Computação Gráfica

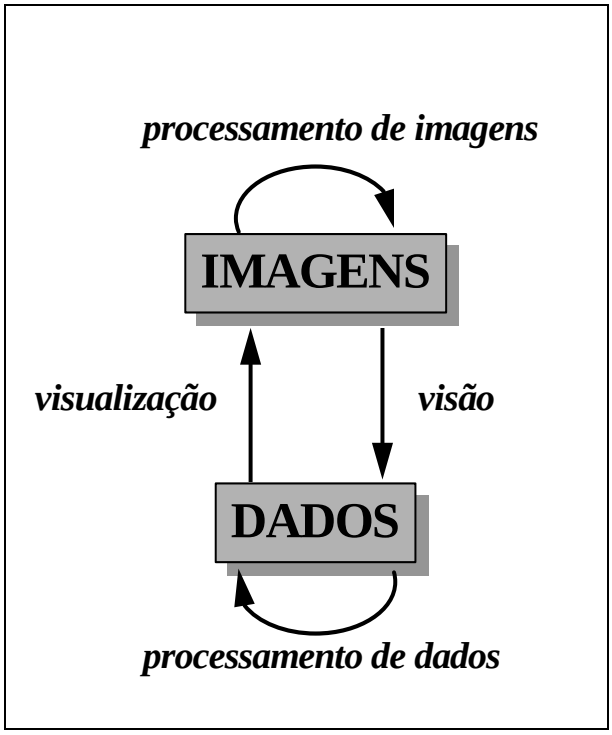
Arlindo Cardarett Vianna

Sumário

Capítulo 1 - Introdução a Computação Gráfica

Neste curso serão abordados, de uma forma genérica, vários tópicos relacionados à computação gráfica, com o objetivo de dar ao leitor uma visão geral das áreas existentes dentro desta disciplina.

Por uma definição formal, entende-se que : "Computação Gráfica é a disciplina que trata das técnicas e dos métodos computacionais, que convertem dados para dispositivos gráficos e vice-versa". Em uma definição mais informal pode-se definir que: "Computação Gráfica é o veículo de comunicação homem/máquina mais adequado à percepção humana."



Partindo-se destas definições pode-se então, conceber um modelo para representação das três grandes áreas da Computação Gráfica (como esquematizado na figura ao lado).

Como **visualização** entende-se que a imagem é gerada através de um modelo matemático que contém os elementos gráficos básicos (linhas , áreas, textos, etc.). Este área é mais conhecida como **Computação Gráfica Gerativa** e posui diversas áreas de atuação dentro da engenharia, geologia, cartografia, dentre outras.

O **Processamento de Imagens** busca o realismo da imagem digital, tentando torná-la mais acessível à percepção humana. Exemplos deste tipo de computação gráfica podem ser encontrados nas áreas: de pesquisa biológica, de defesa e pesquisa (interpretação de imagens obtidas por satélites), médicas (raios-X, tomografia), de filmes e vídeo, entre outras.

.

A **visão** baseia-se na obtenção da descrição da imagem digital, fornecida comumente por um conversor analógico/digital (**Reconhecimento de Padrões**). Por exemplo tem-se a "visão" de um robô industrial e a "leitura" por computador de caracteres manuscristos**.**

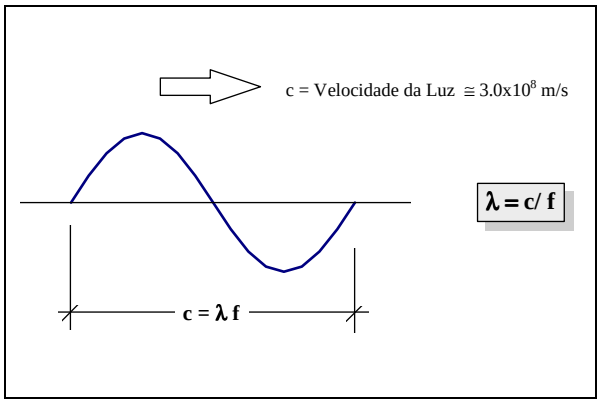
De modo a mostrar a generalidade da figura acima, quando se processa somente os dados e não há geração de imagens tem-se uma outra área dentro da informática que então é denominada de **Processamento de Dados**.

A computação gráfica possui profissionais com formação bastante variada. Engenheiros, arquitetos, matemáticos, artistas, médicos, entre outros, utilizam intensamente esta disciplina. Destacam-se os seguintes grupos nesta área:

- Implementadores de ferramentas: Desenvolvem sistema gráficos básicos (OpenGL, GDI).
- Programadores de Aplicações: Utilizam linguagens de programação para criação de programas específicos utilizando sist. gráficos existentes.
- "Customizadores": Adaptam programas existentes para aplicações específicas.
- Usuários: Utilizam programas existentes para a produção de imagens.

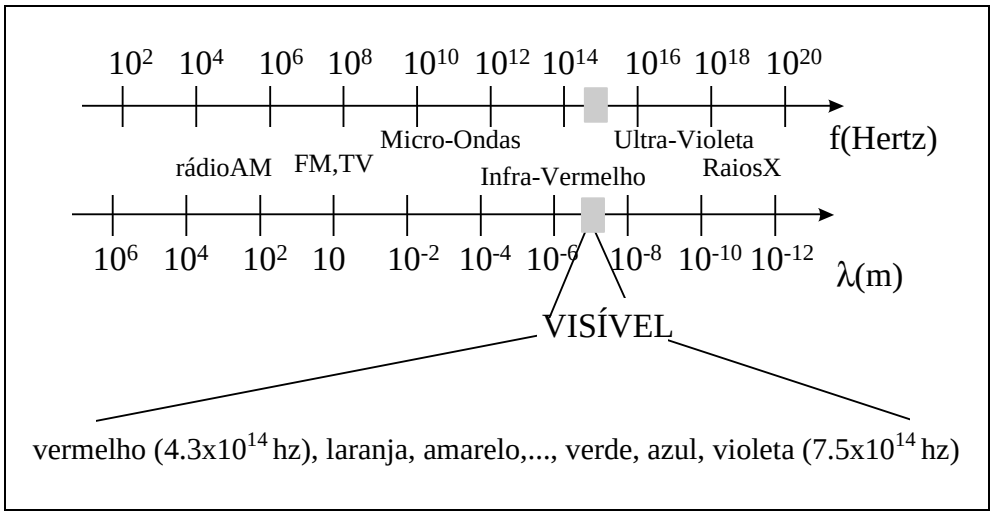
Capítulo 2 - Luz e Cor

O estudo de luz e cor deve ser iniciado pela Física elementar, uma vez que a luz é uma onda eletromagnética.



Sendo assim, da Física vem que, **todas** as ondas eletromagnéticas se propagam no vácuo com a mesma velocidade *c* com o valor de $3x10^8$ m/s (velocidade da luz). Em decorrência deste fato, sabendo-se a freqüência de de uma onda eletromagnética (f), no vácuo, pode-se determinar o comprimento de onda (λ) desta radiação, através da seguinte equação: $\lambda = c/f$.

Desta forma, pode-se então exemplificar as ondas eletromagnéticas de maior importância nas pesquisas e nas aplicações práticas, em função do comprimento de onda (propriedade que fornece uma das principais características da onda): Raios-X (faixa de 10^{-1} até 10 A), ondas ultravioletas (faixa de 1 até 400 μm), o espectro de luz visível (faixa de 400 até 700 μm), ondas infravermelhas (faixa de 700 μm até 1 mm) e faixas de radiofreqüência que variam de 20 cm até 10^5 m.

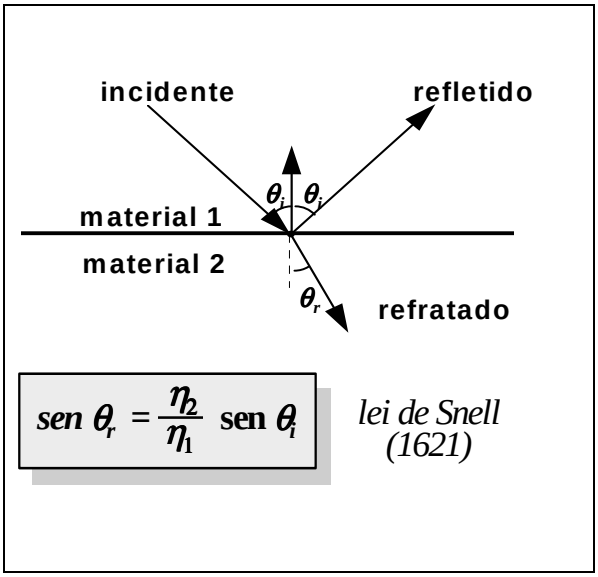


O espectro de luz visível, pode então assumir diversas cores (desde o violeta até o vermelho), em função do comprimento de onda, como exposto na tabela ao lado.

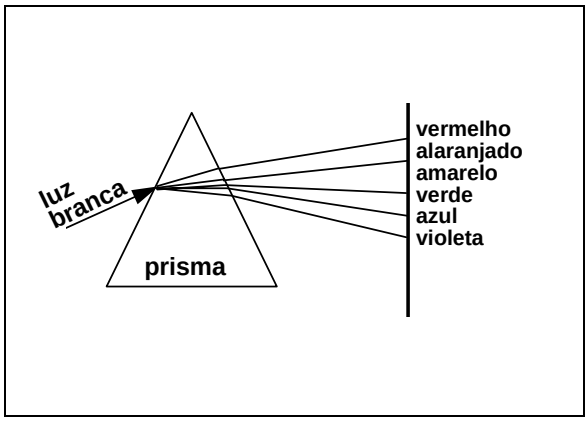
Cor	λ
Violeta	380-440 mμ
Azul	440-490 mμ
Verde	490-565 mμ
Amarelo	565-590 mμ
Laranja	590-630 mμ
Vermelho	630-780 mμ

Como o comprimento de uma onda da luz é muito pequeno (da ordem de 10^{-5} cm), a teoria da física se divide em dois grandes grupos: Ótica Física, que trata dos fenômenos ondulatórios da luz e Ótica Geométrica, que estuda o comportamento da onda quando esta interage com objetos muito maiores que o comprimento da onda da luz. Com relação ao nosso estudo se dará enfoque à Ótica Geométrica que assume que a direção de propagação da luz seja dada a partir de raios luminosos.

Desta forma, vai-se discutir agora dois fenômenos da Ótica Geométrica: a refelexão e a refração. Para tal, supõe-se que haja um plano, ao qual incide um raio luminoso e que parte deste raio seja refletido por este plano e parte seja refratado. Define-se como ângulo de incidência como sendo o ângulo formado pelo raio e a normal a este plano, ângulo de reflexão entre a normal do plano e raio refletido e ângulo de refração como sendo entre a normal e o raio refratado. Pode-se provar (por ex. pela Lei da Conservação da Quantidade de Movimento) que o ângulo de incidência é igual ao ângulo de reflexão (**Lei da Reflexão**), e que o ângulo de refração pode ser dado pela **Lei de Snell**, de acordo com o índice de refração de cada material.



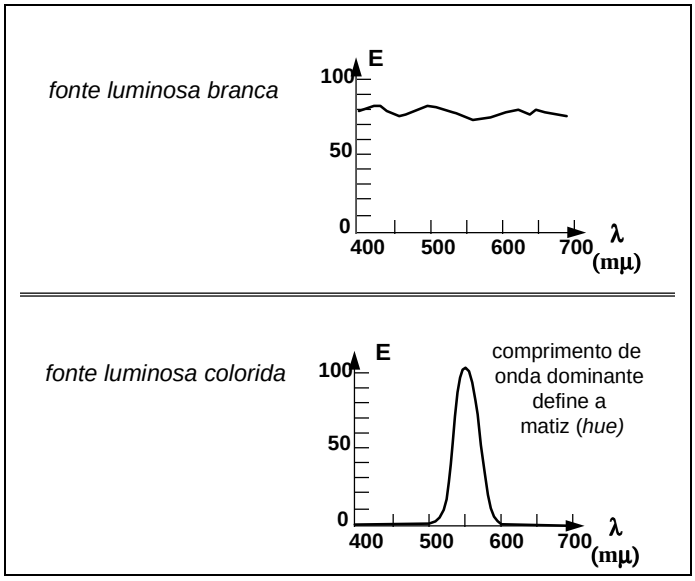
Como já foi dito anteriormente, as ondas eletromagnéticas se propagam no vácuo com a mesma velocidade *c*, ou seja, a velocidade da luz. Entretanto, quando estas ondas se propagam em um meio material, a velocidade de propagação de cada onda (v) passa a ser função do comprimento de onda da radiação. Sendo assim, pode-se definir como o índice de refração de uma luz monocromática como sendo $\eta = c / v$. Estes fenômenos de reflexão e refração estão presentes no dia a dia, e devido a eles que ocorrem as miragens no deserto, o efeito de uma estrada parecer molhada e o fenômeno do arco-íris.

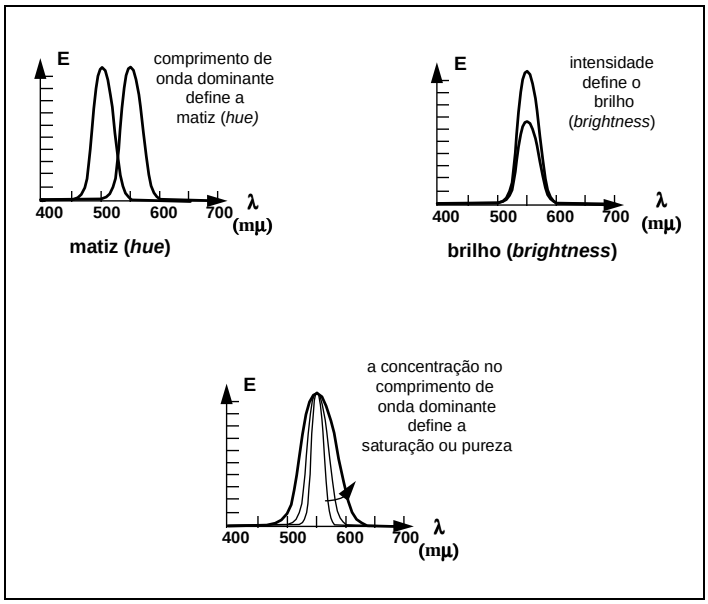


Um experimento do conhecimento de todos é que quando a luz branca incide em um prisma, há a decomposição desta nas cores do arco-íris.

Utilizando os conceitos de refração, Isaac Newton provou que a luz branca continha todos os comprimentos de onda e que quando esta incidia no prisma, havia então a decomposição desta nas cores do arco-íris. Para provar tal fato, Newton utilizou dois prismas, colocando o segundo recebendo as cores geradas pelo prmeiro e com-pondo novamente a luz branca. Esta experiência foi necessária, pois na época, acreditava-se que o prisma criava as cores espectrais.

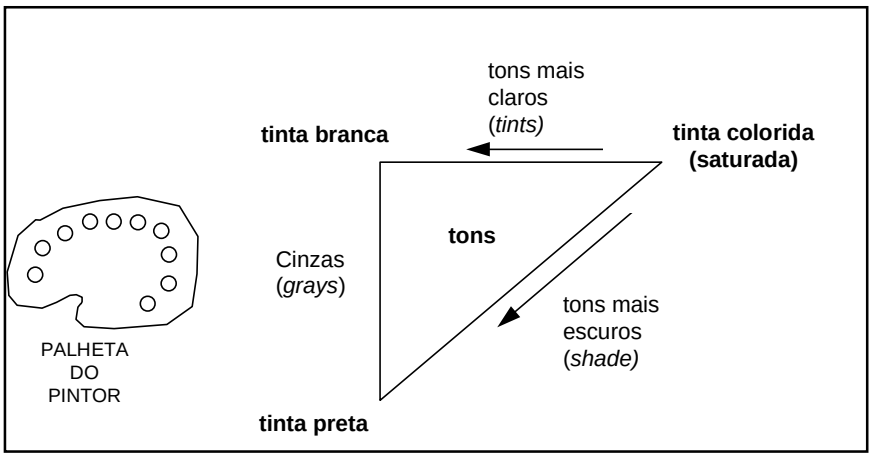
Aproveitando-se então a conclusão de Newton, pode-se então definir que as fontes luminosas brancas possuem todos os comprimentos de onda. Em conseqüência, uma fonte luminosa colorida tem um comprimento de onda dominante que define a sua matiz.

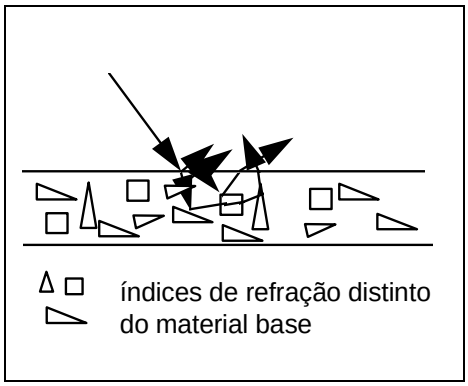




As fontes luminosas não são somente caracterizadas pela matiz (*hue)* que é a presença de um comprimento de onda dominante, também pode-se definir a intensidade ou brilho (*brightness*) - amplitude do comprimento de onda, e a saturação que é a concentração em torno do comprimento de onda dominante.

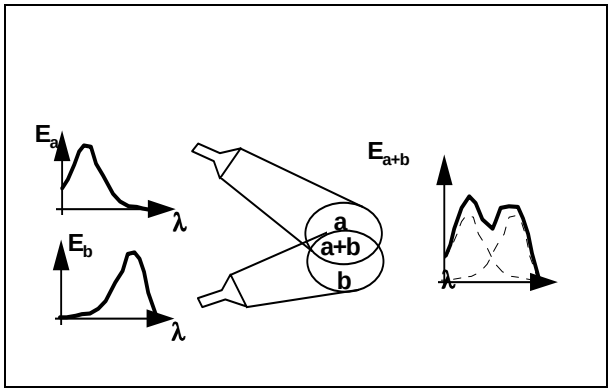
Tendo-se em mente, estas três principais características de uma fonte luminosa (matiz, brilho e saturação), vamos destacar um processo de formação de cores baseado na palheta de um pintor. Basicamente, tem-se de um lado tinta branca, do outro tinta preta e em uma outra extremidade tinta colorida (saturada). É intuitivo que ao se misturar a tinta saturada com a tinta branca há uma perda de pureza, tornando esta tinta mais clara (*tints*). Por outro lado, ao misturar-se esta tinta saturada com o preto ocorrerá uma perda de luminância, ou seja, tons mais escuros (*shade*). Os diversos tons de cinza (*grays*) aparecerão ao misturar-se a tinta branca com a preta, e todos os outros tons existentes ficarão espalhados dentro deste triângulo definido pelas cores branca, preta e tinta saturada, como mostra a figura a seguir.

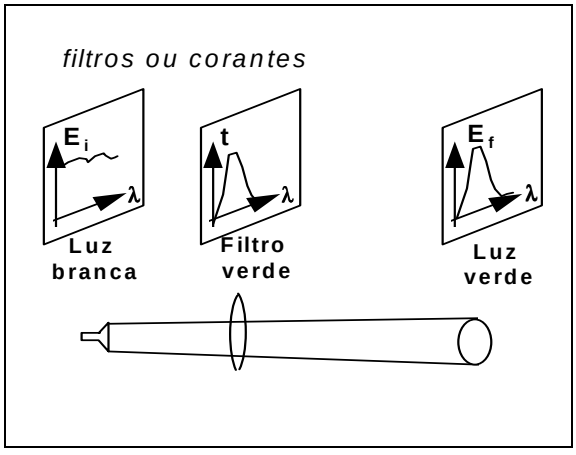




O processo de formação de cores por pigmentação, baseia-se na descrição da palheta do pintor, uma vez que a luz ao atingir a camada de pigmentos sofre processos de reflexão, absorção e transmissão (fenômeno conhecido como espalhamento) produzindo assim a(s) cor(es) desejada(s). Esta técnica, como não poderia deixar de ser, é muito utilizada na pintura de quadros.

Um outro processo de formação de cores, é o chamado processo aditivo, um exemplo deste processo pode ser visto ao lado, onde duas fontes luminosas de cores diferentes são projetadas em duas regiões. Na área de interseção há a formação de uma nova cor, uma vez que, o olho não consegue distinguir componentes. O processo aditivo é usado, largamente nas televisões comerciais.

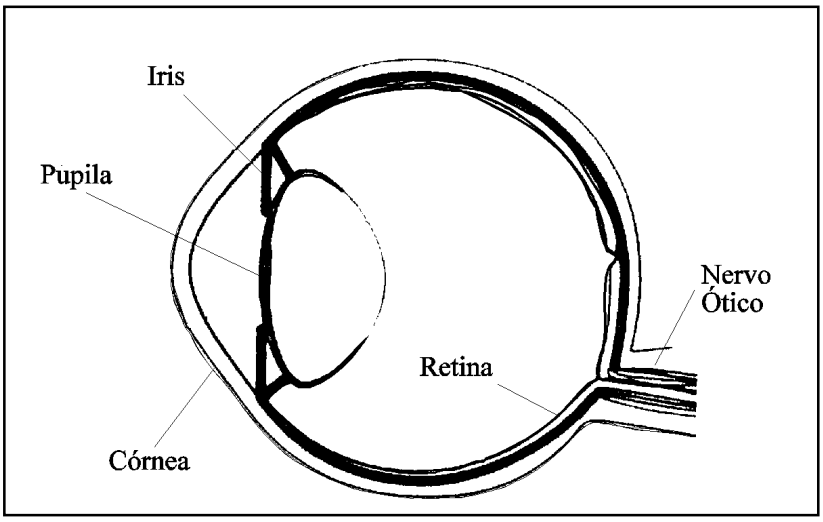


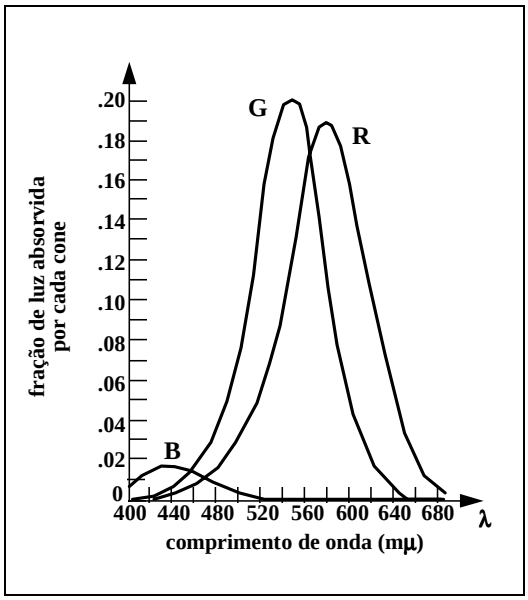


Um outro processo de formação de cores é o subtrativo que é o processo utilizado em *slides*. Este processo baseia-se no uso de filtros ou corantes que tem por objetivo filtrar determinados comprimento de onda. Exemplificando, ao se emitir uma luz branca (que possui todos os comprimentos de onda) sobre um filtro verde, este filtra **todos** os comprimentos de onda deixando só "passar" o comprimento de onda relativa a cor verde, produzindo assim o verde. Na utilização de corantes o processo é o mesmo só que são usados pigmentos que absorvem e refletem alguns comprimentos de onda.

Como já foi mencionado anteriormente no processo aditivo de formação de cor, o olho humano não consegue diferenciar componentes e sim a cor resultante; diferentemente do ouvido que consegue distinguir, por exemplo, dois instrumentos diferentes tocados simultaneamente. Desta forma, seria então, interessante saber algo mais sobre o olho humano, responsável pela visão.

