JavaMorph Tutorial

An image processing program

Content

Document history

Version	Date	Remark
1.0	2009-01-07	Initial version.

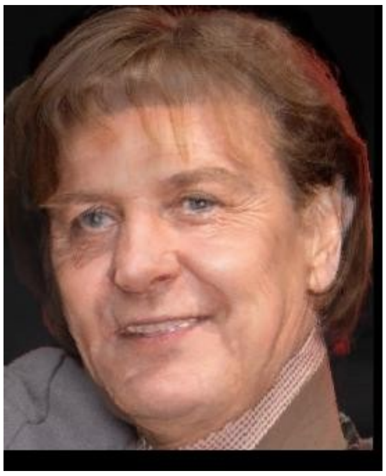
Document history

Who is “Moeder”

The creation of Moeder

Moeder is an hermaphroditic, optical merged, medial being. It was created between the former German federal chancellor Mr. Gerhard Schroeder and the current office holder Mrs. Angela Merkel. The creation of Moeder has been done by the morph program “Java Morph” which is the subject of this tutorial. Depending on the configuration of JavaMorph Moeder can have different numbers of siblings with several mix ratios between its both parents.

Schroeder


Moeder

Merkel

The morphing method in film industry

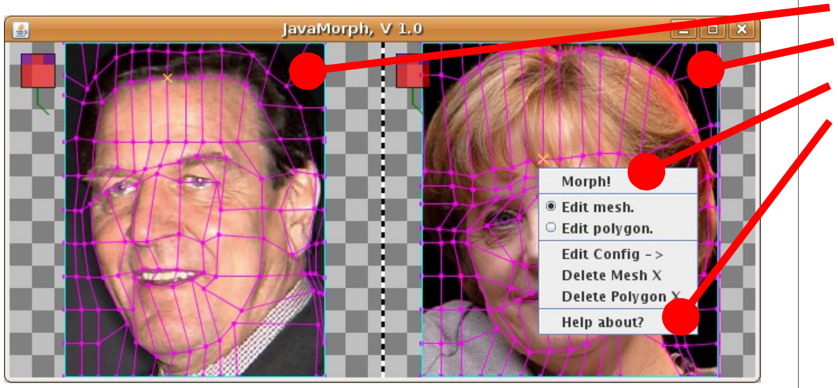
It may be the intention of one author to merge two similar but different pictures into one result picture. The common way is to add two half transparent layers of two input pictures with the same size. But there is a general problem: Exponent regions of both pictures can be situated at different coordinates even if both pictures have the same size. An additional processing step is necessary while rendering. The intermediate copies of both input pictures have to be deformed in that manner that all exponent regions are moved to their congruent place within the result picture. Because the calculation logic is not able to find exponent regions (those are eyes, nose, lips and other in the case of a portrait picture) the morph computer program lays two identical meshes over both input pictures. The author has to move the mesh points over the exponent points of the pictures. Now the morph program is able to create the merged pictures.

Start the program with its sample data

To install the program you need two items:

- The small JavaMorph program as a *.jar archive.
- Sun's **J**ava **R**untime **E**nvironment Version 1.6 alias Version 6 which is distributed stand alone or as a part of the Java development kit.

If You are not sure whether the JRE is installed on Your computer → open a console window an type: **>java -version**! A version of 1.6 ore higher should be displayed. If this is not the case please download the JRE from: http://java.sun.com/javase/downloads/?intcmp=1281! To start JavaMorph → open a console window again and enter: **>java -jar JavaMorph<version_extension>.jar**. When called for the first time JavaMorph creates a folder on the Desktop named with the program name. No other files will be created in the file system.



Screen shot of JavaMorph

- Left input
- Right input
- Merge command
- Global info

Screen shot

After You have started the program You will find a prepared sample configuration with meshes. All You have to do is to click the **Morph** item from the mouse-button-right-pop-up-menu. JavaMorph will create a sequence of morphed pictures which You can find in the following directory:

Desktop\JavaMorph\output\*.jpg

while the Desktop directory is situated within the user's home.

