1> ANALYSIS OF Y (ONLY SUBSAMPLING) – LUMINANCE

U=0, V= 0, Vary Y

U=1, V=0, Vary Y

U=0, V=1, Vary Y

Y:U:V -> 0:0:0

Y:U:V -> 0:1:0

Y:U:V -> 0:0:1

Y:U:V -> 1:0:0

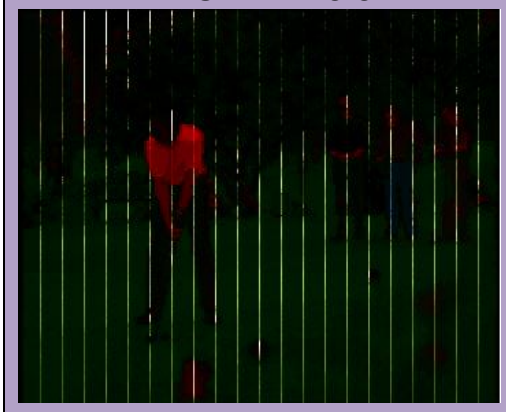
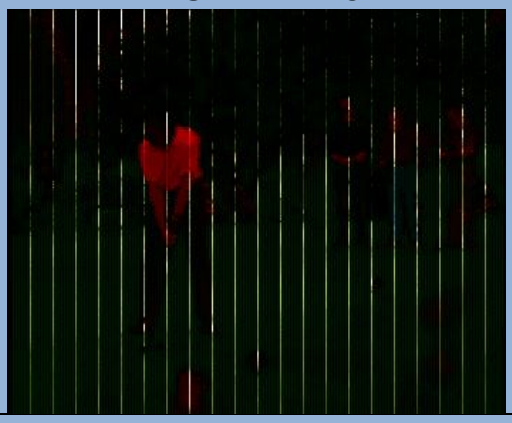
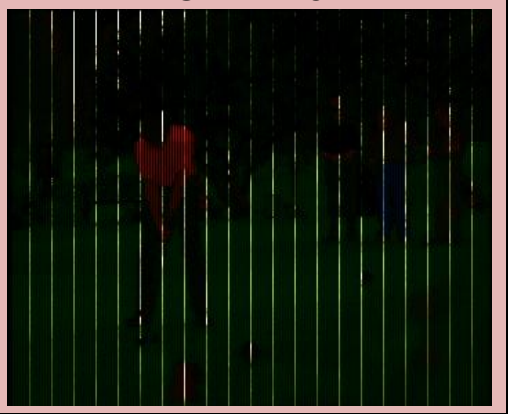
Y:U:V -> 1:1:0

Y:U:V -> 1:0:1

Y:U:V -> 2:0:0

Y:U:V -> 2:1:0

Y:U:V -> 2:0:1

Y:U:V -> 4:0:0 	**Y:U:V -> 4:1:0** 	**Y:U:V -> 4:0:1** 
Y:U:V -> 10:0:0 	**Y:U:V ->10:1:0** 	**Y:U:V -> 10:0:1** 
Y:U:V -> 17:0:0 	**Y:U:V -> 17:1:0** 	**Y:U:V -> 17:0:1** 

Observation:

Column 1 : Sampling luminance reduces the clarity and brightness in the image. Image gets darkened as the sampling increases. Since the Color components are not sampled, chrominance is not affected even when the complete image is darkened **(Unacceptable : 17:0:0)**
Column 2 : Color component V is sampled and we can observe a loss of color i.e; chrominance is affected. Sampling luminance reduces the light in the image. Image gets darkened as the number of samples increases. **(Unacceptable : 17:1:0)**
Column 3 (Vary Y with V=constant and U=0): Color component U is sampled and we can observe a loss of color, many colors are losing their brightness. Sampling luminance reduces the clarity in the image. Image gets darkened as the number of samples increases. . **(Unacceptable : 17:0:1)**

ANALYSIS OF U (ONLY SUBSAMPLING) – CHROMINANCE

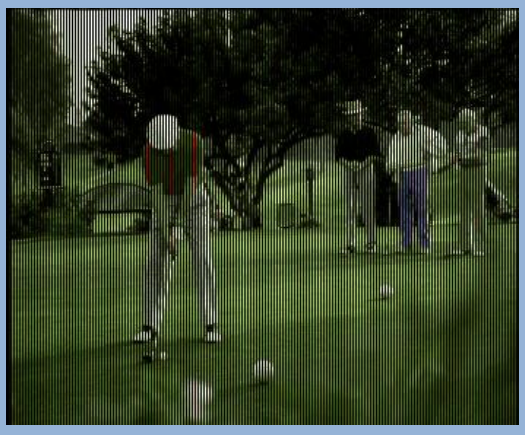
Y=V= 0, Vary U	Y=1, V=0, Vary U	Y=0, V=8, Vary U
Y:U:V -> 0:0:0	Y:U:V -> 1:0:0	Y:U:V -> 0:0:8
		
