Nome: André Luiz N. Carneiro De Castro Ra: 92854

1) No **algoritmo do pintor**, utilizasse de técnicas de pinturas das quais pintamos primeiros os elementos que estão na primeira “camada” da nossa figura, isso dá a impressão de profundidade de nossa imagem, sempre utilizando o Z medio da nossa figura.
Com essa restrição de pintar coisas estritamente de trás para frente ou (para algoritmo de pintor reverso) da frente para trás. Se pintar de trás para a frente; Tudo o que pintar por último ficará completamente visível. Este algoritmo tem problemas quando um grupo de objetos não podem ser alinhado de trás para a frente.
Devemos lembrar que esse algoritmo possui problemas com o processamento e intersecção de polígonos com isso ele pode ser usado mais para renderização, já que não importa tanto o tempo.

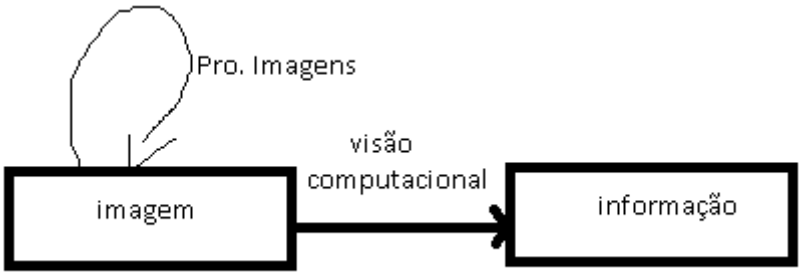
O **Zbuffer** é o mais utilizado, inclusive em muitas placas de vídeos existem muitas otimizações para ele, e ele tem um desempenho de processamento muito melhor em relação ao outro algorítmo. Ele tem como característica duas matrizes, uma pra cor e uma pra profundidade. Na hora de realizar o procedimento em que serão colocados os elementos, e até mesmo na hora de pintá-los, temos como análise verificar o objeto se está mais a frente ou mais atrás que o outro analisado no momento, com isso sempre que for preencher um pixel, deve ser analisado qual está mais na frente. Fazendo isso pixel a pixel teremos o resultado desejado e teremos uma perfeição em relação a representação de profundidade.
Portanto o Z-Buffer é diferente do algoritmo do pintor, porque as informações de profundidade são mantidas. Você pode pintar algo de perto; e depois podemos pintar algo que está atrás dele. Uma vez que a “tela” lembra o quão perto está a primeira coisa, o “close up” não será coberto por nada que esteja por trás dele em termos de profundidade.
Zbuffer é mais utilizado em jogos, devido a sua maestria.

2) Para removermos as superfícies escondidas, primeiramente pegamos o objeto e o analisamos calculando a normal de todos os polígonos e com isso estipulamos onde está o nosso observador, para assim saber se a face está escondida ou não.
Se a seta (normal) estiver no mesmo **sentido** da seta do observador(câmera) ou em um mesmo sentido perpendicular, não conseguimos ver, com isso podemos adotar que essas faces estão escondidas, assim, eliminamos muito processamento de imagem. Além disso, podemos testar a visibilidade de cada pixel, pegando o lugar onde a menor coordenada de Z é visível, assim pintaremos a parte em que teremos o menor Z escondendo as faces que estão atrás.

3) A iluminação global é algo extremamente importante no universo 3d. Ela pode ser dividida em vários subgrupos, sejam eles: brilho, reflexão, textura, sombras. E todos eles dependem da iluminação e não o contrário.
Como por exemplo, podemos ter iluminação sem brilho, mas não podemos ter brilho sem iluminação. E essa iluminação global não se limita apenas as fontes de luzes, elas em um ambiente 3D é uma consequência de uma interação entre objetos. Podemos ter um objeto de uma determinada cor que pode impactar no reflexo de algum outro objeto, um objeto isolado dependendo da perspectiva do nosso observador, pode causar alguma inconsistência no resultado que queremos obter, seja ele relacionado a sombra ou algum outro tipo de resultado de imagem que esperamos como resultado.

4) Como nós seres humanos, usamos os olhos para compreender o mundo, dado a reflexão da luz nos objetos, nossos olhos conseguem interpretar e deter informações sobre determinados objetos. A computação gráfica, como foi dado em aula, nos proporcionou recriar certos comportamentos de reconhecimento de padrões/informações relevantes, no caso, vimos a detecção de características de pessoas em uma dada imagem.

Em alguns casos o processamento de imagem tem que coexistir, para que a visão computacional consiga ser executada com maestria. Nossa imagem encontrada, pode produzir uma imagem de saída e com isso, ela também pode transmitir informações.



O reconhecimento de faces, como foi visto em aula e que também é usado em desbloqueio de celulares sendo de uso mais comum no uso no dia a dia, também utilizasse visão computacional. A face é filmada pela câmera e um algorítimo é utilizado para ver se aquela face é dada ao dono do dispositivo e pode ser desbloqueado, ou se é outra pessoa.

No google, temos outro exemplo, para pesquisarmos uma determinada figura dado a similaridade de uma outra, pode-se apontar várias outras imagens que tem aquela mesma similaridade apontada. Utilizando algoritmos de visão computacional conseguimos obter esses resultados.

Cada imagem que o computador vê, é transposta por um conjunto de pixels, do qual cada um tem um valor que representa as cores que aquele objeto vai ter, ou a intensidade da cor que aquele objeto terá. No sistema RGB, cada pixel não tem só um, mas sim três valores: (90, 90, 10). → (R, G, B)
Indo de 0 a 255, 255 sendo o máximo que podemos ter, significando um byte e mostrando como aquele pixel vai ser mostrado, desenhado e representado.

No entanto, existem certos padrões que o computador não consegue identificar tão perfeito como nós seres humanos, como um exemplo de uma imagem ofuscada, ou com algum certo problema visual. Nós, conseguimos desconstruir aquele padrão de imagem e conseguimos detectar o que está por trás na maioria das vezes. Algo que o computador está longe de conseguir.

5) Para criar um detector para aplicativos com realidade aumentada devemos:
1. Dividir a imagem em regiões

2. Detectar bordas em regiões
Em seguida, uma derivada de Gaussiana é convolvida em cada linha de varredura para estimar o componente do gradiente de intensidade ao longo da linha de varredura.

3. Encontrar segmentos nas regiões
Após a detecção das bordas, devemos usar um algoritmo para construir segmentos de linha em cada região

4. Mesclar segmentos com linhas
Quando todos os segmentos possíveis são mesclados em sua própria região, a mesma operação é repetida para todos os segmentos em toda a imagem.

5. Estender linhas ao longo das bordas

6. Manter as linhas com cantos

7. Encontrar marcadores