License

JavaMorph is distributed under **GPLv2**. You are free to copy and use the program without payment. All modifications are allowed. Please provide another unique version number if You have made modifications! Author, so far, is: claus.erhard.wimmer@googlemail.com. Find sources also in the unique distribution or in CVS!

Where can I find what?

The .jar archive

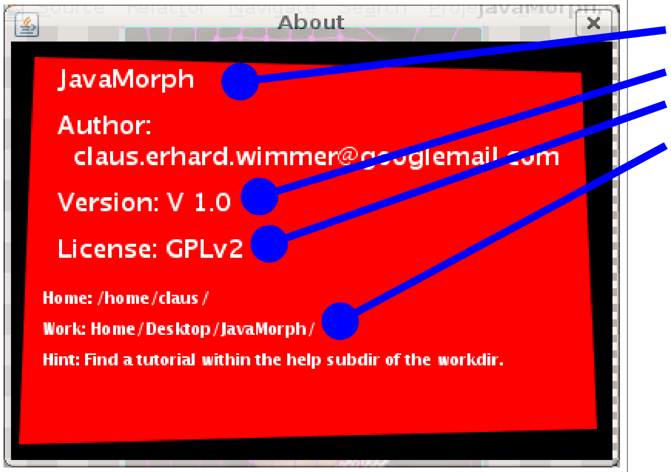
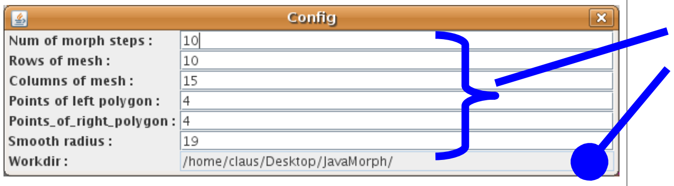
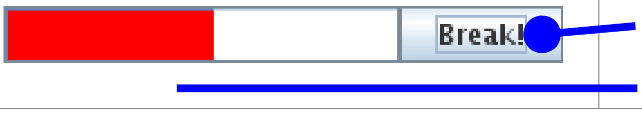
This archive acts as the program distribution. It contains a sample origin of the later copied working directory on the desktop, the binary distribution, the source files and a JavaDoc folder. While calling the .jar archive with the Java runtime interpreter the working directory on the desktop is created if it doesn't exist. The file's name of the .jar archive is "JavaMorph_yyyy_mm_dd.jar". This name is also defined within the class CStrings.java within the sources. It shall be modified according to increased version numbers of future program versions. Hint: You can open the .jar archive with every common .zip program by replacing the extension .jar by the extension .zip temporarily.

Folders in the working directory

Folder's name	Remark
help*.pdf	Help file & tutorial in PDF format. Open with a PDF reader like acrobat.
output*.jpg	Generated morph result. Old files deleted before a new batch run.
input*.jpg	Both input files as left.jpg & right.jpg. Edge length ~100 ... ~1000.
mesh*.msh	Both mesh files. Renewed with defaults, after closing, if deleted.
polygon*.pol	Both clip polygon files. Same behaviour like mesh files.
debug*.png	Smoothed clip matrix view with dilatation borders.
.*.props	Property file with polygon and mesh dimensions. Read &edit also as ASCII text! Created new if removed.

Directories

The modal dialogues

Dialogue name	Screen shot	Description
About	 About JavaMorph Author: claus.erhard.wimmer@googlemail.com Version: V 1.0 License: GPLv2 Home: /home/claus/ Work: Home/Desktop/JavaMorph/ Hint: Find a tutorial within the help subdir of the workdir.	• Program's name • Version • License • Working directory
Configuration	 Config Num of morph steps : 10 Rows of mesh : 10 Columns of mesh : 15 Points of left polygon : 4 Points_of_right_polygon : 4 Smooth radius : 19 Workdir : /home/claus/Desktop/JavaMorph/	• Settings • Path of the working directory
Progress		• Abort button • %

