Table of Contents


```
% Connor Dupuis
% Section: 28944
% TA: Noaki Sawahashi
```

QUESTION 1 COMMENTING

```
% DO NOT REMOVE THE LINE BELOW
% MAKE SURE 'eel3135_lab03_comment.m' IS IN SAME DIRECTORY AS THIS
 FILE
clear;
type('eel3135_lab03_comment.m')


% USER DEFINED VARIABLES
w = 20;           % Width
x = 0:1:79;       % Horiztonal Axis
y = 0:1:79;       % Vertical Axis

% ==> Equation for a 80x80 circle<==
z = round(exp(-1/w.^2*(((y.'-50)/1.5).^2+((x-20)).^2)));

% ==> Applying the two functions to the inputs z and zs respectively
 <==
[xs,ys,zs] = image_system1(z,3,6);
za         = image_system2(zs,90,-4);

% PLOT RESULT WITH SUBPLOT
figure(1);
subplot(1,3,1);         % ==> Creating subplot on firgure 1 <==
imagesc(x, y, z);       % ==> Plotting z <==
axis square; axis xy; % ==> Setting axis values <==
title('Original')
subplot(1,3,2);         % ==> Creating subplot on firgure 1 <==
imagesc(xs, ys, zs); % ==> Plotting zs <==
```

```
axis square; axis xy; % ==> Setting axis values <==
title('After System 1')
subplot(1,3,3);         % ==> Creating subplot on firgure 1 <==
imagesc(xs, ys, za); % ==> Plotting za <==
axis square; axis xy; % ==> Setting axis values <==
title('After System 2')


function [xs, ys, zs] = image_system1(z,Ux,Dy)
%IMAGE_SYSTEM1   ===> Adds Ux zero-valued pixels inserted between each
 pixel
%                     on the x axis (vertical lines). Samples every Dy
 pixels
%                     along the y axis.<===

% ==> Creates a new image of zeros of correct size based of off Ux and
 Dy <==
zs = zeros(ceil(size(z,2)/Dy),ceil(Ux*size(z,1)));

% ==> Creates new borders ys and xs of off correct size <==
ys = 1:ceil(size(z,1)/Dy);
xs = 1:ceil(Ux*size(z,2));

% ==> Asssigns the correct 1 values from Ux and Dy <==
zs(1:end,1:Ux:end) = z(1:Dy:end,1:end);

end

function [za] = image_system2(z,Sx,Sy)
%IMAGE_SYSTEM2   ===> Shifts the image over by Sx pixels and up/down
 Sy by Sy pixels <===

% ====> Creates a new image of zeros of correct size <====
za = zeros(size(z,1), size(z,2));

for nn = 1:size(z,1)
 for mm = 1:size(z,2)
  % ====> Checks boundry conditions so the values are applied in the
 correct location <====
  if nn > Sy && nn-Sy < size(z,1) && mm > Sx && mm-Sx < size(z,2)
   % ====> Assigns 1 values to correct location based off boundary
 conditions <====
   za(nn,mm) = 1/2*z(nn-Sy,mm-Sx);
  end
 end
end


end
```

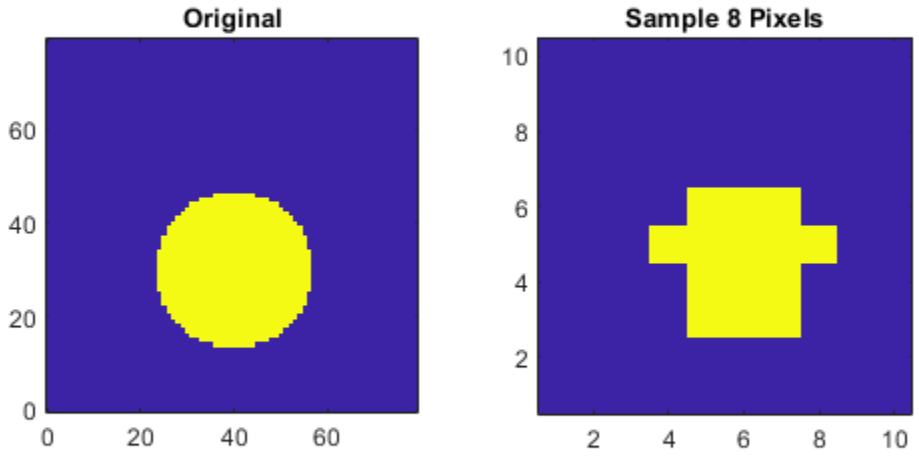
QUESTION 2 VARIABLES (DO NOT CHANGE)

```
w = 20;         % Width
```

```
x = 0:1:79;  % Horiztonal Axis
y = 0:1:79;  % Vertical Axis

% PLOT CIRCLE
z = round(exp(-1/w.^2*((y.'-30).^2+(x-40).^2)));
```

