Program

Program to capture a video from the default camera (0), compute the 2D DCT on the Green component, take the magnitude (phase) and display it live on the screen, divide the picture into blocks of 8x8 pixels and apply a 2D DCT to each, low pass filter, and iverse transform.

- Gerald Schuller, Nov. 2014

- Import relevant modules.

In [1]:
```
import cv2
import numpy as np
import scipy.fftpack as sft
```

- Number of bits per Y pixel, resulting quantization step size for 2^{bits} steps:
  Stufen fuer unterschiedliche Ortsfrequenzen:

In [2]:
```
bits=4
quantstufe1=5.0/(2**bits-1)
bits=3
quantstufe2=1.0/(2**bits-1)
bits=2
quantstufe3=0.6/(2**bits-1)
bits=1
quantstufe4=0.4/(2**bits-1)
bits=0
quantstufe5=8.0/(2**bits-0.99)
```

- vermeide div. durch 0!
  Zus.: 1X4 bits + 2 X 3 bits + 3X2bits + 4X1 bits fuer 64 pixel, also 0.3125 bit pro pixel!

- Quantisierungsstufen in "Maske", anti-diagonalen haben gleiche quantstufen:

In [3]:
```
M = np.zeros((8,8))
M[0,0] = quantstufe1
M = M + np.fliplr(np.diag([1, 1], 6)) * quantstufe2
M = M + np.fliplr(np.diag([1, 1, 1], 5)) * quantstufe3
M = M + np.fliplr(np.diag([1, 1, 1, 1], 4)) * quantstufe4
M = M + np.fliplr(np.tril(np.ones((8, 8)), 3)) * quantstufe5

print(M)
```

```
[[  3.33333333e-01   1.42857143e-01   2.00000000e-01   4.00000000e-01
    8.00000000e+02   8.00000000e+02   8.00000000e+02   8.00000000e+02]
 [  1.42857143e-01   2.00000000e-01   4.00000000e-01   8.00000000e+02
    8.00000000e+02   8.00000000e+02   8.00000000e+02   8.00000000e+02]
 [  2.00000000e-01   4.00000000e-01   8.00000000e+02   8.00000000e+02
    8.00000000e+02   8.00000000e+02   8.00000000e+02   8.00000000e+02]
 [  4.00000000e-01   8.00000000e+02   8.00000000e+02   8.00000000e+02
    8.00000000e+02   8.00000000e+02   8.00000000e+02   8.00000000e+02]
 [  8.00000000e+02   8.00000000e+02   8.00000000e+02   8.00000000e+02
    8.00000000e+02   8.00000000e+02   8.00000000e+02   8.00000000e+02]
 [  8.00000000e+02   8.00000000e+02   8.00000000e+02   8.00000000e+02
    8.00000000e+02   8.00000000e+02   8.00000000e+02   8.00000000e+02]
 [  8.00000000e+02   8.00000000e+02   8.00000000e+02   8.00000000e+02
    8.00000000e+02   8.00000000e+02   8.00000000e+02   8.00000000e+02]
 [  8.00000000e+02   8.00000000e+02   8.00000000e+02   8.00000000e+02
    8.00000000e+02   8.00000000e+02   8.00000000e+02   8.00000000e+02]]
```

In [4]:
```
cap = cv2.VideoCapture(0)
#Get size of frame:
[retval, frame] = cap.read()
[r,c,d]=frame.shape
print(r,c)
```

```
(480, 640)
```

- Create quantization mask of the size of the image, using kronecker product:

In [5]:
```
r8 = r/8
c8 = c/8
Mframe = np.kron(np.ones((r8, c8)), M)
print(Mframe)
```

```
[[  3.33333333e-01   1.42857143e-01   2.00000000e-01 ...,   8.00000000e+0
2
    8.00000000e+02   8.00000000e+02]
 [  1.42857143e-01   2.00000000e-01   4.00000000e-01 ...,   8.00000000e+0
2
    8.00000000e+02   8.00000000e+02]
 [  2.00000000e-01   4.00000000e-01   8.00000000e+02 ...,   8.00000000e+0
2
    8.00000000e+02   8.00000000e+02]
 ...,
 [  8.00000000e+02   8.00000000e+02   8.00000000e+02 ...,   8.00000000e+0
2
    8.00000000e+02   8.00000000e+02]
 [  8.00000000e+02   8.00000000e+02   8.00000000e+02 ...,   8.00000000e+0
2
    8.00000000e+02   8.00000000e+02]
 [  8.00000000e+02   8.00000000e+02   8.00000000e+02 ...,   8.00000000e+0
2
    8.00000000e+02   8.00000000e+02]]
```

- Mask to set to zero the 3/4 highest frequencies, only kep the 1/4 lowest frequencies in each direction for the 8x8 DCT, because of the DCT no longer symmetric about the center:

- Grid of 8x8 blocks:

In [6]:
```
gc=np.zeros((1,c))
gc[0,0:c:8]=np.ones(c/8)
gr=np.zeros((r,1))
gr[0:r:8,0]=np.ones(r/8)
grid=np.ones((r,1))*gc+gr*np.ones((1,c))
#print(grid[0:9,0:9])
```

- Start capturing and process it blockwise:

In [7]:
```
while(True):
    # Capture frame-by-frame
    [retval, frame] = cap.read()
    Y=(0.114*frame[:,:,0]+0.587*frame[:,:,1]+0.299*frame[:,:,2])/255;
    cv2.imshow('Original Video, Y Komponente, 8 bit/Pixel',Y)


    #compute magnitude of 2D DCT of blocks of 8x8 pixels of the green co
mponent
    #by first reshaping the image to width 8 and applying the 1D DCT all
rows, then reshape it back,
    #then transpose it, and again reshape it to width 8 and apply the 1D
DCT to each row, reshape it back,
    #and transpose it back.
    #with norm='ortho' for "energy conservation" in the subbands and for
    #invertibiltity without factor.

    #First reshape green frame as frame with rows of width 8, (rows: ord
er= 'C' ),
    #and apply DCT to each row of length 8 of all blocks:
    frame=np.reshape(frame[:,:,1],(-1,8), order='C')
    X=sft.dct(frame/255.0,axis=1,norm='ortho')

    #shape it back to original shape:
    X=np.reshape(X,(-1,c), order='C')
    #Shape frame with columns of hight 8 by using transposition .T:
    X=np.reshape(X.T,(-1,8), order='C')
    X=sft.dct(X,axis=1,norm='ortho')

    #shape it back to original shape:
    X=(np.reshape(X,(-1,r), order='C')).T
    #Quantize:
    #print('Quantisieren mit Quantisierungsmaske:')
    indices=np.round(X/Mframe)
    #print('De-Quantisieren')
    #de-quantization in the decoder:
    Xrek=indices*Mframe


    #Inverse 2D DCT,
    #Rows:
    X=np.reshape(Xrek,(-1,8), order='C')
    X=sft.idct(X,axis=1,norm='ortho')
    #shape it back to original shape:
    X=np.reshape(X,(-1,c), order='C')
    #Shape frame with columns of hight 8 (columns: order='F' convention)
:
    X=np.reshape(X.T,(-1,8), order='C')
    x=sft.idct(X,axis=1,norm='ortho')
    #shape it back to original shape:
    x=(np.reshape(x,(-1,r), order='C')).T

    cv2.imshow('De-Quantizer mit Quant.-Maske und Inverse 2D DCT (0.3125
bit/Pixel)', x)

    #Keep window open until key 'q' is pressed:
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
```

- When everything done, release the capture:

In [8]:
```
cap.release()
cv2.destroyAllWindows()
```