Dialogues

Mesh Algorithm

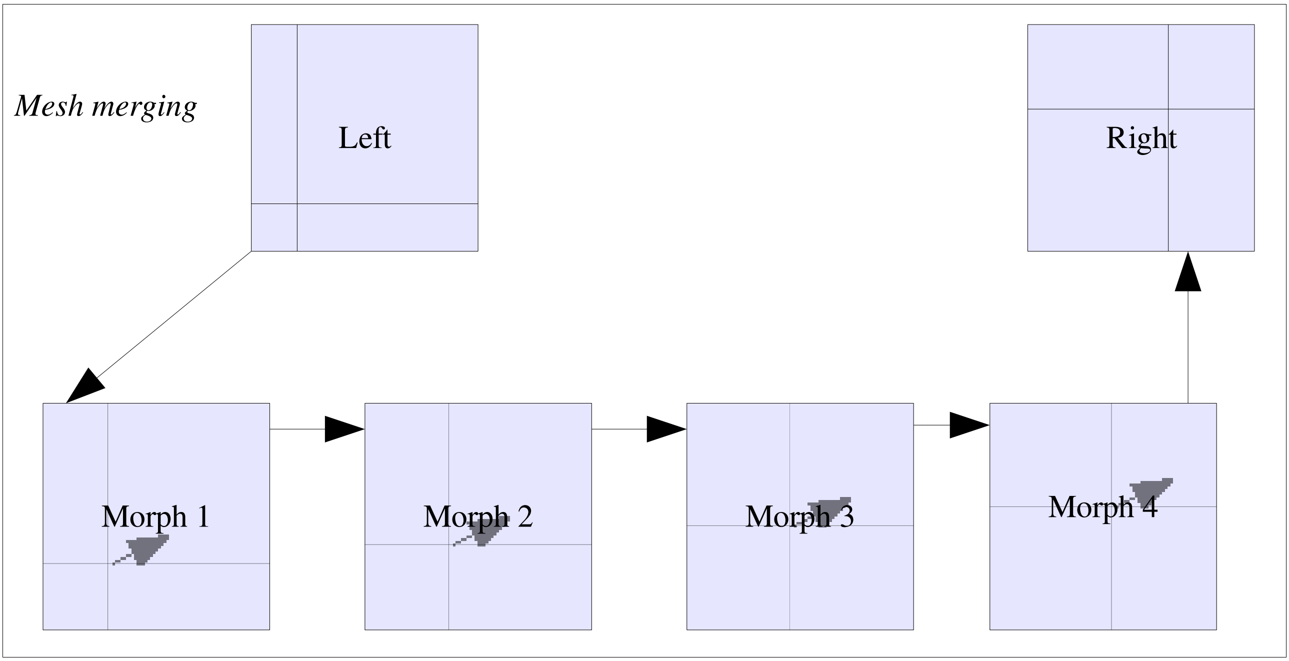
Corner point merging

Subject of the mesh algorithm is it to calculate intersection points of the result matrix as weighted middle values from intersection points, with identical row & column index, fetched from both input meshes. Corner points are calculated with the following formulas:

$$x = x_{left} * (1 - ratio) + x_{right} * ratio$$

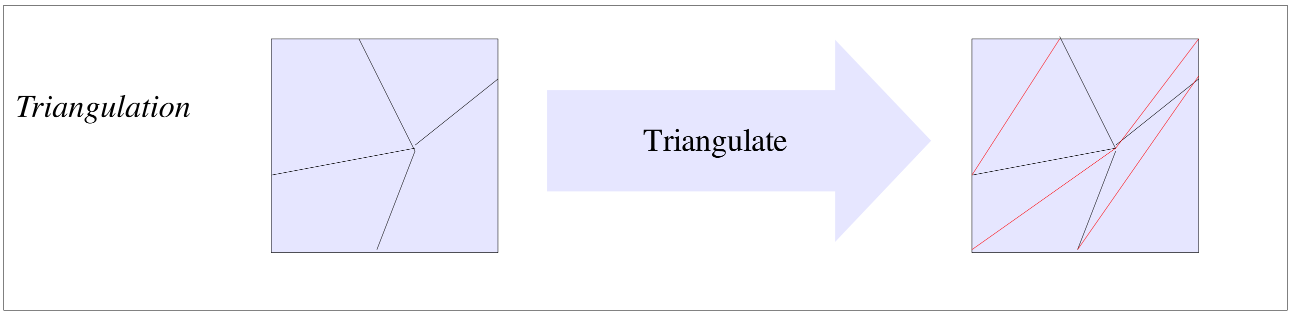
$$y = y_{left} * (1 - ratio) + y_{right} * ratio$$