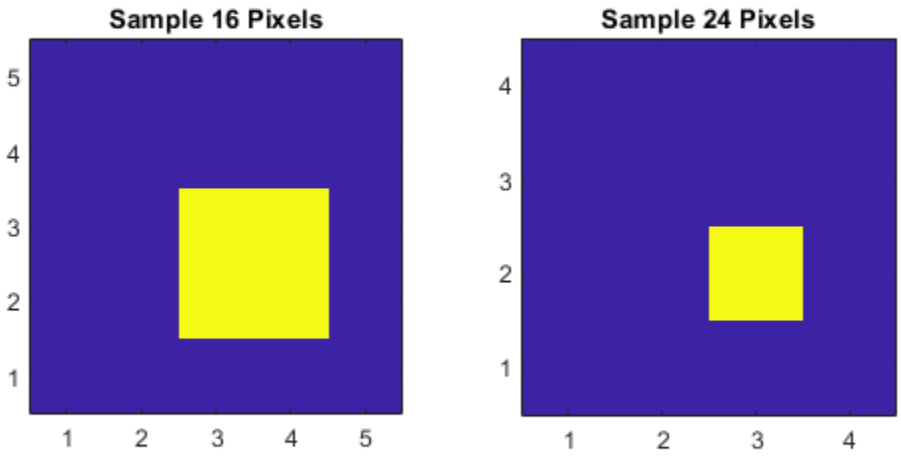
2(b) PEFORM SAMPLING

```
[x8,y8,z8] = sample(z,8);
[x16,y16,z16] = sample(z,16);
[x24,y24,z24] = sample(z,24);

figure(1);

subplot(1,2,1)
imagesc(x, y, z);
axis square; axis xy;
title('Original');

subplot(1,2,2);
imagesc(x8, y8, z8);
axis square; axis xy;
title('Sample 8 Pixels')
```




```
figure(1);
```

```
subplot(1,2,1);
imagesc(x16, y16, z16);
axis square; axis xy;
title('Sample 16 Pixels')

subplot(1,2,2);
imagesc(x24, y24, z24);
axis square; axis xy;
title('Sample 24 Pixels')
```



