Computational Photography
Project 5, Fall 2014

November 13, 2014

This project is about essential methods and applications of image warping. Submit material as described in the *turn-in instructions* for each part on ILIAS. The deadline is November 27, 14:00.

1. Morph Tool (4 points)

In this assignment, you will implement a morph animation from one image to another.

Through a user interface, the user can specify corresponding feature points on the two images. The feature points are then triangulated to form a cover of triangles. To compute individual frames of the animation, the corresponding feature points are linearly interpolated. The content of the triangles are related by affine transformation.

The affine transformation between two triangles $t_a(x_{ai}, y_{ai})$ and $t_b(x_{bi}, y_{bi})$ for $i = 1, 2, 3$ is given by:

$$T_{ab} = \begin{pmatrix} x_{b1} & x_{b2} & x_{b3} \\ y_{b1} & y_{b2} & y_{b3} \\ 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} x_{a1} & x_{a2} & x_{a3} \\ y_{a1} & y_{a2} & y_{a3} \\ 1 & 1 & 1 \end{pmatrix}^{-1}$$

Here is the algorithm in detail:

- Load two (preferably equal sized) images as source and target of the morph.
- Using `cpselect`, specify the corresponding points on the two images. We recommend you to save and load the points for extended editing.
- Calculate the Delaunay triangulation on one of the two point sets using `delaunay`. You can display the triangulation using `triplot`.
- For every interpolated frame: calculate interpolation parameters (for position and color) and interpolate the positions of all Delaunay triangles.
- For every triangle being rasterized:
  - Calculate its bounding box.

  - Find the two affine transformations from the triangle being rasterized to the corresponding triangles in the source and target frame using above equation.
  - For every pixel within the bounding box, perform an inside-triangle test. You should use the Matlab file `rasterize.m` that we provide on ILIAS to do this. Perform the affine transformation to the source and target frame for every pixel inside the triangle, and look up the corresponding pixel values using bilinear interpolation. Then, blend the two pixel values using the color interpolation parameter.

- Create a movie by converting images to movie frames using `im2frame`. You can then play back the frame sequence with `movie`. To save a movie use `avifile`, `addframe`, and `close`, or save an image sequence using `imwrite`.

Bilinear Interpolation

Implement bilinear interpolation as a separate function, as it will also be used in the following assignments.

Consider $c_{00}, c_{10}, c_{01}, c_{11}$ to be four neighboring pixels surrounding a position $p(x, y)$. If (u, v) is the fractional part of (x, y), then the bilinear interpolated color c at position p is given by:

$$\begin{aligned} c_0 &= (1-u)\, c_{00} + u\, c_{10} \\ c_1 &= (1-u)\, c_{01} + u\, c_{11} \\ c &= (1-v)\, c_0 + v\, c_1 \end{aligned}$$

Avoiding For-Loops over Pixels Completely (optional)

Try to speed up your algorithm by avoiding for-loops over pixels. Color look-up using nearest neighbor or bilinear interpolation and color blending can be done without a loop! Study `rasterize.m` and the excercise class slides for more hints.

Parallel For-Loop (optional)

Depending on the size of the images and how many frames you create, calculating the morph in Matlab can take a lot of time. This is due to the slow execution of for loops in Matlab. As a remedy, the parallel for-loop statement `parfor` parallelizes expensive for-loops by exploiting multiple CPU cores.

To enable parallel for loops, you need to call `matlabpool` once per session. We recommend you to add the statement to `startup.m`.

The use of the `parfor` command imposes some constraints which may require you to refactor the code. Firstly, it can only be used on the outer most loop. Secondly, write access to matrices using the index of the `parfor` loop must be simplified. Specifically, no index calculation is allowed, and in most cases, the matrix must be preallocated, e.g., using `zeros`.

Non-linear interpolation parameter (optional)

When calculating the interpolation parameter use a cosine ramp instead of a linear ramp for plausible acceleration and deceleration. Try using different interpolation parameters for the vertex motion and the color blending. For example, what happens when you time-reverse the ramp for color blending?

Turn-in: Matlab code and a few interpolated images or a video. You can use the freeware tools SequImago (Mac) or VirtualDub (Windows) to assemble an image sequence into a movie.

2. Rectification using Homography (2 points)

In architecture photography, the problem often arises that a building cannot be taken head on, due to the size and location of the building. The consequence is that pictures of such buildings have edges converging towards the sky.

The goal is to rectify the picture such that the vertical edges of the building become parallel. To solve this problem, we need to transform a quadriliteral, i.e., four points rather than three. While three pairs of points define an affine transformation, four pairs of points define a *homography* (up to a scalar factor).

We can use homography as a tool for rectification if we define the first set of four points as the user specified quadriliteral which ought to be rectified, and the other point set as a rectangle.

