Image Processing I

EECE 4353 / 5353 Fall 2018

Laboratory 1: Introduction to Images and Matlab

Due Midnight Friday 7 September 2018

The goals of this lab are (1) to learn how to load, to display, and to save 24-bit, truecolor images and (2) color mapped images using Matlab, (3) to learn how to perform basic mathematical and logical operations on images and to understand their results, and (4) to understand the difference between truecolor and color mapped images and their various internal data formats.

In this lab assignment (and all the others) you will be writing your own image processing functions. Unless explicitly stated otherwise, you may not use the functions from the Matlab image processing toolkit by themselves or within your functions to solve the problems in the labs. You may compare the results of your own functions against those of the IP toolkit for your own information but those results may not be used in your lab reports.

Help can be gotten on any Matlab command (or function) by typing the command's name in the text box in the upper right corner of the main Matlab window. It will often give you many choices in the widow that drops down. Ususally you will want to select the first choice that has "Matlab " beside it on the right. Also you can type `help command_name`, in the command window. *E.g.* to get help on `imwrite`, type `help imwrite` at the command prompt. This will print the help information in the command window itself. Another way is to press the <F1> key, select the "Index" tab in the window, and type the command name into the search box.

Always open a new figure window before displaying an image, otherwise the image will appear in the previous window with the attributes of that window such as its size and color map (if there is one). Close the figure windows that are not relevant to the part of the assignment your working on so not to overwhelm your computer's resources.

Get in the habit of terminating every line of Matlab code with a semicolon, ";" If you do not, the program writes the result of the operation to the command window. If an image happens to be the result, all $R \times C \times B$ numbers will be dumped to the screen. (Doesn't hurt anything but it's annoying.)

In completing this assignment (and the others) if specific images are not supplied, you may use any images you like providing that they are of the type specified by the problem description (*e.g.* 24-bit truecolor). Please do not use images that are obscene or gruesome. In general, an image is OK if you could show it to your grandmother without upsetting her or embarrassing yourself. A wide variety of images are available under creative commons license on the web site `www.flickr.com`.

Provide a photo credit for every image you use. That is, |em the first time you use an image in your report it must have a reference. That reference should include the name of the photographer, if it is available. Credit yourself if you took the picture. Include the source of the image, *e.g.* a URL. Use "personal collection" for your own images. If you know neither the photographer nor the source you should state that. There is no need to credit the same photo more than once. There are examples in figures 1-4 at the end of this document.

In your report, include the images you use, the numerical results of any experiments you run, and copies of any transformed images. Always include a description of what you did, and an explanation of results.

Laboratory 1: Introduction to Images and Matlab

1. Image classes.

   (a) Load a truecolor image of your choice into a matrix `I` in Matlab using `imread`. Display it and include a copy of it in your lab report. What is the image's class? What is its size? How many bands dose it have? Find its minimum and maximum values using

   ```
   >>  m = min(I(:));
   >>  M = max(I(:));
   ```

   (b) Convert the image into a 64-bit floating-point number array using the command

   ```
   >>  D = double(I);
   ```

   and check its class. Find the maximum and minimum values of `D`.

   (c) Now display `D` using

   ```
   >>  figure; image(D); truesize;
   ```

   What happens? Read the Matlab help pages on images as necessary to understand what is going on.[1] What arithmetic operation must you perform on `D` to make it displayable with `image(D)`? That is, replace `D` with an arithmetically altered version of `D` – *i.e.* $D = f(D)$ where $f()$ is some arithmetic operation like multiplication or division by a constant. Note that as you experiment with this, if you replace `D` with the wrong thing, you will have to execute `D = double(I);` to get it back. Find the maximum and minimum values of your altered `D`.

   (d) Once you are able to run `>>  figure, image(D), truesize;` and get an image that looks correct, convert `D` back to 8 bits per pixel by entering

   ```
   >>  U8 = uint8(D),
   ```

   then do

   ```
   >>  figure; image(U8); truesize;.
   ```

   What do you see? Find the maximum and minimum values of `U8`. What has the conversion done? Now do this:

   ```
   >>  figure; image(U8*255); truesize;.
   ```

   What if any colors do you see? Why would a binary image have more colors than just black and white?