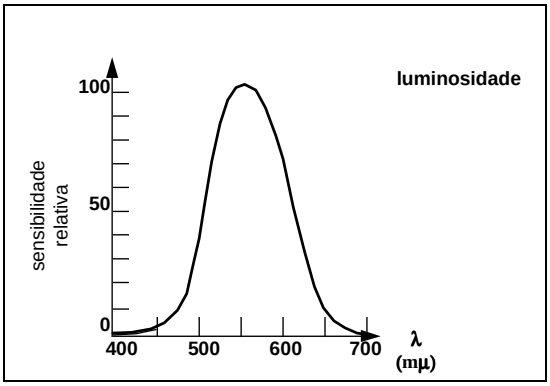
Os raios luminosos incidem na córnea sendo então refratados. A seguir estes incidem sobre a lente que tem por objetivo projetá-los na retina. Na retina encontram-se dois tipos de fotoreceptores os cones e os bastonetes, que convertem a intensidade e a cor da luz recebida em impulsos nervosos. Estes impulsos são enviados ao cérebro através do nervo ótico e então tem-se a percepção de uma imagem.



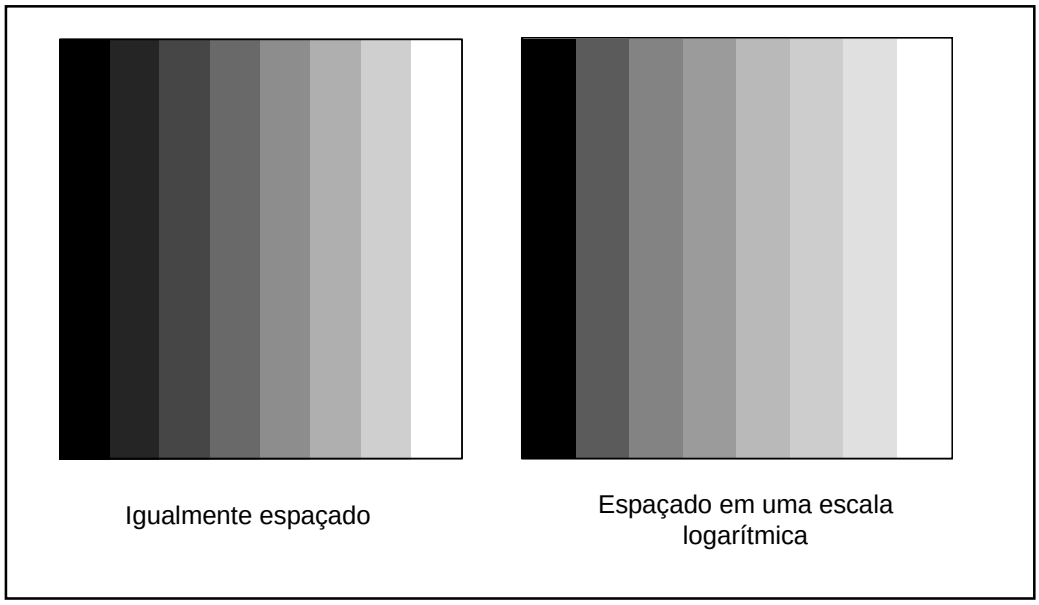


Os fotoreceptores do olho humano apresentam características totalmente diferentes. Existem na verdade três tipos de cones que respondem a espectro de cores distintos (vermelho, verde e azul), como mostrado ao lado. Sendo assim, diz-se que o sistema visual humano distingue as cores pelo processo da **tricromacia**. Nota-se que a eficiência do cone que responde a cor azul possui uma eficiência bem menor do que os outros dois tipos de cones.

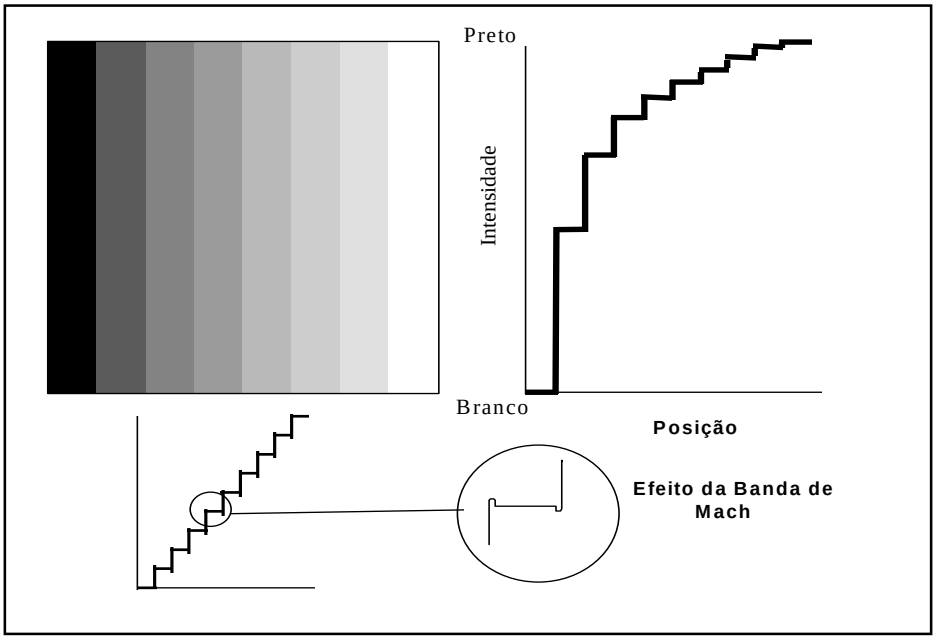
Os bastonetes por sua vez, embora sejam maioria absoluta, só conseguem captar a luminosidade da cor, ou seja, só respondem a um espectro e desta forma não diferenciam cores.



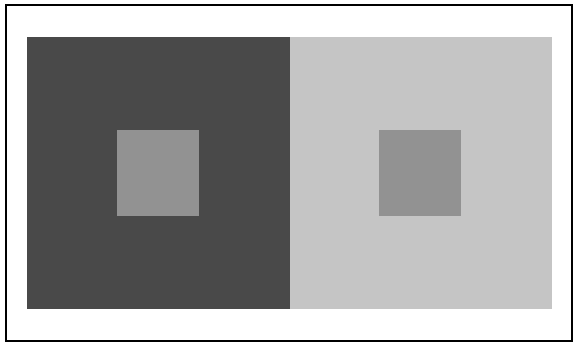
Sendo assim, na formação da imagem há uma interação dos cones e dos bastonetes, e decorrente desta interação ocorrem alguns fenômenos no sistema visual humano. O primeiro a ser destacado é que a percepção visual humana é logarítmica. Na figura a seguir, no primeiro quadro, os tons de cinza foram igualmente espaçados não se tendo uma impressão homogênea, parecendo que a faixa escura é mais densa. No segundo quadro, os tons de cinza foram perceptualmente espaçados, chegando-se aproximadamente numa escala logarítmica.

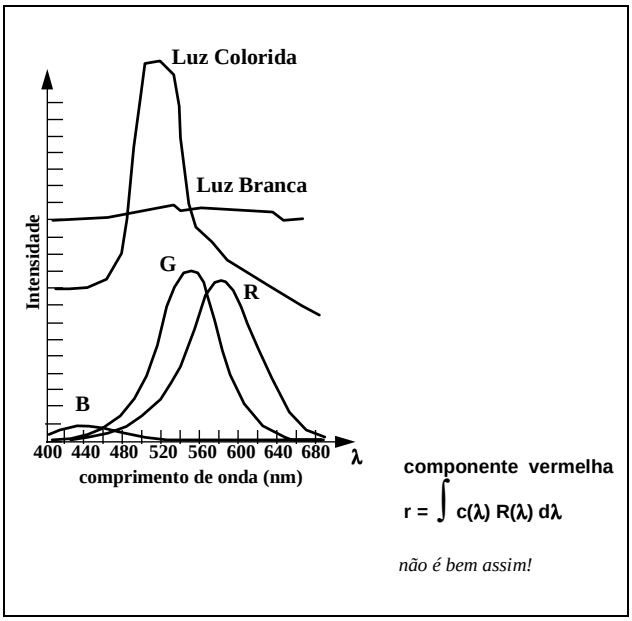


O segundo aspecto é o que se denomina de **Efeito da Banda de Mach**. Analisando-se os tons de cinza da figura a seguir, da cor mais escura para a mais clara, tem-se a impressão que existem pequenas discontinuidades na interface entre as cores (aumento da luminosidade - faixa constante - diminuição da luminosidade).



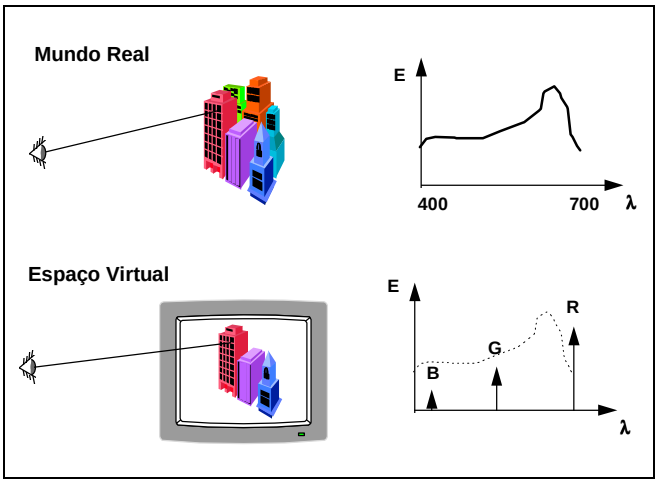
O último aspecto a ser abordado é o que se chama de **Contraste Simultâneo**. Analisando-se a figura ao lado, tem-se a impressão que o quadrado interno da esquerda é mais claro, embora possuam a mesma cor. Este fenômeno pode ser explicado a partir da luminosidade das áreas envolventes, ou seja, quando se tem uma área externa mais escura o quadrado interno parece ser mais claro.

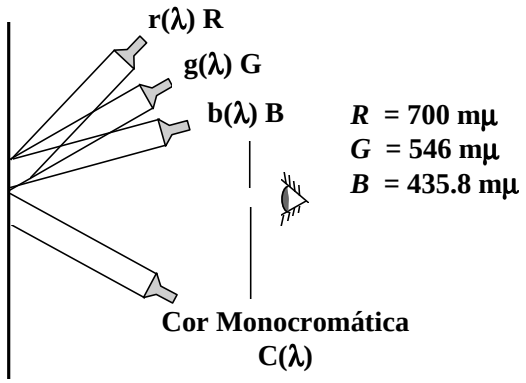




Uma vez vistos os aspectos principais do sistema visual humano, resta saber como se dá a percepção de uma cor? Matematicamente falando, deve-se compor em uma integral as componentes vermelha, verde e azul, para obter-se a cor desejada. Este pode ser o processo utlizado por um *scanner*, mas não pelo olho humano.

Restringindo-se o problema de reprodução de cores em Computação Gráfica, há necessidade de uma "combinação linear" das cores básicas para então formar as cores desejadas. A este processo dá-se o nome de **metamerismo**, ou seja, quando se tem a mesma sensação de cor.

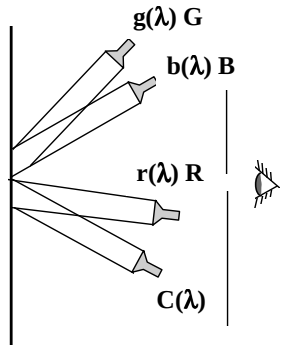




C(λ) = r(λ) R + g(λ) G + b(λ) B

A Comission Internationale de l'Éclairge (**CIE**), propôs um método para representação perceptual de cores, utilizando as cores básicas vermelho (*Red*), verde (*Green*) e azul (*Blue*), daqui para frente sendo denominado de RGB. Como já foi visto anteriormente, o olho humano não consegue perceber componentes, desta forma o que se fez foi projetar uma luz em um detreminado anteparo e regulando-se a intensidade das cores RGB tentar produzir a mesma sensação de cor (**metamerismo**).

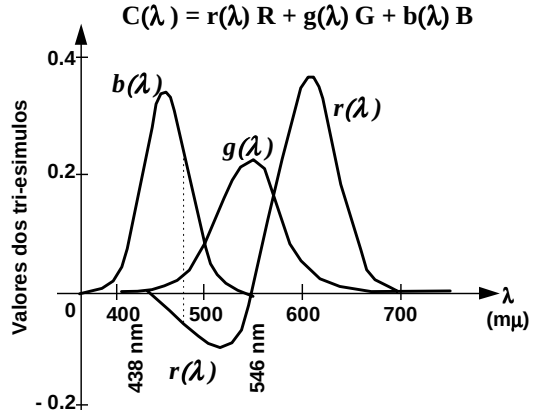
Entretanto, com o sistema proposto anteriormente, não se consegue representar todas as cores visíveis. A solução para contornar este problema, foi de utilizar o **artifício da subtração**, ou seja, faz-se uma das componentes RGB interagir com a cor desejada, produzindo então uma nova cor procurando com as outras duas restantes conseguir o **metamerismo**.



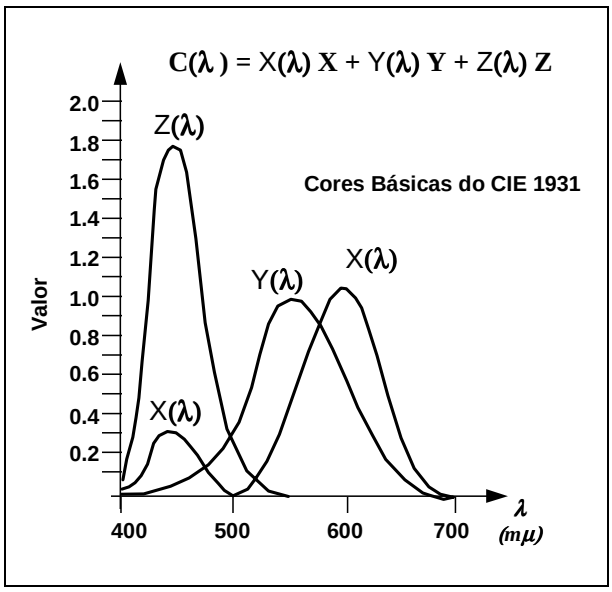
C(λ) + <u>r</u>(λ) R = g(λ) G + b(λ) B

C(λ) = r(λ) R + g(λ) G + b(λ) B, onde **r(λ) = - <u>r</u>(λ)**

C(λ) = r(λ) R + g(λ) G + b(λ) B



De acordo com o artifício de subtração utilizado, pode-se montar funções auxiliares, como mostrado ao lado, de modo a formarem as cores espectrais. Deve-se somente frisar que estas funções mostradas não são as distribuições espectrais, e sim funções que se combinadas reproduzem as cores espectrais, como por exemplo a cor C(λ).



De forma a não utilizar valores negativos, o **CIE**, em 1931, definiu padrões primários - X, Ye Z, para substituir as cores RGB, para representar o espectro de cor, como por exemplo apresentado para a cor C(λ). Estes padrões não correspondem a estímulos reais de cor, ou seja, não são cores visíveis. Um outro detalhe importante, é que o padrão Y foi escolhido, de forma a ser semelhante à curva de sensibilidade do olho humano (luminância).

Como já foi mencionado, as cores do sistema XYZ não são realizáveis fisicamente. Sendo assim, pode-se obter as grandezas colorimétricas desse sistema a partir do sistema **CIE-RGB**, a partir das seguintes hipóteses:

- as componentes de cor devem ser positivas,
- deve-se obter o maior nº possível de cores espectrais com algumas coordenadas de cromaticidade nula e,
- duas primárias devem ter luminância ZERO.

Finalmente, definindo os vetores da cor branca de referência de cada sistema e fazendo uma transformação inversa, obtem-se as grandezas do sistema XYZ em função de RGB como apresentado ao lado.

$$C(\lambda) = r(\lambda)\,R + g(\lambda)\,G + b(\lambda)\,B$$

Escolhendo-se XYZ tal que:

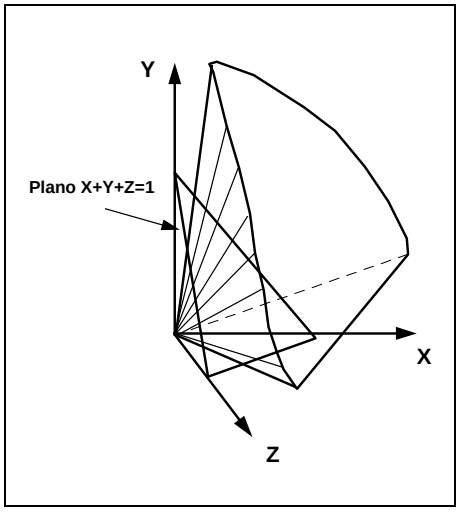
$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 2.36470 & -0.51515 & 0.00520 \\ -0.89665 & 0.14264 & -0.01441 \\ -0.46808 & 0.08874 & 1.00921 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

tem-se

$$C(\lambda) = X(\lambda)\,X + Y(\lambda)\,Y + Z(\lambda)\,Z$$

onde

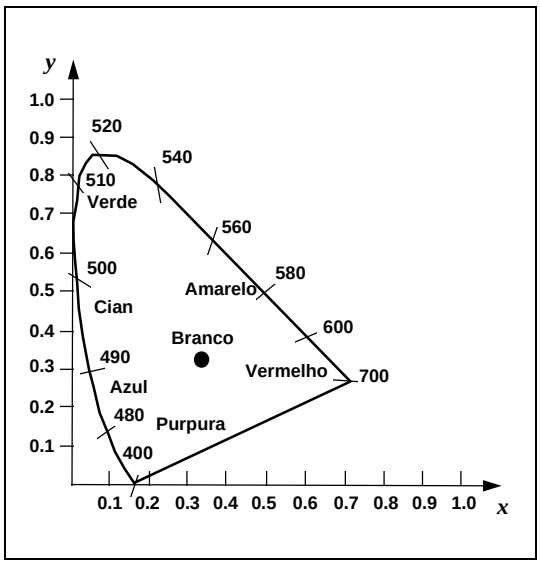
$$X(\lambda) = 2.36470r(\lambda) - 0.89665g(\lambda) - 0.46808b(\lambda)$$
$$Y(\lambda) = -0.51515r(\lambda) + 0.14264g(\lambda) + 0.08874b(\lambda)$$
$$Z(\lambda) = 0.00520r(\lambda) - 0.01441g(\lambda) + 1.00921b(\lambda)$$



A forma do sólido de cor **CIE XYZ** (contem todas as cores visíveis) pode ser visto ao lado. Basicamente, é de forma cônica, com o vértice na origem. É apresentado também o plano de crominância ou o plano de Maxwell (X + Y + Z = 1), que tem importância para se obter uma representação paramétrica do espaço de cromaticidade. Pode-se destacar também o triângulo formado pela interseção deste plano com os eixos do espaço de cor XYZ que é chamado triângulo de Maxwell. Analisando-se a figura ao lado, pode-se concluir que as cores visíveis se encontram no primeiro octante do espaço de cor.

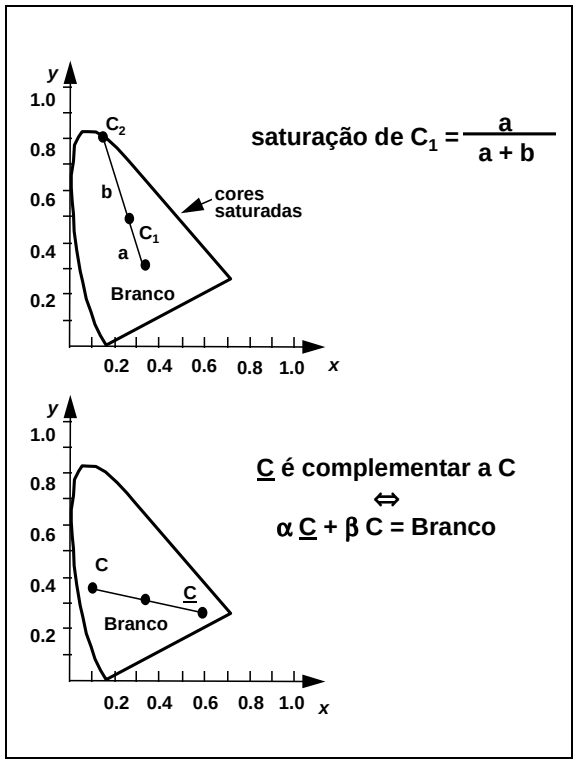
Uma cor **C** pode ser representada da seguinte forma **C** = X **X** + Y **Y** + Z **Z**. Pode-se definir valores de cromaticidade, que dependem somente dos comprimentos de onda dominantes e da saturação e são independentes da parcela de energia luminosa (luminância), a partir das seguintes equações:
x = X / (X+Y+Z); y = Y/(X+Y+Z); z = Z/(X+Y+Z).

É interessante notar que x + y + z = 1, uma vez que x, y e z estão no plano X + Y + Z =1.



Sendo assim, retirando o brilho ou a luminosidade da definição da cor em **CIE XYZ**, e utilizando as coordenadas de cromaticidade x e y, obtem-se o **Diagrama de Cromaticidade do CIE**. O interior e o contorno deste diagrama com forma de ferradura representam todas as cores visíveis. Todas as cores puras do espectro estão localizadas na região curva do contorno, enquanto que a linha reta deste contorno é chamada de Linha Púrpura ou Linha *Magenta*, uma vez que ao longo desta linha se encontram as cores púrpuras e *magentas* saturadas. Estas cores não podem ser definidas por um comprimento de onda dominante e desta forma são denominadas não-espectrais. Destaca-se ainda neste diagrama a luz branca padrão que é definida em um ponto próximo de x = y = z = 1/3.

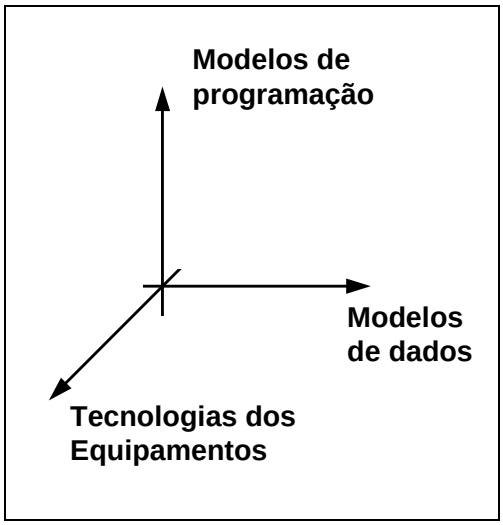
Utilizando-se o **Diagrama de Cromaticidade do CIE**, pode-se visualizar com mais facilidade conceitos como saturação de uma cor e cores complementares. Na parte superior da figura ao lado pode-se dizer que a saturação da cor C_1 é definida como sendo a / (a+b). Na parte inferior verifica-se que C é complementar a C pois são cores que quando combinadas produzem a luz branca. Exemplos de cores complementares são: o ciano - vermelho, magenta - verde e amarelo - azul. Este diagrama pode ser útil na visualização de gamutes de monitores e impressoras, e serão vistos adiante no item sobre sistemas de cores utilizados nos dispositivos.



Capítulo 3 - Equipamentos

Os equipamentos desempenham um papel de grande importância dentro da Computação Gráfica, uma vez que a partir deles é que são visualizadas as imagens dos programas gráficos tornando possível a interação com o usuário. Serão abordados vários temas relacionados aos equipamentos como: histórico, evolução, os principais dispositivos gráficos hoje existentes, técnicas utilizadas e novas tendências.

