Active Shape Models with Stasm

Stephen Milborrow

Document version 11 for Stasm version 2.4 with updates for Marki 2.4e

May 9, 2009

Contents

1 Introduction

Stasm is a software package for locating landmarks using Active Shape Models (ASMs). The package comes preconfigured for locating landmarks in faces. It also allows you to build new ASMs. The software is written in C++ and the source code is available under the Gnu Public License Version 2.

Name	**Description**
`Stasm`	locate facial landmarks
`Wasm`	Windows front-end to Stasm
`Minimal`	small program showing how to use AsmSearch() to locate landmarks
`Tasm`	train ASMs
`Fdet`	find faces using Rowley or Viola Jones face detectors
`Marki`	manually landmark images
`Mdiff`	like UNIX diff but ignores text in [brackets]
`summary.R`	R language utility to summarize and plot fits

Figure 1: *Programs in the Stasm package.*

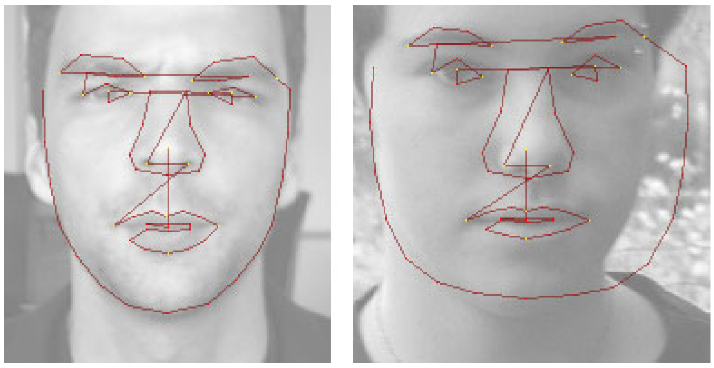

Figure 2: *Good and bad results from Stasm. The left image is from the BioID set [JKF01]*

This document assumes that you would like some understanding of Stasm's internals. If you just want to see Stasm locating landmarks, try the Wasm program (Section 3). To build the executables, see Section 11.2. To call the facial landmark locater from your own program, see Section 11.4.

Figure 1 is a list of executables. Note that Stasm is the name of the package and also of an executable in the package. The context make it clear if we are talking about Stasm the package or Stasm the program.

Some familiarity with ASMs is assumed. Cootes et al. [CT04] or Chapter 2 of [Mil07] are good places to start. Stasm supports some extensions to the classical shape model, as summarized in [MN08].

The `HISTORY.txt` file lists the differences between versions of Stasm.

2 Stasm: finding landmarks

Stasm is a command line utility for finding landmarks.

Stasm writes the results to a text file called `stasm.log`. Stasm also writes an image called `stasm-IMAGE.bmp` to the current directory. If you don't want the image, use the `-if` flag.

2.1 Expectations

Stasm is designed to work on "passport style" photographs i.e. on front views of upright faces with neutral expressions. You will often see poor fits on faces at angles or with expressions such as smiling mouths. Like all automatic techniques at the current time it is not as accurate as a human landmarker and will sometimes make quiet bad location errors (Figure 2).

2.2 Hints

The face should be at least a quarter of the image wide. Faces smaller than that are ignored by (Stasm's configuration of) the Viola Jones detector. If there are multiple faces in the image, Stasm uses the most central face.

Stasm scales the face width internally to 180 pixels. Large images with faces much wider than that slow down the face detector search without improving landmark accuracy.

Stasm knows how to process JPEGs, BMPs, PGMs, and PPMs. It uses the file extension to determine the type of image. JPEGs are smaller than PGMs but reading them is slower, which usually only matters if you are processing large numbers of images. BMPs are both bigger and slower than PGMs.

2.3 Stasm flags

Stasm is invoked as

```
    stasm [FLAGS] IMAGE [IMAGE2 ...]
```

where the optional flags are

-r use the Rowley face detector (default is Viola Jones)

Before the search for the landmarks begins, Stasm must find the position of the face in the image using a global face detector. The default is the Viola Jones detector (but was Rowley prior to Stasm version 1.7).

Stasm uses the OpenCV implementation of the Viola Jones detector [VJ01] [LM02]. The "slow but accurate" settings are used.

The Rowley detector [RBK98] is slower and sometimes fails to find the face, but often gives slightly better fits.

-i[sSfF] write images (default is `-iF` meaning write image showing final shape)

`-iS` start shape
`-if` no final shape i.e. generate no images.

-c FILE.conf config file name

Tell Stasm what parameters and ASM model(s) to use by invoking config file(s) with the `-c` flag. Config files are described in Section 5. Use `-c` twice to specify stacked models.

The default is
`-c ../data/model-1.conf -c ../data/model-2.conf`
i.e two stacked models: the first using 1D profiles with 68 points; the second using 2D profiles with 84 points.

If you would like to use just 1D profiles, invoke Stasm as
`stasm -c ../data/model-1.conf *.jpg`

```
    file   npoints      start      final
B0000_01        17    0.33296    0.23045
B0001_01        17    0.35287    0.24544
B0002_01        17    0.24411    0.21619
```

Figure 3: *An example "tab" file.*

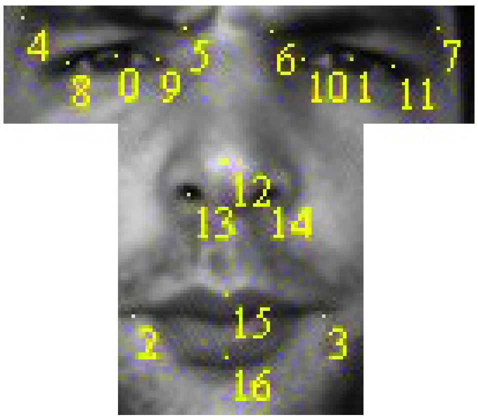

Figure 4: *The me17 landmarks. Note that the landmarks are all internal to the face. The fit is normalized by dividing by the distance between the eye pupils. (This image is from the BioID set.)*

-t TABFILE SHAPEFILE use a reference shape file

This flag has two effects (i) write fits to TABFILE (see the next section) and (ii) use the face detector locations in SHAPEFILE.

Example: `stasm -t test.tab ../data/68.shape *.jpg`.

With this flag, Stasm uses the face detector locations saved in the shape file rather than actually invoking the face detector. It matches the (base of the) image file name against the tag string in the shape file. The advantage is speed, especially when using the Rowley detector.

Stasm will report that it can't find the face if the face detector location is not in the shape file (change that with the config option `fStasmSkipIfNotInShapeFile` described in Section 5.2).

The face detector locations are available in the standard shape files `68.shape` and `84.shape` (Section 4.5). You can put detector locations into a new shape file using Fdet (Section 9).

2.4 Measuring the fit

With the `-t` flag, Stasm measures the final fit against the specified reference shape file and writes the results to a "tab" file (Figure 2.3). Fits are measured as the sum of euclidean distances between fitted points and corresponding reference points in the shape file.

File	**Description**
`model-1.conf` `model-2.conf`	Config files specifying which ASM to use, number of eigs, etc.
`model-1.asm`	68 point ASM model with 1D profiles
`model-2.asm`	84 point ASM model with 2D profiles
`haarcascade_frontalface_alt2.xml`	Viola Jones detector data

`*mask*.pgm`
`*.net`
`*.wet`
	Rowley detector data

Figure 5: *Data files read by Stasm when run with the default options.*

The fit is measured on all landmarks unless the config option `fMe17` is set, in which case the `me17` subset of points is used (Stasm's default config files set `fMe17` true, Section 5.1). Following Cristinacce section 6.1.4 [Cri04], the `me17` is calculated on 17 points internal to the face (Figure 4), and the fit for each face is scaled by dividing by the distance between the eye pupils on the reference image. An advantage is that measurements can be made on shapes with different numbers of landmarks, as long the `me17` points appear in both shapes (which is the case for the XM2VTS, AR, and BioID shapes). The measure, like any, is to some extent arbitrary. It ignores, for example, points on the face perimeter.

The R language program `summary.R` in the `tools` directory reads `.tab` files, summarizes them, and plots fits. It prints the min, max, mean, and median fits. (The R web page is `http://www.r-project.org`.)

2.5 Files used by Stasm

Figure 5 shows the files used by Stasm. Data files are stored in the `../data` directory relative to the executable.

3 Wasm: a Windows front-end to Stasm

Wasm is a front end to the Stasm routines. Figure 6 shows a screenshot. Wasm was designed to be a simple downloadable utility for people try out the feature detector.

