



LINKÖPINGS TEKNISKA HÖGSKOLA
CAMPUS NORRKÖPING - ITN
TNM078 IMAGE-BASED RENDERING, MODELING AND LIGHTING
2005-03-23

V I E W M O R P H I N G



INTERPOLATION BETWEEN TWO EXTERNAL VIEWS

Alexander Petrovski
alepe261@student.liu.se

Frank van der Stelt
frava974@student.liu.se

Contents

1 INTRODUCTION	1
1.1 BACKGROUND AND AIM.....	1
1.2 METHOD AND SOURCES.....	1
2 VIEW MORPHING	2
2.1 PREWARPING.....	2
2.2 MORPHING.....	5
2.3 POSTWARPING	6
3 RESULT.....	7
3.1 ISSUES	7
3.2 RUNNING THE CODE AND BASIC STEPS.....	8
4 FUTURE WORK.....	8
REFERENCES	9

1 Introduction

Image based rendering, modelling and lighting is a quite new area of research and development but several techniques is present to increase dynamics and making photorealistic scenes easier than traditional computer graphics. Creating the real world inside a computer is getting more and more interesting as the development of computers gives rise to machines that can perform billions of operations per second. We are exploring the method “View Morphing” which allows interpolation between two images of an object taken from two different viewpoints. This produces the illusion of physically moving a virtual camera.

1.1 Background and aim

The course aim is to create knowledge and skills in current computer graphics methods where photographic source material is utilized for the creation of virtual models and for increasing the realism of synthetic images. The aim of this project is to get deeper knowledge and investigate feature-based image morphing and view morphing.

This report is written as an obligatory part of the project in the course TNM078 and was chosen between different areas of Image based rendering, modeling and lighting.

It is composed as a technical report and key elements needed to create a system making it possible to create synthesized change both in viewpoint and image structure via simple image transformations.

Algorithms will be implemented in Matlab using images from the internet and images from a digital camera. The program will output N images that perform the morphing. These images can then be edited together into a short animated sequence.

Some basic knowledge of physics and programming is required to understand details of the report. Almost nothing is written about the working process.

The project team consists of two persons and the time for this project is estimated about 2.5 weeks of work.

1.2 Method and sources

This project is based on the paper *View Morphing* written by Steve M. Seitz and Charles R. Dyer that was presented on SIGGRAPH 1996 [2]. Seitz Ph.D. dissertation [5], gave clarity where it was needed. By reading papers over and over again, information and knowledge have been gained. When problems have been encountered, discussions with each other in the project team to solve these in cooperation with more information about the problem to get a better understanding and salvation have been done. Other sources that have been used can be found on the reference page of this document.

To Solve Matlab-oriented programming and syntax problems the documentation has been supportive and sometimes online research e.g. Google.com has been used.

2 View morphing

Morphing is a technique that deals with the creation of smooth transitions between two images. Standard morphing techniques combine 2D shape and colour interpolations to get these in-between images. One of the problems with standard morphing techniques is that they do straight-forward morphing without keeping the geometry of the images. In in-between images you can see geometric distortions. An in-between image of a computer generated cube may look a child's drawing of a similar cube. The lines aren't straight anymore and a smooth interpolation needs good geometry preservation. View morphing deals with this problem by taking the positions of the cameras into account. The technique uses correspondence points in the images and epipolar geometry not just to interpolate shape and colour, but to interpolate camera positions. In this way the geometric structure of the scenery is preserved.

The view morphing algorithm consists of three steps. The first step is to prewarp the two images. This process changes the orientation of the image planes so that they share the same normal direction. The optical centres of the cameras are unchanged. The second step is to morph these prewarped images. The strength of the view morphing algorithm lies partly in the fact that it can be used with existing morphing algorithms. The morphing algorithm used in this project was the one proposed in "feature-based image metamorphosis" by Beier and Neely [1], a classic morphing technique made famous by the *Black or White* video by Michael Jackson. This technique will be described in more detail in the morphing section. The morphing step changes the optical centre to the optical centre of the desired virtual view. The last step is the postwarping step which transforms the virtual image to its new position and orientation.