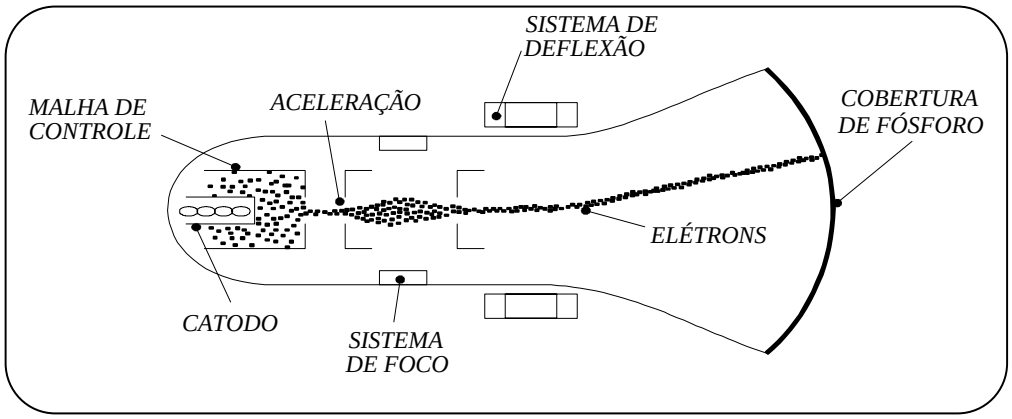
Antes porém de inicializar o estudo sobre equipamentos, deve-se fazer uma consideração sobre a evolução destes. Analisando-se o triedro ao lado pode-se fazer a seguinte indagação "O que observar na evolução dos equipamentos?". É óbvio que com o avanço da tecnologia os equipamentos evoluem. Esta evolução contribui para que determinadas áreas ligadas a modelos de programação e dados necesitem cada vez mais de equipamentos sofisticados (modelos bidimensionais para modelos tridimensionais, animação em tempo real). Desta forma, conclue-se que a evolução dos equipamentos não depende somente do avanço da tecnologia, mas de um contexto geral, como apresentado ao lado.



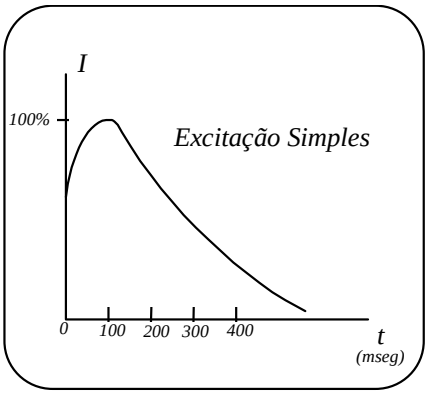
Os dispositivos gráficos, de uma forma geral, podem ser classificados de acordo com a sua função básica de transmitir dados de, ou para o computador da seguinte forma:
- dispositivos de saída interativos,
- dispositivos de saída passivos e
- dispositivos de entrada.

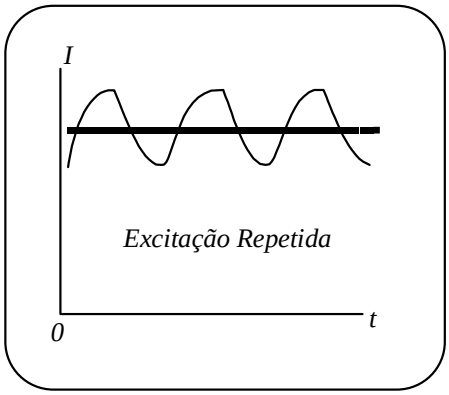
Os dispositivos que exibem dados do computador, de forma com que o usuário interaja na criação do desenho, são ditos **dispositivos de saída interativos**. A maioria destes dispositivos, se baseia no tubo de raios catódicos (*Cathode Ray Tube* ou CRT), cujo funcionamento é ilustrado abaixo.

Através do tubo, um feixe de elétrons é dirigido a tela.

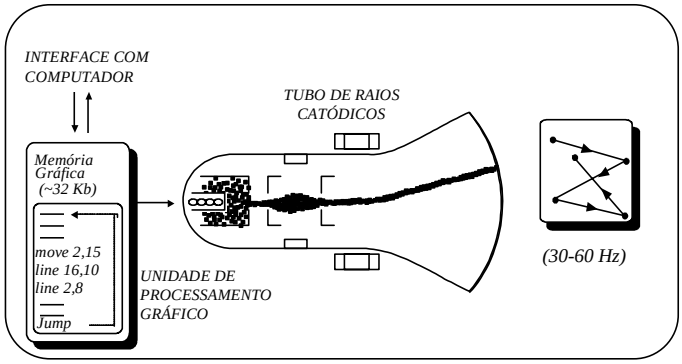


A tela por sua vez, é coberta por uma camada de fósforo, elemento sensível, que brilha ao ser atingido pelo feixe , mas possui decaimento rápido (após 0.2 segundos do impacto, cerca da metade da luminosidade já se extinguiu, ver o comportamento do fósforo abaixo). As diversas tecnologias, até hoje existentes baseadas em CRT, diferem na forma de manter o desenho "aceso", fornecendo ao usuário a impressão de uma imagem constante na tela do terminal, uma vez que o olho humano possui uma limitação pra a visualização que é de 1/20 segundos.



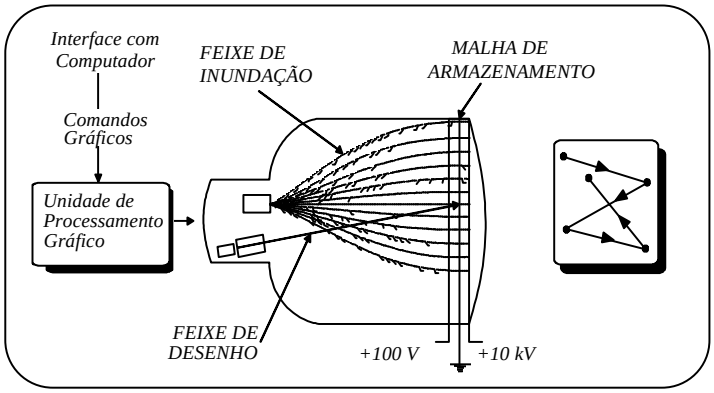


O primeiro dispositivo desenvolvido foi o vetorial de retraçamento (*vector refresh*), sendo utilizado entre aproximadamente 1960 e 1985. Apresenta uma técnica sofisticada para o traçado de linhas que consistia em redesenhar várias vezes a figura na tela (de 30 a 60 vezes por segundo, para evitar que a imagem fique piscando, fenômeno conhecido por *flickering*). Se cada retraçamento o desenho a ser iluminado muda ligeiramente de posição, cria-se a ilusão de movimento. Por esta razão, estes terminais eram conhecidos por dinâmicos. O funcionamento esquemático destes dispositivos é apresentado ao lado.

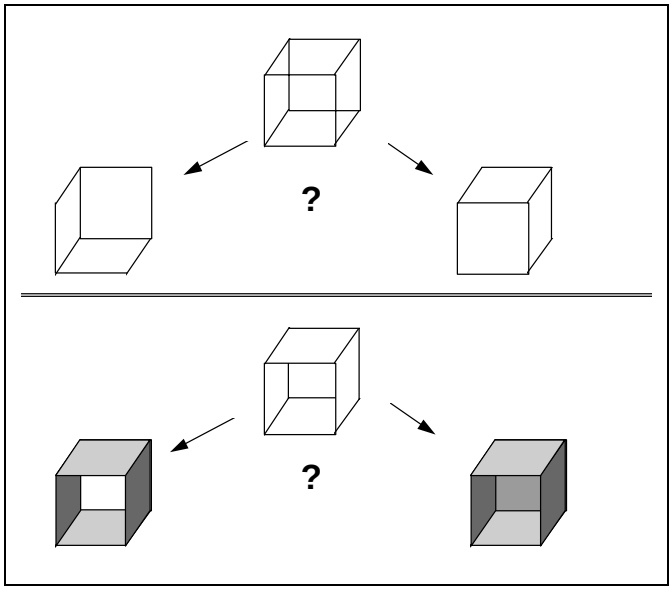


Nos terminais vetoriais dinâmicos a primitiva é a reta, e tinha como principais características: alta resolução, bom contraste, possibilidade limitada de cor, limitação de apresentarem imagens por linhas (modelos ditos de arame - *wire frame*), o alto preço e embora fossem capazes de redesenhar a tela rapidamente sem que o usuário percebesse, se houvesse uma quantidade de linhas muito grande poderia ocorrer *flickering*.

Os dispositivos vetoriais de armazenamento (*vector storage*) surgiram como solução para diminuir os custos dos equipamentos gráficos (1970 a 1985). Ao invés do refrescamento, as linhas a serem iluminadas ficam identificadas em uma placa cujo potencial é constante, conforme apresentado ao lado. Desta forma , a luminosidade é mantida sem necessidade das altas taxas de transmissão do modelo de retraçamento. Como vantagens tinha-se o preço acessível, uma alta resolução, tamanho grande de tela, não ocorre *flickering*, e a primitiva continuava a ser a reta e possuia uma boa qualidade.



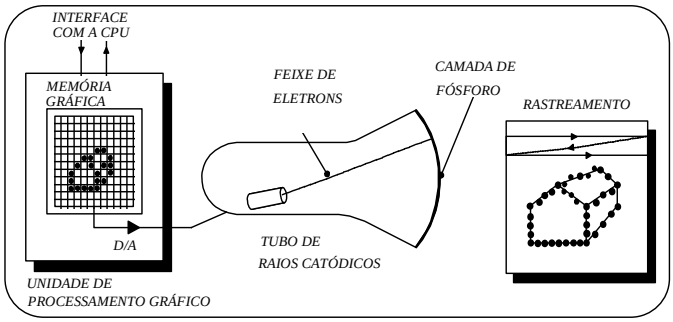
As desvantagens deste tipo de terminal eram: capacidade somente para representar modelos *wire frame*, não havia simulação de movimento e nem possibilidade de cor e a limpeza da tela era acompanhada de um brilho súbito.

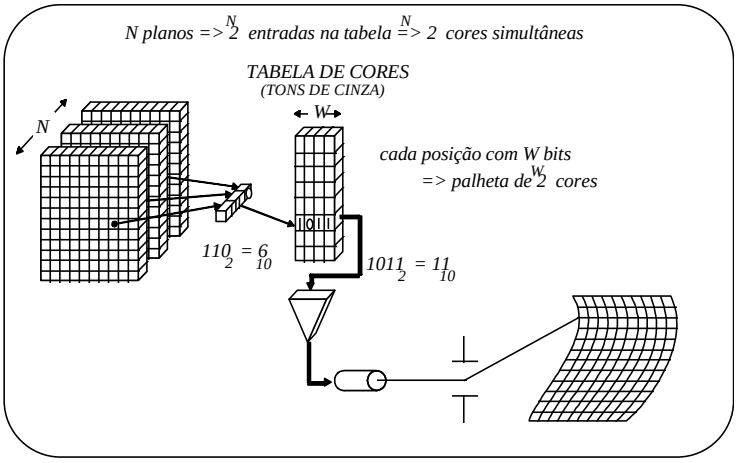


Os dois tipos de dispositivos acima descritos apresentavam como primitiva a reta. Entretanto, a representação de modelos vetoriais gera um problema muito sério na computação gráfica - a ambiguidade. Este problema pode ser facilmente visualizado nos exemplos apresentados ao lado. Na parte superior pode-se ter uma visualização do cubo nas duas formas mostradas dependendo da posição do observador e na parte inferior, um problema clássico de *rendering* de superfície.

De forma a contornar os problemas dos dispositivos vetoriais e acompanhando a evolução da tecnologia, surgiram os dispositivos matriciais de rastreamento (*raster*) em torno da década de 80. Estes dispositivos utilizam a mesma tecnologia da televisão comercial, ou seja, a tela é composta de pontos, mas a exemplo dos vetoriais de retraçamento, o terminal não possui nenhuma capacidade de armazenamento, sendo assim, é necessária a reanimação contínua da tela, o que é feito linha a linha (rastreamento). A principal diferença entre estes terminais *raster* e os dinâmicos, é que na tecnologia *raster* todos os pontos da tela são "reanimados", enquantos os dinâmicos, somente redesenha-se as linhas que compõem a figura.

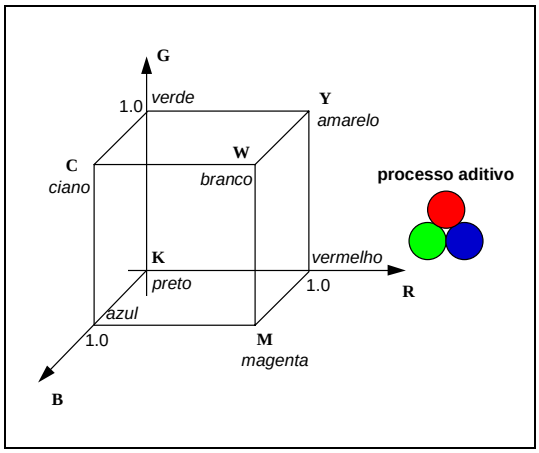
A taxa de refrescamento - *refresh cycle*, varia entre 24 e 72 Hz. A primitiva é o ponto (*picture element* ou *pixel*), menor unidade para atribuições. Os terminais matriciais fazem uso de uma memória gráfica (*frame buffer memory*, *graphic memory*, ou *bit plane*), a fim de armazenar informações para cada um dos pontos endereçáveis da tela. Há pelo menos um *bit* para cada ponto da tela. O funcionamento esquemático deste tipo de terminal é apresentado ao lado - monocromático com dois níveis.



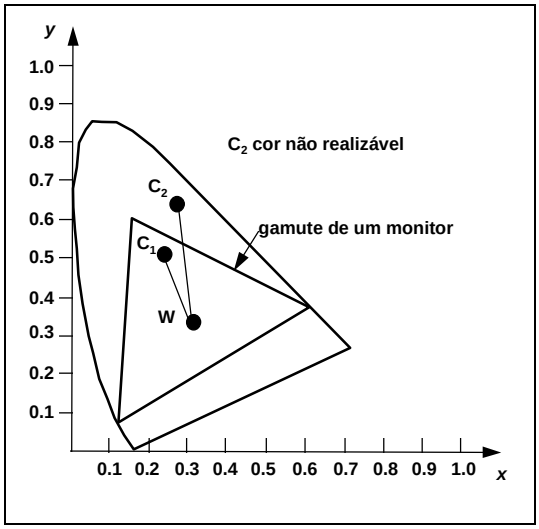


Diferentes níveis de intensidade podem ser incorporados por meio de *bit planes* adicionais. A combinação destes *bits* nos *bit planes* especificaria um nível de intensidade do feixe de elétrons, como mostrado ao lado, gerando tons de cinza.

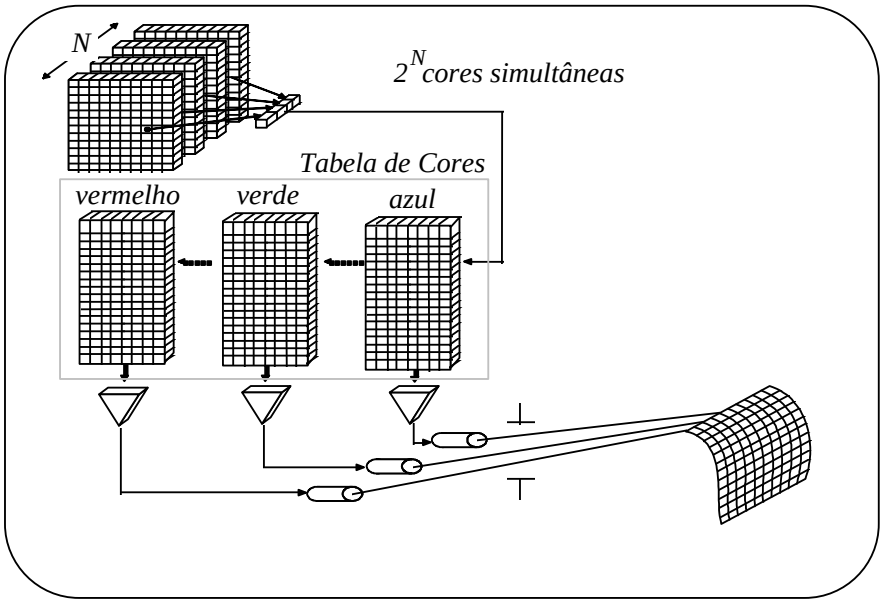
O sistema de cor de um monitor colorido possui uma base das cores primárias vermelho (*red*), verde (*green*) e azul (*blue*) e será denominado de **mRGB**. O sólido de cor deste sistema (cubo RGB) é um subconjunto do sólido gerado pela cores primárias, uma vez que cada cor tem uma intensidade máxima que é determinada pelo tipo de fósforo utilizado. O processo de formação de cores é o aditivo e a cor de cada *pixel* é determinada pela intensidade de cada cor primária incidente.

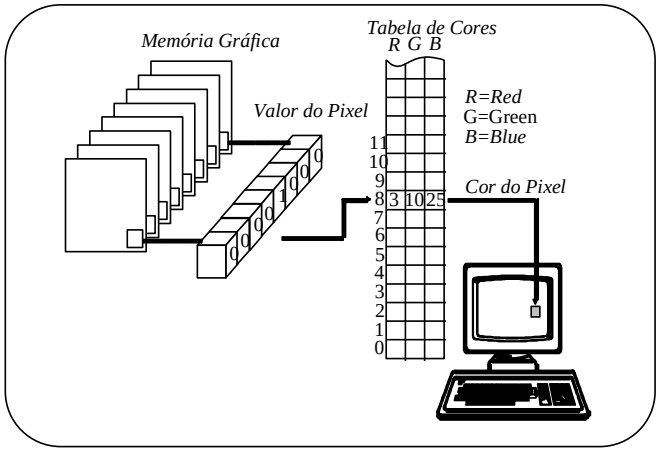


Utilizando-se o **Diagrama de Cromaticidade do CIE** pode-se então definir o gamute de um monitor baseando-se nos tipos de fósforos utilizados. Como apresentado na figura ao lado o triângulo formado apresenta todas as cores possíveis de se visualizar em um monitor colorido, excluindo-se a informação da luminância.

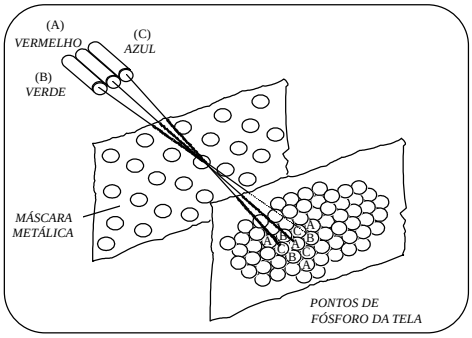


Desta forma, o funcionamento de telas com possibilidades de cores baseia-se na mistura do vermelho, verde e azul. O conceito de *bit planes* e tabela continuam válidos, no caso de monitores coloridos com *look-up table*, e o esquema do dispositivo colorido pode ser visualizado nas duas figuras a seguir.

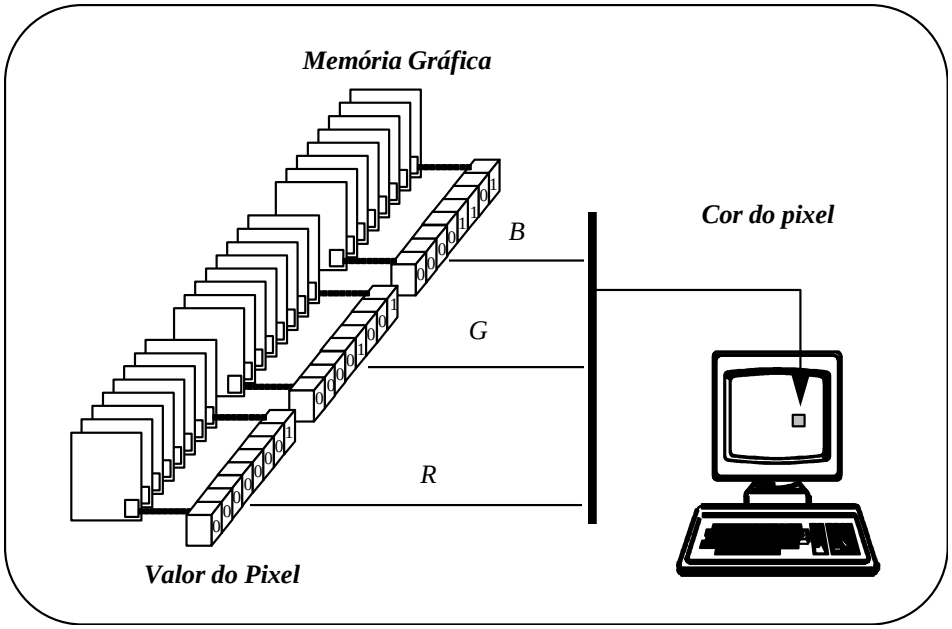




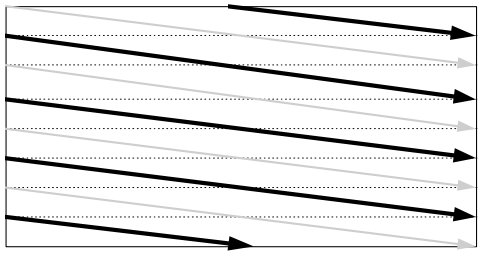
Como é mostrado no funcionamento esquemático dos monitores coloridos com *look-up table*, existem três "canhões" que bombardeam os três tipos de fósforo existentes na tela. Este bombardeamento é feito utilizando-se uma máscara metálica que permite acessar somente determinados fósforos "gerando a cor do *pixel*". Esta máscara pode ser vista ao lado.



A tendência atual dos dispositivos é eliminar a tabela de cores, são os dispositivos chamados *true color*, por reservarem para cada cor primária 1 *byte*. Na verdade, o olho humano só consegue distinguir aproximadamente 400 mil cores ($< 2^{19}$), logo 19 *bits* deveriam ser necessários. Entretanto, utiliza-se os 24 *bits* pois muitas destas cores são iguais perceptualmente, para a visão humana. A forma esquemática deste tipo de dispositivo pode ser vista abaixo.

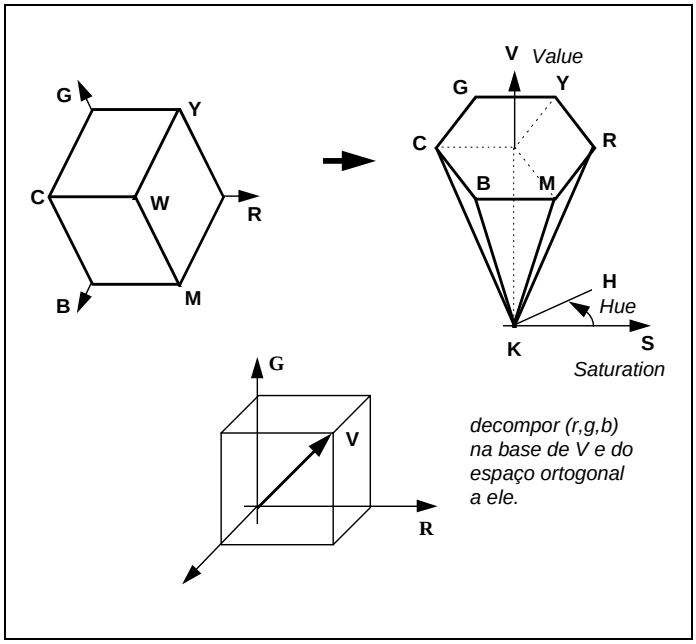


Um aspecto importante, que vale a pena ser ressaltado, são os padrões de varredura vertical de um monitor, que pode ser não-entrelaçado ou entrelaçado. O padrão entrelaçado é aquele que para fazer a varredura utiliza duas etapas (primeiro a dos campos ímpares e depois dos pares), adotando-se este comportamento permite-se frequências mais baixas para o retraçamento. O padrão não-entrelaçado é aquele que faz a varredura de uma única vez, e é o tipo utilizado na computação gráfica.