Wasm has no command line options. It uses the Viola Jones detector and the same config files used by default by the Stasm executable (`model-1.conf` and `model-2.conf`).

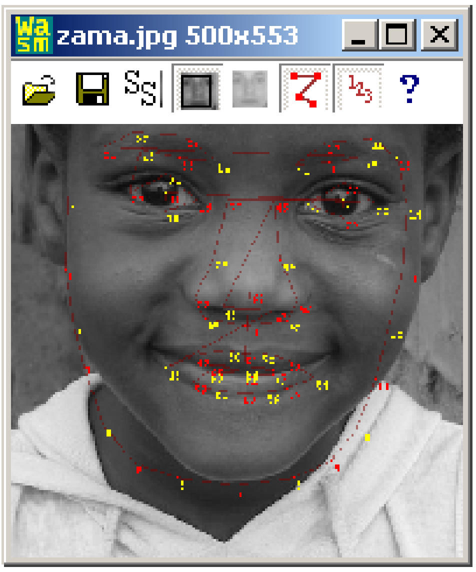

Figure 6: *Wasm: a Windows front-end to Stasm.*

```
ss # first two characters must be "ss", comments are preceded by "#"

# "Directories" is the image search path with each directory separated by a semicolon

Directories /faces/xm2vts;/faces/biod;/faces/ar

"0000 image1" # tag string has attributes (in hex) and image name
{ 68 2        # a mat with 68 points, each with 2 coordinates
  -101.6 6.2
  -103.3 -33.9
...
}
"0000 image2"
{ 68 2
  -101.6 6.2
  -103.3 -33.9
...
}
...
"1000 image1"  # Viola Jones locations have attributes 0x1000
{ 1 4          # a vector with 4 elements (x, y, width, and height of the face box)
  8 -24 279 279
}
...
"2000 image1"  # Rowley locations have attributes 0x2000
{ 1 8          # x, y, width, height, xleye, yleye, xreye, yreye
  12 -46 257 257 -50 8 59 5
}
...
```

Figure 7: *An example shape file. This example includes saved face detector locations, which are optional in a shape file.*

BadImage	`0001`	image is ”bad” in some way (blurred, face tilted, etc.)
Glasses	`0002`	face is wearing specs
Beard	`0004`	beard including possible mustache
Mustache	`0008`	mustache but no beard occluding chin or cheeks
Obscured	`0010`	faces is obscured e.g. by subject’s hand
EyesClosed	`0020`	eyes closed (partially open is not considered closed)
Expression	`0040`	non-neutral expression on face
NnFailed	`0080`	Rowley search failed (a rough measure of face quality)
Synthesize	`0100`	synthesize eye points from twin landmark
ViolaJones	`1000`	Viola Jones detector results
Rowley	`2000`	Rowley detector results

Figure 8: *Shape attribute bits* (`atface.hpp`).

4 Shape files and tag strings

Shape files are a central part of the Stasm software. Figure 7 shows an example.

The `Directories` string at the top of the file specifies an image search path with each directory separated by a semicolon. You should change the string in the standard shapes files `68.shape` and `84.shape` for your environment.

The *tag string* before each shape identifies the image attributes and name.

4.1 Image names in shape files

File name extensions (such as `.jpg`) should not be used in tag strings (the software will issue a warning when it reads the file). When looking for a file matching a tag, the software searches for a matching file name with any of the standard image suffixes in the directories listed in `Directories`. The “standard images suffixes” are `.jpg`, `.bmp`, `.pgm`, and `.ppm`. The search order is unpredictable (well, actually it isn’t, but for efficiency changes dynamically. See `imgiven.cpp`).

This behaviour is different from versions of Stasm prior to 1.7, where file name extensions were stored in the shape files. To convert an old to a new file, remove the file extensions in a text editor.

4.2 Shape attributes

Attributes in a tag string are a bit field specified as a four digit hex number. For example, the Viola Jones detector positions have attributes of `1000`. See Figure 8.

If you are creating a new shape file, attributes (except `ViolaJones` and `Rowley`) (for saved face detector shapes) are entirely optional and only necessary if you want to train on subsets of images defined by attributes. The standard shape files `68.shape` and `84.shape` supplied with Stasm (Section 4.5) are “fully tagged” with the bits in Figure 8.

4.3 Shape coordinates

Stasm uses the following coordinate system. The center of the image is [0,0]. As x increases, you move to the right; as y increases, you move up the image. In other words, they are standard Cartesian coordinates centered on the center of the image.

To convert Stasm coordinates to coordinates with x,y at the top left image corner:

```
x = x + ImageWidth / 2
y = ImageHeight / 2 - y
```

See `CONVERT_TO_OPENCV_COORDS` in `stasm/main.c` for an example. See `fViolaJonesFindFace()` for conversion in the other direction.

4.4 Unused landmarks

An unused or uninitialized landmark has a “position” in a shape matrix with both x and y equal to 0. The x position of a valid landmark that happens to be at [0,0] is thus jittered by Marki to 0.1 (a one tenth of a pixel offset).

Unused landmarks are for more esoteric applications of Tasm where landmarks are synthesized or models are built with different sized shapes. It is unlikely that you will need unused landmarks when creating your own shape files.

4.5 68.shape and 84.shape

The principal shape files used by Stasm are `68.shape` and `84.shape`. Section 8 shows how to use Marki to examine landmarks in these files.

Modify these files for your environment by changing the directories in the `Directories` string.

`68.shape` specifies the following:

1. The manually landmarked positions for the XM2VTS, AR, and BioID images. The landmarks from the sites listed below were converted to Stasm coordinates and then inserted into the shape file (Section 4.3):
   `http://www-prima.inrialpes.fr/FGnet/data/05-ARFace/tarfd_markup.html`
   `http://www-prima.inrialpes.fr/FGnet/data/07-XM2VTS/xm2vts_markup.html`
   `http://www-prima.inrialpes.fr/FGnet/data/11-BioID/bioid_points.html`
   A big thanks goes out to the people who did the manual landmarking and made the results available.

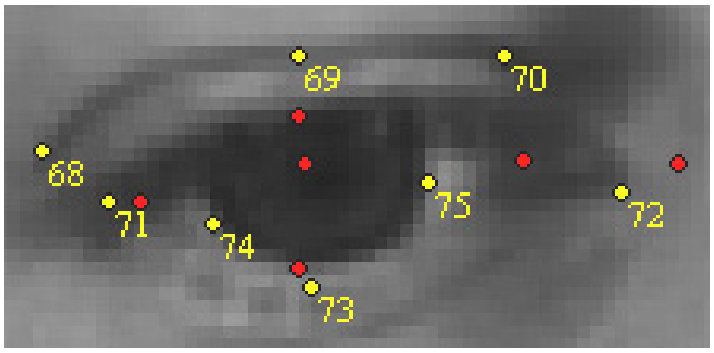

Figure 9: *Left eye with new landmarks in yellow, existing XM2VTS landmarks in red.*

The BioID names in the file have the form `Bxxxx_ii` where `ii` is the individual number (`01` to `27`). The individual numbers were added as part of the Stasm project, so the names are not exactly the standard BioID names. The XM2VTS names are prefixed with an "m".

2. The landmark positions for the mirrored XM2VTS, AR, and BioID images (Section 4.6). The names of these images have `r` (for "reversed") as the second character.

3. The global face positions discovered by the Rowley and Viola Jones detectors. These positions were generated by running Fdet and pasting the output into `68.shape`, as described in Section 9. The face positions are used by Stasm with the `-t` flag as described in Section 2.3, and by Tasm as described in Section 7.3.

`84.shape` is identical to `68.shape` except that the XM2VTS shapes have been extended to 84 points. We call these the "extended" XM2VTS landmarks. For most of these shapes, the extra points are set to [0,0], meaning unused. However, 723 of the shapes have extra eye landmarks (Figure 9). See `landmarks.hpp`.

Actually only the extra *left* eye landmarks were manually landmarked. The extra right eye landmarks remain at [0,0] in the shape file and are synthesized by Tasm by symmetry from the left eye landmarks (if the config option `fSynthEyePoints` is set, which it always must be to use XM2VTS shapes in `84.shape` successfully).

4.6 Mirrored images

If you are creating you own shape file, you can double the number of training images by mirroring the existing images. That saves a lot of extra manual landmarking. Mirrored images can be mechanically generated in any image editing program such as the GIMP, ImageMagick, or Photoshop.

By convention in the Stasm software, the second character of the file name of mirrored images is `r` (for "reversed"). Thus `B0000_01.jpg` becomes `Br0000_01.jpg`.