2.1 Prewarping

In the appendix of [2] the authors describe a method for automatically computing the image prewarping transforms H_0^{-1} and H_1^{-1} from the images themselves. These transforms are applied to images I_0 and I_1 to produce prewarped images \hat{I}_0 and \hat{I}_1 . The first step in obtaining these transforms is to determine the fundamental matrix. The fundamental matrix is the geometric representation of epipolar geometry.

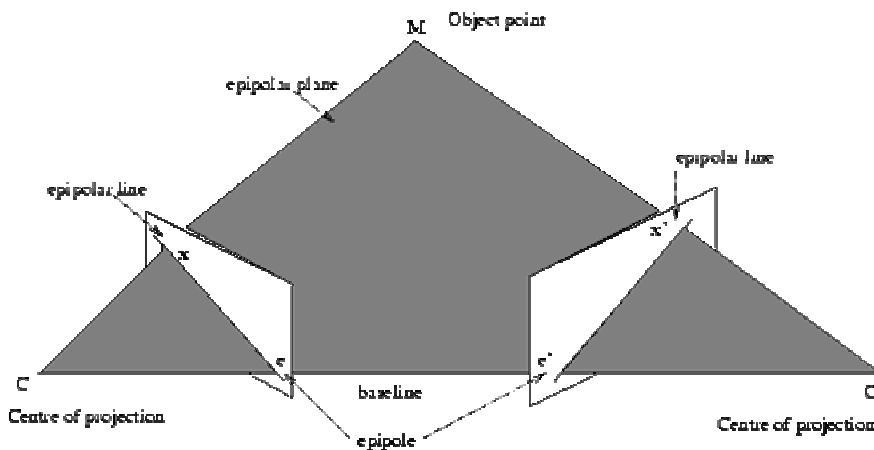


Fig 1
Epipolar geometry explained

Two stereo images have corresponding points. Points in image I_0 are mapped to lines in I_1 . Epipolar geometry deals with this mapping from points to lines. A point X in the real-world scene has an equivalent in images I_0 and I_1 , namely points x and x'. The line between x and X is

¹ http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/OWENS/LECT10/node3.html

projected onto I_l and we know that x' has to be somewhere on this projected line. This makes searching for that point a lot easier since we only have to search for the point on this line. The epipole is the intersection point of the line between the camera centres, which is called the baseline, with the image plane. An epipolar plane is a plane containing the baseline. And an epipolar line is the intersection of an epipolar plane with the image plane. The fundamental matrix is the algebraic representation of epipolar geometry [3]. The fundamental matrix \mathbf{F} satisfies the correspondence condition:

$$x'^T \mathbf{F} x = 0 \quad (1)$$

The fundamental matrix is a unique 3×3 rank 2 homogeneous matrix. Equation (1) is true for any pair of corresponding points, $x' \in I_1$, $x \in I_0$.

The fundamental matrix is computed using Hartley's eight-point algorithm. This algorithm requires 8 or more corresponding points. As a rule of thumb it is said that eight points is a minimum and 12 should be sufficient for this algorithm. The fundamental matrix is computed using a linear method, the solution is the least eigenvector, \mathbf{f} of $A^T A$, where A is the equation matrix [4].

The prewarping transforms or homographies H_0^{-1} and H_1^{-1} can then be computed. These homographies transform the input images so that their scanlines are aligned. This means that every corresponding point-pair has the same y-coordinate. The appendix of [2,5] describe a way of making the images parallel by first applying a 3D rotation to make the image planes parallel, followed by a 2D affine transformation to align the scanlines. The rotation matrix for image I_0 is given by the following:

$$R_{g_0}^{d_0} = \begin{bmatrix} (d_0^x)^2 + (1 - (d_0^x)^2) \cos g_0 & d_0^x d_0^y (1 - \cos g_0) & d_0^y \sin g_0 \\ (d_0^x d_0^y (1 - \cos g_0)) & (d_0^y)^2 + (1 - (d_0^y)^2) \cos g_0 & -d_0^x \sin g_0 \\ -d_0^y \sin g_0 & d_0^x \sin g_0 & \cos g_0 \end{bmatrix}$$