This leads to the following behaviour:



Triangulation of the mesh's rectangles

As the merger of both inputs a resulting mesh with a deformed grid appears. The mesh consists of a number of irregular quads. Each quad is limited by four of the meshes intersection points which act as the quad's corners. Next each quad is split into 2 triangles:



The Intention is, to deform the coordinates of every point from each input mesh to to corresponding point of the result mesh. This is done by a transformation matrix with 2 rows and 3 columns. A quadratic matrix isn't necessary, a concatenation isn't needed to be done. Because no gaps in the result will be accepted, for every point of the result picture the corresponding points of the input pictures are calculated reversely with the following formulas:

$x_{input} = a_{11} * x_{result} + a_{12} * y_{result} + a_{13}$

$y_{input} = a_{21} * x_{result} + a_{22} * y_{result} + a_{23}$

All in all a triangle can be transformed by the matrix. Corner points are matched exactly. Area & line points are interpolated. Additionally to the normal affine transform with translate rotate and scale the coordinate system of the triangle is not orthogonal. The reader can obtain the transform matrix between a single pair of triangles by solving a system of 6 equations with six variables. This can be done by insert / set equal / add pairs of equations.

(1) $x_{input\_1} = a_{11} * x_{result\_1} + a_{12} * y_{result\_1} + a_{13}$

(2) $y_{input\_1} = a_{21} * x_{result\_1} + a_{22} * y_{result\_1} + a_{23}$

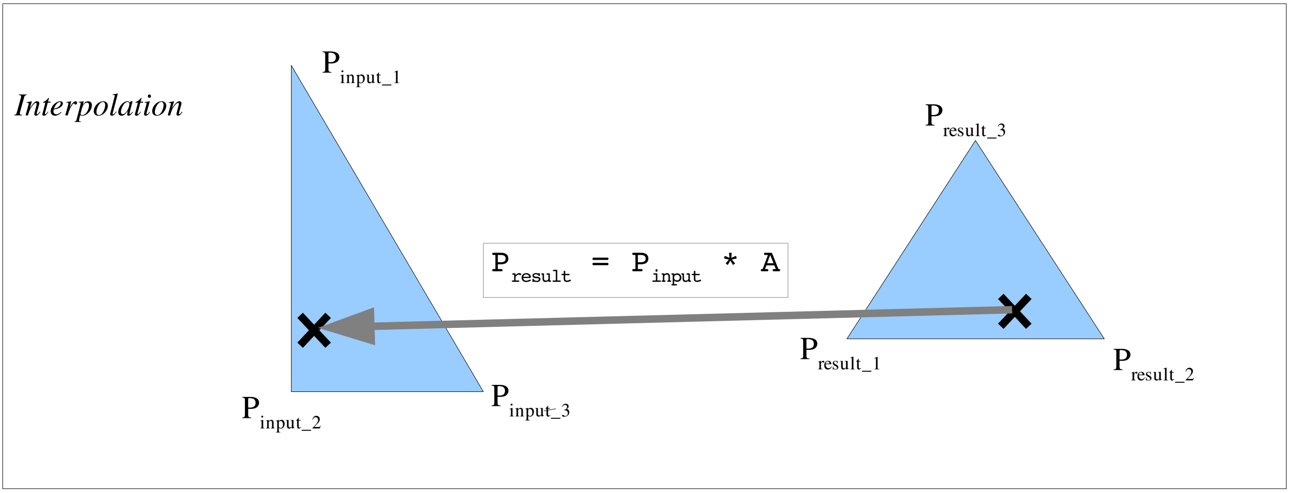
(3) $x_{input\_2} = a_{11} * x_{result\_2} + a_{12} * y_{result\_2} + a_{13}$

(4) $y_{input\_2} = a_{21} * x_{result\_2} + a_{22} * y_{result\_2} + a_{23}$

(5) $x_{input\_3} = a_{11} * x_{result\_3} + a_{12} * y_{result\_3} + a_{13}$

(6) $y_{input\_3} = a_{21} * x_{result\_3} + a_{22} * y_{result\_3} + a_{23}$

Now the points are transformed. All points belong to one index of triangle in the list.