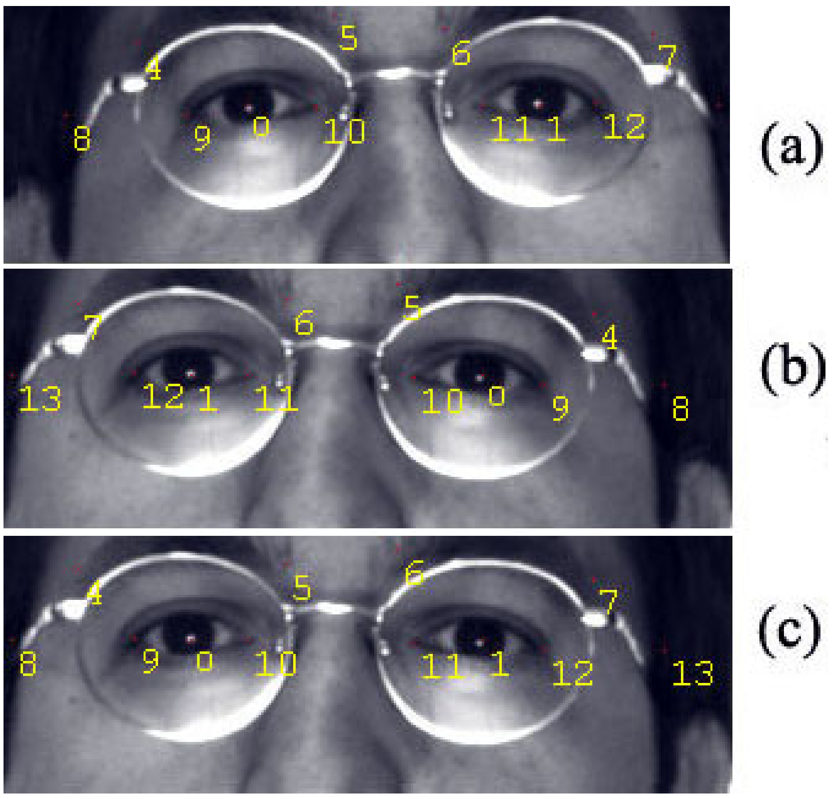

Figure 10: *Mirroring an image:*
a) Original image with landmarks
b) Mirrored image. Landmarks mirrored but not renumbered – wrong!
c) Mirrored image. Landmarks mirrored and correctly renumbered.
The image is from the AR set [MB98].

You will need to add the mirrored landmarks to the shape file. One way of doing that is illustrated by the utility `mirror-ar.awk` in the `tools` directory. This generates mirrored landmarks for AR shapes by reading the existing AR landmarks in `68.shape`. There are two steps (`mirror-ar.awk` does both):

1. negate the x coordinates (Stasm coordinates are defined in Section 4.3)
2. change the order of the points i.e. move them to their mirrored positions in the shape array (Figure 10). It is easy to forget this step, which really messes up the shape model. Marki is useful for checking.

5 Config files

Stasm and Tasm are configured by reading config files holding *config options*. Figure 11 shows an example config file. Nearly all options have defaults and an entry in a config file is only

```
STASM1 # first word in file must be STASM1 or TASM1
{
sAsmFile "model-1.asm"
fMe17    1   # use me17 measure for fitness
nEigs    20  # nbr of shape model eigs to use
}
```

Figure 11: *An example config file for Stasm.*

Type	Naming convention	Example
String	`s` prefix	`sShapeFile "68.shape"`
Integer	`n` prefix	`nEigs 22`
Boolean	`f` prefix (for "flag")	`fPrescaleBilinear 1`
Hex	none	`AttrMask0 0x00f1`
Real	none	`MaxShapeAlignDist 1e-6`

Figure 12: *Config option types.*

needed if the default is not correct. Parameters can appear in any order in the config file. If a parameter appears twice, the last occurrence is used. By convention, config file names for Tasm begin with `tasm-`.

Some of the options used by Tasm must be passed on to Stasm (for example `NormalizedProfLen`). Tasm writes these options into the ASM file so Stasm can read them.

Stasm also reads its own options from separate config file(s). By default, Stasm uses a stacked model defined by `model-1.conf` and `model-2.conf`. Change that with the `-c` flag.

Figure 12 shows the different types of config option. In the sources, config options are prefixed with `CONF_`. Ugly but clear.

In versions prior to 1.7, the Stasm package used `#defines` in the source code instead of config options. This caused confusion — you were often not sure which options were used for the particular version of your executable, or if your Stasm matched your Tasm (which is not to say that config files completely resolve such issues).

5.1 Common Tasm options

This section briefly describes some Tasm options. See `tasm.cpp` for more options and details. The effect of most of these options is evaluated in [Mil07]. This section is perhaps best read after the section describing Tasm (Section 7).

sShapeFile The shape file. No default.

sTagRegex Tasm only loads shapes where the file name in the tag string matches `sTagRegex`, which is an egrep-style case-insensitive regular expression. (The tag string is the string before the shape in the shape file, see Section 4). For example (note the space in `" B"`):

```
sTagRegex " B"
```

matches only BioID shapes (including mirrored BioID shapes). Section 8.3 has further examples.

The default value is `""`, which matches all shapes. The config files used to build `model-1.asm` and `model-2.asm` have this option set to `" m"` i.e. match XM2VTS files.

AttrMask1, AttrMask0 Tasm only uses shapes that satisfy

```
(Attr & AttrMask0) == AttrMask1
```

where `Attr` is the hex number at start of the tag string for the shape. `AttrMask0` and `AttrMask1` are specified in the shape file as hex numbers (Figures 7 and 8). Section 8.3 has examples. The defaults `0x0` and `0x0` match all shapes. If `sTagRegex` is set, then both it and the attribute masks apply.

The config file used to to build `model-2.asm` has these options at `0xf1` and `0x0` to exclude faces with eyes closed and so on.

nLevs The number of pyramid levels. Default is 4. This is one of a host of options not explicitly mentioned here that specify basic parameters of the ASM model. See `tasm.cpp`.

nMaxShapes Default is 0. If not 0, specifies the number of shapes to use for training (after filtering by `sTagRegex`, `AttrMask1`, and `AttrMask0`). If the config option `nSeed_SelectShapes` is 0, Stasm uses the first `nMaxShapes`; else it selects a random sample (without replacement) with a random seed of `nSeed_SelectShapes`. See `ReadSelectedShapes()`.

nLev2d Tasm generates 2D profs for pyramid levels less than or equal to `nLev2d`. Pyramid level 0 is full size; level 1 is half size (assuming the config option `PyrRatio` is at its default of 2); and so on. The default `nLev2d` is -1 meaning no 2D profiles i.e. all 1D profiles.

ProfType Profile type for 1D profiles. Currently only "Classic Cootes" profiles are supported i.e. take the difference between this pixel and the next, and normalize by dividing by the sum of absolute values in the profile. See `prof.hpp`.

The Masm research code supported a number of other 1D profile types, but they were removed in the interest of simplicity. Add other profile types for your application by lifting code from the Masm sources, or you could add your own profile types (possibly a good research direction).

ProfType2d Profile type for 2D profiles ("profile" is a slight misnomer here). Currently only the following 2D profile type is supported: sum the difference between this pixel and the pixel to the right and the difference between this pixel and the pixel below it, then normalize by dividing by the sum of absolute values in the profile using a small amount of sigmoid normalization. See the above remarks about the Masm code.

nStandardFaceWidth Default is 180 pixels. If non zero, faces are scaled (by both Tasm and Stasm) so their width is `nStandardFaceWidth`. The start shape is used to estimate the width of the face. The "width" is the horizontal distance between the left- and the right-most landmarks.

fXm2vts Default is false. If set, treat landmarks as being numbered using the standard or extended XM2VTS numbering scheme. "Extended" XM2VTS numbering means that the first 68 landmarks use the standard XM2VTS numbering, and the remaining landmarks (up to 84, all additional eye landmarks) are numbered as laid out in `landmarks.hpp`.

