Interpolação

Para a resolução da questão 5, foram desenvolvidos dois algoritmos, um algoritmo de método de interpolação por ordem zero e um de interpolação bilinear para gerar um zoom de uma imagem. Após, foram feitas comparações com os resultados dos dois algoritmos e conclusões a respeito de suas respectivas eficiências.

Interpolação de Ordem Zero

O algoritmo de ordem zero utilizado foi o resize 7/3 que utiliza um método de replicação para, a partir de uma imagem de entrada, gerar uma imagem de saída com altura e largura 7/3 maior que a anterior. Abaixo, temos o código em matlab:

function I\_out = resize3to7(I\_in)

%Entrada: I\_in uma imagem em tons de cinza com larguras m por n

%Saída: I\_out uma imagem em tons de cinza com larguras m\*7/3 por n\*7/3

%resultado de uma interpolação resize 7/3 de I\_in

%Inicialização da matriz da imagem de saída

[m,n]=size(I\_in);

I\_out = zeros(floor(m/3)\*7,floor(n/3)\*7);

%Escrita dos valores originais em blocos 7x7

x = 1;

for i = 1:m

y = 1;

for j = 1:n

I\_out(x,y) = I\_in(i,j);

y = y + 2;

if mod(y,7) == 0

y = y + 1;

end;

end;

x = x + 2;

if mod(x,7) == 0

x = x + 1;

end;

end;

%Escrita dos blocos 2x2 com valore na esquerda superior de cada bloco

x = 1;

for i = 1:m

y = 1;

for j = 1:n

I\_out(x + 1,y) = I\_out(x,y);

I\_out(x,y + 1) = I\_out(x,y);

I\_out(x + 1,y + 1) = I\_out(x,y);

y = y + 2;

if mod(y,7) == 0

y = y + 1;

end;

end;

x = x + 2;

if mod(x,7) == 0

x = x + 1;

end;

end;

%Escrita nas colunas finais dos blocos 7x7

y = 7;

for j=1:n/3

for i = 1:floor(m/3)\*7

I\_out(i,y) = I\_out(i,y - 1);

end;

y = y + 7;

end;

%Escrita nas linhas finais dos blocos 7x7

x = 7;

for i=1:m/3

for j = 1:floor(n/3)\*7

I\_out(x,j) = I\_out(x - 1,j);

end;

x = x + 7;

end;

Para imagens coloridas, será distribuído os canais rgb da imagem e, então, será feita a interpolação de cada canal individualmente e, após, a concatenação dos resultados das interpolações de cada canal. Veja o código a seguir:

function I\_out = resize3to7color(I\_in)

%Entrada: I\_in uma imagem rgb com larguras m por n

%Saída: I\_out uma imagem rgb com larguras m\*7/3 por n\*7/3 resultado de uma interpolação resize 7/3 de I\_in

I\_r = I\_in(:,:,1); % Red channel

I\_g = I\_in(:,:,2); % Green channel

I\_b = I\_in(:,:,3); % Blue channel

I\_out = cat(3, resize3to7(I\_r), resize3to7(I\_g), resize3to7(I\_b));