   (e) In your own words, write out a short explanation of the differences between images of class `uint8` and `double`. Explain what it takes to convert between the formats without losing the intensities in the images.

[1] See "Documentation Center>Matlab >Graphics>Images." The "Examples and How To" section under "Image File Operations" is very useful, especially subsection, "8-Bit and 16-Bit" images. Also enlightening is the section, "Modifying Images>Concepts>Image Types."

2. Loading, displaying and saving color mapped (a.k.a. indexed) images.

   Download from Blackboard the files, `cwheel.bmp and cwheel_mapped.gif`. A `.gif` file (graphics interchange format) contains a color mapped image that contains no more than 256 colors.[2] It is, in fact, lossless for images with 256 or fewer distinct colors. It's not so good for continuous tone color images.

   (a) Load `cwheel_mapped.gif` and its color map into a matrix in MATLAB .

   ```
   >>  [J cmap] = imread('cwheel_mapped.gif');
   ```

   Use the `class(·)` function to determine the type of the image and use `size(·)` to determine its dimensions. Note these in your report.

   Also, note the file size in bytes of `cwheel_mapped.gif`. How does this compare to the dimensions and file size of `cwheel.bmp`?

   `imread` returns two data structures. The first one is a matrix containing the image. The second is the image's color map. Think of the color map as a palette of colors arranged in a $N \times 3$ table, where N is the number of colors. The value at a pixel in the image is not an intensity, but an index into the color map – the number of the row that contains the color vector for that pixel. How many colors are in the palette for `cwheel_mapped.gif`?

   (b) Use

   ```
   >>  figure; image(J); truesize;
   ```

   to display the image on the screen. What do you see? Describe its appearance. Repeat the commands to get the same image in a new figure window. Now use the command

   ```
   >>  colormap(cmap)
   ```

   to associate with the new figure the color map loaded along with `cwheel_mapped.gif`. Now what do you see? How does it compare to the first figure you generated in part 2a?

   (c) Create a new color map, say `cmap2`, from the old one, `cmap`, as follows:

   ```
   >>  temp = mean(cmap,2);
   >>  cmap2 = [temp temp temp];
   ```

   What did this just do? Select the figure that displays `cwheel_mapped.gif` to which you did not did not apply a color map, and apply the new color map using

   `colormap(cmap2)`.

   Now what do you see?

   (d) Now you are going to compare the 24-bit image, `cwheel.bmp`, with its color mapped version, `cwheel_mapped.gif`, displayed with its original color map – the one you loaded along with the image in part 2a.

   Describe the differences in appearance between `cwheel.bmp` and `cwheel_mapped.gif`. Based on their relative appearances, describe how gradual changes in color are implemented (or approximated) by the color mapped image.

   Like you did in Homework 1, use MATLAB 's native indexing to cut a 64×64 sub-image out of the original truecolor image. Select a region where there is a smooth change in intensity or color.

[2]See https://en.wikipedia.org/wiki/GIF for a more detailed description of the format.

Explain how you did this. Write down the coordinates you chose. Use the MATLAB function, `imresize(·)`, to enlarge the small image to 512×512. **Use the nearest neighbor mode.** Display the new image in a figure window. The individual pixels should be clearly, distinctly visible as squares. Use the `imwrite(·)` function to save the enlarged image as a `.bmp` file.

Do exactly the same thing with `cwheel_mapped.gif` using the same image coordinates, so that you get exactly the same regions out of both images. Save the enlarged region as a BMP image with the original color map from `cwheel_mapped.gif` to include in your report. If the enlarged region of the colormapped image is in array `J512`, then this will save the image with the colormap as a `.bmp` file named `cwheel_mapped_detail.bmp`:

```
>>  imwrite(J512,cmap,'cwheel_mapped_detail.bmp','bmp');
```

(e) It may appear that the color mapped image is inferior to the truecolor version. That is usually so for images that are color photographs. But if the image is a graphic with no more than the number of colors that can be displayed with an indexed image then the indexed (color mapped) version may be better, depending on the file format in which it is stored.