$d_0 = [d_0^x \ d_0^y \ 0]^T \in I_0$ is the axis of rotation and given $Fd_0 = [x \ y \ z]^T$ the corresponding axis in image I_l is found through $d_1 = [-y \ x \ 0]^T$. The epipoles are needed in order to perform the 3D image rotation. The epipoles $e_0 = [e_0^x \ e_0^y \ e_0^z] \in I_0$ and $e_1 = [e_1^x \ e_1^y \ e_1^z] \in I_1$ are the unit eigenvectors of \mathbf{F} and \mathbf{F}^T respectively. The desired angle of rotation is given by:

$$g_i = \tan^{-1} \left(\frac{e_i^z}{d_i^y e_i^x - d_i^x e_i^y} \right)$$

This rotation is performed on both images. Applying rotation matrices $R_{g_0}^{d_0}$ and $R_{g_1}^{d_1}$ to I_0 and I_l respectively gives us the rotated images. The next step is the 2D affine transformation where we rotate the images so that the epipolar lines are horizontal. The new epipoles are given by:

$$R_{g_i}^{d_i} = [\hat{e}_i^x \ \hat{e}_i^y \ 0]^T$$

The angles of rotation are given by:

$$g_i = -\tan^{-1} \left(\frac{\hat{e}_i^y}{\hat{e}_i^x} \right)$$

$$R_{g_i} = \begin{bmatrix} \cos \mathcal{G}_i & -\sin \mathcal{G}_i & 0 \\ \sin \mathcal{G}_i & \cos \mathcal{G}_i & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The final version of the fundamental matrix has the following form:

$$\tilde{F} = R_{g_1} R_{g_1}^{d1} F R_{-g_0}^{d0} R_{-g_0} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & a \\ 0 & 1 & b \end{bmatrix}$$

The second image is translated and vertically scaled by the following matrix:

$$T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -a & -b \\ 0 & 0 & 1 \end{bmatrix}$$

The homographies are obtained by the following two equations:

$$H_0 = R_{g_0} R_{g_0}^{d0}$$

$$H_1 = T R_{g_1} R_{g_1}^{d1}$$

The inverse of these homographies are used to do the projective transform. The reprojection transformation in [1] is used.

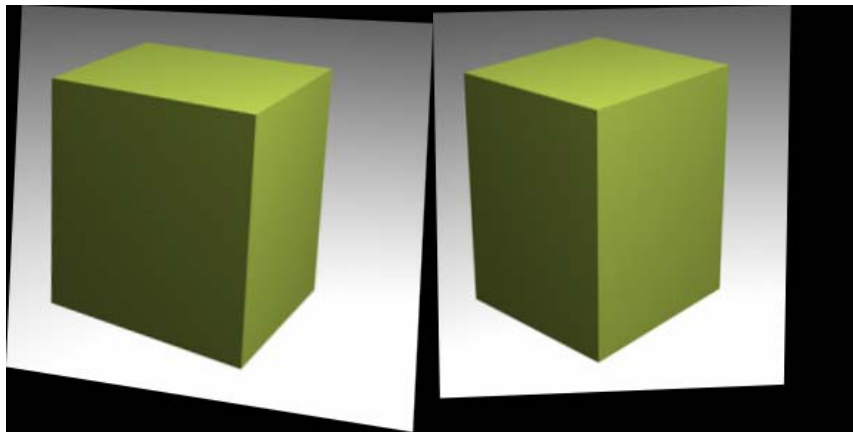


Fig 2

Two prewarped images, the corresponding points lay in the same scanline, they have the same y-coordinates.