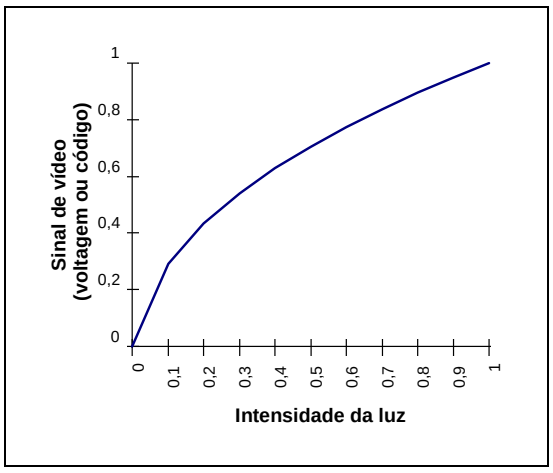


Monitores Entrelaçados
X
Monitores não-Entrelaçados

Continuando a abordagem sobre cores, é muito difícil para uma pessoa determinar uma cor qualquer utilizando o conceito das cores primárias, ou seja, quanto se deve ter de intensidade de vermelho, verde ou azul para a formação de uma determinada cor? Realmente, esta é uma pergunta difícil de ser respondida, uma vez que o sistema mRGB é definido para *hardware*. Desta forma, houve a necessidade da criação de sistemas que fossem mais apropriados para interface com o usuário, de forma que pudesse ser determinada uma cor mais intuitivamente. Um dos sistemas mais apropriados para desempenhar tal tarefa é o HSV (H*ue*-S*aturation*-V*alue*), proposto por A. R. Smith em 1978. Como o próprio nome já diz, o sistema baseia-se na matiz da cor, na sua saturação e intensidade (cores claras/ escuras).



Estas três componentes variam em um sistema de coordenadas cilíndricas, onde a matiz varia ao londo dos círculos horizontais, a saturação varia na direção radial e a intensidade varia em um plano ortogonal ao plano da matiz-saturação. Basicamente, este sistema é uma transformação não-linear do sistema **mRGB**, uma vez que a base desta pirâmide hexagonal corresponde à projeção vista ao longo da diagonal principal do cubo mRGB (da cor branca para a preta), como pode ser visto ao lado.

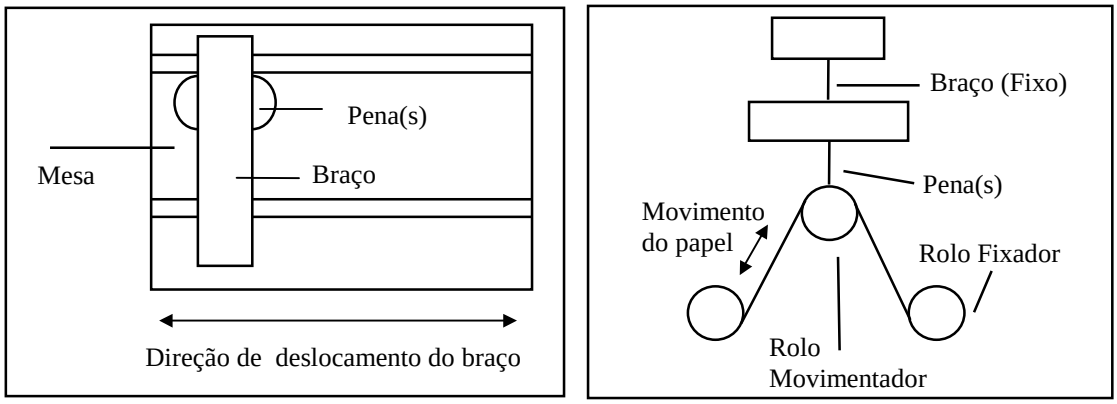


Toda a discussão sobre cores em monitores assume que existe uma relação linear entre o valor real de RGB e a intensidade produzida na tela. Entretanto, esta relação na prática não ocorre. Por exemplo, a intensidade vermelha produzida em um monitor (R_m) dada por uma cor vermelha (R_i) é definida como: $R_m=K(R_i)^{\gamma}$. Este fator γ serve para corrigir perceptualmente as cores uma vez que a visão do olho humano não é linear e sim em uma escala logarítmica. No gráfico ao lado pode-se visualizar como é feita esta correção de intensidades.

Os dispositivos matriciais possibilitam completa variedades de cores, permitindo a geração de imagens com foto realismo (figuras com sombra, reflexão, refração e textura). Atualmente , é a tecnologia dominante no mercado. Entretanto, existem outras duas tecnologias que apareceram na última década: plasma e cristal líquido. Os painéis de plasma tem comportamento semelhante aos do vetoriais de armazenamento, embora a primitiva seja o *pixel*. A ativação deste, é feita através de uma descarga elétrica em um gás (neon), produzidos em dois eletrodos, os painéis são planos, telas transparente e não ocorre *flickering* . Apresentam entretanto, limitada capacidade de interação, pequena velocidade de desenho e resolução apenas razoável. Os dispositivos de cristal líquido, hoje dominam o mercado nas faixas de computadores portáteis do tipo *laptop* e *notebook*. Suas principais vantagens são baixo consumo de energia e tela plana (menor volume e peso). A qualidadede de sua imagem, entretanto, não alcança a qualidade dos terminais baseados em CRT.

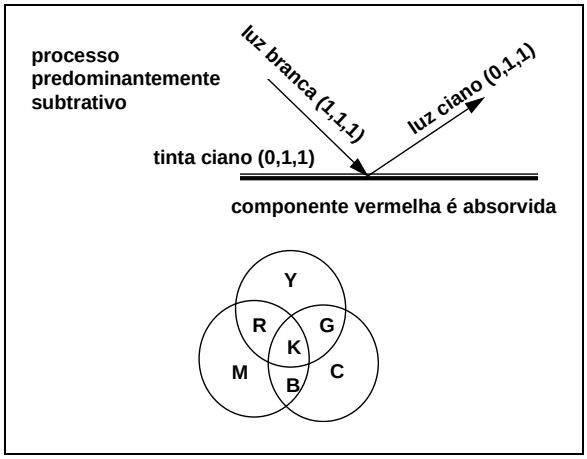
Os **dispositivos de saída passivos** têm como finalidade a produção de cópias permanentes sem interação com o usuário.

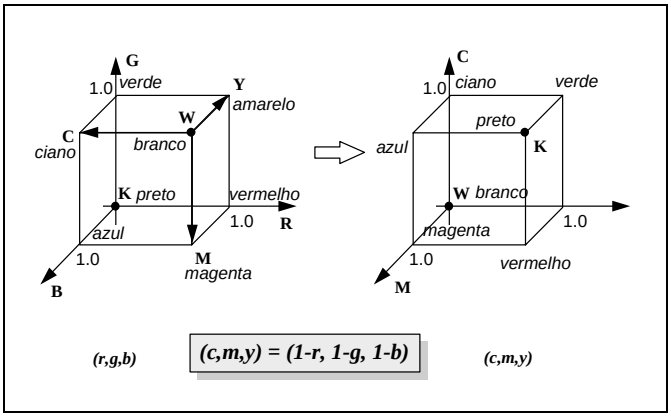
O primeiro tipo destes dispositivos são os vetoriais de acesso randômico, nestes terminais a primitiva gráfica é a reta, que pode ser executada em qualquer posição da superfície de visão (papel, microfilme, transparência, etc.). Pertecendo a esta categoria temos as plotadoras de pena de mesa ou de rolo. As plotadoras de mesa - *flatbed plotters*, possuem uma superfície horizontal onde o papel é preso, um braço mecânico que se movimenta em uma direção da mesa, e nele existem uma ou mais canetas se deslocando perpendicularmente à direção do braço. O funcionamento esquemático deste tipo de terminal é mostrado abaixo. Nas plotadoras de rolo - *drum plotters*, o papel não fica horizontal, um tambor (ou rolo) executa o movimento em uma direção, para frente e para trás , auxiliado por outros dois que seguram o papel, enquanto que sob o braço, agora fixo, as canetas se deslocam na outra direção. As plotadoras de mesa são mais rápidas e precisas que as de rolo. Estas entretanto, apresentam como vantagem uma dimensão do desenho praticamente ilimitada e se justificam em aplicações que não necessitam de muita precisão, dispensando portanto, o uso de uma tecnologia mais cara.



Um segundo tipo dos dispositivos de saída passivos são os matriciais por rastreamento, onde a primitiva gráfica destes dispositivos é o ponto. As figuras são armazenadas em matrizes de pontos, e cada um é associado a um caracter, uma cor e/ou uma intensidade. A impressão se processa linha a linha. Funcionam com esta técnica as plotadoras eletrostáticas, copiadoras de tela , as plotadoras de injeção de tinta, as impressoras matriciais e as impressoras a *laser*. Nas plotadoras eletrostáticas a imagem é obtida em duas fases : primeiro o papel é carregado, ponto a ponto, com cargas do mesmo sinal para posições a serem escurecidas; muito rapidamente a seguir, o papel é exposto a uma tinta com carga oposta e o desenho se forma. Os desenhos geralmente são preto e branco e alguns com tonalidades de cinza. Têm como principal vantagem a velocidade. As copiadores de tela são para reproduzir imagens de um terminal gráfico. É feita a partir do mapeamento da tela e da conversão do sinal de vídeo para dados de plotadoras matriciais. Os resultados são rápidos, porém, de baixa qualidade. As plotadoras de injeção de tinta foram desenvolvidas para representar áreas ao invés de linhas coloridas. O papel passa através de um rolo, onde um braço fixo permite o movimento lento de três injetores de tinta (vermelho, verde e azul), em direção perpendicular ao andamento do papel. As impressoras matriciais e as impressoras a *laser* possuem uma memória gráfica semelhante a um terminal matricial de rastreamento de um *bit* de profundidade. A qualidade da impressão está vinculada à resolução, que varia de 60 a 120 DPI (pontos por polegada) nas impressoras matriciais e de 300 a 600 DPI na impressão a *laser*.

Uma questão importante sobre as impressoras matriciais coloridas é com relação ao sistema de cor utilizado, que não é o mesmo utilizado pelos dispositivos matriciais de rastreamento, que é o sistema aditivo **mRGB**. Nas impressoras coloridas utiliza-se o sistema subtrativo **CMY** (*Cyan*, *Magenta*, *Yellow*). A escolha deste sistema é simples, porque no papel o processo de percepção de cor é diferente, pois neste caso a luz é refletida no papel e não emitida como nos monitores, como a reflexão é um processo subtrativo utiliza-se então as cores complementares do RGB, ou seja CMY.

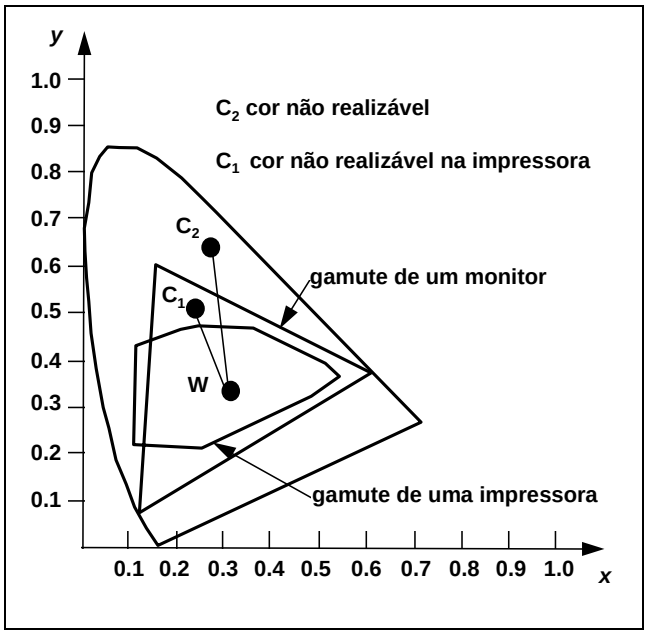




A conversão do sistema **mRGB** para o **CMY** é trivial, devido a complementaridade tem-se:
(c,m,y) = (1-r,1-g,1-b).

No sistema **CMY** para se obter o preto deve-se subtrair todas as cores. Nota-se que a cor atingida não é propriamente o preto mas sim, uma tonalidade próxima do cinza escuro. Desta forma, criou-se um outro sistemam **CYMK**, em que a letra K representa a cor preta (*blacK*). Sendo assim, a cor preta foi adicionada ao sistema de cor e tratada praticamente como uma cor primária independente.

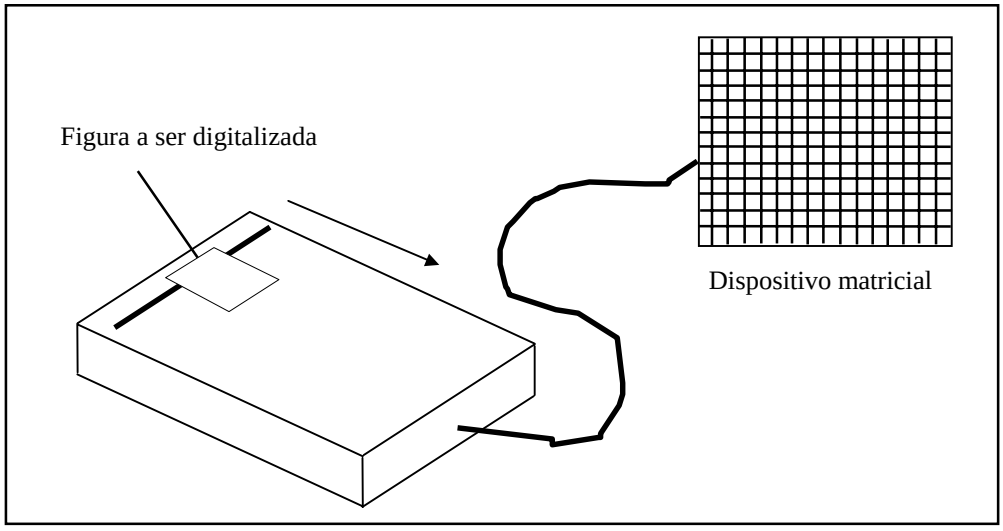
Um problema clássico em computação atualmente está ligado a representação de cores, ou seja, dada uma determinada figura colorida na tela, é uma tarefa muito difícil reproduzi-la em uma impressora colorida da mesma forma. Este fato pode ser facilmente visualizado com o auxílio do **Diagrama de Cromaticidade do CIE** quando se coloca os gamutes de dispositivos: um monitor e uma impressora coloridos.



Existem diversos tipos de **dispositivos gráficos de entrada** utilizados na computação gráfica e que podem ser classificidados de acordo com a tarefa de interação que eles desempenham. Merecem destaques os localizadores, selecionadores e valoradores. Os localizadores são dispositivos que permitem obter a localização (coordenadas) de um ponto na superfície de visão (ex. terminal gráfico). O mecanismo físico típico desta classe é a mesa digitalizadora - *data tablet*. A mesa digitalizadora é capaz de reconhecer a posição de uma caneta (*stylus*), através de pertubações magnéticas . Um outro dispositivo que funciona como localizador é o *touch panel*. Os valoradores servem para obter uma valor real dentro de uma escala pré-fixada. O dispositivo físico mais representativo deste conceito é o potenciômetro. Existem também o *joystick*, *track ball* e o *mouse*. Os selecionadores selecionam objetos de uma figura. O dispositivo natural para representar um selecionador é a caneta ótica - *ligth pen*. A caneta ótica é um dispositivo que é capaz de perceber quando os elementos do desenho estão sendo redesenhados em um terminal do tipo vetorial dinâmico ou matricial de rastreamento. Desta forma, o elemento de desenho e o segmento a que ele pertence são enviados ao computador.

Existem ainda um segundo grande grupo que tem por objetivo a **captura de imagens**. Pode-se dividir este grupo dedendendo do tipo de imagem a ser capturada: os *scanners* que capturam imagens impressas e os *frame grabbers* que capturam imagens de vídeo.

Os *scanners* são muito semelhantes às máquinas foto-copiadoras, onde uma fonte de luz em forma de uma linha varre a imagem impressa e mede a quantidade de luz refletida ou transmitida em cada ponto. A luz refletida é convertida em sinal elétrico através de um conjunto de foto-detetores que também formam uma linha. O sinal elétrico é finalmente digitalizado e enviado ao computador. Existem dois tipos de *scanners* os manuais e o de mesa. A diferença entre eles é com relação a verredura, que no primeiro caso é feita manualmente e no segundo caso é feita automaticamente, onde o próprio aparelho desloca a fonte de luz e os foto-detetores. A resolução da imagem pode variar de 70 até 4000 DPI, que podem gerar imagens de altíssima qualidade porém com um consumo de memória extremamente alto (Projeto Portinari - qualidade da imagem de um *slide* aprox. 245 *Mbytes*).



Frame Grabbers são dispositivos que permitem que uma imagem de vídeo gerada por uma câmera ou um vídeo-cassete possa ser digitalizada diretamente a partir do sinal elétrico que a representa, que em geral está disponível como uma saída do equipamento. O sinal de vídeo é organizado em linhas de imagem, que contem um sinal análogico cada uma, dependendo do sistema utilizado (PAL ou NTSC). Como se trata de um sinal contínuo para uma televisão, não se tem somente uma única imagem estática e sim várias imagens (30 quadros por segundo). Portanto, para se adequarem a este sincronismo, digitalizadores de sinal de vídeo são bem mais rápidos que *scanners*, e geralmente fornecem imagens pequenas e de uma qualidade inferior a estes.

Todos os aplicativos existentes atualmente, possuem um apelo visual muito grande, pois têm um alta interação com os usuários. Desta forma, hoje em dia, é inconcebível que programas não possuam dispositivos de interação como ícones, janelas, menus, etc. Até bem pouco tempo atrás, estes aplicativos eram previlégios somente das grandes e caras estações de trabalho, entretanto, por razões econômicas, há uma migração destes *software* gráfico para plataformas PCs. Entretanto, o desempenho destes aplicativos eram extremamente baixos, havendo então a necessidade de uma mudança tecnológica tão avassaladora quanto a provocada pelo próprio surgimento da GUI. Desta forma, surgiram placas gráficas (aceleradores gráficos), que possuem um processador dedicado e memória RAM, liberando assim, a CPU de grande parte do processamento gráfico, tornando então viável, a migração destes aplicativos. Existem três grandes áreas de atuação destas placas : gráficos 2D, gráficos 3D e vídeo.

A aceleração em gráficos 2D, como é chamada, consiste em acrescentar rapidez a operações *BitBlt's*.

A utilização dos gráficos 3D está cada vez mais emergente e encontram-se nos pacotes comerciais de CAD, aplicações científicas, em jogos, em interfaces de sistemas de realidade virtual e em modelagens de simulações realísticas. O *software* 3D dá a ilusão de profundidade, efeitos de luz, sombra e movimento em um espaço tridimensional, construindo um mundo extremamente próximo da realidade. A aceleração 3D desta forma, é um processo minucioso que envolve várias etapas, oferecendo realismo em troca de um alto custo em desempenho. As principais etapas que envolvem o *rendering* podem ser assim enumeradas: filtro para o *rendering* 3D, introdução de luzes e sombras, projeção e aplicação de texturas.

Na área de multimidia, um mercado cada vez mais explorado na área de informática, encontram-se então as placas aceleradoras de vídeo. Na verdade, este nome aceleradora não é o corretamente empregado, uma vez que nenhum clipe será exibido em uma velocidade maior do que foi gerado, ou aumentará a resolução ou grau de detalhe da imagem. O que este dispositivos realmente fazem é aumentar as proporções do vídeo para além do seu tamanho natural - tornando mais fácil a visualização de detalhes - sem reduzir a velocidade de exibição e uma consequente perda de quadros.

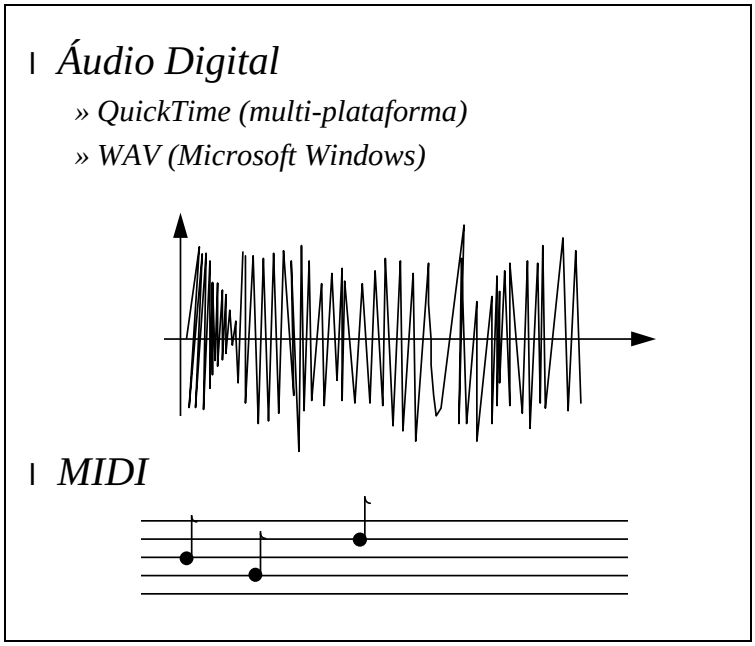
O sistema de cores geralmente empregados são os chamados sistemas de cor baseados em decomposição da luminância-crominância. Este sistema tem como princípio que o sistema visual humano tem menor sensibilidade para detectar variações de cor do que variações de luminância

Um problema crítico para utilização de vídeos é quanto ao armazenamento. Por exemplo, um único quadro não comprimido de 24 *bits*, digitalizado com uma resolução de 640 por 480 pontos (VGA) precisa de aproximadamente 900 *kbytes* para ser armazenado. Lembrando que deve-se mostrar 30 quadros por segundo, um CD-ROM conseguiria exibir apenas 25 segundos de vídeo. Sendo que para isso, deveriam existir leitoras de CD-ROM que operem a uma taxa cinco vezes maior que as de uma unidade de CD-ROM com velocidade quádrupla. Os produtores de *software* multimidia resolveram este problema de dois modos: comprimiram o vídeo usando chips conhecidos como *codec* e digitalizaram os clipes em uma resolução de 160 por 120 ou 320 por 240. Entretanto, este trabalho acaba criando uma sobrecarga nas CPUs, e sendo assim, as aceleradoras de vídeo reduziram em muito este problema, tornando para si a tarefa de ajustar as dimensões que podem ser vistas confortavelmente. Com relação a este aspecto, deve-se lembrar que a ampliação de imagens exige a adição de novos *pixels*, por meio de replicação, interpolação ou as duas técnicas combinadas.

O funcionamento básico destes aceleradores é ler o arquivo .AVI para que se conheça o proecesso *codec* utilizado para comprimir o clipe. A partir daí é chamado o *codec* apropriado, que começa a recompor o vídeo em sua resolução e espaço em cores originais, normalmente em **YUV** (um sistema de padrão que define as cores em função da crominância e luminância). A partir daí, é feito o tratamento adequado, e então convertida a imagem para **RGB** (sistema de cores da placa).

Finalizando esta parte sobre equipamentos, será abordado as placas de som, uma vez que atualmente o "visual" somente não é suficiente. As imagens transmitem todas as informações possíveis, mas a música e os efeitos sonoros podem ajudar a prender a atenção da audiência por um tempo maior.

De uma forma geral, acrescentar som a sua aplicação é feita basicamente por duas tecnologias distintas: áudio digital e MIDI (*Musical Instrument Digital Interface*, ou Interface Digital para Instrumentos Digitais).



O áudio digital permite que se grave e/ou reproduza qualquer tipo de som, com uma alta qualidade (captura a expressividade dos instrumentos), principalmente quando se trata da voz humana. As gravações soam iguais quando reproduzidas em qualquer equipamento. Entretanto, necessita de grandes quantidades de armazenamento, por exemplo, se fosse alcançado a satisfação do ouvido humano - sons na freqüência de 40 kHz (40 vezes por segundo) - um ponto de 16 *bits* em dois canais estereofônicos nesta freqüência seriam necessários aproximadamente 10 *Mbytes* por minuto. O que realmente se faz, em muitos *software* na área de multimidia é adotar metade desta freqüência em um canal mono com 8 *bits*.

