Program will be invoked using seven parameters where

YourProgram.exe C:/myDir/myImage.rgb Y U V S_w S_h A

- The first parameter is the name of the image, which will be provided in an 8 bit per channel RGB format (Total 24 bits per pixel). You may assume that all images will be of the same size for this assignment (HD size = 1920wx1080h), more information on the image format will be placed on the class website

- The next three parameters are integers control the subsampling of your Y U and V spaces respectively. For sake of simplicity, we will follow the convention that subsampling occurs only along the width dimension and not the height. Each of these parameters can take on values from 1 to n for some n, 1 suggesting no sub sampling and n suggesting a sub sampling by n

- The next two parameters are single precision floats S_w and S_h which take positive values < 1.0 and control the scaled output image width and height independently.

- Finally a integer A (0 or 1) to suggest that antialiasing (prefiltering needs to be performed). 0 indicates no antialiasing and vice versa

Example invocations shown below should give you a fair idea about what your input parameters do and how your program will be tested.

1. YourProgram.exe image1.rgb 1 1 1 1.0 1.0 0

    No subsampling in the Y, U or V, and no scaling in w and h and no antialiasing, which implies that the output is the same as the input

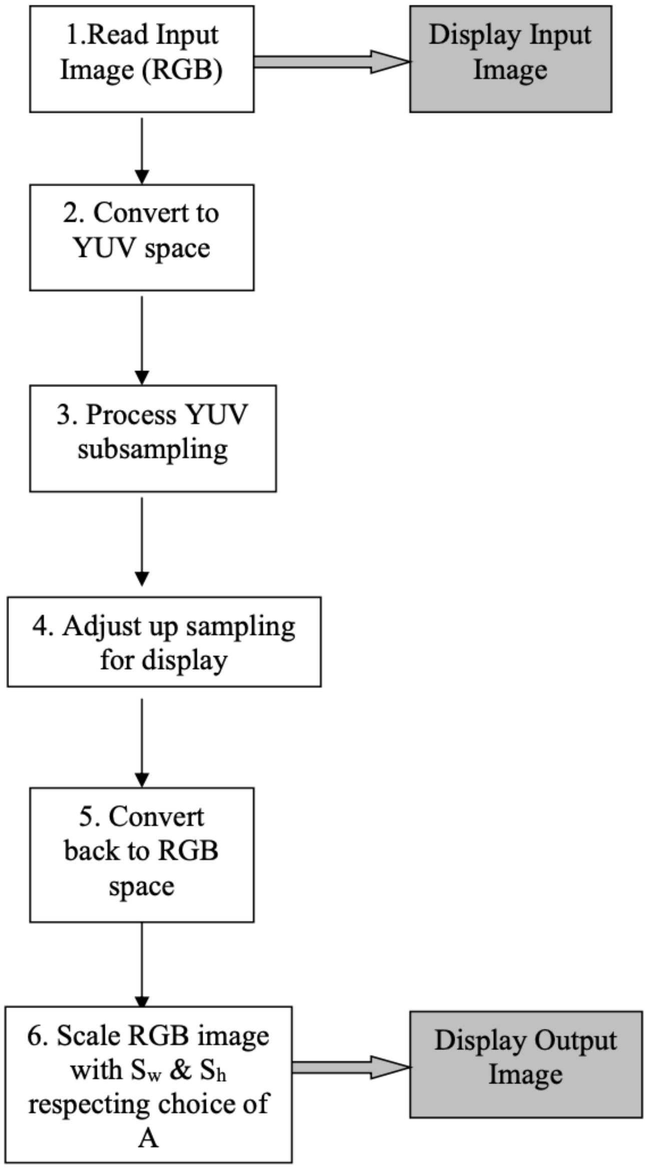
2. YourProgram.exe image1.rgb 1 1 1 0.5 0.5 1

    No subsampling in Y, U or V, but the image is one fourth its original size (antialiased)

3. YourProgram.exe image1.rgb 1 2 2 1.0 1.0 0

    The output is not scaled in size, but the U and V channels are subsampled by 2. No subsampling in the Y channels.

Now for the details - In order the display an image on a display device, the normal choice is an RGB representation. Here is the dataflow pipeline that illustrates all the steps.