2.2 Morphing

In the morphing phase, the two prewarped images are blended by linearly interpolating the colours and positions of the corresponding points. The interpolation of two corresponding points $p_0 \in I_0$ and $p_1 \in I_1$ generates a new point. This point, p_s is defined by:

$$p_s = (1-s)p_0 + sp_1 \quad s \in [0,1] \quad (2)$$

The morphing method used here is the feature-based image metamorphosis method which uses fields of influence around control primitives. These user-defined control primitives, multiple pair of lines, are used to transform the picture using an inverse mapping scheme. Inverse mapping guarantees us that every pixel is set to an appropriate value without having to deal with unpainted pixels [1]. For each pixel in the destination image we find a value in the source image through some sort of transfer function. PQ is the line where P and Q are the endpoints of the user-defined line in the destination image. P'Q' is the corresponding line in the source image, all primed variables are values defined relative to the source image. Variables u and u' are values for the positions on the line and are between [0,1]. Variables v and v' are distances from the line. These corresponding lines define a coordinate mapping from pixel coordinates X to X' such that:

$$u = \frac{(X - P) \cdot (Q - P)}{\|Q - P\|^2}$$

$$v = \frac{(X - P) \cdot \text{Perpendicular}(Q - P)}{\|Q - P\|}$$

$$X' = P' + u \cdot (Q' - P') + \frac{v \cdot \text{Perpendicular}(Q' - P')}{\|Q' - P'\|}$$

The function *Perpendicular()* returns the vector that is perpendicular to, and has the same length as the input vector. This transformation is for one pair of lines. The generalization of this transformation is given the so-called multiple line algorithm.

The defined lines are weighted by a weighing function. The equation for this weighing that is used both in [1] and in our implementation and the multiple line algorithm are given by:

$$\text{weight} = \left(\frac{\text{lenght}^p}{a + \text{dist}} \right)^b$$

The variables a, b and p are variables that determine the relative effect of the lines. If the value of variable a is near zero and the distance from the line to the pixel is zero, the weight is nearly infinite. Variable b determines how the strength of the lines falls of with distance and p the strength of the lines. If p is zero then all lines have the same strength. Longer lines will weight more if the value for p is larger.

The multiple line algorithm, written in pseudo code [1]:

```

For each pixel  $\mathbf{X}$  in the destination
   $\mathbf{DSUM} = (0,0)$ 
   $\mathbf{weightsum} = 0$ 
  For each line  $\mathbf{P}_i\mathbf{Q}_i$ 
    calculate  $\mathbf{u},\mathbf{v}$  based on  $\mathbf{P}_i\mathbf{Q}_i$ 
    calculate  $\mathbf{X}'_i$  based on  $\mathbf{u},\mathbf{v}$  and  $\mathbf{P}_i'\mathbf{Q}_i'$ 
    calculate displacement  $\mathbf{D}_i = \mathbf{X}'_i - \mathbf{X}_i$ 
     $\mathbf{dist} =$  shortest distance from  $\mathbf{X}$  to  $\mathbf{P}_i\mathbf{Q}_i$ 
     $\mathbf{weight} = (\mathbf{length}^p / (\mathbf{a} + \mathbf{dist}))^b$ 
     $\mathbf{DSUM} += \mathbf{D}_i * \mathbf{weight}$ 
     $\mathbf{weightsum} += \mathbf{weight}$ 
   $\mathbf{X}' = \mathbf{X} + \mathbf{DSUM} / \mathbf{weightsum}$ 
  destinationImage( $\mathbf{X}$ ) = sourceImage( $\mathbf{X}'$ )

```

This procedure is done for the two input images and the output is linearly interpolated by an equation that is similar to (2). Variable s in this case is the so-called cross-dissolve factor and is between $[0,1]$. A cross-dissolve factor of 0.5 generates an image that is in the middle of the two images. If the value is closer to 0, the image looks more like image I_0 and if the value is closer to one, the image is more like I_1 . By generating a lot of images with cross-dissolve increasing from zero to one, an animation can be obtained. But the generated images and animations often suffer from ghosting artefacts and it takes a lot of control primitives in order to get a decent result.