The next section explains what has to be done with the colour values of the point lists.

Polygon clipping

Creating the clip matrix

An additional feature of JavaMorph is the possibility to limit the information, moving from the input picture into the result picture, by defining a polygon. The inner of the polygon appears within the result when the ratio points to the specified input picture. A clip matrix with the dimensions of the result picture is created. Each point of the clip matrix corresponds to one pixel at the picture. If the value of the clip picture is 0.0 then the pixel is not shown. If the value is 1.0 then the pixel is shown. Smoothed edges of the polygon lead to intermediate values.

Line algorithm

For each line of the polygon a pixel proof line within the clip picture is created. Exactly the line shall be proof in orthogonal directions but not in diagonal directions. This is done by the parameter p.

$dx = x_2 - x_1$

$dy = y_2 - y_1$

$p\_max := maximum(dx, dy) \mid p\_max > 0$

For all integer p from 0 to p_max do the following:

$x = x_1 + p * (x_2 - x_1)$

$y = y_1 + p * (y_2 - y_1)$

Flood flood fill algorithm

Now the inner of the polygon shall be filled. First one inner point is sought by the scan line algorithm. The scan line algorithm scans all lines of the clip picture. If it founds one point satisfying the following conditions it returns the coordinates of the point p:

- A preceding colour(p) is 1.0.
- The colour(p) itself is 0.0.
- A following colour(p) is 1.0 again.

Then the morph operator starts the following recursive process with the obtained p:

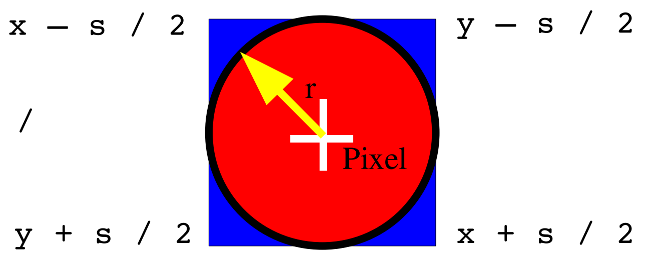
1. If colour(p) is 0.0 then set it to 1.0 -> continue with 2 | else do nothing.
2. Repeat step 1. for each orthogonal neighbour of p.

Smooth algorithm

With increasing distance from the outer edge of the polygon the value of the clip picture ramps down linear from 1.0 to 0.0. A smooth function renders this pattern into the clip matrix. It is executed for each pixel of each line of the polygon. A copy of the line algorithm calculates the pixels for the smooth algorithm. The smooth function acts within a quadratic area around the pixel.

