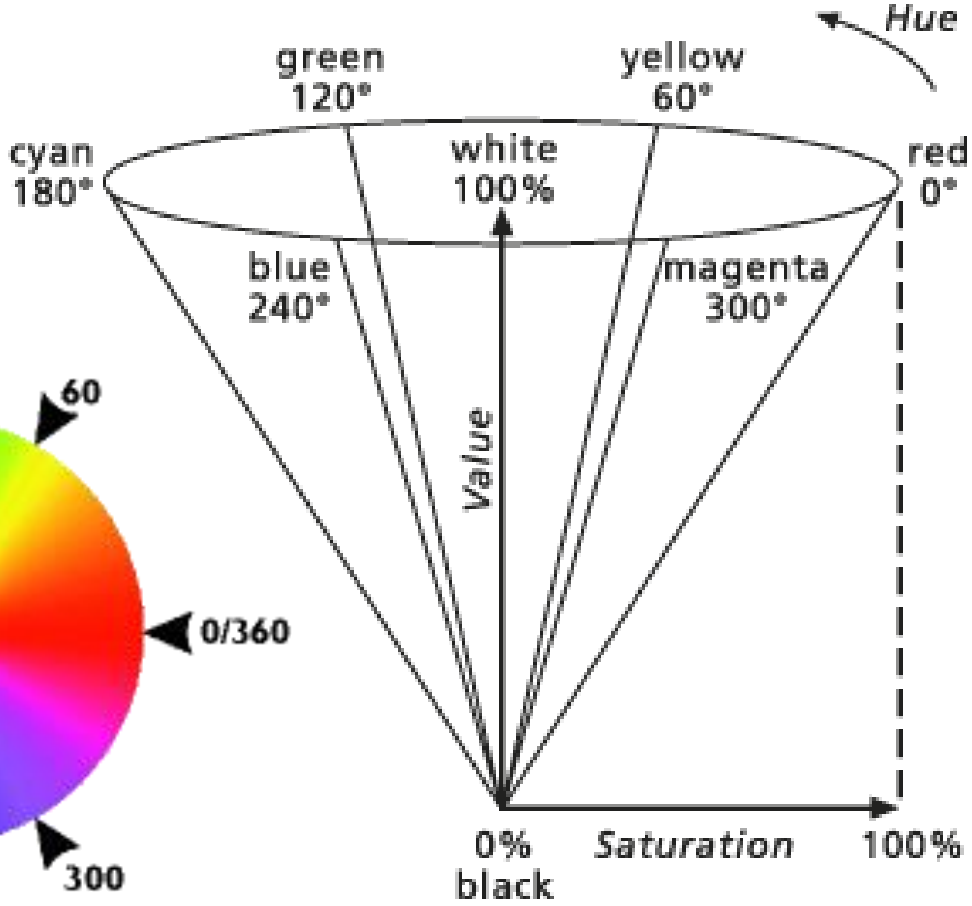
No espaço **HSV** é possível **seleccionar a tonalidade** desejada, e posteriormente **realizar ajustes na saturação e intensidade**.

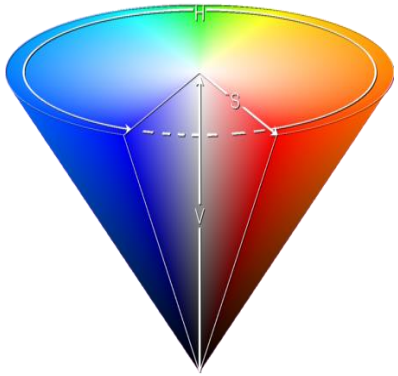
Esta separação entre a componente da luminância (luz) e da crominância (cor) traz vantagens relativamente ao espaço RGB quando se pretende realizar operações sobre cores.

Duarte Duque | dduque@ipca.pt

- **Espaço de Cor HSV**







Duarte Duque | dduque@ipca.pt

• Conversão HSV<->RGB

- **Cálculo da componente Valor** (**Value**):

Sendo:

- R, G, B as componentes vermelho, verde e azul de uma determinada cor;
- 'Max' o maior das três componentes de cor;

Então:

$$Max = MÁXIMO \{R, G, B\}$$

Valor (V no HSV) é o maior de todos os componentes R, G, B:

Valor = Max

[Mais ou menos luz]

- **Conversão HSV<->RGB**

- **Cálculo da componente Saturação** (**Saturation**):

Sendo:

- R, G, B as componentes vermelho, verde e azul de uma determinada cor;
- 'Min' o menor das três componentes de cor;