The following image illustrates another situation in which a color mapping can be useful. Download `plume_gs.bmp` from Blackboard, read and display it and its color map in MATLAB . This image of a smoke plume is such that the intensity (grayscale) of the image at a pixel is directly proportional to the density of the smoke at the corresponding location in space. Display the image in two other figure windows so that you have three copies of it. Apply the the color map you loaded with the image to one of figures. Apply the `jet(256)` color map to the second figure. Can you see structures in the plume that were not visible before? If so, describe them. Apply a random color map `rand(256,3)` to the third figure. Repeat `colormap(rand(256,3))` a few times until you get an image with good contrast. Again, can you see structures that were not before visible?

Save the smoke plume image with the `jet(256)` color map and save it with the random color map for inclusion in your report.

3. Arithmetic operations on images and sub-images.

(a) Acquire a 24-bit image about the same size as `cwheel.bmp` that has a wide range of intensities (dark to light). Use an image of your choice, but not `cwheel.bmp`.

(b) Assume matrix `J` contains your image. Display (i) `J/2`, (ii) `J*2`, (iii) `J-128`, and (iv) `J+128`. Describe the results. (You may if you want, but you do not have to include these four images in your report.) What happens when the image is of class `uint8`, and the result of arithmetic on a pixel would yield a negative value? What happens when the result is greater than 255?

(c) If your image has different dimensions from `cwheel.bmp`, cut out pieces of one or both images (using native MATLAB indexing) so that they are both have exactly the same dimensions.

Now read `cwheel.bmp` into variable `S` in MATLAB . Form and display the sum, `J+S`, and the product,`J.* S`. (Note the '.*'. How is that different from '*'?) Describe the results.

What is the maximum possible value of a sum of two images? How might you adjust the sum so that the maximum fits in an 8-bit image?

What is the maximum possible value of a product of two images? How might you adjust the product so that the maximum fits in an 8-bit image?

4. Looping in MATLAB

If you are not familiar with MATLAB functions, then please read the help pages for "function," in "MATLAB > Programming Scripts and Functions > Functions > Function Basics."

MATLAB provides a way to time the execution of its code, the pair of functions, `tic` and `toc`. They are placed in a function before and after the code to be timed. For example

```
J = DivIbyConstLoops(I,c)
    tic
    % code to time goes here
    toc
end
```

(a) Write a function that divides every pixel in the image by a constant `c` using loops. That is use `for` statements to make 3 nested loops. For example, have the the first loop index the bands. Inside that loop index across the columns. And inside that, loop down the rows. Any permutation of the order is OK but there need to be three loops. Note that in the function before the loops you will need to use the `size` function to get the dimensions of the image.

Place `tic` on a line before the first `for` statement. Place `toc` on a line after all three loops (the last of 3 `end` statements). Load an image and run the function on it. Note the elapsed time.

(b) Write a function that divides every pixel in the image by a constant `c` without using loops. This is particularly simple,

```
J = DivIbyConstNoLoops(I,c)
    tic
    J=uint8(double(I)/c);
    toc
end
```

(Note the type conversions.) Load an image and run the function on it. Note the elapsed time.

What do the 2 times tell you about using loops in matlab?

5. Students taking the course for graduate credit do these questions in addition to the others.

(a) Vectorization.

Please read
https://www.mathworks.com/help/matlab/matlab_prog/vectorization.html,
https://www.mathworks.com/help/matlab/ref/bsxfun.html, and
https://www.mathworks.com/help/stats/pdist.html
before doing these problems.

i. Write a function that outputs an image `J` made from input image `I` such that `J(r,c,b)` is the average of `I(r,c,b)` and the 8 pixels that surround it. Use three nested loops, over the bands, the rows, and the columns.

In the function, record the size of `I` in the three scalars `R`, `C`, and `B`. Convert `I` to double. Allocate `J` as a zero image the same size as `I`. Loop over all `B` bands. So as not to go outside of `I` when constructing `J`, loop the rows from 2 to `R-1` and the columns from 2 to `C-1`. That means the resulting `J` will have a 1-pixel wide band of zeros around it. Convert `J` to `uint8`.