1.Read Input Image (RGB)
Display Input Image
2. Convert to YUV space
3. Process YUV subsampling
4. Adjust up sampling for display
5. Convert back to RGB space
6. Scale RGB image with S_w & S_h respecting choice of A
Display Output Image

This code is already provided to you, if you choose to make use of it

The RGB to YUV with the conversion matrix is given below

Sub sample Y U and V separately according to the input parameters

Adjust sample values. Although samples are lost, prior to further process, all values must be interpolated in place

Apply the inverse matrix to get the RGB data

Scale the RGB image and display final output

Conversion of RGB to YUV
Given R, G and B values the conversion from RGB to YUV is given by a matrix multiplication

$$\begin{array}{ccccccc} Y & & 0.299 & 0.587 & 0.114 & & R \\ U & = & 0.596 & -0.274 & -0.322 & & G \\ V & & 0.211 & -0.523 & 0.312` & & B \end{array}$$

Remember that if RGB channels are represented by n bits each, then the YUV channels are also represented by the same number of bits.

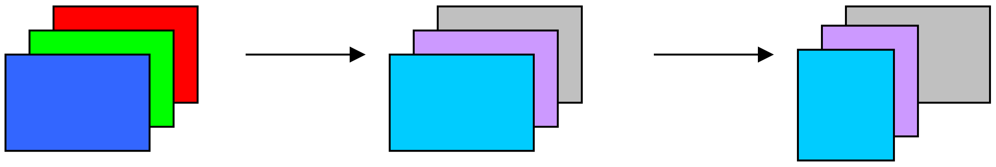
Conversion of YUV to RGB
Given R, G and B values the conversion from RGB to YUV is given by the inverse matrix multiplication

$$\begin{array}{ccccccc} R & & 1.000 & 0.956 & 0.621 & & Y \\ G & = & 1.000 & -0.272 & -0.647 & & U \\ B & & 1.000 & -1.106 & 1.703 & & V \end{array}$$

Sub sampling of YUV & processing
Sub sampling, as you know will reduce the number of samples for a channel.
Eg for the input parameters

YourProgram.exe image1.rgb 1 2 2 256

In this example, the YUV image is not subsampled in Y, but by 2 in U and by 2 in V resulting in



When converting back to the RGB space, all the YUV channels have to be of the same size. However the sampling throws away samples, which have to be filled in appropriately by average the neighborhood values. For example, for the above case a local image area would look like

$Y_{11}U_{11}V_{11}$ Y_{12} $Y_{13}U_{13}V_{13}$ Y_{14}	line 1
$Y_{21}U_{21}V_{21}$ Y_{22} $Y_{23}U_{23}V_{23}$ Y_{24}	line 2
$Y_{31}U_{31}V_{31}$ Y_{32} $Y_{33}U_{33}V_{33}$ Y_{34}	line 3
$Y_{41}U_{41}V_{41}$ Y_{42} $Y_{43}U_{43}V_{43}$ Y_{44}	line 4

The missing values may be filled in using filters. Here is an example

$U_{12} = (U_{11} + U_{13})/2$ $V_{12} = (V_{11} + V_{13})/2$
$U_{14} = (U_{13} + U_{15})/2$ $V_{14} = (V_{13} + V_{15})/2$

Or you may choose to invent your own filter using appropriate valid neighborhood samples

$Y_{11}U_{11}V_{11}$ $Y_{12}U_{12}V_{12}$ $Y_{13}U_{13}V_{13}$ $Y_{14}U_{14}V_{14}$	line 1
$Y_{21}U_{21}V_{21}$ $Y_{22}U_{22}V_{22}$ $Y_{23}U_{23}V_{23}$ $Y_{24}U_{24}V_{24}$	line 2
$Y_{31}U_{31}V_{31}$ $Y_{32}U_{32}V_{32}$ $Y_{33}U_{33}V_{33}$ $Y_{34}U_{34}V_{34}$	line 3
$Y_{41}U_{41}V_{41}$ $Y_{42}U_{42}V_{42}$ $Y_{43}U_{43}V_{43}$ $Y_{44}U_{44}V_{44}$	line 4

Note the samples that you take to fill in values will change depending on the subsampling parameters. The YUV components can now be converted to RGB space.

Scaling with Antialiasing
Your output image width and height will change to a new (smaller) value depending on scale factors S_w and S_h. You will need to create the output image by resampling the input image. This can be achieved by inverse mapping all destination pixel indexes [i,j] to their source location indexes. Depending on whether you need to perform antialiasing the destination resampled pixel value can be the value of your inverse mapped source pixel (A=0) or the average of small neighborhood around the inverse mapped source pixel (A=1). To compute the average, you may use a 3x3 kernel.

What should you submit ?

- Your source code, and your project file or makefile, if any, using the submit program. ***Please do not submit any binaries or data files***. We will compile your program and execute our tests accordingly.
- Along with the program, also submit an electronic document (word, pdf, pagemaker etc format) using the submit program that answers the fore-mentioned analysis questions. You may use any (or all) input images for this analysis.