STONY BROOK UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CSE 391/591: COMPUTATIONAL PHOTOGRAPHY AND INTRO. TO COMPUTER VISION

Handed out: November 11,2008 **Due on:** November 25, 2008

Problem Set #3

Submit the homework by email to cs391@cs.sunysb.edu

Problem 1: *Multiscale Blending*

Setup: Download Eero Simoncelli's steerable pyramid code from http://www.cns.nyu.edu/~eero/steerpyr/. Please make sure that you have read the `readme` file and added the path ../matlabPyrTools and ../matlabPyrTools/MEX1 where is the parent directory of the toolbox.

Multiscale Blending: Download from the course web page the Burt and Adelson 1983 paper, "A Multiresolution Spline with Application to Image Mosaics". I suggest that you read the whole paper though – it is well written and will reinforce the concepts mentioned in class.

a. Use the provided `orange_peel_cropped.jpg` and `apples_honeycrisp_cropped.jpg` to generate two "new" fruits: ***orange_apple***, where the top side (top 155 rows) is orange and the bottom side is apple, and ***apple_orange***, where the top side (top 155 rows) is apple and the bottom side is orange. Use hmask1.bmp and hmask2.bmp respectively. Display the original and "new" fruits side by side.

Implement Burt and Adelson's method to blend the two images on Laplacian pyramid using the appropriately smoothed and down-sampled mask. You can imagine that a Gaussian pyramid is built for the mask and the mask at the same level as the Laplacian pyramid is used for blending.

You may use MATLAB function `imfilter` and `imresize` for filtering and downsampling. The `matlabPyrTools` contains functions for computing Gaussian and Laplacian pyramids (and reconstructing them): `buildGpyr, buildLpyr` and `reconLpyr`. Note that you should not directly manipulate the `pyr` or indices structure: use the functions `pyrBand` and `setPyrBand` to set/get the sub-bands of the pyramid.

b. Blend the two apple logo images provided and use the `logo_mask.bmp`

c. Implement the algorithm that splines regions of arbitrary shape together, as described in section 3.2 of the paper. (page 230 of the paper.) Pick two images containing objects that you want to blend together. Extra credit will be given for the artistic/horror/shock value of the final image, so choose something fun! Then generate the mask image to select the blending boundary. The `roipoly` command may be useful. (For inspiration check last few slides of lecture 16.)

Deliverables:

- Code for multiscale blending
- Display your results on your webpage. Display the original and resulting images side by side.

(Let's keep the results a surprise. Create a directory with an unusual name that your classmates will not be able to guess.)

Problem 2: ***Image Mosaics and Panoramas***

In this problem we will guide you to build panorama by yourself. You are not allowed to use any other software for this problem set. We ask you to generate intermediate results at each step. You are also NOT allowed to use MATLAB functions such as cp2tform, tformarray, tformfwd, tforminv, maketform or imtransform. We want you to write your own code.

Step 1. Take Pictures

Take at least four photographs of the Stony Brook Campus for each panaroma that you will generate. The transform between the photographs should be projective (a.k.a. perspective). One way to do this is to shoot from the same point of view but with different view directions, and with overlapping fields of view. Another way to do this is to shoot pictures of a planar surface (e.g. a wall) or a very far away scene (i.e. plane at infinity) from different points of view.

The easiest way to acquire pictures is using a digital camera. Make sure to use the highest resolution setting (important for homography calculation; you can always downsample it later). Matlab's `imread` can take most popular image formats; use unix convert for the more obscure ones.

While we expect you to acquire most of the data yourself, you are free to supplement it with other sources (old photographs, scanned images, architectural drawings, the Internet). Be creative! We're not particular about how you take your pictures or get them into the computer, but we recommend:

- Make sure images overlap by ~40%.
- Keep the same viewpoint. -- If you're shooting a non-planar scene, then shoot pictures from the same position (turn camera, but don't translate it). A tripod can help in this, particularly if objects are close.
- See with distinct features (not repetitive).
- Use identical aperture & exposure settings, if possible. On most "idiot cameras" you don't have control of this, unfortunately. It's nice to use identical exposures so that the images will have identical brightness in the overlap region.
- Avoid lenses with too much distortion (fish eye and other lenses that do not preserve straight lines). Avoid the wide-angle range of zooms because they tend to have such distortion.
- Down-sample the images to be smaller than 800x600 before processing. Debug with even lower resolution, e.g. 400x300.
- No moving objects in the scene for the first trial. You may shoot moving objects and try to display them properly later for extra credits.
- Shoot as close together in time as possible, so lighting doesn't change too much (unless you want this effect for artistic reasons) and your subjects don't move on you.