To find a homography between four corresponding points p_i, q_i for $i = 1, 2, 3, 4$, we need to solve the following equations:

$$q_i = H\, p_i \quad \text{with} \quad H = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix}$$

It can be shown that solving for the homography is equivalent to solving the following linear system:

$$Ah = 0 \quad \text{with} \quad A = \begin{pmatrix} a_{x1} \\ a_{y1} \\ \vdots \\ a_{xN} \\ a_{yN} \end{pmatrix}$$

and

$$\begin{aligned} h &= (h_{11}, h_{12}, h_{13}, h_{21}, h_{22}, h_{23}, h_{31}, h_{32}, h_{33})^T \\ a_x &= (-x_1, -y_1, -1, 0, 0, 0, x_2x_1, x_2y_1, x_2) \\ a_y &= (0, 0, 0, -x_1, -y_1, -1, y_2x_1, y_2y_1, y_2) \end{aligned}$$

where (x_1, y_1) and (x_2, y_2) are pairs of corresponding points.

This linear system can be solved using `svd`. SVD decomposes the matrix as $A = USV$. The homography vector is then the last column vector of V ($h = V(:, 9)$). Since the homography is defined up to a scalar value, the vector must be homogenized after transformation:

$$\begin{aligned} p &= Hq \\ p' &= \frac{p}{p_z} = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \end{aligned}$$

With this background, you can implement the rectification:

- Load and display an image and choose four points using `ginput` and associate them with the corresponding points of a rectangle, e.g., bounding box of the image.
- Find the homography for the four corresponding points using the method described above. Write a separate function to compute the homography as it will also be used in the third assignment.
- Rasterize the new image using the inverse homography and bilinear interpolation.
- Try to avoid for-loops over pixels (Optional).

Turn-in: Matlab code, a pdf file containing the pictures before and after rectification. Pick an interesting example, e.g., rectify a distorted building or project an image onto a planar surface.

3. Panorama Stitching (4 points)

In this assignment, you will stitch two images together to form a single panorama image. For simplicity, we assume a planar projection instead of a spherical or cylindrical projection. We also assume that each picture was shot from the same position, with the same focal length and zoom. If we also ignore lens distortions, then the pictures are related by a homography, more precisely by a rotation followed by a projection.

The homography can be estimated if we can find four corresponding points between the pictures. The first step is to find feature points in both images using *scale-invariant feature transform* (SIFT). The next step is then to find matches between the two point sets, based on the Euclidian distance of the feature vectors of the points. The resulting pairs of corresponding points serve as good candidates for estimating the homography.

Since the correspondences are only based on local feature criteria, some correspondences may be wrong. To handle such outliers, we use RANSAC which is a particularly stable fitting algorithm.

Once a homography relating the two images is found, we can use it to transform one image into the image plane of the other and render a unified image.

Here are the details for stitching paranoramas:

- Download and install VLFeat library. Follow the installation instructions by performing the necessary changes to `startup.m`. Restart Matlab and verify that VLFeat has been added to the path. It is also advisable to have a look at the SIFT tutorial.
- Load two images from a panorama set. The overlapping region between the two images should be sufficiently large (about 50%).
- Retrieve SIFT features from both images using `vl_sift`. You can use `vl_plotframe` to display the features.
- Retrieve the corresponding feature points using `vl_ubcmatch`.
- Perform RANSAC to find the best points to compute a homography.
  - Randomly choose four pairs of corresponding points.
  - Calculate the homography for these points.
  - Transform points from one image to the other using the homography.
  - Count inliers based on user specified minimal distance.
  - Keep homography with maximum number of inliers.
  - Loop until a sufficient number of iteration is reached.
- Find the best homography by recalculating a least squares homography using all inliers from the previous homography. This step eliminates a bias toward a specific set of points to be used as homography. However, you may not be able to notice a big visual difference.
- Rasterize a single panorama image using the found homography. Compute the bounding boxes to calculate the new image size. As before, perform rasterization using bilinear interpolation and use an appropriate blending function.
- Finally, manually crop the image using `ginput`.

Turn-in: Matlab code, a pdf file containing your own example pictures (source and stitched panorama).

Bonus (2 points)

This project includes two bonus assignements. We will reward solving them with one point each.

1. Write a cylindrical panorama stitcher accroding to the explanations in the lecture. Demonstrate it using at least one example with your own input images. We have a tripod that you can borrow if needed.
2. Implement a resampling method using Gaussian filters according to the lecture slides. Use it instead of bilinear interpolation in at least one of the previous assignments. Demonstrate the antialiasing capabilities with an appropriate example such as a checkerboard image. Please indicate where you use this method in your code by adding a comment.