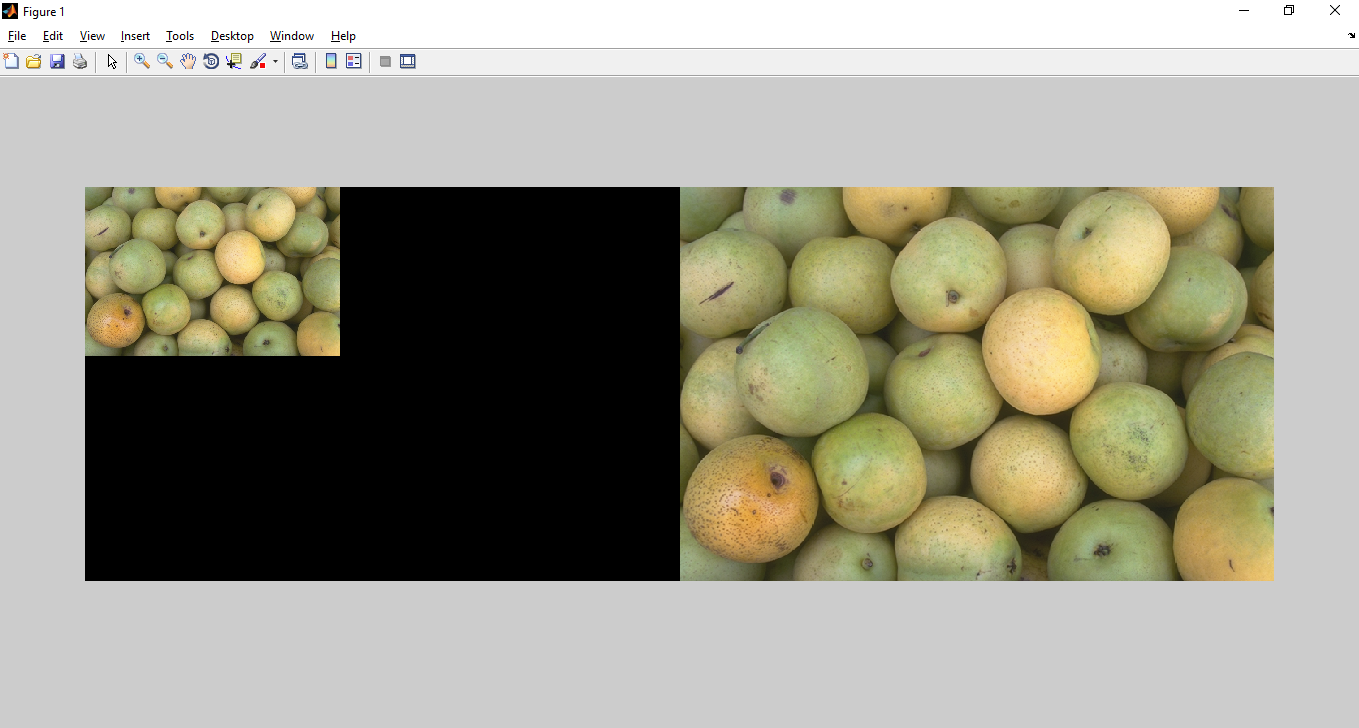
Podemos fazer o teste utilizando o código a seguir:

img = imread('pears.png');

img\_new = resize3to7color(img);

imshowpair(uint8(img), uint8(img\_new), 'montage');

A seguir, a imagem da esquerda é a original e a da direita é a imagem resultado da operação:



Podemos ver que houve um aumento na altura e largura da imagem 7/3 e um alto surgimento de borramento de borda.

Interpolação Bilinear com Fator Dois

O próximo passo é desenvolver um algoritmo de interpolação bilinear com fator dois. Para isso, foi utilizado o método rápido para interpolação bilinear mostrado em aula. Abaixo, temos o código para imagens em tons de cinza:

function I\_out = BilinearInterpolation(I\_in)

%Entrada: I\_in uma imagem em tons de cinza com larguras m por n

%Saída: I\_out uma imagem em tons de cinza com larguras 2\*m-1 por 2\*n-1

%resultado de uma interpolação bilinear com fator dois de I\_in

%Inicialização da matriz da imagem de saída

[m,n]=size(I\_in);

I\_out = zeros(2\*m - 1,2\*n - 1);

%Escrita dos valores originais

x = 1;

y = 1;

for i=1:m

for j=1:n

I\_out(x,y) = I\_in(i,j);

y = y + 2;

end;

y = 1;

x = x + 2;

end;

%Escrita dos valores médios das colunas

x = 1;

y = 1;

for i=1:m

for j=1:n - 1

I\_out(x,y + 1) = (1/2)\*(I\_out(x,y) + I\_out(x,y + 2));

y = y + 2;

end;

y = 1;

x = x + 2;

end;

%Escrita dos valores médios das linhas

x = 1;

for i=1:m - 1

for j=1:2\*n - 1

I\_out(x + 1,j) = (1/2)\*(I\_out(x,j) + I\_out(x + 2,j));

end;

x = x + 2;

end;