Y:U:V -> 0:1:0	Y:U:V -> 1:1:0	Y:U:V -> 0:1:8
		
Y:U:V -> 0:2:0	Y:U:V -> 1:2:0	Y:U:V -> 0:2:8
		

Y:U:V -> 0:6:0 	**Y:U:V -> 1:4:0** 	**Y:U:V -> 0:4:8** 
Y:U:V -> 0:10:0 	**Y:U:V ->1:10:0** 	**Y:U:V -> 0:10:8** 
Y:U:V -> 0:17:0 	**Y:U:V -> 1:17:0** 	**Y:U:V -> 0:17:8** 

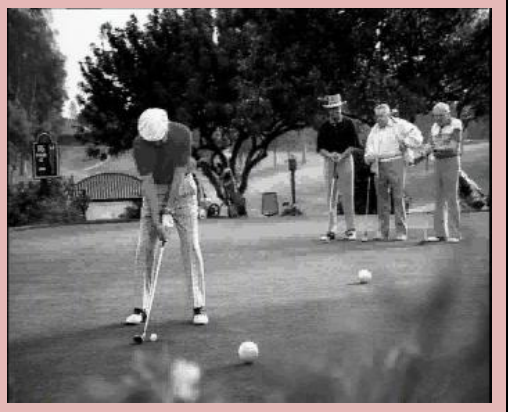
Observation:

Column 1: luminance is not sampled and hence we can see the consistent brightness in the images. Since the Color component **U** is sampled, chrominance is affected. **Green and blue shades are diminished**. **(Unacceptable : 0:17:0)**
Column 2: Sampling luminance deteriorates the brightness in the image. **Green and blue shades are diminished along with the image brightness**, chrominance is affected. Sampling luminance reduces the light in the image. Image darkness remains the same since **Y** is kept constant. **(Unacceptable : 1:17:0)**
Column 3: luminance is not sampled and hence we can see the consistent brightness in the images . Component **V** is sampled and is kept constant throughout. Color component **U** is sampled and we can observe gray scale image because both **U** and **V** are sampled. **(Unacceptable : 0:17:8)**

ANALYSIS OF V (ONLY SUBSAMPLING) – CHROMINANCE

Y=U= 0, Vary V	Y=1, U=0, Vary V	Y=0, U=8, Vary V
Y:U:V -> 0:0:0 	Y:U:V -> 1:0:0 	Y:U:V -> 0:8:0 
Y:U:V -> 0:0:1 	Y:U:V -> 1:0:2 	Y:U:V -> 0:8:1 
Y:U:V -> 0:0:2 	Y:U:V -> 1:0:4 	Y:U:V -> 0:8:2 

Observation:

Y:U:V -> 0:0:4 	**Y:U:V -> 1:0:8** 	**Y:U:V -> 0:8:4** 
Y:U:V -> 0:0:10 	**Y:U:V ->1:0:14** 	**Y:U:V -> 0:8:10** 
Y:U:V -> 0:0:17 	**Y:U:V -> 1:0:18** 	**Y:U:V -> 0:8:17** 

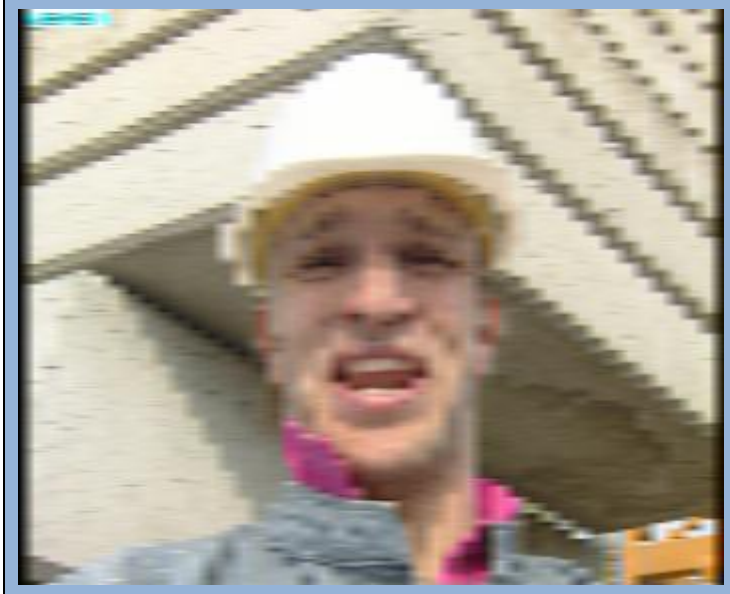
Column 1: luminance is not sampled and hence we can see the consistent brightness in the images. Since the Color component **V** is sampled, chrominance is affected. **Red shades are diminished**. **(Unacceptable : 0:0:17)**

