Subject: Re: Image warping in IDL
Posted by David Fanning on Wed, 08 Nov 2006 14:01:29 GMT
View Forum Message <> Reply to Message

Wox writes:

> I have a question concerning image warping: "Is there a fast way of
> doing forward mapping in IDL?"

A "question"!? A treatise might be closer to the mark.

I'm pretty sure you already know more about this problem
than most of us. If you are looking for a (fast) IDL solution,
I think you are doomed. You might have a chance at writing
such a thing in C and linking it to IDL, but I'll have
to study the question for a few more days to understand
just what you want here. :-)

Cheers,

David
--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Sepore ma de ni thui. ("Perhaps thou speakest truth.")

Subject: Re: Image warping in IDL
Posted by Wox on Wed, 08 Nov 2006 14:50:58 GMT
View Forum Message <> Reply to Message

On Wed, 8 Nov 2006 07:01:29 -0700, David Fanning <news@dfanning.com>
wrote:

> ...but I'll have
> to study the question for a few more days to understand
> just what you want here. :-)

The problem is not straitforward. Let me try again:

1. I have two arrays with the same dimensions:
- input image with pixels [Xi,Yi]: this contains my image
- output image with pixels [Xo,Yo]: this "will" contain the warped
image

2. There are two ways of warping:
a. Inverse warping: You have two surfaces (Xo,Yo)->Xi and (Xo,Yo)->Yi

where (Xo,Yo) are irregular and non-integer. First these two surfaces are evaluated for the pixels of the output image, i.e. regular-integer (Xo,Yo). Now we know where each output pixel is located in the input image (hence the name "inverse" warping). These locations are non-integer, so we have to INTERPOLATE (bilinear, cubic, whatever...).

=> gridding + interpolation

b. Forward warping: You have two surfaces (Xi,Yi)->Xo and (Xi,Yi)->Yo where (Xi,Yi) are irregular and non-integer. First these two surfaces are evaluated for the pixels of the input image, i.e. regular-integer (Xi,Yi). Now we know where each input pixel is located in the output image (hence the name "forward" warping). These locations are non-integer, so we have to RESAMPLE.

=> gridding + resampling

Forward warping is slower because the resampling, as I implemented it, loops over all colums and row (and the for each pixel in the row/column). I use forward warping, because I only have the surfaces (Xi,Yi)->Xo and (Xi,Yi)->Yo (actually I have the coefficients of two 2D splines).

Now the question again:
1. Can I make the resampling faster in IDL (avoid the looping)?
2. Can I avoid the resampling completely by somehow converting the 2 spline surfaces to (Xo,Yo)->Xi and (Xo,Yo)->Yi, so I can use inverse warping.

Subject: Re: Image warping in IDL
Posted by JD Smith on Wed, 08 Nov 2006 17:07:40 GMT
View Forum Message <> Reply to Message

On Wed, 08 Nov 2006 15:50:58 +0100, Wox wrote:

> On Wed, 8 Nov 2006 07:01:29 -0700, David Fanning <news@dfanning.com>
> wrote:
>
>> ...but I'll have
>> to study the question for a few more days to understand
>> just what you want here. :-)
>
> The problem is not straitforward. Let me try again:
>

> 1. I have two arrays with the same dimensions:
> - input image with pixels [Xi,Yi]: this contains my image
> - output image with pixels [Xo,Yo]: this "will" contain the warped
> image
>
> 2. There are two ways of warping:
> a. Inverse warping: You have two surfaces (Xo,Yo)->Xi and (Xo,Yo)->Yi
> where (Xo,Yo) are irregular and non-integer. First these two surfaces
> are evaluated for the pixels of the output image, i.e. regular-integer
> (Xo,Yo). Now we know where each output pixel is located in the input
> image (hence the name "inverse" warping). These locations are
> non-integer, so we have to INTERPOLATE (bilinear, cubic, whatever...).
>
> => gridding + interpolation
>
> b. Forward warping: You have two surfaces (Xi,Yi)->Xo and (Xi,Yi)->Yo
> where (Xi,Yi) are irregular and non-integer. First these two surfaces
> are evaluated for the pixels of the input image, i.e. regular-integer
> (Xi,Yi). Now we know where each input pixel is located in the output
> image (hence the name "forward" warping). These locations are
> non-integer, so we have to RESAMPLE.
>
> => gridding + resampling

I don't see how forward and reverse mapping in this context are any different from each other. You have two images, with an irregular grid of matching "anchor" points mapping between them (I'm visualizing warping two eyes to two other positions in an image). To map *all* points of the input image to the output image, you triangulate (TRIANGULATE) that irregular grid (eyes, nose, ears, etc.), lay the triangulation down on the output surface, and use interpolation (TRIGRID) to figure out how points get mapped from input->output in the space between anchor points. Then INTERPOLATE actually samples the input image at those traingulated output positions. This simply describes the steps WARP_TRI does for you (you could also do them yourself easily).

This operation is symmetric under interchange of the sets of anchor points in the input and output images. Simply swap them, and that's you're reverse map. Certainly you don't need to loop over pixels yourself. Perhaps you were looking for a convenient way to calculate both forward and reverse maps at once?