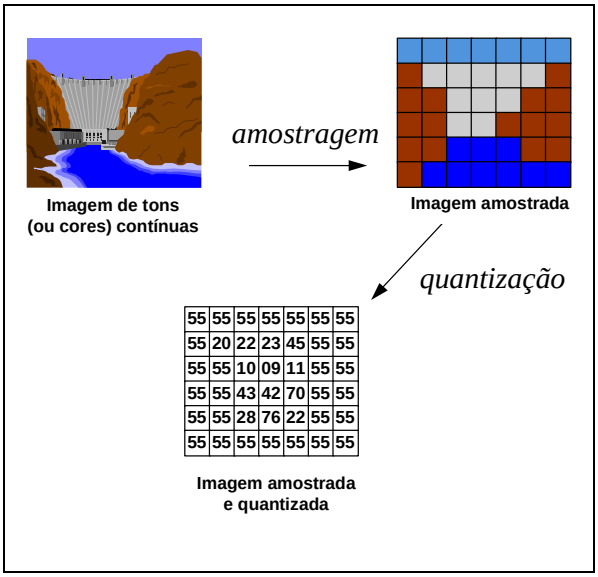
A MIDI permite que se grave ou reproduza apenas as notas geradas pelo *software* MIDI ou por instrumentos. A grande vantagem de se usar MIDI é que pode-se controlar inteiramente a música (reproduzir o som em qualquer instrumento que o sintetizador da placa de som possa tocar) e controle completo da edição, uma vez que, com MIDI não é armazenado o som propriamente dito e sim dados, que podem ser editados como se faz em edição de textos. Sendo assim, os dados armazenados ocupam bem menos espaço do que uma gravação digital. Entretanto, a MIDI, possue duas grandes desvantagens a música geralmente soa diferente quando tocada por sintetizadores diferentes e instrumentações sintetizadas normalmente perdem a expressividade que uma gravação em audio digital consegue capturar.

Capítulo 4 - Imagens Digitais

Neste capítulo serão abordados os conceitos fundamentais sobre imagem digital, examinando desde a sua criação, convenções, termos, limitações até o seu armazenamento.

Uma imagem (natural) é uma variação **contínua** de tons e cores. No caso de uma fotografia, por exemplo, os tons variam de claros a escuros e as cores variam de vermelho até azul, abrangendo desta forma, "todo" o espectro de cores visíveis. Estas variações sempre se dão de forma contínua (sem variações abruptas ou "degraus") de modo a reproduzir fielmente a cena original.

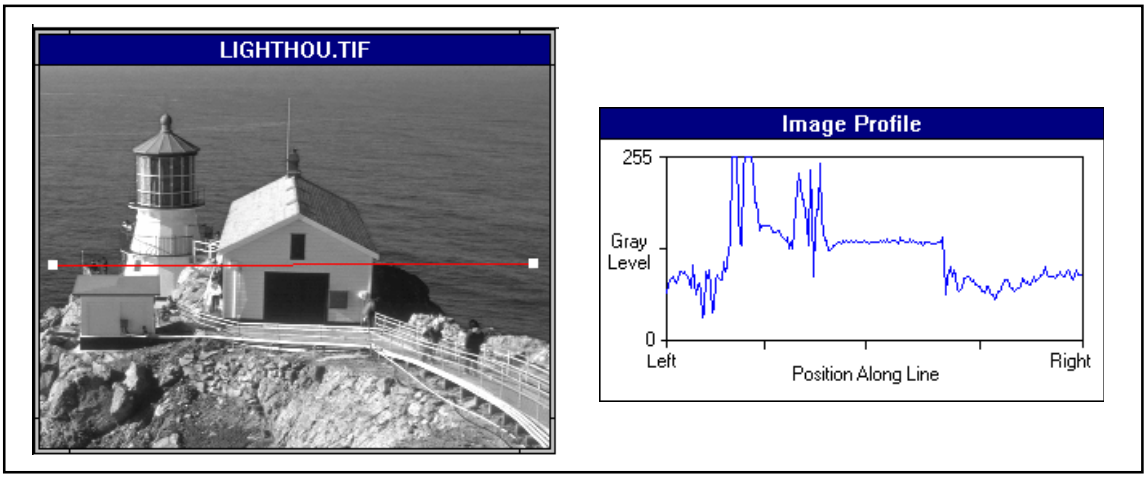
Uma imagem digital entretanto, é composta por pontos **discretos** de tons e/ou cores, ou brilho, e não por uma variação contínua. Para a criação de uma imagem digital, deve-se dividir a imagem contínua em uma série de pontos que irão possuir uma determinada tonalidade (*gray-scale*) ou cor (colorido). Adicionalmente a este processo de divisão, deve-se descrever cada ponto por um valor digital. Os processos de divisão da imagem contínua e determinação dos valores digitais de cada ponto são chamados de amostragem e quantização, respectivamente, e são ilustrados na figura ao lado. A combinação destes dois processos é o que se denomina de digitalização de imagens, sempre se referindo ao *pixel*, por ser a menor unidade de atribuição de uma imagem digital.



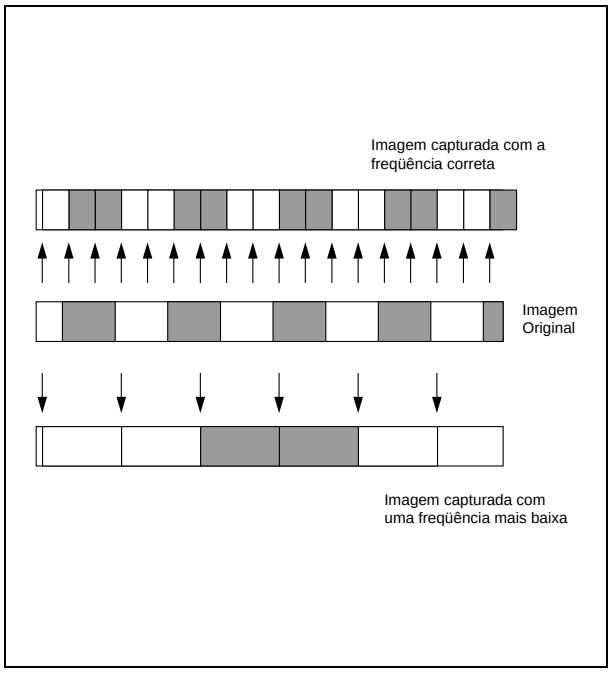
Analisando-se a figura anterior, pode-se questionar a qualidade da digitalização de uma imagem. Esta qualidade está diretamente relacionada com o número de *pixels* e linhas, e com a gama de intensidades de brilho que se pode ter em uma imagem. Estes dois aspectos são conhecidos como resolução da imagem, que pode ser definida por dois fatores: a "resolução espacial" e a "resolução de brilho" (ou "resolução de cores" no caso de se tratar de imagens coloridas). Quando se trata de movimento, um outro aspecto importante é a taxa que se apresenta cada quadro, de modo que se tenha a ilusão do movimento.

O termo "resolução espacial" (no noso caso, espacial se refere ao espaço 2D) é usado para descrever quantos *pixels* compõem a imagem digital, desta forma, quanto maior o número de *pixels* maior será a "resolução espacial". O número de *pixels* da imagem digital depende do refinamento da amostragem.

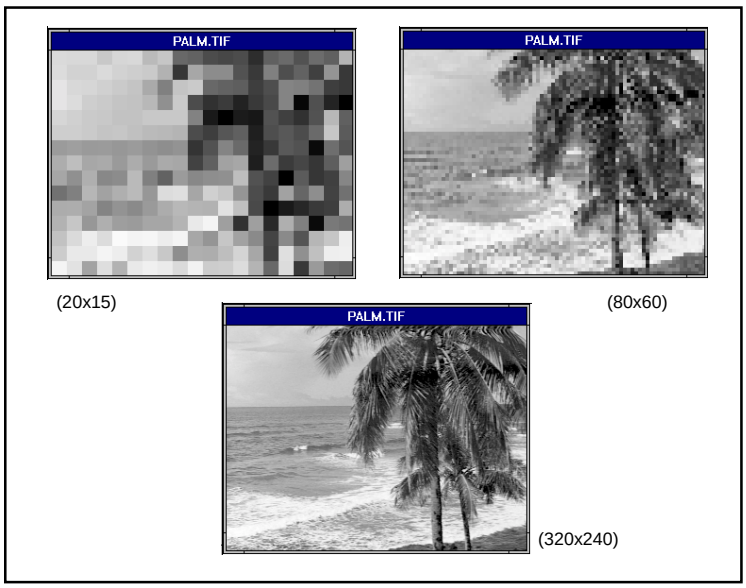
De modo a determinar o grau de refinamento que uma imagem deve possuir, deve-se definir o conceito de "freqüência espacial". Partindo do princípio que todas as imagens possuem detalhes e que estes são formados por transições de brilho que variam ciclicamente, ou seja, de tons escuros para claros e voltando para tons escuros (*gray-scale*), define-se "freqüência espacial" como sendo a taxa de variação da mudança de brilho. A figura a seguir apresenta uma figura e um gráfico mostrando o brilho de cada *pixel* ao longo de uma linha definida na imagem. Esta imagem apresenta detalhes diferentes, dependendo da região da imagem que se olha. Na região da casa por exemplo, tem-se uma variação muito suave dos detalhes, consequentemente a variação de brilho é muito pequena. Nesta região diz-se que a "freqüência espacial" é baixa. Na região do farol, entretanto, a variação do brilho se dá rápida e bruscamente, dizendo assim, que tem-se uma região com uma alta "freqüência espacial".



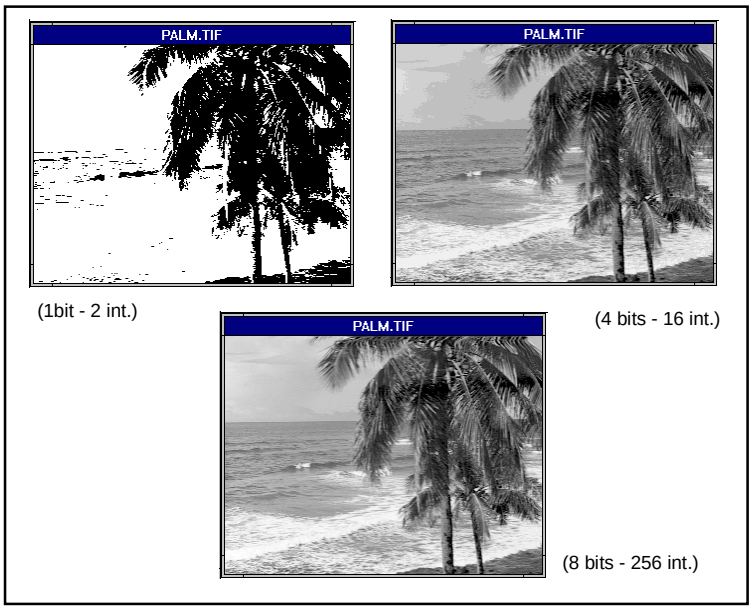
Para determinar a taxa de amostragem necessária que uma imagem digital deve ter para conseguir "capturar" todos os detalhes contidos em uma imagem natural (contínua), deve-se usar o teorema da amostragem que diz: "Deve-se amostrar uma imagem numa taxa pelo menos duas vezes maior que a maior freqüência contida nesta". Este teorema assegura que pelo menos duas amostras irão captar o detalhe desta fre-qüência. A figura ao lado ilustra o fato de que ao se adotar freqüências menores perde-se detalhes da imagem. Se a taxa de amostragem for inferior a taxa dada pelo teorema da amostragem, detalhes que possuam freqüên-cias espaciais altas serão perdidos na imagem digital. Sendo assim, a imagem digital aparen-ta ter uma resolução espacial menor do que da imagem original, isto por que não existem *pixels* sufucientes para representar adequada-mente os detlhes contidos na imagem original.



Pode-se observar na figura abaixo o efeito de amostrar uma imagem em diferentes resoluções espaciais.



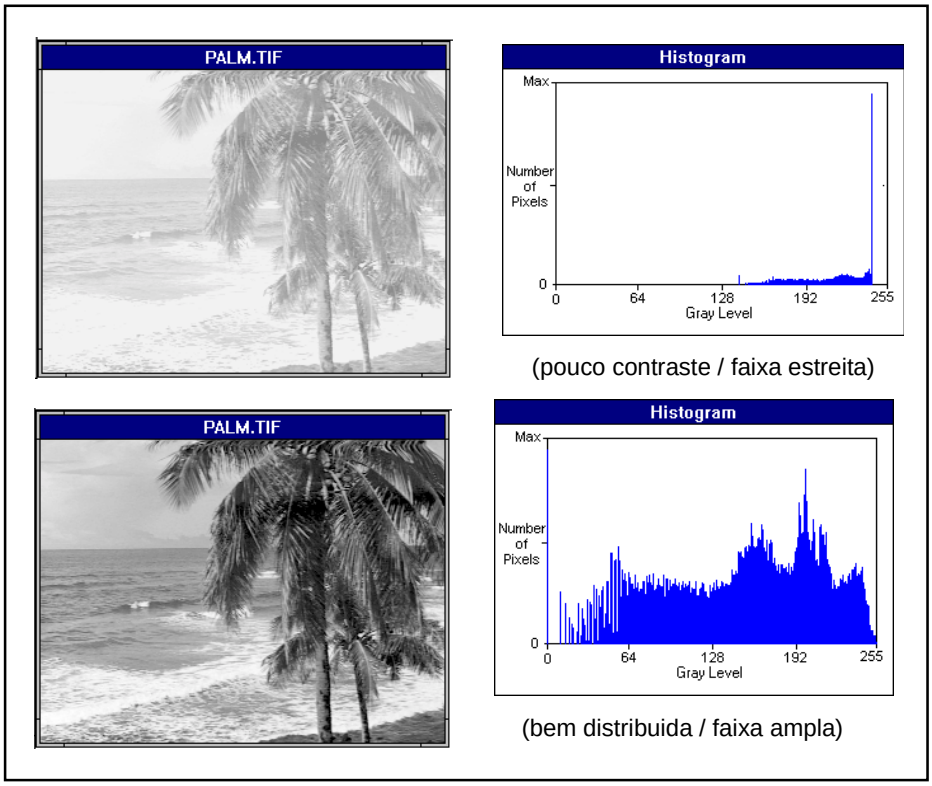
Cada *pixel* em uma imagem digital representa a intensidade luminosa de um determinado ponto da imagem original. Sendo assim, o conceito de "resolução de brilho" refere-se à quão preciso é o brilho de cada *pixel* para representar a intensidade luminosa da imagem original. Com já foi visto, após o processo de amostragem, cada amostra é quantizada. Este processo de quantização converte uma intensidade de tons contínuos, em um valor de brilho. A precisão deste valor digital está diretamente relacionada com o número de *bits* que serão utilizados na quantização. Como exemplo, adotando-se uma imagem digital que possua somente tons de cinza, se forem utilizados 3 *bits*, o brilho pode ser convertido em somente 8 tons de cinza, ao passo que se forem utilizados 8 *bits*, este valor passará para 256 tons. A figura a seguir apresenta imagens com diferentes "resoluções de brilho".



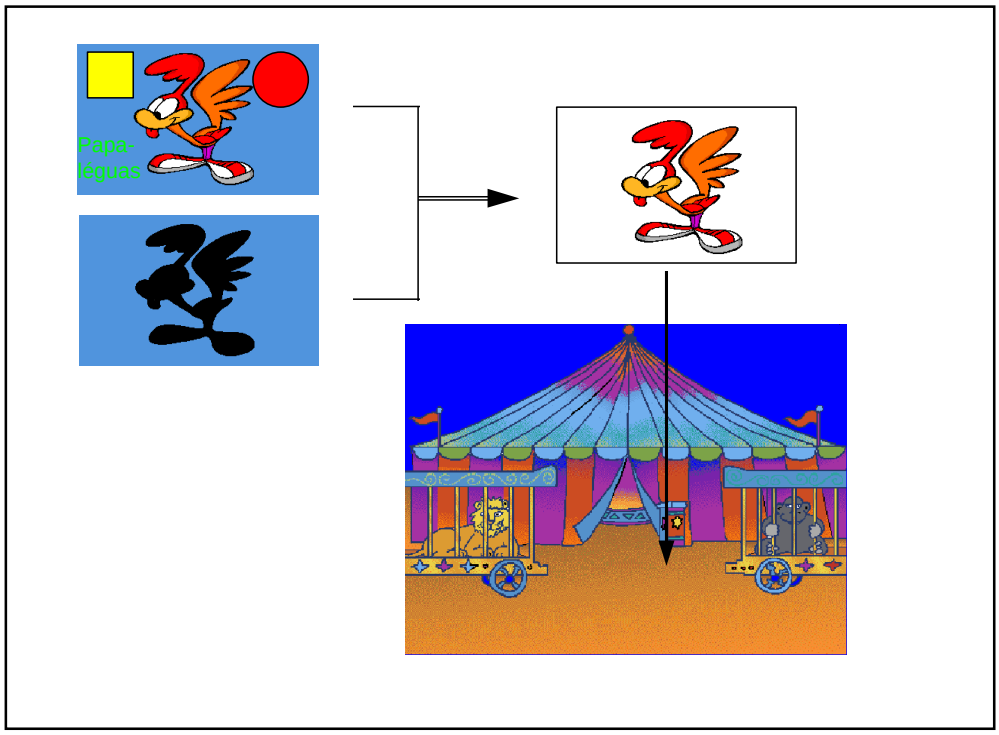
A qualidade de uma imagem digital é função das resoluções espaciais e de brilho, como exposto anteriormente. Uma ferramenta importante que pode auxiliar na medida de qualidade de uma imagem é o histograma de brilho (imagens com tons de cinza) ou histograma de cores (imagens coloridas). Utilizando tais histogramas pode-se visualizar certas deficiências da imagem e até procurar fazer algumas correções de modo a minimizar estas deficiências. Nos parágrafos seguintes será dado enfoque somente aos histogramas de brilho, uma vez que histogramas de cores nada mais são do que três histogramas de brilho, um para cada componente de cor.

Em termos gerais, um histograma é uma distribuição gráfica de uma série de números. Um histograma de brilho é uma distribuição gráfica de tons de cinza dos *pixels* de uma imagem digital. Um histograma apresenta na escala horizontal o "brilho" (utilizando-se uma imagem *8-bit gray-scale*, valores qua vão de 0 a 255), e na escala vertical o número de *pixels*. Este histograma apresenta um modo extremamente conveniente de representar a concentração de *pixels* versus o brilho da imagem. Utilizando este gráfico pode-se facilmente visualizar se uma imagem é clara ou escura, ou se possui muito ou pouco contraste. Além disto, pode-se utilizá-lo como base de operações de melhoria de contraste para que a imagem se torne mais agradável ou que seja mais facilmente interpretada por um observador.

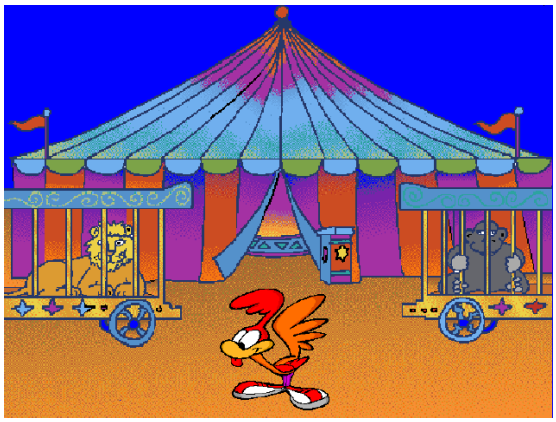
Ilustrando a utilização destes histogramas, observa-se na parte superior da figura seguinte, que esta apresenta pouco contraste, ou seja, concentração dos *pixels* em uma faixa estreita do histograma. Na parte inferior entretanto, apresenta-se um contraste bem balanceado ocupando toda a área do histograma.



Na área de manipulação de imagens digitais, existem vários assuntos interessantes que poderiam ser abordados. Neste curso será abordado somente, operações para combinação de imagens. Como resultado destas operações, obtem-se uma nova cena que provavelmente nunca existiu na realidade. Desta forma, imagens que eram feitas em laboratórios fotográficos, atualmente, com o avanço das técnicas utilizadas no processamento de imagens, podem ser feitas rapidamente e com um custo menor.



A idéia básica do processo pode ser ilustrada na figura da página anetrior e ao lado, onde deseja-se mostrar o circo com o papa-léguas, tendo-se duas imagens distintas a do circo e do papa-léguas. A primeira etapa é definir uma "máscara" para que se consiga capturar somente o papa-léguas, o que se consegue subtraindo-se a imagem original pela máscara. A seguir define-se onde colocar o papa-léguas criando-se então a imagem final composta, utilizando-se o conceito de transpa-rência.



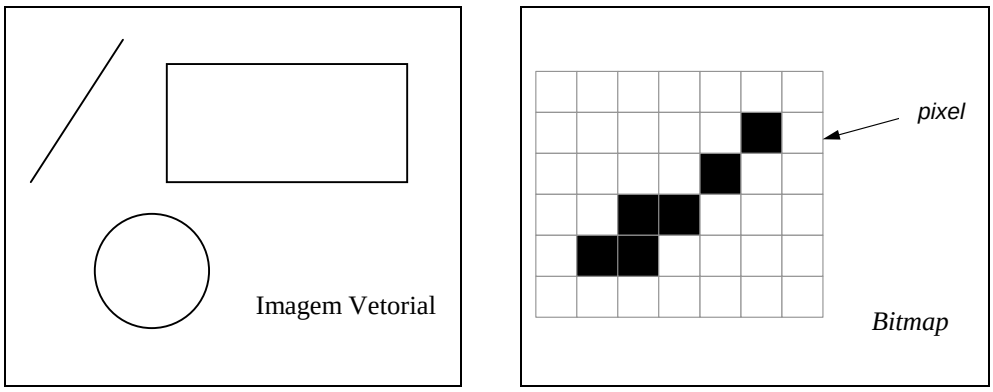
Seguindo a mesma idéia de composição de imagens tem-se a animação por *sprites*. As diferenças básicas são que existem várias figuras que serão compostas, dependendo da interação com o usuário, e que antes de se compor a imagem deve-se armazenar a região da tela que está se colocando a imagem, e nos quadros subsequentes restaurá-la, armazenar a nova região da tela e colocar a imagem.

Um assunto de extrema importância quando se trata de imagens digitais é o armazenamento destas. Como já foi visto, em imagens que possuam muitos detalhes, e que estes tenham uma alta taxa de variação da mudança de brilho, é necessário também, um grande número de *pixels* para poder representar tal imagem ("resolução espacial"). Aliado a este fato, tem-se também a resolução de brilho que indica o número de *bits* necessários para representar uma determinada intensidade luminosa. Por exemplo, uma imagem que consiga reproduzir 256 tons de cinza e que possua o tamanho de um monitor VGA, ocupará um espaço de 300 *kbytes* (640x480x8 *bits*). Pode-se então deduzir que para imagens coloridas e maiores este espaço aumentará exponencialmente. Com esta preocupação, foram desenvolvidos vários processos de compressão de imagens tendo-se então dois grupos distintos: um sem perda e outro com perda. Os métodos sem perda preservam exatamente o conteúdo da imagem e chega a taxas de compressão da ordem de 1/3, dependendo da imagem a ser comprimida. Os processos com perda preservam o nível da qualidade da imagem, chegando em alguns casos a taxas de compressão da ordem de 1:100.