Surround outside loop with `tic` and `toc` so that when you run the function it tells you the amount of time it took to execute the loops.

Load a truecolor image of your choice into MATLAB . Run your function on it, note the time and display the image. Include your code, the size of the image, and the execution time in your report. You need not include the input image nor the output image.

ii. Now here is where vectorization comes into play. This entire function can be written as one line of code – simple MATLAB code using only `[`, `]`, `/`, and indexing.[3] Write a second function that uses the one-line solution. Put tic and toc around the line in the function. Run the function on the same image as in the previous section and note the elapsed time. By what factor is the second function faster than the first? Include the numbers and your code, but not the images in your report.

Hints: (1) You will need to use `I` nine times in the single line program. (2) Note that

```
[zeros(1,C,B);I(1:R-1,:,:)]
```

slides `I` down one row, drops the last row and replaces the first one with zeros.

```
[zeros(R,1,B) I(:,1:C-1,:)]
```

slides `I` to the right one column, drops the last column and replaces the first one with zeros.

Repeat the two methods several times to get a typical execution time. Were the results what you expected? Explain why the results might be so.

(b) Write two functions that accept an $M \times N$ matrix and output an L_2 distance matrix. Assume that the input matrix contains N M-vectors (each column is a vector). The distance matrix will be $N \times N$. Element (i, j) contains the L_2 (a.k.a. Euclidean) distance between the i^{th} and j^{th} vectors in the input matrix.

The first program should be written using loops. Note that there are symmetries in the resultant matrix that can be exploited to reduce execution time.

The second program should use no loops. That can be accomplished using `bsxfun`, but there may be other ways. Consider computing the distance matrix using the expansion of the L_2 distance into an expression in terms of L_2 inner products. *I.e.* ,

$$\|\mathbf{x}_i - \mathbf{x}_j\| = \text{'?'}$$

Compare the execution times of the two on a random 10×1000 matrix using `tic` and `toc`. Compare those results to that of the MATLAB built-in `pdist()` from the statistics toolbox. Repeat all three several times to get a typical time. Were the results what you expected? Explain why the results might be so.

[3] OK it is a *long* line of code that includes 8 additions and one division by a constant.

Rules for laboratory assignments

1. Perform all the tasks listed in the instructions.
2. Explain the tasks you performed in detail.
3. Answer in writing in your report all the questions asked in the instructions.
4. Include in the report the original images you used and those resultant images that were specified in the instructions.
5. Include all computer code that you wrote and used, clearly documented, in an appendix.
6. All work must be yours and yours alone. Collaboration on the laboratory assignments is forbidden, with the following exceptions:
   (a) You may obtain help on any aspect of the homework from either Prof. Peters or the TA for this course.
   (b) You may obtain technical help on MATLAB from anyone you wish. However *you may not get direct help on the implementation of the specific algorithm* from another person except as noted in (a) above.
   (c) You may get help in obtaining the *input* images for the assignments from anyone you wish.
   (d) You may get help in the formatting or storing or transmission of your reports, but *not the content*, from anyone you wish.
7. Write your results in a classical technical paper form (as shown in the example lab) using MS Word, LaTeX, or any other word processor with which you can embed images in text. I prefer that the reports be submitted in .pdf format, but that is not required. Submit your report to me as a file on Blackboard. If for some reason this does not work, you may submit your report on a CD-ROM.
8. Name the files using the form:

   `EECE_X353_F17_Lab_1_LastName_FirstName.ext`

   where you would replace X with 4 or 5 depending on the version in which you enrolled. Replace "LastName" and "FirstName" with your actual last name and first name. Replace "ext" with pdf or doc. For example:

   `EECE_4353_F17_Lab_1_Llurbybabbin_Lamoid.pdf`
9. Assignments are due at midnight on the day specified in the instructions or in class. The grade on a laboratory report will be reduced by 10 points (out of 100) for every day (24 hours) that it is late.