This option itself has no effect but must be true to allow certain other options to be set. These are the options which need to know what part of the face corresponds to a given landmark. Such options include the Tasm options `fSynthEyePoints` and `fUnobscuredFeats` and the Stasm options `fMe17`, `nVjMethod`, and `nRowleyMethod`.

fSynthEyePoints Default is false. Requires `fXm2vts`. Use symmetry to synthesize missing right eye landmarks (at indices greater than 68) from left eye landmarks. See the description of `84.shape` in Section 4.5 and `eyesynth.cpp`.

fUnobscuredFeats Default is false. Requires `fXm2vts`. If set, during training Tasm uses a landmark from an image only if that landmark is not obscured in that image. Thus if set, eye landmarks for example will not be used in faces that are wearing glasses.

nWhich2d Default is 0. In a pyramid level that supports 2D profiles, `nWhich2d` specifies which landmarks are actually 2D. Can take values `All=0`, `InternalExceptMouth=1`, `Internal=2`, `Eyes=3`, or `EyesExt=4` (for precisely what these mean, see `GetGenProfSpec()`). Only 0 is allowed unless `fXm2vts` is set.

fTasmSkipIfNotInShapeFile Default is false. This applies to the phase of Tasm that builds a transform from the Viola Jones frame (and the Rowley frame, in an independent phase) to the positions of the training faces (using a technique submitted by GuoQing Hu). Tasm searches first in the given shape file for a saved face detector location. If the location is not in the file, Tasm then invokes the face detector, unless `fTasmSkipIfNotInShapeFile` is true.

The idea with this flag true is that if you have already put all possible face detector locations for the training images in the shape file (using Fdet), then it is pointless and time consuming to re-invoke the face detector. "All possible" means that only face locations in the training set where the face detector failed are missing from the shape file. The standard shape files `68.shape` and `84.shape` include all possible detector locations for the XM2VTS, AR, and BioID images and their mirrors.

See also the Stasm option `fStasmSkipIfNotInShapeFile`.

nSleep Described in Section 5.2.

5.2 Common Stasm options

This section briefly describes some Stasm options. See `initasm.cpp` for more options and details. Stasm inherits some options from Tasm via the ASM file and these are not mentioned below (see the header of any `.asm` file to see what they are).

sAsmFile The ASM file. No default.

`sAsmFile` is a relative path name (relative to the directory holding the config file) unless the entry is prefixed with `/` or `./` indicating that the path name should be used as is. When testing new models, it is **easy to use the wrong ASM file**. Stasm prints out which file it is using and it is sensible to check that.

nEigs Specify the number of eigenvectors to use in the shape model. The default is one third of the eigenvectors, which will be wrong for most applications, but there is no simple correct default.

There are related options to loosen up the shape model at specific points in the search (`nEigsLev0`, `nEigsLev0_2d`, and `nEigsFinal`). See the sources and [Mil07] for details.

BMax Default is 1.8. Specifies the limit "b" in the shape model.

There are related options to loosen up the shape model at specific points in the search (`BMaxLev0`, `BMaxLev0_2d`, and `BMaxFinal`).

nPixSearch Default is 3. Specifies how far along the whisker to search (along each side of the whisker) for 1D profiles.

nPixSearch2d Default is 2. Like `nPixSearch` but for 2D profiles, Applies to both the x and y directions.

nMaxSearchIters Default is 4. Specifies the maximum of iterations in the ASM search at each pyramid level. See the related option `nQualifyingDisp` in the source code.

nMaxSearchIters2d Default is 4. Like `nMaxSearchIters` but for pyramid levels with 2D profiles.

n2ndModelStartLev Pyramid level at which the second of the stacked models starts its search (assuming stacked models are being used). Default is 1 (i.e. one level below full size). Experiments show that it is pointless starting the second model at the bottom of the image pyramid (it is slower and results are not better).

VjScale Default is 0.9. Stasm downscales the Viola Jones face box by this amount before using it to find the location of the start shape.

The idea is that when positioning the start shape, it is better to err in making the start shape too small than too big (because profiles matches inside the face are more reliable than those outside the face). Measurements on a validation set (not the training set) show this to be true.

RowleyScale Same as `VjScale` but for the Rowley detector. Default is 1.2.

nRowleyMethod Default is 1 (in the code, but set 0 in the default Stasm config files). Requires (the Tasm option) `fXm2vts` to be set in the ASM file.

If 0, use the hand crafted method of fitting the start shape to the Rowley face detector. The hand-crafted method uses the detected eye as well as the face positions. It is a bit of a hack but does give better results.

nVjMethod Default is 1 (in the code, but set 0 in the default Stasm config files).
Like `nRowleyMethod` but for the Viola Jones detector.

fMe17 Default is false (in the code, but set true in the default Stasm config files). Only applies if Stasm's `-t` flag is used. Requires (the Tasm option) `fXm2vts` to be set in the ASM file.

If the Stasm flag `-t` is used, Stasm measures the fit of the landmark points against the manually landmarked fits saved in the shape file.

If `fMe17` is false, all points are used when measuring the fit.

If `fMe17` is true, the subset of 17 points defined by the `me17` measure is used to measure the fit.

An advantage is that fits can be measured on shapes with a number of landmarks different from those in the training set. See Section 2.4 for a description of `me17`.

fStasmSkipIfNotInShapeFile Default is true. Only applies if Stasm's `-t` flag is used.

When the `-t` flag is used, Stasm searches first in the given shape file for the saved face detector location for the current image. If the location is not in the shape file, Stasm prints a warning and moves on to the next input image. However, if `fStasmSkipIfNotInShapeFile` is false, Stasm will instead call the face detector if a location is not found in the shape file. See the Tasm option `fTasmSkipIfNotInShapeFile` for a rationale.

nSleep Default is 10 milliseconds. Stasm and Tasm are system hogs, but do at least "sleep" periodically to give other processes a chance. A non-zero `nSleep` makes the overall system more reponsive when searching many images, but slows down the Stasm search slightly. The default 10 milliseconds makes searches about 10% slower (on a 1.5 GHz machine), but allows you to read the internet, say, while waiting for Stasm to process many images. Set `nSleep` to 0 for the fastest possible execution. On the other hand, you may want to sprinkle in a few more calls to `ReleaseProcessor()`.

6 ASM files

The term *ASM file* refers (in the Stasm context) to a file defining an ASM model created by Tasm. We use the terms "model file" or "`.asm` file" to mean the same thing. Model files are text files (as are all data files in the Stasm package) and thus can be examined in any text editor. Figure 13 shows the structure of a model file.

6.1 Converting old ASM files

The current ASM file format is different from that in versions of Stasm prior to 1.7. The tools directory has a utility to convert old format files to new. It cannot generate the `RowleyAv` and `VjAv` shapes, so generates dummies for these. As a consequence, if necessary Stasm will issue a warning and force the config options `nVjMethod` and `nRowleyMethod` to 0. Example conversion:

```
awk  -f asm0to1.awk  stasm15/stasm/data/model-1.asm  >converted-model-1.asm
```

```
ASM1              # first word must be ASM1, where 1 is the ASM file version
# Command: tasm -o ...  # comment showing how Tasm was invoked
{                 # Tasm config options made available for Stasm
  nPoints 68
  fXm2vts 1
  ...
}
# comments here list the options used by Tasm
# ...
# ...

"EigVals"
{ 168 1
  1183.43
  ...
}
"EigVecs"
{ 168 168
  0.0349382 0.15091 -0.212292 0.110971 -0.0659393 ...
  ...
}
"AvShape"         # mean shape after alignment
{ 68 2
  -89.2    41.3
  ...
}
"VjAv"            # mean shape in VJ frame, for generating start shape
{ 68 2
  -79.1    20.3
  ...
}
"RowleyAv"        # mean shape in Rowley frame, for generating start shape
{ 68 2
  -96.5    51.5
  ...
}

Lev 3             # data for pyramid level 3 follows (which in this case is lev 3)
"ProfSpecs Lev 3" # profiles types for level 3, bit fields as a hex number
{ 68 1
  41              # 41 is a "classic Cootes" 1D profile, see prof.hpp
  ...
}
"Covar Lev 3 Point 0"
{ 17 17
  1231 561 353 342 369 382 ...
  ...
}
...
"Lev 3 Prof 0"
{ 1 17 -0.0216047 0.0771605 ...
}
...
...               # other pyramid levels follow, level 0 (full size) is last
```

Figure 13: *The structure of an ASM file.*

File	**Description**
`tasm-example.conf`	Config file specifying which shape file to use, 1 or 2D profiles, etc.
`example.shape`	Shape file specified by `tasm-example.conf` This file contains manual landmarking information and possibly saved face detector locations
`haarcascade_frontalface_alt2.xml`	Viola Jones detector data
`*mask*.pgm` `*.net` `*.wet`	Rowley detector data

Figure 14: *Data files read by Tasm with* `tasm ../tasm-example/example.shape`

7 Tasm: building ASMs

Tasm builds ASM models. Tasm creates a `.asm` file and a log file `tasm.log`. It reads

1. A `.conf` file which specifies the shape file and other parameters.
2. A `.shape` file which specifies the manual landmarks
3. The set of images listed in the shape file (or a subset of those images, depending on the config options).
4. Data files for the global face detectors. (Tasm uses the face detectors to generate a transform between the detector box and the mean shape. With `nVjMethod` or `nRowleyMethod` set to 1, Stasm uses this transform to generate a start shape.)

