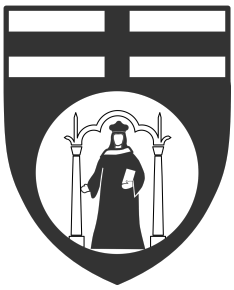

Università degli Studi di Genova

Computer Vision

Assignment 1: Image warping and bilinear interpolation

Giovanni Battista Borrè, Tommaso Gruppi, Federico Tomat

April 1, 2019

Contents

1 Introduction

1.1 Forward Warping

The Forward warping algorithm is used to transform an image $f(x)$ into an image $g(x')$ through the parametric transformation $x' = h(x)$. The following limitations must be considered. The process of copying a pixel $f(x)$ to a location x' in g is not well defined when x' has a non-integer value: in this case the value of x' can be rounded to the nearest integer coordinate and the pixel can be copied there, but the resulting image has severe aliasing and pixels that jump around a lot when animating the transformation. You can also "distribute" the value among its four nearest neighbors in a weighted (bilinear) fashion, keeping track of the per-pixel weights and normalizing at the end. This technique is called *splatting* and is sometimes used for volume rendering in the graphics community, even if it suffers from both moderate amounts of aliasing and a fair amount of blur (loss of high-resolution detail). The second major problem with forward warping is the appearance of cracks and holes, especially when magnifying an image. Filling such holes with their nearby neighbors can lead to further aliasing and blurring, so a preferable solution is to use Backward warping.[1, 2]

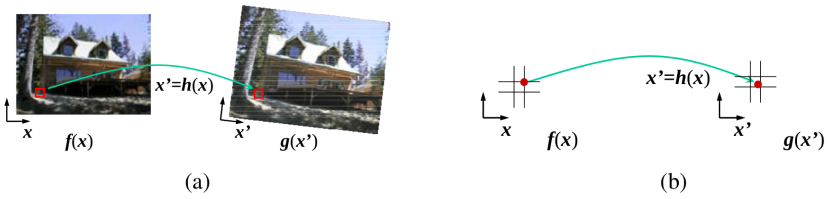

Figure 1: Graphical representation of image warping algorithm

1.2 Backward Warping

In the Backward Warping each pixel in the destination image $g(x')$ is sampled from the original image $f(x)$ and since $h(x')$ is (presumably) defined for all pixels in $g(x')$, we no longer have holes, but the most important feature of this algorithm is that the resampling of an image at non-integer locations is a well-studied problem and high-quality filters that control aliasing can be used.

The function $\hat{h}(x)$ can simply be computed as the inverse of $h(x)$; in fact, all of the parametric transforms have closed form solutions for the inverse transform. In other cases, it is preferable to formulate the problem of image warping as that of resampling a source image $f(x)$ given a mapping $x = \hat{h}(x')$ from destination pixels x' to source pixels x.

For example, in optical flow, we estimate the flow field as the location of the source pixel which produced the current pixel whose flow is being estimated, as opposed to computing the destination pixel to which it is going. Similarly, when correcting for radial distortion, we calibrate the lens by computing for each pixel in the final (undistorted) image the corresponding pixel location in the original (distorted) image.

For the resampling process are suitable any kinds of interpolation filters can be used, including *nearest neighbor, bilinear, bicubic*, and *windowed sinc* functions. While *bilinear* is often used for speed (e.g., inside the inner loop of a patch-tracking algorithm), *bicubic*, and *windowed sinc* are preferable where visual quality is important.

Unfortunately, for a general (non-zoom) image transformation, the resampling rate is not well defined.[1, 2]

Figure 2: Graphical representation of image backward warping algorithm

2 Matlab Resolution

The passages that we have used for image processing are :

- RGB Channel Separation, in order to manage the transformations individually
- Backward Warping , a processing technique that starts from the final domain in order to obtain the value in the original domain.
- Final Interpolation, that permits an approximation of the value after transformation.
- RGB Channel Merge, to reconstruct the matrix.

With our code it is possible to perform, starting from the original image, a sequence of different transformations like translation, rotation and horizontal and vertical shear. Please note that we are able to perform just affine transformations, because we decide to use *affine2d*[3] object in Matlab®.
This object has some useful functions build in. More precisely we use *transformPointsInverse*[4] in order to apply the inverse selected trasformation to the point obtained via *meshgrid* [5] function.
Generally, before performing the backward warping we operate a rigid translation of the center of the image, that is by default on the high left corner of the image. Finally using the *griddata*[6] function we are able to interpolate the surface at the desired points, specified as parameters of the function, and return the interpolated values outputs. Then we use the *cat*[7] function to merge the matrices relative to the three different color channels and then cast them to *uint8* to display them.