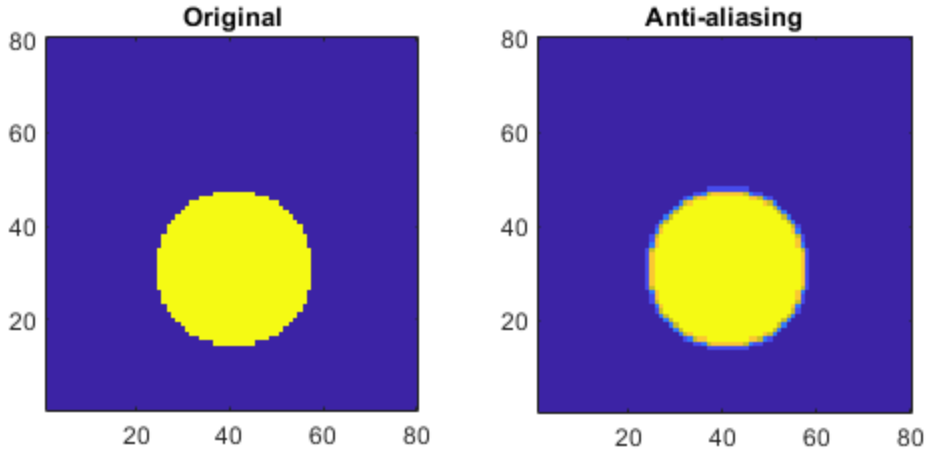
2(c) ANSWER QUESTION

Due to the high amount of aliasing, the circle is turned into a sqaure. Pixels are only taken at every X pixels, so the higher the X, the lower the resolution the image will be. The higher the X the lower the sampling rate.

3(b) PEFORM ANTI-ALIASING (NO SAMPLING)

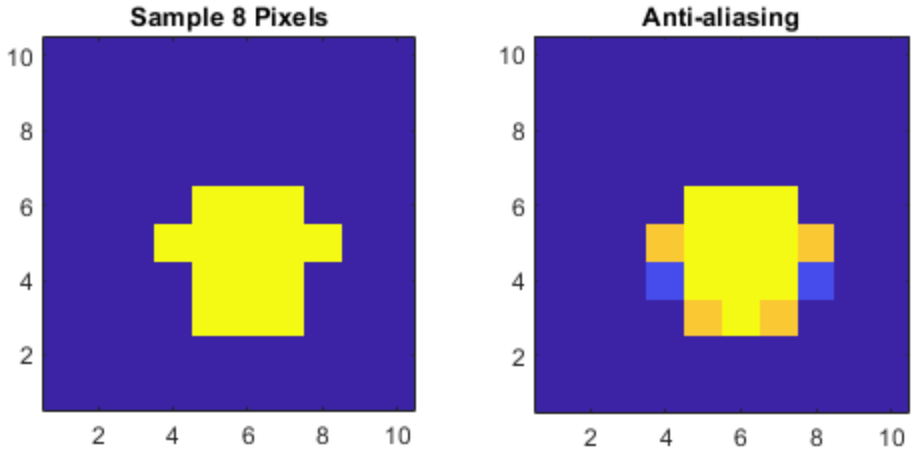
```
zaa = antialias(z);

figure(1);

subplot(1,2,1);
imagesc(z);
axis square; axis xy;
title('Original')
```

```
subplot(1,2,2);
imagesc(zaa);
axis square; axis xy;
title('Anti-aliasing')
```



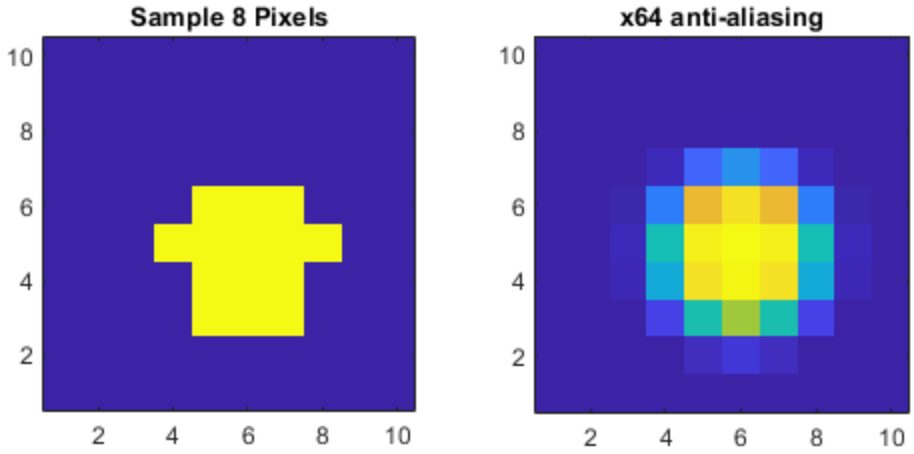
3(c) PEFORM ANTI-ALIASING (WITH SAM-PLING)

```
[x8a,y8a,zaas] = sample(zaa,8);

figure(1);

subplot(1,2,1);
imagesc(z8);
axis square; axis xy;
title('Sample 8 Pixels')

subplot(1,2,2);
imagesc(zaas);
axis square; axis xy;
title('Anti-aliasing')
```