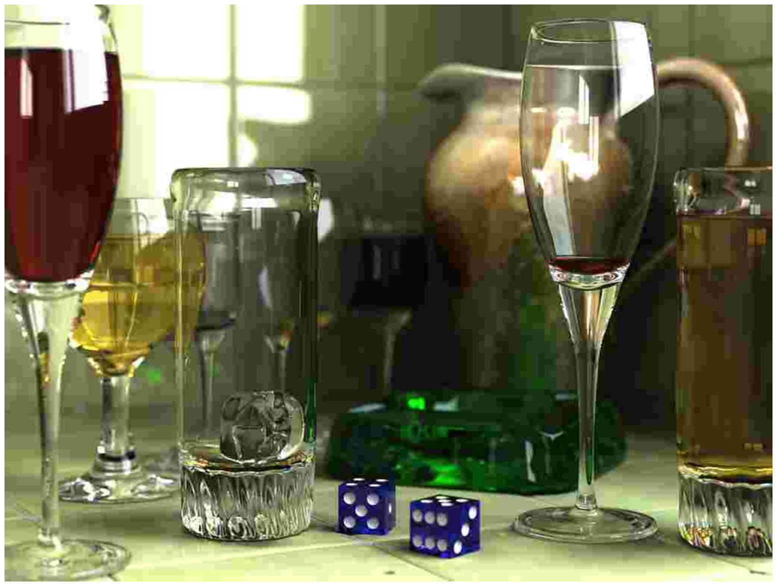

Figure 1: Gilles Tran, *Glasses*, Computer graphics model rendered with POV-Ray using ray tracing and renderosity. No post processing. Downloaded from http://www.oyonale.com/modeles.php?page=40.

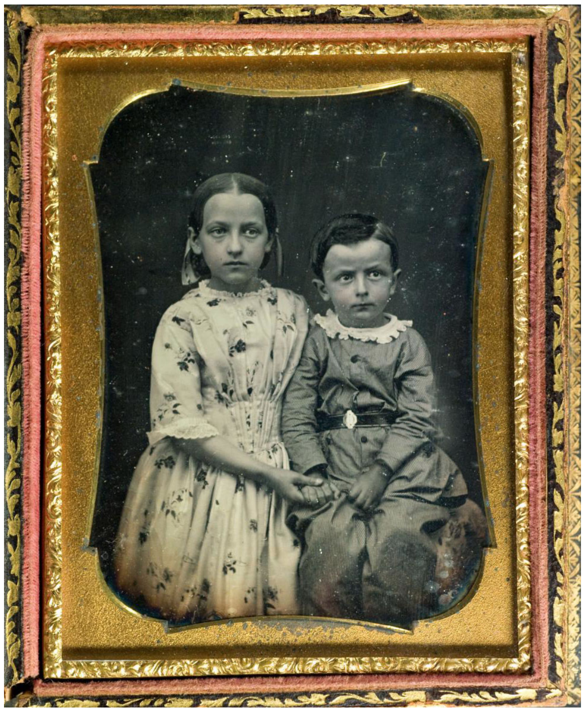

Figure 2: James Morely, *Photograph of a Daguerreotype, ca. 1850*, Original photographer and subjects unknown. Downloaded from https://flic.kr/p/dwaE6v.