Então:

$$\text{Min} = \text{MÍNIMO } \{R, G, B\}$$

Saturação (S no HSV) é definido como:

$$\textbf{Saturação} = \textbf{(Max} - \textbf{Min)} / \textbf{Valor}$$

[Mais ou menos cor]

Duarte Duque | dduque@ipca.pt

IPCA
INSTITUTO POLITÉCNICO DO CÁVADO E DO AVE
ESCOLA SUPERIOR DE TECNOLOGIA

- **Conversão HSV<->RGB**

Saturação = (Max – Min) / Valor

Note que esta expressão será **indefinida** quando a componente **Valor for igual a zero**, o que acontece somente na **cor preta**. Neste caso, à Saturação deverá atribuir o número 0 (zero).

Observe também que a **Saturação se torna zero** quando **Max = Min**, que só pode acontecer quando os componentes **vermelho**, **verde** e **azul** do espaço RGB **são todos iguais**, que descreve um **tom de cinza**.

Assim, a Saturação pode ser pensada como o oposto de "cinzenta".

- **Conversão HSV<->RGB**

- **Cálculo da componente Matiz** (**Hue**):

Matiz é definida em casos, dependendo de qual dos componentes vermelho, verde e azul da cor é a maior:

- Quando o verde é o maior componente de cor, Hue cairá entre 60 e 180 (i.e., 120 ± 60);
- Quando o azul é o maior componente de cor, Hue vai cair entre 180 e 300 (i.e., 240 ± 60);
- Quando o vermelho é o maior, Hue será um ângulo entre 300 e 360 ou entre 0 e 60.

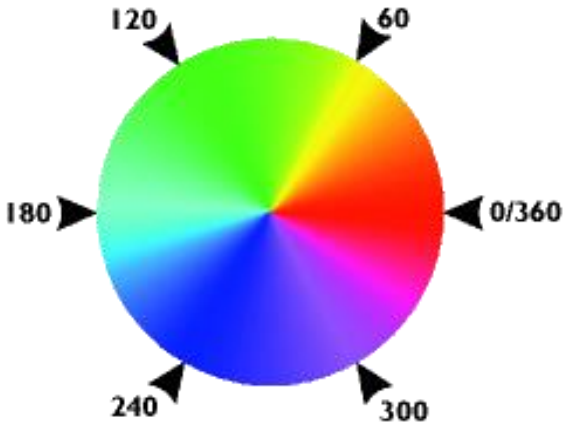
A definição de Hue é a seguinte:

Se, (Max = R) e (G ≥ B), então Hue ← 60 * (G - B) / (Max - Min)

Se, (Max = R) e (B > G), então Hue ← 360 + 60 * (G - B) / (Max - Min)

Se Max = G, então Hue ← 120 + 60 * (B - R) / (Max - Min)

Se Max = B, então Hue ← 240 + 60 * (R - G) / (Max - Min)

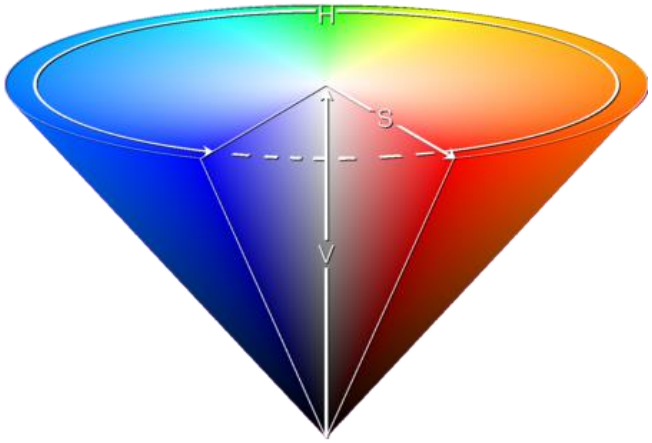


- **Conversão HSV<->RGB**

Tal como acontece com Saturação, a **Matiz será indefinida quando Max = Min**, que descreve um tom de cinza.

Uma cor sobre o eixo central não tem uma posição bem definida sobre o eixo de rotação.

De igual modo ao que acontece com a saturação, **podemos atribuir arbitrariamente à Matiz de cores em tons de cinza** o **número zero**, para evitar variáveis indefinidas.