Figure 14 show the files used by Tasm.

7.1 Tasm flags

Tasm is invoked as

```
tasm [-o OUTFILE.asm] FILE.conf
```

This creates the model file `OUTFILE.asm` as specified by the config file `FILE.conf`. The `-o` flag specifies the output file name (the default is `temp.asm`).

The first shape in the shape file (that matches `sTagRegex`, `AttrMask1`, and `AttrMask0` if used) is taken as the reference shape. The reference shape is used as a base for shape alignment when building the shape model (in `tasmshapes.cpp`).

7.2 Examples using the standard shape files and XM2VTS images

It is easiest to get started with Tasm by looking at a few examples. These examples require XM2VTS images. If don't have those, you can run the examples in the next section instead.

You will need to edit the `Directories` string in `68.shape` and `84.shape` for these examples to work in your environment: The file paths in the examples assume you are working in the `linux`, `msoft`, `vc6`, or `mingw` directory.

(i) To rebuild `model-1.asm` (1D profiles, 68 XM2VTS points).

```
tasm -o model-1.asm ../data/tasm-68-1d.conf
```

(ii) To rebuild `model-2.asm` (2D profiles, XM2VTS points extended to 84 points):

```
tasm -o model-2.asm ../data/tasm-84-2d.conf
```

This uses the "extended" XM2VTS shapes (Section 4.5). and thus the config option `fSynthEyePoints` is set true in `tasm-84-2d.conf`.

(iii) Say you want to build and use a version of `model-2.asm` that uses 2D profiles for all the interior points of the face (`model-2.asm` uses 1D profiles for the mouth points). The steps are:

1. Copy `../data/tasm-84-2d.conf` to, say, `tasm-test.conf` and change `nWhich2d 1` in the file to `nWhich2d 2` (described in Section 5.1).
2. Build a new ASM file: `tasm -o test.asm tasm-test.conf`
3. Copy `../data/model-2.conf` to, say, `test.conf` and change `sAsmFile "model-2.asm"` in the file to `sAsmFile "test.asm"`.
4. Run Stasm with the original 1D ASM file and your new 2D file:
   `stasm -c ../data/model-1.conf -c test.conf ../data/test-image.jpg`

In our tests this does not perform as well as the standard `model-2.asm`. Our tests were based on the AR set and the `Me17` (which ignores points on the edge of the face) so your results may differ. It also runs a little slower (because it has more 2D profiles, which are slower than 1D profiles).

(iv) Other examples:

```
tasm -o 68-2d.asm ../data/tasm-68-2d.conf  # 68 point model with 2D profiles
tasm -o 84-1d.asm ../data/tasm-84-1d.conf  # 84 point model with 1D profiles
```

(v) The makefile has further examples that are used for testing, for example

```
make test-varied
```

7.3 An example using non XM2VTS shapes

The directory `tasm-example` contains files to build a 12 point model from 20 images. The images needed to run this example are included in the Stasm package (they are from the BioID set [JKF01]). The file paths in the example assume you are working in the `linux`, `msoft`, `vc6`, or `mingw` directory.

Look at the landmarks with Marki:

```
marki ../tasm-example/example.shape
```

Build a new ASM file with Tasm using `example.shape`:

```
tasm -o example.asm ../tasm-example/tasm-example.conf
```

Note that during the "Generating Rowley mean shape" phase, Tasm uses only 19 of the 20 shapes, and prints "`1 no eyes, 4 one eye`", meaning that two eyes were not always found. This does not matter because there are enough successful Rowley searches to build the Rowley mean shape.

Use the created `example.asm` file with the following command (the fit will not be good because there are only 20 training images):

```
stasm -iS -c ../tasm-example/example.conf ../data/test-image.jpg
```

This uses `example.asm` in the current directory because the `sAsmFile` entry in `../tasm-example/example.conf` is `./example.asm` i.e. it has a `./` prefix. See the description of `sAsmFile` in Section 5.2. Check the messages printed by Stasm to make sure you are using the right ASM file.

Build 2D instead of 1D profiles by adding the following line to `tasm-example.conf` (see Section 5.1 on Tasm options):

```
nLev2d 3 # generate 2d profiles for levels 0 to 3 (i.e. all levels)
```

In this example the shape file is already built (although with only 20 shapes — not enough for good results). If you are starting from scratch you will need to build the shape file first, typically with Marki (Section 8).

You will notice that Tasm is quite slow during the "Generating Rowley mean shape" phase. That is because Tasm has to invoke the Rowley detector for each image (and likewise for the Viola Jones detector, which is faster so the slowdown not so noticeable). Speed up this phase by pre-storing the face detector locations in `example.shape`. Use Fdet to do that as described in Section 9.

7.4 ASMs for objects other than faces

The Stasm package been used thus far only for locating features in faces (as far as we know), but it can "easily" be used for other objects.

Summarizing, there are two important aspects (i) using a detector for your object to replace the Viola Jones detector (requires a little understanding of the Stasm code) and (ii) manually landmarking your training images (easy but requires patience and coffee).

You first need to obtain an object detector for your object (analogous to the Viola Jones face detector). One possibility of many is [HJLM07]. Modify `fFindDetParams()` in `startshape.cpp` by putting in your object detector.

Tasm and Stasm assume that there are two object detectors, viz. Viola Jones and Rowley. It is easiest to keep the ASM file structure the same, at least initially, so replace the Viola Jones detector with your new detector but provide a dummy Rowley detector (which could just duplicate the new detector).

If your object detector gives the `x`,`y` position with `width` and `height`, then without changes Tasm can generate the data needed for Stasm to position the start shape. That is done in `detav.cpp` — it generates `VjAv` and `RowleyAv` in the ASM file. Set the config options `nVjMethod`, and `nRowleyMethod` to 1. For best results you might need to adjust the config option `VjScale` (typically by testing on a validation set).

To build new shape and config files, it is perhaps easiest to start with the model in the `tasm-example` directory and modify that (Section 7.3).

Manually landmark your training images with Marki or the program on Tim Cootes' site.

You will have to decide if the profile types supported by Tasm suffice for your objects. See the description of `ProfType` and `ProfType2d` in Section 5.1.

Config options such as `nPixDist` may need adjustment because the defaults have been optimized for faces.

We would be interested in any projects along these lines. The Stasm web page has an email address.

8 Marki: manually landmarking images

Marki is a Windows program to manually landmark images and to look at landmarks. A screen shot is shown in Figure 15. Users have reported that the landmarking tool on Tim Cootes' site also works well.

On bootup, Marki reads a shape file. (A shape file consists of a list of images with their associated landmarks, as described in Section 4.) A simple example:

```
    marki ../tasm-example/example.shape
```