O método de compressão sem perda mais simples chama-se RLE - *Run-Length Encoding*. A idéia básica deste processo é eliminar as redundâncias. Em uma imagem digital é comum ter-se a mesma intensidade para uma série de *bits*, sendo assim, o que se faz é dizer quantas vezes aquela intensidade se repete. Por exemplo, ao armazenar a *string* "AAAAAAxxxBBBBBB" seriam gastos 15 caracteres, entretanto, usando o método RLE o armazenamento desta *string* ocuparia um espaço bem menor, pois seria armazenado somente "6A3x6B". Sendo assim, dependendo da imagem pode-se conseguir altas taxas de compressão, ou em determinados casos até aumentar o espaço de armazenamento.

Tratando-se ainda de armazenamento de imagens digitais, dependendo do tipo de imagem a ser armazenada pode-se ter a seguinte classificação:

- Imagens Vetoriais (AutoCAD DXF, Microstation DGN),
- *Bitmaps* - *pixel* a *pixel* (BMP, PCX, GIF, TIFF),
- *Metafiles* = Vetorial + *Bitmaps* (CGM, Microsoft Windows Metafile).



Apenas com caráter ilustrativo, nesta questão sobre armazenamento, dois tipos de aplicações vem ganhando destaques cada vez maiores na área de informática e criando formatos próprios para armazenamento que são:

- Animação (FLI, GRASP),
- Multimídia (RIFF, Apple QuickTime).

Finalizando, será analisado o formato de armazenamento Microsof Windows Bitmap - BMP, que apresenta as seguintes características:

a) o armazenamento pode ser feito utilzando-se 1-*bit*, 4-*bit*, 8-*bit*, 24-*bit* por *pixel*,
b) pode-se adotar o tipo de compressão RLE, entretanto, a maioria destes arquivos não é comprimido,
c) o tamanho máximo para armazenamento destes arquivos é 64k x 64k *pixels.*

As seções (versão 3.x em diante) que devem ser lidas e/ou escritas deste tipo de arquivo se encontram a seguir:

a) **Header**

```
typedef struct _Win3xBitmapHeader          {
  WORD    Type;         /* Image file type 4D42h ("BM")   */
  DWORD   FileSize;     /* File size (bytes)              */
  WORD    Reserved1;    /* Reserved (always 0)            */
  WORD    Reserved2;    /* Reserved (always 0)            */
  DWORD   Offset;       /* Offset to bitmap data  in bytes */
  } WIN3XHEAD;
```

b) **Information Header**

```
typedef struct _Win3xBitmapInfoHeader      {
  DWORD  Size;                  /*  Size of this Header  (40)        */
  DWORD  Width;                 /*  Image width (pixels)             */
  DWORD  Height;                /*  Image height (pixels)            */
  WORD   Planes;                /*  Number of Planes (always=1)      */
  WORD   BitCount;              /*  Bits per pixel (1/4/8 or 24)     */
  DWORD  Compression;           /*  Compression (0/1/2)              */
  DWORD  SizeImage;             /* Size of bitmap (bytes)            */
  DWORD  XPelsPerMeter;         /* Horz. resol.(pixels/m)            */
  DWORD  YPelsPerMeter;         /* Vert. resol.(pixels/m)            */
  DWORD  ClrUsed;               /* Num of colors in the image        */
  DWORD  ClrImportant;          /* Num of important colors           */
  } WIN3XINFOHEADER;
```

c) **Pallete**

```
typedef struct _Win3xPalette        {
  RGBQUAD Palette[ ];               /* 2, 16, or 256 elem.      */
  } WIN3XPALETTE;

typedef struct _Win3xRgbQuad        {
  BYTE  Blue;             /* 8-bit blue component             */
  BYTE  Green;            /* 8-bit green component            */
  BYTE  Red;              /* 8-bit red component              */
  BYTE  Reserved;         /* Reserved (= 0)                   */
  } RGBQUAD;
```

d) **Image Data**

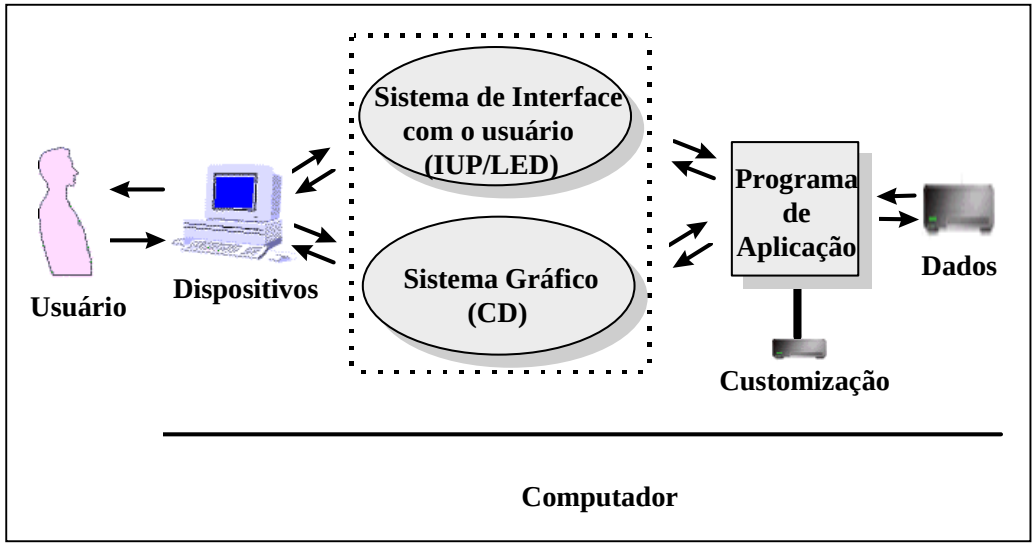
Armazena os valores de cada *pixel*.

Notas:

- Cada *scan line* em um arquivo BMP é sempre um múltiplo de 4.
- Imagens com1-, 4-, e 8-*bits* usam uma palheta de cores.
- Imagens com 24-*bits* guardam a cor diretamente, na ordem azul, verde e vermelho.
- O armazenamento da imagem é sempre feito a partir do canto esquerdo inferior.
- BYTE - *byte*.
- WORD/UINT - positivo com 2 *bytes*.
- DWORD/LONG - inteiro (positivo) com 4 *bytes*.

Capítulo 5 - Sistemas Gráficos

O modelo conceitual de programação gráfica pode ser visualizado na figura seguinte. Nota-se que ao longo do percurso da informação existem as seguintes interfaces: (1) do usuário com o dispositivo; (2) das funções gráficas com o dispositivo; (3) do programa de aplicação com as funções gráficas e (4) do programa de aplicação com os dados. Cada uma destas interfaces é objeto de discussão e padronização na ISO (*Internacional Organization for Standartization*), na ANSI (*American National Standards Institute*), na ABNT (Associação Brasileira de Normas Técnicas) e em outros órgãos normativos dos demais países.



A padronização da interface com o usuário permite que pessoas treinadas na utilização de um programa possam utilizar outros com um mínimo de aprendizado. Nesta área destacam-se :

- *Motif* da *OSF (Open Software Foundation)*
- *Desktop* da *Macintosh*
- *Windows* da *Microsoft*

A padronização dos arquivos que descrevem figuras permite que desenhos gerados em um sistema possam ser exportados para outros. Segundo a ISO, o formato oficial de desenhos é o CGM (*Computer Graphics Metafile*). Existem, entretanto, diversos padrões de fabricantes. Entre eles destacam-se:

- *Postscript* da *Adobe*
- DXF da *Autodesk*
- DGN da *Intergraph*

A padronização da interface dos dados da aplicação permite que modelos analisados por um sistema possam ser lidos e tratados por outros. Para área de CAD/CAM a ANSI propõe o IGES (*Inicial Graphics Exchange Specification*) como padrão.

A interface com os dispositivos refere-se ao protocolo de conversão de comandos independentes de dispositivos para comandos do próprio equipamento. A ANSI e a ISO possuem uma proposta para um a interface bidimensional, denominada CGI (*Computer Graphics Virtual Device Interface*).

A padronização da interface do programa de aplicação com as funções gráficas foi criada com o objetivo de eliminar os principais problemas existentes, na década de 60, na área de computação gráfica. Uma vez que, os programas da época utililizavam comandos específicos do próprio terminal, escreviam diretamente na memória gráfica, e utilizavam rotinas fornecidas pelo fabricante. Esta metodologia gerava problemas para transferência de programas gráficos de uma instalação para outra, ou seja, os programas eram dependentes de equipamentos. Sendo assim, houve necessidade de elaborar programas gráficos compatíveis com qualquer equipamento.

Atualmente, esta interface é a que tem recebido maior atenção por parte das comunidades acadêmica e industrial. Nesta área encontram-se os sitemas de interface com o usuário e os sistemas gráficos básicos. Neste capítulo serão abordados os **sistemas gráficos** baseados em transformações *window-to-viewport*. A ISO adota atualmente, dois padrões para esta área: o **GKS** (*Graphical Kernel System*) e o **PHIGS** (*Programammer's Hierarquical Interactive Graphics System).* Hoje em dia, o sistema gráfico mais utilizado é o **OpenGL.**

O sistema gráfico CD foi desenvolvido pelo TeCGraf e pela PETROBRAS de modo a atender à grande maioria das aplicações na área da computação gráfica. É um sistema baseado na transformação *window-to-viewport* e tem como principais características ser multi-plataforma e conseguir tratar imagens (*pixel*). As rotinas existentes no CD podem ser agrupadas, segundo suas características funcionais:

- Controle da Superfície de Visualização.
- Primitivas de saída.
- Atributos.
- Sistemas de Coordenadas e *clipping*.
- Sistema de cor
- Imagem

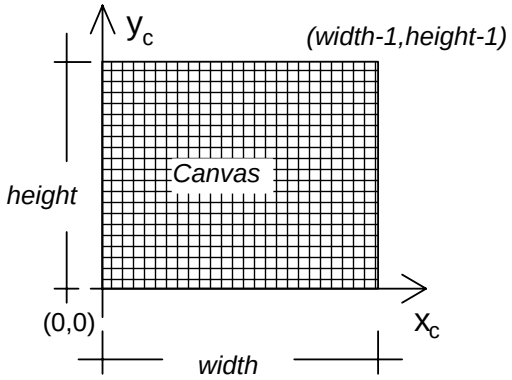
Como o CD é um sistema gráfico independente de equipamentos, a área de desenho é chamada genericamente de superfície de visualização, **VS (Visualization Surface).** Uma VS pode ser um *canvas*, um papel, uma impressora ou plotadora, uma área de transferência do sistema nativo (*clipboard*), um arquivo *postscript* ou um arquivo *metafile*. Desta forma, é necessário que se crie um *canvas*, de acordo com a VS a ser utilizada, o que pode ser feito utilizando a função **`cdCreateCanvas`**.

As funções primitivas de desenho do CD, como por exemplo a **`cdLine`** (para desenhar uma linha), não possuem na sua lista de parâmetros nenhuma referência para onde a linha deve ser desenhada e, obviamente, um mesmo programa de aplicação pode desenhar em diversas destas VS durante sua execução. Esta flexibilidade, entretanto, é limitada a se desenhar em uma VS de cada vez. O CD desenha apenas na VS que estiver **ativa**. Através da função **`cdActivate`** o programa desativa a VS corrente e ativa uma outra. Assim, por exemplo, um programa pode ativar um *canvas*, desenhar nele, ativar outro *canvas*, desenhar, ativar o *clipboard*, desenhar e assim sucessivamente.

Quando a superfície de visualização for um papel de uma impressora ou plotadora a função **`cdFlush`** representa um avanço da folha.

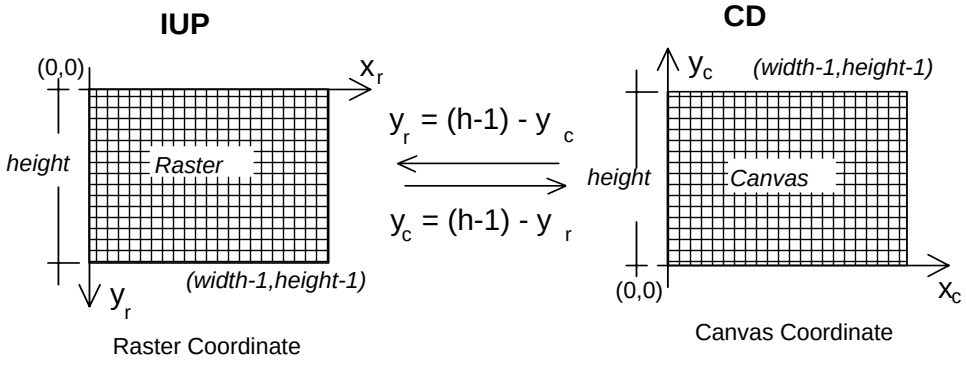
Uma última função relacionada com as superfícies de visualização como um todo é a função **`cdClear`** que limpa toda sua área (incluindo a região fora do retângulo de *clipping*).

As coordenadas nativas do CD são *raster*, ou seja em *pixels*, sendo que a origem se situa no canto inferior esquerdo. Informações sobre o tamanho do *canvas* tanto em *pixels* como em milímetros podem ser obtidas através da função **`cdGetCanvasSize`**.

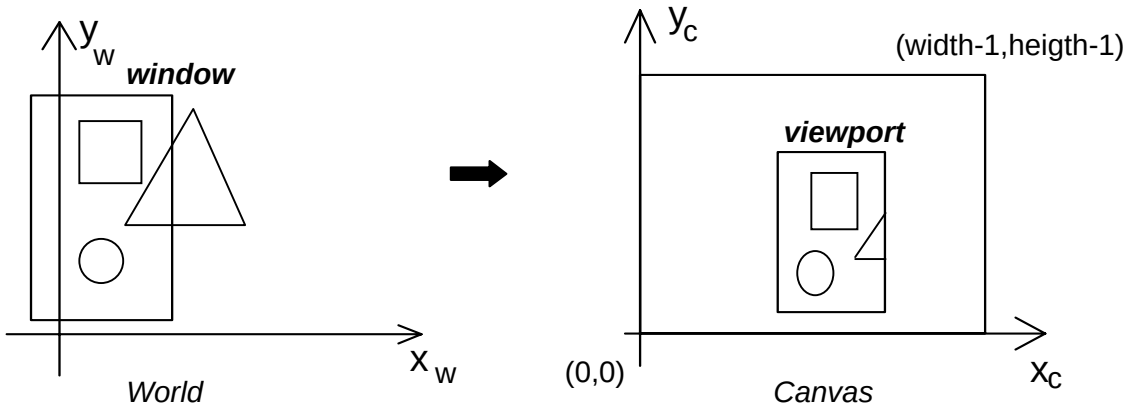


O CD permite ainda que a aplicação especifique um retângulo de *clipping* (cerceamento), que limita a atuação das primitivas para uma área restrita de um *canvas*. Ocorre que, durante a execução de um programa gráfico interativo, podem haver mudanças no tamanho de um *canvas*. Nestes casos, como o CD não tem como ser notificado de um evento de mudança de tamanho (*resize*) do *canvas*, a área de *clipping* corrente não é alterada. Isto pode tornar a área de *clipping* inconsistente com o novo tamanho do *canvas*, ou incompatível com o objetivo da aplicação cliente do CD. Como para muitas aplicações a área de *clipping* deve sempre abranger todo o *canvas*, o CD oferece a função **`cdClip`** para ativar e desativar o efeito de *clipping*. Com isso, o efeito indesejado do evento de *resize* fica minimizado e o tratamento de *clipping* nos casos onde ele é necessário deve ser feito explicitamente através de consultas ao tamanho corrente do *canvas* e de *clipping* através das funções **`cdGetCanvasSize`** e **`cdGetClipArea`**.

Um ponto importante a ser destacado é a relação entre as coordenadas do IUP (sistema de interface) e do CD. Uma é obtida da outra a partir da inversão do eixo y. A função **`cdCanvas2Raster`** que converte de coordenadas *raster* do IUP para as coordenadas *canvas* do CD e vice-versa (a equação é a mesma).



Para facilitar aplicações onde os objetos são descritos em valores reais (`double`), foi construída uma pequena camada denominada WD (*World Draw*) sobre o CD. O WD simplesmente mapeia estas coordenadas para as coordenadas em *pixel* (`int`) do *canvas*. Esta transformação se faz através do tradicional par *window-viewport*, como ilustrado na figura a seguir.



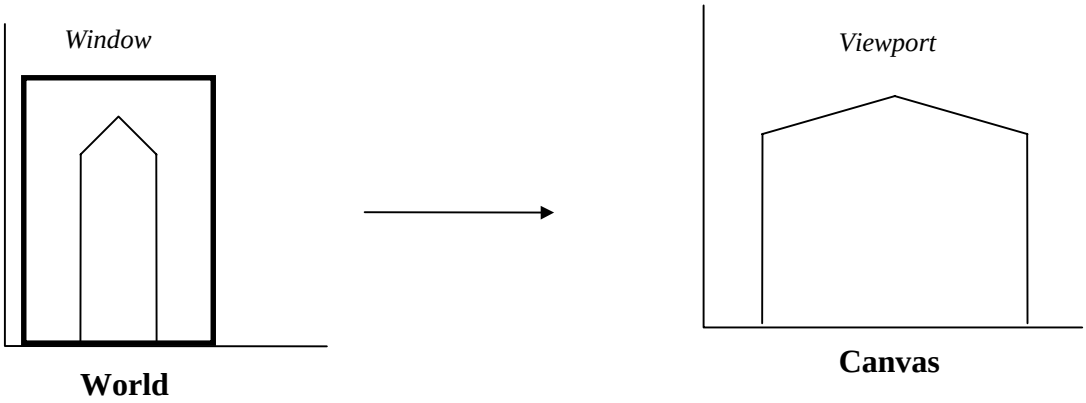
A *window* e a *viewport* corrente são definidas pelas funções **`wdWindow`** e **`wdViewport`**. Os valores *default* da *window* são `(0, width_mm, 0, height_mm)` e os da *`viewport`* `(0, width-1, 0, height-1)` com isto na transformação default as figuras são desenhadas em milímetros na escala 1:1.

Um outro ponto importante a destacar é que, como o uso da transformação *window-viewport* é opcional, estas funções não definem retângulos de *clipping*. No CD, a aplicação deve chamar explicitamente a função **`cdClipArea`** com as coordenadas da *viewport* para limitar as ações da primitivas a este retângulo.

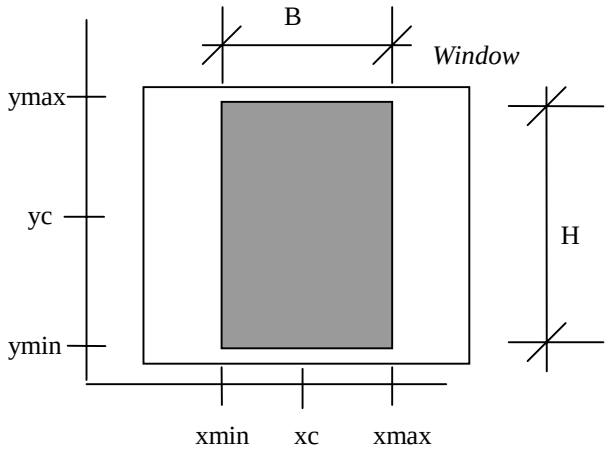
Embora para cada função primitiva de desenho do CD em coordenadas *pixel* exista outra equivalente no WD em coordenadas reais, o CD provê duas funções auxiliares para conversão de coordenadas do mundo para coordenadas do *canvas*: **`wdWorld2Canvas`** e **`wdCanvas2World`**.

No CD os retângulos de *clipping*, *window* e *viewport* são armazenados como atributos do *canvas*, ou seja, quando a aplicação muda de *canvas* (através da função **`cdActivate`**) o CD salva os retângulos do *canvas* antigo e restaura os do novo. Ou seja, cada *canvas* tem seu retângulo de *clipping* e sua transformação *world-canvas* e eles só podem ser alterados quando o respectivo *canvas* estiver ativo.

Quando os retângulos que definem *window* e *viewport* não são proporcionais, ocorre o que se chama de distorção, fenômeno este que pode ser visualizado na figura abaixo, onde a “torre de uma igreja” passou a ser vista como um “galpão”.



Sendo assim, para que tal efeito não ocorra, é necessário que os lados da *window* e *viewport* sejam proporcionais. Uma das formas de mapear a *window* para uma *viewport* **quadrada** sem que haja distorção, pode ser vista a seguir, com o auxílio da figura abaixo.



H/B > 1.0	H/B < 1.0
Ly = H	Lx = B
Lx = Ly	Ly = Lx

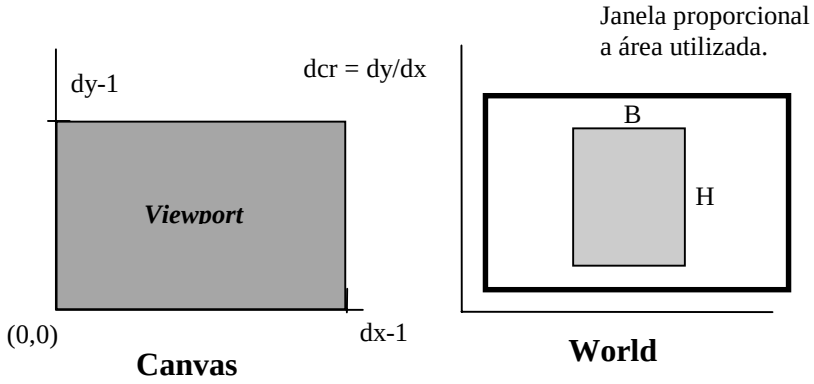
Definição da *window* com 10% de folga:

wxmin = xc - 1.05 *(Lx/2.0) wxmax = xc + 1.05 *(Lx/2.0)
wymin = yc - 1.05 *(Ly/2.0) wymax = yc + 1.05 *(Ly/2.0)

De um modo geral, uma superfície de visualização pode ter qualquer forma, não sendo obrigatoriamente quadrada. Desta forma para que se consiga defnir um par *window-viewport* de modo a não ocorrer a distorção e utilizar toda a superfície de visualização, deve-se proceder da seguinte forma:

1. Determinar as dimensões da superfície de visualização, utilizando a função **`cdGetCanvasSize`**.
2. Calcular a razão do *canvas* cr = dy/dx (altura/largura).
3. Definir a *viewport* do tamanho do *canvas* definido.
4. Definir a *window* para que não haja distorção.

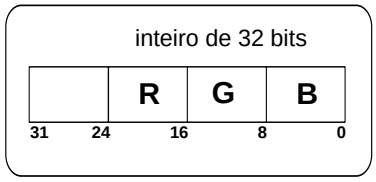
As figuras a seguir ilustram o exposto acima:



Janela proporcional a área utilizada.