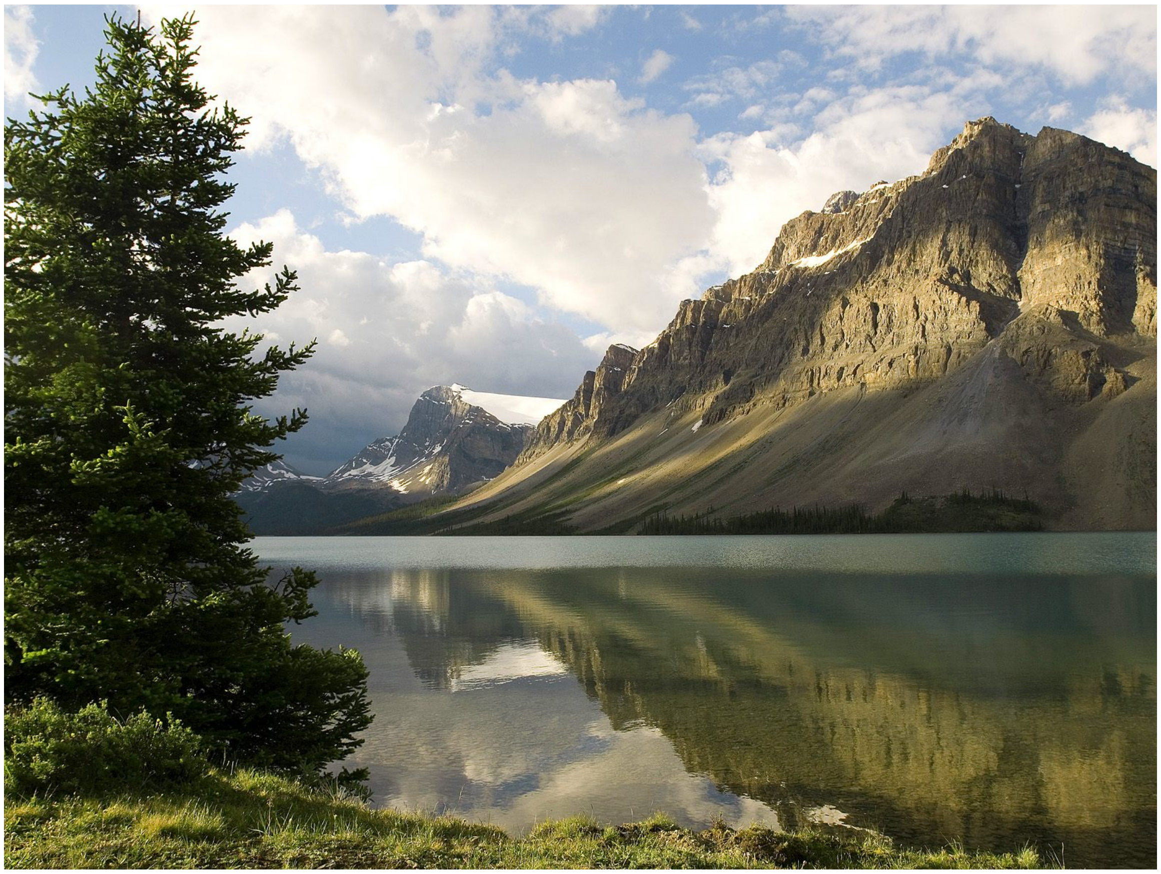

Figure 3: Photographer unknown, *Bow Lake, Banff National Park, Alberta, Canada*, `http://www.photography-match.com/wallpapers/7873_bow_lake_banff_national_park_alberta_canada_1.jpg`