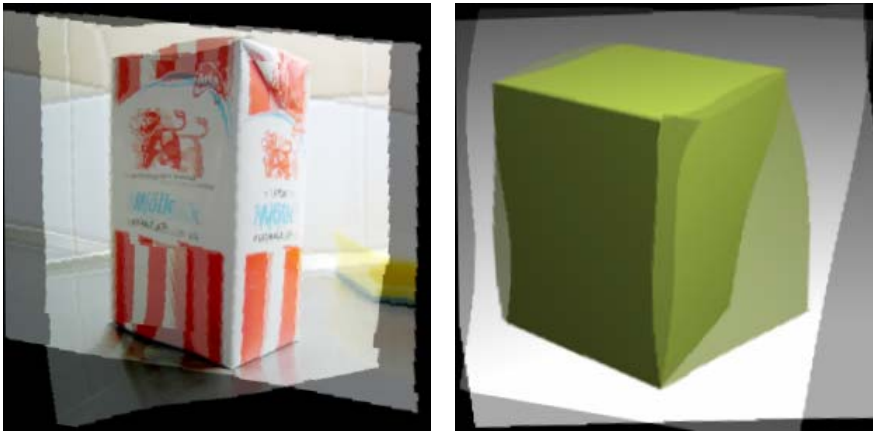


Fig 3
Two in-between images with heavy ghosting artefacts

2.3 Postwarping

The postwarp operation is defined by applying the projective transform H_s to image \hat{I}_s which gives us the desired image I_s . This transform changes the current new view image plane to its new orientation and position. This is done by letting the user define desired configuration of the postwarp. This method is described in [5] and is similar to the process of finding the constrained linear system for the fundamental matrix. The user defines four points such that:

$$p = [x \ y \ 1]^T \text{ and } \hat{p} = \begin{bmatrix} \hat{x} & \hat{y} & 1 \end{bmatrix}^T$$

These points are related by:

$$\hat{c} p = H_s p = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} p$$

The unknown variable c is eliminated and all that is to be done is to solve two linear equations per control point in the coefficients of H_s .

$$x(h_{31} \hat{x} + h_{32} \mathcal{G} + h_{33}) - h_{11} \hat{x} - h_{12} \mathcal{G} - h_{13} = 0$$

$$y(h_{31} \hat{x} + h_{32} \mathcal{G} + h_{33}) - h_{21} \hat{x} - h_{22} \mathcal{G} - h_{23} = 0$$

In the current implementation, the postwarp is not calculated in this way. We use a method that linearly interpolates four points in each of the original images and four corresponding points in each of the prewarped images. The final image I_s is transformed with these values and gives reasonable results.

3 Result

As presented in the View Morphing paper, transformations between two images from different viewpoints or even two different images, can be morphed or changed to desired camera view. This morph uses a 2-dimensional technique and it is no need to have earlier knowledge about 3D. It appears on the resulting image or images, like it would have been a real 3-dimensional transformation.

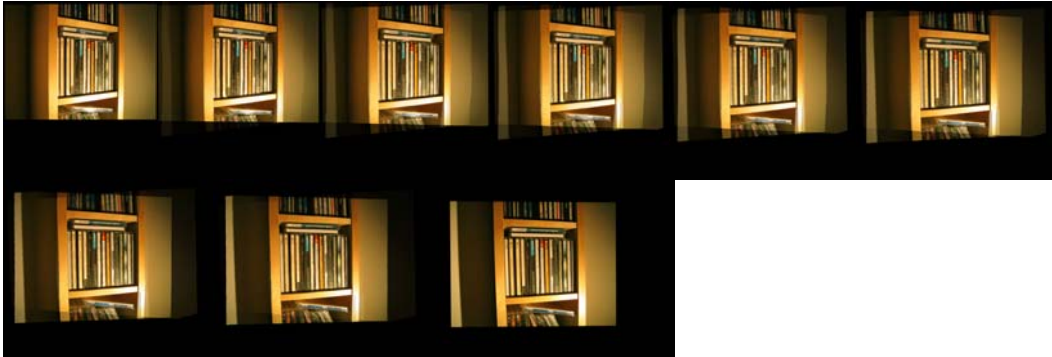


Fig 4

This sequence shows a short animation representing a camera movement and a view change.

3.1 Issues

There are problems with this technique. To achieve good results on the interpolation between views, everything seen on one image must be seen in the other. This means that there is a restriction in the angle of camera movement.

To select corresponding points is also difficult to manage. It has to be done precisely and it is also hard to get an understanding of how many corresponding points that will be enough to get the program to understand the camera change in view. It is not possible in the program to move the points afterwards.