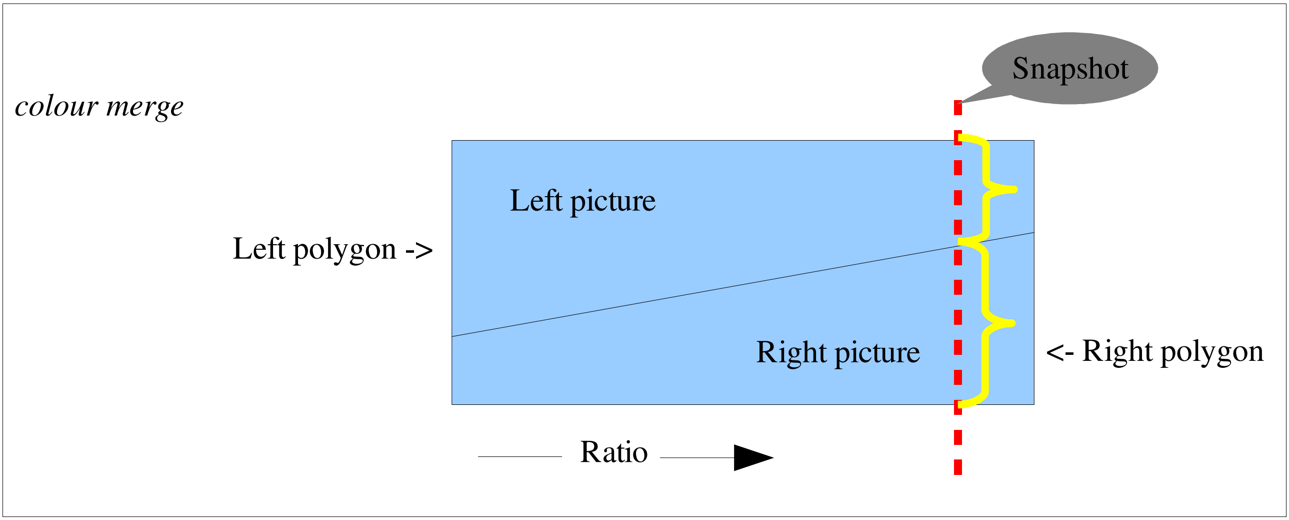
Smooth algorithm

$Color := 1.0 - r / s$



Weighted pixel merger

Finally the morph algorithm merges the colours of the two input pixels into the colour of the result pixel. So far the algorithm doesn't distinguish between red green and blue. The output colour depends now on step ratio, clip matrix and input colours from left & right.



Program design

Decorator pattern

The program displays both input pictures in separate JComponents implemented as Class CFrame. While CFrame only displays the background of the picture, four decorators render a number of details for the frame layout. First the CPictureDecorator paints the picture, scaled to the window size but with fixed edge ratio. Second the CMeshDecorator paints the mesh and receives user edit events. Third the CPolygonDecorator paints the clip polygon makes it editable and provides the clip picture, generated by the algorithms explained above. Fourth the CPopupMenuDecorator paints the mouse symbol and shows the pop up menu. All four classes implement the IDecorator interface. Here the paint() method is the most important procedure. CFrame calls sequentially the paint() method of all four decorators in the described order.

Configuration classes

The first configuration class is CStrings. This class holds a number of directory paths and file names. The name of the .jar archive of this application shall be entered here. The second configuration class is CConfig. This class holds the numerical configuration data for the morph batch. When the program starts CConfig reads a property file where it also saves the numerical data during the program shut down.

The main class

CMain is the main entry for JavaMorph. It contains the main function and it is referenced by the automatically generated manifest file of the .jar archive. This class instances also the main JFrame of the application plus the three dialogues. The JFrame itself shows a left and a right picture, painted by CFrame which are separated by a vertical line painted by a CSeparator class.

Dialogues

Additionally to the main window there are three classes holding dialogues. The first dialogue is CConfig which acts also as store for configuration data. It contains several input fields realized as CEdit class instances. The second dialogue is CAbout which shows the program's meta data, like version info & working directory, to the user. The third dialogue is CProgress. CProgress is the modal progress bar. It blocks main thread execution during the morph batch. That's why the morph batch has its own thread.

Mathematical classes

The central mathematical class is CMorphOperator. It performs the morph batch with the number of configured intermediate pictures. The mesh transform & pixel merge algorithms belong to this class. CMorphOperator has three helper classes. CGeo calculates triangulation & transformation matrices. CTriangle is a set of points limited by three corners. It has multiple instances. CTransform contains the elements of the transformation matrix which is explained above.

Usage guide

Menu functions

Use the right mouse button to open the pop up menu!

Function's name	**Description**
Morph!	Starting the morph batch. Deletes the old output files. Progress bar is shown when preparations are done and rendering of the output files begins.
Edit mesh.	Edit mesh of corresponding picture. Stop edit polygon. User can move the nearest mesh point by dragging with the left mouse button. When selected for both pictures: The other picture shows the corresponding mesh point.
Edit polygon.	Edit polygon of corresponding picture. Stop edit mesh. User can move the corner points by dragging with the left mouse button.
Edit config ->	Open the dialogue to edit numerical configuration values which influence the morph batch. Values are stored by closing the dialogue window.
Delete mesh X	Delete those mesh which belongs to the corresponding picture from the file system. A default mesh with the configured dimensions is provided and painted. The program saves the new mesh during shut down.
Delete polygon X	Delete those polygon which belongs to the corresponding picture from the file system. A default polygon with the configured dimensions is provided and painted. The program saves the new polygon during program shut down.
Help about?	Show program info. Here You can find the path to the working directory, the author's name and the version info.