Column 2: Sampling luminance deteriorates the brightness in the image. **Red shades are diminished along with the image brightness**, chrominance is affected. Sampling luminance reduces the light in the image. Image darkness remains the same since **Y** is kept constant. **(Unacceptable : 1:0:18)**

Column 3: luminance is not sampled and hence we can see the consistent brightness in the images . Component **U** is sampled and is kept constant throughout. Color component **V** is sampled and we can observe **gray scale** image because both **U** and **V** are sampled. **(Unacceptable : 0:8:17)**

ANALYSIS OF Y (SUBSAMPLING & UPSAMPLING) – LUMINANCE

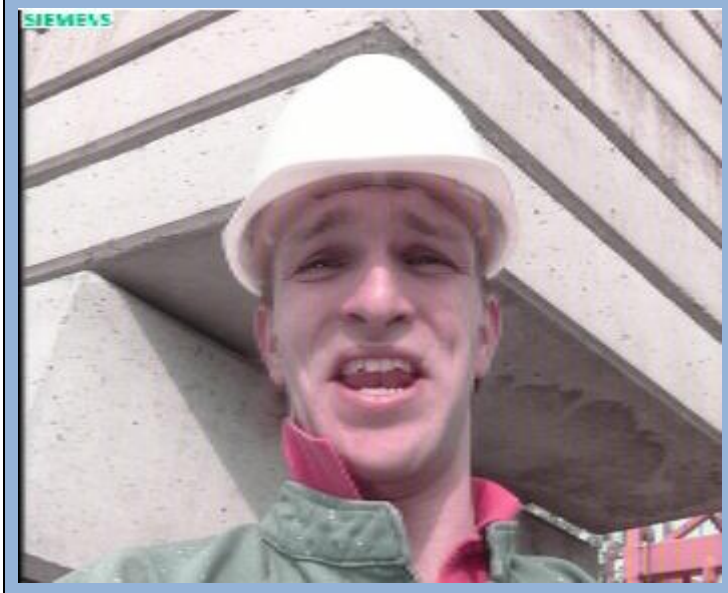
U=0, V= 0, Vary Y

Y = 1	Y = 2
	
Y = 3	**Y = 4**
	

	
Y = 5	**Y = 6**

Observation: Sampling luminance degrades the image quality very drastically. Sampling of level 6 (2^6) results in a image which is unacceptable. Linear interpolation is performed along the width of the image, so the resulting upsampled images shows horizontal stretches of color to fill the empty sampled slots. In this case color is not sampled, therefore the color gradient of the image still remains even when the image is completely distorted at Y=6.

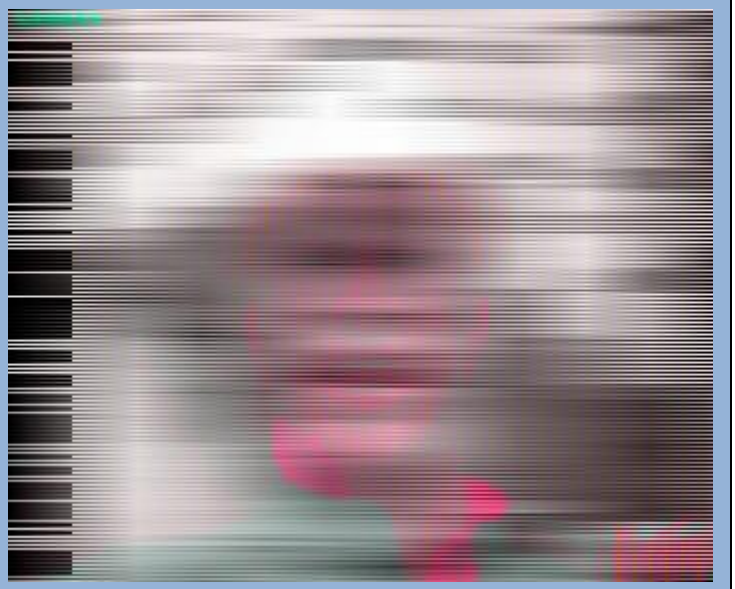
U=3, V= 0, Vary Y

U = 3, Y = 1	**U = 3, Y = 2**
	

U = 3, Y = 3

U = 3, Y = 4

U = 3, Y = 5



U =3, Y = 6

Observation: Sampling luminance degrades the image quality very drastically similar to case 1. Sampling of level 6 (2^6) results in an image which is unacceptable. In this case color component “U” is sampled, therefore the color gradient of the image is changed to somewhat pinkish color. Since Y is sampled at every step, the distortion keeps peaking up and finally reaches unacceptable level at Y=6. Chrominance needs two channels U and V. Since we are sampling only U, we can see some colors in the original image is sampled and changes in the image color can be witnessed.