H/B > dcr	H/B < dcr
Ly = h	Lx = b
Lx = Ly/dcr	Ly = Lx * dcr

As cores no CD são referenciadas por suas componentes vermelho, verde e azul - RGB. Para fazer referência a uma cor com apenas uma variável simples o CD fornece um par de funções, **`cdEncodeColor`** e **`cdDecodeColor`**, que "empacota" e "desempacota" as componentes RGB em um inteiro de 32 *bits*. Assim, quando uma aplicação deseja utilizar uma cor, ela chama a função **`cdEncodecolor`** passando as componentes RGB, entre 0 e 255, e recebe de volta o valor inteiro que codifica aquela cor. A codificação utilizada no CD é simplesmente: $R*2^{16}+G*2^{8}+B$. A figura ao lado ilustra o armazenamento das componentes RGB no inteiro de 32 *bits.*

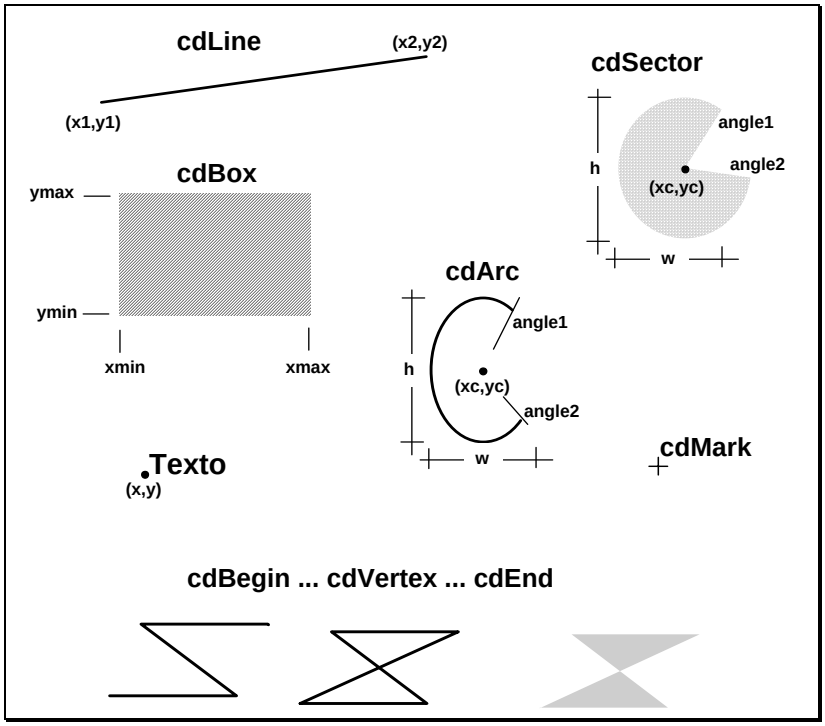


Para facilitar a escolha de cores, o CD possui constantes pré-definidas com a codificação (R,G,B) com nomes do tipo: CD_BLACK, CD_WHITE ou CD_BLUE. O arquivo `cd.h` inclui estas definições.

Quando o dispositivo funciona com palheta de cores o CD procura aproximar a cor especificada com uma cor disponível na palheta do sistema e utiliza esta última em substituição a primeira. Esta aproximação ou é feita simplesmente através da distância entre as componentes RGB, ou é feita nativamente pelo ambiente em que o programa estiver rodando.

Para tornar a limitação de cores de sistemas baseados em palhetas menos severa o CD possui uma função **`cdPalette`** que permite que o programa de aplicação defina um conjunto de cores que gostaria que o CD colocasse na palheta do sistema. É recomendado que aplicações rodando em sistemas com limitações de cores definam as cores que gostariam de utilizar. Para determinar se um sistema possui limitação de cores o CD possui a função **`cdGetColorPlanes`**.

As primitivas de desenho do CD são as mostradas na figura abaixo.



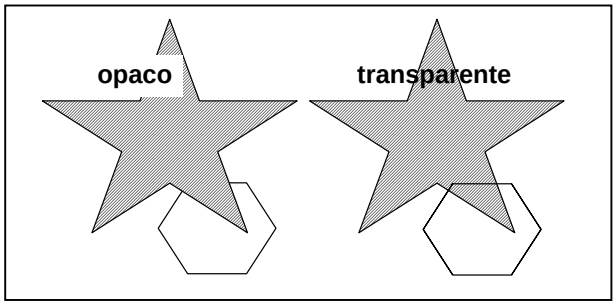
Existem duas funções para cada primitiva, uma CD e outra WD. Na primeira, as coordenadas são definidas em *pixel* com valores inteiros (`int`)**.** Na segunda, as coordenadas são definidas no espaço dos objetos de desenho em reais (`double`). A diferença entre elas é que a segunda converte as coordenadas pela transformação *window-to-viewport* corrente e chama a primeira.

O CD possui um conjunto de atributos com valores correntes que definem a aparência das primitivas de desenho no momento de sua criação. Ou seja, os valores correntes de cor, de largura e estilo de linha, por exemplo, definem a aparência de uma linha no momento da chamada **`cdLine`**. Obviamente, mudanças posteriores não alteram primitivas já desenhadas.

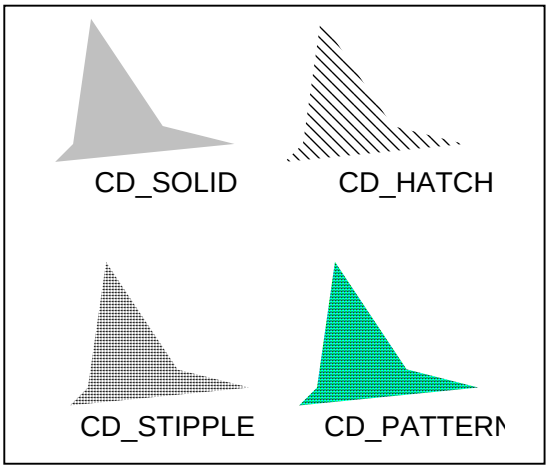
No CD os valores correntes dos atributos são globais e independente da VS. Ou seja, os valores correntes não mudam quando a aplicação muda de *canvas*. Coerentemente, a mudança de um atributo afeta todas as novas primitivas que vierem a ser desenhadas na VS ativa, ou nos próximos VS que vierem a ser ativadas.

Alguns destes atributos afetam todas as primitivas, enquanto outros são específicos de uma dada classe. Os atributos globais dizem respeito as cores e opacidade. Os atributos específicos definem aparência de linhas, marcas, áreas e textos.

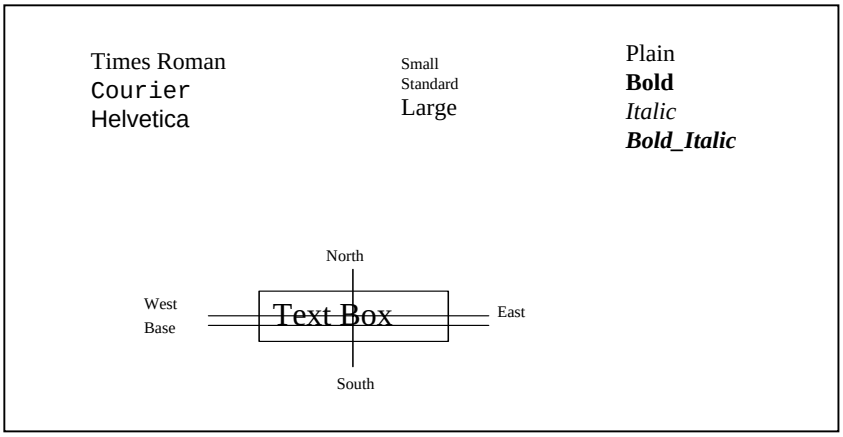
No CD, existem duas cores especiais: a cor de fundo (*background*) e a cor de desenho (*foreground*). A cor do fundo está presente nas primitivas que possuem caixas ou padrões, como textos e áreas preenchidas com hachuras. Quando o atributo de transparência destas primitivas é definido como opaco (CD_OPAQUE), a cor do fundo é atribuida e apaga o que estiver nestes *pixels*; quando o atributo for transparente (CD_TRANSPARENT), o CD não pinta os *pixel* correspondentes ao fundo da primitiva. A figura abaixo ilustra este atributo de opacidade para a primitiva texto e *fill* com hachura.



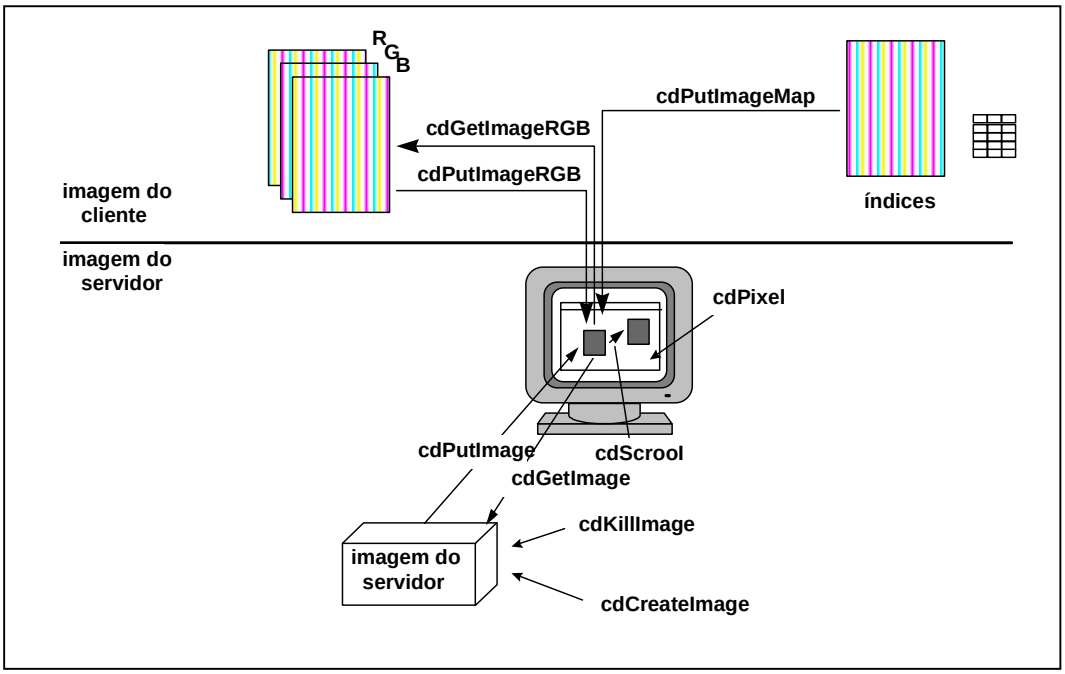
Os atributos de linha são: estilo e espessura; e os atributos de marcas são: tipo e tamanho. Uma área preenchida (**`cdBox`**, **`cdSector`**, ou **`cdBegin...cdVertex...cdEnd`**) pode ter quatro estilos: sólido, hachura, *stipple* e *pattern*. No estilo sólido a cor *foreground* é atribuida a todo interior da primitiva. No estilo hachura o interior é hachurado de acordo com o valor do estilo de hachura corrente (vertical, horizontal, etc..). No estilo *stipple* o interior é preenchido com as cores de *foreground* e *background* de acordo com uma matriz de 0's e 1's. Zero representa o *background* e um o *foreground*. Finalmente, no estilo *pattern* o interior é preenchido com um padrão de cores definidas por uma matriz de cores no formato do CD. A figura a seguir ilustra estes atributos de áreas.



Os atributos que afetam a aparência das primitivas tipo texto são: fonte, tamanho, estilo e alinhamento.



No CD existem dois tipos de imagens: do cliente e do servidor. A primeira tem um formato simples para ser fácil de ser modificada pelo programa de aplicação e a segunda tem o formato do equipamento que é eficiente para tirar e colocar na tela. O CD provê também funções para converter de um formato para outro. A figura abaixo ilustra esta estratégia.



Nesta figura estão também ilustradas as funções de imagens do CD. Para simplesmente mover ou tirar e colocar retângulos de pixels do *canvas* o CD prove as funções **`cdScroll`**, **`cdGetImage`** e **`cdPutImage`**. A função **`cdScroll`** simplesmente move de um determinado deslocamento. As funções **`cdGetImage`** e **`cdPutImage`** copiam o conteúdo da tela para uma imagem de servidor. Esta imagem é simplesmente uma estrutura em C num formato não transparente (é o tipo de dado **`cdImage`**) que serve apenas para armazenar temporariamente esta informação durante a execução do programa. A alocação de memória para esta estrutura é feita pelo **`cdCreateImage`** e a liberação pela função **`cdKillImage`**.

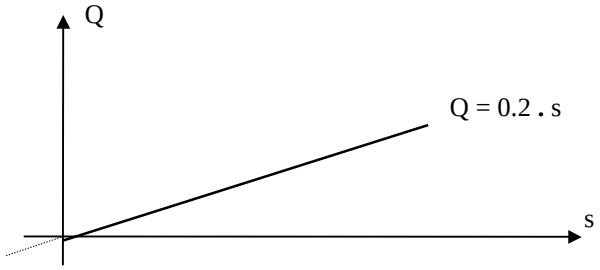
Para mudar valores de *pixels* de uma imagem o CD supõe que primeiramente a aplicação transforme esta imagem do servidor em um dos dois formatos de imagem de cliente: RGB ou *map*. No formato RGB a imagem é transformada em três matrizes de *bytes*. No formato *map* a imagem é transformada em uma matriz de índices e num um vetor de cores relacionado a este mapa. Ou seja, a cor de um determinado *pixel* é fornecida pela componente do vetor apontada pelo índice da matriz. O CD provê então quatro funções para estas tarefas: **`cdEncodeImage`**, **`cdDecodeImage`** e **`cdEncodeMap`**.

Obviamente, uma aplicação pode também criar uma imagem de cliente, convertê-la e colocá-la na tela. É importante notar que em qualquer procedimento a aplicação é sempre responsável por alocar e liberar espaço de memória para as imagens do cliente e do servidor. Ela é também responsável por garantir que a imagem do cliente e do servidor tenham o mesmo número de linhas e colunas.

Capítulo 6 - Transformações Geométricas

Funções lineares descrevem o tipo mais simples de dependência entre variáveis. Por exemplo, se de 1.0 kg de soja são extraídos 0.2 l de óleo, de uma produção de **x** kg de soja, seriam extraídos 0.2**x** l de óleo. Pode-se escrever o que foi dito acima através de uma função que seria:

Q(s) = 0.2 **.** s, onde: Q = quantidade em litros de óleo de soja
s = quantidade em kilogramas de soja



Propriedades das funções lineares:
1. $Q(s_1 + s_2) = Q(s_1) + Q(s_2)$
2. $Q(\mathbf{k} \cdot s) = \mathbf{k} \cdot Q(s)$

Analogamente, ao que foi desenvolvido para as funções lineares, pode-se dizer que, matematicamente, as transformações nos espaços vetoriais cartesianos são relações do $\mathbf{R}^n$ no $\mathbf{R}^m$ e podem ser escritas como:

$T : \mathbf{R}^n \rightarrow \mathbf{R}^m$
$\{ P \} \rightarrow \{ P' \} = T \{ P \}$

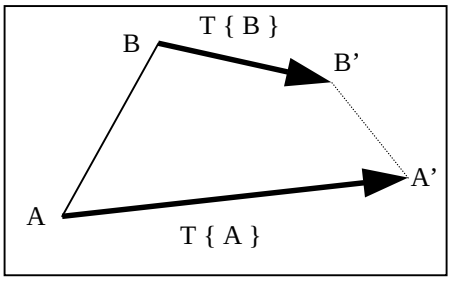
Uma transformação é dita linear **se e somente se**:

$$T \{ \alpha \{ P \} + \beta \{ Q \} \} = \alpha T \{ P \} + \beta T \{ Q \}$$
$$\forall \alpha, \beta \in \mathbf{R} \text{ e } \{ P \}, \{ Q \} \in \mathbf{R}^m$$

Uma primeira propriedade que pode-se notar é que o vetor nulo de um espaço vetorial **V**, leva ao vetor nulo de outro espaço vetorial, ou seja: $T \{ 0 \} = \{ 0 \}$. Esta propriedade permite identificar uma transformação que não seja linear, entretanto, não é suficiente para provar que uma transformação é linear.

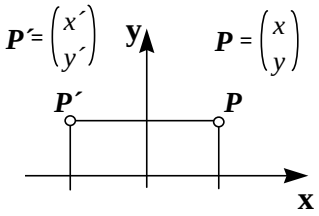
A segunda propriedade das transformações lineares diz que estas podem ser expressas matricialmente, da forma: $\{ P' \} = [M] \{ P \}$, onde diz-se que $[M]$ é a matriz de transformação.

A terceira propriedade importante das transformações lineares, sob o aspecto de eficiência computacional, é que a imagem de uma transformação linear genérica aplicada sobre um segmento de reta, também é um segmento de reta. Isto permite transformar apenas os pontos extremos dos segmentos e depois ligá-los para obter o resultado final da transformação



Como exemplos de transformações lineares, serão analisados algumas transformações geométricas corriqueiras na área de computação gráfica, como será exposto a seguir:

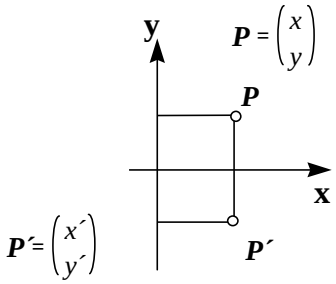
- Espelhamento em relação ao eixo “Y”



$$x' = -x \qquad y' = y$$

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

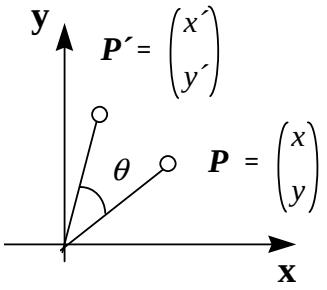
- Espelhamento em relação ao eixo “X”



$$x' = x \qquad y' = -y$$

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

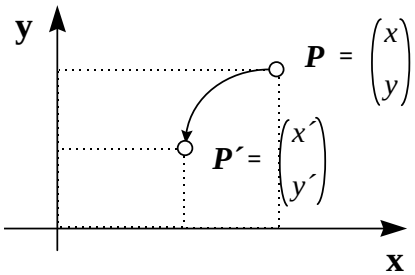
- Rotação em relação a origem



$$x' = x.\cos\theta - y.\sin\theta$$
$$y' = x.\sin\theta + y.\cos\theta$$

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

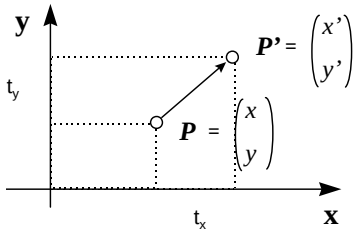
- Escala em relação a origem



$$x' = S_x \cdot x \qquad y' = S_y \cdot y$$

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

- Translação



$$x' = x + t_x \qquad y' = y + t_y$$

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix}$$

Analisando-se com cuidado a transformação geométrica de translação nota-se que esta não é uma transformação linear, pois a primeira propriedade (T { 0 } = { 0 }) não é satisfeita. Sendo assim, diz-se que a translação é uma **transformação linear afim**.

Duas transformações podem ser combinadas, ou concatenadas, de modo a produzir uma única transformação com o mesmo efeito da aplicação seqüencial das duas primeiras. O conceito de concatenação de transformações é de fundamental importância sob o aspecto computacional, pois permite armazenar informações relativas a **n** transformações distintas em uma única matriz de transformação, reduzindo sobremaneira a memória requerida e aumentando a eficiência do processo.

Como já foi visto, existem dois tipos de transformações as lineares e as lineares afins. Desta forma, a concatenação para estes tipos produzem o seguinte resultado:

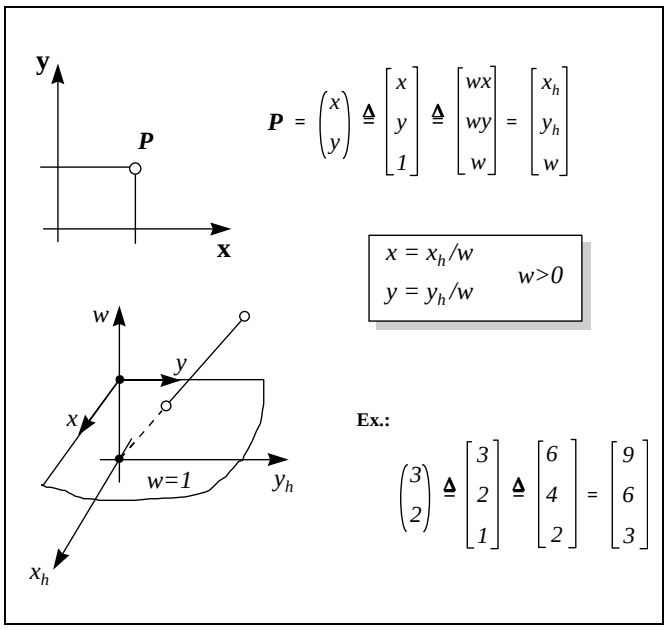
- Transformações Lineares

$$\{ P_1 \} = [M_1] \{ P_0 \}$$
$$\{ P_2 \} = [M_2] \{ P_1 \} \qquad \{ P_2 \} = [M_2] [M_1] \{ P_0 \}$$
$$\vdots$$
$$\{ P_n \} = [M_n] \{ P_{n-1} \}$$
$$[M_{final}] = [M_n] [M_{n-1}] [M_{n-2}] \dots [M_2] [M_1]$$

- Transformações Lineares Afins

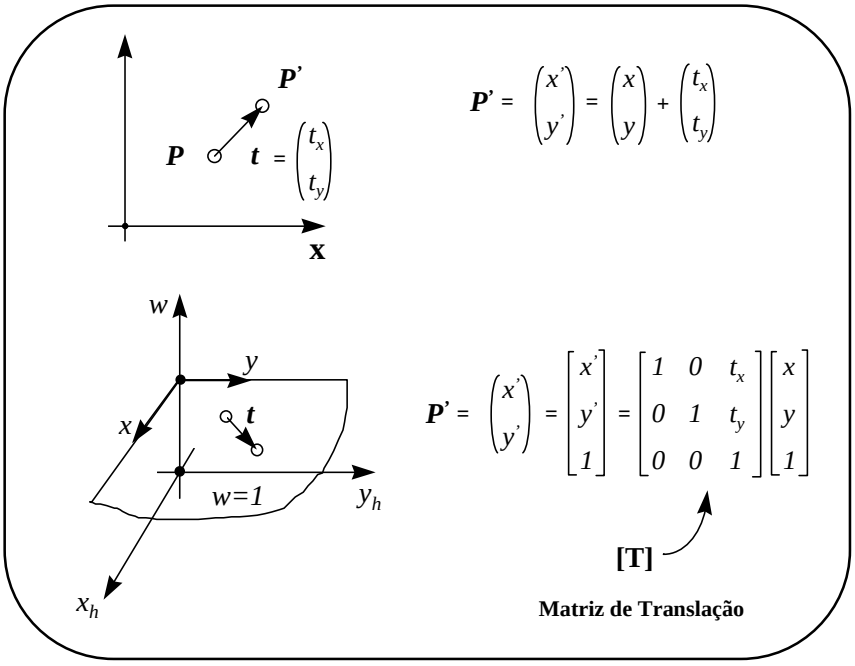
$$\{ P_1 \} = [M_1] \{ P_0 \} + \{ t_1 \}$$
$$\{ P_2 \} = [M_2] \{ P_1 \} + \{ t_2 \} \qquad \{ P_2 \} = [M_2]([M_1] \{ P_0 \} + \{ t_1 \}) + \{ t_2 \}$$
$$\vdots$$
$$\{ P_n \} = [M_n] \{ P_{n-1} \} + \{ t_n \}$$
$$\{ P_n \} = [M_n][M_{n-1}]\dots[M_1] \{P_0\} + [M_n][M_{n-1}]\dots[M_2] \{ t_1 \} + \dots$$
$$[M_n] \{ t_{n-1} \} + \{ t_n \}$$

Nota-se que as transformações lineares possuem uma representação matricial mais simples que as transformações lineares afins, no que se refere a concatenação. Para que se tenha um ganho na eficiência computacional que abrange o processo de transformações, existem artifícios que permitem tratar transformações lineares afins da mesma forma que as transformações lineares, como será visto a seguir, com o estudo das coordenadas homogêneas.



A representação em coordenadas homogêneas de um ponto do espaço $\mathbf{R}^n$, são pontos pertencentes ao espaço $\mathbf{R}^{n+1}$, ou seja, o ponto (x,y) do $\mathbf{R}^2$, em coordenadas homogêneas, é representado por (x,y,1), como é mostrado na figura ao lado. Nota-se que existem infinitas representações para um único ponto dependendo do fator de escala **w**.

Analisando-se a transformação geométrica de translação sob o ponto de vista de coordenadas homogêneas, tem-se o seguinte:



Sendo assim, os pontos (x,y) $\in \mathbf{R}^2$ podem ser identificados como pontos (x,y,1) $\in \mathbf{R}^3$ e translações no $\mathbf{R}^2$ podem ser tratadas como transformações lineares no $\mathbf{R}^3$.