Menu functions

Configuration variables

You can edit this variables by opening the configuration dialogue. Invalid values are clipped suiting into the, as tool tips, displayed, boundaries. Close the dialogue window to store!

Variable's name	Description
Num of morph steps	When entering <N> then the distance between left and right is divided into <N> steps. The output directory will contain the left & the right picture plus <N-1> mergers.
Rows of mesh	Number of horizontal rows of quads for both meshes. When modified: Old mesh is deleted & new default mesh is provided.
Columns of mes	Number of vertical rows of quads for both meshes. When modified: Old mesh is deleted & new default mesh is provided.
Points of left polygon	Corner points of the left polygon. When modified: Old left polygon is deleted & new default left polygon is provided.
Points of right polygon	Corner points of the right polygon. When modified: Old right polygon is deleted & new default right polygon is provided.
Smooth radius	Size of the fuzzy area around the polygons in both clip matrices. Used to hide borders when You use JavaMorph to mount two pictures.
Workdir	Read only but copy able. Absolute path of the working directory.

Configuration variables

Hints

After rendering the sample you can feel free to substitute the pictures left.jpg & right.jpg in the input directory by your own both. Delete also the meshes therefore & adopt new default meshes. Consider that the CPU time grows square wise with the picture's diagonal. However, it isn't necessary that both pictures have identical dimensions. The dimension of the result is the union of the dimensions from left.jpg & right.jpg. You can also deal with copies of the whole working directory. This may be useful if You intend to archive a number of morph projects.

Make a film

A simple way to create a film is the rendering as animated .gif by the **G**nu **I**mage **M**anipulation **P**rogram. Try the following:

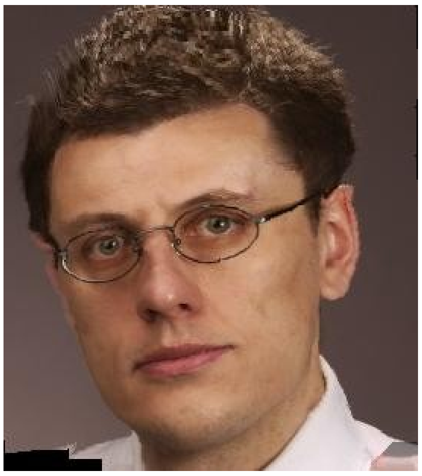
1. Open Gimp.
2. Click **File->Open** (000.jpg from output directory).
3. Click **File->Open_as_Layers** (Mark all other files of output directory) – in picture window.
4. Click File->**SaveAs** (*.gif).
5. Confirm **SaveAs_Animation** + **Make_Index**!
6. Start **Export**.

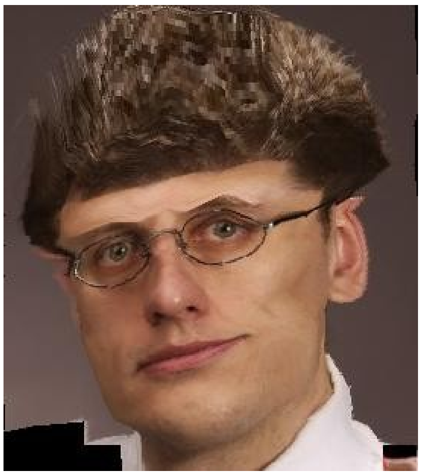
7. Choose loop mode & speed.
8. Click **OK!**
9. Open the .gif with a web browser.

Werkel

Finally I want to improve my apply photo to get a better job in a big company. It is not easy for me to smile on photos. That's why I decided to merge a little bit Mrs. Merkel into my photo. To reduce the forgery, this time I decided to steal only the proportions from Mrs. Merkel, step by step. But I want to hold my own pixels on all pictures.

The author


Werkel


Werkel square wise