It is also not trivial to select feature lines. By having “wrong” features selected, ghosting appears on the output image or sequence of images. Ghosting makes the animation worthless and unnatural and in many cases the animation only looks strange. It is necessary to experiment with these features to be able to create animations where the ghosting artefacts are minimized. It is possible to get good transitions between two images, but in many cases it is difficult. A third dimension would not solve all these problems but would give the advantage to be not so sensitive of changes in visibility. This also requires some knowledge of 3D.

3.2 Running the code and basic steps

The implementation is made for Matlab 6.5 and requires at least 256mb of internal memory. To run the implementation, change the current directory to the directory where the Matlab-files are located.

Run the program by writing the following in the Matlab command window:

```
ViewMorph('left.bmp', 'right.bmp', 25, 6);
```

The first two arguments are the paths to the images and the third is the number of output images to render. The last argument is the number of feature lines to select.

The first step is to set a number of corresponding points in each image. A recommendation is to use at least 16 points.

When this step is finished, another image shows. Pick four points and remember these. Pick the same points in the next upcoming image.

Now two pre-warped images will show up. Pick the same points as in the last step on the two pre-warped images.

Now it is time to select feature lines. Select start and endpoint of the line where a feature is present. Do this until the number of line is as specified in the argument. When this is done the computer starts to do calculations. The output directory for this program is hard coded to D:\.

4 Future work

We started of with the idea of doing virtual views and implementing the *High-quality video view interpolation using a layered representation* paper. However, this paper seemed a bit overwhelming and therefore we tried to achieve the same goal by other means. We read a lot of papers on the subject of view morphing, dynamic view morphing and disparity-based view morphing, all having the words view and morphing in common. Trying to implement the view morphing paper was a project that seemed a bit more realistic in terms of time limitations and knowledge levels. View morphing is a well established technique and it is possible to find a lot of references on the subject on the internet.

One of the immediate improvements we would like to do is to fix the postwarp phase. We don't know how this would affect the result but it would probably be a better one. A decent GUI would make the user-definition of the numerous amounts of points a lot easier. It would be nice to have a method that takes the output images and creates an animation. The implementation as it is now is rather instable and slow. The project would benefit from a total rewrite in C++, it would probably make the implementation faster. The goal is to be able to do the morphing in near real-time with dynamic images (video) and to be able to change view points as we please. We are a long way from achieving that goal. But the project gave us deeper insight in the subject of computer vision, view morphing and the hardships of achieving the ambitious goal of creating virtual views from few images.

References

- [1] Beier, T. and Neely, S., *Feature-based image metamorphosis*. Proc. SIGGRAPH 92. In *Computer Graphics* (1992), pp. 35-42
- [2] Seitz, S. and Dyer, C., *View Morphing*, Proc. ACM SIGGRAPH 1996, 1996, pp 21-30
- [3] Hartley, R. and Zisserman, A., *Multiple view geometry in computer vision*, Cambridge University Press, 2000, pp 231-261
- [4] Hartley, R., *In defence of the 8-point algorithm*, Fifth International Conference on Computer Vision, IEEE Computer Society Press, Cambridge Massachusetts, 1995, pp. 1064-1070
- [5] Seitz, S., *Image-Based Transformation of Viewpoint and Scene Appearance*, Ph.D. Dissertation, Computer Sciences Department Technical Report 1354, University of Wisconsin, Madison, October 1997.

Other sources used in this project

<http://www.cs.washington.edu/homes/seitz/> (acc. 23-03-2005)

<http://www.cs.ucf.edu/~vision/projects/viewMorphing/> (acc. 23-03-2005)

<http://www.dgp.toronto.edu/~gelkoura/csc2530/a2/> (acc. 23-03-2005)

http://users.rsise.anu.edu.au/~gareth/vision_course/Lab3/Lab3.PDF (acc. 23-03-2005)

<http://www.csse.uwa.edu.au/~pk/> (acc. 23-03-2005)

<http://www.isr.uc.pt/~nmartins/work.html> (acc. 23-03-2005)

<http://www.cs.ucf.edu/~vision/projects/triview/> (acc. 23-03-2005)