Resumindo-se as prinicipais transformações geométricas 2D em coordenadas homogêneas, tem-se:

Espelhamento (rel. Y)
$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

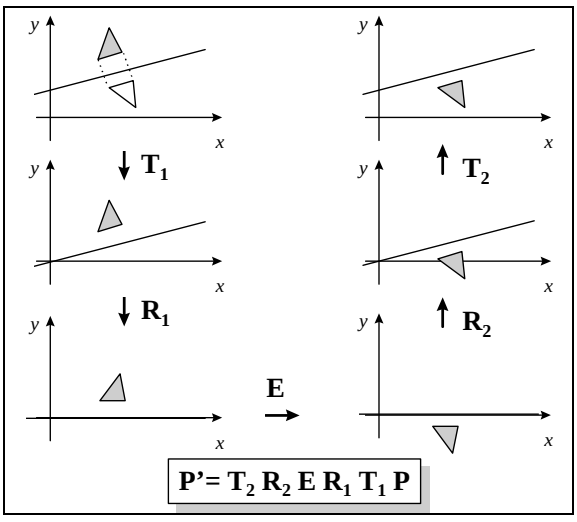
Espelhamento (rel. X)
$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Escala (rel. a origem)
$$\begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Translação
$$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

Rotação (rel. a origem)
$$\begin{bmatrix} \cos\theta & -\text{sen}\,\theta & 0 \\ \text{sen}\,\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

A concatenação de transformações pode ser feita agora de forma simples usando-se então transformações lineares, fazendo-se a multiplicação das matrizes. Entretanto, deve-se somente lembrar-se que a seqüência que que o produto das matrizes é realizado, corresponde a **ordem inversa** que as transformações ocorrem.



Manual de Referência do CD

Tecgraf - Grupo de Tecnologia em Computação Gráfica

Junho, 1996

Funções de Controle

cdCanvas *cdCreateCanvas (cdContext driver, void *data)

Cria um *canvas* CD para uma superfície de visualização virtual, VVS (*Virtual Visualization Surface*). Uma VVS pode ser um *canvas* do IUP, a página de um documento enviado para a impressora, uma imagem *off-screen*, a área de transferência ou um *metafile*, dentre outros. O controlador é definido pela variável **driver** com as informações adicionais dadas no parâmetro **data**. Estão disponíveis os seguintes controladores:

- **CD_IUP** = Canvas do IUP, **data** é o **Ihandle** do canvas. Incluir o arq. cdiup.h.
- **CD_PRINTER** = Impressora, **data** é um ***char** para o nome do documento. Funciona somente no ambiente MS Windows. Incluir o arq. cdprint.h.
- **CD_IMAGE** = Desenho *off-screen*, **data** é um ponteiro para uma **cdImage**. Incluir o arq. cdimage.h.
- **CD_CLIPBOARD** = Área de transferência, **data** é um ***char** para a *string* "***width*x*height [-b]***", onde -b (opcional) indica que um *bitmap* será criado ao invés de um *metafile*. Funciona somente no ambiente MS Windows. Incluir o arq. cdclipbd.h.
- **CD_WMF** = *Metafile* do Windows, **data** é um ***char** para a *string* "***filename width*x*height***". Funciona somente no ambiente MS Windows. Incluir o arq. cdwmf.h.
- **CD_CGM** = ISO *Computer Graphics Metafile*, **data** é um ***char** para a *string* "***filename width*x*height***". Incluir o arq. cdcgm.h .
- **CD_METAFILE** = *Metafile* do CD (funções do CD em um arquivo), **data** é um ***char** para a *string* "***filename width*x*height***". Incluir o arq. cdcgm.h.

void cdKillCanvas (cdCanvas *canvas)

Destrói um *canvas* CD previamente criado.

int cdActivate (cdCanvas *canvas)

Ativa um *canvas* CD para desenhar. Não há um **cdDeactivate** explícito. Quando uma nova superfície de visualização é ativada a corrente é desativada (há apenas uma VVS ativa de cada vez). A função retorna um *status*, **CD_OK** ou **CD_ERROR**, indicando se o *canvas* alvo foi ativado com sucesso ou não.

void cdFlush(void)

Tem um significado diferente para cada *driver*:

- Impressora: avança a página.
- Canvas do IUP no MS Windows / *Off-Screen* / *Metafile* : não faz nada.

void cdClear(void)

Limpa o *canvas* CD usando a cor corrente do *background*.

Sistemas de Coordenadas e *Clipping*

`void cdGetCanvasSize(int *width, int *height, double *mm_width, double *mm_height)`
Retorna o tamanho do *canvas* CD em *pixels* e em milímetros. Assim como em qualquer função do CD, você pode ignorar um parâmetro de retorno passando seu ponteiro como **`NULL`**. Por exemplo, você pode chamar **`cdGetCanvasSize(&w, &h, NULL, NULL)`** para obter apenas os valores em *pixels*.

`int cdClip(int mode)`
Ativa ou desativa o *clipping*. Retorna o *status* anterior. Valores: **`CD_CLIPON`** ou **`CD_CLIPOFF`**.

`void cdClipArea(int xmin, int xmax, int ymin, int ymax)`
Define o retângulo de *clipping*.

`void cdCanvas2Raster(int *x, int *y)`
Converte as coordenadas **`(x,y)`** do sistema canvas para o sistema *raster* ou vice-versa. As coordenadas *raster* são usadas em IUP e em outros sistemas de janelas e têm sua origem no canto superior esquerdo do *canvas* CD com o eixo Y orientado para baixo. O sistema do canvas tem sua origem no canto inferior esquerdo da VS com o eixo Y orientado para cima. Ambos os sistemas são em *pixels*. O valor de X continua o mesmo. É usado apenas por razões estéticas. NOTA: Todas as primitivas, incluindo as imagens, são definidas em CD no sistema de coordenadas do canvas.

Primitivas

`void cdLine(int x1, int y1, int x2, int y2)`
Desenha uma linha de **`(x1,y1)`** a **`(x2,y2)`** utilizando a cor do *foreground* corrente. Ambas as coordenadas são dadas no sistema do canvas.

`void cdArc(int xc, int yc, int w, int h, double angle1, double angle2)`
Desenha um arco de elipse alinhado com o eixo utilizando a cor do *foreground* corrente. **`(xc,yc)`** é o centro da elipse. **`w`** e **`h`** são os eixos elípticos X e Y, respectivamente. **`angle1`** e **`angle2`** definem o início e o fim do arco. O arco
começa no ponto **`(xc+(w/2)*cos(angle1), yc+(h/2)*sin(angle1))`** e termina no ponto **`(xc+(w/2)*cos(angle2), yc+(h/2)*sin(angle2))`**. As coordenadas são dadas no sistema do canvas e os ângulos em graus.

`void cdMark(int x, int y)`
Desenha uma marca em **`(x,y)`** utilizando a cor do *foreground* corrente. As coordenadas são dadas no sistema do canvas.

`void cdBox(int xmin, int xmax, int ymin, int ymax)`
Preenche um retângulo de acordo com o estilo de preenchimento corrente. As coordenadas são dadas no sistema do canvas.

`void cdSector(int xc, int yc, int w, int h, double angle1, double angle2)`
Preenche um arco de elipse alinhado com o eixo de acordo com o estilo de preenchimento corrente. **`(xc,yc)`** é o centro da elipse. **`w`** e **`h`** são os eixos elípticos X e Y, respectivamente. **`angle1`** e **`angle2`** definem o início e o fim do arco. O arco
começa do ponto **`(xc+(w/2)abanicos(angle1), yc+(h/2)*sin(angle1))`** e termina no ponto **`(xc+(w/2)*cos(angle2), yc+(h/2)*sin(angle2))`**. As coordenadas são dadas no sistema do canvas e os ângulos são dados em graus.

`void cdText(int x, int y, char *text)`
Coloca um texto em **`(x,y)`** de acordo com a opacidade corrente. As coordenadas são dadas no sistema do canvas.

`void cdBegin(int mode)`
Inicia a definição de um polígono que deve ser desenhado (ou preenchido) de acordo com **`mode`**: **`CD_CLOSED_LINES`**, **`CD_OPEN_LINES`** ou **`CD_FILL`**.

`void cdVertex(int x, int y)`
Adiciona um vértice à definição do polígono. As coordenadas são dadas no sistema do canvas.

`void cdEnd(void)`
Termina a definição do polígono e o desenha (ou preenche).

Atributos

`int cdBackOpacity(int opacity)`
Define a opacidade do fundo para todas as primitivas de preenchimento baseadas em cores de fundo-frente como *stipple* ou *text*. Se o fundo é opaco, a primitiva de texto, por exemplo, apaga qualquer coisa que esteja em sua caixa envolvente. Se for transparente, apenas as cores de frente são pintadas. O mesmo se aplica a um *hatch* de preenchimento. Valores: **`CD_TRANSPARENT`**, **`CD_OPAQUE`**. Retorna o valor corrente anterior. Valor inicial: **`CD_TRANSPARENT`**.

`int cdWriteMode(int mode)`
Define o **`mode`** de desenho para as primitivas (linhas, áreas, marcas e texto). Valores: **`CD_REPLACE`**, **`CD_XOR`**, **`CD_NOT_XOR`**. Valor inicial: **`CD_REPLACE`**.

`int cdLineStyle(int style)`
Seleciona o estilo de linha corrente como: **`CD_CONTINUOUS`**, **`CD_DASHED`**, **`CD_DOTTED`**, **`CD_DASH_DOT`** ou **`CD_DASH_DOT_DOT`**. Retorna o valor corrente anterior. Valor inicial: **`CD_CONTINUOUS`**.

`int cdLineWidth(int width)`
Define a espesura da linha (em *pixels*). Retorna a espessura anterior. Valor inicial: **`1`**.

`int cdMarkerType(int type)`
Seleciona o tipo de marca corrente: **`CD_PLUS`**, **`CD_STAR`**, **`CD_CIRCLE`**, **`CD_X`**, **`CD_BOX`**, **`CD_DIAMOND`**, **`CD_HOLLOW_CIRCLE`**, **`CD_HOLLOW_BOX`**, **`CD_HOLLOW_DIAMOND`**. Retorna o tipo anterior. Valor inicial: **`CD_STAR`**.

`int cdMarkerSize(int size)`
Determina o tamanho da marca em *pixels*. Retorna o tamanho anterior. Valor inicial: **`10`**.

`int cdInteriorStyle(int style)`
Seleciona o estilo corrente para primitivas de preenchimento de área: **`CD_SOLID`**, **`CD_HATCH`**, **`CD_STIPPLE`** ou **`CD_PATTERN`**. Valor inicial: **`CD_SOLID`**.

`int cdHatch(int style)`
Seleciona um estilo predefinido de hatch (**`CD_HORIZONTAL`**, **`CD_VERTICAL`**, **`CD_FDIAGONAL`**, **`CD_BDIAGONAL`**, **`CD_CROSS`**, ou **`CD_DIAGCROSS`**) e seleciona o estilo de interior como **`CD_HATCH`**.

`void cdStipple(int w, int h, char *fgbg)`
Define uma matriz **`w`**x**`h`** de zeros (fundo) ou uns (frente) armazenada em **`fgbg`** e seleciona o estilo de interior como **`CD_STIPPLE`**. Para evitar a manipulação de matrizes em C, o elemento **`(i,j)`** de **`fgbg`** é definido como **`fgbg[j*w+i]`**.

`void cdPattern(int w, int h, long int *color)`
Define uma matriz **`w`**x**`h`** de cores e seleciona o estilo de interior como **`CD_PATTERN`**. Para evitar a manipulação de matrizes em C, o elemento **`(i,j)`** de **`color`** é definido como **`color[j*w+i]`**.

`void cdFont(int typeface, int style, int size)`
Seleciona uma fonte de texto. O parâmetro **`typeface`** pode ter os valores: **`CD_SYSTEM`**, **`CD_COURIER`**, **`CD_TIMES_ROMAN`**, ou **`CD_HELVETICA`**. **`style`** pode ser: **`CD_PLAIN`**, **`CD_BOLD`**, **`CD_ITALIC`** ou **`CD_BOLD_ITALIC`**. **`size`** é dado em pontos (1/72 polegada) e para tornar a escolha mais fácil o CD oferece três constantes: **`CD_SMALL=10`**, **`CD_STANDARD=12`** e **`CD_LARGE=18`**. Os valores iniciais são: **`CD_SYSTEM`**, **`CD_PLAIN`** e **`CD_STANDARD`**.

`void cdFontDim(int *maxwidth, int *height, int *ascent, int *descent)`
Retorna a largura máxima, a altura, o *ascent* e o *descent* dos caracteres da fonte atualmente selecionada. A altura da linha é a soma de *ascent* e *descent* mais algum espaço entre linhas (se esse for o caso). Todos os valores são dados em *pixels*.

`void cdTextSize(char *text, int *width, int *height)`
Retorna a altura e a largura tamanho da caixa envolvente de um texto na fonte atual.

`int cdTextAlignment(int alignment)`
Define o alinhamento horizontal e vertical do texto como: **`CD_NORTH`**, **`CD_SOUTH`**, **`CD_EAST`**, **`CD_WEST`**, **`CD_NORTH_EAST`**, **`CD_NORTH_WEST`**, **`CD_SOUTH_EAST`**, **`CD_SOUTH_WEST`**, **`CD_CENTER`**, **`CD_BASE_LEFT`**, **`CD_BASE_CENTER`**, **`CD_BASE_RIGHT`**. Retorna o alinhamento anterior. Valor inicial: **`CD_BASE_LEFT`**.

Cores

`long int cdEncodeColor(unsigned char red, unsigned char green, unsigned char blue)`

Retorna um trio (r, g, b) codificado como **`0x00RRGGBB`**. Onde **`RR`** ;é a componente vermelha, **`GG`** a verde e **`BB`** a azul. Esse código é usado pelo CD para definir cores. Existem algumas cores pré-definidas no arquivo cd.h: `CD_RED, CD_GREEN, CD_BLUE, CD_DARK_RED, CD_DARK_GREEN, CD_DARK_BLUE, CD_YELLOW, CD_MAGENTA, CD_CYAN, CD_DARK_YELLOW, CD_DARK_MAGENTA, CD_DARK_CYAN, CD_WHITE, CD_BLACK, CD_DARK_GRAY, CD_GRAY.`

`void cdDecodeColor(long int color, unsigned char *red, unsigned char *green, unsigned char *blue)`

Fornece as componentes vermelha, verde e azul de uma cor do CD.

`long int cdForeground(long int color)`

Seleciona uma nova cor para a frente e retorna a antiga. Essa cor é usada em todas as primitivas (linhas, áreas, marcas e textos). Valor inicial : `CD_BLACK.`

`long int cdBackground(long int color)`

Seleciona uma nova cor para o fundo e retorna a antiga. Isso, entretanto, não muda automaticamente a cor de fundo do canvas. Para isso é preciso chamar **`cdClear`**. Valor inicial : `CD_WHITE.`

`int cdGetColorPlanes(void)`

Retorna um número, digamos *p*, que define o número de cores suportadas pelo dispositivo corrente como 2^p, representando, o número de *bits* por *pixel*.

`void cdPalette(int n, long int *color, int mode)`

Em sistemas limitados a palhetas de 256 cores essa função procura colocar **`n`** cores na palheta do sistema. Nesses sistemas, cores escolhidas para frente ou fundo e que não se encontram na palheta são aproximadas pelas cores mais próximas disponíveis. **`mode`** pode ser `CD_FORCE` ou `CD_POLITE`. O primeiro modo ignorará as cores definidas pelo sistema e pelos elementos de interface, como *menus* e diálogos, podendo torná-los ilegíveis, mas garantindo o uso de mais cores; entretanto, é recomendado utilizar o segundo modo.

Funções de Imagem de Cliente

`void cdGetImageRGB(unsigned char *r, unsigned char *g, unsigned char *b,`
`int x, int y, int w, int h)`

Retorna as componentes RGB (vermelha, verde e azul) de cada *pixel* de uma imagem de servidor. As componentes RGB são dadas em três matrizes armazenadas em vetores de *bytes*. Os componentes **`(i,j)`** dessas matrizes são encontrados nos endereços **`(j*w+i)`**. Assim como em todas as primitivas do Canvas Draw, o *pixel* **`(0,0)`** está no canto inferior esquerdo e **`(w-1,h-1)`** é o canto superior direito do retângulo da imagem.

`void cdPutImageRGB(int iw, int ih, unsigned char *r, unsigned char *g,`
`unsigned char *b, int x, int y, int w, int h)`

Coloca em uma área específica do *canvas* uma imagem de servidor que tem as componentes RGB definidas nas três matrizes. Essas matrizes são armazenadas em vetores de *bytes*. Os componentes **`(i,j)`** dessas matrizes são encontrados nos endereços **`(j*w+i)`**. Assim como em todas as primitivas do Canvas Draw, o *pixel* **`(0,0)`** está no canto inferior esquerdo e **`(w-1,h-1)`** é o canto superior direito do retângulo da imagem.

`void cdPutImageMap(int iw, int ih, unsigned char *map, long int *palette,`
`int x, int y, int w, int h)`

Função análoga a **`cdPutImageRGB`** a não ser pelas cores, que são dadas em uma matriz de índices (**`map`**). As cores correspondentes aos índices são dadas em **`palette[índice]`**. **`map`** é também uma matriz armazenada como um vetor de *bytes*.

Funções de Imagem de Servidor

`void cdPixel(int x, int y, long int cd_color)`
Acende o *pixel* **`(x,y)`** com a cor **`cd_color`**.

`void cdCreateImage(int w, int h)`
Cria uma imagem compatível na memória do servidor com **`w`**x**`h`** *pixels*. A imagem compatível tem a mesma representação de cores (número de bits por *pixel*) que a imagem do servidor.

`void cdGetImage(void *image, int x, int w)`
Copia uma região retangular da tela para a memória (**`image`**). **`(x,y)`** são as coordenadas do canto inferior esquerdo da região retangular. As dimensões da imagem são definidas na estrutura da imagem (são definidas quando a imagem é criada).

`void cdPutImage(void *image, int x, int w)`
Copia uma imagem em uma região retangular do canvas com canto inferior esquerdo em **`(x,y)`**. É o inverso da função **`cdGetImage`**.

`void cdKillImage(void *image)`
Libera a memória alocada para a imagem.

`void cdScrollImage(int xmin, int xmax, int ymin, int ymax, int dx, int dy)`
Copia o retângulo definido pelas coordenadas **`(xmin,ymin)`** e **`(xmax,ymax)`** para o retângulo definido por **`(xmin+dx,ymin+dy)`** e **`(xmax+dx,ymax+dy)`**.

Funções em coordenadas do mundo

`int wdActivate (cdCanvas *canvas)`
Ativa o CD e a camada WD responsável pela funções de coordenadas do mundo. Esta função deve ser chamada ao invés da `cdActivate` se a camada WD for utilizada. A função retorna um *status*, **`CD_OK`** ou **`CD_ERROR`**, indicando se o *canvas* alvo foi ativado com sucesso ou não.

`void wdWindow(double xmin, double xmax, double ymin, double ymax)`
Define uma janela no sistema de coordenadas do mundo para uso na transformação de coordenadas do mundo (**`double`**) em coordenadas do canvas (**`int`**). A janela inicial é uma aproximação em milímetros de todo o *canvas*.

`void wdViewport(int xmin, int xmax, int ymin, int ymax)`
Define um *viewport* no sistema de coordenadas do canvas para uso na transformação de coordenadas do mundo (**`double`**) em coordenadas do canvas (**`int`**).

`void wdWorld2Canvas(double xw, double yw, int *xv, int *yv)`
Converte coordenadas do mundo em coordenadas do canvas.

`void wdCanvas2World(int xv, int yv, double *xw, double *yw)`
Converte coordenadas do canvas em coordenadas do mundo.

`void wdPixel2MM(int dx, int dy, double *mm_dx, double *mm_dy)`
Converte tamanhos em *pixels* (coordenadas do canvas) para tamanhos em milímetros.

`void wdMM2Pixel(double mm_dx, double mm_dy, int *dx, int *dy)`
Converte tamanhos em milímetros para tamanhos nas coordenadas do canvas (*pixels*).

`void wdGetWindow(double *xmin, double *xmax, double *ymin, double *ymax)`
Retorna a janela atual em coordenadas do mundo que está sendo usada na transformação de coordenadas do mundo em coordenadas do canvas (e vice-versa).

`void wdGetViewport(int *xmin, int *xmax, int *ymin, int*ymax)`
Retorna o *viewport* atual que está sendo usado na transformação de coordenadas do mundo em coordenadas do canvas (e vice-versa).

`void wdLine(double x1, double y1, double x2, double y2)`
Desenha uma linha de **`(x1,y1)`** a **`(x2,y2)`**. Ambas as coordenadas são dadas no sistema de coordenadas do mundo.

`void wdArc(double xc, double yc, double w, double h, double angle1, double angle2)`
Desenha um arco de elipse alinhado com o eixo. **`(xc,yc)`** é o centro da elipse. **`w`** e **`h`** são os eixos elípticos X e Y, respectivamente. **`angle1`** e **`angle2`** definem o início e o fim do arco. O arco começa no ponto **`(xc+(w/2)*cos(angle1), yc+(h/2)*sin(angle1))`** e termina no ponto **`(xc+(w/2)*cos(angle2), yc+(h/2)*sin(angle2))`**. As coordenadas são dadas no sistema de coordenadas do mundo e os ângulos em graus.

`void wdMark(double x, double y)`
Desenha uma marca em **`(x,y)`**. As coordenadas são dadas no sistema do mundo.

`void wdBox(double xmin, double xmax, double ymin, double ymax)`
Preenche um retângulo. As coordenadas são dadas no sistema de coordenadas do mundo canvas.

`void wdSector(double xc, double yc, double w, double h, double angle1, double angle2)`
Preenche um arco de elipse alinhado com o eixo. **`(xc,yc)`** é o centro da elipse. **`w`** e **`h`** são os eixos elípticos X e Y, respectivamente. **`angle1`** e **`angle2`** definem o início e o fim do arco. O arco começa do ponto **`(xc+(w/2)*cos(angle1), yc+(h/2)*sin(angle1))`** e termina no ponto **`(xc+(w/2)*cos(angle2), yc+(h/2)*sin(angle2))`**. As coordenadas são dadas no sistema de coordenadas do mundo e os ângulos são dados em graus.

`void wdText(double x, double y, char *text)`
Coloca um texto em **`(x,y)`**. As coordenadas são dadas no sistema do mundo.

`void wdVertex(double x, double y)`
Adiciona um vértice à definição do polígono. As coordenadas são dadas no sistema de coordenadas do mundo.

Funções de Texto Vetorial

`void wdVectorTextDirection(double x1, double y1, double x2, double y2)`
Define a direção do texto como a da reta definida por **`(x1,y1)`** e **`(x2,y2)`**.

`void wdVectorTextSize(const double size_x, const double size_y, const char *string)`
Muda a escala do texto vetorial para que **`string`** fique contido dentro da caixa envolvente definida por **`size_x`** e **`size_y`**.

`void wdGetVectorTextSize(const char *string, double *size_x, double size_y)`
Retorna a caixa envolvente do texto.

`void wdVectorCharSize(double size)`
Define o tamanho dos caracteres (altura e largura máxima).

`void wdVectorText(double x, double y, const char *s)`
Desenha uma linha de texto vetorial referente às coordenadas **`(x,y)`** do mundo. Respeita o alinhamento definido por **`cdTextAlignment`**. Ignora a opacidade.

`void wdMultiLineVectorText(double x, double y, const char *s)`
Desenha várias linhas de texto vetorial referentes às coordenadas **`(x,y)`** do mundo. Respeita o alinhamento definido por **`cdTextAlignment`**. Ignora a opacidade.