Creating an animation and movie in Matlab using chroma key compositing (green screening)

Chroma key compositing of images, or chroma keying, is also known as green screen or blue screen technology

We want to move an irregularly shaped object (e.g., a person) across a scene (e.g., a TV weather map).

These are the starting images that we use in this example. The face is the object and the forest is the scene. The face is contained in a rectangular JPG image, so it is read into Matlab into three rectangular pages (one for each RGB color) of a 3D array. The background of the irregularly shaped object image is surrounded by a background of one color - one set of RGB values.

The RGB color values of the screen or background against which the object is drawn or photographed should be chosen to not include color values in the object.

When you get an object image file from the web or by a screenshot, it may be hard to get a background that consists of only one set of RGB values. In that case, you would have to use a paint program to edit the object image's background. For the purposes of this explanation, we created the object image in a paint program so that we could make sure the background was completely uniform.

Here we are using a black background instead of a blue or green screen. I used a simple paint program to make a happy face.

Using Matlab's image or imshow functions, the vertical or y coordinate is the row number in the scene image matrix and increases in value down the screen, from 1 to the number of rows in the scene.

The horizontal or x coordinate is the column number in the scene image matrix and increases in value from left to right, from 1 to the number of columns in the scene.

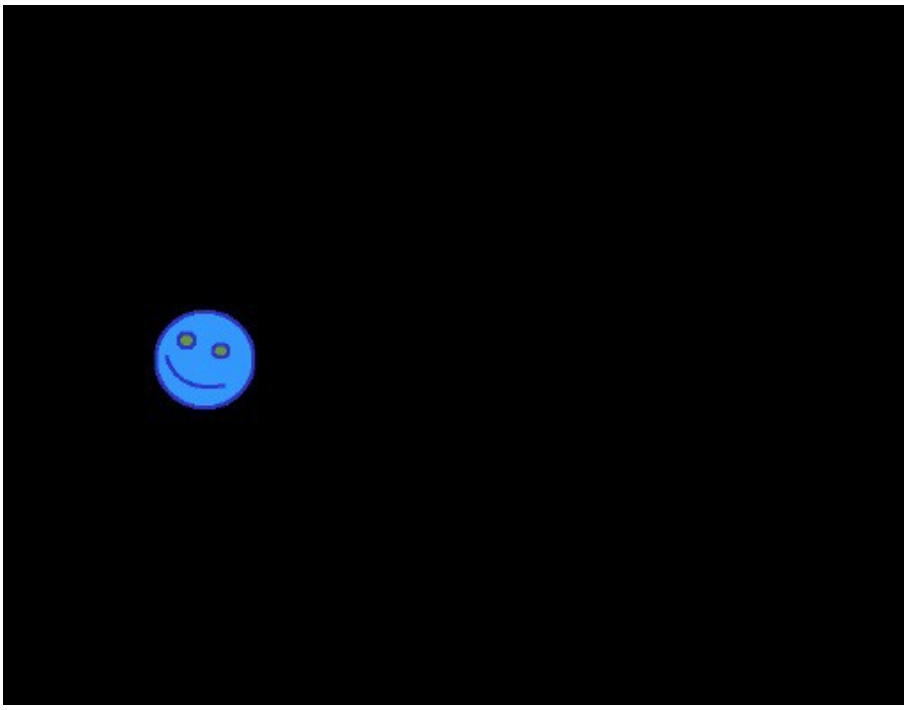
Let's say we have arrays of x and y values (column and row values) at which we want to place the object. Here a given x and y value will specify the location of the left-top corner of the complete object image, which here includes the back border around the face.

The first step is to create 3D image array of the same size of the scene image array and the same color - the "green screen" color - as the background of the object image. This is the "canvas" in our example Matlab scripts.

```
gs = [0 0 0] % "green screen" RGB color values
pg = zeros(brows,bcols,'uint8');% uint8 is 8-bit unsigned integers to save memory
canv = cat(3,pg+gs(1),pg+gs(2),pg+gs(3));
```

Now put the object on the canvas or "green screen" (black here). This is done by replacing the desired range of rows and columns in the three pages of the 3D canvas array with the 3D matrix of the object image.

```
% put object on canvas at x,y coordinates
canv(1+y : orows+y, 1+x : ocols+x, :) = object; % object size is orows,ocols
```



Then get the array indices of all elements that are not the background color.

```
% find linear (sequential) indices of elements of object on canvas
% whose values are not equal to green screen values
ind1 = find(canv(:,:,1) ~= gs(1));
ind2 = find(canv(:,:,2) ~= gs(2));
ind3 = find(canv(:,:,3) ~= gs(3));
```

Then replace the values in the scene at these indices with the values from the object on the canvas.

```
% make copies of each page of canvas and scene
% since we have linear indices for each page
canv1 = canv(:,:,1);
canv2 = canv(:,:,2);
canv3 = canv(:,:,3);

scene1 = scene(:,:,1);
scene2 = scene(:,:,2);
scene3 = scene(:,:,3);

% replace elements in scene pages with elements from canvas pages
scene1(ind1) = canv1(ind1);
scene2(ind2) = canv2(ind2);
scene3(ind3) = canv3(ind3);

% concatenate (cat) the pages into one composite image 3D array
compImage = cat(3,scene1,scene2,scene3);
```

Repeat this process using different x,y coordinates to move the object across the scene. Start with a blank canvas each time, or you will get a trail of left-over object images.