Good scenes are: building interiors with lots of detail, inside a canyon or forest, tall waterfalls, panoramas. The mosaic can extend horizontally, vertically, or can tile a sphere. You might want to shoot several such image sets and choose the best. Shoot and digitize your pictures early - leave time to re-shoot in case they don't come out! Print and lay out your photos on a table to see approximately what the mosaic will look like.

Deliverable: Display input photos.

Step 2. Mark correspondences

While it is possible to create a panorama generation program that is entirely automatic, using feature detectors and descriptors to find correspondences, in order to simplify the problem, we will skip that step and allow you to mark correspondences by hand. Establishing point correspondences is a tricky business. An error of a couple of pixels can produce huge changes in the recovered homography. The typical way of providing point matches is with a mouse-clicking interface. You can write your own using the bare-bones `ginput` function. Or you can use `cpselect` (it's a bit flaky). After defining the correspondences by hand, it's often useful to fine-tune them automatically. This can be done by SSD or normalized-correlation matching of the patches surrounding the clicked points in the two images (see `cpcorr`), although sometimes it can produce undesirable results.

Select the center picture as the reference, and mark correspondence between each image and the reference. If it is impossible to mark correspondence to the reference, mark to the nearest and then you will propagate the correspondence in a later step. For this case you may want to build a tree structure of the images, where the reference is the root and each picture can correspond to the reference picture by tracking the tree to the root. To mark correspondence between a pair of images, you can use your favorite tool. One suggestion is to use MATLAB function subplot to display the two images

side by side, and ginput to click the feature points. You may click one feature point at image 1, and then click the corresponding feature point at image 2, and so on. At the returning matrix from ginput, the odd rows will be the feature points at image 1 and the even rows be the corresponding feature points at image 2.

Deliverable: Plot correspondences on top of your photos. Put a red bounding box around the reference frame.

Step 3. Recover Homographies
Before you can warp your images into alignment, you need to recover the parameters of the transformation between each pair of images. In our case, the transformation is a homography: **p'=Hp**, where **H** is a 3x3 matrix with 8 degrees of freedom (lower right corner is a scaling factor and can be set to 1). One way to recover the homography is via a set of `(p',p)` pairs of corresponding points taken from the two images . You will need to write a function of the form:

```
H = computeH(im1_pts,im2_pts)
```

where `im1_pts` and `im2_pts` are n-by-2 matrices holding the `(x,y)` locations of n point correspondences from the two images and H is the recovered 3x3 homography matrix. In order to compute the entries in the matrix H, you will need to set up a linear system of n equations (i.e. a matrix equation of the form **Ah=b** where h is a vector holding the 8 unknown entries of **H**). If `n=4`, the system can be solved using a standard technique. However, with only four points, the homography recovery will be very unstable and prone to noise. Therefore more than 4 correspondences should be provided producing an overdetermined system which should be solved using least-squares. In Matlab, both operations can be performed using the "\" operator (see help mldivide for details).

Deliverable: Display the 3x3 homography transform matrix for each pair of images. Both transforms and inverse transforms are needed for generating panorama.

Step 4. Warp the Images
Now that you know the parameters of the homography, you need to warp your images using this homography. Write a function of the form:

```
imwarped = warpImage(im,H)
```

where `im` is the input image to be warped and **H** is the homography. You can use either forward of inverse warping (but remember that for inverse warping you will need to compute **H** in the right "direction"). You will need to avoid aliasing when resampling the image. Consider using `interp2`, and see if you can write the whole function without any loops, Matlab-style. One thing you need to pay attention to is the size of the resulting image (you can predict the bounding box by piping the four corners of the image through **H**, or use extra input parameters). Also pay attention to how you mark pixels which don't have any values. Consider using an alpha mask (or alpha channel) here.