3(d) PEFORM x64 ANTI-ALIASING (WITH SAM-PLING)

```
zaas = z;
for k = 1:64
    zaas = antialias(zaas);
end

[xaas,yaas,zaas] = sample(zaas,8);


figure(1);

subplot(1,2,1);
imagesc(z8);
axis square; axis xy;
title('Sample 8 Pixels')

subplot(1,2,2);
imagesc(zaas);
axis square; axis xy;
title('x64 anti-aliasing')
```

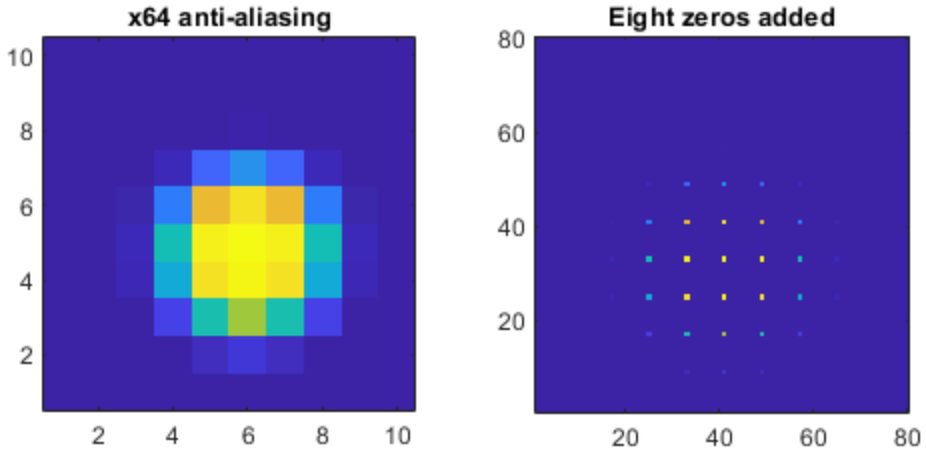


3(e) ANSWER QUESTION

The anti-aliasing filter smooths jagged edges by blending the color of an edge with the color of pixels around it This is usefuel in many areas such as a low resolution texture being percieved more smooth or higher resolution.

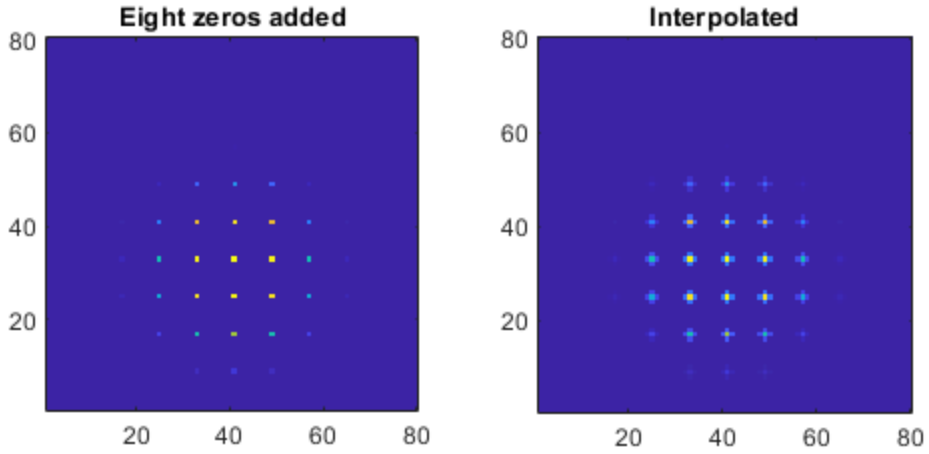
4(b) PERFORM ZERO-ADDING

```
[xz, yz, zz] = addzeros(zaas,8);

figure(1);

subplot(1,2,1);
imagesc(zaas);
axis square; axis xy;
title('x64 anti-aliasing')

subplot(1,2,2);
imagesc(zz);
axis square; axis xy;
title('Eight zeros added')
```