This is the same procedure used in professional video production such as weather reports, first-down lines on a football field, advertisements on the wall behind a baseball batter.

We are manipulating data in image arrays in order to learn, in a fun way, how to manipulate data using Matlab.

Video production people don't have to directly manipulate the image arrays like we are doing. They use commercial software that does this. The same principles are involved. The commercial software runs faster than our programs because they are compiled to machine code (in binary 0's and 1's) that runs directly on the computer hardware and, thus, runs very fast.

Part 2 - Matlab scripts

Below is a Matlab script that creates an animation and saves it to a movie file. The soccer ball is the "object" and the bench and the locker room is the "scene." The original soccer ball image has a rectangular white background. The script places the ball in the scene using "chroma key compositing" or "green screening" using a green screen background or background of another color - white in this case for the white background of the original soccer ball image. The final animation isn't perfect - there is some extraneous green screen left - but the purpose here is to learn the principles.

The subroutines are listed below except for fGetCoordinates, which generates arrays of x and y coordinates (column and row values) on which to place the object in the scene. The function fGetCoordinates is what you would need to write to match the system that you are modeling.

```
% MAIN
% animation of object moving in scene
% Uses user-written functions:
%   fLoadImageFile, fImageRotate
%   fGetCoordinates, fPlaceImageOnScene

% this example moves one object across a scene
% you can move more than one object, see OPTION notes below

clear
close all
clc

% LOAD BACKGROUND SCENE FROM JPG FILE IN MEMORY
%    file needs to be in this folder or use file path
```

```
scene = fLoadImageFile('scene.jpg');

% LOAD OBJECT FROM JPG FILE INTO MEMORY
object = fLoadImageFile('ball.jpg');

% OPTION: load more objects

% OPTION: rotate object and store images in cell array
nr = -10; % number rotated images, nr > 0 is CCW, nr < 0 is CW, 1 for no rotation
gs = 255; % background "green screen," varies with object image background
thr = 15; % threshold for background green screen
objectCellArray = fImageRotate(object,nr,gs,thr);

% CALL FUNCTION fGetCoordinates TO GET x,y COORDINATES FOR ENTIRE ANIMATION

% get number rows & columns of scene and object
[sRows sCols p] = size(scene);
[oRows oCols p] = size(objectCellArray{1});

[x y c] = fGetCoordinates(oRows,oCols,sRows,sCols,abs(nr));
% each element in arrays x,y,c correspond to a different time step
% array x contains horizontal coordinates of image location
% array y contains vertical coordinates of image location
% array c contains rotation angle index of object

% OPTION: get coordinates for any additional objects
% OPTION: start playing a sound track (not used here)

% STEP THROUGH ANIMATION

% when use getframe to make movie
% make sure figure is separate floating window and not
% docked in main Matlab master window
fps = 16; % frames per second
fpp = 1/fps; % frame period
% frame rate won't be accurate in repeat
% because of time to generate images
% but will be accurate when movie has been captured and is replayed

for i = 1:length(x)

    % OPTION: get desired rotation of object image
    object = objectCellArray{c(i)};

    % place object on scene
    % gs (green screen) value here has values for all three RGB layers
    gs = [255 255 255];
    compositeImage = fPlaceImageOnScene(scene,object,gs,x(i),y(i));

    % OPTION: place any additional object(s) on composite image

    % display final composite image for this frame
    % options are to use imshow or image

    imFlag = 1; % choose 1 for imshow, 2 for image

    switch imFlag
        case 1
            % imshowMag at 20 percent full scale saves as 8.3 MB avi movie
            % imshowMag at 40 percent full scale saves as 33.4 MB avi movie
            % WARNING: imshow at full scale takes too much memory
            imshowMag = 40; % percent of full scale
            imshow(compositeImage,'InitialMagnification',imshowMag)
        case 2
            % image produces a 40.9 MB avi file
```

```
            image(compositeImage)
            axis off % on or off, on shows axis ticks and coordinate values
    end

    % control play rate
    pause(fpp)
    % OPTION: get movie frame
    M(i) = getframe;
end

% now play the movie we recorded
n = 1; % number times to play
movie(M,n,fps)

% now save movie
fprintf('SAVING MOVIE - THIS MAY TAKE SOME TIME - WAIT UNTIL MATLAB NOT BUSY \n')
fprintf('DO NOT TRY TO OPEN MOVIE DISK FILE UNTIL MATLAB NOT BUSY \n')
% Mac can only write with 'Compression','None'
% Win can also use 'Compression','Cinepak'
movie2avi(M,'benchAndBall.avi','Compression','None');
```