- **Exercícios:**

- Construa uma função que converta uma imagem no espaço RGB para uma imagem no espaço HSV.

```
int vc_rgb_to_hsv(IVC *src, IVC *dst);
```

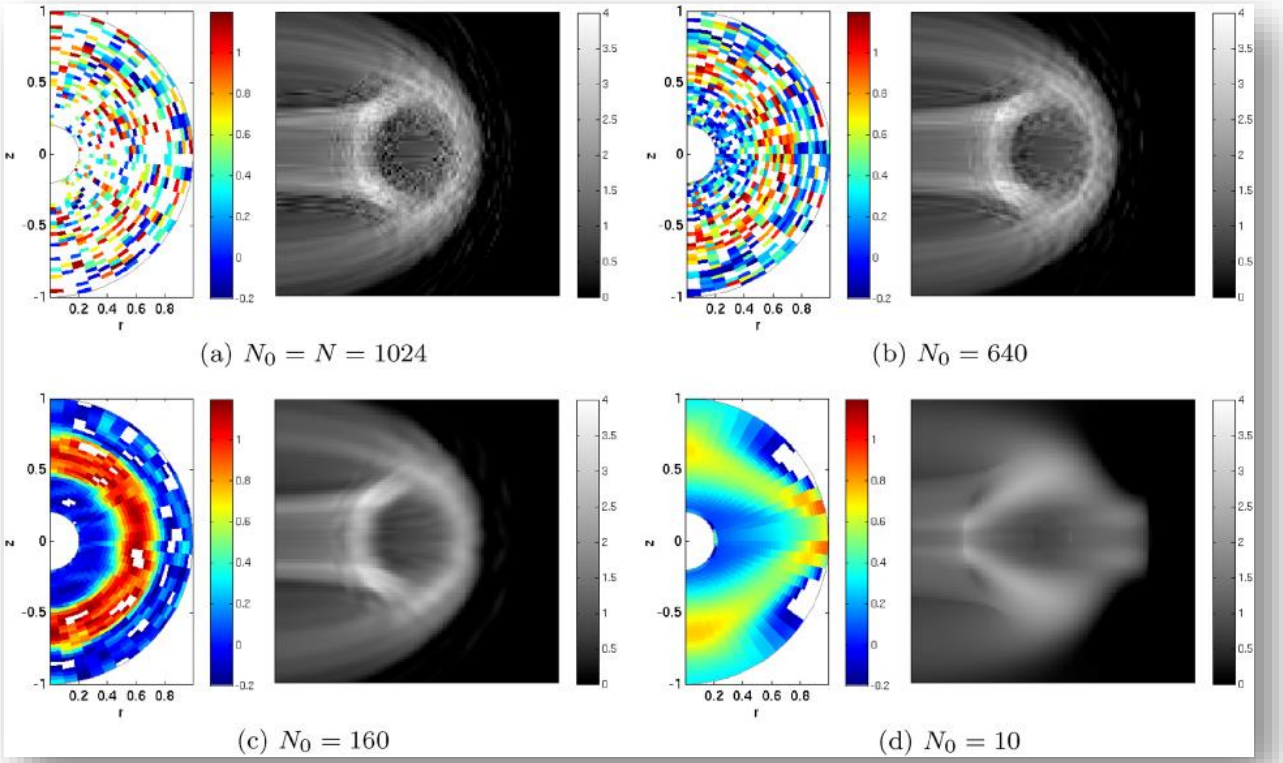
- Construa uma função que receba uma imagem HSV e retorne uma imagem com 1 canal (admitindo valores entre 0 e 255 por pixel). Essa função deverá receber ainda os intervalos de valores da Matiz (H), Saturação (S) e Brilho (V) da cor que se pretende segmentar. A imagem de saída deverá apresentar a branco (255) os pixéis que estão dentro desse intervalo, e a preto (0) todos os outros.

```
int vc_hsv_segmentation(IVC *src, IVC *dst, int hmin, int hmax, int smin,
                        int smax, int vmin, int vmax);
```

VISÃO POR COMPUTADOR

Espaços de Cor

- **Representação de Informação Através de Escalas de Cor**



Duarte Duque | dduque@ipca.pt

IPCA
INSTITUTO POLITÉCNICO DO CÁVADO E DO AVE
ESCOLA SUPERIOR DE TECNOLOGIA

- **Representação de Informação Através de Escalas de Cor**

Em inúmeras situações é necessário **representar graficamente a informação**.

Gráficos de temperatura, altitude, tensão, (entre muitas outras grandezas) são frequentemente apresentados através de **escalas de cor**.

Através de uma imagem em **tons de cinzentos**, esta representação é bastante intuitiva. Ao **valor mínimo** da escala atribui-se o valor **0 (zero)**, enquanto que o **valor máximo** deverá ser representado pelo valor de intensidade **255**.