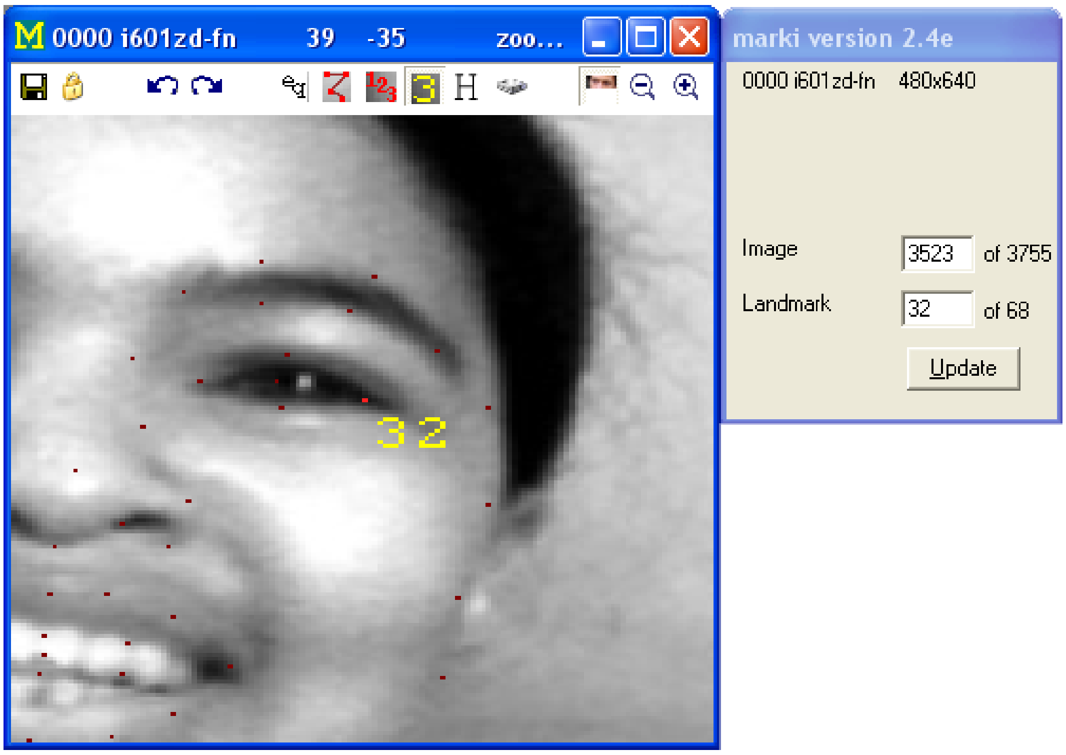

Figure 15: *Marki: a tool for manual landmarking.*

The first shape (that matches the `-p` and `-P` flags, if used) becomes the reference shape. Shapes that have a number of points different from the reference shape are ignored with a message. In Marki the reference shape is in all other aspects treated just like any other shape.

Figure 16 shows Marki's command line flags. Use `marki -?` to get the current list of command line flags. You can stipulate which landmark to edit, and so on.

Marki prints progress messages to a log file `marki.log`. It also prints the messages to the screen if you start Marki from the command prompt (recommended, so you can see the messages).

Things can easily go wrong when manually landmarking images. It is important that you make frequent backups of the shape file. Do this outside of Marki, and keep more than one level of backup.

Our experience has been that you should preprocess the images for manual landmarking — equalize the original images and possibly blur them slightly to get rid of scan lines — that makes manual landmarking easier.

8.1 Using Marki

Figure 17 describes the buttons at the top of the Marki window.

```
Usage: marki [FLAGS] FILE.shape\n

-B      Show all buttons (allows you to change current
        landmark number by clicking on buttons)

-F      Fresh start (ignore saved registry entries for window positions etc.)

-l LANDMARK
        Landmark to be altered by left mouse click
        Default: landmark used last time

-o OUTFILE.shape
        Output filename
        Default: FILE.shape (i.e. overwrite the original file)

-p PATTERN
        PATTERN is an egrep style pattern (not a file wildcard)
        Load only shapes in FILE.shape with tags matching case-independent PATTERN
        Example: -p "xyz" loads filenames containing xyz
        Example: -p " m000| m001" loads filenames beginning with m000 or m001
        Default: all (except face detector shapes)

-P Mask0 Mask1
        Load only shapes which satisfy (Attr & Mask0) == Mask1
        Attr is the hex number at start of the tag, Mask0 and Mask1 are hex
        This filter is applied after -p PATTERN
        Example: -P 2 2 matches faces with glasses (FA_Glasses=2, see atface.hpp)
        Example: -P 2 0 matches faces without glasses
        Default: no filter (Mask0=Mask1=0)

-Q      Quiet (no sound when wrapping from last image to first)
```

Figure 16: *Marki command line flags.*

`Save`	Save landmarks to the shape file
`Lock`	Lock (disables left mouse click)
`Undo`	Undo previous actions (undo depth is unlimited)
`Redo`	Redo previous actions, i.e., undo undo
`Equalize`	Equalize on-screen image. Keyboard: e (or right-mouse-click)
`ConnectDots`	Connect the dots. Keyboard: d
`ShowNbrs`	Show landmark numbers
`ShowCurrentNbr`	Show current landmark number. Keyboard: n
`HideLandmarks`	Hide original landmarks. Keyboard: h
`WriteImage`	Write the displayed image (will create a unique file name)
`Crop`	Crop on-screen image. Keyboard: c
`Zoom out`	Less detail. Keyboard: -
`Zoom in`	More detail. Keyboard: +
`AutoNextImage`	Move to the next image automatically after a left mouse click
`PrevImage`	Previous image. Keyboard: PageUp
`NextImage`	Next image. Keyboard: SPACE or PageDown
`AutoNextPoint`	Move to the next landmark automatically after a left mouse click
`PrevPoint`	Previous landmark (only if -B flag was used)
`NextPoint`	Next landmark (only if -B flag was used)
`JumpToLandmark`	Auto jump to landmark (only if -B flag was used)
`Help`	Help

Figure 17: *Marki buttons, in the order they appear above the image window.*

```
SPACE       next image
PgDown      next image         | PgUp        previous image
Home        first image        | End         last image

Following only available if -B flag was used:
RightArrow  previous landmark  | UpArrow     previous landmark
DownArrow   next landmark      | LeftArrow   next landmark

Hold Ctrl at same time to repeat above commands 10 times

c          crop image
+ and -    zoom cropped image
e          equalize image
[ and ]    change amount of equalization
d          connect the dots
n          show current landmark number
h          hide original landmarks
Ctrl-Z     toggle undo/redo
```

Figure 18: *Marki key assignments.*

Figure 18 shows the keys that you can use in Marki. Click on the `Help` button to get the latest list. For example, `PageDn` moves to the next image. `Ctrl-PageDn` skips 10 images ahead.

Change the position of a landmark with a left mouse click (see below).

Cycle forwards and backwards through images in the shape file with the buttons, keystrokes, or the mouse wheel. You can also choose an image or landmark using the dialog window. The attributes in the tag string of the current shape are shown in the dialog window.

Change the current landmark number with the dialog window.

Or if the `-B` command line flag was used, you can change the landmark number with buttons and arrow keys. To minimize mishaps, this is not the default.

Or click on the JumpToLandmark button to dynamically select the landmark nearest the mouse pointer. This is intended to allow easy touchups to a set of images and is currently experimental.

Save the modified landmarks to a new shape file using the `Save` button. Note by default that this will **overwrite** the original shape file (this is different from versions of Marki prior to 2.4a). If you don't want that, change the output shape filename with the `-o` command line flag. On exit Marki will ask if you want to save the landmarks if you have not already done so.

Clicking on the `Save` button also locks the landmark positions — click on the `Lock` button to unlock. The idea is to minimize the effects of incorrect mouse clicks if Marki is left unattended.

Be careful: only the shapes that were read in are saved. If the `-p` and `-P` flags are used, this will be a subset of the shapes in the original shape file. Use a text editor to merge the shape files.

Marki remembers the current image number, current landmark, screen layout, and button settings. The next time you use Marki you will start where you left off. (Marki uses the Windows registry entry `HKEY_CURRENT_USER/Software/Marki/Config`.) Override the automatic return to the same landmark number with the `-l` flag. Or revert to all the defaults with the `-F` flag (for "fresh").

8.2 The mouse

Left click changes the position of the current landmark.

The current landmark is shown in red and the others in dull red. After the click, the new position is shown in cyan and the old in orange. Undo the click (and just about anything else) with the `Undo` button.

If the button `AutoNextImage` is set, Marki will automatically move to the next image after the click. This is useful when you want to move "laterally", doing just one landmark in each image.

If `AutoNextLandmark` is set, Marki will move instead to the next landmark after the click. This is useful when you want to do all landmarks in one image before moving on to the next.

AutoNextImage and AutoNextLandmark are mutually exclusive. Our experience is that you should use AutoNextImage to manually landmark a set of images most quickly and consistently

Mouse clicks are ignored if the current shape is a face detector shape.

Right click toggles equalization The amount of equalization is determined by the -e command line flag, or can be changed with the [and] keys.

The wheel moves backwards and forwards through the images.

8.3 The -p and -P flags

Specify a subset of shapes in the shape file with Marki's -p and -P flags. These flags take the same arguments as the Tasm config options sTagRegex, AttrMask0, and AttrMask1 (Section 5.1 and atface.hpp).

To read the BioID shapes in 68.shape use (note the space in " B"):

```
marki -p " B" ../data/68.shape
```

Using " b" instead of " B" would give the same result, because the match against the -p string is case insensitive.