U=0, V= 3, Vary Y

V = 3, Y = 1	V = 3, Y = 2
	
	
V = 3, Y = 3	V = 3, Y = 4

	
V = 3, Y = 5	**V =3, Y = 6**

Observation: Sampling luminance degrades the image quality very drastically similar to case 1 and case 2. Sampling of level 6 (2^6) results in an image which is unacceptable. In this case color component "V" is sampled and kept constant all over, therefore the color gradient of the image is changed to somewhat greenish color. Since Y is sampled at every step, the distortion keeps peaking up and finally reaches unacceptable level at Y=6. Chrominance needs two channels U and V. Since we are sampling only V, we can see some colors in the original image have lost their original color. Many other up sampling methods are possible and using a more effective algorithm to upsample these sampled images can be implemented.

ANALYSIS OF U (SUBSAMPLING & UPSAMPLING) – CHROMINANCE

Y=0, V= 0, Vary U

U = 1	**U = 2**
	

U = 3

U = 4

U = 5

U = 6

Observation: Sampling Chrominance degrades the image color very drastically. Sampling of level 6 (2^6) results in an image which is unacceptable. In this case color component “U” is sampled and the luminance is not touched (0) all over, therefore the color gradient of the image is changed and the green color channel of chrominance is completely disappeared at U=6 (2^6) . Since U is sampled at every step, the color degradation keeps peaking up, Chrominance needs two channels U and V. Since we are sampling only U, we can see some colors in the original image have lost their original color(green).

Y=2, V= 0, Vary U

Y = 2, U = 1 	**Y = 2, U = 2** 
 **Y = 2, U = 3**	 **Y = 2, U = 4**

Y = 2, U = 5

Y = 2, U = 6

Observation: Sampling Chrominance degrades the image color very drastically. Sampling of level 6 (2^6) results in an image which is unacceptable. In this case color component "U" is sampled along with the luminance. Luminance is sampled only once and is kept constant in all the images above, therefore the color gradient of the image is changed and the green color channel of chrominance is completely disappeared at U=6 (2^6) and the image looks blurry because of the up sampling . Since U is sampled at every step, the color degradation keeps peaking up, Chrominance needs two channels U and V. Since we are sampling only U, we can see some colors in the original image have lost their original color(green). Final image at U = 6 is a blurry image with loss of color.

Y=0, V= 2, Vary U

V = 2, U = 1

V = 2, U = 2

	
V = 2, U = 3	**V = 2, U = 4**

Observation: Sampling Chrominance degrades the image color very drastically. Sampling of level 4 (2^4) results in an image which is unacceptable. In this case color component "U" is sampled along with its associated chrominance component V. Component V is responsible for the red shades in the image and since we are sampling V at level 2 all over the above image series, we can observe a small amount of dullness induced into the red colored parts of the image. Since U is sampled at every step, the color degradation keeps peaking up, Chrominance needs two channels U and V. Since we are sampling both U and V, we can see colors in the original image have lost their original color and the final image at U = 4 and V=2 is almost a gray scale image with a huge loss of color.

ANALYSIS OF V (SUBSAMPLING & UPSAMPLING) – CHROMINANCE

Y=0, U= 0, Vary V

V = 1	**V = 2**
	

V = 3

V = 4

V = 5

V = 9

Observation: Sampling Chrominance degrades the image color very drastically. Sampling of level 9 (2^9) results in an image which is unacceptable. In this case color component “V” is sampled and the luminance is not touched (0) all over, therefore the color gradient of the image is changed and the red color channel of chrominance is completely disappeared at V=9 (2^9) . Since V is sampled at every step, the color degradation keeps peaking up, Chrominance needs two channels U and V. Since we are sampling only V, we can see some colors(red) in the original image have lost their original color(red).

Y=2, U= 0, Vary V

Y = 2, V = 1 	**Y = 2, V = 2** 
 **Y = 2, V = 3**	 **Y = 2, V = 4**

	
Y = 2, V = 5	**Y = 2, V = 6**

Observation: Sampling Chrominance degrades the image color very drastically. Sampling of level 6 (2^6) results in an image which is unacceptable. In this case color component "V" is sampled along with the luminance. Luminance is sampled only once and is kept constant in all the images above, therefore the color gradient of the image is changed and the red color channel of chrominance is completely disappeared at V=6 (2^6) and the image looks blurry because of the up sampling . Since V is sampled at every step, the color degradation keeps peaking up, Chrominance needs two channels U and V. Since we are sampling only V, we can see some colors(red) in the original image have lost their original color(red). Final image at U = 6 is a blurry image with loss of red shades.