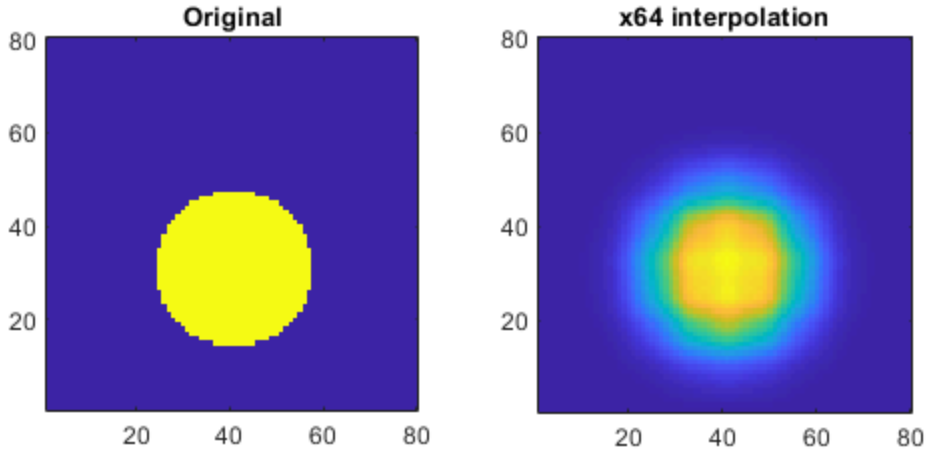
4(c) PEFORM INTERPOLATION

```
zzaa = antialias(zz);

figure(1);

subplot(1,2,1);
imagesc(zz);
axis square; axis xy;
title('Eight zeros added')

subplot(1,2,2);
imagesc(zzaa);
axis square; axis xy;
title('Interpolated')
```



4(d) PEFORM INTERPOLATION x64

```
zzaa = zz;
for k = 1:64
    zzaa = antialias(zzaa);
end

figure(1);

subplot(1,2,1);
imagesc(z);
axis square; axis xy;
title('Original')

subplot(1,2,2);
imagesc(zzaa);
axis square; axis xy;
title('x64 interpolation')
```



4(e) ANSWER QUESTION

The interpolated filter fills in the gaps between pixels based on their surroundings. This is practical in resizing or scaling non vector images.

ALL FUNCTIONS SUPPORTING THIS CODE % %

```
function [xs, ys, zs] = sample(z, D)
%SAMPLE   ===>  Inputs a high-resolution
%              image z and samples every D pixels in both the
 horizontal
%              and vertical direction. <===

zs = zeros(ceil(size(z,1)/D),ceil(size(z,2)/D));

xs = 1:ceil(size(z,1)/D);
ys = 1:ceil(size(z,1)/D);

zs(1:end,1:end) = z(1:D:end,1:D:end);
end

function zaa = antialias(z)
%ANTIALIAS   ===>  Inputs a high-resolution image z and
```

```
%                  outputs high-resolution anti-aliased image zaa <===

zaa = zeros(size(z,1), size(z,2));

for m = 1:size(z,1)
 for n = 1:size(z,2)
        if n > 1 && n < size(z,2) && m > 1 && m < size(z,1)
            zaa(n,m) = 1/2*z(n,m) + 1/8*(z(n-1,m) + z(n+1,m) +
 z(n,m-1) + z(n,m+1));
        end
 end
end


end

function [xz, yz, zz] = addzeros(zaas, U)
%ADDZEROS   ===> The function outputs the image zz, which contains U
%                zero-valued pixels inserted between each pixel from
 zaas,
%                both horizontally and vertically. <===

zz = zeros(U*ceil(size(zaas,1)),ceil(U*size(zaas,2)));

yz = 1:ceil(U*size(zaas,1));
xz = 1:ceil(U*size(zaas,2));

zz(1:U:end,1:U:end) = zaas(1:end,1:end);
end
```

Published with MATLAB® R2020a