JD

Subject: Re: Image warping in IDL

Posted by Robbie on Thu, 09 Nov 2006 05:49:19 GMT
View Forum Message <> Reply to Message

I don't really understand the problem fully, but I'm sure that INTERPOLATE does all the hard work for you.

I suspect that allocating a very large mapping array might result in an overhead in memory allocation. If there is a sensible way to split up your mapping function into logical chuncks then you could optimise that way.

For example, I do a rotation of a 3D object in the X-Y plane. For very large images it is more efficent to consider each plane independantly and run INTERPOLATE on each of those slices (a.k.a ROT).

Robbie

http://www.barnett.id.au/idl/

Subject: Re: Image warping in IDL
Posted by Wox on Thu, 09 Nov 2006 10:53:52 GMT
View Forum Message <> Reply to Message

On Wed, 08 Nov 2006 10:07:40 -0700, JD Smith <jdsmith@as.arizona.edu> wrote:
> I don't see how forward and reverse mapping in this context are any
> different from each other.

The approach is different. And yes, if you just had the tie points in input and output image, you could choose between forward and reverse mapping.

But as stated after the description of forward and reverse mapping: "I only have the surfaces (Xi,Yi)->Xo and (Xi,Yi)->Yo". I have these surfaces as 2D splines, I don't have the anchor points.

So I can't just "swap the anchor points", because I don't have them, I only have the coefficients of two 2D splines.

The thing is, how to "swap these surfaces", if you know what I mean.

Off course, one could define some arbitrarly tie points, evaluate the spline for them and then "swap the anchor points" to do reverse mapping. But how to define these tiepoints?

Subject: Re: Image warping in IDL
Posted by Wox on Thu, 09 Nov 2006 12:37:21 GMT
View Forum Message <> Reply to Message

On 8 Nov 2006 21:49:19 -0800, "Robbie" <retsil@iinet.net.au> wrote:

> I don't really understand the problem fully, but I'm sure that
> INTERPOLATE does all the hard work for you.

That would be for reverse mapping. In forward mapping, it's the resampling (the for-loops in the original post) that takes processing time. There isn't really a memory problem here.

So the problem in short:
1. How to get ride of the looping in the resampling step
OR
2. How to prevent having to do the resampling in the first place
(going to reverse mapping by having a "magical operation" converting the 2D spline coefficients)

Subject: Re: Image warping in IDL
Posted by Jeff Hester on Sat, 18 Nov 2006 23:22:56 GMT
View Forum Message <> Reply to Message

Wox wrote:

> On Wed, 08 Nov 2006 10:07:40 -0700, JD Smith <jdsmith@as.arizona.edu>
> wrote:
>
>> I don't see how forward and reverse mapping in this context are any
>> different from each other.
>
>
> The approach is different. And yes, if you just had the tie points in
> input and output image, you could choose between forward and reverse
> mapping.
>
> But as stated after the description of forward and reverse mapping: "I
> only have the surfaces (Xi,Yi)->Xo and (Xi,Yi)->Yo". I have these
> surfaces as 2D splines, I don't have the anchor points.
>
> So I can't just "swap the anchor points", because I don't have them, I
> only have the coefficients of two 2D splines.
>
> The thing is, how to "swap these surfaces", if you know what I mean.
>
> Off course, one could define some arbitrarly tie points, evaluate the

> spline for them and then "swap the anchor points" to do reverse
> mapping. But how to define these tiepoints?
>

Apologies if I'm repeating something that has been said, but when I am faced with this problem, I do the following:

(1) Set up a grid of points x_i, y_i spanning the image that you want to warp, then transform them into eta_i, xce_i in space you are warping into. (This is the transformation that you know how to do.)

(2) Do a least squares fit for some function, (x_i, y_i) = F(eta_i, xce_i) using these sample points.

(3) Do the "reverse" transformation in the standard way, marching through the output (eta, xce) space using F() to map the regularly gridded coordinates back into the original image.

This runs very efficiently in IDL since you can transform large arrays of coordinates at once. (I usually do it with a loop over rows, but there is no reason you can't pass coordinates for the entire array in one go.)

The key is in choice of the mapping function. Functions of the form x = a + b*eta + c*xce + d*eta^2 + e*xce^2 + f*eta*xce + (whatever order terms you care about) usually work pretty well for my applications (which typically involve optical distortions). You can look at the errors in the transformation to judge whether they are good enough for your purposes.

The other thing that you can do is treat the coordinate transformation as an interpolation problem from an irregularly gridded array to get values for x_i, y_i on a regular grid of points eta_i, xce_i. Then as above use these values to do the reverse transformation.

As an aside, I sometimes have to put images through a long string of transformations. This is messy, because each time you transform an image you introduce resampling errors. So I set up a couple of coordinate arrays, X and Y, in the original image. (X just has the x coordinate for each pixel, while Y has the Y coordinate.) I then put the X and Y arrays through each transformation that I apply to the image I am working with. At the end of the sequence, the transformed X and Y arrays contain the non-integer coordinates in the input array corresponding to each pixel in the output array. This provides the coordinates that I need to go back and redo the entire sequences of resampling operations in a single step.

Hope this helps.

Jeff Hester