The above example includes the mirrored BioID shapes (which have names of the form Brxxxx_xx, the second character is "r"). Since the -p string is a regular expression, exclude the mirrored BioID shapes with

```
marki -p " B[^r]" ../data/68.shape
```

In the following example, the reference shape is the first XM2VTS shape (because the XM2VTS shapes happen to be first in 68.shape). Thus the AR and BioID shapes in 68.shape will be ignored (because they have a different number of points to the reference shape).

```
marki ../data/68.shape
```

Other examples

```
marki -p " a"              ../data/68.shape  # AR faces
marki -p " B" -P 2 2       ../data/68.shape  # BioID faces with glasses
marki -p " B" -P 2 0       ../data/68.shape  # BioID faces without glasses
marki -P 1000 1000         ../data/68.shape  # Viola Jones detector boxes
marki -P 2000 2000         ../data/68.shape  # Rowley detector boxes
marki -p " m" -P 1004 1004 ../data/68.shape  # VJ box for bearded XM2VTS faces
```

8.4 Preparing a shape file for Marki

Marki requires a shape file (Section 4) — you can't mark images that are not in a shapefile.

Shapefiles are text files and so can be created and modifed by hand.

You should create an initial shape file for images that you want to mark. If the rough positions of the landmarks are available then manual landmarking tends to be easier if you use those positions. Generate the approximate positions of the landmarks with Stasm and paste the results from the the Stasm log file into the shape file.

Alternatively, if you are not marking faces, one approach is to create the file using Emacs keyboard macros. Set the landmarks to dummy [x,y] values, for example [10,10], [20,20], [30,30].

Don't use [0,0] for dummy values because that is used to denote an unused landmark, as described in Section 4.4. (Actually all that happens is that Wasm gives warnings and ignores [0,0] points when displaying the shapes.)

8.5 Marki annoyances

In this version of the software, Marki reports errors correctly when reading the input files but does not always recover from them gracefully (it crashes), although Marki is stable once it has booted. The error handling functions that Marki uses were originally written to exit on error for Stasm and Tasm, and the retrofit for a windowed environment is incomplete. It's not a big deal in practice.

9 Fdet: locating faces

Fdet reads images and writes the face positions found by the Viola Jones or Rowley detector (select which one with the `-r` command line flag, output goes to `fdet.log`). Fdet can also write images showing the face positions (use the `-i` flag).

Fdet was used to create the face detector data saved in `68.shape` and `84.shape` as follows (your directory names would differ):

```
    fdet   faces/bioid/*   faces/ar/*   faces/xm2vts/*   faces/mirror/*
```

After running Fdet as above, the Fdet log file `fdet.log` was manually cut and pasted into the shape files.

Use Marki to look at face detector shapes in a shape file (see the examples at the end of Section 8.3 for an example with `68.shape`).

10 Mdiff: comparing text files

Mdiff is a poor man's version of the Unix utility `diff` that ignores text in either input file that is between [and].

The makefiles use Mdiff during testing for comparing files that should be the same apart from [times] enclosed in brackets.

Mdiff prints only the first line of multiple consecutive different lines. It then resynchs and looks for further differences. It prints up to a maximum of to 10 differences.

11 Software

This section touches on a few aspects of the software. As usual, nothing replaces perusing the code itself.

11.1 Building the executables under Linux

The README.txt file in the `linux` directory has straightforward instructions to build Stasm in an Ubuntu environment (basically, you install the jpeg etc. libraries and run make). Building in other Linux environments should be simple but we haven't tried it.

11.2 Building the executables under Windows

There are three Windows environments in which you can make Stasm, each with its own directory:

1. `mingw` gcc — we used GCC 4.2.1-sjlj (mingw32-22)
2. `msoft` Visual C++ 9.0 2008 (free Express edition suffices)
3. `vc6` Visual C++ 6.0

11.2.1 Building from the Windows command line

Here is how to build Stasm from the command line:

1. Install OpenCV from the OpenCV download page:

   `http://sourceforge.net/project/showfiles.php?group_id=22870&package_id=16937.`

2. Adjust the paths in the makefiles for your environment

3. If you are doing a VisualC build, make sure that the Debug and Release directories exist below `msoft` or `vc6`. Some versions of `tar` don't create these directories.

4. In one of the directories above, run `make` in the `msoft` or `vc6` directory. This calls up a batch file.

11.2.2 Building from within the Visual C IDE

Here is how to build Stasm in the Visual C IDE:

1. Install OpenCV from the OpenCV download page:

   `http://sourceforge.net/project/showfiles.php?group_id=22870&package_id=16937.`

2. Copy the OpenCV DLLs to your build directory (which must be `vc6` or `msoft`):

   ```
   copy "C:\Program Files\OpenCV\bin\cv100.dll" .
   copy "C:\Program Files\OpenCV\bin\cxcore100.dll" .
   copy "C:\Program Files\OpenCV\bin\highgui100.dll" .
   copy "C:\Program Files\OpenCV\bin\libguide40.dll" .
   ```

   This step is necessary unless the OpenCV DLLs are aready on your DLL search path.

3. From within the IDE, use "Open Workspace" to open `vc6/stasm.dsw` (for VC 6.0) or `msoft/stasm.sln` (for VC 9.0 2008). Or open `tasm.sln` or `marki.sln` if that is what you want to build.

4. Check that the active project is Release and not Debug (unless you don't want that).

5. "`Rebuild all`" (in VC 6) or "`Rebuild Solution`" (in VC 9). You may get some messages "`could not find the file *.h`". These appear to be harmless.

Caveat: the IDE project files were created by someone who seldom uses the IDE.

11.2.3 Some common build errors

The VisualC 9 IDE sometimes crashes if you use "`Build Solution`". Work around that by using "`Rebuild Solution`" first.

A message like
This application has failed because cv100.dll was not found
means that that you need to copy the missing DLL to the current directory. For other solutions to this problem, see http://msdn.microsoft.com/en-us/library/7d83bc18(VS.80).aspx?ppud=4 or Google "`DLL search path`"

A message like
cp: /Program Files/OpenCV/bin/cv100.dll: The system cannot find the file specified.
means that you need to install the OpenCV libraries.

	VC 6	VC 9	gcc	
Stasm on the BioID set	450	480	380	ms per image
	1.00	1.07	0.84	relative time
Tasm building model-1.asm	250	190	230	seconds
	1.00	0.76	0.92	relative time

Figure 19: *Execution times on a 1.6 Ghz Pentium M. You can expect some variance in these times.*

A VisualC compiler message
warning D9035: option 'GX' has been deprecated
means you are running the VC 9 compiler with the VC 6 makefile.

VisualC linker messages such as
error LNK2001: unresolved external symbol ___security_cookie
or
warning LNK4078: multiple ".CRT" sections found with different attributes
mean that you have mixed VC 6 and and VC 9 objects. After fixing your environment, do a "`make clean`" and try again.

11.3 Comparing the executables

Numerical issues mean that the output of the executables built with gcc differs slightly from those built with Visual C. However, the mean landmark location accuracy is virtually identical.

Figure 19 shows execution times for the various builds. TODO The mingw Tasm time could be improved — see `CollectProf()`.

11.4 Calling the landmark locator from your own code

If you want to call the landmark locater from your own code, take a look at the minimal implementation in `minimal.cpp` (reproduced in Figure 20). This program finds the landmarks in a test image and writes an image showing the results. You will probably also want to read `stasm/main.cpp`.

The Stasm object files must also be linked in — see the makefiles for details. Some Stasm source files may need to be replaced or modified. For example, `err.cpp` handles error reporting and you probably have your own error routines. System dependent code is in `err.cpp` (or `werr.cpp` for windowed apps), `misc.hpp`, `release.cpp` (or `wrelease.cpp` for windowed apps), and `util.cpp`.

Make execution of the minimal program quieter by changing the definition of

```
#include "stasm.hpp"

int main(void)
{
    const char *sImage = "../data/test-image.jpg";

    RgbImage Img(sImage);       // read the image

    SHAPE StartShape;           // dummy arg for AsmSearch
    DET_PARAMS DetParams;       // dummy arg for AsmSearch
    double MeanTime;            // dummy arg for AsmSearch

    SHAPE Shape = AsmSearch(StartShape, DetParams, MeanTime, Img, sImage);

    if (Shape.nrows())          // successfully located landmarks?
        {
        DrawShape(Img, Shape);  // draw landmark shape on image
        CropImageToShape(Img, Shape);
        WriteBmp(Img, "search-results.bmp", VERBOSE);
        // Shape.print("Landmarks:\n"); // print the landmarks if you want
        }

    return 0;
}
```

Figure 20: *Example code for finding landmarks.*

VERBOSE_ASM_SEARCH in stasm.hpp.