Y=0, U= 4, VaryV

U = 4, V = 1	**U = 4, V = 2**
	

U = 4, V = 3

U = 4, V = 4

U = 4, V = 5

U = 4, V = 9

Observation: Sampling Chrominance degrades the image color very drastically. Sampling of level 9 (2^9) results in an image which is unacceptable. In this case color component “V” is sampled along with its associated chrominance component U. Component U is responsible for the green shades in the image and since we are sampling V at level 4 all over the above image series, we can observe a complete removal of the green color components of the image. Since V is sampled at every step, the color degradation keeps peaking up, Chrominance needs two channels U and V. Since we are sampling both U and V, we can see colors in the original image have lost their original color and the final image at U = 4 and V=9 is a complete gray scale image with a complete loss of color.

Conclusion:

Varying U and V removes the color from the image and leaves behind a gray scale version of the image. Varying Y removes the clarity in the image and results in a blurred version. U removes the green component in the image and retains the red component. V removes the red component and retains the green component. Images lose their originality once the sampling increases more than 4 (2^4 = 16 = block size, one sample out of 16 values are taken).

Analysis Question 2:

Image quality of a subsampled image can be improved by using an efficient up sampling algorithm. The method used in the current implementation is to up sample by **Linearly Interpolating** Y, U and V starting from the first pixel in the block till the last one. This contributes to image quality only as long as the number of consecutive missing color values are less than 3- 4. Beyond 4, up sampling cannot regenerate the Un-sampled original image using linear interpolation but generates a "Blurred" version of the original image.

Method 1: Up sampling can be improved by using a more efficient method to fill in the empty color values. The missing values can be filled by taking an average of all the surrounding pixel values in adjacent rows (top and bottom) and columns (right and left). Every pixel which is not on the edges of the image will have 8 different neighboring pixels. Average of all these 8 surrounding pixels is used to fill in the center pixel.

Method 2: Another method of up sampling is done by using a median filter. To run this, you can take a pixel in the image and sort it along with its neighboring pixels based on their intensities. The missing pixel value can be replaced with the median value of the sorted list.

Method 3: Another method to remove noise is by convolving the original image with a mask that represents a low-pass filter or a smoothing operation. For example, the Gaussian mask comprises elements determined by a Gaussian function. This convolution brings the value of each pixel into closer harmony with the values of its neighbors. In general, a smoothing filter sets each pixel to the average value, or a weighted average, of itself and its nearby neighbors; the Gaussian filter is just one possible set of weights.

Results:

Following is the result of implementing **Method 1** to improve up sampling,

Up sampling is performed by filling the empty slot with the average of the adjacent 8 pixels. This method shows a drastic improvement over the **linear interpolation** scheme for up sampling.

Un-sampled : Original Image

Sampled (0:1:0)

Up sampled : Linear Interpolation

Up sampled : Surrounding pixels average

Execution of new Upsampling method:

Changing a global preprocessor constants as follows will execute the new up sampling method,

Change the following default value in **Image.cpp (Line 11-12)**,

In file Image.cpp:

Change the following default values
For Linear interpolation,

```
#define UPSAMPLE_LINEAR_INTERP 1
#define UPSAMPLE_SURROUND_PIXELS 0
```

For Method 1 up sampling,

```
#define UPSAMPLE_LINEAR_INTERP 0
#define UPSAMPLE_SURROUND_PIXELS 1
```

For no up sampling,

```
#define UPSAMPLE_LINEAR_INTERP 0
#define UPSAMPLE_SURROUND_PIXELS 0
```

Implementation of **Method 1** currently works best for sampling rate of 1 (2^1 = 2, 1 sample in a block of 2) because we are taking the average of the first level of adjacent pixels values to fill in the empty slots. For upsampling of a high sampled image, we need to take the average of multiple levels of adjacent pixels.

EXTRA CREDIT:

1> Find at least four dithering algorithms including the Floyd–Steinberg dithering algorithm, and discuss pros and cons of each algorithm.

Solution:

1. **Floyd–Steinberg dithering algorithm:**
   Pros:
   One of the strengths of this algorithm is that it minimizes visual artifacts through an error-diffusion process (error-diffusion algorithms typically produce images that more closely represent the original than simpler dithering algorithms). This algorithm only diffuses the error to neighboring pixels. This results in very fine-grained dithering.
   Cons:
   Slow in execution.

2. **Thresholding algorithm (also known as average dithering)**
   Pros:
   This is the simplest dithering algorithm that exists.
   Cons:
   This algorithm results in immense loss of detail and contouring.