2.1 Translation

$$[transfImg, RGB] = translation(inputImage,\ xTr,\ yTr) \tag{1}$$

The function has input the image we want to transform and the translations that we want to apply respectively on the x and the y axis are input as parameters. The function constructs the corresponding matrix T as the following:

$$T = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ xTr & yTr & 0 \end{pmatrix}$$

That will be encapsulated in the *affine2d*[3] object and used to apply that transformation to the image through the function *applytrf*. The result could be eventually printed via *PrintRGB* function.

2.2 Rotation

$$[transfImg, RGB] = rotation(inputImage,\ theta) \tag{2}$$

This function works as previous one, the only difference in the construction of transformation matrix, that is a rotation matrix around z axis with amplitude theta. For doing that it creates the matrix T:

$$T = \begin{pmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

that performs the transformation through the function *applytrf* and prints the result through the function.
Please note that rotation matrix is such defined because of the defauld axis orientation on Matlab *PrintRGB*.

2.3 Shear

$$[transfImg, RGB] = shear(inputImg,\ xSh,\ ySh) \tag{3}$$

Also in this case the function constructs the matrix T corresponding to the vertical and the horizontal shear factor that we want to apply to the image(xSh and ySh). Trasformation matrix is such define:

$$T = \begin{pmatrix} 1 & ySh & 0 \\ xSh & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

After that it applies the corresponding transformation again through the function *applytrf* and plots it through *PrintRGB* function.

2.4 Scaling

$$[transfImg, RGB] = scale(inputImg,\ xSc,\ ySc), \tag{4}$$

This function is used to perform a zoom in and zoom out on the image of a factor xSc for x axis and ySc for y axis using the following transformation matrix.

$$T = \begin{pmatrix} xSc & 0 & 0 \\ 0 & ySc & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

As before we pass the transformation matrix T as input to *applytrf* in order to obtain the trasformed image.

2.5 Applytfr

$$RGB = applytrf(Img,\ T) \tag{5}$$

This function is used to perform the transformations to the all three channel of input image that is passed as the first parameter to the function. For doing that, as stated before, we used the Matlab®object *affine2d*[3] in order to use *transformPointsInverse*[4] that applies the inverse transformation to output point to recover the original coordinates. Before applying the transformation we have traslated the center of the axis in the middle of the image and after the transformation it was shifted again in the original position. Then we perform a bilinear interpolation with *griddata*[6]. Finally the RGB image is reconstructed using *cat*[7].

2.6 PrintRGB

$$PrintRGB(Img,\ RGB,\ String). \tag{6}$$

This function it is used to display the colored image and its three channels R, G and B in unique subplot.

3 Result

The purpose of this project is to show different image transformation with backward warping algorithm. Please note that more transformations could be applied in series, in order to obtain more complex transformation.

Figure 3: Comparison between the original image and the modified ones

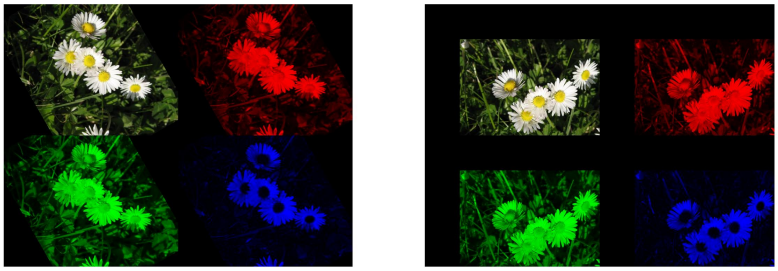

Figure 4: Rotation and translation for all channel of the image

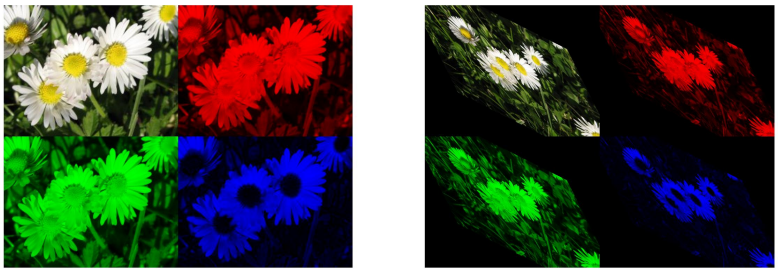

Figure 5: Scale and shear for all channel of the image

For the images transformation we used the following parameter:

Variable	Mean	Unit of measure
xTr	100	pixels
yTr	100	pixels
xSh	0.5	/
ySh	0.5	/
xSc	2	/
ySc	2	/
theta	pi/3	rad

Table 1: Parameters

4 GitHub Repository

https://github.com/federicotomat/Computer-Vision

Bibliography

[1] Richard Szeliski, *Computer Vision: Algorithms and Applications*, (University of Washington, 2010).

[2] Francesca Odone & Fabio Solari, *Computer Vision Course: Image Foundamental*, (University of Study of Genoa, 2019).

[3] Mathworks documentation on affine2d

[4] Mathworks documentation on transformPointsInverse

[5] Mathworks documentation on meshgrid

[6] Mathworks documentation on griddata

[7] Mathworks documentation on cat