Deliverable: Display the masks and the warped frames.

Step 5. Blend the images into a mosaic
You now have n images at the final resolution and n corresponding binary masks. You need to blend them together.

To compose the final panorama we need to generate the coordinate system for the big panorama picture that can embrace all the images warped to the reference frame. You may warp the corners of each picture to the coordinate of the reference frame, and then choose the minimum and maximum of the corners to determine the frame of the panorama. Use MATLAB function `line` to display the frame of each picture in the panorama in blue, and the reference frame in red.

First, implement a naïve solution where you use the binary mask to blend the pictures in an arbitrary order (e.g. from left to right). Start from an empty image and for each warped picture, overwrite the pixels where the mask is 1. You will probably have artifacts at the boundary, but it will allow you to verify that the images are actually aligned.

Next, instead of having one picture overwrite the other, which would lead to strong edge artifacts, use weighted averaging. You can leave one image unwarped and warp the other image(s) into its projection, or you can warp all images into a new projection. Likewise, you can either warp all the images at once in one shot, or add them one by one, slowly growing your mosaic.

If you choose the one-shot procedure, you should probably first determine the size of your final mosaic and then warp all your images into that size. That way you will have a stack of images together defining the mosaic. Now you need to blend them together to produce a single image. If you used an alpha channel, you can apply simple feathering (weighted

averaging) at every pixel. Setting alpha for each image takes some thought. One suggestion is to set it to 1 at the center of each (unwarped) image and make it fall off linearly until it hits 0 at the edges (or use the distance transform bwdist). However, this can produce some strange wedge-like artifacts. You can try minimizing these by using a more sophisticated blending technique, such as a Laplacian pyramid from problem 1. If your only problem is "ghosting" of high-frequency terms, then a 2-level pyramid should be enough. Of course, if your pictures align perfectly, then you don't need any blending at all, but that rarely happens in practice.

If your mosaic spans more than 180 degrees, you'll need to break it into pieces, or else use non-projective mappings, e.g. spherical or cylindrical projection.

Deliverable: Show naïve solution and final solution.

Deliverables:

- You will need to submit all your code.
- Create web page displaying a complete mosaics (see lecture 13 for inspiration). Show all the intermediate results associated with easch step.

Extra Credit:

Blending and Compositing*:* Use homographies to combine images (or images and video) in interesting and creative ways. For inspiration, check the last few slides of lecture 13. Here are a few suggestions:

- Put fake graffiti on buildings or chalk drawings on the ground
- Replace a road sign with your family portrait
- Project a movie onto a building wall
- Create a mosaic by spatially blending images taken at different times (day vs. night) or during different seasons
- Create a mosaic by spatially blending a historic photograph with a modern picture of the same place
- Create an interesting/bizarre mosaic, like the ones with multiple copies of the sample person…etc.
- Other suggestions?

Spherical/Cylindrical/polar mapping: Instead of projecting your mosaic onto a plane, try using another surface, such as a sphere or a cylinder. This is often a better way to represent really wide mosaics. Be clever: do the inverse sampling from the original pre-warped images to make your mosaic the best possible resolution. Pick the focal length (radius) that looks good.

Use 3D rotational model: If your mosaic is a rotation about the same point and you don't change zoom, you can use a simpler rotation-only transformation, which is more robust and requires less correspondences. This approach should also in theory help you find the focal length of your camera.

360 Cylindrical panorama: Instead of a planar-projection mosaic, do a cylindrical projection instead. Perform a cylindrical warp on all your input images and stitch them together using translation only. This is one way to produce a full 360 degree panorama (you can use a nifty Live Picture Viewer to display your panorama!). The down side is that this method places more requirements on your camera (you need to know the focal length and radial distortion coefficients), and your data (the images have to be exactly horizontal – use a tripod).

Video mosaics: Capture two (or more) stationary videos (either from the same point, or of a planar/far-away scene). Compute homography and produce a video mosaic. You will need to worry about video synchronization (not too hard – a single parameter search). Also make sure that you shoot something where things are happening over short periods of time – video data gets really big really quickly. A good example would be capturing a football game from the sides of the stadium.