If you have existing matrix or image libraries you may want to replace the interface in the mat and image directories. However it is usually much simpler (although inelegant) not to do so and to keep both your libraries and the Stasm code in parallel.

11.5 Aspects of the Stasm code

All programs in the Stasm library return 0 on success and non-zero otherwise.

The Stasm code is in C++ but generally written in a C-like way.

The code uses a form of Hungarian notation, for better or worse (http://en.wikipedia.org/wiki/Hungarian_notation). Figure 21 shows the naming scheme. The prefix characters can be combined, so you will see prefixes like sg for global strings. Additionally, defines, constants, and typedefs are usually in upper case or have an upper case prefix. There is a fair amount of inconsistency, which usually occurs when code is pulled in from other libraries.

In function parameter lists, parameters that are updated appear before other parameters.

Function declarations have a space before the "("; function usages and externs don't. This is useful when searching for a function definition.

A comment line of dashes separates functions, and means that code in functions does not need to be offset from the left margin for legibility.

Prefix	**Type**	**Example**
n	number	nShapes is the number of shapes
i	index	iShape is a shape index, typically in an array
p	pointer	pShape points to a shape
s	string	sShape is a shape name
g	global var	gShapes is a global array of shapes
f	boolean	fFound, f for flag (b is used for bits but unnecessary in Stasm)
e	enum	eLandmarks

Figure 21: *Naming conventions in the Stasm package.*

Several utility functions and variables are listed below.

Err Prints an error message and exits the program. Takes arguments like `printf`.

Warn Prints a warning message. Takes arguments like `printf`.

lprintf Prints to the screen and to the log file. Calling `printf` directly is unusual (Tasm uses it only to print user pacifiers for long processes). Prepare `lprintf` at the start of the program by calling `pOpenLogFile`, else `lprintf` just prints to the screen (which is appropriate for some applications, e.g. `Fdet`).

logprintf Prints to the log file (or nowhere if not prepared by calling `pOpenLogFile`, which is legal).

Fopen, Fprintf, Fwrite, Fread These are like their counterparts with lower case names but give an error message and exit on failure

Fgets Like `gets` but skips white space lines and comments (lines beginning with #).

nGetLineNbr(FILE *pFile) Returns the current line number in `pFile`. This is slow (because it scans the file from the beginning) but is only used for reporting errors.

sgBuf A global string buffer for general use.

The memory allocation routines `malloc` and `calloc` are redefined in `safe_alloc.hpp` to issue an error message if there is no available memory.

The files `werr.cpp` and `wrelease.cpp` are versions of `err.cpp` and `release.cpp` that are used in windowed applications (applications that put a window on the screen, as opposed to command line applications).

The directory `image` has standard monochrome and RGB image classes. These were written before portable image libraries were easily available (before OpenCV for example). People sometimes ask us to chuck out these classes and replace them with their favorite library. What makes that non-trivial (but not hard) is different coordinate systems (Section 4.3).

Files in the `mat` directory define a matrix class `Mat` and a matrix view class `MatView`. They are built on the matrix routines in the Gnu Scientific Library and based on the `gslwrap` library (Section 12). The header of `mat.hpp` has a full description. You can do the usual things like adding matrices with an overloaded `+` operator. Matrix *views* allow you to access a matrix or part of that matrix without copying data — so you can, for example, treat an entire matrix as one long row vector using `viewAsRow()`. Assignment (`=`) of matrices will give a runtime error message if the matrices are not conformable, with the exception that assignment to a 0 by 0 matrix (i.e. an uninitialized matrix) is always allowed. This is intended to protect against programming errors. Use the `assign` method to assign non-conformable matrices. These classes were written a while ago, before there were easily available matrix classes that compiled in both the Microsoft and GCC environments.

The file `matvec.hpp` has facilities to manipulate STL vectors of `Mats`.

Stasm tends to use row vectors instead of column vectors (for compact printing when debugging, a decision made when the project started, possibly not the correct choice). This means, for example, that the software uses $\mathbf{xAx^T}$ where you might expect to see $\mathbf{x^TAx}$.

The tokens `VX` and `VY` (defined as `0` and `1`) by convention specify the x or y column in a shape matrix.

In some cases, program execution continues after hitting Control-C. This behavior seems to have come in with version 1.0 of the OpenCV library. Hit Control-C again if necessary.

Prior to Stasm, the Rowley detector was used extensively in other projects and the code has been considerable modified. In particular, error and file handling has been unified with the approach used in Stasm (so gives Stasm-style error messages for example).

12 Acknowledgments

I take pleasure in acknowledging the following people who provided ideas, code, data, and techniques used in Stasm.

Tim Cootes `http://www.isbe.man.ac.uk/~bim`

David Cristinacce `http://mimban.smb.man.ac.uk`

Fred Nicolls `http://www.dip.ee.uct.ac.za/~nicolls`

Brad Yearwood and **Pierre Moreels** for help with the Ubuntu port.

GuoQing Hu for his start-shape alignment technique `http://digface.t35.com`

Roger Willcocks for his prototype ASM code `http://www.rkww.com`

Eugen Dedu for his line drawing code `http://lifc.univ-fcomte.fr/~dedu`

Henry Rowley, Shumeet Baluja, and Takeo Kanade for the Rowley face detector `http://vasc.ri.cmu.edu/NNFaceDetector`

M. Galassi, J. Theiler, and others for the GSL library `http://www.gnu.org/software/gsl`

Ramin Nakisa and others for gslwrap (used as a basis for Stasm's Mat class) `http://sourceforge.net/projects/gslwrap`

Paul Viola and Michael Jones for their face detector `http://research.microsoft.com/~viola` and `http://www.merl.com/people/mjones`

Rainer Lienhart for his implementation of the Viola Jones detector `http://www.lienhart.de`

Jesorsky, Kirchberg, and Frischholz for making available the BioID data `http://www.bioid.com/downloads/facedb`

Aleix Martinez and Robert Benavente for the AR database `http://cobweb.ecn.purdue.edu/~aleix/aleix_face_DB.html`

The **XM2VTS people at Surrey** for the XM2VTS database `http://www.ee.surrey.ac.uk/CVSSP/xm2vtsdb`

The **manual landmarkers at Manchester** for their work `http://david.cristinacce.net` and `http://www.isbe.man.ac.uk/~kob`

The Independent JPEG Group for their library `http://www.ijg.org`

The **OpenCV** vision library `http://www.intel.com/technology/computing/opencv/index.htm`

References

[Cri04] D. Cristinacce. *Automatic Detection of Facial Features in Grey Scale Images (Doctoral Thesis).* University of Manchester (Faculty of Medicine, Dentistry, Nursing and Pharmacy), 2004. `http://david.cristinacce.net/index.php`.

[CT04] T. F. Cootes and C. J. Taylor. *Technical Report: Statistical Models of Appearance for Computer Vision.* The University of Manchester School of Medicine, 2004. `http://www.isbe.man.ac.uk/~bim/Models/app_models.pdf`.

[HJLM07] Gary B. Huang, Vidit Jain, and Erik Learned-Miller. *Unsupervised joint alignment of complex images.* ICCV, 2007. Source code available at `http://vis-www.cs.umass.edu/code/congealingcomplex`.

[JKF01] O. Jesorsky, K. Kirchberg, and R. Frischholz. *Robust Face Detection using the Hausdorff Distance.* AVBPA, 2001. `http://www.bioid.com/downloads/facedb`.

[LM02] Rainer Lienhart and Jochen Maydt. *An Extended Set of Haar-like Features for Rapid Object Detection.* ICIP, 2002.

[MB98] A.M. Martinez and R. Benavente. *The AR Face Database.* CVC Tech. Report 24, 1998. http://rvl1.ecn.purdue.edu/~aleix/aleix_face_DB.html.

[Mil07] S. Milborrow. *Locating Facial Features with Active Shape Models (Master's thesis).* University of Cape Town (Department of Image Processing), 2007. http://www.milbo.users.sonic.net/stasm.

[MN08] S. Milborrow and F. Nicolls. *Locating Facial Features with an Extended Active Shape Model.* ECCV, 2008. http://www.milbo.users.sonic.net/stasm.

[RBK98] Henry A. Rowley, Shumeet Baluja, and Takeo Kanade. *Neural Network-Based Face Detection.* PAMI, Volume 20, pages 23–38, 1998. http://vasc.ri.cmu.edu/NNFaceDetector.

[VJ01] P. Viola and M. Jones. *Rapid object detection using a boosted cascade of simple features.* CVPR, Volume 1, pages 511–518, 2001. http://citeseer.ist.psu.edu/article/viola01rapid.html.