3. **Random dithering algorithm**
   Pros:
   This was the first attempt to remedy the drawbacks of thresholding algorithm. Each pixel value is compared against a random threshold, resulting in a static image. This method doesn't generate patterned artifacts.
   Cons:
   The noise tends to swamp the detail of the image.

4. **Patterning dithering algorithms**
    a. This algorithm uses a fixed pattern. For every pixel in the image the value of the pattern at the corresponding location is used as a threshold. Neighboring pixels do not affect each other,
    b. Making this form of dithering is suitable for use in animations. Different patterns can generate completely different dithering effects.

    There are two Variations of patterning dithering algorithm,

    Halftone dithering algorithm
    Pros:
    This is a form of clustered dithering where the dots tend to cluster together. This helps in hiding the adverse effects of blurry pixels found on some older output devices.

    Ordered dithering algorithm
    Pros:
    Fast and powerful
    Cons:
    This is a form of dispersed dithering. Because the dots don't cluster, the result looks much less grainy. Though simple to implement, this dithering algorithm is not easily changed to work with free-form, arbitrary palettes.

2> Convert a sample color image into a gray scale image.
Solution: Implementation of converting a color image to gray scale image can be executed by changing the following preprocessor constant,

Change the following default value in **Image.cpp (Line 13-15)**,

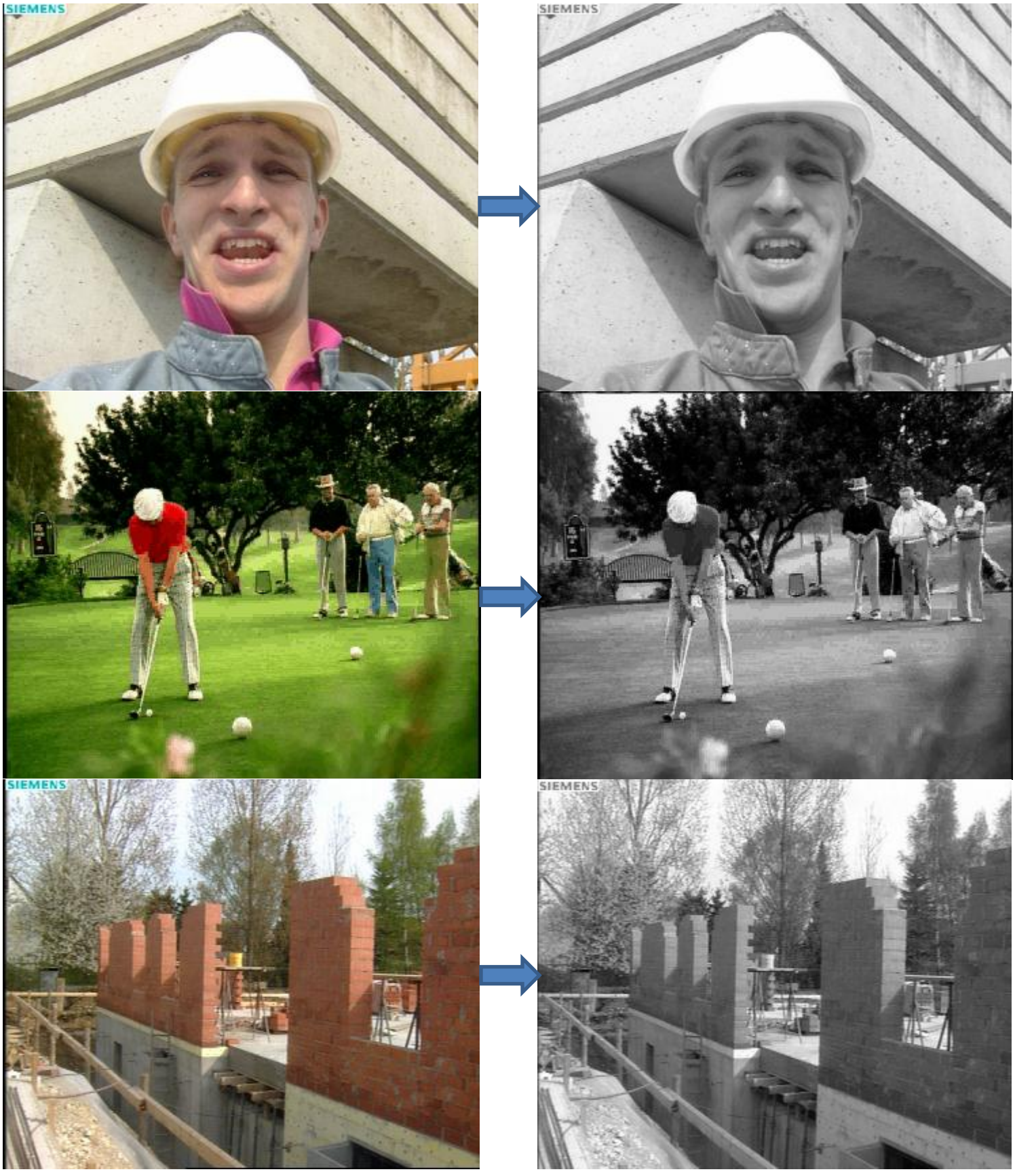
No gray scale conversion:
```
#define GRAYSCALE 0
#define FLOYDSTEINBERG 0
#define AVERAGEDITHERING 0
```