Para imagens coloridas, será feito o mesmo que o outro algoritmo, distribuído os canais rgb da imagem e, então, feita a interpolação de cada canal individualmente e, após, a concatenação dos resultados das interpolações de cada canal. O código está a seguir:

function I\_out = ColorBilinearInterpolation(I\_in)

%Entrada: I\_in uma imagem rgb com larguras m por n

%Saída: I\_out uma imagem rgb com larguras 2\*m - 1 por 2\*n - 1 resultado de

%uma interpolação bilinear com fator dois de I\_in

I\_r = I\_in(:,:,1); % Red channel

I\_g = I\_in(:,:,2); % Green channel

I\_b = I\_in(:,:,3); % Blue channel

I\_out = cat(3, BilinearInterpolation(I\_r), BilinearInterpolation(I\_g), BilinearInterpolation(I\_b));

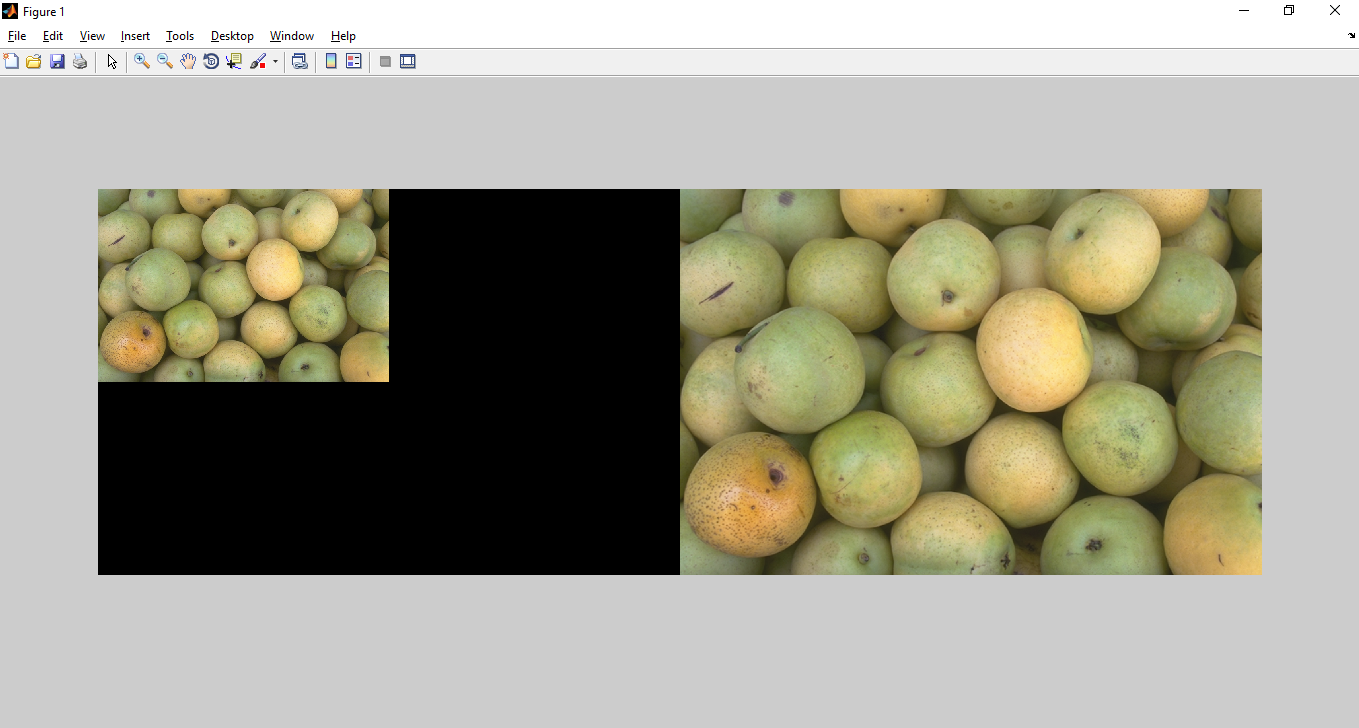
Podemos fazer o teste utilizando o código a seguir:

img = imread('pears.png');