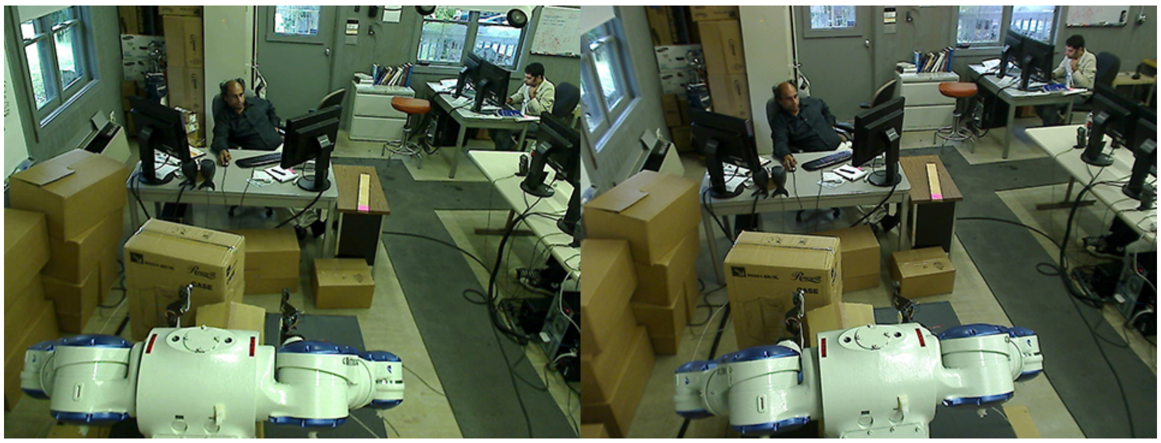

Figure 4: Alan Peters, *Ravi and Aditya at Universal Robotics*, Reverse stereo pair taken through robot's vision system, 16 Sept. 2009.

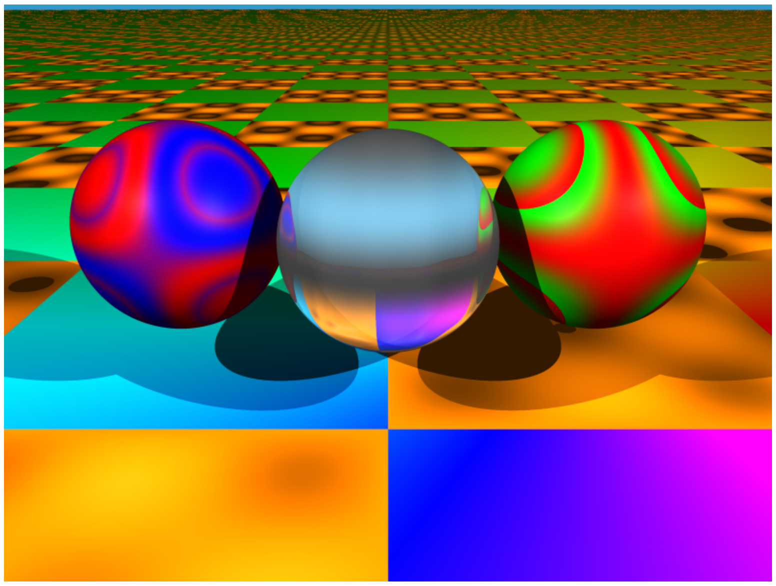

Figure 5: Photographer unknown, *cwheel.bmp*, Taken from the database, *CIPR Still Images*, Center for Image Processing Research at Rensselaer Polytechnic Institute `http://www.cipr.rpi.edu/resource/stills/graphics.html`.

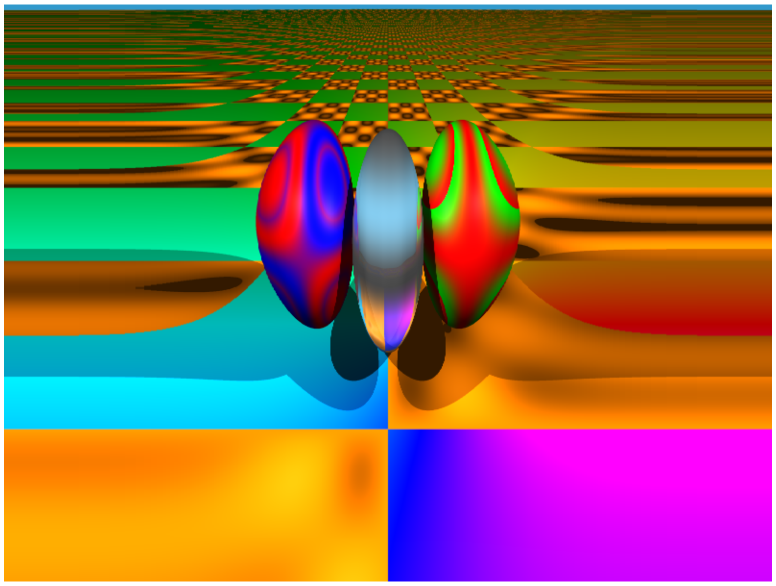

Figure 6: Example of a warped image. The warping was done by sampling the image at pixels determined by a Gaussian distribution with mean C_0 = the index of the middle column of the image and standard deviation $C_0/4$.