FUNCTION FILE

```
function im = fLoadImageFile(tFile)
    % FUNCTION: fLoadImageFile
    % RETURNS: array with JPG image data
    % INPUTS: tFile = file path to JPG image file
    im = imread(tFile);
end
```

FUNCTION FILE

```
function bb4 = fImageRotate(bb,nr,gs,thr)
    % FUNCTION: fImageRotate
    % RETURNS: cell array with copies of rotated image
    % INPUTS:
    %   bb is 3D array of JPG image (bb from orig bouncing ball)
    %   nr is number of rotations = number of images returned
    %      nr < 0 for clockwise rotation, 1 for image with no rotation
    %   gs is "green screen" setting, 0 for black background,
    %      255 for white
    %   thr is threshold for setting background light gray to white,
    %      and dark gray to black, that is
    %      background > (255-thr) is set to 255, or < thr is set to 0
    % needs improvement to handle more complex "green screening"

    c = 0; % initialize cell counter for cell array

    while c < abs(nr) % want nr total copies

        c = c+1; % increment cell counter
        a = (c-1)*360/nr; % angle in degrees
        br = imrotate(bb,a,'nearest','crop'); % rotate image

        % set "green screen" with threshold value for 0=black, 255=white
        switch gs
            case 255
                % image has white background
                % rotating produces black corners
                % so set those corners to white
                % note: needs improvement, sets all black to white now
                filter = find(br == 0); % indices of br elements == 0
                br(filter) = 255; % set those elements to gs value
                % change almost white pixels to white
```

```
                filter = find(br > 255-thr);
                br(filter) = 255;
            case 0
                % image has black background
                filter = find(br < thr);
                br(filter) = 0;
            otherwise
                disp('NO THRESHOLD FOR GS NOT 0 OR 255')
        end

        bb4{c} = br; % add rotated image to new cell in cell array

    end
```

FUNCTION FILE

```
function compImage = fPlaceImageOnScene(scene,obj,gs,x,y)
    % FUNCTION: fPlaceImageOnScene
    % RETURNS: 3D array with JPG image data
    % INPUTS:
    %   scene is 3D array with background (scene) image data
    %   obj is 3D array with object image data
    %   gs is array of 3 RGB values of green screen (object background)
    %     which is background of object image
    %   x and y are coordinates of scene at which to place
    %     top-left corner of object
    % places an image on a scene using "chroma key compositing"
    % using a "green screen" (or other color screen)
    % for object image background surrounding object
    % of irregular shape
    % see http://en.wikipedia.org/wiki/Chroma_key

    [srows scols spg] = size(scene);
    [orows ocols opg] = size(obj);

    % create our "green screen"
    % first, make one page of all zeros
    % JPG uses data type uint8 to save memory space
    % uint8 is 8-bit unsigned integers vs. default 64-bit double-precision
    % note binary 2^8 = 256, and 0-255 is 256 numbers
    pg = zeros(srows,scols,'uint8');

    % make the green screen canvas
    canv = cat(3,pg+gs(1),pg+gs(2),pg+gs(3));

    % put object on canvas at x,y coordinates
    canv(1+y : orows+y, 1+x : ocols+x, :) = obj;

    % find linear (sequential) indices of elements of object on canvas
    % whose values are not equal to green screen values
    ind1 = find(canv(:,:,1) ~= gs(1));
    ind2 = find(canv(:,:,2) ~= gs(2));
    ind3 = find(canv(:,:,3) ~= gs(3));

    % put object from canvas onto scene to form composite image

    % make copies of each page of canvas and scene
    % since we have linear indices for each page
    canv1 = canv(:,:,1);
    canv2 = canv(:,:,2);
    canv3 = canv(:,:,3);
```

```
    scene1 = scene(:,:,1);
    scene2 = scene(:,:,2);
    scene3 = scene(:,:,3);

    % replace elements in scene pages with elements from canvas pages
    scene1(ind1) = canv1(ind1);
    scene2(ind2) = canv2(ind2);
    scene3(ind3) = canv3(ind3);

    % concatenate (cat) the pages into one composite image 3D array
    % which will be returned by this function
    compImage = cat(3,scene1,scene2,scene3);

end
```