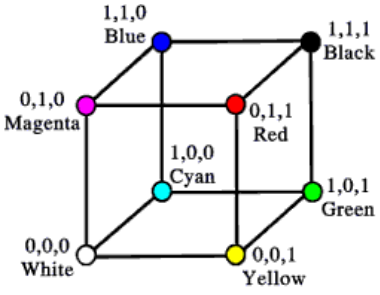
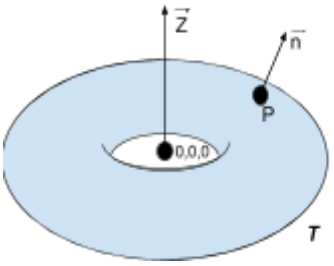
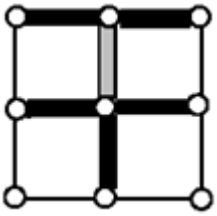
P1 CG 2011.2

	Computação Gráfica 1 Prof. Rodrigo de Toledo	Data: 10/10/2011 P1







1) (2,5 pontos) Dada uma cor ϕ, cujas componentes r,g,b estão descritas no sistema RGB, como converte-la para o sistema CMYK de uma impressora que usa a tinta preta (K) para minimizar o gasto com as tintas coloridas? R,G,B,C,M,Y,K ∈ [0,1]

2) (2,5 pontos) Dado um torus *T* de raio maior R=1, raio menor r, centrado em 0,0,0 e com direção principal em Z, calcule geometricamente a normal **n** no ponto $P_{x,y,z}$ sobre sua superfície.

3) (2,5 pontos) Dada uma malha poligonal cuja topologia 2D é descrita por uma estrutura *half-edge*:

```
class Vertex { Point2D p; H_Edge hEdge;} //hEdge cuja origim é o ponto p
class H_Edge { Vertex vOrig; H_Edge eTwin; Face f; H_Edge eNext;}
class Face { H_Edge HEdge;}
```

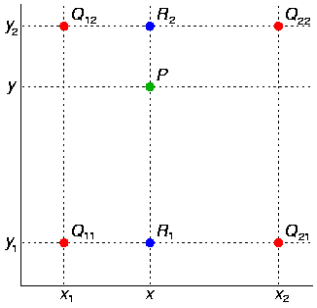
Faça uma função `PrintNeighborEdges(H_Edge edge)` que imprime as arestas vizinhas a uma aresta. Considere que já existe uma função `PrintEdge(H_Edge edge)` que imprime o `id` de uma edge. Atenção, não deve haver repetição de `id` impresso.

4) (2,5 pontos) Assim como o pixel é o menor elemento de uma imagem 2D (*pixel = picture element*), o voxel é o menor elemento de um dado volumétrico 3D (*voxel = volume element*). Em 2D, uma função importante é a interpolação bilinear, que usa os 4 pixels vizinhos de um ponto para determinar o seu valor. Da mesma forma, também é possível fazer a interpolação trilinear de dados volumétricos em um determinado ponto do espaço, usando os 8 vizinhos mais próximos do ponto desejado. Faça um pseudo-código (pode ser em C) da função trilinear do tipo float, descrita pelo cabeçalho abaixo.

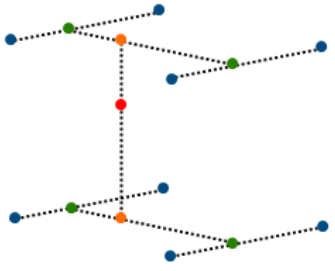
Observações:

- Deixe sempre claro se suas variáveis são do tipo inteiro ou float.
- Atenção nas conversões entre inteiro e float.
- O tamanho discreto do dado volumétrico é dado pelos inteiros `TAM_X, TAM_Y, TAM_Z.`
- `xi, yi, zi` ∈ [0,1], onde 1 representa a maior dimensão na sua direção (ex: `TAM_X` para `xi`)

```
float Trilinear( float V[TAM_Z][TAM_Y][TAM_X], float xi, float yi, float zi);
```



Bilinear



Trilinear

```
def interp3d(x,y,z,cd,xi,yi,zi):
```

```
    '''
    interpolate a cubic 3D grid defined by x,y,z,cd at the point
    (xi,yi,zi)
    '''

    def get_index(value,vector):
        '''
        assumes vector ordered decreasing to increasing. A bisection
        search would be faster.
        '''
        for i,val in enumerate(vector):
            if val > value:
                return i-1
        return None

    xv = x[:,0,0]
    yv = y[0,:,0]
    zv = z[0,0,:]

    a,b,c = xi, yi, zi

    i = get_index(a,xv)
    j = get_index(b,yv)
    k = get_index(c,zv)

    x1 = x[i,j,k]
    x2 = x[i+1,j,k]
    y1 = y[i,j,k]
    y2 = y[i,j+1,k]
    z1 = z[i,j,k]
    z2 = z[i,j,k+1]

    u1 = cd[i, j, k]
    u2 = cd[i+1, j, k]
    u3 = cd[i, j+1, k]
    u4 = cd[i+1, j+1, k]
    u5 = cd[i, j, k+1]
    u6 = cd[i+1, j, k+1]
    u7 = cd[i, j+1, k+1]
    u8 = cd[i+1, j+1, k+1]

    w1 = u2 + (u2-u1)/(x2-x1)*(a-x2)
    w2 = u4 + (u4-u3)/(x2-x1)*(a-x2)
    w3 = w2 + (w2-w1)/(y2-y1)*(b-y2)
    w4 = u5 + (u6-u5)/(x2-x1)*(a-x1)
    w5 = u7 + (u8-u7)/(x2-x1)*(a-x1)
    w6 = w4 + (w5-w4)/(y2-y1)*(b-y1)
    w7 = w3 + (w6-w3)/(z2-z1)*(c-z1)
    u = w7

    return u
```