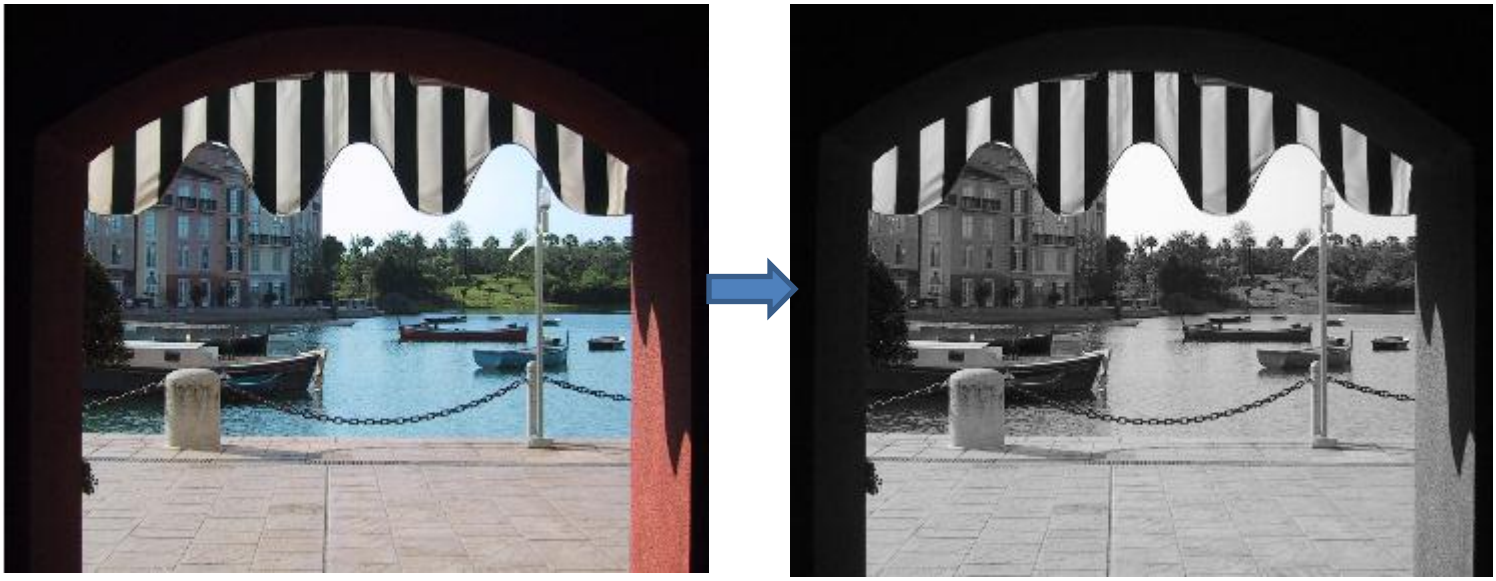
Gray scale conversion:
```
#define GRAYSCALE 1
#define FLOYDSTEINBERG 0
#define AVERAGEDITHERING 0
```

```
Note: Command line arguments are taken as input, but are ignored.
```

Once this change is made, compile and run the program

Examples:





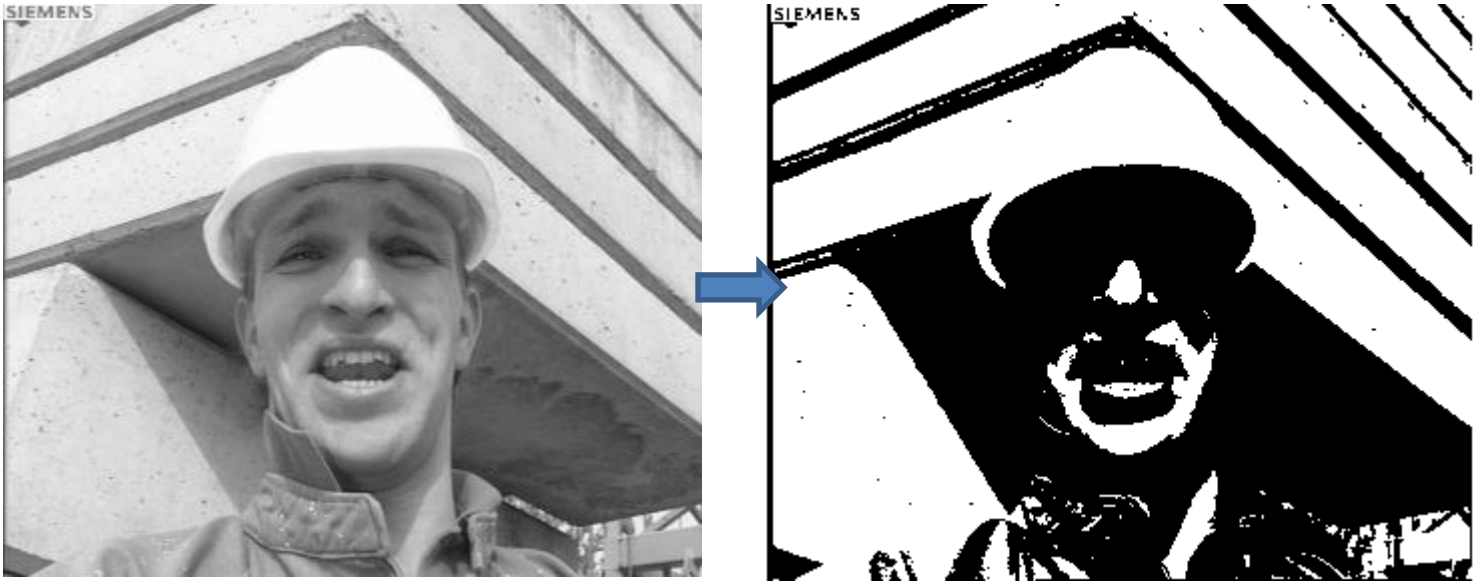
3> Implement the Floyd–Steinberg dithering algorithm and another one of four algorithms that you find, and apply the algorithms to a sample gray scale image. Discuss the difference qualitatively between the original gray scale image and dithered image.

Solution:

Implementation of Thresholding algorithm (also known as average dithering)

a. Average of each of the channels R, G and B are computed
b. Each pixel is compared with the average R, G and B values, if the channel value of the pixel is greater than the channel overall average then push the value to white (255) and if it is less than or equal to the average then push it to black (0).
c. This algorithm takes only 1 bit, but results in immense loss of detail and contouring

Results:




SIEMENS

To Execute "Average Dithering algorithm":

Change the following default value in **Image.cpp (Line 13-15)**,

Default:

```
#define GRAYSCALE 0
#define FLOYDSTEINBERG 0
#define AVERAGEDITHERING 0
```

To run Average Dithering algorithm

```
#define GRAYSCALE 1
#define FLOYDSTEINBERG 0
#define AVERAGEDITHERING 1
```

```
Note: Command line arguments are taken as input, but are ignored.
```

Implementation of Floyd Steinberg Dithering algorithm

Reduce the number of variations (Quantize) and perform error diffusion into other pixels surrounding the current one for obtaining a better granular quantized image.

Gray scale image has got 256 different variations for each color channel. We quantize this into just two different variations which are black and white.

- For every gray scale pixel in the image find the position of gray shade of the pixel within the range **0-1**.
- If the gray value is greater than 0.5 then make that gray pixel to white by storing 255.
- If the gray value is less than or equal to 0.5 then make the gray pixel to black by storing 0.
- Now the **error** in converting the gray shades to black or white will be the difference between the old gray value and the new black/white color which has replaced the gray shade.
- This obtained error is **diffused to the surrounding pixels** in different ratios.

When compared to Average dithering algorithm, this algorithm generates a more granular output. Only disadvantage with Floyd Steinberg algorithm is that it is slow when compared to Average dithering algorithm.

Quantitative improvement:

Number of bits used to store the gray scale image pixel is reduced from 8bits to 1 bit. This reduces a lot of memory that is used to store the image. This dithering algorithm can be used to convert a gray scale image into a black & white image to give it as an input for a black & white output device.

Image detail is improved in Floyd Steinberg algorithm when compared to average dithering algorithm.

Results: Comparison between Average dithering and Floyd Steinberg Dithering

Average Dithering **Floyd Steinberg Dithering**

To Execute "Floyd Steinberg Dithering algorithm":
Change the following default values in **Image.cpp (Line 13-15)**,

Default:

```
#define GRAYSCALE 0
#define FLOYDSTEINBERG 0
#define AVERAGEDITHERING 0
```

To run Average Dithering algorithm

```
#define GRAYSCALE 1
#define FLOYDSTEINBERG 1
#define AVERAGEDITHERING 0
```

```
Note: Command line arguments are taken as input, but are ignored.
```