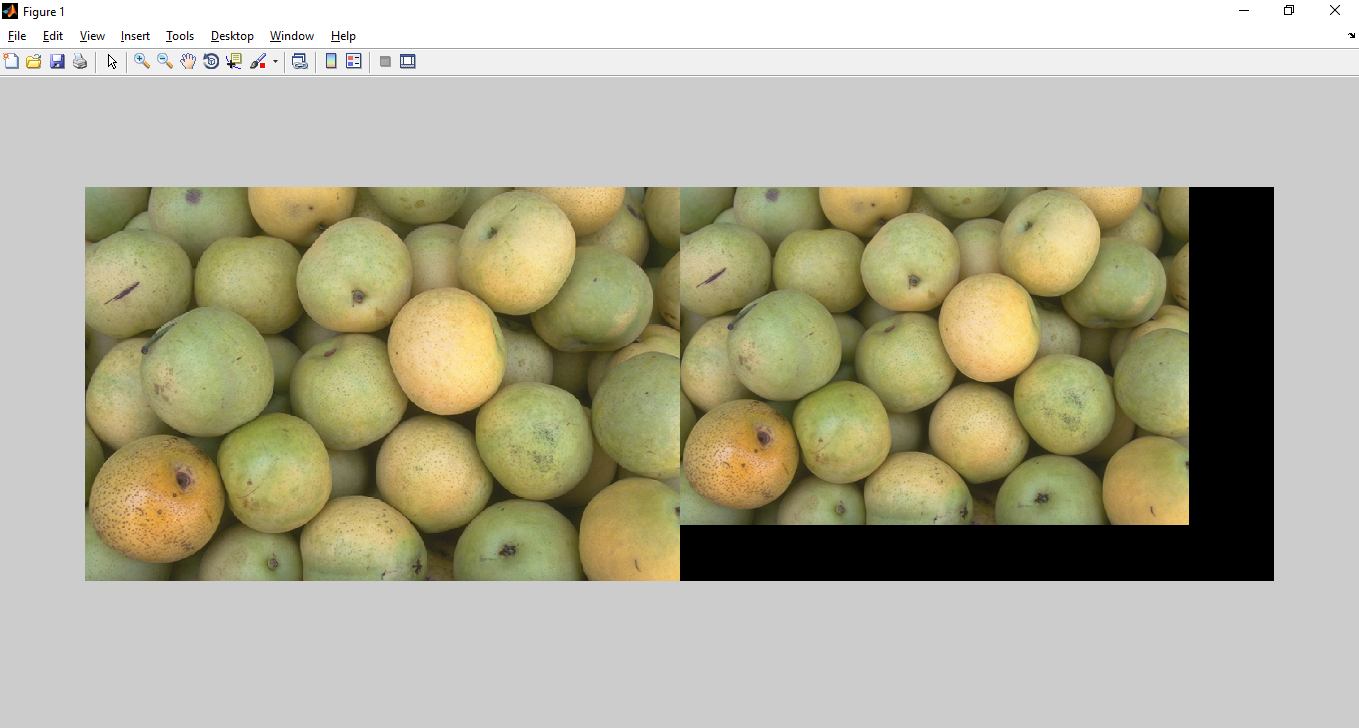
img\_new = ColorBilinearInterpolation(img);

imshowpair(uint8(img), uint8(img\_new), 'montage');

A seguir, a imagem da esquerda é a original e a da direita é a imagem resultado da operação:



Podemos ver que a imagem dobrou de tamanho, não alcançando um zoom tão grande quanto o do primeiro teste e ainda existe borramento de borda. A seguir, veremos a comparação dos dois resultados:



Através da comparação final, podemos perceber que, apesar de um pouco menor, o resultado da interpolação bilinear com fator 2 é mais eficiente por apresentar menos borramento nas bordas.