Duarte Duque | dduque@ipca.pt

IPCA
INSTITUTO POLITÉCNICO DO CÁVADO E DO AVE
ESCOLA SUPERIOR DE TECNOLOGIA

• Representação de Informação Através de Escalas de Cor

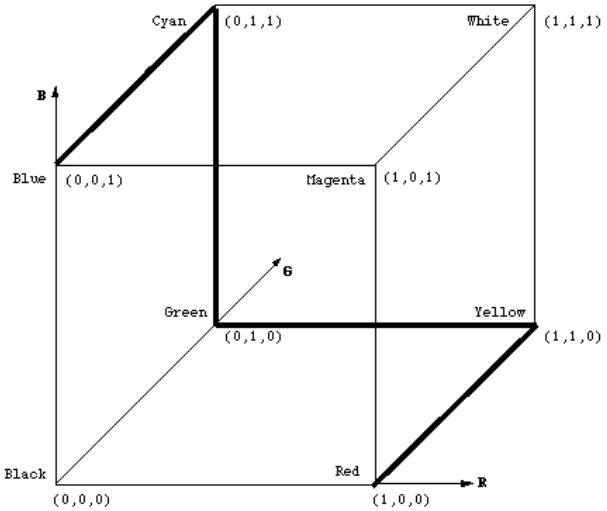
Contudo, se se pretender utilizar uma **imagem colorida** para representar uma determinada escala, pode-se então recorrer ao **espaço de cor RGB**.

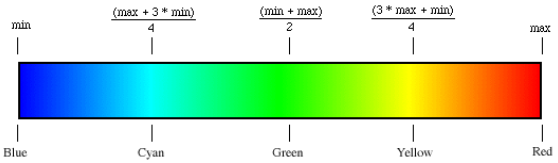
A representação por cores possibilita um vasto leque de gradações:

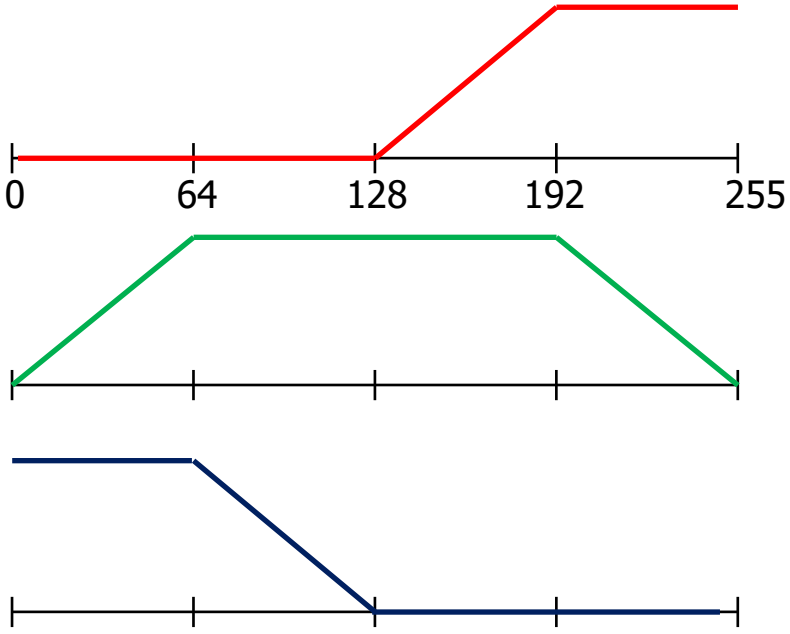
- **RGB**: Vermelho – Verde – Azul;
- **BGR**: Azul – Verde – Vermelho;
- **RWB**: Vermelho – Branco – Azul;
- **BWR**: Azul – Branco – Vermelho;
- Etc…

- **Representação de Informação Através de Escalas de Cor**

Assim, para uma gradação em que o valor **mínimo** é representado pela cor **azul**, o valor **médio** pela cor **verde**, e o valor **máximo** pelo valor **vermelha**, devem-se respeitar as seguintes regras:







Duarte Duque | dduque@ipca.pt

- **Exercícios:**

- Construa uma função que converta uma imagem com escala de intensidades em cinzento numa imagem com uma escala em cores do espaço RGB.

```
int vc_scale_gray_to_rgb(IVC *src, IVC *dst);
```

VISÃO POR COMPUTADOR

Duarte Duque
dduque@ipca.pt